University of Zurich[UZH]

Department of Informatics

Agile Aerial Autonomy: Planning and Control

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
Philipp Foehn
from Schwyz, Switzerland

approved in February 2022

at the request of
Prof. Dr. Davide Scaramuzza, advisor
Prof. Dr. Moritz Diehl, examiner
Prof. Dr. Luca Carlone, examiner
Prof. Dr. Roland Siegwart, examiner

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, February 16, 2022

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

What I cannot create,
I do not understand.
— Richard Feynman

To my advisor and colleagues, a thriving inspiration,
guiding me along hard problems to elegant solutions.

&

To my love, an infinite source of peace and calm,
unconditional empathy, support, and motivation.

&

To my family, the idol, spark, and foundation
of my curiosity and fascination, enabling my dreams.

Acknowledgements

I am extremely thankful to my supervisor *Prof. Davide Scaramuzza*, who gave me the opportunity to contribute to the research community, inspired me with his interesting visions, supported me with great motivation, and always had an open ear for questions, critique, and discussions. It is outstanding how much time Davide invests in guiding us students along our journey, while allowing great freedom and creativity.

A special thank also goes to my colleague *Elia Kaufmann*, who not only became an inspiring coauthor of mine, but also a true friend. As someone working on rather different research topics, Elia was never shy of sharing his alternative views on challenging problems. He was often of great help, whether it was when discussing new ideas, sharing insights into difficult problems, thinking outside the box, or putting research in the context of a big vision.

I wish to express my gratitude to all the current and past members, visitors, and students who I encountered during my journey as a PhD student at the *Robotics and Perception Group*. I would particularly like to thank Matthias Fässler, Davide Falanga, Titus Cieslewski, Antonio Loquercio, Mathias Gehrig, Daniel Gehrig, Manasi Muglikar, Yunlong Song, Giovanni Cioffi, Nico Messikommer, Jeff Delmerico, Guillermo Gallego, Dario Brescianini, Sihao Sun, Robert Penicka, Christian Pfeiffer, Alessandro Simovic, Julien Kohler, Thomas Längle, Manuel Sutter, Naveen Kuppuswamy, and Tamar Tolcachier. Furthermore, I was fortunate enough to work with great international collaborators, namely Prof. Vladlen Koltun and Prof. Russ Tedrake.

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. Moritz Diehl, Prof. Luca Carlone, and Prof. Roland Siegwart for accepting to review my thesis.

Personally, I thank my family and friends for all their support and motivation throughout the years, tolerating my ups and downs, and even my absences when deadlines came closer.

Last but not least, I express my deepest gratitude to my girlfriend, spouse, and future wife, Nicole. She has been supporting me from the initial decision to do a PhD, until the last character of my thesis was written, and beyond that. Her calm and comforting nature

helped me during stressful and troubling times, her positive attitude inspired me to never give up. Her genuine interest in my work and problems, together with an outstanding portion of empathy and emotional intelligence, often allowed me to see challenges from a different point of view, and set me free from biased opinions. Nicole, you helped me change into the personality I am today, and I couldn't be more grateful for it.

Zurich, December 2021 *Philipp Foehn*

Abstract

More than 80 billion Dollar, that's the drone market value by 2025, according to Forbes [5] prediction in 2020. The potential addressable market for autonomous drone applications is gigantic, reaching from consumer products over cinematography, remote sensing, and entertainment, to delivery, transportation, emergency services, and search and rescue. And yet, there is an immense disconnect between the flight performance of autonomous systems and human pilots, capable of guiding remote-controlled drones at impressive speeds through a track in a sport called drone racing. Why is there such a dwelling gap between human and machine performance? Clearly, industrial applications would greatly benefit from agile and fast flight capabilities in terms of profitability and robustness, and companies aspire to achieve ever-increasing efficiency and productivity. What hinders the industry from exploring and exploiting the full potential of drones, such as quadrotors?

In fact, quadrotors are amongst the most agile and maneuverable vehicles ever created. Combined with their mechanical simplicity, low cost, and vertical takeoff and landing as well as hover capabilities, they rose to become the most prominent aerial platform for emerging technologies. Even though quadrotors are a single rigid body with thrusters, their fast rotational dynamics and powerful but underactuated acceleration capabilities render them both surprisingly maneuverable and difficult to control. Challenges arise especially from their non-linear and underactuated system dynamics, for which hierarchical planning algorithms and cascaded control architectures are inadequate solutions. These classic approaches to mobile robot navigation typically assume linear time-invariant dynamics and unbounded inputs, both of which are improper assumptions for quadrotors and lead to overly conservative performance, infeasible actuator commands, or even instability. However, to allow navigation algorithms to exploit the full capabilities of quadrotors and achieve agile yet robust control performance, they need to be able to deal with non-linearities and actuator bounds, both in planning and in control.

On the other hand, in the 1960s, Kalman laid the foundation for both optimal linear quadratic control and estimation. These methods were only one part of the history that forms today's optimal control paradigm, which comes in a number of approaches to estimation, planning, and control, applied to a vast variety of systems. One subset of these methods is especially interesting for quadrotors, namely non-linear model-predictive control (NMPC), and its sibling, trajectory optimization. NMPC exploits analytical models of the dynamics to predict a system's state over a receding horizon and optimize the system inputs and prediction to minimize a, typically quadratic, cost function. It is highly advantageous in that it can account for non-linear systems and actuator constraints, and enables versatile task formulations while providing intuitive performance tuning.

Abstract

Recent advances in compute capabilities and implementation strategies of the resulting numerical optimizations nowadays allow even complex problems to be optimized in real-time onboard resource-limited vehicles, such as size and weight constrained aerial platforms.

This thesis investigates how such novel optimization-based planning and control strategies can be applied to enable autonomous quadrotors to exploit their full performance envelope and allow for agile and robust autonomous flight. Among others, this thesis provides contributions allowing quadrotors to fly at their actuation limit, culminating in a framework that even beats expert human drone racing pilots in a time trial demonstrator. The presented methods first address the problem of time-optimal planning for quadrotors, proposing a novel progress-based formulation allowing the computation of trajectories through multiple waypoints at the limit of the quadrotor's performance envelope. This planning approach is followed by building up a non-linear model-predictive control framework for accurate, agile, and robust navigation, allowing the real-world demonstration of the previous time-optimal planning approach. To complete the time-optimal planning and control contributions, this thesis also provides insight into high-fidelity modelling of aerodynamic effects on quadrotors and provides a method to measure those effects in a visual-inertial odometry framework.

Overall, the contributions of this work aim to answer the question of *How can control and planning strategies exploit the full capabilities of quadrotors*? In the following is a list of contributions:

- The first method to generate time-optimal quadrotor trajectories through multiple waypoints under non-linear system dynamics and actuator constraints.
- A non-linear model-predictive control (NMPC) framework including the full quadrotor dynamics and single-rotor actuation constraints, and, for the first time, also perception objectives.
- A complete framework for agile drone flight providing open-source and open-hardware resources, incuding mission and task logic, state estimation, trajectory planning, and non-linear model-predictive control. This framework has been used in collaborative work providing further insights into comparative evaluations of NMPC and classic control strategies, rotor-failure-tolerant NMPC, approximate time-optimal NMPC, and aerodynamic modelling for simulation and control purposes.
- A method to estimate residual forces within an visual-inertial odometry, which can be used to identify residual terms not covered by the modelled system dynamics, such as aerodynamic forces.

List of Contributions

The * symbol indicates shared first authorship.

Publications under Review

- Fang Nan, Sihao Sun, **Philipp Foehn**, Davide Scaramuzza, "Nonlinear MPC for Quadrotor Fault-Tolerant Control", *IEEE Robotics and Automation Letters (RA-L)* (2022), DOI: arxiv:2109.04210, Links: PDF, Video

- **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* (2021), Links: Webpage, Code, Appendix B

- 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)* (2021), DOI: arxiv:2109.01365, Links: PDF, Video

- Angel Romero, Sihao Sun, **Philipp Foehn**, Davide Scaramuzza, "Model Predictive Contouring Control for Near-Time-Optimal Quadrotor Flight", *IEEE Transactions on Robotics (TRO)* (2021), DOI: arxiv:2108.13205, Links: PDF, Video

Journal Publications

- **Philipp Foehn**, Angel Romero, Davide Scaramuzza, "Time-Optimal Planning for Quadrotor Waypoint Flight", *Science Robotics* (2021), DOI: 10.1126/scirobotics.abh1221, Links: Science Robotics Article, PDF, Video, Code, Appendix A

- **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, Links: PDF, Video, Talk, Appendix C

- Drew Hanover, Elia Kaufmann, **Philipp Foehn**, 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, Links: PDF, Video

- Barza Nisar*, **Philipp Foehn***, Davide Scaramuzza, "VIMO: Simultaneous Visual Inertial Model-based Odometry and Force Estimation", *IEEE Robotics and Automation Letters (RA-L)* (2019) and *Robotics: Science and Systems (RSS)* (2019), DOI: 10.1109/LRA.2019.2918689, Links: PDF, Video, Code, Appendix E

- 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, Links: PDF, Video, Code

Peer-Reviewed Conference Papers

- 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, Links: PDF, Video, Code

- **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, Links: PDF, Video, Talk, Appendix C

- Balazs Nagy, **Philipp Foehn**, Davide Scaramuzza, "Faster than FAST: GPU-Accelerated Frontend for High-Speed VIO", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), DOI: 10.1109/IROS45743.2020.9340851, Links: PDF, Video, Code, Appendix D

- 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, Links: PDF, Video, Appendix F

- Davide Falanga*, **Philipp Foehn***, Davide Scaramuzza, "PAMPC: Perception-Aware Model Predictive Control for Quadrotors", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018),
DOI: 10.1109/IROS.2018.8593739, Links: PDF, Video, Code, Appendix G

- **Philipp Foehn**, Davide Scaramuzza, "Onboard State Dependent LQR for Agile Quadrotors", *IEEE International Conference on Robotics and Automation (ICRA)* (2018), DOI: ICRA.2018.8460885, Links: PDF, Video

- **Philipp Foehn**, Davide Falanga, Naveen Kuppuswamy, Russ Tedrake, Davide Scaramuzza
"Fast Trajectory Optimization for Agile Quadrotor Maneuvers with a Cable-Suspended Payload", *Robotics: Science and Systems (RSS)* (2017),
DOI: 10.15607/RSS.2017.XIII.030, Links: PDF, Video

Open-source Software

- **Agilicious Flightstack:** Paper B
  available at https:agilicious.dev
- **Time-Optimal Planning:** Paper A
  available at https://github.com/uzh-rpg/rpg_time_optimal
- **Perception-Aware MPC:** Paper G
  available at https://github.com/uzh-rpg/rpg_mpc
- **Visual-Inertial Model-based Odometry:** Paper E
  available at https://github.com/uzh-rpg/vimo
- **GPU-Accelerated Corner Tracker:** Paper D
  available at https://github.com/uzh-rpg/vilib

Awards

- *Robotics: Science and Systems (RSS) 2020*, **Best System Paper Award** for Paper C and invitation to publish in *Springer: Autonomous Robots*.
- *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 F.
- *Robotics: Science and Systems (RSS) 2017*, **Best Student Paper Award Finalist** with my paper on "Fast Trajectory Optimization for Agile Quadrotor Maneuvers with a Cable-Suspended Payload" [70].

Contents

1 Introduction

This thesis presents novel planning and control methods for rotorcraft vehicles bridging the gap between human and autonomous piloting performance. In the past decade, we have seen the initial rise of remotely piloted and autonomous aerial rotorcraft vehicles, from here on called *drones* in general and *quadrotors* more specifically. Even though autonomous solutions have been developed and deployed in various consumer products [217, 53], human pilots have outperformed algorithms in terms of robustness, agility, and speed. Especially with the advent of drone racing [247], the gap between human piloting capabilities and autonomous vehicles became embarrassingly apparent, leading to a mismatch between the publicly perceived capabilities of drones and the actual robustly feasible autonomous capabilities. More importantly, applications such as package delivery, warehouse inventory tracking, remote sensing and observation, cinematography, and search and rescue inherently require a sophisticated level of robustness and speed to be of any economic value. Or as Steven Hawking said in his book [93]: "Robots will definitely speed up the online retail process. But to revolutionize shopping, they need to be fast enough to allow same-day delivery on every order." Is it possible to exploit the full capabilities of quadrotors with autonomous systems? This thesis addresses the robustness and speed gaps between human pilots and autonomous systems by proposing planning and control algorithms that enable a new regime in autonomous agile flight and even surpass human capabilities.

Quadrotors are exceptional mobile robots, being both mechanically simple but dynamically complex. Even though quadrotors are a single rigid body with thrusters, their fast rotational dynamics and powerful but underactuated acceleration capabilities render them both surprisingly maneuverable and difficult to control. Classic autonomous navigation algorithms separate perception, planning, and control into separate subsystems and split the control architecture into multiple cascaded feedback loops. Both of these practices, separation and cascading, are common and well-established techniques originating from linear system control theory under the assumption of precise state estimation or measurements and non-saturating actuators. However, the dynamics of quadrotors are highly non-linear on multiple time scales, underlie strict actuator constraints, and introduce a coupling between perception and action. As a result, cascaded controllers must apply restrictive linearizations and conservative bounds, while planning approaches often rely on

unrealistic approximations of the real system, both of which severely limit the performance and restrict autonomous systems to a small subset of the feasible flight envelope. Finally, because mobile robots can be deployed in arbitrary environments, they have to rely mostly on proprioceptive sensing and onboard estimation, such as visual-inertial odometry. These measurement modalities are influenced by the drone's egomotion, which implies that overly aggressive maneuvers degrade measurement quality, e.g. through motion blur in camera images. Therefore, to exploit the true capabilities of quadrotors and achieve robust and agile flight, we must face the fundamental technical challenges of accounting for real non-linear dynamics, actuation bounds, and perception-action-coupling in all stages of a navigation system, which formulates the research question of my work.

Specifically:

- How can we generate fast and agile flight paths?
- How can we simultaneously achieve robust, versatile, and agile control for fast and reliable mission execution with drones?
- Is it possible to exploit the coupling between action and perception on aerial robots?

To address these questions, my work combines theories and methods from model-predictive control (MPC), numerical optimization, and vision-based estimation.

In 1960, Kalman presented "Contributions to the Theory of Optimal Control" [107] and "A New Approach to Linear Filtering and Prediction Problems" [106], both of which laid the foundation for linear-quadratic regulator (LQR) and Gaussian estimators, respectively. These were the first provably optimal approaches to estimate and control linear time-invariant systems under Gaussian noise assumptions and unbounded states and inputs over infinite time horizons. As such, these methods are extremely powerful and have not only been deployed in countless applications, but also commenced a whole new research field on optimal control. More recent developments adapted Kalman's principles by introducing repeated linearization for time-varying or non-linear systems, discretization of finite time horizons (also known as "sliding-window"), and constrained optimization to address state and input bounds. More importantly, these advances apply to both estimation and control, which in fact share their problem structure as optimizations under Gaussian assumptions, and can be reduced to non-linear constrained least-squares formulations. The culmination of those improvements is non-linear model-predictive control and its sibling, non-linear sliding-window estimation, both of which became influential research directions and have lately been applied in the field of chemical process control, aeronautic and space exploration, autonomous driving, general mobile robotics, and even state estimation on handheld devices for augmented and virtual reality.

Inspired by the capabilities of these optimal methods and the impressive maneuverability

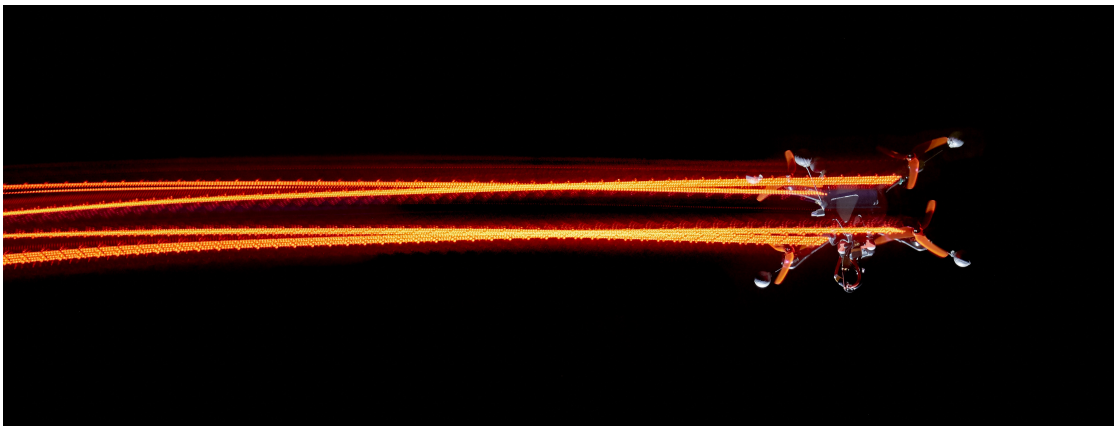


Figure 1.1: A quadrotor flying along a time-optimal trajectory using non-linear model-predictive control. In such fast flight maneuvers, it is crucial to account for non-linear system dynamics, acutator saturation, and high-fidelity models.

of quadrotors, this work will answer the research questions through a combination of optimization-based planning and control.

As a common demonstrator, I will consider the problem of autonomous drone racing. In order to be successful, an autonomous system must be able to fly as fast as possible, or even time-optimal, while ensuring completion, since any crashes or missed gate passes count as a failed attempt. Therefore, achieving optimal performance requires the navigation system to be *fast*, *robust*, and *precise*, challenging both the planning and the control strategies. Furthermore, this demonstrator brings the advantage of having access to highly proficient baselines in the form of expert human pilots. The relevance of this demonstrator is linked to the requirements for safe and efficient deployment of aerial drones in industrial and consumer-oriented applications. First, delivery, mapping, exploration, and inspection tasks profit from efficient mission execution. Second, cinematography through subject tracking requires a high level of agility to chase a subject along arbitrary movements and record visually appealing footage. Third, in search and rescue tasks time-optimal location of casualties is critical and can even be life-saving. But all those applications only profit from speed if and only if a mission can be completed successfully, which implies that fast execution only brings value if it does not harm robustness, which closely relates those tasks to the chosen demonstrator, *drone racing*.

This thesis is split into three parts. First, I present an approach to time-optimal planning for quadrotors. While there already exist multiple approaches to plan quadrotor trajectories, none of these do so in a time-optimal fashion while respecting the true dynamics and actuator limits of the vehicle. In this part, I introduce a novel method to plan trajectories through a sequence of waypoints, while ensuring feasibility at the limit of the vehicle's capability, and for the first time exploiting the quadrotor's full flight performance envelope. Second, I apply novel control strategies enabling robust execution of a large variety of tasks. This is done in multiple successive steps, starting

with a state-dependent linear-quadratic regulator (LQR) controller to relax linearisation approximations, followed by including system state and input boundaries through NMPC, which is extended to multiple different applications. Finally, in the third part of the thesis, I present state estimation and system modelling techniques, two critical components when applying previously established methods to real-world systems.

All parts address the driving research question of this work - *how can we exploit the full capabilities of quadrotors?* - from a different perspective but with the same goal: enabling robust and agile autonomy.

This thesis is structured in the form of a collection of papers. An introductory section that highlights the concepts and ideas behind the thesis is followed by self-contained publications in the appendix. Chapter 1.1 elaborate on the relevance of the proposed research questions and motivates the research objective of this work. Chapter 1.2 places this research in the context of the related work. Chapter 2 explains the contributions of the published research papers and their relation. Finally, Chapter 3 provides future research directions.

1.1 Motivation

Quadrotors are amongst the most agile and maneuverable aerial robots [2, 242], and have disrupted industries such as agriculture, cinematography, architecture and building information management, warehouse inventory tracking, search and rescue, and remote sensing. Exploiting their agility in combination with full autonomy is crucial for a large variety of missions, such as inspection and observation, delivery and transportation [192], and even entertainment such as drone racing [129, 247]. For this reason, over the past decade, research on autonomous, agile quadrotor flight has continually pushed platforms to higher speeds and agility [151, 130, 152, 238, 110, 132, 260, 68, 125, 171, 112, 71]. Million-dollar projects, such as AgileFlight [40] and Fast Lightweight Autonomy (FLA) [152], have also been funded by the European Research Council and the United States government, respectively, to further push research. Additionally, many competitions have been organized, such as the autonomous drone racing series at the recent IROS and NeurIPS conferences [153, 38, 110, 140] and the AlphaPilot challenge [88, 68], with the goal to develop autonomous systems that will eventually outperform human pilots. However, human pilots accomplish drone racing with astonishing performance, guiding their quadrotors through race tracks at speeds so far unreached by any autonomous system.

Nevertheless, the adaption of autonomous vehicle utilization for industrial and commercial applications is increasing at an astonishing rate, with over 300'000 commercial drones registered in the US alone [212] and an estimated global drone service market value rise from 4.4B$ to more than $60B by 2025 according to Business Insider [170], and to over

$80B according to Forbes [5].

To meet this promising demand, live up to the expectations of the industry, and enable novel applications beyond today's, autonomous mobile robots must become robust enough to earn the trust of consumers, and fast enough to be of any significant profit.

1.1.1 Advantages

Optimization-based methods such as model-predictive control and trajectory planning bring a number of advantages that significantly advance robustness and exploit agility for fast mission completion. The following are three significant advantages these technologies could enable in the future.

Accurate Dynamic Models and Actuator Bounds Model-predictive control, as the name implies, exploits the fact that the evolution of a mechanical system over time can be modeled and, given the current state, predicted along a future time horizon. This enables approaches which not only react based on measurements made at specific time instances, but also their impact on the system in the future, allowing to trade-off momentary and future effects. This allows creating control policies that, different from classic state-feedback control, not only suppress momentary tracking errors, but exploit the dynamics of the system to find actions that enable the best possible performance in the future. Similarly, such model-based approaches allow one to correctly account for actuator bounds over the entire prediction horizon.

These properties are especially beneficial for the control of quadrotors due to their non-linear integral system dynamics and bounded low-level actuation. Short-time prediction of the quadrotor's future trajectory helps to anticipate rapid changes in the flight path and balance disturbance rejection and alignment with the desired future trajectory. Moreover, it enables foreseeing actuator saturation, e.g. due to large perturbations, and counteract them preemptively.

Last but not least, dynamic models can not only be used in trivial simulations or short-term predictive control, but also to plan and refine trajectories spanning a complete task or even mission. This is done by leveraging analytical models in numerical optimization to find the best possible solutions to certain problems. Compared to alternative approaches, which rely on closed-form solutions or dynamic programming, numerical optimization does not rely on system approximations or hierarchical abstractions. In the context of fast and agile flight, trajectory optimization enables the computation of feasible trajectories at the system's true dynamic limits, as shown in Paper A [71].

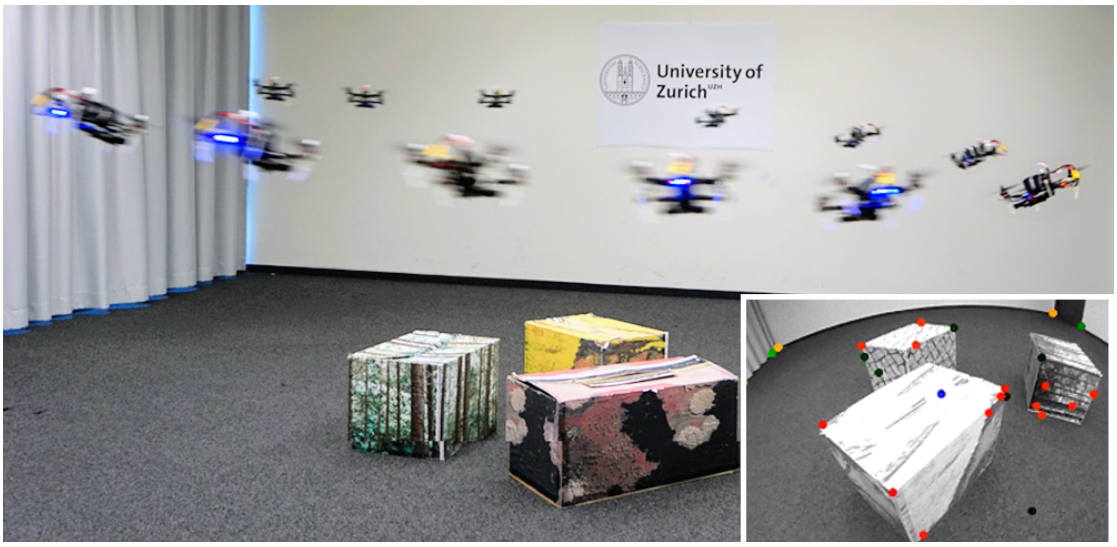


Figure 1.2: The approach from Paper G exploiting novel perception-aware model-predictive control and task formulation in the cost function to keep a point of interest in the camera view.

Versatile Task Formulation and Tuning Formulating control or planning problems as the optimization of a cost function over a state and input space allows encapsulating nearly arbitrary task descriptions. Different from cascaded control loops, MPC does not require the definition and tuning of feedback laws for all intermediate states but allows to penalize task-specific choices of direct states, inputs, or functions of those. This versatility in cost function design not only avoids over-penalization of underactuated state spaces, such as in the case of quadrotos, but also enables novel objective formulations, such as perception-aware control strategies in Paper G [59] (Figure 1.2), progress-maximizing controllers [197], or time-optimal planning in Paper A [71].

Additionally, even though popular methods exist to tune proportional-integral-derivative controller (PID) controllers [262], those rely on identification of certain (closed-loop) system properties, typically require manual adjustments, and for non-linear systems result in either over-conservative or potentially unstable performance. Conversely, in an MPC approach, the most fundamental cost function is a weighted quadratic penalty on the variable space, which allows the intuitive interpretation as a joint tradeoff between the variance of states and inputs over the prediction horizon. As an example, designing reference tracking controllers becomes as simple as defining the individual relevance of actuation and tracking errors for all states, or, in the case of a quadrotor, all differentially flat outputs and (optionally) their derivatives.

Simple Extension to New Capabilities The aforementioned versatility also enables fast prototyping and adaptation to new inputs, system dynamics, and control objectives. With each advancement in robotics, researchers face new challenges, which necessitates

adaptive methodologies that can grow in complexity and fidelity as robotics grows into maturity. Optimization-based techniques like MPC are well suited for this task since the underlying system dynamics, actuation and sensing modalities, and objective formulations can develop independently and enable both bottom-up and top-down approaches.

As an example, quadrotor control and planning algorithms [149, 159, 58] typically use a certain abstraction level for their actuation modality. Traditionally collective thrust and bodyrates are used as high-level commands, which are tracked by low-level controllers, but neglect the actual actuator bounds. NMPC allows us to model also these low-level dynamics, whether or not they are used as actuation commands, increasing the modelling accuracy and therefore also the prediction and control performance, as demonstrated in [239].

This extendability also allows adapting existing solutions to new challenges. For example, flying accidents owing to system malfunctions, such as rotor failures, are major stumbling blocks to the development and the public acceptance of the drone industry. However, thanks to their non-linear dynamics, quadrotors can still remain airborne and under control, even when one actuator fails, but the control strategies developed so far are highly specified and do not allow to recover from arbitrary maneuvers [228]. To achieve fault-tolerance and even recover from aggressive maneuvers without having to design specific new algorithms, the general-purpose MPC developed in the context of Paper B [73] has been adapted in [167] to allow robust recovery after rotor failures and full translational control, even tracking fast trajectories with a damaged quadrotor. While this contribution could potentially save countless drones from crashes due to damaged propulsion systems, it consists only of slight adaptions of the cost function and a change in the actuator constraints upon failure detection, but does not limit or impair the control performance.

Interpretability and Relation to Estimation Problem Many problems can be efficiently formulated by minimizing quadratic cost functions, which closely relate or even are equal to least-squares formulations. Naturally, these quadratic minimizations represent the optimal solution under Gaussian noise or disturbances, which renders them well suited for real-world applications, where Gaussian noise often is a viable approximation or even an accurate representation of true uncertainty.

Furthermore, this allows an intuitive interpretation of the main configuration parameters of such methods, namely the individual weights of a quadratic loss function. In particular, quadratic cost functions correspond in their minima to the minimal Malahanobis norm, equal to the maximum likelihood given a Gaussian distribution, where the cost function weights are the inverse of the distribution's variance. This gives a simple interpretation of the cost function, for example, in the case of controller tuning as mentioned in the previous Paragraph 1.1.1.

Finally, and most importantly, the similarity between LQR, MPC, trajectory planning, and least-squares problems also relates them to most estimation problems that follow Gaussian noise assumptions, such as the Kalman filter [106] in the simplest case, or factor-graph representations of moving-horizon estimators [46]. Since both, MPC and moving-horizon estimator (MHE), typically optimize quadratic costs over a moving relative time window, similar methods and practices can be used to solve them efficiently and potentially even unify them. This could allow profiting from shared data (measurement and task-related), better account for perception-action coupling on mobile robots, and lower computational costs causing reduced latency and, therefore, better control performance.

1.1.2 Challenges

Despite their numerous advantages, optimization-based methods also bring their specific challenges. These challenges arise from two main difficulties: i) they rely on system dynamic models where any inaccuracies or uncertainties result in degraded solution quality; and ii) non-linear optimization with limited compute resources strictly relies on convex problem formulations and even then is difficult to formulate into computationally tractable descriptions.

Dynamic Model Accuracy and Complexity First and foremost, the aforementioned opportunities all rely on modelling the dynamics of a robot. While this is possible using first-order principles and system identification, it is often tedious or even impossible to achieve arbitrary model fidelity. Simple models, such as the quadrotor's translational and rotational high-level dynamics, can be derived ideally as integrators without parameters. However, lower-level models, such as the acceleration dynamics, are not only specific to the vehicle's physical parameters but also include aerodynamic effects, which are not only complex to model but even chaotic in nature.

Therefore, one of the challenges is to pick a level of model-fidelity that is accurate enough for all tasks at hand yet simple enough to obtain with practical effort. Models of low quality inherently deteriorate real-world performance since the planned trajectories or predicted motions would not comply with the real system behavior. However, high-fidelity models often imply significantly increased computational effort, can potentially overfit and become decremental due to variations in the real system, and might even render applicability in optimizations unfeasibly by introducing multimodal behavior or severe non-convexity.

A prominent example of model complexity mismatch appears in trajectory planning for quadrotors, which has been done for years through simplified models relying on dynamic-flatness [149, 159, 58] or even point-mass assumptions, but is incapable of producing trajectories at the boundary of the real actuator constraints. My work demonstrates a

gradual increase of the model fidelity, starting with simple models in [72] and Paper G [59], to true actuator limits in Paper A [71] and Paper B [73], and finally extending to aerodynamic identification in [239, 19].

Computational Tractability Optimization problems of arbitrary size and nature are typically solved using iterative approximative methods, such as gradient descent or Newton's method for unconstrained problems, Gauss-Newton method and Levenberg-Marquardt for unconstrained least-squares, or interior-point methods for constrained (and desireably convex) non-linear problems. However, all of those methods rely on the iterative linearization of a non-linear system with differentiable cost and constraint functions. This generally implies two challenges: i) calculation of the Jacobians and linear decompositions (e.g. in interior-point methods) can be computationally extremely demanding, and ii) convergence on non-linear problems can be arbitrarily slow. Therefore, non-linear optimization can require a significant amount of computation time, rendering it challenging to apply those methods to real-time systems.

While planning problems typically do not have strict limits on computation time, it directly affects the possible frequency at which replanning can be performed. The lower the computation time of a planning algorithm, the faster it can adapt to changes in the environment or in the task, such as the appearance of dynamic obstacles, large deviations from the planned flight path, or refinement of a waypoint location. The latency at which this adaption can be handled directly impacts how fast the vehicle can move without risking invalidating the previously planned trajectory.

Depending on the application and the environment, it can be feasible to perform planning purely prior to execution time, e.g. in a static environment. On the other hand, in case of MPC or MHE, the real-time requirements are as strict as the dynamic bandwidth of the respective system. In the case of quadrotor control, this means that optimization solutions need to be available hundreds of times per second with low latency. Most mobile systems also require onboard execution of these problems, which often tremendously limits the available computational resources due to size and weight constraints on mobile aerial vehicles.

As a result, the feasible complexity and time horizon of MPC problems are strictly constrained. To handle this challenge, it is critical to consider complexity, convexity, linearization properties, and the solution process when designing such optimization problems. Additionally, there is one strong real-world property that can be exploited to design an efficient solution scheme: dynamic systems only evolve at a certain rate, which implies that solutions of consecutive timesteps typically are similar and within a certain distance measure from each other. This allows one to use previous solutions as an initial guess for consecutive problems, and in the extreme case, even use individual iteration results as intermediate best solutions. This practice described in [52, 253] allows

using approximations of solutions at extremely high rates and with low latency, enabling otherwise impossible closed-loop control with optimal policies even on computationally limited platforms.

Convexity Last but not least, convexity is one of the most desireable – but usually unattainable – problem properties for non-linear optimization approaches. If a problem is not convex, it can have more than one local extremum, which can or can not coincide with the true global extremum. In this case, it becomes non-trivial or even impossible to verify that any stable point of the cost function is a global extremum. Methods that can partially overcome such problems apply sampling techniques, which are costly to run and therefore unfeasible for time-critical problems. On the other hand, convex (non-linear) problem formulations, or initializations within a convex region around the global optimum, can be solved quickly and reliably using iterative methods, e.g. gradient-descent, Newton's method, Gauss-Newton algorithm for quadratic cost functions, amongst others.

As a result of the real-time requirements in control, special care has to be taken when designing NMPC formulations. Possible solutions to circumvent convexity issues are relaxations and local parametrizations that render the problem at least quasi-convex (gradient towards the optimum), combined with initialization or warm-starting in the local neighborhood of the global optimum. Such techniques are already used to model rotations, e.g. through tangent-space formulations for quaternions. However, sometimes it is also possible to apply a solution even though it is suboptimal, as long as it is feasible. This is especially true for NMPC and planning approaches, where it is preferable to apply suboptimal but feasible (and stable) candidate solutions than discarding solutions and jeopardizing stability.

These challenges have been faced and discussed specifically in my Paper A, where I present an approach to time-optimal planning that always generates feasible solutions but can not guarantee global optimality. Special cases are demonstrated where a trivial initial guess can change the solution from local to global optimality.

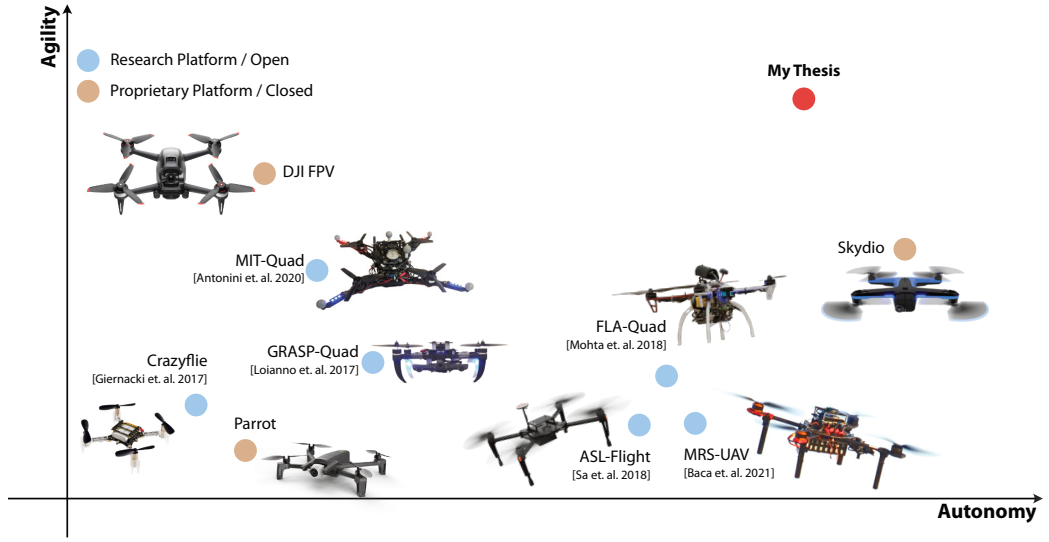


Figure 1.3: A qualitative visualization of existing autonomous quadrotor platforms used for research, industrial and commercial applications. The open-source frameworks FLA[152], ASL[206], and MRS[13] have relatively high autonomy capabilities but low agility. While the DJI[53], Crazyflie[86], and Parrot[208] have some supportive autonomy features, they do not allow autonomous deployment in any form. The MIT[10] and GRASP[130] platforms support a variety of autonomous flight functionality with mediocre agility, but are intended for research purposes and, in the case of the MIT-quad, limitted to instrumented environments. Finally, the Skydio[217] is provides multiple fully autnomous application capabilities for commercial and industrial applications at a reasonable level of agility. The goal of this thesis (top-right) is to provide methods that push the state of the art towards a higher level of autonomy while significantly increasing the agility and flight performance envelope.

1.2 Related Work

Over the past decade, research on autonomous, agile quadrotor flight has continually pushed platforms towards higher auonomy, speeds, and agility [92, 147, 86, 208, 217, 151, 130, 58, 152, 206, 260, 125, 171, 176, 13] (Figure 1.3).

To further advance the field, several competitions have been organized—such as the autonomous drone racing series at the recent IROS and NeurIPS conferences [153, 38, 110, 140] and the AlphaPilot challenge [88, 68]—with the goal to develop autonomous systems that will eventually outperform expert human pilots.

This chapter summarizes the state of the art in control, planning, and estimation for agile quadrotor flight. It also indicates how the work of this thesis relates to and improves on state-of-the-art methods.

Figure 1.4: When flying complex agile maneuvers through structured or unstructured environments, accurate control, reactive replanning, and versatile objective descriptions are required.

1.2.1 Control

As most aerial (and ground) vehicles, quadrotors are highly unstable systems with a cascade of non-linear underactuated dynamics. Feedback control is required to stabilize such systems on static or along time-varying references. To guarantee stability at the limit of the systems capabilities, the control architecture must be capable to exploit the full non-linear system and respect actuation limits. On the other hand, to react to disturbances, changes in the environment, or new observations, the control architecture must adapt and optionally replan with minimal latency. There exists a variety of approaches which can be split in three paradigms: cascaded geometric control, linear-quadratic regulator, and model-predictive control. The following paragraphs will establish the related state of the art by going bottom-up through those three categories.

Geometric Control Following a bottom-up approach, early research first considered the stabilization of the quadrotor's attitude dynamics [27, 28] and later on extended to full state control [29]. Preceding approaches exploit the time-scale separation, and differential flatness of these systems [150, 141]. They rely heavily on cascaded control schemes [95, 149, 58] and *decouple* the rotational and translational dynamics via a geometric tracking controller as described in [122, 149]. Often used in conjunction with analytic trajectory planning [149, 160], these approaches are computationally extremely efficient. However, geometric approaches can only incorporate severly simplified actuation constraints, requiring them to stay within conservative margins from the performance limits. Furthermore, they are based on momentary state feedback without any predicitve capabilities to anticipate upcoming flight state changes, feasibility violations, or non-linear system dynamics.

Linear-Quadratic Regulators Based on Kalman's introduction of the linear-quadratic regulator [107], first approaches to optimal control for quadrotors linearized the system at a given stable time-invariant state [195] or use a precomputed library of state-dependent LQR gains [194]. However, these approaches cannot adjust to the full state space, time-varying state or input costs, non-linear dynamics, or changing system parameters at

execution time. Moreover, they often separate orientation and position control, similarly to the aforementioned geometric control. On quadrotors, this implies that direct control over the attitude itself is lost and the demanded acceleration is subject to several limitations in change rate, magnitude, and direction.

Paper H proposes a unified LQR for the rotational and translational quadrotor subsystem by relinearizing the system at the current state and providing time-varying optimal LQR gains for compute-constrained platforms.

Model-Predictive Control However, all approaches using LQR suffer from two fundamental problems: they neither respect any state or input constraints nor predict the non-linear state evolution. Model-predictive control [193] is based on minimizing a quadratic cost over a receding finite horizon subject to the robot's system dynamics as well as state and input constraints. The advantages of MPC are many: the system is linearized around each predicted time point, actuator constraints can be accounted for or even compensated, and cost functions can not only include system states and inputs but also employ alternative measures defining a task or objective. On the other hand, MPC is computationally much more demanding than geometric or LQR control, since it relies on repeatedly solving a constrained non-linear optimization problem spanning multiple timestep integrations of the dynamic system. Therefore, modern approaches apply a real-time iteration scheme [52, 99, 253], updating control inputs and initial state between each iteration of the numerical solution process, instead of using the result only after convergence is reached. The intention behind this scheme is that the iterations converge faster than the systems states change, where the problems contractivity has been proven in [52] and the stability of the approach is shown for non-linear constrained systems in [253].

For the specific case of quadrotor control, early works on MPC decouple the translational and rotational state space, similar to previous geometric and LQR control, to simplify the problem and computational requirements [191, 109, 252, 210], which, while suboptimal, proved feasibility. Following the early success, non-linear methods have been established [16, 108, 169, 36, 171] and demonstrated superior performance. However, all approaches approximate, simplify, or split the dynamic system to make the computation tractable onboard the quadrotor, or run offboard.

Based on the previous work in quadrotor control [72, 59], Paper B proposes a complete open-source control architecture with a non-linear model-predictive control approach running onboard and including system dynamics and constraints down to single-rotor thrusts for multiple platforms and tasks. This architecture allows accurate tracking of highly-agile maneuvers and was used to execute the trajectories from Paper A, and led the foundation of my further collaborative work in [239, 19, 227, 197].

Perception-Action Coupling All the aforementioned works advanced the field in providing novel, more accurate, and versatile control methods. However, none of those works account for the fact that mobile robots directly affect their sensor measurements through their egomotion. This perception-action-coupling implies that the measurements and their quality are affected by the robot's actions and can therefore be controlled. Most approaches to MPC use state and input, waypoint, or analytical spatial references to define their objectives. However, analytical expressions of sensor models allows one to include them in the optimization and compromise between action and perception objectives.

Especially on aerial robots using visual and inertial sensing modalities, egomotion can lead to motion blur or adversely affected tracking. There exist a handful of approaches to planning and control that include certain aspects of the perception system.

These systems are either based on information gain [41, 77], reprojection and field-of-view limitations [215, 223, 183, 188], or view-point optimization for cinematography [164, 165]. However, while [183, 215] are pure offline planning approaches and incapable of running in real time for replanning, [223, 41, 77] do not account for the system dynamics and underactuation of quadrotors, and [188] neglects reprojection velocity and therefore motion blur.

Paper G improves on these works by proposing an MPC approach, which includes a pinhole camera model to optimize perception objectives such as reprojection of a point-of-interest into the camera frame. It achieves this within a real-time control framework, which optimizes both the translational and rotational trajectory, and significantly supports visual tracking systems in difficult scenarios.

1.2.2 Planning

If a quadrotor's task consists of visiting multiple waypoints (delivery and transportation [192], inspection, drone racing [153, 38, 140]), doing so in minimal time is often desired, and, in the context of search and rescue or drone racing, even the ultimate goal. For simple point-mass systems, time-optimal trajectories can be computed in closed-form, resulting in bang-bang trajectories [121, 186]. However, quadrotors are underactuated systems [29, 141] introducing a coupling in the achievable linear and rotational accelerations, which renders time-optimal planning extremely challenging.

Two common approaches for planning quadrotor trajectories exist, continuous-time polynomials and discrete-time state-space representations. Polynomial formulations [141, 149, 159] exploit the quadrotor's differential-flatness with high computational efficiency. However, the smoothness of those polynomials limits the rate of state or input changes and only reaches the input limits for infinitesimal short durations or constantly for the entire trajectory time, rendering them time-suboptimal. On the other hand, time-

discretized trajectories can be found using search and sampling-based methods [245, 4, 127, 258] or optimization-based methods [90, 85, 12, 183]. Unfortunately, sampling the 4-dimensional continuous input space over many discrete time steps with sufficient resolution is computationally intractable. Because of this, prior work [245, 4, 127, 258] restores to point-mass, polynomial, or differential-flatness approximations, and therefore does not handle realistic actuator constraints.

Therefore, planning time-discretized trajectories with optimization-based methods is the only viable solution in the short-medium term. For a time-optimal solution, the trajectory time is part of the optimization variables and is the sole term in the cost function. However, if multiple waypoints must be passed, these must be allocated as constraints to specific nodes on the trajectory. This time allocation is a priori unknown since the time spent between any two waypoints is unknown, which renders traditional discretized state space formulations ineffective for time-optimal trajectory generation.

The two earlier works [94, 131] extend the bang-bang approach but are restricted to 2-dimensional maneuvers. Another approach is taken in [222], where a change of variables along an analytic reference path is used, but the platform limits are simplified to collective thrust and bodyrates, neglecting realistic actuator saturation. A completely different approach is followed in [204], where the segment times of a polynomial trajectory are refined based on learning a Gaussian Classification model predicting feasibility. However, this approach is still constrained to polynomials and only refines the execution speed of a predefined trajectory rather than modifying the trajectory itself.

To overcome the limitations of previous works, Paper A investigates this problem and provides a solution that allows simultaneously optimizing the trajectory and waypoint allocation in a given sequence, exploiting the full actuator potential of a quadrotor. It presents a novel formulation based on a progress measure for each waypoint along the trajectory, indicating completion of a waypoint together with a complementary progress constraint [42] that allows completion only in proximity to a waypoint.

1.2.3 Estimation

Visual Inertial Odometry Recent advances in robot perception have led to several visual-inertial odometry (VIO) systems becoming more robust and accessible solutions for state estimation and navigation, such as [157, 123, 162, 25, 56, 241, 128, 76, 189, 48, 126, 225].

The approaches typically consist of a frontend responsible for visual measurement extraction and a backend responsible for pose estimation. There exist two common approaches for the backend: filtering and sliding-window estimation. Filter-based approaches such as [157, 25, 128, 76] are typically implemented as extended Kalman filter and provide high robustness at low computational costs. On the other hand, sliding-window-based

approaches like [123, 189, 126, 225] can achieve higher accuracy and more versatile measurement models but also incur far higher computational costs. These approaches commonly describe and interprete the underlying estimation problem as a factor-graph [46] to retrieve a maximum-a-posteriori estimate, which boils down to solving a non-linear least-squares problem under the typically assumption of Gaussian noise distributions. To render the resulting non-linear least-squares optimization problem computationally tractable, many approaches employ IMU preintegration, first proposed in [138] to avoid repeated integration of the state-dependent IMU measurements, and later modified in [74] to address the manifold structure of the rotation group.

Meanwhile, the frontend is responsible for providing feature tracks that can be used as visual measurement residuals in the backend. Most pipelines use Harris [91], Shi-Tomasi [205], or FAST features [200] for feature tracking, and ORB [202], SIFT [135], or SURF [20] as descriptors. There also exist GPU-accelerated implementations within OpenCV [30] and ArrayFire [249]. However, neither of them guarantees spatial feature distribution. The reason why it is interesting to have fast implementations is the fact that feature tracking and detection profits in accuracy and robustness if the changes between subsequent images are small. Given that the robot's egomotion can desirably be rather fast, high framerates are the only remaining option to reduce apparent motion, which also implies that the frontend (and optionally the backend) have to run at a high rate with low latency. Additionally, modern image sensors can provide high light sensitivity, allowing for short exposure times, which lowers motion blur and benefits high frame rates.

However, state-of-the-art implementations do not yet provide solutions to deal with high frame rates on resource-constrained platforms. Therefore, Paper D aims at providing exactly this capability to modern embedded robotic compute platforms with integrated small but efficient GPUs. It achieves more than 1000Hz feature tracking and enables future applications in fast and agile flight scenarios.

Additionally, Paper H, G, F, C, and B make use of state-of-the-art VIO implementations.

Disturbance Force Estimation While the aforementioned approaches to visual-inertial odometry enable autonomous navigation, they all neglect the robot's dynamics and therefore cannot sense external forces or disturbances and do not consider the fundamental distinction between the desired motion and unwanted perturbations. Adding the system dynamics to a VIO system (i)) allows the perception of an external force acting on a robot, and (ii)) adds information to the estimation problem, resulting in increased accuracy.

There are two scenarios in which one might be interested in estimating external forces: model identification and interaction with the environment. Existing approaches typically use an estimator loosely-coupled with an odometry system [237, 250, 203, 145, 11,

231], introducing latency, computational overhead, and neglecting correlation among the estimated variables and their noise characteristics.

Previous approaches on external force estimation can be split into two groups: deterministic and probabilistic. Deterministic approaches estimate external force by subtracting the collective thrust from the inertial measurements [237] or employ decoupled estimators [250, 203], but fail to correctly account for actuation and measurement noise and bias. Probabilistic approaches employ loosely-coupled filters [11, 1, 145, 231] but neglect correlations between all state variables and introduce latency and computational overhead.

On the other hand, several approaches have tried to integrate the robot's system dynamics into state estimation but had moderate success. [9] combined the idea of incorporating dynamic factors for localization of UAVs from [230] with the preintegration scheme from [74] to develop a model-based visual-inertial state estimator but without considering external forces. Their implementation showed improved robustness, especially in slow-speed flights when accelerometer measurements have a low signal-to-noise ratio. But, as shown in [1] this approach tends to wrongly adjust the IMU biases under the presence of wind and external forces, and therefore only works in a disturbance-free environment, as confirmed by the authors in [9]. Following a completely different approach, [117] proposed to use Dynamic Differential Programming to estimate the state, parameters, and disturbances (forces) in a synthetic planar motion example, assuming perfect data association, which did not translate to real-world applications.

Differently from [117], Paper E extends an optimization-based VIO framework with motion factors to simultaneously estimate state and external force in real time on real-world data. Those external force estimates can be used to detect and control interactions or measure unmodelled (residual) dynamics. The latter is especially interesting for model-based control methods such as MPC since it allows to measure the forces resulting from unmodeled effects. Models can then be refined by regressing or validating new model hypotheses or reacting against the measured disturbances. As shown in some of my collaborative work, such model adaptions can include deterministic, probabilistic [239], or even learned [19] representations.

1.2.4 Drone Racing

Drone racing requires fast navigation through a series of waypoints given by gates. To complete and win a race, the navigation system must achieve two performance measures: reliable race completion without crashing and the fastest laptime. The first measure requires robustness and precision when passing the gates, relying on accurate localization relative to the gates. The latter goal of achieving the best laptime can be solved by finding the time-optimal trajectory, which has already been addressed in Section 1.2.2 and an offline solution is presented in Paper A. What remains are the questions of

how to do accurate gate-relative localization and online trajectory (re-)planning on compute-constrained platforms.

Traditional approaches to aerial autonomous navigation build on visual-inertial odometry as described in Section 1.2.3. While these methods can be used to perform visual teach and repeat as in [64], they assume a static world and accurate pose estimation: assumptions that are commonly violated in the real world. Furthermore, 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 [47].

In [124], a handcrafted process is used to guide the drone towards the visible gate. While the approach is computationally very lightweight, it struggles when multiple gates are visible and does not allow more sophisticated planning and control algorithms. However, recent advances in deep-learning enabled alternative approaches to autonomous navigations, which typically predict actions directly from images. Output representations range from predicting discrete navigation commands (classification in action space) [114, 87, 134] to direct regression of control signals [161], or regressing cost functions [54].

Another line of work proposed intermediate representations as in [103, 111]. While [104] uses line-of-sight tracking and therefore always needs to see the next gate, [111] directly regressed high-level steering data from images. As an advantage, the latter method can exploit track-specific context but also requires a large amount of track-specific training data, rendering it difficult to deploy in the real world.

I addressed these shortcomings in a seminal collaborative work on drone racing in Paper F, which won first place in the "2018 IROS Autonomous Drone Racing Challenge". As opposed to existing work, our approach combines an intermediate measurement representation in the form of learned gate pose detections with an apriori known but inaccurate gate map and a VIO/EKF/MPC combination. It operates reliably even when no gate is in sight while eliminating the need to retrain the perception system for every new track, enabling rapid deployment on complex tracks. However, the perception system does not perform well when multiple gates are visible, as is frequently the case for drone racing, and the navigation strategy had no notion of even approximately time-optimal planning.

Assuming knowledge of the platform state and the environment, there exist many approaches which can reliably generate feasible trajectories with high-efficiency [149, 160]. Other approaches additionally incorporate obstacle avoidance [258, 84] or perception constraints [59, 221]. But for time-optimal planning, there exists only a handful of approaches [94, 131, 204] including my work in Paper A. However, while [94, 131] are limited to 2D scenarios and only find trajectories between two given states, [204] requires simulation

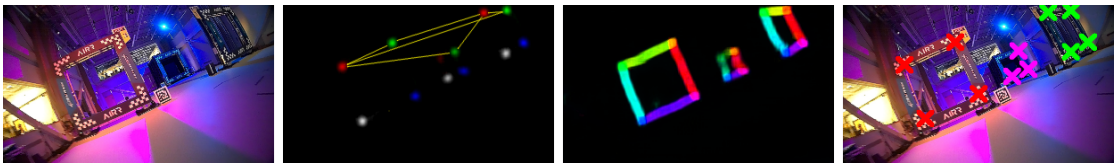


Figure 1.5: Visualization of the gate detection procedure in Paper C. Left: the capture image. Mid-left: detected corners with a few possible corner connections visualized as yellow lines. Mid-right: part-affinity fields to resolve matching corner connections. Right: final detections visualized on the original image.

and real-world data obtained on the track, and the method of A is not applicable due to computational constraints.

To address the shortcoming of multiple gate observations and efficient onboard planning for drone racing, my collaborative work in Paper C improves over Paper F with a new measurement modality capable of detecting multiple partially visible gates (Figure 1.5), and an approximately time-optimal planning algorithm.

2 Contributions

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

In total, this research has been published with my first-authorship in three journal publications (two in *Science Robotics*, one in *Springer: Autonomous Robots*) and three peer-reviewed conference publications, and with my co-authorship in five peer-review conference publications and four journal publications (two in the *IEEE Transactions on Robotics* and two in the *Robotics Automation Letters (RA-L)*). A complete list of all publications can be found on Page ix.

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*.
- *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 F.
- *Robotics: Science and Systems (RSS) 2017*, **Best Student Paper Award Finalist** with my paper on "Fast Trajectory Optimization for Agile Quadrotor Maneuvers with a Cable-Suspended Payload" [70].

Software:

- **Agilicious Flightstack:** Paper B
  available at https:agilicious.dev
- **Time-Optimal Planning:** Paper A
  available at https://github.com/uzh-rpg/rpg_time_optimal
- **Perception-Aware MPC:** Paper G
  available at https://github.com/uzh-rpg/rpg_mpc
- **Visual-Inertial Model-based Odometry:** Paper E
  available at https://github.com/uzh-rpg/vimo
- **GPU-Accelerated Corner Tracker:** Paper D
  available at https://github.com/uzh-rpg/vilib

2.1 Time-Optimal Planning for Quadrotors

In the first part of this thesis, I consider the problem of generating time-optimal trajectories for quadrotor flight. Fast and robust flight requires the ability to generate trajectories that exploit the full actuation space of the quadrotor without exceeding actuation limits. Therefore, it is important to understand how actuation limits restrict and influence the trajectory and then take advantage of a methodology that can handle and exploit them. A common task in aerial navigation is the flight through a series of waypoints, as necessary in inspection, observation, delivery, and drone racing. This task sets the context for the research question of "What does it mean for a quadrotor to fly agile and how can we generate fast flight paths?".

One fundamental property of quadrotors is their underactuated system dynamics. Since their four rotors lie in a single plane, they generate torque in all three spatial dimensions but collective thrust only in a single direction. In order to control translational acceleration, velocity, and position, the vehicle has to change its orientation. This renders the system differentially-flat, where all states and inputs can be expressed through a differentiable formulation of the position and yaw subspace, as exploited in many works so far [149, 159, 58]. On the other hand, the quadrotor's underactuation also couples the extrema of linear and angular acceleration through the actuator limits. This poses the question of *if* and *how* we can find the optimal trade-off between linear and angular acceleration.

The works [149, 159, 58] exploit the differential-flatness to describe quadrotor trajectories as four individual polynomials in position and yaw parametrized by time. Those polynomials are extremely efficient to compute but bear some disadvantages. Since the polynomials represent decoupled axes of a non-linear system, it is difficult to directly relate them to the actual single-rotor thrusts and renders application of actuator constraints extremely challenging. In fact, the aforementioned works vastly relax the real single-rotor constraints to higher-level collective-thrust and bodyrate constraints, which either leads to overly conservative or infeasible solutions. Another challenge is presented by the fact that polynomials only reach extrema in points or constantly but can't reach an extremum for a partial duration, limiting the solution-space to provable sub-optimality.

My work is therefore focused on discretized trajectories combined with numerical optimization. Contrary to polynomials, discretized trajectories allow for both arbitrary state and input changes between individual time steps and enforcing realistic system dynamics on any desired level. However, they also bear a challenge: to formulate the task of passing through multiple waypoints, they must be assigned to one or a subset of the discretized states. Since for time-optimal planning, the actual time at which a waypoint is passed is a priori unknown, this assignment is not possible, and a different method to enforce waypoint passing without time-allocation needs to be researched.

The following work proposes a solution to these challenges.

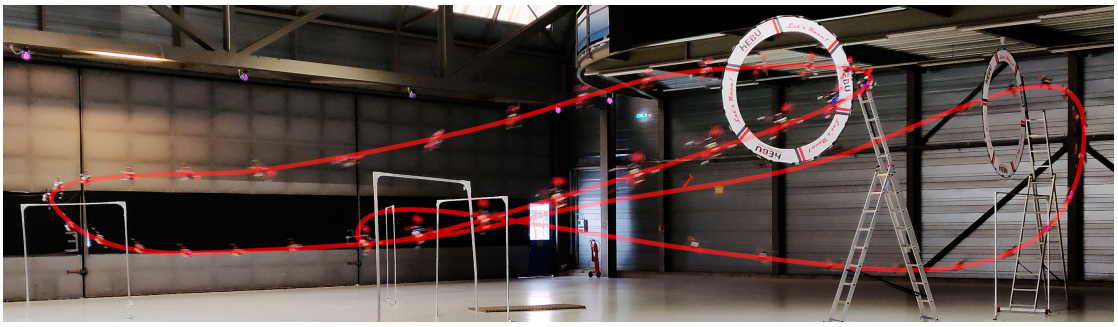


Figure 2.1: This time-optimal flight path was computed using the proposed complementary progress constraints (CPC) and outperforms expert human piots.

2.1.1 Paper A: Time-Optimal Planning for Quadrotor Waypoint Flight

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. URL: https://robotics.sciencemag.org/content/6/56/eabh1221

In this work, I propose a novel method for time-optimal waypoint flight (see Figure 2.1) for quadrotors at their actuation limit. For the first time, this method is able to generate trajectories that beat expert human drone racing pilots. Due to the suboptimality of polynomials, my method relies on numerical optimization with a discretized trajectory formulation, including the system dynamics on a single-rotor-thrust level through a multiple-shooting scheme. This allows to account for realistic dynamics and actuator constraints, as opposed to approximations used in other works to date [94, 131, 222, 204], but requires waypoints to be allocated as costs or constraints at specific discrete times. To generate truly time-optimal trajectories, I propose a solution to the time allocation problem by introducing a formulation of progress along the trajectory, together with a complementary progress constraint. More specifically, I formulate two factors that must complement each other, where, in my case, one factor is the completion of a waypoint (progress), while the other factor is the local proximity to a waypoint. This enables the simultaneous optimization of the time-allocation and the trajectory itself. I compare my method against related approaches [94, 131] and validate it in real-world flights with the control strategies developed in Section 2.2, where I outperform human expert drone pilots in a drone-racing task. Finally, I discuss practical deployment limitations, which are further addressed in the collaborative work in 2.2.4, the sub-optimality of polynomials, and the convexity and convergence properties of my formulation, supported by a multitude of simulation experiments.

My specific contributions to this work include the complete problem formulation, the derivation of the proposed complementary progress constraints, as well as the simulation and real-world experiment design, execution, and evaluation.

Related Software

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

Related Video

(V1) https://youtu.be/ZPI8U1uSJUs

2.2 Optimal Control for Quadrotors

Classic control strategies rely on multiple cascaded feedback loops, each with its own set of assumptions, approximations, and parameters. In the case of a non-linear or time-varying system, this necessitates extremely conservative performance for robustness or jeopardizes the closed-loop system stability. However, to reach a level of robustness and agility that meets the modern requirements for safety and profitability, navigation systems must be capable of exploiting the non-linear system dynamics while respecting the actuator limits to ensure stability and feasibility and dealing with complex task formulations. Therefore, this part of the thesis will answer the research question of "How can we simultaneously achieve robust, versatile, and agile control for fast and reliable mission execution with drones?" and "Is it possible to exploit the coupling between action and perception on mobile robots?".

To achieve this, I will investigate and leverage model-based control using numerical optimization and apply them to the quadrotor system in a three-step approach: i) Paper H first improves on cascaded control by proposing a quadratic cost formulation and an approximation of the non-linear system through a time-varying linear-quadratic regulator; ii) Paper G then advances to non-linear model-predictive control to accurately represent the non-linear dynamics over a moving horizon, account for simplified actuation limits, and include perception models; iii) Paper B finally refines the model-fidelity and includes the real single-rotor actuation constraints, and proposes a complete software and hardware system for agile and robust flight. Based on these three contributions, I further developed multiple improvements in collaboration with my colleagues, ranging from rotor-failure handling over comparative studies to model-predictive contouring control, all summarized in 2.2.4.

2.2.1 Paper H: Onboard State Dependent LQR for Agile Quadrotors

Philipp Foehn and Davide Scaramuzza. "Onboard State Dependent LQR for Agile Quadrotors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018. DOI: 10.1109/ICRA.2018.8460885

This first contribution acts as a preliminary investigation of the advantages of task description and tuning using quadratic state and inputs costs, together with an improved approximation of the non-linear system over traditional cascaded strategies. I propose an LQR controller, which: (i)) is linearized depending on the quadrotor's state; (ii)) unifies the control of rotational and translational states; (iii)) handles time-varying system dynamics and control parameters. I verify the approach in four experiments: (i)) controlling at hover state with large disturbances; (ii)) tracking along a trajectory; (iii)) tracking along an infeasible trajectory; (iv)) tracking along a trajectory with disturbances. All experiments were done using only onboard state estimation and LQR computation, and show good tracking performance, versatile deployment, and are computationally easily tractable. This work laid the foundation for the following contributions, which advance this preliminary investigation to a non-linear model-predictive control strategy.

This work consist purely of my own contributions including the methodology and experiments.

Related Videos

(V2) https://youtu.be/8OVsJNgNfa0

2.2.2 Paper G: PAMPC: Perception-Aware Model Predictive Control for Quadrotors

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

To fully exploit the non-linear dynamics and incorporate the actuation constraints of a quadrotor, this work advances from the simplified infinite-time LQR formulation of Paper H to a moving-horizon model-predictive control formulation. Furthermore, we also leverage the MPC's capabilities to exploit the coupling between action and perception objectives of a quadrotor vehicle to allow for robust and reliable sensing. Considering both perception and action objectives for motion planning and control is challenging due to the possible conflicts arising from their respective requirements. For example, for a quadrotor to track a reference trajectory, it needs to rotate to align its thrust with the direction of the desired acceleration. However, the perception objective might require minimizing such rotation to maximize the visibility of a point of interest. MPC is not only capable of accounting for the non-linear system dynamics and actuation constraints but through its quadratic cost formulate also enables the inclusion of such perception costs

and trading-off multiple objectives. The result is a perception-aware model predictive control framework, which works in a receding-horizon fashion by iteratively solving a non-linear optimization problem. It is capable of running in real-time, fully onboard a lightweight, small-scale quadrotor using a low-power ARM computer, together with a visual-inertial odometry pipeline. The approach is validated in experiments demonstrating (i)) the conflict between perception and action objectives, and (ii)) improved behavior in extremely challenging lighting conditions. This approach was also used in the drone racing demonstrator Section 2.4.1, where it enabled both agile flight and focusing on specific perception targets. However, this approach still uses a separate low-level controller and therefore approximates the actuation constraints as maximal collective thrust and bodyrates. This is a simplification of the real single-rotor thrust constraints, which can still lead to reduced performance in agile maneuvers, and is therefore alleviated in the next contribution.

My contributions to this work include the formulation and design of the MPC, its implementation, as well as the experimental evaluation. The idea for the problem, as well as the derivation of the perception model was done together with my coauthor Davide Falanga.

Related Software

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

Related Video

(V3) https://youtu.be/9vaj829vE18

2.2.3 Paper B: 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

Based on the insights from previous contributions in Paper G and the demonstrator in Paper C, it became clear that there is a necessity for a streamlined control pipeline, a unified hardware platform, and an NMPC approach with higher model-fidelity. Therefore the contribution in this paper is threefold: i) I extend the previous NMPC formulation from Paper G by changing the actuation constraint to the true single-rotor thrust constraints; ii) I propose a software architecture and a complete hardware design that is targeted at agile flight, which allows the seamless transition between different control architectures, platforms, and even deployment in multiple simulations and the real world; and iii) we make it available open-source and open-hardware to further support the community working on (agile) quadrotor flight. Figure 2.2 shows a condensed overview of

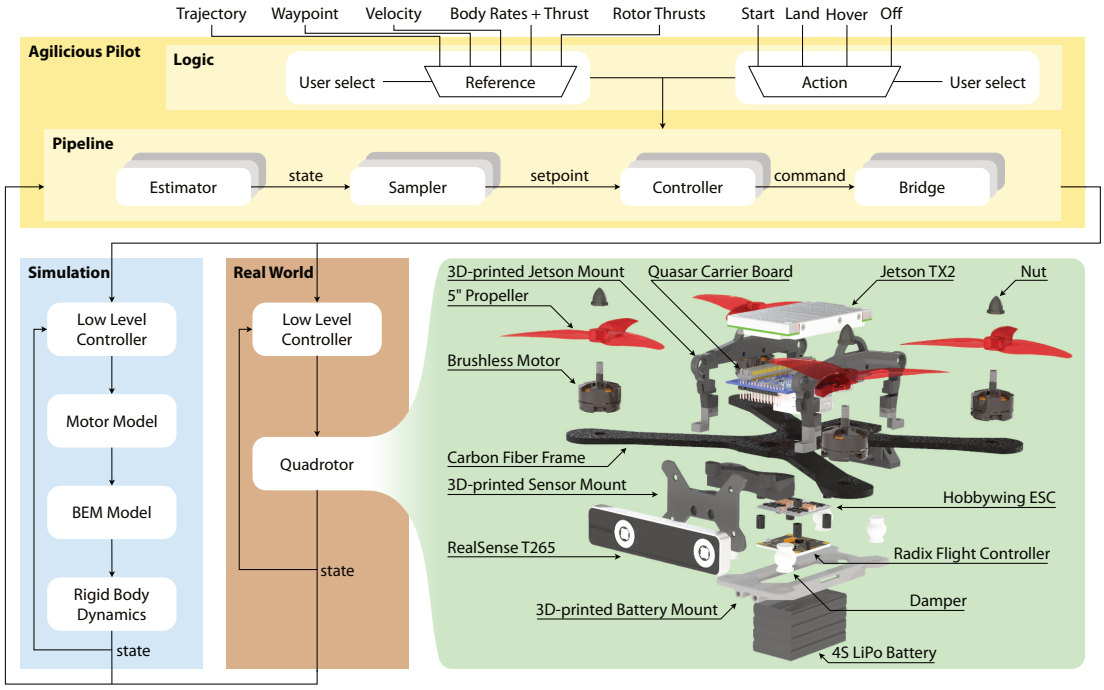


Figure 2.2: An overview of the software and hardware in my open-source flight stack called *Agilicious*. It provides all required resources to start research and development on state-of-the-art agile aerial autonomy, including a versatile control pipeline allowing fast prototyping and a hardware platform capable of aggressive flight maneuvers.

the open-source soft- and hardware of my *Agilicious* flight stack.

The NMPC proposed in this work finally accounts for the non-linear system dynamics down to the level of rotational and translational accelerations, incorporates single-rotor thrust constraints, allows flexible task formulations including stable setpoint references, polynomial and sampled time-varying reference, and even approximative tracking of infeasible references, all while running onboard the proposed hardware platform at more than 200Hz. Furthermore, even though it uses the single-rotor thrusts as an input modality, it can also be used for bodyrate and collective thrust commands or allows to be extended with further low-level controllers.

My contributions to this paper include the complete software architecture, implementation of multiple software modules such as the MPC and estimators, software utilities, as well as the evaluations presented in the experiment section. Furthermore, I helped in the development of the low-level flight controller, designing the hardware concept and evaluating hardware components.

This contribution laid the foundation for countless further studies evaluating and improving on the proposed control architecture. All studies that I have been collaboratively involved in are listed and explained in the following Section 2.2.4.

Related Software

(S3) https://agilicious.dev and https://github.com/uzh-rpg/agilicious

2.2.4 Additional Contributions

In the following works, my contributions are limitted to the design of the discussed MPC implementations together with my coauthors, as well as software architecture contributions in Paper B, and help with the design and evaluation of experiments.

A Comparative Study of non-linear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight

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.* (2021). arXiv: 2011.11104 [cs.RO]

In this collaborative study we compare the non-linear model-predictive control (NMPC) from Paper B and differential-flatness-based control (DFBC) and demonstrate under which conditions NMPC is superior. The study evaluates both methods with and without an additional low-level controller using incremental non-linear dynamic inversion (INDI). It presents both simulation and real-world comparisons, ablates the performance on feasible and infeasible trajectories, and under varying levels of latency and model mismatch. We conclude that NMPC in combination with a fast low-level controller and given accurate models and low-latency systems is superior in all practical tasks. However, we also find that latency and large model mismatch can have a higher impact on NMPC methods than on DFBC.

Related Video

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

Model Predictive Contouring Control for Near-Time-Optimal Quadrotor Flight

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 [cs.RO]

In this work, we again consider the task of time-optimal flight through multiple waypoints or along a given reference. While my contributed Paper A is capable of producing truly time-optimal trajectories, it requires computation times of up to an hour. Furthermore, the precomputed trajectories are only practically feasible if the platform has some margin for control in the actuator constraints to counteract disturbances. This implies that the method is not well suited for dynamic, unknown, or uncertain environments. To deal with these limitations, we develop a method that reaches similar performance compared to Paper A but does not require any compute-intensive planning. This is achieved by formulating a model-predictive contouring control (MPCC) problem. Different from

MPC which tracks a predefined time-based trajectory, MPCC tracks a spatial path and maximizes the progress along this path online. Through this formulation, it is possible to simultaneously maximize the progress and therefore fly as fast as possible while also accounting for the dynamics and actuation constraints. Furthermore, it is more robust to disturbances since it does not track fixed time-based reference. We show how this control strategy approximates the time-optimal solutions from Paper A and how it can be used to achieve better performance in real-world deployment.

Related Video

(V5) https://youtu.be/mHDQcckqdg4

Nonlinear MPC for Quadrotor Fault-Tolerant Control

Fang Nan, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. "Nonlinear MPC for Quadrotor Fault-Tolerant Control". In: *IEEE Robot. Autom. Lett.* 2022

In this collaborative publication, we extend the capabilities of the MPC in Paper B to a fault-tolerant control formulation. This work is based on the problem implied by the unstable and underactuated dynamics of quadrotors in combination with a rotor failure. To address this problem, we propose a fault-tolerant controller using the NMPC to stabilize and control a quadrotor subjected to the complete failure of a single rotor. Differently from existing works that either rely on linear assumptions or resort to cascaded structures neglecting input constraints in the outer-loop, our method leverages full nonlinear dynamics of the damaged quadrotor and considers the thrust constraint of each rotor. It can track agile reference trajectories with and without rotor failure, transition into and from failure states, and even recover from extreme failure states, such as upside-down and agile maneuvers. Our approach was verified in extensive simulations and real-world experiments, which demonstrates that the proposed method can effectively recover the damaged quadrotor even if the failure occurs during aggressive maneuvers, such as flipping and tracking agile trajectories. Therefore it significantly contributes to the safety of future systems and allows to avoid catastrophic damage while not restricting the flight envelope, paving the way for the deployment of agile navigation approaches.

Related Video

(V6) https://youtu.be/Cn_836XGEnU

Performance, Precision, and Payloads: Adaptive non-linear MPC for Quadrotors

Drew Hanover, Elia Kaufmann, Philipp Foehn, 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

This collaborative work addresses the problem of model-mismatch when deploying MPC on real systems. While the control strategy developed in the context of Paper B exploits its predictive nature to accurately incorporate non-linear models and actuator constraints, it suffers from performance degradation if there is a significant mismatch between the modelled and the real-world dynamics. Therefore, we propose $\mathcal{L}_1$-NMPC, a novel hybrid adaptive NMPC to learn model uncertainties online and immediately compensate for them, drastically improving performance over the non-adaptive baseline with minimal computational overhead. Our proposed architecture generalizes to many different environments from which we evaluate wind, unknown payloads, and highly agile flight conditions. The proposed method demonstrates improved tracking performance while providing immense flexibility and robustness.

Related Video

(V7) https://youtu.be/8oB1rG5iYc4

2.3 State-Estimation and Modelling for Quadrotors

The previous contributions all considered the planning and control problem of a quadrotor assuming accurate dynamic models. In reality, however, accurate models are difficult to acquire and or even impossible to represent given the aerodynamic (and partially chaotic) effects on aerial vehicles. Therefore, it is important to identify the predictable contributions of such high-complexity models and compensate or suppress disturbances. This is possible in two steps: i) first it is necessary to be able to establish an estimate or measure of the residual (unmodelled) dynamics, ii) then it is possible to create a model that captures those dynamics and enables improved control performance.

To do so, this chapter first considers the research question of "Is it possible to exploit the coupling between action and perception on mobile robots?" from the perspective of state estimation for external force detection (Figure 2.3), and then extends to possible approaches to model external forces caused by aerodynamic effects.

In fact, given a model of the vehicle, we can include its dynamics into a visual-inertial odometry (VIO) pipeline and simultaneously estimate the system state and the external or residual forces. For this, I propose a sliding-window-based VIO backend, which includes the modelled actuated dynamics into the perception system, followed by a proposal of a fast feature detection frontend to help visual-inertial odometry approaches to transition into the regime of fast and agile flight.

Finally, in collaborative contributions, two models capturing aerodynamic effects are proposed, one focused on accurate simulation (Section 2.3.3), completed by a second one focused on realtime inference in an MPC framework (Section 2.3.3).

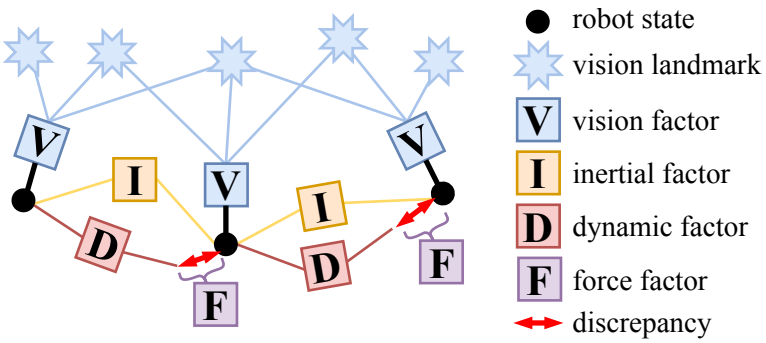


Figure 2.3: A simplified illustration of a factor graph with included dynamics and residual terms. Different from classic VIO approaches, the graph does not only include visual (V) and inertial (I) factors, but also the model dynamics (D) and the residual forces (F) from unmodelled effects such as aerodynamics, interaction, or disturbances.

2.3.1 Paper E: VIMO: Simultaneous Visual Inertial Model-based Odometry and Force Estimation

Barza Nisar, Philipp Foehn, Davide Falanga, and Davide Scaramuzza. "VIMO: Simultaneous Visual Inertial Model-based Odometry and Force Estimation". In: *Robotics: Science and Systems (RSS)*. 2019

In this paper, I present an approach to visual-inertial odometry capable of exploiting the robot's dynamics and known actuation inputs, and differentiating between desired motion due to actuation and unwanted perturbation due to external or aerodynamic force. For many robotic applications, such as agile flight, it is often essential to identify or even model the external force acting on the system due to e.g. interactions or unmodelled dynamics. My approach enables this by simultaneously estimating the motion and the effects of external forces or residual dynamics. I propose a relative motion constraint combining the robot's dynamics and the external force in a preintegrated residual, resulting in a tightly-coupled, sliding-window estimator exploiting all correlations among all variables. The results show that my approach increases the accuracy of the estimator and provides external force estimates at no extra computational cost. The accuracy of the force estimates is validated by measuring the groundtruth force of an interaction with a loadcell sensor. This force estimate allows to identify and model the residual aerodynamic effect allows to identify and model the residual aerodynamic effects and exploit such high-fidelity models in simulation or MPC.

My contribution to this work includes the idea for, as well as the derivation of the preintegrated model residual and the real-world experimental validation. Implementation and simulation validation was mainly done by my coauthor Barza Nisar.

Related Software

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

Related Video

(V8) https://youtu.be/t2GdZZp7xQE

2.3.2 Paper D: Faster than FAST: GPU-Accelerated Frontend for High-Speed VIO

Balazs Nagy, Philipp Foehn, and Davide Scaramuzza. "Faster than FAST: GPU-Accelerated Frontend for High-Speed VIO". in: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2020

To enable fast and agile flight outside of instrumented environments, mobile robots need to be able to perform fast, robust, and precise onboard state estimation. VIO prevails with its low cost, complementary measurement modalities, universal applicability, and increasing maturity and robustness, but it is still computationally expensive and introduces significant latency. The latter is especially true for aerial vehicles with size

and weight constraints limiting the available computation power while requiring real-time execution of the VIO and control pipeline to guarantee stable, robust, and safe operation. Besides latency, one may also witness a disconnect between the available sensor capabilities and the actual information processing capabilities of mobile systems. While off-the-shelf cameras are capable of capturing images above 100fps, many algorithms and implementations are not able to handle visual information at this rate.

Moreover, fast and agile flight introduces large apparent changes in the captured image or even leads to motions blur. Both factors cause visual feature tracking to be less precise and robust or even fail completely. However, modern sensors with a good signal-to-noise ratio and high framerates allow reducing exposure time and the apparent motion between consecutive frames, resulting in reduced motion blur a robust tracking, respectively. By lowering the frame processing times, we can simultaneously minimize latency and also reduce the displacement between frames due to fast motions. This paper proposes a novel feature tracking frontend to help visual-inertial odometry approaches to transition into the regime of fast and agile flight. This work first revisits the problem of non-maxima suppression for feature detection, followed by proposing an enhanced FAST feature detector specifically designed to exploit parallelization on small GPUs. Finally, the proposed method is evaluated against other state-of-the-art CPU and GPU implementations, where it always outperforms all of them in feature tracking and detection, resulting in over 1000fps throughput on an embedded Jetson TX2 platform, as used in Paper B. Additionally, this fast frontend is integrated into a VIO pipeline achieving a metric state estimation at ∼200fps.

My contribution to this work consists of the idea and methodology for the approach and experiment design, while implementation and validation have been executed by my coauthor Balazs Nagy.

Related Software

(S5) https://github.com/uzh-rpg/vilib

Related Video

(V9) https://youtu.be/5Ndi9IYpI68

Figure 2.4: A quadrotor in a high-acceleration flight maneuver through smoke to visualize the air flow arround the vehicle. Real-world flight data can be used to estimate and model the residual aerodynamic forces not explained by the rigid-body quadrotor dynamics.

2.3.3 Additional Contributions

The following publications include mainly the work of my coauthors which I have supervised and assisted in the derivation of the methodology and the experimental validation, as well as supported the software development within Paper B.

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)

In fact, aerodynamic forces render accurate high-speed trajectory tracking with quadrotors extremely challenging since they become a significant disturbance at high speeds, introducing large positional tracking errors. Moreover, these effects are specific to the shape and specifications of a vehicle and extremely difficult to model since they typically range from simple linear effects to non-linear and chaotic disturbances, depending on the relative airspeed and the vehicle's operation point, as visualized in Figure 2.4.

In most applications, agile flight also implies fast flight at high average velocity, so fast in fact, that classic first-principle models come to their limits, and aerodynamic effects become the dominant model defect. Accurate modeling is needed to design robust high-performance control systems and enable flying close to the platform's physical limits. Therefore, 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. The resulting model captures the aerodynamic thrust, torques, and parasitic effects with high accuracy and outperforms existing models such as [58]. However, my method is mainly targeted at simulation and does neither provide any smoothness, continuity, nor gradients necessary to embed it in optimization-based MPC. Therefore, the following contribution (Section 2.3.3) provides a simplified model, allowing embedding in an MPC.

Related Software

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

Related Video

(V10) https://youtu.be/Nze1wlfmzTQ

Data-Driven MPC for Quadrotors

Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza. "Data-driven mpc for quadrotors". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3769–3776

To fly at high speeds, feedback control must be able to account for these aerodynamic effects in real-time, necessitating a modeling procedure that is both accurate and efficient to evaluate. In this paper, we therefore present an approach to model aerodynamic effects using Gaussian Processes, which we incorporate into our MPC to achieve efficient and precise real-time feedback control. This allows reducing trajectory tracking errors by up to 70% at high speeds. We verify our method by extensive comparison to a state-of-the-art linear drag model from [58] in synthetic and real-world experiments at speeds of up to 14m/s and accelerations beyond 4g.

Related Software

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

Related Video

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

(a) View from the drone of Paper F

(b) The drone and track of Paper C

Figure 2.5: Scenes of the two racing demonstrators. Paper F in Figure 2.5a is only able to detect the next gate and uses the perception-aware MPC from Paper G to focus and fly through the gates. Figure 2.5b shows the drone and track where the approach of Paper C was deployed. This approach is capable of perceiving multiple gates at once an plan approximate time-optimal trajectories through multiple gates ahead.

2.4 Demonstrators

The following demonstrators have been done in the context of two international drone racing competitions, where Paper F was based on my previous contribution in Paper G, and the challenges met in Paper C inspired the later contributions in Paper A and Paper B.

2.4.1 Paper F 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. URL: https://doi.org/10.1109/ICRA.2019.8793631

This demonstrator was developed in the context of the *2018 IROS Drone Racing Competition* [154, 153], in which I and my team scored first place.

Autonomous drone racing requires precise and fast navigation, which is often difficult to achieve in previously unseen and uninstrumented environments due to the drift of onboard localization methods. Therefore, autonomous navigation algorithms so far required a precise metric map of the environment, which is often highly impractical to acquire. To alleviate this necessity, we propose an approach that can fly a new track in a previously unseen environment without a precise map or expensive data collection. This method exploits the previously contributed method in Paper G by using the perception-aware model-predictive control to focus on the next gate known from a coarse gate map. This allows stable perception of the next gate, an image of which is fed to a convolutional network predicting the pose of the closest gate together with its 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 the perception-aware MPC (Paper G) to quickly and accurately navigate through the track. This is done by tracking straight line segments between the gates, which the MPC uses to generate feasible trajectories without the need for a higher-level planning system. We conduct extensive experiments in the physical world, demonstrating agile and robust flight through complex and diverse previously-unseen race tracks.

My contributions include both the Kalman filter and the MPC, whereas the learning-based gate detection has been developed by my coauthors. The evaluation of all experiments has been a joint effort of all authors.

Related Video

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

2.4.2 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). URL: https://link.springer.com/article/10.1007/s11370-018-00271-6
and
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

Based on the experience from Paper F we further improved our approach to drone racing in this contribution describing our strategy for the *2019 AlphaPilot Challenge* world championship.

Contrary to the previous Paper F, which only detects the next gate, this approach makes use of any visible gate and takes advantage of multiple, simultaneous gate detections to compensate for drift of a VIO 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 racecourse even when the gates are not immediately visible and further enables to plan a near time-optimal path through the racecourse 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*.

However, the proposed system also highlighted two major problems of autonomous drone racing. First, the hardware platform was defined by the race organizer and brought some tremendous limitations in compute resources, control architecture, and mechanical system performance. The conclusions drawn from those limitations led to the development of a new drone platform led by me and presented in Paper B. Second, during the development of the planning algorithms for the *2019 AlphaPilot Challenge*, it became clear that there is a lack of time-optimal planning approaches for quadrotors and that unrealistic approximations dominated the state-of-the-art methods. This led to further investigation of time-optimal trajectory planning and resulted in my contributed Paper A.

My contributions to this work include the filter-based alignment of the VIO state estimate and the map, as well as the overall system architecture, evaluation, and participation in the *2019 AlphaPilot Challenge*.

Related Videos

(V13) https://youtu.be/DGjwm5PZQT8

(V14) https://youtu.be/ZIHjswKDods

3 Future Directions

Optimization-based techniques have proven themself to be valuable tools for research and development, as they allow to find solutions to complex and abstract problem formulations. The rapid groth of available compute resources together with efficient implementations allow for unforseen performance in estimation, planning, and control tasks, revolutionizing robotics. However, neither have we yet exploited their full potential, nor are there known solutions to all problems faced by the research community. These methods still rely on accurate system models and convex problem formulations. Furthermore, most navigation systems are developed as a sequence of modules that are executed consecutively, rather than conjointly. Additionally, these modules typically use their own specialized measurement and data representations, and largly rely on unimodal Gaussian noise assumptions.

These disjoint execution and separate data representations hinder many state-of-the-art approaches from exploiting systematic (perception-action) and algorithmic (estimation, planning, control) coupling. On the other hand, perception-action coupling can often be expressed by combining the measurement models and the system dynamics, while algorithmic could be achieved thanks to the similarity in structure of estimation, planning, and control problems. Already in the 1960s, this similarities became obvious from Kalman's publications [107, 106], but were never explored further than the linear-quadratic-gaussian control principle and separation theorem. Finally, convex problem formulations and unimodal noise distributions pose a significant challenge in task and environment modelling, e.g. when describing discrete decisions such as in obstacle avoidance or outlier rejection.

I envision the fundamental combination of estimation, planning, and control into a unified problem formulation as one of the most powerful concepts in the future of robotics. Additionally, I see a clear deficit in modelling techniques for uncertainty, dynamics, tasks, and the robot's environment. The following sections describe what I perceive to be the highest priority of future work.

3.1 Transient Estimation and Control

Both, estimation and control problems can be formulated as optimizations, and in the case of Gaussian noise assumptions often boil down to non-linear constrained least-squares problems. The similarities become obvious once we look at the mathematical representation of the two problems. MPC uses a discretized state space over a moving time horizon, optimizing a quadratic cost under system dynamics and actuation constraints. Furthermore, we can set the weigths for the quadratic cost such that they represent the relative tolerated standard deviation, simplifying their interpretability. Meanwhile, moving-horizon estimator (often also formulated as a pose-graph) also use a discretized state space, over which they find the maximum-a-posteriori probability of a set of measurements given the state space. Given Gaussian noise, this maximum-a-posteriori in the probability corresponds to the least-squares solution weighted by the inverse of the covariance.

Comparing the two approaches, we can see that both use discretized state spaces covering a moving horizon as optimization parameters, and quadratic costs weighted by expectance. The difference lies in the exact implementation of the residuals contributing to the cost function. In the case of estimation is measurement data from the past horizon together with its model, while in the case of control it represents the task formulation for the future horizon, where both intersect at the present state. Therefore, unifying these two similar structures could be as easy as concatenating them, which allows simultaneous processing and enables tight coupling between all modelled and estimated quantities. This enables direct incorporatioin of estimated variables such as states, but also parameters, environment representations, or landmarks, into the control problem, and tightly-coupled perception-action objectives, such as maximizing the information gain from future observations.

Additionally, the requirement to solve such optimization problems on resource-constrained hardware in realtime brought up interesting approaches which exploit the similarity between consecutive problem instances (temporal similiarity). Specifically, the real-world dynamics imply that any macroscopic system undergoes continuous smooth, but never arbitrarily fast state changes, resulting in consecutive optimization problems being similar and therefore close to the initial guess, i.e. the previous iteration solution. In the case of MPC, the so-called real-time iteration scheme [52, 253] exploits exactly this temporal similarity, allowing near-optimal solutions in real time with low latency.

However, this solving scheme has been investigated in [248], but has not yet been exploited by most estimation approaches, eventhough the same temporal-similarity assumptions apply. A unified problem formulation would also require efficient online solution of the resulting problem, possibly based on the real-time iteration scheme. It remains an open question of how fast or how frequently these iterations must be run to capture the target dynamic bandwidth given a certain set of possibly asynchronous sensors.

3.2 Uncertainty in Modelling, Environments, and Task Formulation

Since optimization-based approaches rely on models of the physical system, they are often limitted in performance by the accuracy of such models. While one could achieve maximal model fidelity by iteratively measuring, extending, and refining existing dynamic models, this is neither practical, nor economic, and unfortunately also ineffective. The reason for this is three-fold: i) real-world imperfections render even equal designs with slight difference, ii) certain tasks require interactions or changes of the robot dynamics that are difficult to measure or forsee (e.g. inertia of deliviered package), and iii) there are environmental changes that are practically impossible to model, e.g. weather, chaotic and turbulent aerodynamics, or human behavior. Therefore, accurate modelling of the nominal system dynamics is not sufficient, but realistic uncertainty description and prediction is required.

State-of-the-art approaches use unimodal Gaussian noise to describe uncertainty, since it is often a good approaximation of the real distribution, and allows for closed-form solutions for multivariate inference. As a unimodal distribution it also brings the computational advantage of convexity, rendering it well suited for iterative methods. This unfortunately also implies the incapability to describe non-convex and therefore multiodal distributions. However, exactly these distributions are often met in practical applications, where any discrete decision could imply a non-convex state space. A simple example is obstacle avoidance for mobile robots, where something as trivial as the question of "whether to go left or right of a tree" can pose an intractable issue.

This could be achieved by novel analytic uncertainty formulations similar to the Gaussian derivatives like the Kalman filter and LQR, systematic sampling approaches such as branch-and-bound, or random sampling such as employed in particle filters, random sample consensus (RANSAC), and rapidly-exploring random trees (RRT). I strongly believe however, that we need new methods to describe arbitrary multimodal uncertainty, which enables us to convexify or employ them in any other way directly into optimization problems. This would not only allow to predict multiple, possibly discrete, future system evolutions, but also regress multimodal hypotheses on measured data, with huge implications on state estimation under outliers, and also machine learning.

Interestingly, similar questions arise when dealing with task and mission formulations to command mobile robots, especially when those high-level references also need to be generated by computer programs. How can we discribe ordered or unordered sets of tasks for robots or even heterogenious fleets of robots, where these task sets can also include topology, incomplete task descriptions arising from previously unknown environments, multiple options for task completion, or collaborative tasks? To achieve this, we need methods to model partially unknown, multimodal, and universally interpretable objective models. This problem is nowadays typically circumvented with context-specific and

manually designed heuristics, which is not an option if we strive to achieve ubiquitous robotic solutions that integrate with a heterogenious infrastructure ecosystem. As of these reasons, the fundamental issue of how to communicate tasks between humans and robots, and inbetween robotic systems, remains an open question.

3.3 Machine Learning

Even though my thesis never directly included any machine learning techniques, I strongly believe that they have and will further develop their own unique advantages. The vast variety of machine learning appraoches and architectures, such as deep, convolutional, adversarial, and recurrent neural networks, shares both, similarities and differences with optimiztion-based approaches. They are similar in that they use numerical optimization to find local, global, or approximate extremas, yet are very different in i) how cost and gain (or loss and reward, respectively) are formulated, ii) the vast number of variables and residuals, and iii) how the optimization is performed. Neural networks are extremely powerful in that they allow to capture complex correlations, abstractions, or relations without the need to specifically formulate them. Essentially, they allow to capture (and sometimes compress) knowledge in a resource intensive, a priori computation (the *trainig*) and provide it in a way efficienct for querying (the *inferrence*). These networks can abstract high dimensional data into interpretable representations, such as images to labels, bounding boxes, and poses, or encode complex relations,

In many aspects neural networks can complement optimization-based methods: they allow discontinuities and non-convexity in the value function, can capture discrete decisions in high dimensional space, and compress complex patterns enabling fast inference. In my work on autonomous drone racing [110, 69] I have colaborated with my colleagues to exploit those complementary properties and shown their effectivity.

However, my work has only touched the surface of what's possible, and the true power of their complementary nature remains to be discovered. Infact, I believe that novel approaches to uncertainty modelling and processing could revolutionize robotics by enabling a cardinal conjunction of inference over data approximations and optimization over such.

A Time-Optimal Planning for Quadrotor Waypoint Flight

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

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. URL: https://robotics.sciencemag.org/content/6/56/eabh1221

Time-Optimal Planning for Quadrotor Waypoint Flight

Philipp Foehn, Angel Romero, Davide Scaramuzza

Abstract — Quadrotors are amongst the most agile flying robots. However, planning time-optimal trajectories at the actuation limit through multiple waypoints remains an open problem. This is crucial for applications such as inspection, delivery, search and rescue, and drone racing. Early works used polynomial trajectory formulations, which do not exploit the full actuator potential due to their inherent smoothness. Recent works resorted to numerical optimization, but require waypoints to be allocated as costs or constraints at specific discrete times. However, this time-allocation is a priori unknown and renders previous works incapable of producing truly time-optimal trajectories. To generate truly time-optimal trajectories, we propose a solution to the time allocation problem while exploiting the full quadrotor's actuator potential. We achieve this by introducing a formulation of progress along the trajectory, which enables the simultaneous optimization of the time-allocation and the trajectory itself. We compare our method against related approaches and validate it in real-world flights in one of the world's largest motion-capture systems, where we outperform human expert drone pilots in a drone-racing task.

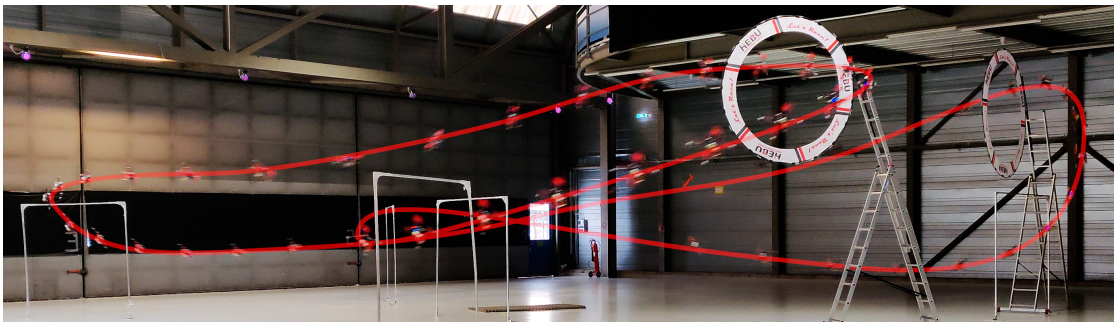


Figure A.1: A time-optimal trajectory. This time-optimal flight path was computed using the proposed complementary progress constraints (CPC) and executed in a motion capture system, outperforming the best human expert.

A.1 Introduction

Autonomous drones are nowadays used for inspection, delivery, cinematography, search-and-rescue, and entertainment such as drone racing [129]. The most prominent aerial system is the quadrotor, thanks to its simplicity and versatility, ranging from smooth maneuvers to extremely aggressive trajectories. This renders quadrotors amongst the most agile and maneuverable aerial robots[2, 242].

However, quadrotors have limited flight range, dictated by their battery capacity, which limits how much time can be spent on a specific task. If the task consists of visiting multiple waypoints (delivery, inspection, drone racing [153, 68, 132]), doing so in minimal time is often desired, and, in the context of search and rescue or drone racing (Fig. A.1), even the ultimate goal. In fact, expert human drone racing pilots accomplish this with astonishing performance, guiding their quadrotors through race tracks at speeds so far unreached by any autonomous system. This begs the question of how close human pilots fly to the theoretical limit of a quadrotor, and whether planning algorithms could find and execute such theoretical optima.

For simple point-mass systems, time-optimal trajectories can be computed in closed-form, resulting in bang-bang acceleration trajectories [121], which can be sampled over multiple waypoints [186]. However, quadrotors are underactuated systems that need to rotate to adjust their actuated acceleration direction, which always lies in the body z-axis [29, 141]. Both the linear and rotational acceleration are controlled through the rotor thrusts, which are physically limited by the actuators. This introduces a coupling in the achievable linear and rotational accelerations. Therefore, time-optimal planning becomes the search for the optimal tradeoff between maximizing these accelerations.

A.1.1 Related Work

Two common approaches for planning quadrotor trajectories exist, continuous-time polynomials and discrete-time state space representations. The first option is the widely used polynomial formulation [141, 149, 159] exploiting the quadrotor's differentially-flat output states with high computational efficiency. However, these polynomials are inherently smooth and therefore cannot represent rapid state or input changes (e.g. bang-bang [121]) at reasonable order, and only reach the input limits for infinitesimal short durations, or constantly for the full trajectory time. This renders polynomials suboptimal since they cannot exploit the full actuator potential. Both problems are visualized and further explained in Section A.5, in the supplementary material.

The second option includes all approaches using time-discretized trajectories which can be found using search and sampling-based methods [245, 4, 127, 258] or optimization-based methods [90, 85, 12, 183]. However, sampling the 4-dimensional continuous input space over many discrete time steps with sufficient resolution quickly becomes computationally intractable, which is why prior work [245, 4, 127, 258] restores to point-mass, polynomial, or differential-flatness approximations, and therefore does not handle single-rotor thrust constraints. Therefore, planning time-discretized trajectories with optimization-based methods is the only viable solution in the short-medium term. In such methods, the system dynamics and input boundaries are enforced as constraints. In contrast to the polynomial formulation, this allows the optimization to pick any input within bounds for each discrete time step. For a time-optimal solution, the trajectory time t_N is part of the optimization variables and is the sole term in the cost function. However, if multiple waypoints must be passed, these must be allocated as constraints to specific nodes on the trajectory. This time allocation is a priori undefined, since the time spent between any two waypoints is unknown, which renders traditional discretized state space formulations ineffective for time-optimal trajectory generation through multiple waypoints.

We investigate this problem and provide a solution that allows simultaneously optimizing the trajectory and waypoint allocation in a given sequence, exploiting the full actuator potential of a quadrotor. Our approach formulates a progress measure for each waypoint along the trajectory, indicating completion of a waypoint (see Fig. A.6). We then introduce a "Complementary Progress Constraint" (CPC), that allows completion only in proximity to a waypoint. Intuitively, proximity and progress must complement each other, enforcing completion of all waypoints without specifying their time allocation.

There already exists a number of works towards time-optimal quadrotor flight [94, 131, 222, 204], which, however, all suffer from severe limitations, such as limiting the collective thrust and bodyrates, rather than the actual constraint of limited single rotor thrusts.

The two earlier works [94, 131] are based on the aforementioned bang-bang approaches extended through numerical optimization of the switching times [94] and a trajectory

representation using a convex combination of multiple analytical path functions [131]. However, both are restricted to 2-dimensional maneuvers, whereas our approach generalizes to arbitrary 3D waypoint sequences.

Another approach is taken in [222], where a change of variables along an analytic reference path is used to put the vehicle state space into a traverse-dynamics formulation. This allows using the arc length along the reference path as a progress measure and enables the formulation of costs and constraints independent of the time variable. However, as in the previous works, they simplify the platform limits to collective thrust and bodyrates, neglecting realistic actuator saturation. Furthermore, due to the use of Euler angles, their orientation space only covers a subset of the feasible attitudes and limits the solutions to a, possibly sub-optimal, subspace.

Finally, [204] uses a completely different approach, where the segment times of a polynomial trajectory are refined based on learning a Gaussian Classification model predicting feasibility. The classification is trained on analytic models, simulation, and real flight data. While emphasizing real-world applicability, this approach is still constrained to polynomials and requires real-world data specifically collected for the given vehicle. Furthermore, it is an approximate method which only refines the execution speed of a predefined trajectory, rather than modifying the trajectory itself to a time-optimal solution, as opposed to our method.

A.1.2 Contribution

In contrast to existing methods, our approach resolves these problems by taking inspiration from optimization under contacts [187], proposing the formulation of *complementary progress constraints*, where we introduce a measure of progress and complement [42] it with waypoint proximity. More specifically, we formulate two factors that must complement each other, where, in our case, one factor is the completion of a waypoint (progress), while the other factor is the local proximity to a waypoint (Fig. A.6). Intuitively, a waypoint can only be marked as completed when the quadrotor is within a certain tolerance of the waypoint (Fig. A.7), allowing simultaneous optimization of the state and input trajectory, and the waypoint time allocation.

We demonstrate how our formulation can generate trajectories that are faster than human expert flights and evaluate it against two professional human drone racing pilots, outperforming them in terms of lap time and consistency on a 3D race track in a large-scale motion capture system (Fig. A.1). Since our proposed optimization problem is highly non-convex, we also provoke non-convexity effects in simulation experiments in the supplementary material (Section A.6).

Our method can not only serve as a baseline for time-optimal quadrotor flight but might also find applications in other fields, such as (multi-) target interception, orbital maneuvers,

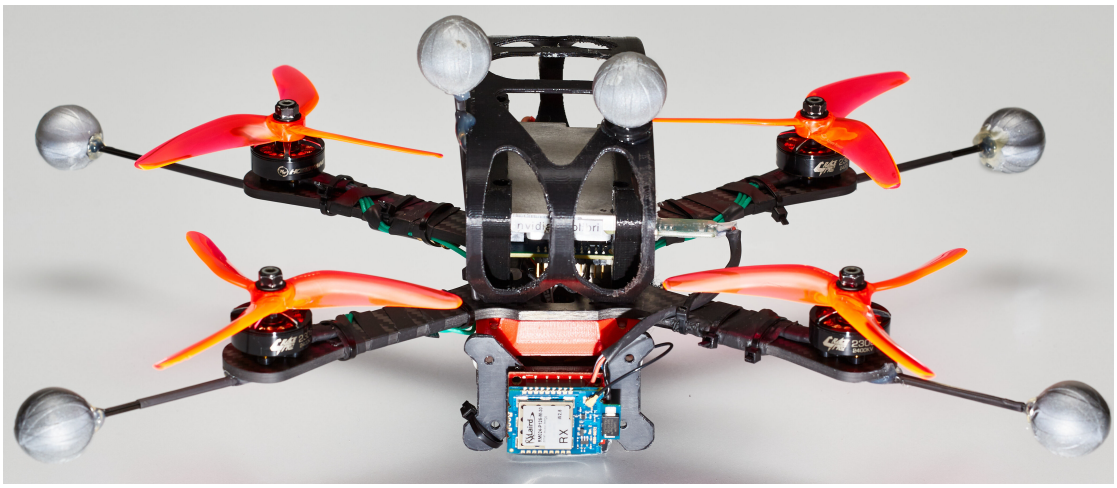


Figure A.2: The quadrotor vehicle. The autonomous platform used for the real-world experiments with a theoretical thrust-to-weight ration of ~ 4 at 0.8 kg weight, equipped with a Jetson TX2, a Laird communication module, off-the-shelf drone racing components, and infrared-reflective markers for motion capture.

avoiding mixed-integer formulations [196], and any problem where a sequence of task goals of unknown duration must be optimized under complex dynamic constraints.

A.2 Results

Video of the Results: https://youtu.be/ZPI8U1uSJUs

We chose drone racing as a demonstrator for our method because in racing the ultimate goal is to fully exploit the actuator potential to accomplish a task in minimal time. In our experiment, we set up a human baseline on a 3D race track with 7 gates (Fig. A.1 & A.5) in a motion capture environment with two professional expert drone racing pilots. We plan a time-optimal trajectory through the same race track and use an in-house developed drone platform and software stack to execute the trajectory in the same motion capture environment. We generate the trajectory at a slightly lower thrust bound than what the platform can deliver, to maintain controllability under disturbances as mentioned in Section A.5.4 and discussed in Section A.4.5. Our results show that we can outperform the humans and consistently beat their best lap time.

A.2.1 Experimental Drone Platform

The experiments are flown with an in-house developed drone platform based on off-the-shelf drone-racing components such as a carbon-fiber frame, BLDC motors, 5” propellers, and a BetaFlight flight controller. The quadrotor is equipped with an NVIDIA Jetson TX2 compute unit with WiFi and a Laird RM024 module for wireless low-latency

Table A.1: Quadrotor Configurations

Property	Race Quad	Airsim Quad	Standard Quad
m [kg]	0.8	1.0	1.0
l [m]	0.15	0.23	0.15
$diag(J)$ [gm^2]	$[1, 1, 1.7]$	$[10, 10, 20]$	$[5, 5, 10]$
$[T_{min}, T_{max}]$ [N]	$[0.0, 8.0]$	$[0.0, 4.179]$	$[0.25, 5.0]$
c_τ [1]	0.01	0.0133	0.01
ω_{max} [rad s^{-1}]	15	10	10
d_{drag} [s^{-1}]	0.4	0.6	–

communication. The static maximum thrust of a single rotor was measured using a load cell and verified in-flight, marking the platform's real maximum limit at a thrust to weight ratio of roughly ~ 4. Other specifications can be taken from Tab. A.1 under the *Race Quad* configuration.

For state estimation (pose, linear and angular velocities), we use a VICON system with 36 cameras. For control, we deploy a Model Predictive Controller, similar to [59] but based on the quadrotor dynamics from (A.15–A.17), with the state space $\mathbf{x} = [\mathbf{p}_{IB}, \mathbf{q}_{IB}, \mathbf{v}_{IB}, \boldsymbol{\omega}_B]^\top$ and input space $\mathbf{u} = [T_1, T_2, T_3, T_4]$. The MPC operates over a horizon of $N_{MPC} = 20$ time steps of $\delta t = 0.05\,\text{s}$, with a quadratic cost function, and also accounts for the single-rotor thrust constraints. The implementation is done using the ACADO[98] toolkit and qpOASES[65] as solver. We execute the MPC in a real-time iteration scheme [51] at a feedback rate of 100 Hz. The low-level BetaFlight controller has access to high-frequency IMU measurements, which allows precise tracking of bodyrate and collective thrust commands. These commands are extracted from the MPC controller and are guaranteed to stay within the platform capabilities due to the single-rotor thrust constraints.

A.2.2 Human Expert Pilot Baseline

Since human pilots so far outperformed autonomous vehicles, we establish a baseline by inviting two professional expert drone racing pilots, Michael Isler and Timothy Trowbridge, both of which compete in professional drone racing competitions. A list of their participation and rankings can be found in Tab. A.4 in the supplementary material. We created a 3D racing track in a VICON motion capture environment spanning roughly $25 \times 30 \times 8$m and let the humans train on this track for hours. We captured multiple races, each consisting of multiple laps, from which we evaluated the one with the overall best lap time, according to our timing strategy described in Section A.2.4. The quadrotor platform used for human flights has the same thrust-to-weight ratio as the autonomous platform. This provides a fair baseline since (I) both the humans and the autonomous drone have the same limitations, (II) the humans are given enough training time to adapt to the track, as is the case at a real drone racing event, and (III) we compare against the race including the absolute best lap time from all runs. Especially the latter point gives

a substantial advantage to the human competitors, since they are typically not capable of reproducing the absolute best lap time reliably, and would therefore fall behind in any multi-lap evaluation. The evaluated best human runs each contain 7 laps.

The human platform is also tracked in the VICON system and resembles the autonomous drone described in Section A.2.1, but drops the Jetson and Laird modules for a remote control receiver and a first-person-view camera system, with no weight difference. To keep a fair baseline, the human platform is restricted to the same maximum thrust-to-weight limit as the autonomous drone. The track, time-optimal reference (thrust-to-weight ratio 3.3), and the human expert 1 trajectory are visualized in Fig. A.5, with their speed and acceleration profile colorized.

A.2.3 Trajectory Generation

To generate the time-optimal trajectories that will be executed by our platform, we set the gate positions of the track shown in Fig. A.5 as waypoint constraints. We generate 2.5 laps to ensure that our experiments are not affected by start and end transient effects or large drops in battery voltage, and to ensure at least two full laps at maximum speed. For optimal results, the laps are generated by concatenating the waypoints for a single lap multiple times and solving the full multi-lap problem at once. Therefore, the optimization passes through 18 waypoints and can be solved on a normal desktop computer in $\sim 40\,\text{min}$. We used $N = 720$ nodes with a tolerance of $d_{tol} = 0.3\,\text{m}$.

Because a time-optimal trajectory exploits the full actuator potential, it is extremely aggressive from start to end. To ensure safe execution on a real drone, we linearly ramp up the thrust limit for the trajectory generation from hover to the full thrust limit, guaranteeing a smooth start of the trajectory. Since we plan over 2.5 laps and exclude the start and end from the timing, this has no notable impact on the reported timing results. The same start and end exclusion are done for the human timings (Section A.2.4).

Additionally, we plan the trajectory for a multitude of thrust-to-weight ratios reaching from 2.5 with a lap time of 7.14 s to a maximum of 3.6 resulting in a lap time of 5.81 s. As expected, the time shrinks with higher thrust-to-weight capabilities. We evaluate two configurations with thrust-to-weight ratios of 3.15 (6.27 s) and 3.3 (6.10 s) in our real-world experiments, reported in the following section. Both of these configurations are consistently faster than the human trajectory while staying within a safe margin of the quadrotors absolute limit (TWR: ~ 4), which allows for robust control even under disturbances, noise, and model imperfections.

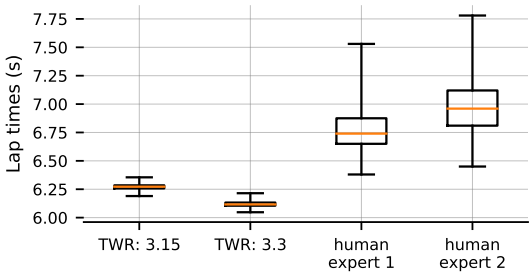


Figure A.3: Timing analysis. Lap time box plot of our time-optimal trajectory executed in real-world for two configurations (two laps), and two human races (seven laps). Note that even with a 5% reduced thrust-to-weight ratio of 3.15, the proposed time-optimal trajectories are faster than the best human lap time, with substantially lower variance.

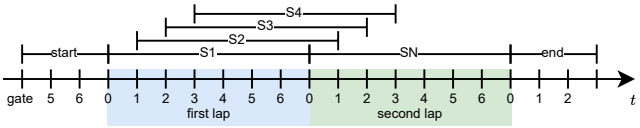


Figure A.4: Visualization of the timing strategy. A race includes two or more laps, over each of which we distribute 40 points at which we take timings of the resulting segments $S1, \ldots, SN$. This allows for statistically meaningful timing extraction.

A.2.4 Timing Analysis

First of all, our real-world experiments should provide a proof-of-concept that time-optimal trajectories planned below the drone's actual thrust limit are a feasible and viable solution. We refer the reader to the accompanying Movie at https://youtu.be/ZPI8U1uSJUs.

Second, we point out that it is possible to generate and execute trajectories that can outperform the human baseline. For this we provide a statistical lap-time analysis in Fig. A.3, indicating the superior performance on both configurations.

To compute reliable lap times, we first define a full lap as each lap that is not affected by any start (take-off) or end (landing) segment. We can then pick a set of timing points $\mathcal{S}_{pt} = \{\mathbf{p}_{t0}, \mathbf{p}_{t1}, \ldots\}$ along the trajectory, and define the timing as the time needed to visit one such point twice, i.e. the time of a segment starting and ending at the same point. This measure allows us to extract a statistically valid timing of a single closed lap, which does not depend on the location of timing start and stop.

Effectively we time the best laps of the human flights, and two full laps of both our race trajectories, each flown twice. Fig. A.3 shows the obtained timing results, where it is clearly visible that the autonomous drone outperforms the human pilots both in absolute time, but also consistency. The latter is expected since once the trajectory is generated, it can be repeated multiple times without variation.

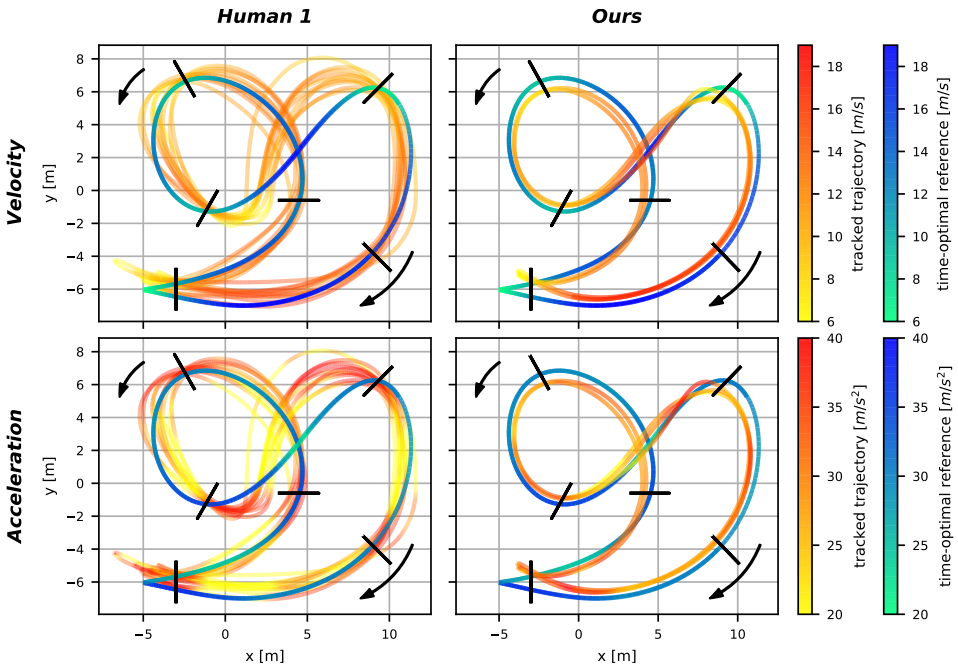


Figure A.5: Comparison against a human pilot. The race track with seven gates visualized with the yellow-red real-world trajectories of the humans (left, all seven laps of the race including the overall best lap time) and the autonomous drone (right, two laps visualized), and the green-blue time-optimal reference trajectory with a thrust-to-weight ratio (TWR) of 3.3. The trajectories are colored by their speed profile (top row) and acceleration profile (bottom row) to indicate the hotspots of highest velocity and acceleration along the track. The two black arrows indicate the direction of flight. The human pilots vary their acceleration substantially more than the autonomous drone and spend more time at sub-optimal acceleration (yellow coldspots on the lower left figure). Note that the gate in the lower-left corner consists of two gates stacked vertically.

Table A.2: Timing Statistics

Timing	human 1	human 2	ours TWR: 3.15	ours TWR: 3.3
mean [s]	6.794	6.987	3.15	6.120
median [s]	6.740	6.960	6.272	6.116
min [s]	6.389	6.450	6.190	6.048
max [s]	7.530	7.780	6.354	6.215
std [s]	0.2556	0.2859	0.0280	0.0278

A.3 Methodology

General Trajectory Optimization

The general optimization problem of finding the minimizer $\mathbf{x}^*$ for cost $L(\mathbf{x})$ in the state space $\mathbf{x} \in \mathbb{R}^n$ can be stated as

$$\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} L(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}) = 0 \quad \text{and} \quad \mathbf{h}(\mathbf{x}) \leq 0 \end{aligned} \tag{A.1}$$

where $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ contain all equality and inequality constraints respectively. The full state space $\mathbf{x}$ is used equivalently to the term *optimization variables*. The cost $L(\mathbf{x})$ typically contains one or multiple quadratic costs on the deviation from a reference, costs on the systems actuation inputs, or other costs describing any desired behaviors.

Direct Multiple Shooting Method

To represent a dynamic system in the state space we use a direct multiple-shooting method [26]. The system state $\mathbf{x}_k$ is described at discrete times $t_k = dt \cdot k$ at $k \in [0, N]$, also called nodes, where its actuation inputs between two nodes are $\mathbf{u}_k$ at t_k with $k \in [0, N)$. The systems evolution is defined by the dynamics $\dot{\mathbf{x}} = \mathbf{f}_{dyn}(\mathbf{x}, \mathbf{u})$, anchored at $\mathbf{x}_0 = \mathbf{x}_{init}$, and implemented as an equality constraint of the 4th-order Runge-Kutta integration scheme (*RK4*):

$$\mathbf{x}_{k+1} - \mathbf{x}_k - \mathbf{f}_{RK4}(\mathbf{x}_k, \mathbf{u}_k, dt) = 0 \tag{A.2}$$

which is part of $\mathbf{g}(\mathbf{x}) = 0$ in the general formulation. Both $\mathbf{x}_k$, $\mathbf{u}_k$ are part of the state space and can be summarized as the vehicle's dynamic states $\mathbf{x}_{dyn,k}$ at node k. Note that this renders the problem formulation non-convex for non-linear system dynamics.

Time-Optimal Trajectory Optimization

Optimizing for a time-optimal trajectory means that the only cost term is the overall trajectory time $L(\mathbf{x}) = t_N$. Therefore, t_N needs to be in the optimization variables $\mathbf{x} = [t_N, \dots]^\top$, and must be positive $t_N > 0$. The integration scheme can then be adapted to use $dt = t_N/N$.

A.3.1 Passing Waypoints through Optimization

To generate trajectories passing through a sequence of waypoints $\mathbf{p}_{wj}$ with $j \in [0, \dots, M]$, one would typically define a distance cost or constraint and allocate it to a specific state $\mathbf{x}_{dyn,k}$ at node k with time t_k. For cost-based formulations, quadratic distance costs are robust in terms of convergence and implemented as

$$L_{dist,j} = (\mathbf{p}_k - \mathbf{p}_{wj})^\top (\mathbf{p}_k - \mathbf{p}_{wj}) \tag{A.3}$$

where $\mathbf{p}_k$, part of $\mathbf{x}$, is the position state at a user defined time t_k. However, such a cost-based formulation is only a soft requirement and if summed with other cost terms does not imply that the waypoint is actually passed within a certain tolerance. To guarantee to pass within a tolerance, constraint-based formulations can be used, such as

$$(\mathbf{p}_k - \mathbf{p}_{wj})^\top (\mathbf{p}_k - \mathbf{p}_{wj}) \leq \tau_j^2 \tag{A.4}$$

which in the general problem is part of $\mathbf{h}(\mathbf{x}) \leq 0$, and requires the trajectory to pass by waypoint j at position $\mathbf{p}_{wj}$ within tolerance τ_j at time t_k.

A.3.2 Progress Measure Variables

To describe the progress throughout a track we want a measure that fulfills the following requirements: (I) it starts at a defined value, (II) it must reach a different value by the end of the trajectory, and (III) it can only change when a waypoint is passed within a certain tolerance. To achieve this, let the vector $\boldsymbol{\lambda}_k \in \mathbb{R}^M$ define the progress variables λ_k^j at timestep t_k for all M waypoints indexed by j. All progress variables start at 1 as in $\boldsymbol{\lambda}_0 = \mathbf{1}$ and must reach 0 at the end of the trajectory as in $\boldsymbol{\lambda}_N = \mathbf{0}$. The progress variables $\boldsymbol{\lambda}$ are chained together and their evolution is defined by

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k - \boldsymbol{\mu}_k \tag{A.5}$$

where the vector $\boldsymbol{\mu}_k \in \mathbb{R}^M$ indicates the progress change at every timestep. Note that the progress can only be positive, therefore $\mu_k^j \geq 0$. Both $\boldsymbol{\lambda}_k$ and $\boldsymbol{\mu}_k$ for every timestep are part of the optimization variables $\mathbf{x}$, which replicates the multiple shooting scheme for the progress variables. To define when and how the progress variables can change, we

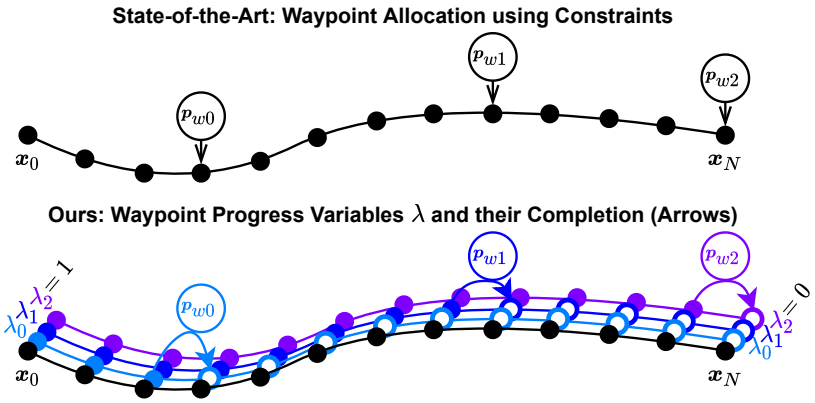


Figure A.6: Progress variables. Top: state-of-the-art fixed allocation of positional waypoints $\mathbf{p}_{wj}$ to specific nodes $\mathbf{x}_i$. Bottom: our method of defining one progress variable λ_j per waypoint. The progress variable can switch from 1 (incomplete) to 0 (completed) only when in the proximity of the relevant waypoint, implemented as a complementary constraint.

now imply a vector of constraints $\mathbf{f}_{prog}$ on $\boldsymbol{\mu}_k$, in its general form as

$$\boldsymbol{\epsilon}_k^- \leq \mathbf{f}_{prog}(\mathbf{x}_k, \boldsymbol{\mu}_k) \leq \boldsymbol{\epsilon}_k^+ \tag{A.6}$$

where $\boldsymbol{\epsilon}^-$, $\boldsymbol{\epsilon}^+$ can form equality or inequality constraints. Finally, to ensure that the waypoints are passed in the given sequence, we enforce subsequent progress variables to be bigger than their precursor at each timestep by

$$\lambda_k^j \leq \lambda_k^{j+1} \quad \forall \quad k \in [0, N], j \in [0, M). \tag{A.7}$$

Note that the last waypoint $\mathbf{p}_{wM}$ is always reached at the last node at t_N, and, therefore, could be implemented as a fixed positional constraint on $\mathbf{x}_N$, without loss of generality.

Complementary Progress Constraints

In the context of waypoint following, the goal is to allow $\boldsymbol{\mu}_k$ to only be non-zero at the time of passing a waypoint. Therefore, $\mathbf{f}_{prog}$ and $\boldsymbol{\epsilon}^- = \boldsymbol{\epsilon}^+ = 0$ are chosen to represent a *complementarity constraint* [187], as

$$\begin{aligned} f_{prog,j}(\mathbf{x}_k, \boldsymbol{\mu}_k) = \mu_k^j \cdot \|\mathbf{p}_k - \mathbf{p}_{wj}\|_2^2 := 0 \\ \forall j \in [0, M] \end{aligned} \tag{A.8}$$

which can be interpreted as a mathematical *NAND (not and)* function, since either μ_k^j or $\|\mathbf{p}_k - \mathbf{p}_{wj}\|$ must be 0. Intuitively, the two elements *complement* each other.

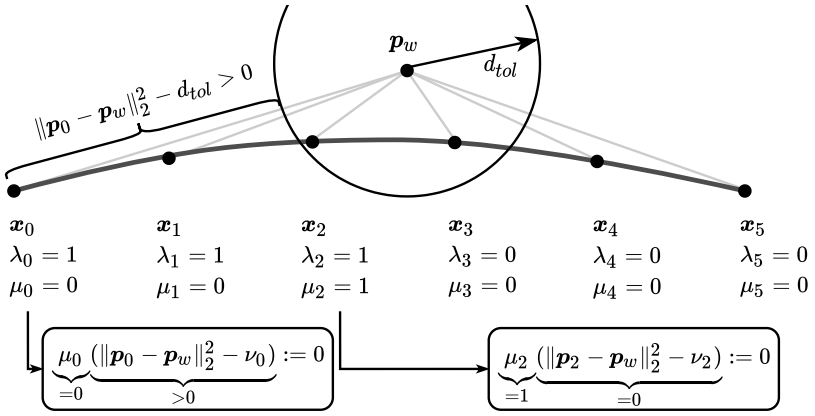


Figure A.7: Complementary progress constraint. The progress change μ can only be non-zero if the distance to the waypoint $\mathbf{p}_w$ is less than the tolerance d_{tol}. This is not the case for $\mathbf{x}_0$, but for $\mathbf{x}_1$, and allowing the progress variable to switch to 0 (complete).

A.3.3 Tolerance Relaxation

With (A.8) the trajectory is forced to pass *exactly* through a waypoint. Not only is this impractical, since often a certain tolerance is admitted or even wanted, but it also negatively impacts the convergence behavior and time-optimality, since the system dynamics are discretized and one of the discrete timesteps must coincide with the waypoint. Therefore, it is desirable to relax a waypoint constraint by a certain tolerance which is achieved by extending (A.8) to

$$\begin{aligned} & f_{prog,j}(\mathbf{x}_k, \boldsymbol{\mu}_k) = \mu_k^j \cdot \left(\|\mathbf{p}_k - \mathbf{p}_{wj}\|_2^2 - \nu_k^j \right) := 0 \\ & \text{subject to} \quad 0 \leq \nu_k^j \leq d_{tol}^2 \forall j \in [0, M] \end{aligned} \tag{A.9}$$

where ν_k^j is a slack variable to allow the distance to the waypoint to be relaxed to zero when it is smaller than d_{tol}, the maximum distance tolerance. This now enforces that the progress variables cannot change, except for the timesteps at which the system is within tolerance to the waypoint. Furthermore, please note that the spatial discretization δs depends on the number of nodes N and the speed profile. It should hold that $\delta s < d_{tol}$, to always allow at least one node to lie within the tolerance, as visualized in Fig. A.7. This can be verified after the optimization and approximated beforehand by $\delta s \approx D/N$, where D is the cumulative distance between all waypoints.

A.3.4 Optimization Problem Summary

The problem can be implemented using CasADi [6] and solved using IPOPT [244]. The full space of optimization variables $\mathbf{x}$ consists of the overall time and all variables assigned to nodes k as $\mathbf{x}_k$. All nodes k include the robot's dynamic state $\mathbf{x}_{dyn,k}$, its inputs $\mathbf{u}_k$, and

all progress variables, $\mathbf{x} = [t_N, \mathbf{x}_0, \ldots, \mathbf{x}_N]$ where

$$\mathbf{x}_k = \begin{cases} [\mathbf{x}_{dyn,k}, \mathbf{u}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \boldsymbol{\nu}_k] & \text{for } k \in [0, N) \\ [\mathbf{x}_{dyn,N}, \boldsymbol{\lambda}_N] & \text{for } k = N. \end{cases}$$

Based on this representation, we write the full problem as

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} t_N \tag{A.10}$$

subject to the system dynamics and initial constraint

$$\mathbf{x}_{k+1} - \mathbf{x}_k - dt \cdot \mathbf{f}_{RK4}(\mathbf{x}_k, \mathbf{u}_k) = 0 \qquad \mathbf{x}_0 = \mathbf{x}_{init},$$

the input constraints

$$\mathbf{u}_{min} - \mathbf{u}_k \leq 0 \qquad \mathbf{u}_k - \mathbf{u}_{max} \leq 0, \tag{A.11}$$

the progress evolution, boundary, and sequence constraints

$$\begin{aligned} &\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k + \boldsymbol{\mu}_k = \mathbf{0} \\ &\boldsymbol{\lambda}_0 - 1 = \mathbf{0} \qquad \boldsymbol{\lambda}_N = \mathbf{0} \\ &\boldsymbol{\mu}_k \geq 0 \quad \lambda_k^j - \lambda_k^{j+1} \leq 0 \quad \forall k \in [0, N), \quad j \in [0, M), \end{aligned} \tag{A.12}$$

and the complementary progress constraint with tolerance

$$\mu_k^j \cdot \left(\|\mathbf{p}_k - \mathbf{p}_{wj}\|_2^2 - \nu_k^j \right) = 0 \tag{A.13}$$

$$-\nu_k^j \leq 0 \qquad \nu_k^j - d_{tol}^2 \leq 0. \tag{A.14}$$

Note that constraints (A.13, A.14) are non-linear due to the norm on distance and the bilinearity of the progress change $\boldsymbol{\mu}$ and tolerance slack $\boldsymbol{\nu}$.

A.3.5 Quadrotor Dynamics

The quadrotor's state space is described between the inertial frame I and body frame B, as $\mathbf{x} = [\mathbf{p}_{IB}, \mathbf{q}_{IB}, \mathbf{v}_{IB}, \boldsymbol{\omega}_B]^\top$ corresponding to position $\mathbf{p}_{IB} \in \mathbb{R}^3$, unit quaternion rotation on the rotation group $\mathbf{q}_{IB} \in \mathbb{SO}(3)$ given $\|\mathbf{q}_{IB}\| = 1$, velocity $\mathbf{v}_{IB} \in \mathbb{R}^3$, and bodyrate $\boldsymbol{\omega}_B \in \mathbb{R}^3$. The input modality is on the level of collective thrust $\mathbf{T}_B = \begin{bmatrix} 0 & 0 & T_{Bz} \end{bmatrix}^\top$ and body torque $\boldsymbol{\tau}_B$. From here on we drop the frame indices since they are consistent

throughout the description. The dynamic equations are

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} & \dot{\mathbf{q}} &= \frac{1}{2}\mathbf{\Lambda}(\mathbf{q}) \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \\ \dot{\mathbf{v}} &= \mathbf{g} + \frac{1}{m}\mathbf{R}(\mathbf{q})\mathbf{T} & \dot{\boldsymbol{\omega}} &= \mathbf{J}^{-1}\left(\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}\right) \end{aligned} \tag{A.15}$$

where $\mathbf{\Lambda}$ represents a quaternion multiplication, $\mathbf{R}(\mathbf{q})$ the quaternion rotation, m the quadrotor's mass, and $\mathbf{J}$ its inertia.

Quadrotor Inputs

The input space given by $\mathbf{T}$ and $\boldsymbol{\tau}$ is further decomposed into the single rotor thrusts $\mathbf{u} = [T_1, T_2, T_3, T_4]$. where T_i is the thrust at rotor $i \in \{1, 2, 3, 4\}$.

$$\mathbf{T} = \begin{bmatrix} 0 \\ 0 \\ \sum T_i \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{\tau} = \begin{bmatrix} l/\sqrt{2}(T_1 + T_2 - T_3 - T_4) \\ l/\sqrt{2}(-T_1 + T_2 + T_3 - T_4) \\ c_\tau(T_1 - T_2 + T_3 - T_4) \end{bmatrix} \tag{A.16}$$

with the quadrotor's arm length l and the rotor's torque constant c_τ. The quadrotor's actuators limit the applicable thrust for each rotor, effectively constraining T_i as

$$0 \leq T_{min} \leq T_i \leq T_{max}. \tag{A.17}$$

In Fig. A.8 we visualize the acceleration space and the thrust torque space of a quadrotor in the xz-plane. Note that the acceleration space in Fig. A.8 is non-convex due to $T_{min} > 0$ for the depicted model parameters from the standard configuration of Tab. A.1. The torque space is visualized in Fig. A.8, where the coupling between the achievable thrust and torque is visible.

A.3.6 Approximative Linear Aerodynamic Drag

Finally, we extend the quadrotor's dynamics to include a linear drag model [58], to approximate the most dominant aerodynamic effects with diagonal matrix $\mathbf{D}$ by

$$\dot{\mathbf{v}} = \mathbf{g} + \frac{1}{m}\mathbf{R}(\mathbf{q})\mathbf{T} - \mathbf{R}(\mathbf{q})\mathbf{D}\mathbf{R}^\top(\mathbf{q}) \cdot \mathbf{v} \tag{A.18}$$

where we approximate $\mathbf{D} = \text{diag}(d_x, d_y, d_z)$ in this work.

A.4 Discussion

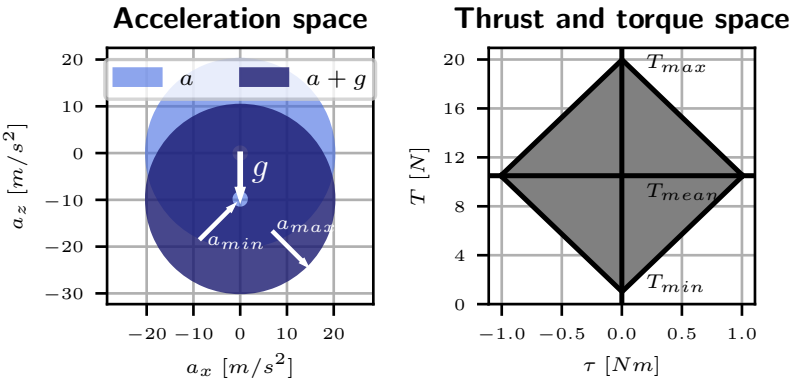


Figure A.8: Quadrotor input space. Acceleration (left) and thrust/torque-space (right) of a standard quadrotor configuration. Note that the acceleration space is non-convex due to the minimum acceleration (the idle thrust at minimum motor speed) $a_{min} > 0$ being non-zero, and the thrust and torque limits are dependent on each other.

A.4.1 Velocity and Acceleration Distribution

Furthermore, we evaluate the velocity and acceleration distribution over the different human and time-optimal flights. We depict the trajectories colored by their speed and acceleration profile in Fig. A.5 with the time-optimal reference with TWR: 3.3. Inspecting the hotspots of darker colors in the velocity plots (Fig. A.5, top row) indicating higher speeds, we can see that the velocity distribution is rather similar for the autonomous time-optimal and the human trajectory. However, comparing the acceleration of the human and the time-optimal trajectory (Fig. A.5, bottom row), we observe that the acceleration of the human varies considerably more, and often dips to lower values than the time-optimal ones. This corresponds to sections where the human pilots do not use the full actuation spectrum of the platform and lose substantial performance over the time-optimal trajectory. This is especially visible in the right-most section of the track, where the flight path has relatively low curvature. The time-optimal trajectory exploits the full acceleration capabilities, whereas the human notably reduces the acceleration between the right-most gates, leading to lower speeds in the following bottom section of the track.

Furthermore, we can identify the high-speed region in the sections of low curvature, where both human and autonomous platforms spend more acceleration in the velocity direction (accelerating and braking), than perpendicular to the velocity (direction change). Although the platforms have equal TWR, and our time-optimal trajectory is planned with a substantial margin to the platform's TWR limit, it exceeds the human speed profile.

A.4.2 Human Performance Comparison

From our findings in Section A.2.4 and A.4.1, we conclude that human pilots are not far away from our time-optimal trajectory, since they reach similar but slower velocity distribution and lap times. However, humans struggle to consistently exploit the full actuation spectrum of the vehicle, resulting in suboptimal performance compared to our approach. One possible reason for this can be found in [184], where we analyzed eye gaze fixations of human pilots during racing and found that they fixate their eyes on upcoming gates well before passing the next gate, indicating that humans use a receding planning horizon, whereas our approach optimizes the full trajectory at once.

A.4.3 Tracking Error Considerations

We want to point out that while we achieved successful deployment on a real quadrotor system, we experience substantial tracking errors in doing so, as visible in Fig. A.5. While this work is not about improving the tracking performance, but should rather serve as a feasibility study given our method, we still feel responsible for pointing out the encountered difficulties. A number of effects lead to this tracking error of $\sim 0.7\,\mathrm{m}$ positional RMSE:

(I) We did not account for any latency correction of the whole pipeline, including motion capture and pose filtering, data transmission to the drone, MPC execution time, and communication to the flight controller.

(II) Our simple linear-drag aerodynamic model was verified in [58] at speeds up to $5\,\mathrm{m\,s^{-1}}$. However, in the vastly higher speed regimes we reach during our real-world experiments, the model seems to be inaccurate, especially in terms of drag at higher speeds and in body z direction. The effect of this is visible in Fig. A.5, where there is a disturbance towards the inside of the curvature for most of the time.

(III) The used BetaFlight controller is a reliable system for human drone pilots. However, as such, it includes many filtering, feed-forward, and control strategies that are tuned for a consistent human *flight-feeling*. Unfortunately, this does not translate to accurate closed-loop control or desirable control-loop shaping and causes more harm than good when used with closed-loop high-level controllers such as our MPC. Not only is it not possible to command the single rotor thrusts from the MPC to BetaFlight, but the provided bodyrate tracking also performed poorly. This, however, could be solved with a custom flight controller and might be part of further studies.

A.4.4 Convexity and Optimality

While the problem of trajectory optimization quickly becomes non-convex when using complex and/or non-linear dynamic models, constraints, or even cost formulations (e.g. obstacle avoidance), it is often a valid approach to generate feasible (in terms of model dynamics) trajectories. In fact, given a non-convex problem, solution schemes such as the used interior point method can only guarantee local optimality and global feasibility, but not global optimality. Within our approach, we can summarize the following non-convex properties:

- The quadrotor dynamics (A.15) are non-linear and therefore non-convex in the context of (A.2).
- The acceleration space of a quadrotor (A.17) is non-convex given non-zero minimal thrust $T_{min} > 0.0$.
- Eqn. (A.13) is non-convex due to the norm on distance and the bilinearity of the progress change $\boldsymbol{\mu}$ and tolerance slack $\boldsymbol{\nu}$.

In our experiments in Sections A.6.5 and A.6.6, we provoke such non-convex properties, and explain how the optimization can be supported with an advanced initialization scheme to start close to the global optimum.

This can be achieved by reducing the non-linear quadrotor model into a linear point-mass model with bounded 3D acceleration input $\mathbf{u} = \mathbf{a}$ where $\|\mathbf{a}\| \leq a_{max}$. The linear model removes the prominent non-convex dynamics and allows us to find a solution from which the original problem with the quadrotor model can be initialized, both in terms of translational trajectory, and also with a non-continuous orientation guess based on the point-mass acceleration direction. While this initial guess is not yet dynamically feasible for quadrotors due to the absent rotational dynamics, it serves as a valid initial guess in the convex region of the global-optimal translation space. Finally, the rotational non-convexity can be resolved by solving multiple problems of different initializations, as demonstrated in Section A.6.5.

Last but not least, the reader should note that even in the case of a local (but not global) optimal solution, the vehicle dynamics are satisfied and the trajectory is dynamically feasible.

A.4.5 Real-World Deployment

There are three challenges when deploying our approach in real-world scenarios.

The first problem is posed by the nature of time-optimal trajectories themselves, as the true solution for a given platform is nearly always at the actuator constraints,

and leaves no control authority. This means that even the smallest disturbance could potentially have damaging consequences for the drone and render the remainder of the trajectory unreachable. One has to define a margin lowering the actuator constraints used for the trajectory generation to add control authority and therefore robustness against disturbances. However, this also leads to a slower solution, which is no longer the platform-specific time-optimal one. In the context of a competition, this effectively becomes a risk-management problem with interesting connections to game theory.

Second, our method is computationally demanding, ranging from a few minutes ($< 20\,\text{min}$) for scenarios as in Sections A.6.2 and A.6.5 towards an hour or more for larger scenarios such as Section A.2 with $\sim 40\,\text{min}$ and Section A.6.7 with $\sim 65\,\text{min}$ on a normal desktop computer. However, this is highly implementation-dependent and could be vastly broken down to usable times, or precomputed for static race tracks and other non-dynamic environments. Furthermore, our approach provides a method for finding the theoretical upper bound on performance, as a benchmark for other methods.

Third, we use a motion capture system to deploy our method. However, in real-world scenarios, such high-performance off-board localization systems are barely ever available. This necessitates the deployment using on-board state-estimation systems, such as visual-inertial odometry. Unfortunately, these systems can suffer from high motion blur in such fast flight scenarios, and therefore need substantial further research and development to be of sufficient robustness for the purpose of time-optimal flight [68, 153]. Despite those difficulties, we have demonstrated that our method *can* be deployed and *is* in fact substantially faster than human experts.

A.5 Preface: Time-Optimal Quadrotor Trajectory

A.5.1 Point-Mass Bang-Bang

Time-optimal trajectories encapsulate the best possible action to reach one or multiple targets in the lowest possible time. We first investigate a point mass in $\mathbb{R}^3$ controlled by bounded acceleration $\mathbf{a} \in \mathbb{R}^3 \mid \|\mathbf{a}\| \leq a_{max}$, starting at rest position $\mathbf{p}_0$ and translating to rest position $\mathbf{p}_1$. The time-optimal solution takes the form of a bang-bang trajectory over the time t_{opt}, accelerating with a_{max} for $t_{opt}/2$, followed by decelerating with a_{max} for $t_{opt}/2$. A trajectory through multiple waypoints can be generated similarly by optimizing or sampling over the switching times and intermittent waypoint velocities. For the general solution and applications, we refer to [186] and [121].

A.5.2 Bang-Bang Relation for Quadrotors

A quadrotor would exploit the same maximal acceleration by generating the maximum thrust. However, due to the quadrotor's underactuation, it cannot instantaneously change the acceleration direction but needs to rotate by applying differential thrust over its rotors. As a result, the linear and rotational acceleration, both controlled through the limited rotor thrusts, are coupled (visualized later in Fig. A.8). Therefore, a time-optimal trajectory is the *optimal tradeoff between rotational and linear acceleration*, where the thrusts only deviate from the maximum to adjust the rotational rates. Indeed, our experiments confirm exactly this behavior (see e.g. Fig. A.11).

A.5.3 Sub-Optimality of Polynomial Trajectories

The quadrotor is a differentially flat system [149] that can be described based on its four flat output states, position, and yaw. This allows representing the evolution of the flat output states as smooth differentiable polynomials of the time t. To generate such polynomials, one typically defines its boundary conditions at the start and end time, and minimizes for one of the derivatives, commonly the jerk or snap (3rd/4th derivative of position, as in [159, 148]). The intention behind such trajectories is to minimize and smooth the needed body torques and, therefore, single motor thrust differences, which are dependent on the snap. Since the polynomials are very efficient to compute (especially [159]), trajectories through many waypoints can be generated by concatenating segments of polynomials, and minimal-time solutions can be found by optimizing or sampling over the segment times and boundary conditions. However, these polynomials are smooth by definition, which stands in direct conflict with maximizing the acceleration at all times while simultaneously adapting the rotational rate, as explained in the previous Section A.5.2. In fact, due to the polynomial nature of the trajectories, the boundaries of the reachable input spaces can only be touched at one or multiple points, or constantly, but

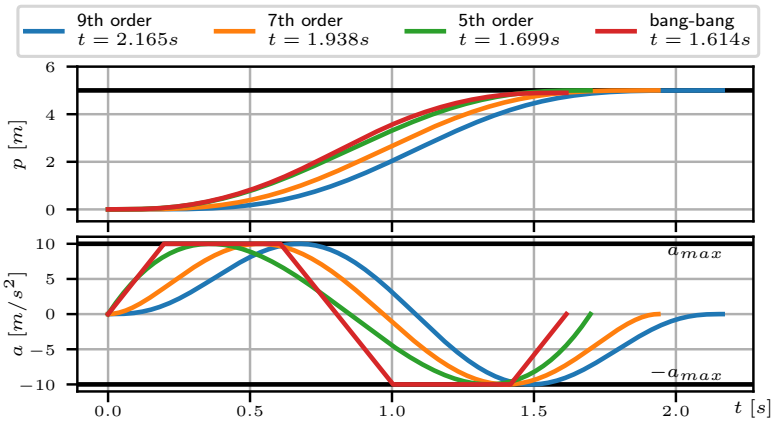


Figure A.9: Bang-bang and polynomial trajectories. Multiple orders of polynomial trajectories and one bang-bang trajectory with limited slope. The polynomial trajectories only touch the input extrema in two points, while the bang-bang spends more time at the limit and achieves a lower overall time.

not at subsegments of the trajectory, as visualized in Fig. A.9.

A.5.4 Real-World Restrictions

While the time-optimal solutions are at the boundary of the reachable input space and cannot be tracked robustly [121], they represent an upper bound on the performance for a given track, intended as a baseline for other algorithms. However, we can lower the input bounds (w.r.t the real quadrotor actuator limit) for the purpose of trajectory generation, allowing for control authority and therefore restoring a margin for robustness, rendering the planned trajectories trackable in the real world. As an application example, we present a demonstration of our method, where we track a time-optimal trajectory with a real quadrotor in a motion capture system and consistently outperform the human expert baseline.

A.6 Simulation Experiments

To demonstrate the capabilities and applicability of our method, we test it on a series of experiments. We first evaluate a simple point-to-point scenario and compare to [94, 131] in Section A.6.2. Next, we investigate the time alignment for multiple waypoints in Section A.6.3, followed by an experimental investigation of the convergence characteristics on short tracks in terms of initialization in Section A.6.4 and (non-) convexity in Section A.6.5. Finally, we demonstrate applicability to longer tracks with ≥ 10 waypoints (Section A.6.7). Note that this section will focus on the trajectory generation, while Section A.2 will demonstrate real-world deployment.

All evaluations are performed using CasADI [6] with IPOPT [244] as solver backend. We use multiple different quadrotor configurations, as listed in Tab. A.1. The first configuration represents a typical race quadrotor, the second one is parameterized after the MicroSoft AirSim [213] SimpleFlight quadrotor, and the third standard configuration resembles the one used in [94, 131].

A.6.1 Initialization Setup

If not stated differently, the optimization is initialized with identity orientation, zero bodyrates, $1\,\mathrm{m\,s^{-1}}$ velocity, linearly interpolated position between the waypoints, and hover thrusts. The total time is set as the distance through all waypoints divided by the velocity guess. The node of passing a waypoint (respectively where the progress variables $\boldsymbol{\lambda}$ switch to zero) is initialized as equally distributed, i.e. for waypoint j the passing node is $k_j = N \cdot j/M$. We define the total number of nodes N based on the number of nodes per waypoint N_w so that $N = MN_w$. We typically chose roughly $N_w \in (50, 100)$ nodes per waypoint, to get a good linearization depending on the overall length, complexity, and time of the trajectory. Note that high numbers of $N_w \gg 100$ help with convergence and achieved stability of the underlying optimization algorithm.

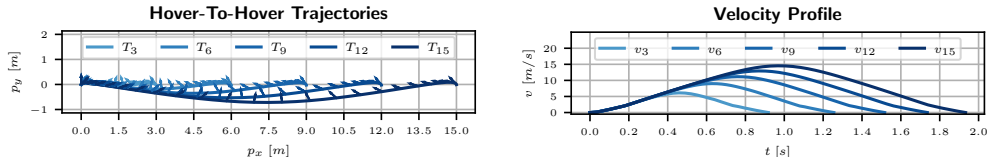


Figure A.10: Baseline comparison. Time-optimal hover-to-hover trajectories between states spaced $p_x = [3, 6, 9, 12, 15]$m apart as in [94, 131]. The top figure depicts the position on the xz-plane, while the bottom figure depicts the velocity profile.

A.6.2 Time-Optimal Hover-to-Hover Trajectories

We first evaluate trajectory generation between two known position states in hover, one at the origin, and one at $p_x = [3, 6, 9, 12, 15]$m, as in [94, 131]. Additionally to the problem setup explained in Section A.3, we add constraints to the end state to be in hover, i.e. $\mathbf{v}_N = \mathbf{0}$ and $\mathbf{q} = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$. We use $N = N_w = 300$ nodes and a tolerance of $d_{tol} = 10^{-3}$. Different from [94, 131], we compute the solution in full 3D space, which however does not matter for this experiment, since the optimal trajectory stays within the y-plane. We defined the model properties so that it meets the maximal and minimal acceleration $[a_{min}, a_{max}] = [1, 20]\,\mathrm{m\,s^{-2}}$ and maximal bodyrate $\omega_{max} = 10\,\mathrm{rad\,s^{-1}}$ as in [94, 131], given by our standard quadrotor configuration (see Tab. A.1).

The solutions are depicted in the xy-plane in Fig. A.10 and the timings are stated and compared to [94, 131] in Tab. A.3. In addition to our proposed single-rotor-thrust model A.15 (reported as *ours* in tab. A.3), we also compute the timings for a simplified model with only collective thrust and bodyrate constraints, but no single-rotor-thrust modeling, as used in [94, 131] (reported as *CPC-RT* in Tab/ A.3). Note that our approach with the simplified model produces results very similar to [94, 131], but with the increased model-fidelity, our approach is 2.00% slower than [94] and 2.68% slower than [131], due to the added rotational dynamics. However, [94, 131] does not allow to compute trajectories through multiple waypoints and [222] employs the same unrealistic bodyrate and thrust limits. In contrast, our method accounts for both, realistic single rotor thrust limits and multiple waypoints, while simultaneously solving the time-allocation, as evaluated in the next section.

p_x vs Time	Ref. [94]	Ref. [131]	CPC-RT	CPC (ours)
3 m	0.898 s	0.890 s	0.891 s	0.918 s
6 m	1.231 s	1.223 s	1.227 s	1.255 s
9 m	1.488 s	1.478 s	1.484 s	1.517 s
12 m	1.705 s	1.694 s	1.702 s	1.736 s
15 m	1.895 s	1.885 s	1.894 s	1.933 s

Table A.3: Comparison of the resulting timings between our approach and [94, 131]. Note that our approach is 2.00% slower than [94] and 2.68% slower than [131], because it accounts for rotation dynamics and single rotor limits. Our approach applied to the dynamics and limits of [94, 131], closely reproduces their results (column *CPC-RT*).

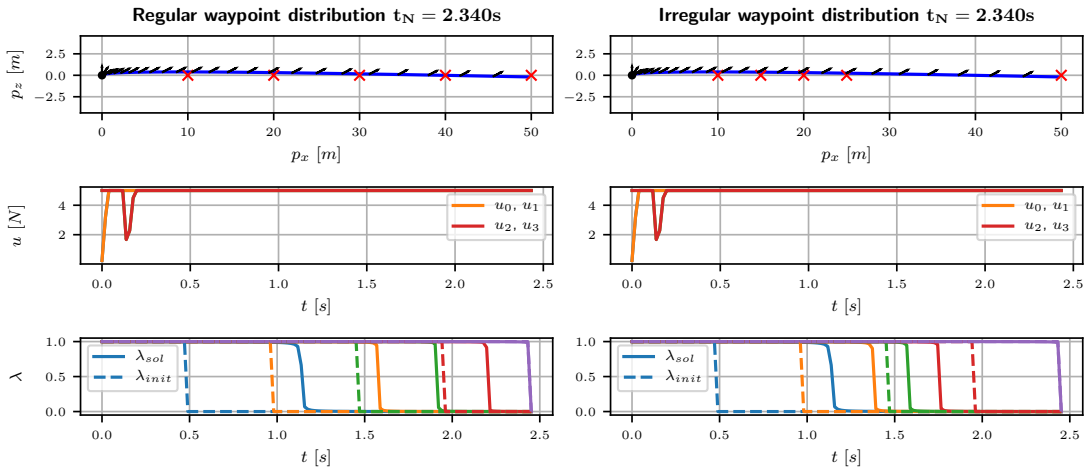


Figure A.11: Time allocation. A trajectory along waypoints distributed on a line over 50 m, flown in 2.430 s. On the left side, the waypoints are equally distributed over the total distance, while on the right side the first 4 out of 5 waypoints are located within the first half of the total distance. The bottom plot depicts the progress variables as initialized (dashed $--$) and as in the final solution (solid $-$). Note that both settings converge to the exact same solution, while the time of passing the waypoints was substantially adjusted from the initialization, and is different between the settings.

A.6.3 Optimal Time Allocation on Multiple Waypoints

In this experiment, we define a straight track between origin and $p_x = 50\,\text{m}$ trough multiple waypoints. The goal is to show how our method can choose the optimal time at which a waypoint is passed. Therefore, we test two different distributions of the waypoints $\mathbf{p}_{wj}$ over the straight track; specifically, we define a regular ($p_{x,reg} = [1, 20, 30, 40, 50]\text{m}$), and an irregular ($p_{x,ireg} = [10, 15, 20, 25, 50]\text{m}$) distribution. We chose $N = 125$ and a tolerance of $d_{tol} = 0.4\,\text{m}$, with the standard quadrotor (see Table A.1).

As expected, both waypoint distributions converge to the same solution of $t_N = 2.430\,\text{s}$, depicted in Fig. A.11, with equal state and input trajectories, despite the different waypoint distribution. Since the waypoints are located at different intervals, we can observe a different distribution of the progress variables in Fig. A.11, while the trajectory time, dynamic states, and inputs stay the same.

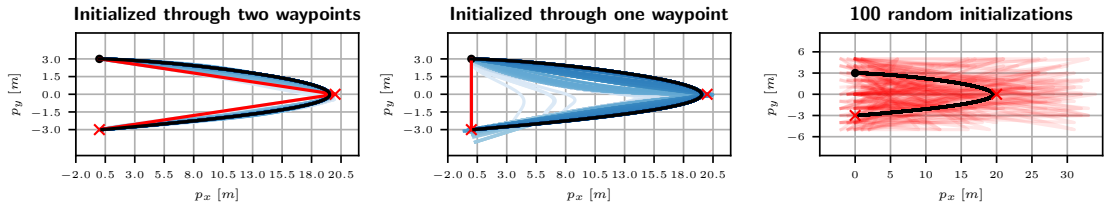


Figure A.12: Convergence quality. Convergence in an open hairpin turn from different initializations in red (—) over the iterations in light blue (—), to the final solution in dark blue(—). The trajectory starts at the top left and passes through the two waypoints (×). While the good initialization in the left figure needs 216 iterations, the poor guess in the middle figure needs 303 iterations, but both converge to exactly the same solution with $t_N = 3.617\,\text{s}$. The rightmost figure additionally shows 100 random initializations, all converging to the same solution.

A.6.4 Initialization & Convergence

As a next step, the method is tested for convergence properties given different initializations. For this, we again use the standard quadrotor configuration and discretize the problem into $N = 160$ nodes with a tolerance of $d_{tol} = 0.4\,\text{m}$. We use a track consisting of a so-called open hairpin, consisting of two waypoints as seen on the xy-plane in Fig. A.12, starting on the top left and passing the waypoint to the far right and bottom left, where the endpoint is not in hover. Multiple setups are tested, where the first one is initialized with the position interpolated between the waypoints (Fig. A.12, left), the second one is initialized with a poor guess interpolated only from start to endpoint (Fig. A.12, middle), and the third one includes 100 random initializations (Fig. A.12, right).

The expected outcome is that all initialization setups should converge to the same solution. Indeed we can observe this behavior in Fig. A.12, where we depict the initial position guess in red and the convergence from light to dark blue. The good initial guess in the leftmost Fig. A.12 results in 216 iterations until convergence, while the poor guess in the middle Fig. A.12 needs 303 iterations. In the rightmost Fig. A.12 we perform 100 uniform random initializations on a 6×6m xy-plane around the start and end point, and a 30×12meter xy-plane around the mid point. All initializations converge to the same solution. Overall, this indicates that our method is not sensitive to initialization variations in the translation space, but profits from good guesses. However, since the acceleration and orientation space of a quadrotor is not convex, the next two experiments elaborate on how to provoke and circumvent possible non-convexity issues in the rotation dynamics.

A.6.5 Provoking Non-Convexity Issues

Since quadrotors can only produce thrust in body z-axis and the motors of most real-world systems cannot turn off, and therefore always produce a positive minimal thrust $T_{min} > 0$, the resulting acceleration space is non-convex. We evaluate this on a vertical turn where we fly from hover at the origin through two waypoints directly above each other, back to the origin but not in hover. The Track can be seen in Fig. A.13.

First, the standard quadrotor configuration is used, with $N = 150$ nodes and a tolerance of $d_{tol} = 0.1\,\text{m}$, with the general initialization setup where the orientation is kept at identity. A second setup uses a different initialization, where we interpolate the orientation around the y-axis between $\alpha_{init} = 0$, $\alpha_0 = \pi$ for the second waypoint and $\alpha_1 = \alpha_2 = 2\pi$ for the remaining waypoints. The solutions and associated initializations are depicted in the top Fig. A.13, in which it is obvious that they do not converge to the same solution. The second setup actually performs a flip which is slightly faster at 4.115 s compared to the first setup at 4.171 s (1.3% faster). This is expected due to the non-convex properties of the problem.

However, a second set of experiments is performed with the same initialization setups but the race quadrotor configuration. This configuration has a minimum thrust of $T_{min} = 0\,\text{N}$, which renders the achievable acceleration space convex. Note that the full problem formulation still is non-convex, due to the non-linear dynamics and constraints. Both setups for the race configuration are depicted in the bottom Fig. A.13, and indeed, both initializations now converge to the same solution, which overlay each other and achieve the same timing at 2.885 s.

A simple solution would be to first solve the same problem using a linear and therefore convex point-mass model and using this to initialize the problem with the full quadrotor model. We further elaborate on the convexity property in the discussion Section A.4.4.

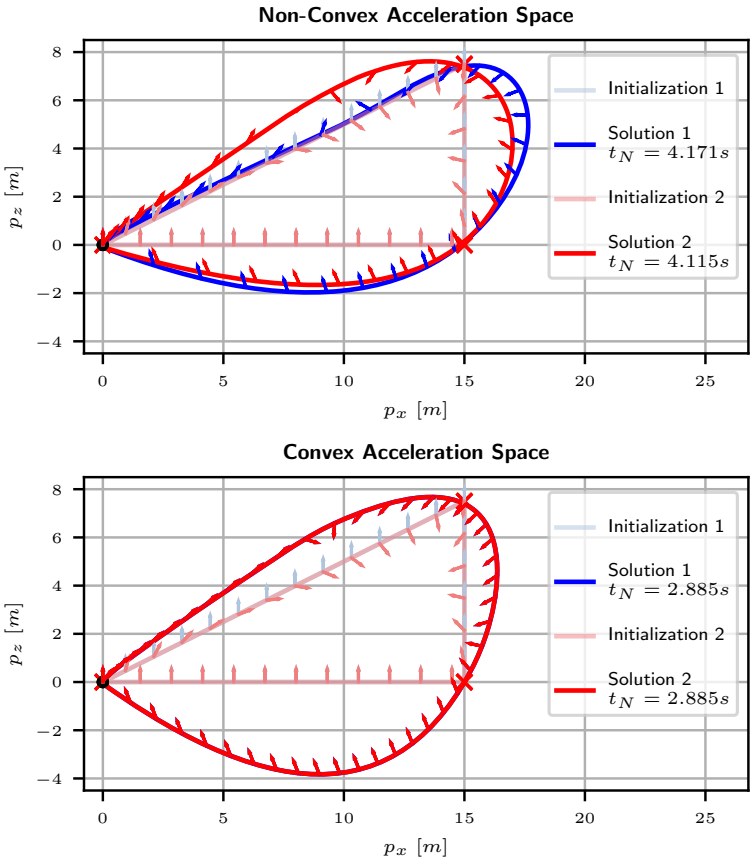


Figure A.13: Influence of convexity. A vertical turn flown starting at the origin in hover and passing through both waypoints from top to bottom, and back to the origin. Two quadrotor configurations are used (standard in the top figure and race configuration in the bottom figure), with two different initialization setups each. The first setup is as described in Section A.6 with identity orientation, while the second setup uses a linearly interpolated orientation guess. The arrows indicate the thrust direction of the quadrotor. The standard quadrotor configuration converges to two different solutions in the top figure depending on the initialization due to its non-convex acceleration space with $T_{min} > 0\,\mathrm{N}$, while the race quadrotor configuration converges to equal (and overlaying) solutions in the bottom figure, due to its convex acceleration space with $T_{min} = 0\,\mathrm{N}$.

A.6.6 Local-vs-Global Optimum

We follow up the previous non-convex example with another edge case where the non-convex dynamics can be used to provoke a local optimum, and we show how to resolve this special case. The problem addressed is the pure-vertical translation from hover to hover, e.g. descending from an initial hover point at 5 m height, to the origin at 0 m, parameterized with $N = 100$ nodes, a tolerance of $d = 0.1$ m, and using the race quad configuration.

When initialized with upright orientation and linear translation, the solution converges to a free-fall trajectory, which is a feasible local optimal solution, depicted in Fig. A.14. However, the optimal solution would exploit the available inputs in addition to the gravity acceleration, effectively performing a flip to accelerate downwards, before turning upright to decelerate into a hover state. We can resolve this problem of local optimality by initializing the problem from a bang-bang point-mass guess. This initial bang-bang guess exploits the full actuation space to accelerate for $t_{acc} = t_N/2$ towards the end state, followed by symmetric deceleration phase of equal $t_{dec} = t_N/2$, under the assumption of no gravity. Essentially, this provides us with an initial guess for our orientation, which resolves the local optimum and converges to the global solution of performing a flip, visible in Fig. A.14. This initialization scheme can be applied for arbitrary trajectories to resolve most local optima.

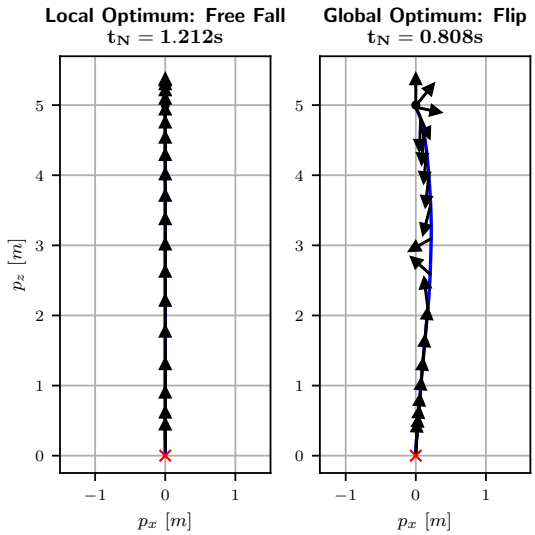


Figure A.14: Local and global optimum. Two solutions to travel from a starting point at 5 m height to the origin at 0 m. The left solution is initialized from an upright linear interpolated guess converging to a locally optimal free-fall descent. In contrast, the right solution is initialized from a bang-bang acceleration guess, reaching the global optimal solution of performing a flip. Note that the global optimal flip is notably faster (0.808 s), than the locally optimal free fall (1.212 s).

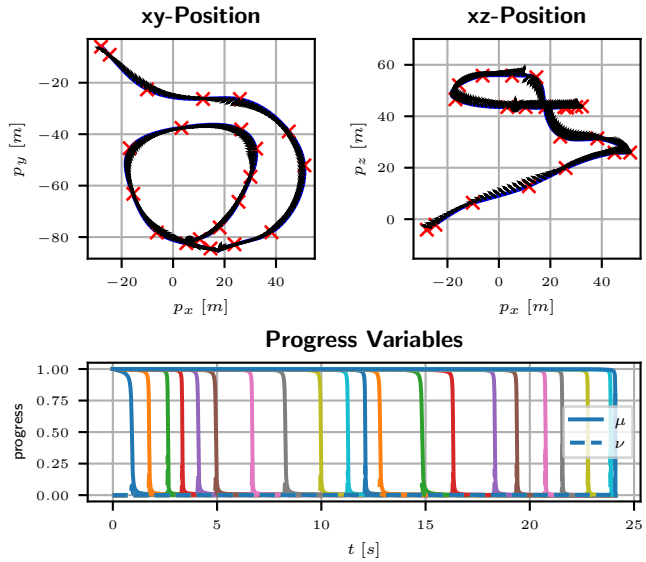


Figure A.15: Airsim qualification. The NeurIPS Airsim Qualification 1 track, covered in $t_N = 24.11\,\mathrm{s}$, as opposed to the best team's 30.11 s. The top row depicts the trajectory in xy- and xy-plane, while the second row depicts the velocity, respectively, plotted over time.

A.6.7 Microsoft AirSim, NeurIPS 2019 Qualification 1

As an additional demonstration, we apply our algorithm on a track from the 2019 NeurIPS AirSim Drone racing Challenge [139], specifically on the Qualifier Tier 1 setup. We choose a quadrotor with roughly the same properties, described as the MS configuration in Tab. A.1. The track is set up with the initial pose and 21 waypoints as defined in the environment provided in [139]. We use a discretization of $N = 3360$ nodes and a tolerance of $d_{tol} = 0.1\,\mathrm{m}$. The optimization of such a large state space took $\sim 65\,\mathrm{min}$ on a normal desktop computer.

The original work by [139] provides a simple and conservative baseline performance of $t_{total} \approx 110\,\mathrm{s}$ under maximal velocity and acceleration of $v_{max} = 30\,\mathrm{m\,s^{-1}}$ and $a_{max} = 15\,\mathrm{m/s^2}$, respectively. However, the best team achieved a time of $t_{total} = 30.11\,\mathrm{s}$ according to the evaluation page from [139]. Our method generates a trajectory that passes all waypoints at a mere $t_N = 24.11\,\mathrm{s}$, visualized in Fig. A.15. Please note that this trajectory should only serve as a theoretical lower bound on the possibly achievable time given the model parameters.

Table A.4: Human Pilot Rankings

Michael Isler

Year	Event	Ranking
2020	Swiss Drone League Race Luzern	2nd
	Highest Drone Race in the World - St. Moritz	1st
2019	Drone Champions League Laax	3rd
	Swiss National Team FAI World Championship Shanghai	team
2018	Southern Germany Race	3rd
	Drag Race Southern Germany	3rd
	Swiss Online Freestyle Challenge	2nd
	Kamikaze Race	1st
	FAI Lausanne	2nd
	Swiss Drone League Race St. Gallen	1st
	Drone Champions League Rapperswil	3rd
	Swiss National Team FAI World Championship Shenzen	team
	Drag Race at FAI World Championship Shenzen	3rd
2017	Swiss Nationals	2nd
	Swiss Indoor Masters	2nd
	Drone Night	1st

Timothy Trowbridge

Year	Event	Ranking
2020	Drone Racing League	par
2019	Drone Champions League Season	2nd
	Drone Champions League Romania	1st
	Drone Champions League Vaduz	2nd
	Swiss Drone League Bern	1st
	Swiss Drone League Luzern	2nd
	Swiss Drone League Basel	1st
	Drone Champions League Turin	1st
2018	Drag Race at FAI World Championship Shenzen	1st
	Drone Champions League Season	1st
	Drone Champions League Zurich	1st
	Drone Champions League Brussels	1st
	Drone Champions League China	1st
	XFly Great Wall of China	1st
	Inter Copter Racing Cup	2nd
	Thüringen Saison Opening	2nd
2017	Moto Drone St. Gallen	2nd
	Drone Champions League Brussels	2nd
	Morph X Masters	2nd
	Athens Drone GP	1st
	Drone Champions League Paris	2nd
	UpGreat Drone Race	1st
2016	Danish Drone Nationals	2nd
	Drone Night	1st

Entries where the participants competed as part of a team are marked as "team", while participation with undisclosed results are marked as "par".

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). URL: https://link.springer.com/article/10.1007/s11370-018-00271-6

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, nonlinear 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*.

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*[1,2], 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 [154, 153] 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) [34]. There exists a variety of VIO algorithms, e.g., [157, 25, 189, 76], that are based on feature detection and tracking that achieve very good results in general navigation tasks [48]. 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 [47].

[1] https://thedroneracingleague.com/airr/
[2] 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.

To overcome this difficulty, several approaches exploiting the structure of drone racing with gates as landmarks have been developed, e.g., [124, 104, 110], where the drone locates itself relative to gates. In [124], a handcrafted process is used to extract gate information from images that is then fused with attitude estimates from an inertial measurement unit (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 [104], 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 [110] 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 ap-

proaches 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 [149, 160], where trajectories are described as polynomials in time. Other approaches additionally incorporate obstacle avoidance [258, 84] or perception constraints [59, 221]. 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 [94, 131, 204, 71]. However, while [94, 131] are limited to 2D scenarios and only find trajectories between two given states, [204] requires simulation and real-world data obtained on the track, and the method of [71] is not applicable due to computational constraints.

C.1.3 Contribution

The approach contributed herein builds upon the work of [110] 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 [124, 104, 110], 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., [121], 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 [68] 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 [88], 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 ±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 ±30° 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 ${}_{\mathcal{I}}\boldsymbol{p}_{CB} = {}_{\mathcal{I}}\boldsymbol{p}_B - {}_{\mathcal{I}}\boldsymbol{p}_C$ 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

Table C.1: Sensor specifications.

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

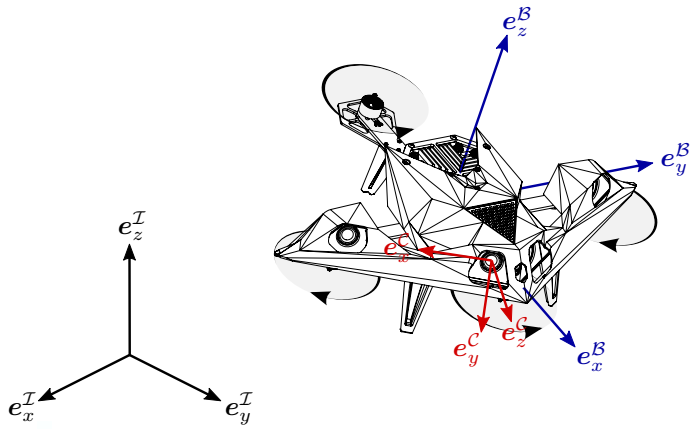


Figure C.2: Illustration of the race drone with its body-fixed coordinate frame $\mathcal{B}$ in blue and a camera coordinate frame $\mathcal{C}$ in red.

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 [105]. The translational degrees-of-freedom are described by the position of its center of mass, $\boldsymbol{p}_B = (p_{B,x}, p_{B,y}, p_{B,z})$, 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, $\boldsymbol{q}_{\mathcal{IB}}$, where $\boldsymbol{R}_{\mathcal{IB}} = \boldsymbol{R}(\boldsymbol{q}_{\mathcal{IB}})$ denotes the rotation matrix mapping a vector from the body-fixed coordinate frame $\mathcal{B}$ to the inertial frame $\mathcal{I}$ [216]. A unit quaternion, $\boldsymbol{q}$, consists of a scalar q_w and a vector $\tilde{\boldsymbol{q}} = (q_x, q_y, q_z)$ and is defined as $\boldsymbol{q} = (q_w, \tilde{\boldsymbol{q}})$ [216]. The drone's equations of motion are

$$m\ddot{\boldsymbol{p}}_B = \boldsymbol{R}_{\mathcal{IB}} f \boldsymbol{e}_z^{\mathcal{B}} - \boldsymbol{R}_{\mathcal{IB}} \boldsymbol{D} \boldsymbol{R}_{\mathcal{IB}}^{\mathsf{T}} \boldsymbol{v}_B - m\boldsymbol{g}, \tag{C.1}$$

$$\dot{\boldsymbol{q}}_{\mathcal{IB}} = \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \otimes \boldsymbol{q}_{\mathcal{IB}}, \tag{C.2}$$

where f and $\boldsymbol{\omega}$ are the force and bodyrate inputs, $\boldsymbol{e}_z^{\mathcal{B}} = (0, 0, 1)$ is the drone's z-axis expressed in its body-fixed frame $\mathcal{B}$, $\boldsymbol{D} = \operatorname{diag}(d_x, d_y, 0)$ is a constant diagonal matrix containing the rotor drag coefficients, $\boldsymbol{v}_B = \dot{\boldsymbol{p}}_B$ denotes the drone's velocity, $\boldsymbol{g}$ is gravity and $\otimes$ denotes the quaternion multiplication operator [216]. The drag coefficients were identified experimentally to be $d_x = 0.5\,\mathrm{kg/s}$ and $d_y = 0.25\,\mathrm{kg/s}$.

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.

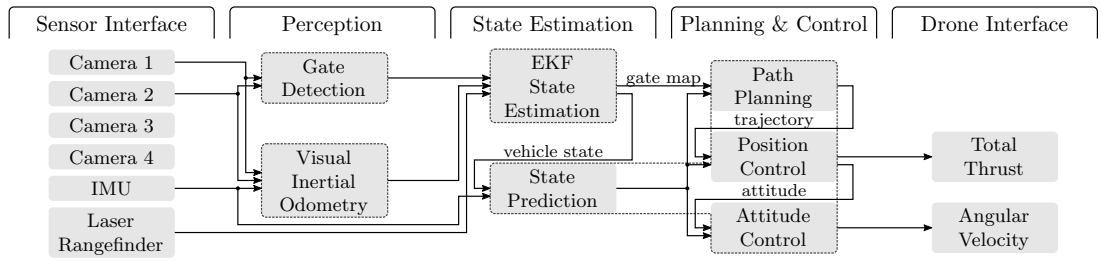


Figure C.3: Overview of the system architecture and its main components. All components within a dotted area run in a single thread.

C.3.1 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 [25] 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.

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 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 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 [35]. These are vector fields, which, in our case, are defined along the edges of the gates, and point from

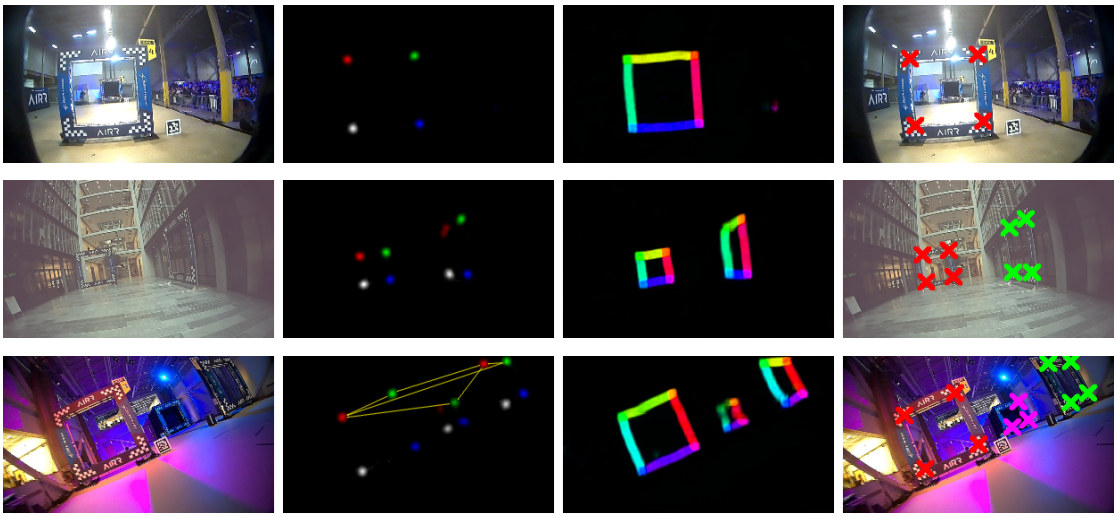


Figure C.4: The gate detection module returns sets of corner points for each gate in the input image (fourth column) using a two-stage process. In the first stage, a neural network transforms an input image, $\boldsymbol{I}_{w\times h\times 3}$ (first column), into a set of confidence maps for corners, $\boldsymbol{C}_{w\times h\times 4}$ (second column), and Part Affinity Fields (PAFs) [35], $\boldsymbol{E}_{w\times h\times(4\cdot 2)}$ (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, $\boldsymbol{C}$ (second column), and PAFs, $\boldsymbol{E}$ (third column), are displayed in a single image each. While color encodes the corner class for $\boldsymbol{C}$, it encodes the direction of the 2D vector fields for $\boldsymbol{E}$. 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.

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, $\boldsymbol{I}_{w\times h\times 3}$, is mapped by a neural network into a set of $N_C = 4$ corner maps, $\boldsymbol{C}_{w\times h\times N_C}$, and $N_E = 4$ PAFs, $\boldsymbol{E}_{w\times h\times(N_E\cdot 2)}$. 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, $\boldsymbol{C}^*_j(\boldsymbol{s})$, 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 $\boldsymbol{s}$ in the image. The value at location

$\boldsymbol{s} \in \boldsymbol{I}$ in $\boldsymbol{C}_j^*$ is defined by a Gaussian as

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

where $\boldsymbol{s}_j^*$ denotes the ground truth image position of the nearest corner with class j. The choice of the parameter σ 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, $\boldsymbol{E}_{(k,l)}^*(\boldsymbol{s})$, 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 $\boldsymbol{s}$ 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 $\boldsymbol{S}_{(k,l),g}$ be the set of image points that are within distance d of the line connecting the corner points $\boldsymbol{s}_k^*$ and $\boldsymbol{s}_l^*$ belonging to gate g. Furthermore, let $\boldsymbol{v}_{k,l,g}$ be the unit vector pointing from $\boldsymbol{s}_k^*$ to $\boldsymbol{s}_l^*$ of the same gate. Then, the part affinity field, $\boldsymbol{E}_{(k,l)}^*(\boldsymbol{s})$, is defined as:

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

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 $\boldsymbol{s}$ lies in $\boldsymbol{S}_{(k,l),g}$ of several gates, the $\boldsymbol{v}_{k,l,g}$ of all corresponding gates are averaged.

C.4.2 Stage 2: Corner Association

At test time, discrete corner candidates, $\boldsymbol{s}_j$, 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, $\{(\boldsymbol{s}_k, \boldsymbol{s}_l)\}$, see the yellow lines in Fig. C.4. Given the corresponding PAF, $\boldsymbol{E}_{(k,l)}(\boldsymbol{s})$, 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

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

where $\boldsymbol{s}(u)$ lineraly interpolates between the two corner candidate locations $\boldsymbol{s}_k$ and $\boldsymbol{s}_l$. 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 [35], finding this optimal assignment corresponds to a K-dimensional matching problem that is known to be NP-Hard [246]. Following [35], 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):

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

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

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

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 [118], 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[3], 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

[3] https://github.com/wkentaro/labelme

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 [198] 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 592×352 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 [106]. 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 $\boldsymbol{p}_V$ and $\boldsymbol{q}_{\mathcal{IV}}$, jointly with the gate positions, $\boldsymbol{p}_{G_i}$, and gate heading, $\varphi_{\mathcal{IG}_i}$. It can thus correct for an imprecise initial position estimate, VIO drift, and uncertainty in gate positions. The EKF's state space at time t_k is $\boldsymbol{x}_k = \boldsymbol{x}(t_k)$ with covariance $\boldsymbol{P}_k$ described by

$$\boldsymbol{x}_k = \left(\boldsymbol{p}_{\mathcal{V}}, \boldsymbol{q}_{\mathcal{IV}}, \boldsymbol{p}_{G_0}, \varphi_{\mathcal{IG}_0}, \ldots, \boldsymbol{p}_{G_{N-1}}, \varphi_{\mathcal{IG}_{N-1}}\right). \tag{C.9}$$

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

$$\boldsymbol{p}_{\mathcal{B}} = \boldsymbol{p}_{\mathcal{V}} + \boldsymbol{R}_{\mathcal{IV}} \cdot \boldsymbol{p}_{\mathcal{VB}}, \qquad \boldsymbol{q}_{\mathcal{IB}} = \boldsymbol{q}_{\mathcal{IV}} \cdot \boldsymbol{q}_{\mathcal{VB}}. \tag{C.10}$$

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:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k, \qquad \boldsymbol{P}_{k+1} = \boldsymbol{P}_k + \Delta t_k \boldsymbol{Q}, \tag{C.11}$$

where $\boldsymbol{Q}$ is the random walk process noise. For each measurement $\boldsymbol{z}_k$ with noise $\boldsymbol{R}$ the predicted *a priori* estimate, $\boldsymbol{x}_k^-$, is corrected with measurement function, $\boldsymbol{h}(\boldsymbol{x}_k^-)$, and

Kalman gain, $\boldsymbol{K}_k$, resulting in the *a posteriori* estimate, $\boldsymbol{x}_k^+$, as

$$\begin{aligned}\boldsymbol{K}_k &= \boldsymbol{P}_k^- \boldsymbol{H}_k^\intercal \left(\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^\intercal + \boldsymbol{R}\right)^{-1}, \\ \boldsymbol{x}_k^+ &= \boldsymbol{x}_k^- + \boldsymbol{K}_k \left(\boldsymbol{z}_k - \boldsymbol{h}(\boldsymbol{x}_k^-)\right), \\ \boldsymbol{P}_k^+ &= \left(\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k\right) \boldsymbol{P}_k^-,\end{aligned} \tag{C.12}$$

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

However, the filter state includes a rotation quaternion constrained to unit norm, $\|\boldsymbol{q}_{\mathcal{IV}}\| \stackrel{!}{=} 1$. 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 [74]. The quaternion $\boldsymbol{q}_{\mathcal{IV}}$ is composed of a reference quaternion, $\boldsymbol{q}_{\mathcal{IV}_{\text{ref}}}$, which is adjusted after each update step, and an error quaternion, $\boldsymbol{q}_{\mathcal{V}_{\text{ref}}\mathcal{V}}$, of which only its vector part, $\tilde{\boldsymbol{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}$, is in the EKF's state space. Therefore we get

$$\boldsymbol{q}_{\mathcal{IV}} = \boldsymbol{q}_{\mathcal{IV}_{\text{ref}}} \cdot \boldsymbol{q}_{\mathcal{V}_{\text{ref}}\mathcal{V}} \qquad\qquad \boldsymbol{q}_{\mathcal{V}_{\text{ref}}\mathcal{V}} = \begin{bmatrix} \sqrt{1 - \tilde{\boldsymbol{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}^\intercal \cdot \tilde{\boldsymbol{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}} \\ \tilde{\boldsymbol{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}} \end{bmatrix} \tag{C.13}$$

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

$$\frac{\partial}{\partial \tilde{\boldsymbol{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}} \boldsymbol{h}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{q}_{\mathcal{IV}}} \boldsymbol{h}(\boldsymbol{x}) \cdot \frac{\partial \boldsymbol{q}_{\mathcal{IV}}}{\partial \tilde{\boldsymbol{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}} \tag{C.14}$$

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

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

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, $\boldsymbol{p}_{VB,k}$ and $\boldsymbol{q}_{\mathcal{VB},k}$, with respect to the VIO frame $\mathcal{V}$. Note thate the VIO estimate is assumed to be a constant parameter, not a filter state, which vastly simplifies derivations ad computation, leading to an efficient yet robust filter.

Gate Measurements

Gate measurements consist of the image pixel coordinates, $\boldsymbol{s}_{Co_{ij}}$, 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 $\boldsymbol{p}_{G_i Co_{ij}} = \frac{1}{2}(0, \pm w, \pm h)$. The measurement equation can be written as the pinhole camera projection [229] of the gate corner into the camera frame. A pinhole camera maps the gate corner point, $\boldsymbol{p}_{Co_{ij}}$, expressed in the camera frame, $\mathcal{C}$, into pixel coordinates as

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

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, $\boldsymbol{p}_{Co_{ij}}$, is given by

$$\boldsymbol{p}_{Co_{ij}} = \boldsymbol{R}_{\mathcal{IC}}^{\intercal} \left(\boldsymbol{p}_{G_i} + \boldsymbol{R}_{\mathcal{IG}_i} \boldsymbol{p}_{G_i Co_j} - \boldsymbol{p}_C \right), \tag{C.17}$$

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

$$\boldsymbol{p}_C = \boldsymbol{p}_V + \boldsymbol{R}_{\mathcal{IV}} (\boldsymbol{p}_{VB} + \boldsymbol{R}_{\mathcal{VB}} \boldsymbol{p}_{BC}), \tag{C.18}$$

$$\boldsymbol{R}_{\mathcal{IC}} = \boldsymbol{R}_{\mathcal{IV}} \boldsymbol{R}_{\mathcal{VB}} \boldsymbol{R}_{\mathcal{BC}}, \tag{C.19}$$

where $\boldsymbol{p}_{BC}$ and $\boldsymbol{R}_{\mathcal{BC}}$ 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,

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

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,

$$\boldsymbol{S}_{\hat{i}} = \{\boldsymbol{s}_{Co_{\hat{i}j}0}, \ldots, \boldsymbol{s}_{Co_{\hat{i}j}m-1}\},$$

corresponding to the unknown gate $\hat{i}$ at known corners $j \in \mathcal{C}_j$. To identify the correspondences between a detection set $\boldsymbol{S}_{\hat{i}}$ 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, $\boldsymbol{s}_{Co_{ij}}$, according to (C.16) and then compute the square error sum between the measurement set, $\boldsymbol{S}_{\hat{i}}$, and the candidates, $\boldsymbol{s}_{Co_{ij}}$. Finally, the correspondence is established to the gate

G_i which minimizes the square error sum, as in

$$\underset{i\in[0,N-1]}{argmin} \sum_{\boldsymbol{s}_{Co_{\hat{i}j}} \in \boldsymbol{S}_{\hat{i}}} (\boldsymbol{s}_{Co_{\hat{i}j}} - \boldsymbol{s}_{Co_{ij}})^\intercal (\boldsymbol{s}_{Co_{\hat{i}j}} - \boldsymbol{s}_{Co_{ij}}). \tag{C.21}$$

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

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

The Jacobian with respect to the state space is again derived by $\frac{\partial h_{\text{LRF}}}{\partial \boldsymbol{p}_V}$ and $\frac{\partial h_{\text{LRF}}}{\partial \boldsymbol{q}_{\mathcal{IV}}}$ 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 $\ddot{\boldsymbol{p}}_B(t) = \boldsymbol{u}(t)$ with the input acceleration $\boldsymbol{u}(t)$ being constrained to $\underline{\boldsymbol{u}} \leq \boldsymbol{u}(t) \leq \overline{\boldsymbol{u}}$ are computed for each axis individually. Without loss of generality, only the x-axis is considered in the following. Using Pontryagin's maximum principle [24], it can be shown

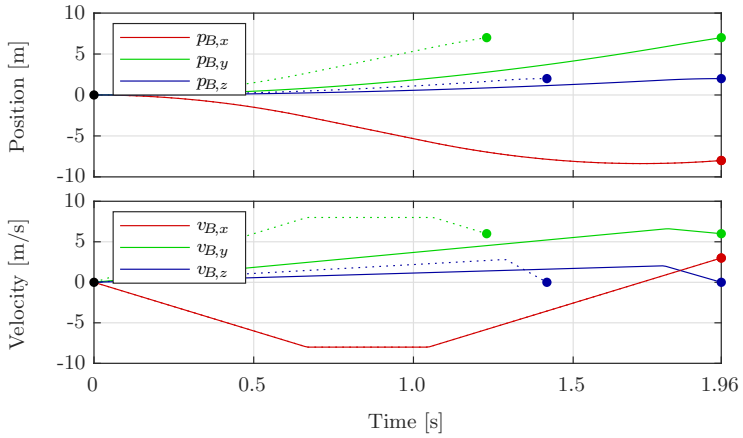


Figure C.5: Example time-optimal motion primitive starting from rest at the origin to a random final position with non-zero final velocity. The velocities are constrained to $\pm 7.5\text{m/s}$ and the inputs to $\pm 12\text{m/s}^2$. The dotted lines denote the per-axis time-optimal maneuvers.

that the optimal control input is bang-bang in acceleration, i.e., has the form

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

or vice versa with the control input first being $\overline{u}_x$ 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 $\underline{\boldsymbol{v}}_B \leq \boldsymbol{v}_B(t) \leq \overline{\boldsymbol{v}}_B$ can be added, in which case the optimal control input has bang-singular-bang solution [144]

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

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 $\alpha\overline{u}_x$, 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 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 [24] resulting in the optimal path $\boldsymbol{p}^*(t)$. 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, $\bar{\boldsymbol{p}}_k(t)$, 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.,

$$\bar{\boldsymbol{p}}_k(t_k) = \boldsymbol{p}^*(t_k), \qquad \bar{\boldsymbol{p}}_k(t_{k+1}) = \boldsymbol{p}^*(t_{k+1}), \tag{C.25}$$

$$\dot{\bar{\boldsymbol{p}}}_k(t_k) = \dot{\boldsymbol{p}}^*(t_k), \qquad \dot{\bar{\boldsymbol{p}}}_k(t_{k+1}) = \dot{\boldsymbol{p}}^*(t_{k+1}), \tag{C.26}$$

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

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

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

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

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

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

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

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 $\zeta_{\text{pos},z}$,

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

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

$$f = \frac{\left[m\left(\ddot{\boldsymbol{p}}_{B_{\text{ref}}} + \boldsymbol{g}\right) + \boldsymbol{R}_{\mathcal{IB}}\boldsymbol{D}\boldsymbol{R}_{\mathcal{IB}}^{\mathsf{T}}\boldsymbol{v}_{\mathcal{B}}\right]_z}{\left[R_{\mathcal{IB}}\boldsymbol{e}_z^{\mathcal{B}}\right]_z}. \tag{C.32}$$

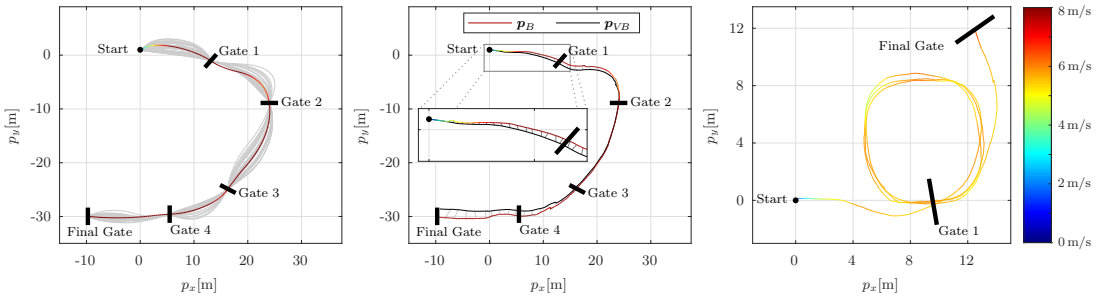


Figure C.6: Top view of the planned (left) and executed (center) path at the championship race, and an executed multi-lap path at a testing facility (right). Left: Fastest planned path in color, sub-optimal sampled paths in gray. Center: VIO trajectory as $\boldsymbol{p}_{\mathcal{V}B}$ and corrected estimate as $\boldsymbol{p}_{\mathcal{B}}$.

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, φ_{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 [31] 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:

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

where q_w, q_x, q_y and q_z are the components of the attitude error, $\boldsymbol{q}_{\mathcal{IB}}^{-1} \otimes \boldsymbol{q}_{\mathcal{IB}_{\text{ref}}}$, 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 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 [198] 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 $2 - 17\,\text{m}$. Fig. C.4 illustrates the quality of detections during the championship race (1st row) as well as for cases with 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.

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

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 [124, 104, 110], 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 $\boldsymbol{p}_{VB}$ and the EKF-corrected trajectory as $\boldsymbol{p}_B$ (the estimated corrections are depicted in gray). Accumulated drift clearly leads to a large discrepancy between VIO estimate $\boldsymbol{p}_{VB}$ and the corrected estimate $\boldsymbol{p}_B$. 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 $8\,\mathrm{m/s}$ and the angle between the velocity and the gate normal limited to $\varphi_{\max} = 30°$. 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 comapred 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

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 [71] (full track)	6.520 s	$4.62 \cdot 10^5$ms 6524%

time of the time-optimal trajectory generation from [71], 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 particularly 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 Faster than FAST: GPU-Accelerated Frontend for High-Speed VIO

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

Balazs Nagy, Philipp Foehn, and Davide Scaramuzza. "Faster than FAST: GPU-Accelerated Frontend for High-Speed VIO". in: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2020

Faster than FAST: GPU-Accelerated Frontend for High-Speed VIO

Balázs Nagy, Philipp Foehn, Davide Scaramuzza

Abstract — The recent introduction of powerful embedded graphics processing units (GPUs) has allowed for unforeseen improvements in real-time computer vision applications. It has enabled algorithms to run onboard, well above the standard video rates, yielding not only higher information processing capability, but also reduced latency. This work focuses on the applicability of efficient low-level, GPU hardware-specific instructions to improve on existing computer vision algorithms in the field of visual-inertial odometry (VIO). While most steps of a VIO pipeline work on visual features, they rely on image data for detection and tracking, of which both steps are well suited for parallelization. Especially non-maxima suppression and the subsequent feature selection are prominent contributors to the overall image processing latency. Our work first revisits the problem of non-maxima suppression for feature detection specifically on GPUs, and proposes a solution that selects local response maxima, imposes spatial feature distribution, and extracts features simultaneously. Our second contribution introduces an enhanced FAST feature detector that applies the aforementioned non-maxima suppression method. Finally, we compare our method to other state-of-the-art CPU and GPU implementations, where we always outperform all of them in feature tracking and detection, resulting in over 1000fps throughput on an embedded Jetson TX2 platform. Additionally, we demonstrate our work integrated into a VIO pipeline achieving a metric state estimation at ∼200fps.

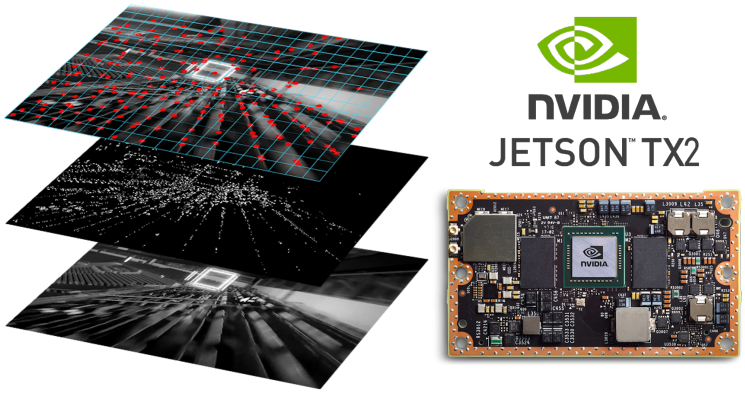


Figure D.1: Our method introduces a novel non-maxima suppression scheme exploiting GPU parallelism and low-level instructions, applied for GPU-optimized feature detection and tracking. We demonstrate a feature detection and tracking rate of **over 1000fps** on an embedded Jetson TX2 platform.

D.1 Introduction

D.1.1 Motivation

As technology became increasingly affordable, vision-based motion tracking has proven its capabilities not only in robotics applications, such as autonomous cars and drones but also in virtual (VR) and augmented (AR) reality and mobile devices. While visual-inertial odometry (VIO) prevails with its low cost, universal applicability, and increasing maturity and robustness, it is still computationally expensive and introduces significant latency. This latency impacts e.g. VR/AR applications by introducing motion sickness, or robotic systems by constraining their control performance. The latter is especially true for aerial vehicles with size and weight constraints limiting the available computation power while requiring real-time execution of the VIO and control pipeline to guarantee stable, robust, and safe operation. Besides latency, one may also witness a disconnect between the available sensor capabilities (both visual and inertial) and the actual information processing capabilities of mobile systems. While off-the-shelf cameras are capable of capturing images above 100fps, many algorithms and implementations are not able to handle visual information at this rate. By lowering the frame processing times, we can simultaneously minimize latency and also reduce the neglected visual-inertial information.

In particular, embedded systems on drones or AR/VR solutions cannot rely on offline computations and therefore need to use all their available resources efficiently. Various heterogeneous embedded solutions were introduced, offering a range of computing architectures for better efficiency. There are three popular heterogeneous architectures: i) the first one uses a central processing unit (CPU) with a digital signal processor (DSP), and therefore it is restricted in its set of tasks; ii) the second one combines a CPU with programmable logic (e.g. FPGA), which is versatile but increases development time; iii) the third solution is the combination of a CPU with a GPU, which is not only cost-efficient

but also excels in image processing tasks since GPUs are built for highly parallel tasks.

On these grounds, our work investigates feature detection and tracking on CPU-GPU platforms, where we build up the image processing from the GPU hardware's perspective. We present a feasible work-sharing between the CPU and a GPU to achieve significantly lower overall processing times.

D.1.2 Related Work

We recapitulate previous approaches according to the building blocks of a VIO pipeline front-end: feature detection, non-maxima suppression, and feature tracking.

Feature Detection

Over the years, feature detection has not changed significantly. Pipelines commonly use Harris [91], Shi-Tomasi [205], or FAST features [200] with one of three FAST corner scores. Harris and Shi-Tomasi are less sensitive to edges and are also widely used independently as corner detectors. They both share the same principles, but their metric of cornerness differs. ORB [202], as an extension to FAST has also appeared in VIO pipelines, presenting a reasonable, real-time alternative to SIFT [135] and SURF [20]. Amongst the above, undoubtedly FAST presents the fastest feature detector. Two variations were proposed from the original authors in the form of FAST [200] and FAST-ER [201], where the latter greatly improves on repeatability while still maintaining computational efficiency. To the best of our knowledge, the fastest CPU implementation of the FAST detector is KFAST [178], which showcases more than 5× speedup over the original implementation. On the GPU, we are aware of two optimized implementations within OpenCV [30] and ArrayFire [249]. Both employ lookup tables to speed up the decision process for determining a point's validity: the latter uses a 64-kilobyte lookup table, while the former an 8-kilobyte one. Although both solutions provide fast cornerness decision, neither of them guarantees spatial feature distribution as they both stop extracting features once the feature count limit is reached.

Non-Maxima Suppression

Non-maxima suppression can be considered a local maximum search within each candidate's Moore neighborhood. The Moore neighborhood of each pixel-response is its square-shaped surrounding with a side length of $(2n + 1)$. Suppression of interest point candidates has been studied extensively in [168, 185], and recently, they have also been reviewed in machine learning applications [179]. The complexity of the proposed algorithms is determined based on the number of comparisons required per interest point. This algorithm requires $(2n + 1)^2$ comparisons, as the comparisons follow the raster scan

order. Förstner and Gülch [78] proposed performing the comparisons in spiral order. Theoretically, the number of comparisons does not change, however, the actual number of comparisons plummeted, because most candidates can be suppressed in a smaller 3x3 neighborhood first, and only a few points remain. Neubeck [168] proposed several algorithms to push down the number of comparisons to almost 1 in the worst-case. Pham [185] proposed another two algorithms (scan-line, and quarter-block partitioning) that drove down the number of comparisons below 2, not only for larger ($n \geq 5$) but also for small neighborhood sizes ($n < 5$). All these approaches first try to perform reduction to a local maximum candidate in a smaller neighborhood, then perform neighborhood verification for only the selected candidates. However, they do not ensure any (spatial) feature distribution and possibly output a large set of feature candidates.

Feature tracking

Feature tracking may be divided into three different categories: i) feature matching, ii) filter-based tracking, and iii) differential tracking. Feature matching i) applies feature extraction on each frame followed by feature matching, which entails a significant overhead. Moreover, the repeatability of the feature detector may adversely influence its robustness. However, there are well-known pipelines, opting for this approach [157, 123, 162]. [25] tracks the features using filters ii), which contain the feature location in its state (e.g. via bearing vectors), and follows features with consecutive prediction and update steps. The third differential iii) approaches aim to directly use the pixel intensities and minimize a variation of the photometric error. From the latter kind, the Lucas-Kanade tracker [136, 15, 14] became ubiquitous in VIO pipelines [75, 76] due to its efficiency and robustness. As it directly operates on pixel intensity patches, GPU adaptations appeared early on. [251] implements a translational displacement model on the GPU with intensity-gain estimation. [115, 100] go even further: they propose an affine-photometric model coupled with an inertial measurement unit (IMU) initialization scheme: the displacement model follows an affine transformation, in which the parameters are affected by the IMU measurements between consecutive frames, while the pixel intensities may also undergo an affine transformation.

D.1.3 Contributions

Our work introduces a novel non-maxima suppression building upon [78] but also exploiting low-level GPU instruction primitives, completed by a GPU-optimized implementation of the FAST detector with multiple scores. Our method combines the feature detection and non-maxima suppression, guaranteeing uniform feature distribution over the whole image, which other approaches need to perform in an extra step. Additionally, we combine our frontend with a state-of-the-art VIO bundle-adjustment backend. All contributions are verified and thoroughly evaluated using the EuRoC dataset [33] on the Jetson

TX2 embedded platform and a laptop GPU. Throughput capabilities of over 1000fps are demonstrated for feature detection and tracking, and ∼200fps for a VIO pipeline recovering a metric state estimate.

D.2 Methodology

D.2.1 Preliminaries on parallelization

We briefly introduce the fundamentals of the Compute Unified Device Architecture, or shortly CUDA, which is a parallel computing platform and programming model proprietary to NVIDIA. CUDA allows developers to offload the central processing unit, and propagate tasks to the GPU even for non-image related computations. Latter is commonly referred to as general-purpose GPU (GPGPU) programming.

The NVIDIA GPU architecture is built around a scalable array of multithreaded streaming multiprocessors (SMs) [43]. Each SM has numerous streaming processors (SP), that are lately also called CUDA cores. GPUs usually have 1-20 streaming multiprocessors and 128-256 streaming processors per SM. In addition to the processing cores, there are various types of memories available (ordered by proximity to the processing cores): register file, shared memory, various caches, off-chip device, and host memory.

NVIDIA's GPGPU execution model introduces a hierarchy of computing units: threads, warps, thread blocks, and thread grids. The smallest unit of execution is a thread. Threads are grouped into warps: each warp consists of 32 threads. Warps are further grouped into thread blocks. One thread block is guaranteed to be executed on the same SM. Lastly, on top of the execution model is the thread grid. A thread grid is an array of thread blocks. Thread blocks within a thread grid are executed independently from each other.

The instruction execution on the GPU needs to be emphasized: every thread in a warp executes the same instruction in a lock-step basis. NVIDIA calls this execution model Single Instruction Multiple Threads (SIMT). It also entails, that *if/else* divergence within a warp causes serialized execution.

The underlying GPU hardware occasionally undergoes significant revisions, hence the differences between GPUs need to be tracked. NVIDIA introduced the notion of Compute Capability accompanied by a codename to denote these differences. With the introduction of the NVIDIA Kepler GPU microarchitecture, threads within the same warp can read from each other's registers with specific instructions. Our work focuses on these warp-level primitives, more specifically, highly-efficient communication patterns for sharing data between threads in the same warp. In previous GPU generations, threads needed to turn to a slower common memory (usually the shared memory) for data sharing, which

Table D.1: Memory selection for the fastest communication

Execution Unit	Execution Unit	Fastest Memory
Threads within warp	identical SM	registers
Warps within thread block	identical SM	shared memory
Thread blocks	any SM	global memory

Algorithm 1: Generalized Feature Detection

```
for Every scale do
    for Every pixel within the region of interest (ROI) do
        if Coarse Corner Response Function (CCRF) then
            Corner Response Function (CRF)
        end
    end
    Non-max. suppression within neighborhood (NMS)
end
• Non-max. suppression within cell (NMS-C)
```

resulted in significant execution latencies. However, with the introduction of the Kepler architecture it became possible to perform communication within the warp first, and only use the slower memory on higher abstractions of execution, i.e. within thread blocks and then within the thread grid. In Table D.1 we summarized the available fastest memories for exchanging data between execution blocks on a GPU.

D.2.2 Feature detector overview

GPUs are particularly well-suited for feature detection, which can be considered a stencil operation amongst the parallel communication patterns. In a stencil operation, each computational unit accesses an input element (e.g. pixel) and its close neighborhood in parallel. Therefore, the image can be efficiently divided amongst the available CUDA cores, such that the memory accesses are coalesced, leading to highly effective parallelization. For feature detection, the input image is first subsampled to acquire an image pyramid. Then, for each image resolution, two functions are usually evaluated at each pixel: a coarse corner response function (CCRF), and a corner response function (CRF). CCRF serves as a fast evaluation that can swiftly exclude the majority of candidates so that a slower CRF function only receives candidates that passed the first verification. Once every pixel has been evaluated within the ROI, non-maxima suppression is applied to select only the local maxima. We summarized the general execution scheme of feature detector algorithms in Algorithm 1. It was discovered in [174, 209, 79], that uniform feature distribution on image frames improves the stability of VIO pipelines. To fulfill this requirement, [75] and [76] introduced the notion of 2D grid cells: the image is divided into rectangles with a fixed width and height. Within each cell, there is only one feature selected - the feature whose CRF score is the highest within the cell. Not only does this

method distribute the features evenly on the image, but it also imposes an upper limit on the extracted feature count. This is shown in Algorithm 1, including the augmentation •.

D.2.3 Non-maxima suppression with CUDA

Feature selection within a cell can be understood as a reduction operation, where only the feature with the maximum score is selected. Moreover, non-maxima suppression within the neighborhood can also be considered a reduction operation, where one reduces the corner response to a single-pixel location within finite neighborhoods.

Our approach divides the corner response map into a regular cell grid. Within the grid on the first pyramid level, cells use a width of an integer multiple of 32, i.e. $32w$, because, on NVIDIA GPU hardware, a warp consists of 32 threads. One line of a cell is referred to as a cell line, which can be split up into cell line segments with 32 elements. We restrict the height of the cells to be $2^{l-1}h$, where l is the number of pyramid levels utilized during feature detection. By selecting an appropriate grid configuration l, w, and h, one can determine the maximum number of features extracted, while maintaining spatial distribution.

Within one cell line segment, one thread is assigned to process one pixel-response, i.e. one warp processes one entire cell line segment (32 threads for 32 pixel-responses). While one warp can process multiple lines, multiple warps within a thread block cooperatively process consecutive lines in a cell. As the corner response map is stored using a single-precision floating-point format in a pitched memory layout, the horizontal cell boundaries perfectly coincide with the L1-cache line boundaries, which maximizes the memory bus utilization whenever fetching complete cell line segments.

For the simplicity of illustration, a 1:1 warp-to-cell mapping is used in a 32x32 cell. As a warp reads out the first line of the cell, each thread within the warp, has acquired one pixel-response. As the next operation, the entire warp starts the neighborhood suppression: the warp starts spiraling according to [78], and each thread verifies whether the response it has is the maximum within its Moore neighborhood. Once the neighborhood verification finishes, a few threads might have suppressed their response. However, no write takes place at this point, every thread stores its state (response score, and x-y location) in registers. The warp continues with the next line, and repeats the previous operations: they read out the corresponding response, and start the neighborhood suppression, and update their maximum response if the new response is higher than the one in the previous line. The warp continues this operation throughout all cell lines until it processed the entire cell. Upon finishing, each thread has its maximum score with the corresponding 2D location. However, as the 32 threads were processing individual columns, the maximum is only column-wise. Therefore, the warp needs to perform a warp-level reduction to get the cell-wise maximum: they reduce the maximum score and location to the first thread

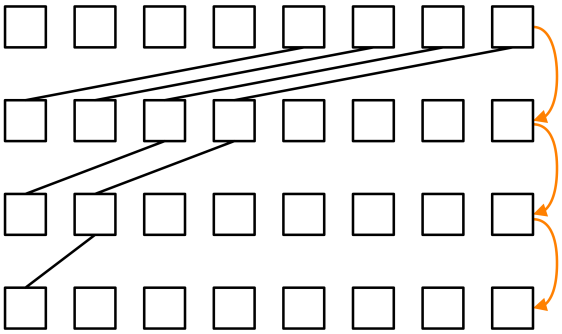


Figure D.2: Warp-level communication pattern during cell maximum selection. At the end of the communication, thread 0 has the valid maximum.

(thread 0) using warp-level shuffle down reduction [50]. The applied communication pattern is shown in Figure D.2. Thread 0 finally writes the result to global memory.

To speed up reduction, multiple (M) warps process one cell, therefore, after the warp-level reduction, the maximum is reduced in shared memory. Once all warps within the thread block wrote their maximum results (score, x-y location) to their designated shared memory area, the first thread in the block selects the maximum for each cell and writes it to global memory, finishing the processing of this cell.

During pyramidical feature detection, we maintain only one grid. On level 0 (original resolution), the above-specified algorithm applies. On lower pyramid levels, we virtually scale the cell sizes, such that the applicable cell size on level k becomes $(\frac{32 \cdot w}{k}, \frac{2^{l-1}h}{k})$. In case the cell width falls below 32, one warp may process multiple lines: if the consecutive cell lines still belong to the same grid cell, the warp can analogously perform the warp-level reduction. Since a lower pyramid level's resolution is half of its upper layer's resolution, we can efficiently recompute where a pixel response falls from lower pyramid levels on the original grid. That is, when we identify a cell maximum on a lower pyramid level, the 2D position (x, y) from the lower resolution can be scaled up to $(2^l x, 2^l y)$.

Looking back at Algorithm 1, our approach combines the regular neighborhood suppression (NMS) and cell maximum selection (NMS-C) into a single step. It also differs from [168, 185], because we first perform candidate suppression within each thread in parallel, then reduce the remaining candidates amongst one cell.

D.2.4 FAST feature detector

The FAST feature detector's underlying idea is simple: for every pixel location in the original image (excluding a minimum border of 3 pixels on all sides) we perform a segment test, in which we compare pixel intensities on a Bresenham circle with a radius of 3. This Bresenham circle gives us 16 pixel-locations around each point (see Figure D.3). We give labels L_x to these points based on a comparison between the center's and the actual

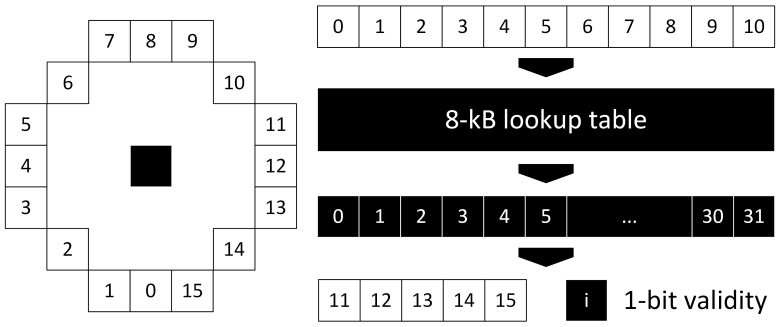


Figure D.3: FAST corner point evaluation with an 8 kilobyte lookup table

point's intensity.

Given there is a continuous arc of at least N pixels that are labeled either *darker* or *brighter*, the center is considered a corner point. To add more robustness to the comparisons, a threshold value (ϵ) is also applied. The comparisons are summarized in (D.1). Both the number of continuous pixels (N) and the threshold value (ϵ) are tuning parameters.

$$L_x = \begin{cases} \text{darker} & I_x < I_{center} - \epsilon \\ \text{similar} & I_{center} - \epsilon \leq I_x \leq I_{center} + \epsilon \\ \text{brighter} & I_{center} + \epsilon < I_x \end{cases} \tag{D.1}$$

Avoiding Execution Divergence in FAST Calculation

If each thread performed the comparisons from (D.1) in NVIDIA's single-instruction-multiple-threads (SIMT) execution model, the comparison decisions in *if/else*-instructions will execute different code blocks. Since all threads execute the same instruction in a warp, some threads will be inactive during the *if*-branch and others during the *else*-branch. This is called code divergence and reduces the throughput in parallelization significantly, but it can be resolved with a completely different approach: a lookup table (Figure D.3).

Our approach stores the result of the 16 comparisons as a bit array, which serves as an index for the lookup table. All possible 16-bit vectors are precalculated: a bit b_x is '1' if the pixel intensity on the Bresenham circle I_x is darker/brighter than the center pixel intensity I_{center}, and '0' if the pixel intensities are similar. As the result is binary for all 2^{16} vectors, the answers can be stored on 2^{16} bits, i.e. 8 kilobytes. These answers can be stored by using 4-byte integers, each of which store 32 combinations (2^5): 11 bits are used to acquire the address of the integer, and the 5 unused bits then select one bit out of the 32. If the resulting bit is set, we proceed with the calculation of the corner score.

The literature distinguishes three different types of scores for a corner point: sum of absolute differences on the entire Bresenham circle (SAD-B); sum of absolute differences on the continuous arc (SAD-A) [200]; maximum threshold (ϵ) for which the point is still considered a corner point (MT) [201]. The corner score is 0 if the segment test fails.

Our approach compresses the validity of each 16-bit combination into a single bit, resulting in an 8-kilobyte lookup table, for which the cache-hit ratio is higher than the work presented in [249]. This results from the increased reuse of each cache line that is moved into the L1 (and L2) caches, therefore improving the access latency.

D.2.5 Lucas-Kanade Feature Tracker

Our approach deploys the pyramidical approximated simultaneous inverse compositional Lucas-Kanade algorithm as feature tracker. The Lucas-Kanade [136] algorithm minimizes the photometric error between a rectangular patch on a template and a new image by applying a warping function on the image coordinates of the new image. The inverse compositional algorithm is an extension that improves on the computational complexity per iteration [15] by allowing to precompute the Hessian matrix and reuse it in every iteration. The simultaneous inverse compositional Lucas-Kanade adds the estimation of affine illumination change. However, as the Hessian becomes the function of the appearance estimates, it cannot be precomputed anymore, which makes this approach even slower than the original Lucas-Kanade. Therefore, our approach applies the approximated version, where the appearance parameters are assumed to not change significantly, and hence the Hessian can be precomputed with their initial estimates [14].

We use a translational displacement model $\boldsymbol{t}$ with affine intensity variation estimation $\boldsymbol{\lambda}$. The complete set of parameters are $\boldsymbol{q} = [\boldsymbol{t}, \boldsymbol{\lambda}]^\intercal = [t_x, t_y, \alpha, \beta]^\intercal$, where t_x, t_y are the translational offsets, while α, β are the affine illumination parameters, resulting in the warping

$$\boldsymbol{W}(\boldsymbol{x}, \boldsymbol{t}) = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix}. \tag{D.2}$$

The per-feature photometric error that we try to minimize for each feature with respect to $\Delta\boldsymbol{q} = [\Delta\boldsymbol{t}, \Delta\boldsymbol{\lambda}]$ is

$$\begin{aligned} \min \sum_{\boldsymbol{x} \in N} \Big[& T(\boldsymbol{W}(\boldsymbol{x}, \Delta\boldsymbol{t})) - I(\boldsymbol{W}(\boldsymbol{x}, \boldsymbol{t})) + \\ & (\alpha + \Delta\alpha) \cdot T(\boldsymbol{W}(\boldsymbol{x}, \Delta\boldsymbol{t})) + (\beta + \Delta\beta) \Big]^2, \end{aligned} \tag{D.3}$$

where $T(\boldsymbol{x})$ and $I(\boldsymbol{x})$ stand for the template image and the current image intensities at position $\boldsymbol{x}$, respectively. The vector $\boldsymbol{x}$ iterates through one feature's rectangular neighborhood ($\mathcal{N}$). We can organize the coefficients of the incremental terms into vector

form as

$$\boldsymbol{U}(\boldsymbol{x}) = \begin{bmatrix} (1+\alpha)\frac{\partial T(\boldsymbol{x})}{\partial x}\frac{\partial \boldsymbol{W}(\boldsymbol{x},\boldsymbol{t})}{\partial t_x} \\ (1+\alpha)\frac{\partial T(\boldsymbol{x})}{\partial y}\frac{\partial \boldsymbol{W}(\boldsymbol{x},\boldsymbol{t})}{\partial t_y} \\ T(\boldsymbol{x}) \\ 1 \end{bmatrix}, \tag{D.4}$$

and the minimization problem can be rewritten as

$$\min \sum_{\boldsymbol{x}\in N} \left[(1+\alpha)T(\boldsymbol{x}) + \beta - I(\boldsymbol{W}(\boldsymbol{x},\boldsymbol{t})) + \boldsymbol{U}^{\intercal}(\boldsymbol{x})\Delta\boldsymbol{q} \right]^2. \tag{D.5}$$

After computing the derivative of (D.5) and setting it to zero, the solution to $\Delta\boldsymbol{q}$ is found using the Hessian $\boldsymbol{H}$ by

$$\begin{aligned} \Delta\boldsymbol{q} &= \boldsymbol{H}^{-1} \sum_{\boldsymbol{x}\in N} \boldsymbol{U}^{\intercal}(\boldsymbol{x}) \left[I(\boldsymbol{W}(\boldsymbol{x},\boldsymbol{t})) - (1+\alpha)T(\boldsymbol{x}) - \beta \right] \\ \boldsymbol{H}(\boldsymbol{x}) &= \sum_{\boldsymbol{x}\in N} \boldsymbol{U}^{\intercal}(\boldsymbol{x})\boldsymbol{U}(\boldsymbol{x}). \end{aligned} \tag{D.6}$$

For this algorithm, there are two GPU problems that need to be addressed: memory coalescing and warp divergence. The problems are approached from the viewpoint of VIO algorithms, where one generally does not track a high number of features (only 50-200), and these sparse features are also scattered throughout the image, which means that they are scattered in memory.

This algorithm minimizes the photometric error in a rectangular neighborhood around each feature on multiple pyramid levels. Consequently, if threads within a warp processed different features, the memory accesses would be uncoalesced, and given some feature tracks do not converge or the number of iterations on the same level differs, some threads within a warp would be idle. To address both of these concerns, one entire warp is launched for processing one feature. We also opted for rectangular patch sizes that can be collaboratively processed by warps: on higher resolutions 16x16, on lower resolutions 8x8 pixels. It solves warp divergence since threads within the warp perform the same number of iterations and they iterate until the same pyramid level. The memory requests from the warp are also split into fewer memory transactions, as adjacent threads are processing consecutive pixels or consecutive lines.

Warp-level primitives are exploited in every iteration of the minimization, as we need to perform a patch-wide reduction: in (D.6) we need to sum a four element vector $\boldsymbol{U}(\boldsymbol{x})^T r(\boldsymbol{x},\boldsymbol{t})$ for every pixel within the patch. One thread processes multiple pixels within the patch, hence each thread reduces multiple elements into its registers prior to any communication. Once the entire warp finishes a patch, the threads need to share

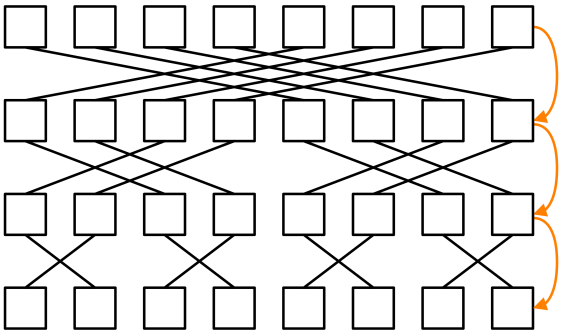


Figure D.4: Warp-level communication pattern during each minimization iteration, after which each thread has the sum of all individual thread results.

their local results with the rest of the warp to calculate $\Delta \boldsymbol{q}$. This reduction is performed using the butterfly communication pattern shown in Figure D.4.

The novelty of our approach lies in the thread-to-feature assignment. Approaches presented in [115, 100] are using a one-to-one assignment, which implies that only large feature-counts can utilize a large number of threads. This adversely affects latency hiding on GPUs with smaller feature counts, which is generally applicable to VIO. Our method speeds up the algorithm by having warps that collaboratively solve feature patches, where each thread's workload is reduced, while the used communication medium is the fastest possible.

D.3 Evaluation

We evaluate in four parts: non-maxima suppression, standalone feature detection, feature tracking (all on the EuRoC Machine Hall 01 sequence [33], including 3,682 image frames), and applicability within a VIO pipeline. The full VIO pipeline is implemented with the bundle-adjustment from [126] and tested on the Machine Hall EuRoC dataset sequences [33].

D.3.1 Hardware

We performed our experiments on an NVIDIA Jetson TX2 and a laptop computer with an Intel i7-6700HQ processor and a dedicated NVIDIA 960M graphics card. The Jetson TX2 was chosen because of its excellent tradeoff between size, weight, and computational capabilities, where we run all experiments with the platform in *max-N* performance mode (all cores and GPU at maximal clock speeds). The properties of the two platforms are summarized in Table D.2.

Table D.2: Comparison of GPU evaluation hardware

	Tegra X2	**960M**
GPU Tier	embedded	notebook
CUDA Capability	6.2	5.0
CUDA Cores	256	640
Maximum GPU Clock	1300 MHz	1176 MHz
Single-precision Performance	665.6 GF/s	1505.28 GF/s
Memory Bandwidth	59.7 GB/s	80 GB/s

D.3.2 Non-Maxima Suppression

The proposed FAST corner response function ($\epsilon = 10$, $N = 10$) was used as input to our non-maxima suppression, run with a grid granularity of 32×32 ($l = 1, w = 1, h = 32$) and compared to Förstner's [78] spiral non-maxima suppression algorithm. The results are listed in Table D.3, showing a 2× speedup for the embedded Jetson TX2 platform.

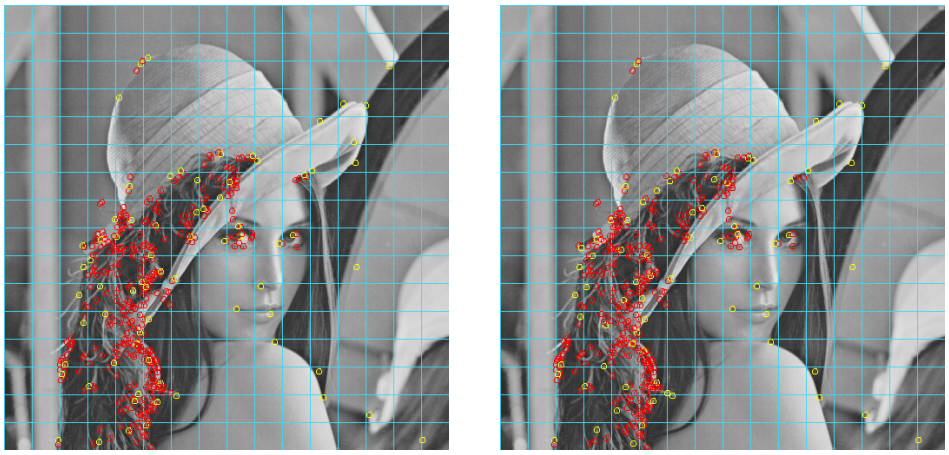
D.3.3 Feature detector

Conformance

We verify our feature detection conformance with the original FAST feature detector, for both suggested score functions: D.5a the sum of the absolute difference between the center pixel and the contiguous arc; D.5b the maximum threshold value, for which the point is detected as a corner. As our combined non-maxim suppression selects a single maximum within each cell, our output only comprises of a subset of the original implementation's output. In Figure D.5 we mark features red ○ which are output from the original detector, yellow ○ which are output from both implementations, and blue ○ false-positives of our detector. Note that there are no false-positives and that our method returns a well-distributed subset of the original response, rendering additional feature selection unneccessary,

Table D.3: Comparison of 2D non-maxima suppression kernels on GPU

	Tegra X2		**960M**	
NMS method	Förstner	Ours	Förstner	Ours
n=1 (3x3)	294.03 μs	**141.36 μs**	96.81 μs	**73.78 μs**
n=2 (5x5)	696.57 μs	**338.03 μs**	245.96 μs	**158.93 μs**
n=3 (7x7)	1207 μs	**604.94 μs**	441.59 μs	**288.39 μs**
n=4 (9x9)	1772 μs	**929.82 μs**	661.90 μs	**450.08 μs**



(a) Sum of absolute differences [200] **(b)** Maximum threshold value [201]

Figure D.5: Conformance verification with the original FAST detector (○) and our combined FAST/NMS (○ for conforming detections, ○ for false-positives). Note that there are no false-positives and that our method returns a well-distributed subset of the original response, therefore rendering additional feature selection unnecessary. Best viewed in digital paper.

Cache-hit ratio

As mentioned in D.2.4, we expect higher GPU cache-hit ratios during feature detection with our bit-based CRF lookup table. This reduces the number of global-memory transactions, resulting in lower kernel execution times. The cache-hit ratios and the resulting CRF timings are listed in Table D.4.

Execution time breakdown

We split the execution time of pyramidal feature detection ($l = 2$) into its constituents: image copy from host to device memory (*Upload*), creation of an image pyramid where each subsequent layer halves the resolution of the previous one (*Pyramid*), corner response function evaluation (*CRF*), non-maxima suppression with cell-maximum selection (*NMS*), and feature grid copy from device memory to host memory (*Download*).

Table D.4: Timing Comparison of different FAST CRF Scores

		Tegra X2		**960M**	
Lookup table		byte-based	bit-based	byte-based	bit-based
SAD-B	**L1 cache-hit rate**	83.9 %	**89.9 %**	68.6 %	**77.4 %**
	CRF kernel	317.3 μs	**298.7 μs**	141.3 μs	**135.9 μs**
SAD-A	**L1 cache-hit rate**	83.9 %	**89.8 %**	68.5 %	**77.3 %**
	CRF kernel	348.4 μs	**334.9 μs**	158.5 μs	**155.1 μs**
MT	**L1 cache-hit rate**	84.0 %	**91.8 %**	71.9 %	**82.7 %**
	CRF kernel	815.1 μs	**784.3 μs**	410.6 μs	**374.8 μs**

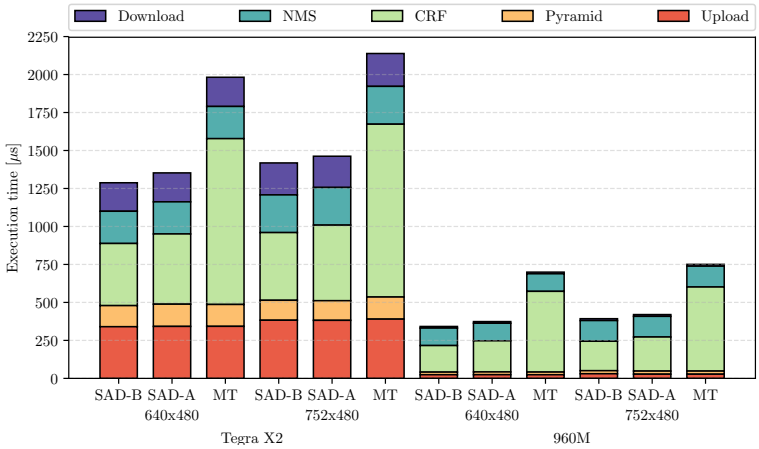


Figure D.6: Feature detector execution time breakdown into image upload, pyramid creation, CRF, NMS, and feature download. Best viewed in color.

Execution time comparison

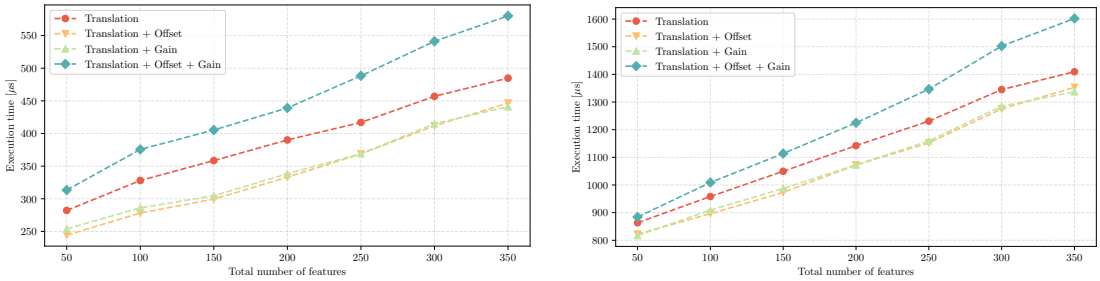
We compare our feature detection with other publicly available FAST implementations, summarized in Table D.5. As the publicly available detectors only support single-scale, we performed these experiments on the original image resolution. Note that our method not only performs feature extraction but simultaneously applies cell-wise non-maxima suppression, still achieving superior execution times. As KFAST uses x86-specific instruction set extensions, while ArrayFire OpenCL was incompatible with our available packages, these could not be run on the Jetson TX2. The GPU timings include the image upload and feature list download times as well.

D.3.4 Feature tracker

We timed our feature tracker implementation by varying the total number of simultaneous feature tracks on both testing platforms, depicted in Figure D.7. We utilized our FAST

Table D.5: FAST feature detector average execution time comparison

	Tegra X2	**960M**
Others (feature detection and NMS)		
OpenCV CPU with MT	4.78 ms	2.23 ms
OpenCV CUDA with MT	2.73 ms	1.09 ms
ArrayFire CPU with SAD-A	60.83 ms	36.42 ms
ArrayFire CUDA with SAD-A	1.47 ms	0.51 ms
ArrayFire OpenCL with SAD-A	-	0.91 ms
KFAST CPU with MT	-	0.63 ms
Ours (feature detection, NMS, and NMS-C)		
Ours with SAD-B	1.11 ms	0.28 ms
Ours with SAD-A	1.14 ms	0.30 ms
Ours with MT	1.59 ms	0.52 ms



(a) Performance on 960M

(b) Performance on Tegra X2

Figure D.7: Feature tracker performance comparison over a varying number of tracked features, with translation, illumination gain and offset estimation, and subsets of those. Best viewed in color.

with score (ii) as feature detector, and triggered feature re-detection whenever the feature track count falls below 30% of the actual target. The evaluations include the full affine intensity and translation estimation, with a total of 4 estimated parameters. We also show the translation-only estimation, translation-gain, as well as translation-offset estimation. Note that translation-only estimation is less efficient since it needs more iterations to converge, as visible in Figure. D.7. Lastly, Table D.6 shows a comparison against OpenCV's CPU and GPU implementation, which we outperform by a factor of 2× and more.

D.3.5 Visual odometry

Lastly, we evaluate the performance of our frontend in combination with a VIO bundle-adjustment backend. We chose ICE-BA [126] as backend, since it is accurate, efficient and also achieves extremely fast execution times. We implement two test cases: first, we run the ICE-BA backend with their proposed frontend as a baseline and then compare it against our frontend with their backend. Both frontends employ FAST features (our implementation vs. theirs), a Lucas-Kanade feature tracker with 70 tracked features. We tuned the original ICE-BA configuration [126] and reduced the local bundle adjustment (LBA) window size to 15 frames for both cases, while we kept other parameters unchanged.

Table D.6: Feature tracker average execution time comparison, tracking 100 features, re-detection at 30 features

	Tegra X2	**i7-6700HQ+960M**
OpenCV CPU	1.88 ms	1.38 ms
OpenCV CUDA	3.96 ms	0.74 ms
Ours trans. only	0.96 ms	0.33 ms
Ours trans. & offset	0.90 ms	0.28 ms
Ours trans. & gain	0.91 ms	0.29 ms
Ours trans. & gain & offset	1.01 ms	0.38 ms

We summarize our findings in Table. D.7, which shows an average speedup factor of 2.25× of the combined front- and back-end on both platforms, with an average accuracy loss of 0.47%. Our tracking performance on MH_04 and MH_05 is affected by the faster motions and dark scenes. However, according to [49], most pipelines suffer from largely increased tracking error on these two sequences. This is due to the relatively dark appearance combined with fast motions of some scenes in these sequences, introducing higher noise in feature tracking. The VIO pipeline combining our GPU-accelerated frontend and the CPU targeted ICE-BA backend allows us to achieve a throughput of ∼200fps on multiple datasets on the embedded Jetson TX2 platform.

D.4 Conclusion

This work introduces a novel non-maxima suppression exploiting low-level GPU-specific instruction primitives, complemented by a GPU-accelerated FAST feature detector implementing multiple corner response functions. Our approach is unique in the way it combines feature detection and non-maxima suppression, not only guaranteeing uniform feature distribution over an image but also consistently outperforming all other available implementations in terms of execution speed. The speed improvement for the embedded computer is more pronounced, as there's a higher performance penalty for memory interactions than in the case of the laptop GPU. We verified the conformity with the original FAST detector, analyzed the execution timings on two different platforms considering corner response functions, non-maxima suppression, and feature tracking on multiple numbers of features. As opposed to others, our feature tracker utilizes a feature-to-warp assignment, which speeds up tracking operations in typical VIO scenarios. Finally, we demonstrate superior speed in combining our frontend with a VIO bundle-adjustment backend, achieving a metric state estimation throughput of ∼200 frames per second with high accuracy on an embedded Jetson TX2 platform, providing a real-time, heterogeneous VIO alternative.

Table D.7: Results of our frontend combined with a VIO backend [126] achieving ∼200fps throughput on the EuRoC dataset [33]

		Average execution time					Relative translation error (RMSE)				
		MH_01	MH_02	MH_03	MH_04	MH_05	MH_01	MH_02	MH_03	MH_04	MH_05
Tegra X2	**Original**	11.64 ms	12.90 ms	12.91 ms	12.90 ms	12.85 ms	1.08 %	0.71 %	0.40 %	0.81 %	0.50 %
	Ours	4.67 ms	4.93 ms	6.41 ms	6.10 ms	5.87 ms	0.86 %	0.90 %	1.40 %	1.85 %	1.19 %
i7-6700HQ+960M	**Original**	4.11 ms	4.28 ms	4.77 ms	4.63 ms	4.64 ms	0.68 %	0.71 %	0.53 %	0.83 %	1.36 %
	Ours	1.56 ms	1.74 ms	2.54 ms	2.51 ms	2.06 ms	1.00 %	0.55 %	0.74 %	2.14 %	1.68 %

E VIMO: Simultaneous Visual Inertial Model-based Odometry and Force Estimation

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

Barza Nisar, Philipp Foehn, Davide Falanga, and Davide Scaramuzza. "VIMO: Simultaneous Visual Inertial Model-based Odometry and Force Estimation". In: *Robotics: Science and Systems (RSS)*. 2019

VIMO: Simultaneous Visual Inertial Model-based Odometry and Force Estimation

Barza Nisar, Philipp Foehn, Davide Falanga, Davide Scaramuzza

Abstract — In recent years, many approaches to Visual Inertial Odometry (VIO) have become available. However, they neither exploit the robot's dynamics and known actuation inputs, nor differentiate between desired motion due to actuation and unwanted perturbation due to external force. For many robotic applications, it is often essential to sense the external force acting on the system due to, for example, interactions, contacts, and disturbances. Adding a motion constraint to an estimator leads to a discrepancy between the model-predicted motion and the actual motion. Our approach exploits this discrepancy and resolves it by simultaneously estimating the motion and the external force. We propose a relative motion constraint combining the robot's dynamics and the external force in a preintegrated residual, resulting in a tightly-coupled, sliding-window estimator exploiting all correlations among all variables. We implement our Visual Inertial Model-based Odometry (VIMO) system into a state-of-the-art VIO approach and evaluate it against the original pipeline without motion constraints on both simulated and real-world data. The results show that our approach increases the accuracy of the estimator up to 29% compared to the original VIO, and provides external force estimates at no extra computational cost. To the best of our knowledge, this is the first approach exploiting model dynamics by jointly estimating motion and external force. Our implementation will be made available open-source.

E.1 Introduction

E.1.1 Motivation

Recent advances in robot perception have led to a number of Visual Inertial Odometry (VIO) systems becoming more robust and accessible solutions for state estimation and navigation, such as [76, 189, 123, 162, 56, 128, 48]. Although these systems work well in most conditions, they all neglect the robot's dynamics and cannot sense forces, such as contacts and interactions, and disturbances, such as wind and other environmental influences. Additionally, these approaches do not consider the fundamental distinction between the desired motion due to actuation and unwanted perturbation due to external forces. Adding the system dynamics to a VIO system (i)) allows the perception of external force acting on a robot, and (ii)) adds information to the estimation problem, resulting in increased accuracy.

Applications such as inspection, grasping, manipulation, and delivery require a robot to sense interaction or forces, which are often recovered using an estimator loosely-coupled with an odometry system, as proposed in [237, 250, 203, 145, 11, 231]. Such estimators introduce latency, computational overhead, and neglect correlation among the estimated variables and their noise characteristics. This shows the necessity for joint estimation of motion and external force in a unified approach addressing both, model and sensor noise characteristics.

On the other hand, VIO approaches on Unmanned Aerial Vehicle (UAV), rely on minimal sensor configurations, typically consisting of visual and inertial sensors suffering from additive noise. Thanks to Gaussian filtering theory [106], it is known that additional knowledge and information improves the estimation performance, especially in the presence of Gaussian noise. By adding the system dynamics to a VIO estimation problem, we effectively add information. Intuitively, this additional knowledge allows us to increase the accuracy of the odometry. However, the pure addition of a motion constraint from the system dynamics does not account for any external influences, and may lead to a motion prediction deviating from the actual motion, as depicted in Fig. E.1. Since this would degrade the estimator performance due to a wrong prior, it highlights the importance of including external force and jointly estimating all variables.

To the best of our knowledge, we present the first tightly-coupled approach exploiting the model dynamics while jointly estimating motion and external force. We derive the resulting motion constraint and formulate a dynamic residual. This residual is added to a pose-graph formulation of the VIO approach in [189] and is solved using numerical optimization. The resulting estimator demonstrates up to 29% increased accuracy and inherent ability to sense external force, opening the door to a number of possible future research topics and applications. As a call to the community, we want to raise awareness for the importance of contact-enabled robotics and the need for estimators to provide not

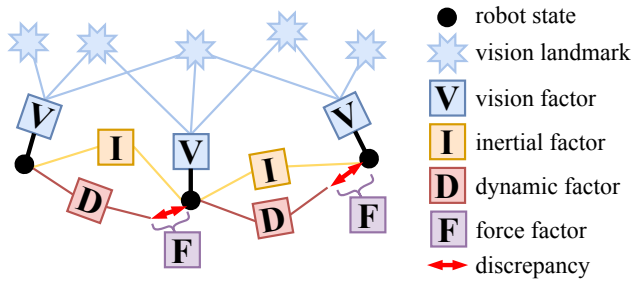


Figure E.1: Factor-graph of our VIMO approach with inertial, dynamic and force factors. The red arrows indicate the discrepancy between the dynamic and VIO factors, which is resolved by including an external force.

only odometry information, but also leverage the robot dynamics to increase accuracy and sense external forces from contacts and interaction.

E.1.2 Related Work

Previous approaches on external force estimation can be split into two groups: deterministic and probabilistic.

Deterministic Approaches

Deterministic approaches estimate external force by subtracting the collective thrust vector from the inertial measurements [237]. [250, 203] proposed a non-linear force and torque observer based on the quadrotor's dynamical model, assuming that an estimate of the robot state is available from another estimator. These deterministic approaches do not consider (i)) the thrust input noise, (ii)) the noise in the state, and (iii)) noise and unknown time-varying bias in the Inertial Measurement Unit (IMU). Hence, deterministic methods only work appropriately in practice when their inputs and outputs are carefully processed or when the signal to noise ratio of the used sensor data is very high.

Probabilistic Approaches

Realizing the drawbacks of deterministic force observers, [145] proposed an Unscented Kalman Filter (UKF) to account for the process and sensors noise and, consequently, improve the force estimate. Other similar filtering-based external force estimators include a Kalman filter [11] and UKF [231]. These methods can be classified as loosely-coupled, since they use the state estimate from a separate estimator [1, 231], and then fuse this estimate with their prediction from the UAV's dynamic model in a separate estimation step. Loosely-coupled estimators do not consider correlations among all estimated variables, which may lead to inaccuracies [123]. Moreover, the external force is estimated in an additional fusion step, which may introduce latency and extra computation cost.

Extension to Sliding Window Smoother

A widely used state estimator for UAVs is Visual Inertial Odometry (VIO) based on sliding-window smoothing [189] with IMU preintegration [74] to make the optimization problem computationally tractable in real time. IMU preintegration was first proposed in [138] and later modified in [74] to address the manifold structure of the rotation group. High-rate IMU measurements are typically integrated between image frames to form a single relative motion constraint. IMU preintegration theory reparameterizes this constraint to remove the dependence of integrated IMU measurements on previous state estimates. This avoids repeated integration when the state estimates change during each iteration of the optimization. [9] combined the idea of incorporating dynamic factors for localization of UAVs from [230] with the preintegration scheme from [74] to develop a model-based visual-inertial state estimator similar to the one proposed in our work, but without considering external forces. [9] showed that in a smoothing-based VIO pipeline, the dynamic residual in combination with the IMU residual acts as an additional source of acceleration information, which adds robustness to state estimation, especially in slow speed flights, when accelerometer measurements have low signal-to-noise ratio. While [9] chose to model air drag but ignored external forces in the dynamic model of the quadrotor, our work includes external forces and estimates them together with the robot state. An implication of not modelling external disturbances, such as wind, in model-aided state-estimation problems was studied in [1]. In the presence of wind or external forces, the estimator from [9] can tend to wrongly adjust the IMU biases due to the mismatch between sensor measurements and vehicle dynamics and therefore only works in a disturbance-free environment, as confirmed by the authors. [117] proposed to use Dynamic Differential Programming to estimate the state, parameters, and disturbances (forces) in a synthetic planar motion example, assuming perfect data association, velocity and landmark position measurement without real-world applications. Their approach is significantly simplified by modelling landmark position measurements, instead of realistic camera projection measurements. Differently from [117], our method extends an optimization-based VIO framework with motion factors to simultaneously estimate state and external force in real time on real-world data. To the best of our knowledge, there is no precedent of a tightly-coupled or smoothing-based method that jointly estimates robot states and 3-dimensional external forces.

E.1.3 Contribution

This work extends an optimization-based VIO in [123, 189, 74] with a residual term integrating the dynamic model of the quadrotor. Our main contribution is the derivation of this residual term from a motion constraint enforced by the model dynamics including external force, enabling a VIO framework to jointly estimate this force in addition to the robot state and IMU bias. Our approach works as a tightly-coupled estimator, using visual-inertial measurements, and the collective thrust input. Since current smoothing-

based VIO systems offer higher accuracy compared to filtering-based methods, we employ non-linear optimization as estimation strategy.

Inspired from IMU preintegration [74], the high-rate thrust inputs are preintegrated, resulting in dynamic factors used as residuals between consecutive camera frames. A factor graph representation of the VIO problem with dynamic factors is depicted in Fig E.1. The dynamic factors represent relative motion constraints similar to the IMU factors but with a different model and source of measurement. In our work, we exploit this redundant motion representation to estimate external force. The dynamic residual is implemented into VINS-mono [189], an open-source sliding-window monocular VIO framework. VINS-Mono was chosen because of its availability, real-time capability, and requirement for only one camera and an IMU. We show on real and simulated data that the proposed estimator compared to VINS-mono, not only increases the accuracy of the estimates (up to 29%) but also offers external force estimates without increasing the computation time. Our approach can be implemented analogously on other robots, such as fixed-wings, manipulators and mobile ground robots.

E.1.4 Structure of this paper

The model-based VIO problem is described in Sec. E.2, followed by the preintegration of the dynamic residual in Sec. E.3. We report our experiments in Sec. E.4 and the limitations in Sec. E.5. Finally the paper is concluded in Sec. E.6.

E.2 Problem Formulation

E.2.1 Notation

All coordinate frames used are depicted in Fig. E.2. The quadrotor pose is the body-fixed frame described in world frame. The IMU frame corresponds to the body frame, attached to the center of mass of the vehicle. The world frame is denoted by $[\,]^w$, the body frame by $[\,]^b$ and the camera frame by $[\,]^c$, while a hat $\hat{[\,]}$ represents noisy measurements. The robot state at the time t_k is defined as

$$\mathbf{x_k} = [\mathbf{p}^w_{b_k}, \mathbf{v}^w_{b_k}, \mathbf{q}^w_{b_k}, \mathbf{b}_{a_k}, \mathbf{b}_{\omega_k}], \qquad k \in [0, n] \tag{E.1}$$

comprised of position $\mathbf{p}^w_{b_k}$, velocity $\mathbf{v}^w_{b_k}$ and Hamilton quaternion $\mathbf{q}^w_{b_k}$ encoding the rotation of the body frame with respect to the world frame, and accelerometer and gyroscope biases $\mathbf{b}_{a_k}, \mathbf{b}_{\omega_k}$ in the IMU body frame. n is the number of the most recent keyframes in the optimization window, where the n^{th} frame is the latest frame that does not need to be a keyframe. The sliding window optimization variables are given by

$$\mathcal{X} = [l_1, \cdots, l_m, \mathbf{x}_0, \mathbf{f}^b_{e_0}, \mathbf{x}_1, \cdots, \mathbf{f}^b_{e_{n-1}}, \mathbf{x}_n] \tag{E.2}$$

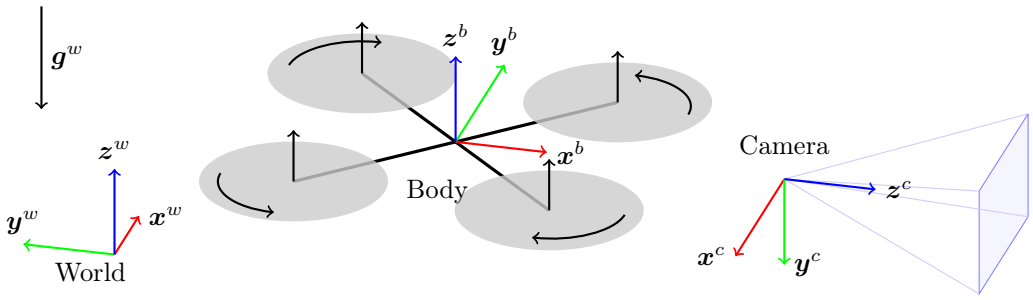


Figure E.2: Quadrotor scheme with world, body and camera frame indicated.

where m is the total number of features in the sliding window and l_i is the inverse depth of the i^{th} feature as in [189]. The total mass normalised external force $\mathbf{f}_{e_k}^b$ is expressed in body frame and experienced by the quadrotor from the time of k to $k+1$ image i.e. during $[t_k, t_{k+1})$. If the duration between consecutive image frames is small, $\mathbf{f}_{e_k}^b$ will be a good approximation of the instantaneous force experienced at t_k.

E.2.2 Dynamic Residual

To include the model dynamics and external force in a non-linear optimization, we formulate a dynamic residual $\mathbf{e}_d^k(\mathbf{x}_k, \mathbf{f}_{e_k}^b, \mathbf{x}_{k+1}, \hat{\mathbf{z}}_{b_{k+1}}^{b_k})$, with the preintegrated measurements $\hat{\mathbf{z}}_{b_{k+1}}^{b_k}$. The full non-linear optimization problem which solves for the maximum aposteriori estimate of $\mathcal{X}$ is formulated as

$$\min_{\mathcal{X}} \sum_{k=0}^{n-1} \left\| \mathbf{e}_d^k(\mathbf{x}_k, \mathbf{f}_{e_k}^b, \mathbf{x}_{k+1}, \hat{\mathbf{z}}_{b_{k+1}}^{b_k}) \right\|_{\mathbf{W}_d^k}^2 + J_{VIO}(\mathcal{X}, \hat{\mathbf{z}}_{b_{k+1}}^{b_k}) \tag{E.3}$$

where J_{VIO} contains the sum of prior residual $\mathbf{e}_p$, the visual residual $\mathbf{e}_v$ of all visible landmark reprojections, and the inertial residual $\mathbf{e}_s$ comprising of the preintegrated measurements. As proposed in [189] we summarize it into:

$$J_{VIO} = \sum_{k=0}^{n} \sum_{j \in \mathcal{J}_k} \rho \left(\left\| \mathbf{e}_v^{j,k} \right\|_{\mathbf{W}_v}^2 \right) + \sum_{k=0}^{n-1} \left\| \mathbf{e}_s^k \right\|_{\mathbf{W}_s^k}^2 + \left\| \mathbf{e}_p^k \right\|^2 . \tag{E.4}$$

$\mathcal{J}_k$ is the set of visible landmarks in frame k, while $\mathbf{e}_v^k$ is robustified by the Huber-norm $\rho(x) = \left(\sqrt{1 + (x/\delta)^2} - 1 \right) \delta^2$. The reader can refer to [189] for the derivation of J_{VIO}.

In the next section, we formulate the dynamic residual $\mathbf{e}_d^k$ as a function of the robot states and external forces at times $[t_k, t_{k+1}]$ and preintegrated thrust inputs and IMU measurements $\hat{\mathbf{z}}_{b_{k+1}}^{b_k}$. Additionally, we derive the weight $\mathbf{W}_d^k$ for the Mahalanobis norm of $\mathbf{e}_d^k$ by propagating the covariance from the measurement noise. While the formulation so far was robot-agnostic, we now focus on the quadrotor model.

E.3 Preintegration of Quadrotor Dynamics

E.3.1 Model Dynamics

In the dynamical model we consider the evolution of position and velocity of the quadrotor subject to three forces: collective rotor thrust $\mathbf{T}_t^b$, external forces $\mathbf{f}_{e_t}^b$, and gravity $\mathbf{g}^w = [0, 0, -9.81]^T \mathrm{m\,s}^{-2}$. The translational dynamics of the quadrotor is given by the following equations:

$$\dot{\mathbf{p}}_{b_t}^w = \mathbf{v}_{b_t}^w \qquad \dot{\mathbf{v}}_{b_t}^w = \mathbf{R}(\mathbf{q}_{b_t}^w)\left(\mathbf{T}_t^b + \mathbf{f}_{e_t}^b\right) + \mathbf{g}^w \tag{E.5}$$

where $\mathbf{R}(\mathbf{q}_{b_t}^w)$ is the rotation matrix corresponding to the rotation from body to world frame. Since we do not know the dynamics of external force, we assume it to be a Gaussian variable $\mathbf{f}_{e_t} = \mathcal{N}(\mathbf{0}, \sigma_f^2)$. This allows the framework to distinguish between slowly walking accelerometer biases and incidental external forces.

Preintegration of the system dynamics requires separation of the residual terms dependent on optimization variables from the terms dependent on the measurement. The rotational dynamics of the quadrotor is not considered here, since the control torques can not be separated from their dependency on the optimization variables rendering preintegration ineffective.

E.3.2 Preintegration of Dynamic Factors

In this section we derive the preintegration of the dynamic factors. The integration of (E.5) requires the evolution of rotation, which is provided by the IMU's rotation model $\dot{\boldsymbol{q}}_{b_t}^w = \frac{1}{2}\mathbf{q}_{b_t}^w \otimes [0, \boldsymbol{\omega}_t^b]^\intercal$ where $\otimes$ is the quaternion multiplication and $\boldsymbol{\omega}^b$ is the angular velocity of the body expressed in the body frame. The involved noisy measurements are the biased angular velocity $\hat{\boldsymbol{\omega}}_t^b = \boldsymbol{\omega}_t^b + \mathbf{b}_{\omega_t} + \boldsymbol{\eta}_\omega$ from the IMU and the collective rotor thrust $\hat{\mathbf{T}}_t^b = \mathbf{T}_t^b + \boldsymbol{\eta}_T$. As in [189], the gyroscope noise is considered as Gaussian $\boldsymbol{\eta}_\omega \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_\omega^2)$ and its bias as random walk $\dot{\mathbf{b}}_{\omega_t} = \boldsymbol{\eta}_{b_\omega}$ with $\boldsymbol{\eta}_{b_\omega} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_{b_\omega}^2)$. Since neither the magnitude nor the direction of the actual thrust is known precisely, we assume Gaussian noise in the thrust as $\boldsymbol{\eta}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_T^2)$. The vehicle state can be propagated between two frames over time interval $\Delta t_k = t_{k+1} - t_k$ by integrating the thrust and

gyroscope measurements:

$$\begin{aligned}
\mathbf{p}_{b_{k+1}}^{w} &= \mathbf{p}_{b_k}^{w} + \mathbf{v}_{b_k}^{w}\Delta t_k + \frac{1}{2}\mathbf{g}^{w}\Delta t_k^2 \\
&\quad + \int\int_{t_k}^{t_{k+1}} \mathbf{R}_{b_\tau}^{w}\left(\hat{\mathbf{T}}_{\tau}^{b} + \mathbf{f}_{e_\tau}^{b} - \boldsymbol{\eta}_T\right) d\tau^2 \\
\mathbf{v}_{b_{k+1}}^{w} &= \mathbf{v}_{b_k}^{w} + \mathbf{g}^{w}\Delta t_k + \int_{t_k}^{t_{k+1}} \mathbf{R}_{b_\tau}^{w}\left(\hat{\mathbf{T}}_{\tau}^{b} + \mathbf{f}_{e_\tau}^{b} - \boldsymbol{\eta}_T\right) d\tau \\
\mathbf{q}_{b_{k+1}}^{w} &= \mathbf{q}_{b_k}^{w} \otimes \int_{t_k}^{t_{k+1}} \frac{1}{2}\boldsymbol{\Omega}\left(\hat{\boldsymbol{\omega}}_{\tau}^{b} - \mathbf{b}_{\omega_\tau} - \boldsymbol{\eta}_\omega\right)\mathbf{q}_{b_\tau}^{b_k}\, d\tau
\end{aligned} \tag{E.6}$$

$$\text{where: } \boldsymbol{\Omega}(\omega) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{E.7}$$

To make the integration of the measurements independent of the states at frame k, we group the terms containing measurements in $\hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k}$, $\hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k}$, $\hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k}$, and change the reference frame from world to body frame as done in IMU preintegration [74]:

$$\begin{aligned}
\hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} &= \int\int_{t_k}^{t_{k+1}} \mathbf{R}_{b_\tau}^{b_k}\left(\hat{\mathbf{T}}_{\tau}^{b} - \boldsymbol{\eta}_T\right) d\tau^2 \\
\hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} &= \int_{t_k}^{t_{k+1}} \mathbf{R}_{b_\tau}^{b_k}\left(\hat{\mathbf{T}}_{\tau}^{b} - \boldsymbol{\eta}_T\right) d\tau \\
\hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k} &= \int_{t_k}^{t_{k+1}} \frac{1}{2}\boldsymbol{\Omega}\left(\hat{\boldsymbol{\omega}}_{\tau}^{b} - \mathbf{b}_{\omega_\tau} - \boldsymbol{\eta}_\omega\right)\hat{\gamma}_{b_\tau}^{b_k}\, d\tau.
\end{aligned} \tag{E.8}$$

We then derive the prediction of the terms in (E.8) from the model equations in (E.6) to form the factors

$$\begin{aligned}
\boldsymbol{\alpha}_{b_{k+1}}^{b_k} &= \mathbf{R}_{w}^{b_k}\left(\mathbf{p}_{b_{k+1}}^{w} - \mathbf{p}_{b_k}^{w} - \mathbf{v}_{b_k}^{w}\Delta t_k - \frac{1}{2}\mathbf{g}^{w}\Delta t_k^2\right) \\
&\quad - \frac{1}{2}\mathbf{f}_{e_k}^{b}\Delta t_k^2 \\
\boldsymbol{\beta}_{b_{k+1}}^{b_k} &= \mathbf{R}_{w}^{b_k}\left(\mathbf{v}_{b_{k+1}}^{w} - \mathbf{v}_{b_k}^{w} - \mathbf{g}^{w}\Delta t_k\right) - \mathbf{f}_{e_k}^{b}\Delta t_k \\
\boldsymbol{\gamma}_{b_{k+1}}^{b_k} &= \mathbf{q}_{w}^{b_k} \otimes \mathbf{q}_{b_{k+1}}^{w}.
\end{aligned} \tag{E.9}$$

E.3.3 Dynamic Residual

Now we can combine (E.8) and (E.9) into the dynamic residual between frames b_k and b_{k+1}, which also includes the zero-mean prior on external forces.

$$\mathbf{e}_d^k = \begin{bmatrix} \boldsymbol{\alpha}_{b_{k+1}}^{b_k} - \hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} \\ \boldsymbol{\beta}_{b_{k+1}}^{b_k} - \hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} \\ \mathbf{f}_{e_k}^b \end{bmatrix} \qquad \mathbf{W}_d^k = \begin{bmatrix} \mathbf{P}_{b_{k+1}[0:5]}^{b_k\,-1} & \mathbf{0} \\ \mathbf{0} & w_f \mathbf{I} \end{bmatrix} \tag{E.10}$$

Finally, the weight of the residual can be formulated by the inverse of the covariance in $\hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k}$ and $\hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k}$ extracted from $\mathbf{P}_{b_{k+1}}^{b_k}$ (derived in Sec. E.3.4) and a diagonal weight w_f for the external force zero-mean prior.

It is important to note that these preintegrated terms still depend on the gyroscope bias. This means that each time an optimization iteration changes the bias estimate slightly, we need to repropagate the measurements. To avoid this computationally expensive repropagation, we will adopt the solution proposed in [74], and explained in Sec. E.3.5.

E.3.4 Propagation Algorithm

We start the propagation from an initial condition of $\hat{\boldsymbol{\alpha}}_{b_k}^{b_k} = \hat{\boldsymbol{\beta}}_{b_k}^{b_k} = \mathbf{0}_{3\times1}$ and $\hat{\boldsymbol{\gamma}}_{b_k}^{b_k} = [1, \mathbf{0}_{3\times1}]$. The Euler integration over timestep δt_i is computed by

$$\hat{\boldsymbol{\alpha}}_{i+1}^{b_k} = \hat{\boldsymbol{\alpha}}_i^{b_k} + \hat{\boldsymbol{\beta}}_i^{b_k}\delta t_i + \frac{1}{2}\mathbf{R}(\hat{\boldsymbol{\gamma}}_i^{b_k})\mathbf{T}_i^b \delta t_i^2 \tag{E.11}$$

$$\hat{\boldsymbol{\beta}}_{i+1}^{b_k} = \hat{\boldsymbol{\beta}}_i^{b_k} + \mathbf{R}(\hat{\boldsymbol{\gamma}}_i^{b_k})\mathbf{T}_i^b \delta t_i \tag{E.12}$$

$$\hat{\boldsymbol{\gamma}}_{i+1}^{b_k} = \hat{\boldsymbol{\gamma}}_i^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}(\boldsymbol{\omega}_{m_i} - \bar{\mathbf{b}}_{\omega k})\delta t_i \end{bmatrix} \tag{E.13}$$

To achieve optimal linearization accuracy, the algorithm is run at the rate of the fastest available measurement, typically the IMU rate. The covariance $\mathbf{P}_{b_{k+1}}^{b_k}$ is derived by linearizing the error $\delta\mathbf{z} = [\delta\boldsymbol{\alpha}, \delta\boldsymbol{\beta}, \delta\boldsymbol{\theta}, \delta\mathbf{b}_\omega]^\intercal$ and noise $\boldsymbol{\eta} = [\boldsymbol{\eta}_T, \boldsymbol{\eta}_\omega, \boldsymbol{\eta}_{b_\omega}]^\intercal$ dynamics between integration steps as

$$\boldsymbol{z}_{i+1}^{b_k} = \mathbf{A}_i \boldsymbol{z}_i^{b_k} + \mathbf{G}_i \begin{bmatrix} \boldsymbol{\eta}_T \\ \boldsymbol{\eta}_\omega \\ \boldsymbol{\eta}_{b_\omega} \end{bmatrix} \qquad \boldsymbol{\gamma}_t^{b_k} \approx \hat{\boldsymbol{\gamma}}_t^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta}_t^{b_k} \end{bmatrix} \tag{E.14}$$

where $\delta\boldsymbol{\theta}$ is the minimal perturbation around the mean of $\boldsymbol{\gamma}$. Finally, $\mathbf{P}_{b_{k+1}}^{b_k}$ is linearly propagated from $\mathbf{P}_{b_k}^{b_k} = \mathbf{0}$ by

$$\mathbf{P}_{i+1}^{b_k} = \mathbf{A}_i \mathbf{P}_i^{b_k} \mathbf{A}_i^T + \mathbf{G}_i \mathbf{Q} \mathbf{G}_i^T \tag{E.15}$$

with the linearization $\boldsymbol{A}_i = \frac{\partial \boldsymbol{z}_{i+1}^{bk}}{\partial \boldsymbol{z}_i^{bk}}$ and $\boldsymbol{G}_i = \frac{\partial \boldsymbol{z}_{i+1}^{bk}}{\partial \boldsymbol{\eta}}$.

E.3.5 Bias Correction

The first-order Jacobian matrix $\mathbf{J}_{b_{k+1}}$ of $\mathbf{z}_{b_{k+1}}^{b_k}$ with respect to $\mathbf{z}_{b_k}^{b_k}$ can be computed recursively by $\mathbf{J}_{i+1} = \mathbf{A}_i \mathbf{J}_i$ starting from the initial Jacobian of $\mathbf{J}_{b_k} = \mathbf{I}$. The preintegrated terms can then be corrected by their first order approximation with respect to the change in gyroscope bias $\delta \mathbf{b}_{\omega_k} = \mathbf{b}_{\omega_k} - \bar{\mathbf{b}}_{\omega_k}$ from the initial estimate $\bar{\mathbf{b}}_{\omega_k}$ as follows:

$$\begin{aligned} \hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} &\leftarrow \hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} + \mathbf{J}_{b_\omega}^{\alpha} \delta \mathbf{b}_{\omega_k} & \mathbf{J}_{b_\omega}^{\alpha} &= \frac{\partial \boldsymbol{\alpha}_{b_{k+1}}^{b_k}}{\partial \mathbf{b}_{\omega_k}} \\ \hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} &\leftarrow \hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} + \mathbf{J}_{b_\omega}^{\beta} \delta \mathbf{b}_{\omega_k} & \mathbf{J}_{b_\omega}^{\beta} &= \frac{\partial \boldsymbol{\beta}_{b_{k+1}}^{b_k}}{\partial \mathbf{b}_{\omega_k}}. \end{aligned} \tag{E.16}$$

E.3.6 Marginalization

We adapt the marginalization strategy proposed in [189], such that when the second last frame in the window is a keyframe, we marginalize out the oldest keyframe's state and external force $\mathbf{f_{e_0}}$. The corresponding visual, inertial, and thrust measurements of the marginalized states are converted into a prior. If the second last frame is not a keyframe, its state, external force and corresponding visual measurements are dropped, while the preintegrated IMU and thrust measurements are kept and continued to be preintegrated till the last frame.

E.4 Experiments

We perform 3 types of experiments: E.4.1: simulation based experiments; E.4.2: evaluation on the Blackbird dataset [8] with real pose, inertial, and rotor speed measurements but synthetic camera frames; E.4.3 real-world experiments.

E.4.1 Simulation

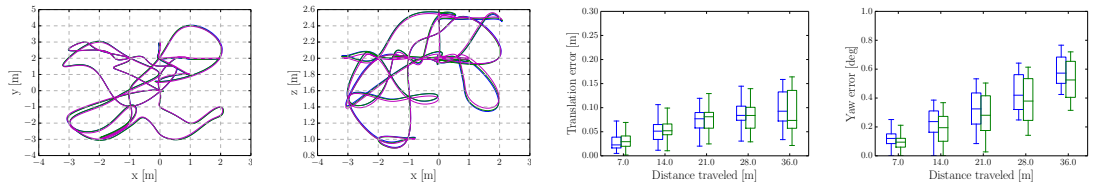
Experiment Setup

To generate repeatable data in a fully controlled environment, we used the RotorS simulator from [80], a Micro-Aerial Vehicle Simulator using Gazebo in ROS. We used a forward looking camera with 752×480 image resolution. The base simulation vehicle was *Hummingbird* from [80] according to which the onboard IMU was corrupted with noise of $\sigma_\omega = 0.004\text{rad/s}\sqrt{\text{Hz}}$ for the gyroscope, $\sigma_a = 0.1\text{m/s}^2\sqrt{\text{Hz}}$ for the accelerometer, and a bias random walk of $\sigma_{b_\omega} = 0.000038\text{rad/s}^2\sqrt{\text{Hz}}$ for the gyroscope, and $\sigma_{b_a} = 0.00004$

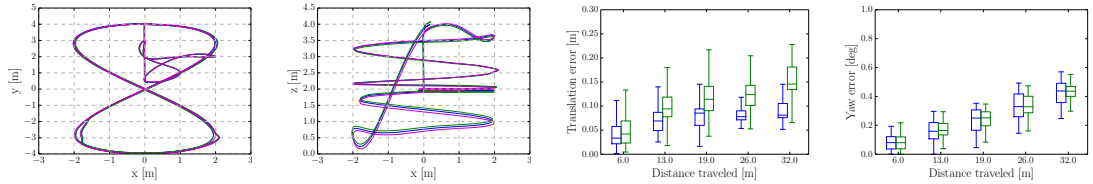
$[\mathrm{m/s^3\sqrt{Hz}}]$ for the accelerometer. The tuning parameters $\boldsymbol{\sigma}_T$ and w_f were hand-tuned and then kept the same across all of the experiments. The dynamic residual was implemented in VINS-Mono with a maximum number of 150 features tracked per frame. For a fair comparison, no loop closure was applied. The estimator is run on a 2.5 GHz Intel Core i7 CPU. VINS-Mono processes frames and provides estimates at 10 Hz, with IMU measurements sampled at 900 Hz, thrust inputs at 150 Hz, and camera images at 40 Hz. An external force is applied programmatically in the simulation, therefore its ground truth is known. We acquired simulated datasets for two trajectory shapes: trajectory 1 is 73.7 m long and is generated by arbitrarily choosing waypoints (Figs. E.3a and E.3b); trajectory 2 is helical eight (Figs. E.3e and E.3f) given by formulation $\mathbf{p}(\theta) = [l_x \sin 2\theta, l_y \cos\theta, \frac{h}{2\pi}(\sin\theta - \theta)]$ with $l_x = 2\,\mathrm{m}$ $l_y = 4\,\mathrm{m}$ and height $h = 3.2\,\mathrm{m}$. In the first set of experiments, the quadrotor flies undisturbed at speeds of $[1, 2, 2.5, 4, 5]\mathrm{m\,s^{-1}}$, while in the second set external forces act on the vehicle flying at $[1, 2, 2.5]\mathrm{m\,s^{-1}}$. In all the experiments, the reference heading was set to sinusoidally change with a magnitude of 30°. We first compare the performance of our approach (VIMO) against VINS-Mono in terms of accuracy and computation times. Finally, we compare the quality of the external force estimate against the estimate obtained from a naive approach.

Comparison with VINS-Mono

Fig. E.3 shows plots comparing simulation performance of VINS-Mono with VIMO on the two trajectory shapes flown at 2.5 m/s top speed and disturbed with external forces. This scenario represents the worst performance of VIMO in comparison with VINS-Mono on trajectory 1 and an average performance for trajectory 2, as visible from Tab. E.1. The plots were generated and the absolute and relative errors were computed using the open source trajectory evaluation toolbox for VIO pipelines [256]. For all the experiments, we align all the estimated states to the ground truth using *posyaw* trajectory alignment method of the toolbox. The top and side view of the estimated trajectories by VINS-Mono and VIMO almost overlap and are very close to the ground truth. For this worst-case scenario, the relative translation error for VIMO is less than or similar to the error for VINS-Mono, while the relative yaw errors for VIMO is slightly higher than VINS-Mono. We report all measured RMSE and computation time for VINS-Mono and VIMO in Table E.1, together with the percentage decrease in RMSE of VIMO compared to VINS-mono. The maximum increase in accuracy is $\sim 40\%$, experienced at a speed of 1 m/s in random trajectory, without external forces, while one outlying experiment (trajectory 1, with forces at $2.5\,\mathrm{m\,s^{-1}}$) showed a decrease of accuracy. Overall, we achieve a decrease in translational RMSE of $\sim 15\%$, and a decrease in rotational RMSE $\sim 25\%$ in the simulated experiments. In general, the addition of dynamic residuals excels especially in scenarios of low signal-to-noise ratio in the IMU data, which occur at low accelerations. While we could tune the parameters $\boldsymbol{\sigma}_T$ and w_f to increase the accuracy of individual experiments, we wanted to fairly evaluate our estimator's performance without tuning between scenarios to accurately represent real-world applications. In addition to increasing the accuracy,



(a) Traj. 1: Top View **(b)** Traj. 1: Side View **(c)** Traj. 1: Pos. Error **(d)** Traj. 1: Yaw Error



(e) Traj. 2: Top View **(f)** Traj. 2: Side View **(g)** Traj. 2: Pos. Error **(h)** Traj. 2: Yaw Error

Figure E.3: Comparison between VINS — (green), VIMO — (ours, blue) and ground truth — (purple) on a random trajectory (top) and a helical eight trajectory (bottom) at $2.5\,\mathrm{m\,s^{-1}}$ with external forces applied. This configuration depicts the worst performance of VIMO compared with VINS-Mono. The two left columns show the estimated trajectories aligned with the ground truth. The two right columns summarize the relative translation and yaw error statistics over trajectory segments. Boxes indicate the middle two quartiles while whiskers denote upper and lower quartiles and the center line indicates the median.

it can be observed in Tab. E.1 that our approach does not increase the average solving time, but keeps it nearly equal to VINS-Mono.

Evaluation of External Force Estimate

In this section we compare VIMO's external force estimate against the estimate obtained from a naive approach and the ground truth. We compute a naive deterministic estimate as $\tilde{\mathbf{f}}_{e_t} = \hat{\mathbf{a}}_t^b - \hat{\mathbf{T}}_t^b$ by simply subtracting the mass normalised thrust $\hat{\mathbf{T}}_t^b$ from the accelerometer measurements $\hat{\mathbf{a}}_t^b$. Fig E.4 shows plots of force estimates obtained for the different

Table E.1: Comparison between performance of VINS and VIMO.

		trans. RMSE (m)			rot. RMSE (deg)			avg solve time (ms)		max solve time (ms)	
top speed (m/s)		VINS	VIMO	% decrease	VINS	VIMO	% decrease	VINS	VIMO	VINS	VIMO
Trajectory 1: 73.7 m without external forces	1.0	0.066	**0.039**	40.9	1.40	**0.57**	59.3	42.0	40.9	52.1	54.7
	2.0	0.093	**0.073**	21.5	0.69	**0.64**	7.2	39.9	39.9	61.8	63.3
	2.5	0.085	**0.076**	10.6	0.60	**0.56**	6.7	38.5	38.7	50.	49.7
	4.0	0.038	**0.033**	13.2	0.49	**0.36**	26.5	37.9	38.0	49.1	50.5
	5.0	0.068	**0.062**	8.8	0.66	**0.47**	28.8	38.3	38.3	51.1	53.8
Trajectory 1: 73.7 m with external forces	1.0	0.105	**0.089**	15.2	1.81	**0.75**	58.6	42.0	40.7	52.2	54.2
	2.0	0.057	**0.051**	10.5	0.75	**0.61**	18.7	39.6	39.7	50.8	55.5
	2.5	**0.055**	0.059	- 7.3	0.71	**0.69**	2.8	39.3	38.8	59.7	51.0
Trajectory 2: 65.8 m without external forces	1.0	0.228	**0.189**	17.1	1.45	**1.12**	22.8	40.7	40.9	54.0	60.7
	2.0	0.147	**0.143**	2.7	0.67	**0.42**	37.3	39.7	39.1	52.6	51.8
	2.5	0.203	**0.158**	22.2	0.74	**0.48**	35.1	39.3	38.5	77.2	54.4
	4.0	0.085	**0.068**	20.0	0.81	**0.65**	19.8	38.3	38.0	50.5	57.4
	5.0	0.073	**0.061**	16.4	0.72	**0.48**	33.3	38.2	38.0	51.8	61.6
Trajectory 2: 65.8 m with external forces	1.0	0.162	**0.154**	4.9	1.29	**1.00**	22.5	40.8	40.9	55.6	61.8
	2.0	0.157	**0.136**	13.4	0.74	**0.62**	16.2	40.2	38.8	84.5	58.7
	2.5	0.094	**0.061**	35.1	0.64	**0.52**	18.8	39.5	38.5	52.1	61.7

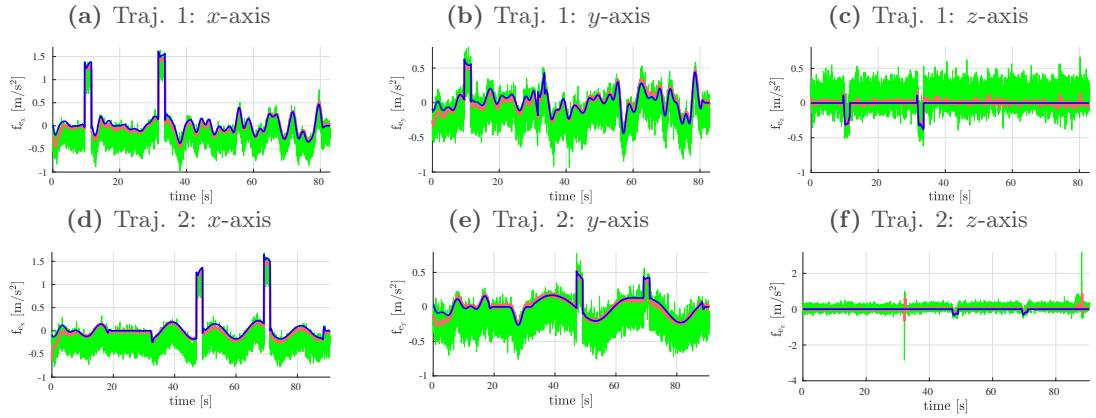


Figure E.4: Comparison between external force estimates from VIMO — (pink), the naive approach — (green) and calculated ground truth — (blue) on the random trajectory (top, a - c) and the helical-eight trajectory (bottom, d - f). The external force estimate consists of air drag in body x- and y-axis and 2 external forces applied at $t = 10\,\text{s}$ and $t = 32\,\text{s}$ for the top experiment and $t = 47\,\text{s}$ and $t = 68\,\text{s}$ for the bottom experiment.

trajectory shapes flown at 2.5 m/s top speed. In both the experiments, we disturb the quadrotor at its center of mass by 2 external forces for 2 seconds each, one after the other, in all three body axes. The ground truth of the external force is computed as a sum of mass normalised external disturbance measured by the force sensor and the drag force. Since RotorS does not provide ground truth of the drag force, we approximate it offline using the linear drag model $-diag([d_x, d_y, d_z])R_b^{wT}v_b^w$ [58], and the ground truth rotation, velocity and mass normalized drag coefficients d_x, d_y, d_z from the simulator. We assume $d_z = 0$ because the drag in body z axis is very small. From the plots it is evident that the naive deterministic estimate needs additional filtering and bias removal steps, whereas our estimator implicitly takes into account the noise characteristics of the IMU, its bias, the noise in the state estimates, and the noise of the commanded thrust. Hence, our estimate lies closer to the computed ground truth force. The plots also show that the force estimates take time to converge at the beginning, as long as the IMU bias estimate is not converged (first $\sim 8 - 10$s). One peculiarity visible in Fig E.4(f) are the peaks in the estimate at $t = 32\,\text{s}$ and $t = 88\,\text{s}$, which are not visible in the ground truth. This is the result of a high change in commanded thrust, while the actual thrust has latency introduced by the motors and speed controllers.

E.4.2 Blackbird Dataset

Experiment Setup

Additionally, we evaluate the performance of VIMO and VINS-Mono on the Blackbird dataset from [8], which uses a motion capture system for closed-loop control of a UAV along fast trajectories, while rendering photorealistic images of synthetic scenes synchronized

Table E.2: Blackbird Dataset Evaluation

	trans. RMSE (m)			rot. RMSE (deg)		
	VINS	VIMO	%decrease	VINS	VIMO	%decrease
star 1m/s	0.102	**0.088**	13.7	**0.46**	0.48	-4.3
star 2m/s	0.133	**0.082**	38.2	0.67	**0.60**	10.5
star 3m/s	0.235	**0.183**	22.1	0.96	**0.88**	8.7
picasso 1m/s	0.097	**0.055**	43.5	**0.67**	0.77	-14.9
picasso 2m/s	0.043	**0.040**	7.8	0.46	**0.43**	9.1
picasso 3m/s	0.045	**0.043**	2.9	0.34	**0.30**	14.6
picasso 4m/s	0.056	**0.049**	11.9	0.67	**0.53**	21.7

with onboard IMU and rotor thrust measurements. We use the two sequences *star* and *picasso* at speeds from 1 to $4\,\mathrm{m\,s^{-1}}$ with the camera forward-facing for the *star* sequence and at a fixed yaw for the *picasso* sequence. Since this dataset does not include any applied external forces, we only evaluate pose estimation as direct comparison on the public available dataset for reproducibility. Since the dataset contains IMU measurements at 100 Hz, we downsample the images, which are available at a faster rate of 120 Hz, to 30 Hz to allow proper IMU preintegration. We use the rotor thrust measurements at the provided $\sim 190\,\mathrm{Hz}$.

Evaluation

Also for the Blackbird dataset [8], we use the trajectory alignment toolbox from [256] with the *posyaw* alignment. Even though this dataset does not include sequences with applied (and measured) external forces, we could measure a slight performance increase as shown in Table E.2. Different from most available datasets (Sec. E.5.2), the Blackbird dataset includes the rotor speed measurements which we exploit through the known system dynamics and achieve superior accuracy in nearly all test sequences. One can observe that in the *star* trajectory the translational errors are generally higher and the highest tested speed is $3\,\mathrm{m\,s^{-1}}$. This is because of the high yaw rate and the resulting high optical flow, rendering the estimation problem more difficult, and causing the system to fail at $4\,\mathrm{m\,s^{-1}}$ without significant retuning.

E.4.3 Real-World Validation

Experiment Setup

To fully validate our approach, we provide a real-world experiment where we record data consisting of camera frames, IMU data, commanded collective thrust, force measurements and quadrotor state ground truth. For the quadrotor, we used an ARM-based platform

with a monochrome global-shutter VGA resolution camera at 30 Hz synchronized with an IMU providing inertial data at 400 Hz, based on the Qualcomm Snapdragon Flight as depicted in Fig. E.5a. For the experiment we used our inhouse-developed flight stack. To provide ground truth data, we employed an OptiTrack motion capture system. As a force ground truth, we used an ATI Mini40-SI-20-1 force and torque sensor (Fig, E.5b) also tracked in our motion capture system to recover the direction of the force. We evaluate disturbance-free figure-eight trajectory flight with $l_x = 2.25\,\text{m}$, $l_y = 1.5\,\text{m}$, $h = 0\,\text{m}$, and disturbing the vehicle in hover with $\sim 3\,\text{N}$ by pushing it with the force-measurement pole.

Evaluation

As a simple validation of our approach, we depict the top view on the position estimate of VINS-Mono, VIMO and ground truth in Fig. E.6a, indicating a very similar performance of both approaches. We evaluate a translational RMSE of 0.106,9 m for VIMO and 0.149,7 m for VINS-Mono, corresponding to 29% error reduction, while the rotational RMSE is at 4.95° for VIMO and 5.15° for VINS-Mono, corresponding to 4% error reduction. Fig. E.6b reports the error statistics on the real-world data computed with the trajectory evaluation toolbox [256]. Additionally, we disturbed the vehicle with $\sim 3\,\text{N}$ while in hover, as shown in Fig. E.6c. The estimate is accurate, while noisy due to high vibrations on the used vehicle.

E.5 Discussion

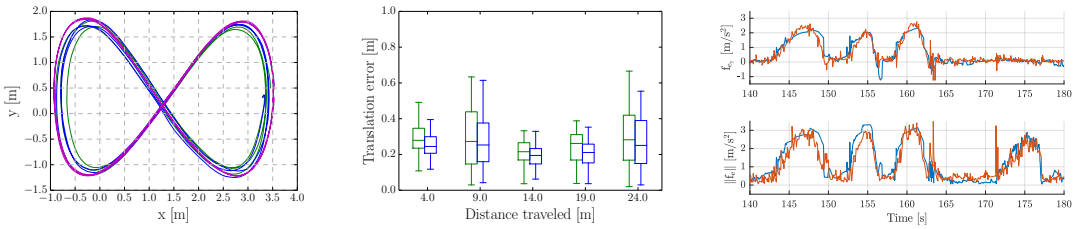
E.5.1 Limitations due to Measurement Modality

While our approach offers the benefits of improving state estimates and estimating external force, it also comes with two limitations due to its measurement modality.

First, we consider the acceleration measurement $\boldsymbol{a}_k^b$ in body frame $\boldsymbol{a}_k^b - \boldsymbol{b}_{a_k}^b = \boldsymbol{T}_k^b + \boldsymbol{f}_{e_k}^b = \boldsymbol{T}_k^b + \bar{\boldsymbol{f}}_{e_k}^b + \boldsymbol{f}_{d_k}^b$ where we have separated $\boldsymbol{f}_{e_k}^b$ from (E.5) into the true external force $\bar{\boldsymbol{f}}_{e_k}^b$ and the aerodynamic drag $\boldsymbol{f}_{d_k}^b$ force. While $\boldsymbol{a}_k^b$ and $\boldsymbol{T}_k^b$ are measured quantities, all other

(a) Snapdragon Flight Quadrotor **(b)** Force sensor ATI Mini40

Figure E.5: Experimental quadrotor platform a and an force/torque sensor b.



(a) Real-world x-, y- position comparison between VINS — (green), VIMO — (ours, blue) and ground truth — (purple) in top view.

(b) Translational errors over trajectory segments of VINS — (green), VIMO — (ours, blue) on real-world data as statistical box plot.

(c) External force estimate f_{e_y} (top) and $\|\mathbf{f}_e\|$ (bottom) on real-world data compared between VIMO — and a force sensor — with a disturbance of $\sim 3\,\mathrm{N}$ magnitude.

Figure E.6: Real-world experiments flying a figure 8 trajectory at $1.5\,\mathrm{m\,s^{-1}}$ depicted in top view x,y-plot (Fig. a) and statistical box plots (Fig. b). Fig. c shows the force estimate and ground-truth (obtained with a force sensor) of a disturbance of $\sim 2\,\mathrm{N}$.

quantities have to be estimated. Due to the additive nature of external ($\bar{\boldsymbol{f}}^b_{e_k}$) and drag ($\boldsymbol{f}^b_{d_k}$) force, one can only estimate the sum of both ($\boldsymbol{f}_{e_k}$, as done in this paper) if one does not add any additional assumption or model the aerodynamic drag. Furthermore, the same additive nature introduces an ambiguity between external force (i.e. summed $\boldsymbol{f}^b_{e_k}$) and the bias $\boldsymbol{b}^b_{a_k}$. But contrary to force and drag, summed external force and bias can be discriminated by their different dynamics, implemented as an additional prior. Due to the nature of IMU bias, we have to assume a random walk prior by $\dot{\boldsymbol{b}}_{a_k} = \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}^2_{b_a})$ with $\mathcal{N}$ as the Gaussian distribution. In contrast, we assume the external forces to be zero-mean Gaussian (E.10), since we are mainly interested in detecting incidental changes in the force. Any constant component in the external force will be estimated as accelerometer bias, since the bias is the only estimated variable without cost on its magnitude, effectively forming a low-pass filter. Further evaluations of the observability of visual-inertial localization can be found in [96]. Finally, we use commanded thrust in the dynamic model whereas the accelerometer detects acceleration due to the actual rotor thrust. Therefore, our estimator comprehends the difference between commanded and actual thrust, if large enough, as external force. This is observed as mentioned before in Sec. E.4.1 as peaks in Fig E.4(f), indicating that VIMO also has the capability to detect model inaccuracy as external force. This difference between commanded and actual rotor thrust could be mitigated by using advanced motor speed controllers with feedback on the actual rotor speed or by modelling the motor dynamics.

E.5.2 Other Datasets

Several existing UAV visual-inertial datasets, such as EuRoC MAV [33], UPenn fast flight [261], Zurich Urban MAV [142], have been used extensively for evaluating the performance of VIO. Although these datasets include synchronized camera and IMU data with accurate ground truth, we could not use them to evaluate our approach since they do not provide rotor speed measurements or commanded thrust.

E.6 Conclusion

This paper extends a visual inertial estimator by adding a motion constraint derived from the dynamic model including external forces. The resulting tightly-coupled system is shown to accurately estimate vehicle's motion, IMU biases, and external forces, from visual and inertial measurements and commanded thrust inputs. Thereby, our approach enables differentiation between actuation and disturbance by the detected external forces. Inspired from IMU preintegration, the high-rate collective rotor thrust is preintegrated into relative motion constraints, implemented as residuals into an existing VIO pipeline (VINS-Mono). Synthetic and real-world experiments, conducted in the presence of external disturbances, illustrate that, compared to VINS-Mono, our estimator not only improves odometry accuracy up to 29% on real-world data, but also estimates time-varying external forces without increasing the computation time. Our unified state and force estimator enables a robot to sense motion and external forces, opening the door to a number of possible future research works and applications. As a call to the community, we want to raise awareness for the importance of contact-enabled robotics and the need for estimators to provide not only odometry information, but also leverage the robot dynamics to increase accuracy and sense external forces from contacts and interaction.

F 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. URL: 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 F.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.

F.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 [154] 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 [156], 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.

Recent work has shown that deep networks can provide drones with robust perception capabilities and facilitate safe navigation even in dynamic environments [111, 103]. 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.

F.2 Related Work

Traditional approaches to autonomous MAV navigation build on visual inertial odometry (VIO) [76, 25, 123, 241] or simultaneous localization and mapping (SLAM) [163, 211], which are used to provide a pose estimate of the drone relative to an internal metric map [128, 64]. While these methods can be used to perform visual teach and repeat [64], they

are not concerned with trajectory generation [148, 160]. 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) [114, 87, 134] to direct regression of control signals [161]. A different line of work combines network predictions with model predictive control by regressing the cost function from a single image [54].

In the context of drone racing, Kaufmann et al. [111] proposed an intermediate representation in the form of a goal direction and desired speed. The learned policy imitates an optimal trajectory [148] 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. [103] 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 [111, 103]. 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.

F.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.

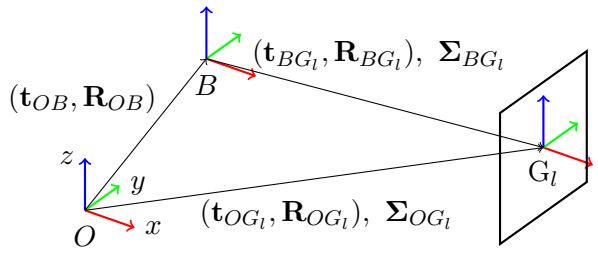


Figure F.2: Relation of odometry O, body B, and gate frame G_l.

F.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 $l \in \{1, \ldots, N_l\}$ and N_l is the number of gate. A schematic overview of the relation between coordinate frames is shown in Figure F.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 $\tilde{\mathbf{\Sigma}}_{BG_l,pol} = \text{diag}(\tilde{\boldsymbol{\sigma}}^2_{BG_l,pol})$ 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 $\hat{\mathbf{\Sigma}}_{OG_l} = \text{cov}\left(\hat{\mathbf{t}}_{OG_l}, \hat{\mathbf{R}}_{OG_l}\right)$ in the odometry frame. This has the advantage that gate poses can be updated independently of each other.

F.3.2 Perception System

Architecture

The deep network takes as input a 320×240 RGB image and regresses both the mean $\tilde{\mathbf{z}}_{BG_l,pol} = [\tilde{r}, \tilde{\theta}, \tilde{\psi}, \tilde{\phi}]^\top \in \mathbb{R}^4$ and the variance $\tilde{\boldsymbol{\sigma}}^2_{BG_l,pol} \in \mathbb{R}^4$ 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.

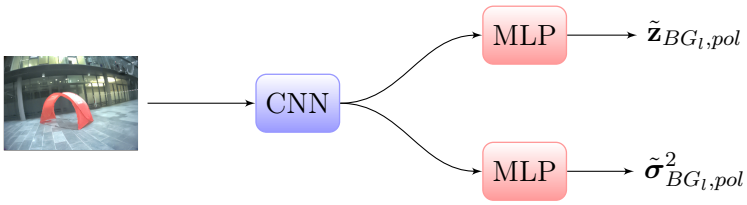


Figure F.3: Schematic illustration of the network architecture. Image features are extracted by a CNN [134] and passed to two separate MLPs to regress $\tilde{\mathbf{z}}_{BG_l,pol}$ and $\tilde{\boldsymbol{\sigma}}^2_{BG_l,pol}$, respectively.

The overall structure of the deep network is shown in Figure F.3. First, the input image is processed by a Convolutional Neural Network (CNN), based on the shallow DroNet architecture [134]. The extracted features are then processed by two separate multilayer perceptrons (MLPs) that estimate the mean $\tilde{\mathbf{z}}_{BG_l,pol}$ and the variance $\tilde{\boldsymbol{\sigma}}^2_{BG_l,pol}$ of a multivariate normal distribution, respectively. A similar network architecture for mean-variance estimation was proposed in [175].

Training Procedure

We train the network in two stages.

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

$$\{\boldsymbol{\theta}^*_{\text{CNN}}, \boldsymbol{\theta}^*_{\tilde{\mathbf{z}}_i}\} = \arg\min_{\boldsymbol{\theta}_{\text{CNN}}, \boldsymbol{\theta}_{\tilde{\mathbf{z}}_i}} \sum_{i=1}^{N} ||\mathbf{y}_i - \tilde{\mathbf{z}}_i||_2^2, \tag{F.1}$$

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 $\boldsymbol{\theta}_{\sigma^2}$ of the subnetwork MLP_{σ^2} are trained, while keeping the other weights fixed. We minimize the loss function proposed by [175], which amounts to the negative log-likelihood of a multivariate normal distribution with uncorrelated covariance:

$$-\log p\left(\mathbf{y} \mid \tilde{\mathbf{z}}_i, \tilde{\boldsymbol{\sigma}}^2\right) \propto \sum_{j=1}^{4} \log \tilde{\sigma}_j^2 + \frac{(y_j - \tilde{z}_j)^2}{\tilde{\sigma}_j^2}. \tag{F.2}$$

Our use of mean-variance estimation is motivated by studies that have shown that it is a computationally efficient way to obtain uncertainty estimates [113].

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. [111], the perception system is never trained on data from tracks and environments it is later deployed in.

F.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 $l = 1...N_l$ 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

$$ {}_{G_l}\hat{\mathbf{t}}_{G_lB,x} \geq 0. \tag{F.3}$$

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

$\tilde{\boldsymbol{\sigma}}^2_{BG,pol}$. The transformation into the Cartesian representation $\tilde{\mathbf{z}}_{BG}$ leads to

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

$$\tilde{\boldsymbol{\Sigma}}_{BG} = \mathbf{J}_{\mathbf{f}}|_{\tilde{\mathbf{z}}_{pol}} \tilde{\boldsymbol{\Sigma}}_{BG,pol} \mathbf{J}_{\mathbf{f}}^{\top}|_{\tilde{\mathbf{z}}_{pol}}, \tag{F.5}$$

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 $\tilde{\boldsymbol{\Sigma}}_{BG}$ 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:

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

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

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{R}_{OB,k}^{-1} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \tag{F.7}$$

$$\boldsymbol{\mu}_k = \begin{bmatrix} -\mathbf{R}_{OB,k}^{-1}\, {}_O\mathbf{t}_{OB,k} \\ -\phi_{OB,k} \end{bmatrix} \qquad \boldsymbol{\Sigma}_k = \tilde{\boldsymbol{\Sigma}}_{BG,k} \tag{F.8}$$

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

$$\hat{\mathbf{x}}_{k+1}^* = \hat{\mathbf{x}}_k \qquad \hat{\mathbf{P}}_{k+1}^* = \hat{\mathbf{P}}_k + \boldsymbol{\Sigma}_Q. \tag{F.9}$$

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

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

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

F.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:

$$\mathbf{p}_{wp,l\pm} =_O \mathbf{t}_{OG_l} + \mathbf{R}_{OG_l}[\pm x_G, 0, 0]^\top, \tag{F.11}$$

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 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 [59]:

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

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.

F.4 Experimental Setup

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

F.4.1 Simulation

We use RotorS [80] and Gazebo [116] 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 lap counts as 33.3% success.

F.4.2 Physical System

In all real-world experiments and data collection we use an in-house MAV platform with an Intel UpBoard [1] as the main computer running the CNN, EKF, and MPC. Additionally we use a Qualcomm Snapdragon Flight [2] as a visual-inertial odometry unit. The platform is shown in Fig. F.4. The CNN reaches an inference rate of $\sim 10\,\text{Hz}$ while the MPC runs at 100 Hz. With a take-off weight of 950 g the platform reaches thrust-to-weight ratio of ~ 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. F.5. In total, we collected 32,000 images.

[1] https://www.up-board.org/up/
[2] https://developer.qualcomm.com/hardware/qualcomm-flight

Figure F.4: Our platform, equipped with an Intel UpBoard and a Qualcomm Snapdragon Flight.

Figure F.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.

F.5 Results

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

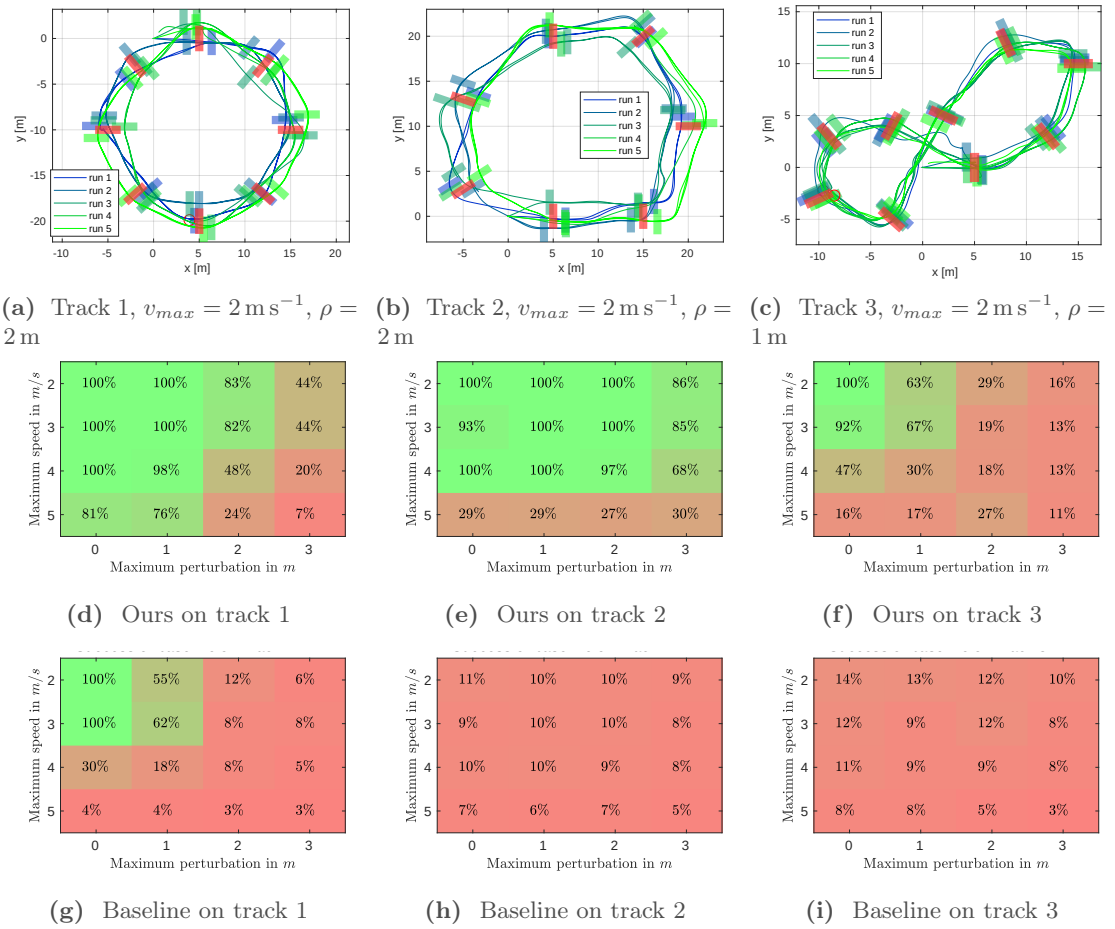
F.5.1 Simulation

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. [103]. The baseline was trained on the same data as our approach.

We evaluate the two methods on three tracks of increasing difficulty. Figs. F.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 $4\,\mathrm{m\,s^{-1}}$ 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. F.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 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.

F.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. F.7. No training data for the perception system was collected in this environment. Fig. F.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



(a) Track 1, $v_{max} = 2\,\mathrm{m\,s^{-1}}$, $\rho = 2\,\mathrm{m}$ **(b)** Track 2, $v_{max} = 2\,\mathrm{m\,s^{-1}}$, $\rho = 2\,\mathrm{m}$ **(c)** Track 3, $v_{max} = 2\,\mathrm{m\,s^{-1}}$, $\rho = 1\,\mathrm{m}$

(d) Ours on track 1 **(e)** Ours on track 2 **(f)** Ours on track 3

(g) Baseline on track 1 **(h)** Baseline on track 2 **(i)** Baseline on track 3

Figure F.6: Results of the simulation experiments. We compare the presented approach to the baseline [103] 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.

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

An example recorded trajectory of our approach is shown in Fig. F.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.

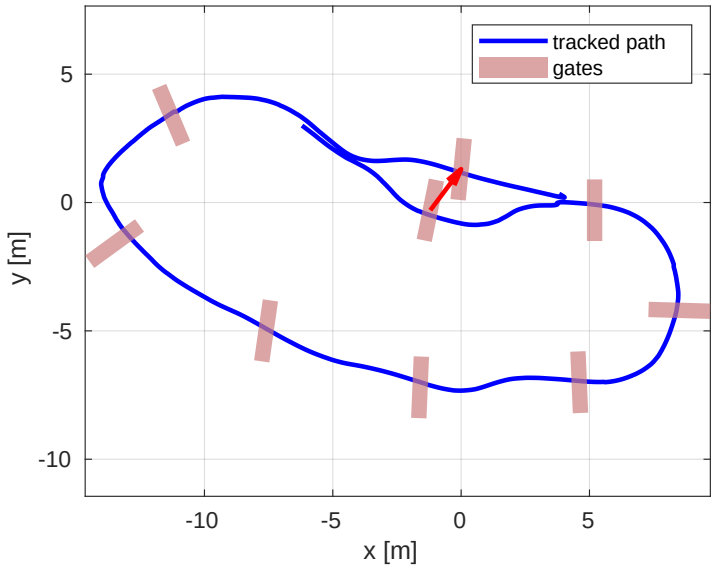


Figure F.7: Trajectory flown through multiple gate, one of which was moved as indicated by the red arrow. For visualization, only a single lap is illustrated.

F.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 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 $\mathrm{m\,s^{-1}}$, 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.

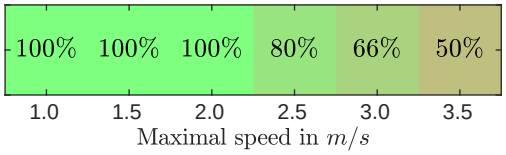


Figure F.8: Success rates of our approach in the real-world experiment. The reader is encouraged to watch the supplementary video to see the presented approach in action.

G PAMPC: Perception-Aware Model Predictive Control for Quadrotors

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

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

PAMPC: Perception-Aware Model Predictive Control for Quadrotors

Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza

Abstract — We present the first perception-aware model predictive control framework for quadrotors that unifies control and planning with respect to action and perception objectives. Our framework leverages numerical optimization to compute trajectories that satisfy the system dynamics and require control inputs within the limits of the platform. Simultaneously, it optimizes perception objectives for robust and reliable sensing by maximizing the visibility of a point of interest and minimizing its velocity in the image plane. Considering both perception and action objectives for motion planning and control is challenging due to the possible conflicts arising from their respective requirements. For example, for a quadrotor to track a reference trajectory, it needs to rotate to align its thrust with the direction of the desired acceleration. However, the perception objective might require to minimize such rotation to maximize the visibility of a point of interest. A model-based optimization framework, able to consider both perception and action objectives and couple them through the system dynamics, is therefore necessary. Our perception-aware model predictive control framework works in a receding-horizon fashion by iteratively solving a non-linear optimization problem. It is capable of running in real-time, fully onboard our lightweight, small-scale quadrotor using a low-power ARM computer, together with a visual-inertial odometry pipeline. We validate our approach in experiments demonstrating (i)) the conflict between perception and action objectives, and (ii)) improved behavior in extremely challenging lighting conditions.

Supplementary material

Video: https://youtu.be/9vaj829vE18

Code: https://github.com/uzh-rpg/rpg_mpc

G.1 Introduction

Thanks to the progresses in perception algorithms, the availability of low-cost cameras, and the increased computational power of small-scale computers, vision-based perception has recently emerged as the de facto standard in onboard sensing for micro aerial vehicles. This made it possible to replicate some of the impressive quadrotor maneuvers seen in the last decade [150, 148, 160, 32], which relied on motion-capture systems, using only onboard sensing, such as cameras and IMUs [63, 128, 224].

Cameras have a number of advantages over other sensors in terms of weight, cost, size, power consumption and field of view. However, vision-based perception has severe limitations: it can be intermittent and its accuracy is strongly affected by both the environment (e.g., texture distribution, light conditions) and motion of the robot (e.g., motion blur, camera pointing direction, distance from the scene). This means that one cannot always replace motion-capture systems with onboard vision, since the motion of a camera can negatively affect the quality of the estimation, posing hard bounds on the agility of the robot. On the other hand, perception can benefit from the robot motion if it is planned considering the necessities and the limitations of onboard vision. For example, to pass through a narrow gap while localizing with respect to it using an onboard camera, it is necessary to guarantee that the gap is visible at all times. Similarly, to navigate through an unknown environment, it is necessary to guarantee that the camera always points towards texture-rich regions.

To fully leverage the agility of autonomous quadrotors, it is necessary to create synergy between perception and action by considering them jointly as a single problem.

G.1.1 Contributions

Model Predictive Control (MPC) has become increasingly popular for quadrotor control [108, 169, 16] thanks to its capability of simultaneously dealing with different constraints and objectives through optimization. In this work, we present an MPC algorithm for quadrotors able to optimize both action and perception objectives.

Our framework satisfies the robot dynamics and computes feasible trajectories with respect to the input saturations. Such trajectories are not constrained to specific time or space parametrization (e.g., polynomials in time or splines), and tightly couple perception

Figure G.1: An example application of our PAMPC, where a quadrotor is asked to fly at $3\,\mathrm{m\,s^{-1}}$ around a region of interest while keeping it visible in the field of view of its camera.

and action. To do so, perception objectives aimed at rendering vision-based estimation more robust are taken into account in the optimization problem. Such objectives are the visibility of a point of interest the robot needs to maintain in the image, and the minimization of the velocity of its projection onto the image plane. The main challenge in this is to simultaneously cope with action (e.g., dynamics, underactuation, saturations) and perception objectives, due to the potential conflicts between them.

To solve this problem, we leverage numerical optimization to compute trajectories that are optimal with respect to a cost function considering both the dynamics of the robot and the quality of perception. To fully exploit the agility of a quadrotor, we incorporate perception objectives into the optimization problem not as constraints, but rather as components to be optimized. This results in a perception-aware framework which is intrinsically tailored to agile navigation, since the optimizer can trade off between perception and action objectives (cf. Fig. G.1, depicting fast circle flight while adjusting the heading to look at a point of interest). Furthermore, considering perception in the cost function reduces the computation load of the model predictive control pipeline, allowing it to run in real-time on a low-power onboard computer. Our approach does not depend on the task and can potentially provide benefits to a large variety of applications, such as vision-based localization, target tracking, visual servoing, and obstacle detection. We validate our perception-aware model predictive control framework in real-world experiments using a small-scale, lightweight quadrotor platform.

G.1.2 Related Work

The aforementioned shift from offboard to onboard sensing based on cameras resulted in an increased number of works trying to connect perception and action.

In [183], the authors proposed a method to compute minimum-time trajectories that take into account the limited field of view of a camera to guarantee visibility of points of interests. Such a method requires the trajectory to be parametrized as a B-spline polynomial, constraining the kind of motion the robot can perform. Also, perception is included in the planning problem as hard constraint, posing an upper-bound to the agility of the robot since such constraints must be satisfied at all times. Furthermore, the velocity of the projection of the points of interest in the image is not taken into account. Finally, the algorithm was not suited for real-time control of a quadrotor, and was only tested in simulations..

In [223], the authors focused on combining visual servoing with active Structure from Motion and proposed a solution to modify the trajectory of a camera in order to increase the quality of the reconstruction. In such a work, a trajectory for the tracked features in the image plane was required, and the null space of the visual servoing task was exploited in order to render it possible for such feature to track the desired trajectory. Furthermore, the authors did not consider the underactuation of the robot, which can significantly lower the performance of the overall task due to potentially conflicting dynamics and perception objectives.

In [41] and [77], information gain was used to bridge the gap between perception and action. In the first work, the authors tackled the problem of selecting trajectories that minimize the pose uncertainty by driving the robot toward regions rich of texture. In the second work, a technique to minimize the uncertainty of a dense 3D reconstruction based on the scene appearance was proposed. In both works, however, near-hover quadrotor flight was considered, and the underactation of the platform was not taken into account.

In [215], a hybrid visual servoing technique for differentially flat systems was presented. A polynomial parameterization of the flat outputs of the system was required, and due to the computational load required by the designed optimization framework, an optimal trajectory was computed in advance and never replanned. This did not allow coping with external disturbances and unmodelled dynamics, which during the execution of the trajectory can lead to behaviours different from the expected one.

In [164] and [165], a real-time motion planning method for aerial videography was presented. In these works, the main goal was to optimize the viewpoint of a pan-tilt camera carried by an aerial robot in order to improve the quality of the video recordings. Both works were mainly targeted to cinematography, therefore they considered objectives such as the size of a target of interest and its visibility. Conversely, we target robotic sensing and consider objectives aimed at facilitating vision-based perception.

In [188], the authors proposed a two-step approach for target-aware visual navigation. First, position-based visual servoing was exploited to find a trajectory minimizing the reprojection error of a landmark of interest. Then, a model predictive control pipeline was used to track such a trajectory. Conversely, we solve the trajectory optimization and tracking within a single framework. Additionally, that work only aimed at rendering the target visible, but did not take into account that, due to the motion of the camera, it might not be detectable because of motion blur. We cope with this problem by considering in the optimization problem the velocity of the projection of the point of interest in the image plane.

G.1.3 Structure of the Paper

The remainder of this paper is organized as follows. In Sec. G.2 we provide the general formulation of the problem. In Sec. G.3 we derive the model for the dynamics of the projection of a 3D point into the image plane for the case of a quadrotor equipped with a camera. In Sec. G.4 we present our perception-aware optimization framework, describing the objectives and the constraints it takes into account. In Sec. G.5 we validate our approach in different real-world experiments showcasing the capability of our framework. In Sec. G.6 we discuss our approach and provide additional insights and in Sec. G.7 we draw the conclusions.

G.2 Problem Formulation

For truly autonomous robot navigation, two components are essential: (i)) perception, both of the ego-motion and of the surrounding environment; (ii)) action, meant as the combination of motion planning and control algorithms. A very wide literature is available for both of them. However, they are rarely considered as a joint problem.

The need for coupled perception and action can be easily explained. To guarantee safety, accurate and robust perception is necessary. Nevertheless, the quality of vision-based perception is strongly affected by the motion of the camera. On the one hand, it can degrade its performance by not making it possible to extract sufficiently accurate information from images. For example, lack of texture or blur due to camera motion can lead to algorithm failure. On the other, the quality of vision-based perception can improve significantly if its limitations and requirements are considered, e.g. by rendering highly-textured areas visible in the image and by reducing motion blur. Therefore it is necessary to create synergy between perception and action.

Let $\mathbf{x}$ and $\mathbf{u}$ be the state and input vectors of a robot, respectively. Assume its dynamics to be described by a set of differential equations $\dot{\mathbf{x}} = \boldsymbol{f}(\mathbf{x}, \mathbf{u})$. Furthermore, let $\mathbf{z}$ be the state vector of the perception system (e.g., 3D points' projection onto the image plane),

and σ a vector of parameters characterizing it (e.g., the focal length of the camera or its field of view). The perception state and the robot state are coupled through the robot dynamics, namely $\mathbf{z} = \boldsymbol{f}_p(\mathbf{x}, \mathbf{u}, \sigma)$. Given certain action objectives, we can define an action cost $\mathcal{L}_a(\mathbf{x}, \mathbf{u})$. Similarly, we can define a cost $\mathcal{L}_p(\mathbf{z})$ for the perception objectives.

We can then formulate the coupling of perception and action as an optimization problem:

$$\begin{aligned} \min_{\mathbf{u}} \quad & \int_{t_0}^{t_f} \mathcal{L}_a(\mathbf{x}, \mathbf{u}) + \mathcal{L}_p(\mathbf{z})\, dt \\ \text{subject to} \quad & \mathbf{r}(\mathbf{x}, \mathbf{u}, \mathbf{z}) = 0 \\ & \mathbf{h}(\mathbf{x}, \mathbf{u}, \mathbf{z}) \leq 0, \end{aligned} \tag{G.1}$$

where $\mathbf{r}(\mathbf{x}, \mathbf{u}, \mathbf{z})$ and $\mathbf{h}(\mathbf{x}, \mathbf{u}, \mathbf{z})$ represent equality and inequality constraints that the solution should satisfy for perception, action, or both of them simultaneously.

G.3 Methodology

Any computer vision algorithm aimed at providing a robot with the information necessary for navigation (e.g., pose estimation, obstacle detection, etc) has two fundamental requirements. First, the points of interest used by the algorithm to provide the aforementioned information must be visible in the image. For example, such points can be the landmarks used for pose estimation by visual odometry algorithms, or the points belonging to an object for obstacle detection. If such points are not visible while the robot is moving, there is no way the algorithm can cope with the absence of information. Second, such points of interest must be clearly recognizable in the image. Depending on the motion of the camera and the distance from the scene, the projection of a 3D point onto an image can suffer from motion blur, making it very complicated, if not impossible, to extract meaningful information. Therefore, the motion of the camera should be thoroughly planned to guarantee robust visual perception.

Based on the considerations above, in this work we consider two perception objectives in our framework: (i)) visibility of points of interest, and (ii)) minimization of the velocity of their projection onto the image plane. In the following, we study the relation between the motion of a quadrotor equipped with an onboard camera and the projection onto the image plane of a point in space. Without loss of generality, we consider the case of a single 3D point of interest. Our goal is to couple perception and action into an optimization framework by expressing the dynamics of its projection onto the image plane as a function of the state and input vectors of a quadrotor.

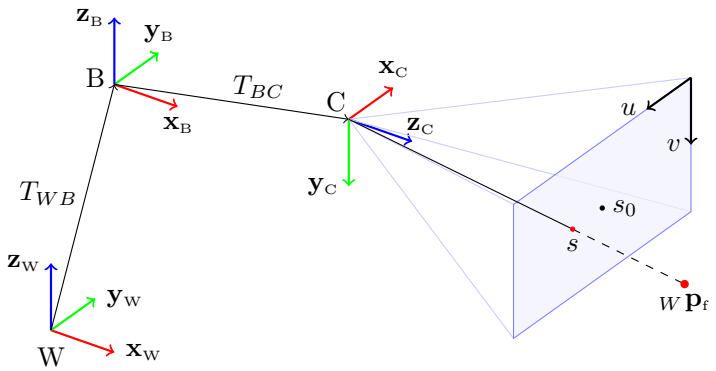


Figure G.2: A schematics representing the world frame W, the body frame B and the camera frame C. The position and orientation of B with respect to W is provided by T_{WB}. The constant rigid body transformation T_{BC} provides the extrinsics of the camera. A feature located at ${}_W\mathbf{p}_f$ is projected into the image plane onto a point of coordinates $\mathbf{s}$. $\mathbf{s}_0$ represents the principal point.

G.3.1 Nomenclature

In this work, we make use of a world frame W with orthonormal basis $\{\mathbf{x}_W, \mathbf{y}_W, \mathbf{z}_W\}$. The quadrotor frame B, also referred to as the body frame, has orthonormal basis $\{\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B\}$. Finally, we assume the robot to be equipped with a camera, whose reference frame C has orthonormal basis $\{\mathbf{x}_C, \mathbf{y}_C, \mathbf{z}_C\}$. Fig. G.2 provides a clear overview about the reference frames.

Throughout this manuscript, we represent vectors as bold quantities having a prefix, representing the frame in which they are expressed, and a suffix, indicating the origin and the end of such a vector. For example, the quantity ${}_W\mathbf{p}_{WB}$ represents the position of the body frame B with respect to the world frame W, expressed in the world frame. To simplify the notation, if a vector has no prefix, we assume it to be expressed in the first frame reported in the suffix (i.e., the frame where the vectors origin is).

We use quaternions to represent the orientation of a rigid body. The time derivative of a quaternion $\mathbf{q} = (q_w, q_x, q_y, q_z)^\intercal$ is given by $\dot{\mathbf{q}} = \frac{1}{2}\Lambda(\omega) \cdot \mathbf{q}$, where the skew-symmetric matrix $\Lambda(\omega)$ of a vector $\omega = (\omega_x, \omega_y, \omega_z)^\intercal$ is defined as:

$$\Lambda(\omega) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}. \tag{G.2}$$

Finally, we use the operator $\odot$ to denote the multiplication between a quaternion and a vector. More specifically, multiplying a vector $\mathbf{v}$ with the quaternion $\mathbf{q}$ means rotating $\mathbf{v}$

by the rotation induced by $\mathbf{q}$. By doing so, we obtain a vector $\mathbf{v}' = \mathbf{v} \odot \mathbf{q} = Q\mathbf{v}$ where:

$$Q = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2(q_x q_y + q_w q_z) & 2(q_x q_z - q_w q_y) \\ 2(q_x q_y - q_w q_z) & 1 - 2q_x^2 - 2q_z^2 & 2(q_y q_z + q_w q_x) \\ 2(q_x q_z + q_w q_y) & 2(q_y q_z - q_w q_x) & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}.$$

G.3.2 Quadrotor Dynamics

Let $\mathbf{p}_{WB} = (p_x, p_y, p_z)^\intercal$ and $\mathbf{q}_{WB} = (q_w, q_x, q_y, q_z)^\intercal$ be the position and the orientation of the body frame with respect to the world frame W, expressed in world frame, respectively (cf. Fig. G.2). Additionally, let $\mathbf{v}_{WB} = (v_x, v_y, v_z)^\intercal$ be the linear velocity of the body, expressed in world frame, and $\mathbf{\Omega}_B = (\omega_x, \omega_y, \omega_z)^\intercal$ its angular velocity, expressed in the body frame. Finally, let $\mathbf{c} = (0, 0, c)^\intercal$ be the mass-normalized thrust vector, where $c = (f_1 + f_2 + f_3 + f_4)/m$, f_i is the thrust produced by the i-th motor, and m is the mass of the vehicle. In this work, we use the dynamical model of a quadrotor proposed in [160]:

$$\begin{aligned} \dot{\mathbf{p}}_{WB} &= \mathbf{v}_{WB} \\ \dot{\mathbf{v}}_{WB} &= {}_W\,\mathbf{g} + \mathbf{q}_{WB} \odot \mathbf{c} \\ \dot{\mathbf{q}}_{WB} &= \frac{1}{2}\Lambda\left(\mathbf{\Omega}_B\right) \cdot \mathbf{q}_{WB} \end{aligned} \tag{G.3}$$

where ${}_W\mathbf{g} = (0, 0, -g)^\intercal$ is the gravity vector, with $g = 9.81\,\mathrm{m\,s^{-2}}$. The state and the input vectors of the system are $\mathbf{x} = [\mathbf{p}_{WB}, \mathbf{v}_{WB}, \mathbf{q}_{WB}]^\intercal$ and $\mathbf{u} = \left[c, \mathbf{\Omega}_B^\intercal\right]^\intercal$, respectively.

G.3.3 Perception Objectives

Let ${}_W\mathbf{p}_f = ({}_W p_{fx}, {}_W p_{fy}, {}_W p_{fz})$ be the 3D position of a point of interest (landmark) in the world frame W (cf. Fig. G.2). We assume the body to be equipped with a camera having extrinsic parameters described by a constant rigid body transformation $T_{BC} = [\mathbf{p}_{BC}, \mathbf{q}_{BC}]$, where $\mathbf{p}_{BC}$ and $\mathbf{q}_{BC}$ are the position and the orientation of C with respect to B. The coordinates ${}_C\mathbf{p}_f = ({}_C p_{fx}, {}_C p_{fy}, {}_C p_{fz})^\intercal$ of ${}_W\mathbf{p}_f$ in the camera frame C are given by:

$$\begin{aligned} {}_C\mathbf{p}_f = &\left(\mathbf{q}_{WB}\ \mathbf{q}_{BC}\right)^{-1} \odot \\ &\left({}_W\mathbf{p}_f - \left(\mathbf{q}_{WB} \odot\ \mathbf{p}_{BC} + \mathbf{p}_{WB}\right)\right). \end{aligned} \tag{G.4}$$

The point ${}_C\mathbf{p}_f$ in camera frame is projected into the image plane coordinates $\mathbf{s} = (u, v)^\intercal$ according to classical pinhole camera model [229]:

$$u = f_x \frac{{}_C p_{fx}}{{}_C p_{fz}}, \quad v = f_y \frac{{}_C p_{fy}}{{}_C p_{fz}} \tag{G.5}$$

where f_x, f_y are the focal lengths for pixel rows and columns, respectively.

To guarantee robust vision-based perception, the projection $\mathbf{s}$ of a point of interest ${}_W\mathbf{p}_f$ should be as close as possible to the center of the image for two reasons. First, keeping its projection in the center of the image results in the highest safety margins against external disturbances. The second reason comes from the fact that the periphery of the image is typically characterized by a non-negligible distortion, especially for large field of view cameras. A number of models for such distortion are available in the literature, as well as techniques to estimate their parameters to compensate the effects of the distortion. However, such a compensation is never perfect and this can degrade the accuracy of the estimates.

As previously mentioned, in addition to rendering the point of interest visible in the image, we are interested in reducing the velocity of its projection onto the image plane. We assume the point of interest to be static, but similar considerations apply to the case where such a point of interest moves with respect to the world frame. To express the projection velocity as a function of the quadrotor state and input vectors, we can differentiate (G.5) with respect to time:

$$\begin{aligned} \dot{u} &= f_x \frac{{}_C\dot{p}_{fx}\ {}_Cp_{fz} - {}_Cp_{fx}\ {}_C\dot{p}_{fz}}{{}_Cp_{fz}^2}, \\ \dot{v} &= f_y \frac{{}_C\dot{p}_{fy}\ {}_Cp_{fz} - {}_Cp_{fy}\ {}_C\dot{p}_{fz}}{{}_Cp_{fz}^2}. \end{aligned} \tag{G.6}$$

Eq. (G.6) can be written in a compact form as:

$$\dot{\mathbf{s}} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{f_x}{{}_Cp_{fz}^2} & 0 \\ \frac{f_y}{{}_Cp_{fz}^2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \left({}_C\mathbf{p}_f \times_C \dot{\mathbf{p}}_f\right). \tag{G.7}$$

To compute the term ${}_C\dot{\mathbf{p}}_f$, we can differentiate (G.4) with respect to time:

$${}_C\dot{\mathbf{p}}_f = -\frac{1}{2}\Lambda\left(\boldsymbol{\Omega}_C\right)\ {}_C\mathbf{p}_f - {}_C\mathbf{v}_{WC}, \tag{G.8}$$

where:

$$\begin{aligned} {}_C\mathbf{v}_{WC} = &\left(\mathbf{q}_{WB}\ \mathbf{q}_{BC}\right)^{-1} \odot \\ &\left(\frac{1}{2}\Lambda\left(\boldsymbol{\Omega}_B\right)\mathbf{q}_{WB} \odot \mathbf{p}_{BC} + \mathbf{v}_{WB}\right), \\ \boldsymbol{\Omega}_C = &\ \mathbf{q}_{BC}^{-1} \odot \boldsymbol{\Omega}_B. \end{aligned} \tag{G.9}$$

G.3.4 Action Objectives

For a quadrotor to execute a desired task (e.g., reach a target position in space), a suitable trajectory has to be planned. In this regard, for a quadrotor two objectives should be considered.

The first comes from the bounded inputs available to the system. The thrust each motor can produce has both an upper and a lower bound, leading to a limited input vector $\mathbf{u}$. Therefore, denoting the subset of the allowed inputs as $\mathcal{U}$, the planned trajectory should be such that the condition $\mathbf{u}(t) \in \mathcal{U}\ \forall t$ can be satisfied.

The second objective to be considered comes from the underactuated nature of a quadrotor. In the most common configuration, all the rotors point in the same direction, typically along the axis $\mathbf{z}_B$ of the body. This means that the robot can accelerate only in this direction. Therefore, to move in the 3D space, it is necessary to exploit the system dynamics (G.3) by coupling the translational and the rotational motions of the robot to follow the desired trajectory.

G.3.5 Challenges

The perception (Sec. G.3.3) and the action (Sec. G.3.4) objectives previously described are both necessary for vision-based quadrotor navigation. Considering them simultaneously is challenging due to the possible conflict among them. Indeed, for a quadrotor to track a reference trajectory, it needs to rotate to align its thrust with the direction of the desired acceleration. However, the perception objective might require to minimize such rotation to maximize the visibility of a point of interest. A model-based optimization framework able to consider both perception and action objectives and couple them through the system dynamics is therefore necessary.

G.4 Model Predictive Control

Formulating coupled perception and action as an optimization problem has the advantages of being able to satisfy the underactuated system dynamics and actuator constraints (i.e., input boundaries) and to minimize the predicted costs along a time horizon. In contrast, classical control schemes are incapable of predicting costs and the corresponding trajectory (e.g., PID controllers) and guaranteeing input boundaries (PID, LQR).

The basic formulation of such an optimization is given in (G.1), which in our case results in a non-linear program with quadratic costs. This can then be approximated by a sequential quadratic program (SQP) where the solution of the non-linear program is iteratively approximated and used as a model predictive control (MPC). To this regard, for the MPC to be effective, the optimization scheme has to run in real-time, at the

desired control frequency. To achieve this, we first discretize the system dynamics with a time step dt for a time horizon t_h into $\mathbf{x}_i \ \forall i \in [1, N]$ and $\mathbf{u}_i \ \forall i \in [1, N-1]$. We define the time-varying state cost matrix as $\mathcal{Q}_{x,i} \ \forall i \in [1, N]$. Furthermore, the time-varying perception and input cost matrices are defined as $\mathcal{Q}_{p,i}$ and $\mathcal{R}_i$, $\forall i \in [1, N-1]$, respectively. Finally, let $\mathbf{z} = [\mathbf{s}, \dot{\mathbf{s}}]$ be the perception function. It is important to recall that $\mathbf{z}$ is a function of the quadrotor's state and input variables, as remarked in Eq. (G.4) to (G.9). The resulting cost function we consider is:

$$\begin{aligned}\mathcal{L} = \bar{\mathbf{x}}_N^\intercal \mathcal{Q}_{x,N} \bar{\mathbf{x}}_N \ + \\ + \sum_{i=1}^{N-1} \left(\bar{\mathbf{x}}_i^\intercal \mathcal{Q}_{x,i} \bar{\mathbf{x}}_i + \bar{\mathbf{z}}_i^\intercal \mathcal{Q}_{p,i} \bar{\mathbf{z}}_i + \bar{\mathbf{u}}_i^\intercal \mathcal{R}_i \bar{\mathbf{u}}_i \right),\end{aligned} \tag{G.10}$$

where the values $\bar{\mathbf{x}}, \bar{\mathbf{z}}, \bar{\mathbf{u}}$ refer to the difference with respect to the reference of each value. In our case, the reference value for $\mathbf{z}$ is the null vector (i.e., center of the image and zero velocity) and the reference for the states and inputs are given by a target pose or a precomputed trajectory (that neglects the perception objectives).

The inputs $\mathbf{u}$, consisting of c and $\boldsymbol{\Omega}_B$, as well as the velocity $\mathbf{v}_{WB}$ are limited by the constraints:

$$c_{min} \leq c \leq c_{max}, \tag{G.11}$$

$$-\Omega_{max} \leq \boldsymbol{\Omega}_B \leq \Omega_{max}, \tag{G.12}$$

$$-v_{max} \leq \mathbf{v}_{WB} \leq v_{max}, \tag{G.13}$$

where $c_{min}, c_{max}, \Omega_{max}, v_{max} \in \mathcal{R}_+$.

To include the dynamics as in (G.3), we use multiple shooting as transcription method and a Runge-Kutta integration scheme. We refer the reader to [99] and [98] for more details on the transcription of the dynamics for optimization.

We approximate the solution of the optimization problem by executing one iteration at each control loop and use as initial state the most recent available estimate $\mathbf{x}_{est}$ provided by a Visual-Inertial Odometry pipeline running onboard the vehicle (see Sec. G.5.1). To achieve good approximations, it is important to run these iterations significantly faster than the discretization time of the problem and to keep the previous solution as initialization trajectory of the next optimization. Such a SQP scheme leads to a fast convergence towards the exact solution, since the system is always close to the last linearization, and the deviation of each state $\mathbf{x}_i$ between two iterations is very small.

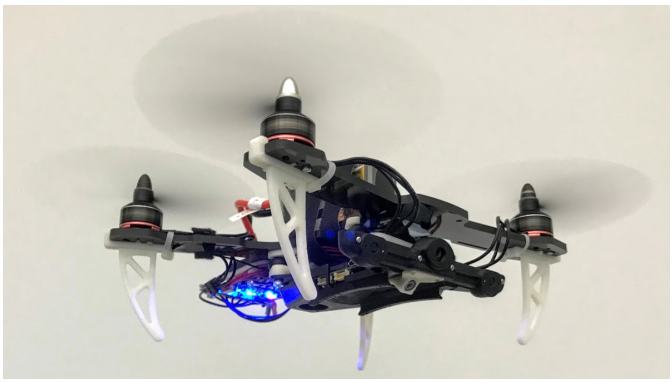


Figure G.3: The quadrotor used for the experiments.

G.5 Experiments

In order to show the potential of our perception-aware model predictive control, we ran our approach onboard a small, vision-based, autonomous quadrotor. We refer the reader to the attached video showcasing the experiments.

G.5.1 Experimental Setup

We used a small and lightweight quadrotor platform to achieve high agility through high torque-to-inertia and thrust-to-weight ratios, and improve simplicity and safety for the user (cf. Fig G.3). The quadrotor had a take-off weight of 420 g, a thrust-to-weight ratio of ~ 2, and a motor-to-motor diagonal of 220 mm. We used a Qualcomm Snapdragon Flight board with a quad-core ARM processor at up to 2.26 GHz and 2 GB of RAM, paired with a Qualcomm Snapdragon Flight ESC. The board was equipped with an Inertial Measurement Unit and a forward-looking, wide field-of-view global-shutter camera tilted down by 45° for visual-inertial odometry (VIO) using the Qualcomm mvSDK. It ran ROS on Linux and our self-developed flight stack. We setup the optimization with ACADO [98] and used qpOASES [65] as solver. As discretization step, we chose $dt = 0.1\,\text{s}$ with a time horizon of $t_h = 2\,\text{s}$ and ran one iteration step in each control loop with a frequency of 100 Hz. Therefore, the iteration ran roughly $10\times$ faster than the discretization time, resulting in small deviations of the predicted state vector between iterations and facilitating convergence. The code developed in this work is publicly available as open-source software.

G.5.2 Experiment Description and Results

To prove the functionality and importance of our PAMPC, we ran three experiments. In the first experiment, the controller modified a circular trajectory to improve the visibility of a point of interest. In the second experiment, the controller handled *hover-to-hover*

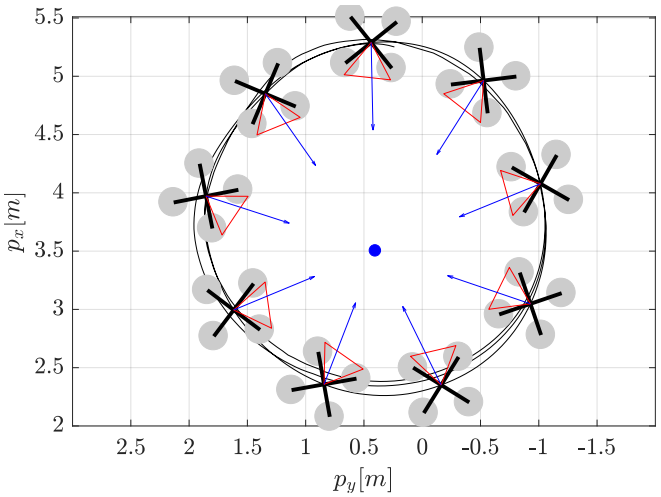


Figure G.4: Executed trajectory with quadrotor heading while the arrow points toward the point of interest (blue).

flight by deviating from a straight line trajectory to keep the point of interest visible. In the third experiment, it enabled vision-based flight in an extremely challenging scenario. All the experiments were conducted with onboard VIO and onboard computation of the PAMPC, without any offline computation and without any motion-capture system.

Circular Flight

We setup a small pile of boxes in the middle of a room otherwise poor of texture. We did this to force the VIO pipeline to use such boxes as features for state estimation. The centroid of these features was set as our point of interest. We provided the robot with a circular reference trajectory around the aforementioned boxes and asked it to fly along such a trajectory while maintaining the boxes visible in the center of the image (cf. Fig. G.1). We evaluated the performance of our framework for speeds along the circle from $1\,\mathrm{m\,s^{-1}}$ to $3\,\mathrm{m\,s^{-1}}$.

The results of one run of the circular flight experiments at $3\,\mathrm{m\,s^{-1}}$ are depicted in Fig. G.4. Despite the agility of the maneuver, which requires large deviations from the hover conditions, the robot is able to keep the point of interest visible in the onboard image. Fig. G.9a reports the reprojection in the image plane for such point of interest.

Hover-To-Hover Flight

In this experiment within the same scenario as in Sec. G.5.2, we showed the capabilities of our framework for *hover-to-hover* flight. More specifically, we requested a pose jump from a position $\mathbf{p}_1$ to $\mathbf{p}_2$ at equal height (cf. Fig. G.7). During that maneuver, the quadrotor had to pitch down to reach the desired acceleration, but controversially should pitch as

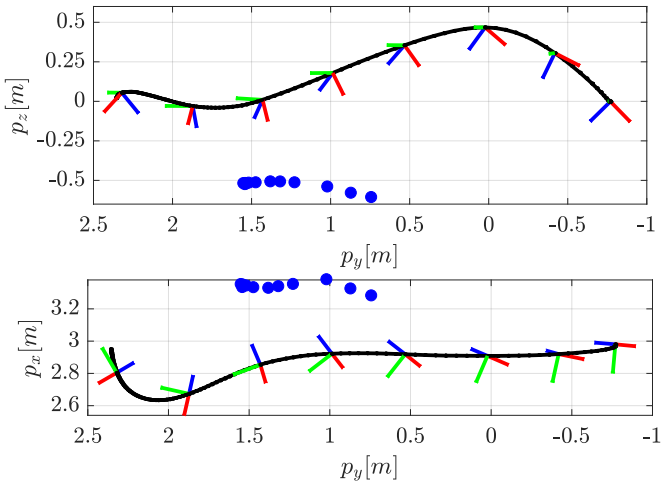


Figure G.5: Quadrotor path in hover-to-hover, looking towards the centroid of tracked features (blue), with the camera frame indicated by $\{x_C, y_C, z_C\}$.

little as possible to keep the point of interest visible. A sequence of this experiment is visible in Fig. G.7.

One can easily see that, despite the start and end positions are at the same height, the quadrotor not only pitches to go towards the new reference in an horizontal motion, but also accelerates upward (i.e., in positive z, cf. Fig. G.5). This results in a smaller pitch angle and a higher thrust to reach the same y-acceleration, which is helpful for perception since it brings the features towards the center of the frame due to the higher altitude. If perception objectives were not considered, the resulting trajectory would have not required any height change, potentially leading to a poor visibility of the point of interest. The full motion of the quadrotor is depicted in Fig. G.5, where the exploitation of the added height and the orientation of the camera frame can be seen. Finally, in Fig. G.9b we show the reprojection in the image plane for the point of interest.

Darkness Scenario

This experiment was targeted towards extremely challenging scenarios, such as flight in a very dark environment, or otherwise difficult illumination conditions (cf. Fig. G.8). To demonstrate the performance in such a scenario, we flew the vehicle several times in a dark room with two illuminated spots. If the illuminated spots left the field of view for a moment, the VIO pipeline would drift quickly or even completely loose track, potentially leading to a crash. Therefore, in such scenarios it was of immense importance to keep the few available features always visible. The flown path was given by a trajectory passing through four waypoints forming a rectangle, but without any heading reference. The quadrotor correctly adjusts its heading to keep the illuminated spot in its field of view, because this is the only source of trackable features. Fig. G.6 visualizes a setup with

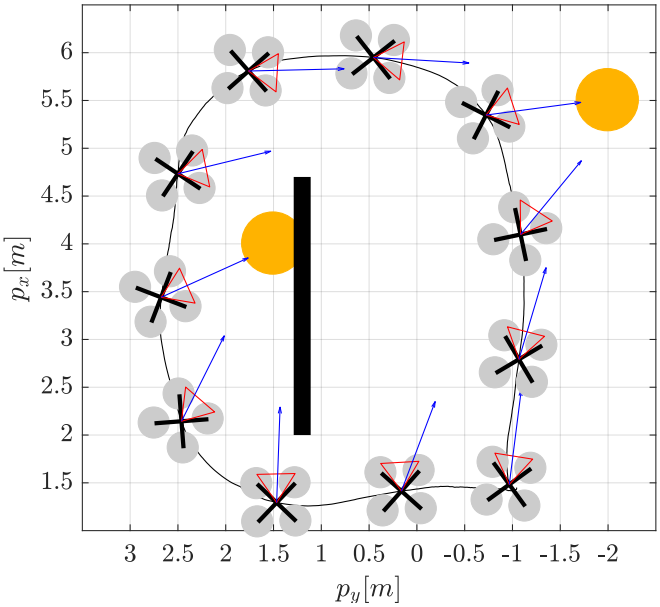


Figure G.6: Path of the quadrotor, looking towards light spots (yellow), with camera direction (red triangle) and point-of-interest direction (blue arrow).

two spotlights and a cardboard wall in between, where the quadrotor first focuses on the upper right illuminated spot, and further down the track switches to the second illuminated spot behind the wall. The reprojection of the point of interest in the image plane is shown in Fig. G.9c.

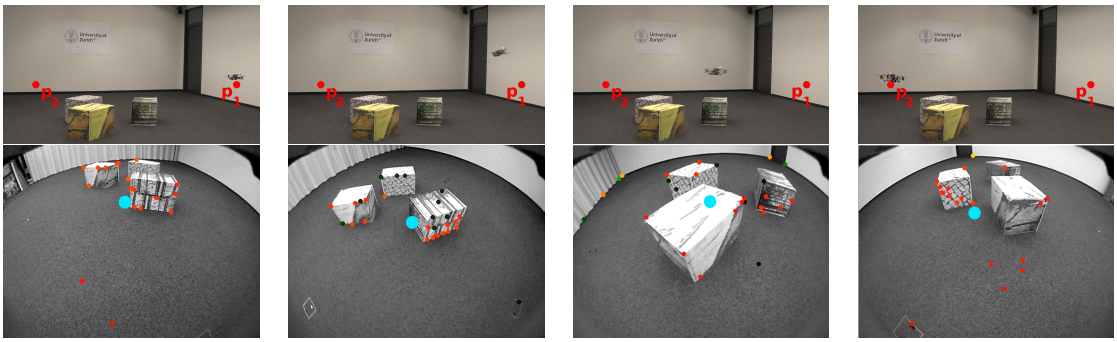


Figure G.7: A sequence of the visibility experiment for the hover-to-hover flight experiment, with time progressing from left to right. The quadrotor performs a maneuver to fly to a new reference pose, exploiting additional height to pitch less and keep the point of interest (centroid of the vision features, marked as cyan circle) in the center of the image. The corresponding footage is available in the accompanying video.

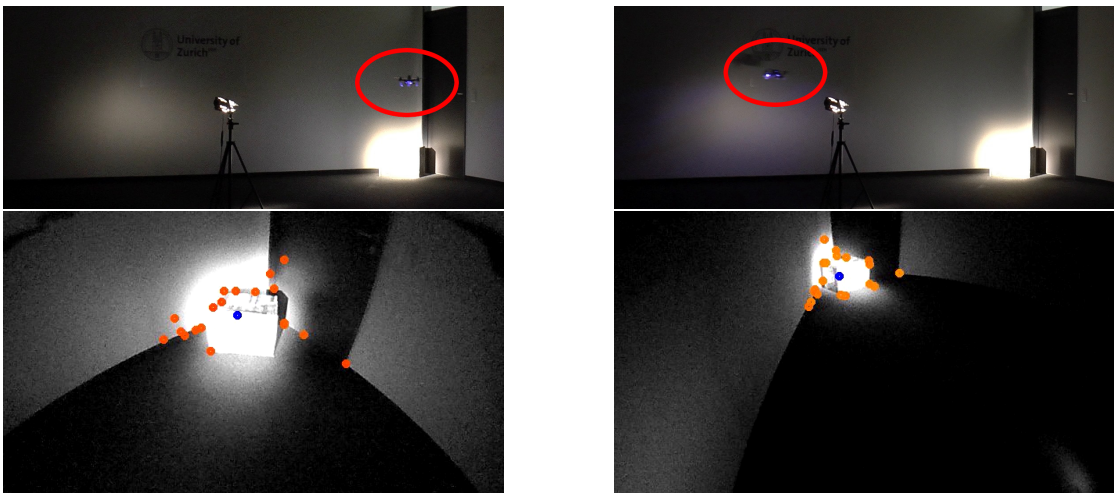
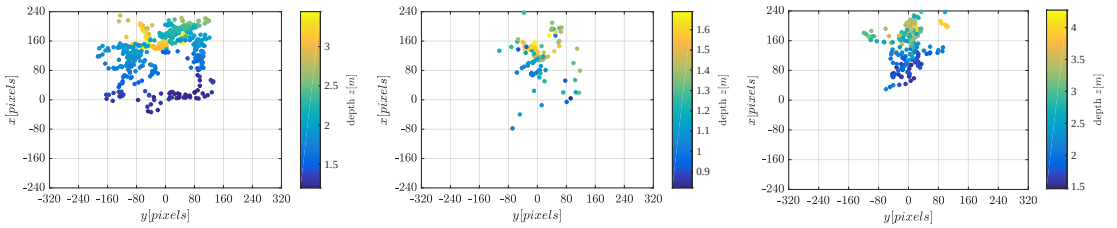


Figure G.8: A sequence of the darkness experiment with time progressing from left to right. The quadrotor, highlighted by a red circle in the figures in the first row, tracks a trajectory and adjusts its heading to keep the point of interest (centroid of the vision features) in field of view.



(a) Reprojection for the circular trajectory. **(b)** Reprojection for the *hover-to-hover* experiment. **(c)** Reprojection for the darkness experiment.

Figure G.9: Reprojection of the point of interest in image plane, colored according to the depth with respect to the camera frame.

G.6 Discussion

G.6.1 Choice of the optimizer

To implement the optimization problem, we chose to use ACADO because of two main reasons: (i)) it is capable of transcribing system dynamics with single- and multiple-shooting and integration schemes, as well as provide an interface to a solver; (ii)) it generates c++ code, which then is compiled directly on the executing platform, which allows it to use accelerators and optimizations tailored to the platform.

G.6.2 Convexity of the problem

Our state and input space is a convex domain, hence also any quadratic cost in those is convex. The perception costs could be argued to be non-convex due to the division by ${}_C p_{fz}$ in the projection (G.5). However, on examination of the projection one will notice that the denominator ${}_C p_{fz}$ is always positive, since the pinhole camera projection model does not allow negative or zero depths. We can therefore constrain ${}_C p_{fz}$ to be positive, rendering all possible solutions in the positive halfplane $\mathcal{R}^+$ and therefore recover convexity.

G.6.3 Choice of point of interest

In our experiments, we used the centroid of detected features as our point of interest. Assuming that all the features are equally important, instead of optimizing for each individually, we can summarize them as their centroid, which results in the same optimal solution.

G.6.4 PAMPC Parameters

We chose a discretization of $dt = 0.1\,\text{s}$ and a time horizon of $t_h = 2\,\text{s}$. One could always argue that a longer time horizon and a shorter discretization step are beneficial, but they also increase the computation time by roughly $\mathcal{O}(N^2)$ with the number of discretization nodes $N = \frac{t_h}{dt}$. In our experience, we could not identify any significant gain from smaller discretization steps nor from a longer time horizon.

G.6.5 Computation Time

Since the computation time must be low enough to execute the optimization in a real time scheme, we show that it is significantly lower than the one required by the controller frequency of 100 Hz. Indeed, our PAMPC requires on average 3.53 ms. It is interesting to note that this is the case for both an idle CPU and while running the full pipeline with

VIO and our full control pipeline. This is due to the quad-core ARM CPU and the fact that our full pipeline without the PAMPC takes up only 3 cores leaving one free for the PAMPC. However, the standard deviation increases significantly if the CPU is under load (from 0.155 ms to 0.354 ms), even though the maximal execution time always stays below 5 ms.

G.6.6 Drawbacks of a Two-Step Approach

An alternative approach to the problem tackled in this work is to use the differential flatness as in [148] to plan a translational trajectory connecting the start and end positions, and subsequentially plan the yaw angle to point the camera towards the point of interest. After planning, a suitable controller could be used to track the desired reference trajectory. Although possible, such a solution would lead to sub-optimal results because of the following reasons: (i)) the roll and pitch angles of the quadrotor would be planned without considering the visibility objective, therefore might render the point of interest not visible in the image despite the yaw control; (ii)) because of the split between planning and control, even if the first would provide guarantees about visibility, these could not be preserved during the control stage due to deviations from the nominal trajectory; (iii)) it would be challenging to provide guarantees about the respect of the input saturations. Therefore, our proposed approach considering perception, planning and control as a single problem leads to superior results.

G.7 Conclusions

In this work, we presented a perception-aware model predictive control (PAMPC) algorithm for quadrotors able to optimize both action and perception objectives. Our framework computes trajectories that satisfy the system dynamics and inputs limits of the platform. Additionally, it optimizes perception objectives by maximizing the visibility of a point of interest in the image and minimizing the velocity of its projection into the image plane for robust and reliable sensing. To fully exploit the agility of a quadrotor, we incorporated perception objectives into the optimization problem not as constraints, but rather as components in the cost function to be optimized. Our algorithm is able to run in real-time on an onboard ARM processor, in parallel with a VIO pipeline, and is used to directly control the robot. We validated our approach in real-world experiments using a small-scale, lightweight quadrotor platform.

H Onboard State Dependent LQR for Agile Quadrotors

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

Philipp Foehn and Davide Scaramuzza. "Onboard State Dependent LQR for Agile Quadrotors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018. DOI: 10.1109/ICRA.2018.8460885

Onboard State Dependent LQR for Agile Quadrotors

Philipp Foehn and Davide Scaramuzza

Abstract — State-of-the-art approaches in quadrotor control split the problem into multiple cascaded subproblems, exploiting the different time-scales of the rotational and translational dynamics. They calculate a desired acceleration as input for a cascaded attitude controller but omit the attitude dynamics. These approaches use limits on the desired acceleration to maintain feasibility and robustness through the control cascade. We propose an implementation of an LQR controller, which: (i)) is linearized depending on the quadrotor's state; (ii)) unifies the control of rotational and translational states; (iii)) handles time-varying system dynamics and control parameters. Our implementation is efficient enough to compute the full linearization and solution of the LQR at a minimum of 10 Hz on the vehicle using a common ARM processor. We show four successful experiments: (i)) controlling at hover state with large disturbances; (ii)) tracking along a trajectory; (iii)) tracking along an infeasible trajectory; (iv)) tracking along a trajectory with disturbances. All experiments were done using only onboard visual inertial state estimation and LQR computation. To the best of our knowledge, this is the first implementation and evaluation of a state-depending LQR capable of onboard computation while providing this amount of versatility and performance.

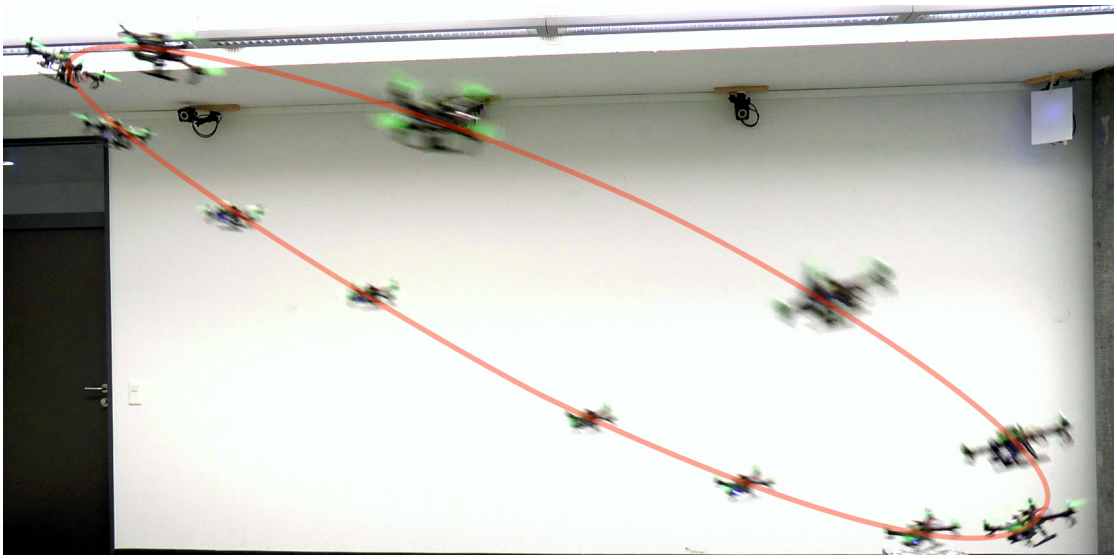


Figure H.1: Flying an ellipse with state-dependent LQR control. Our quadrotor is based on a Qualcomm Snapdragon Flight board with the Pixhawk PX4 flight stack, onboard visual inertial odometry and commercial electronic speed controllers, motors, propeller and frame. It has a take-off weight of 255 g.

H.1 Introduction

H.1.1 Motivation

With the recent advances in control of Micro Aerial Vehicles (MAVs), it is possible to use them in a wide variety of applications, ranging from search and rescue scenarios, observation, hobbyist drone racing to transportation and delivery. Many of these scenarios include difficult, cluttered environments, such as post-disasters or emergency situations. In all these applications, robust control of the vehicle plays a crucial role for success, which becomes particularly challenging considering the quadrotor as an under-actuated system with sublime agility and simplicity.

Many groups have shown examples of complex, agile maneuvers [63, 148, 158, 234, 70] which rely on excellent tracking of a given trajectory. This tracking control is often based on assumptions and simplifications, exploiting the time-scale separation between the translational and rotational dynamics, but introducing certain limitations.

Richard Bellmans work on dynamic programming [22, 21] and Rudolf Kalmans optimal control theory [106] formed the realm of the Linear Quadratic Regulator (LQR), a powerful tool towards optimal control. Until today, LQR has often only been used to solve parts of the control problem on MAVs, such as attitude stabilization, but never to provide a full-state controller unifying rotational and translational states in one feedback loop. Exactly this unified control scheme is where LQR can exceed other approaches.

H.1.2 Related Work & Problem Statement

State-of-the-art approaches in control and trajectory generation exploit the time-scale separation and differential flatness of these systems [150]. They rely heavily on cascaded control schemes [95] and *decouple* the rotational and translational dynamics via a geometric tracking controller as described in [122]. These control schemes leverage the desired acceleration to prescribe the attitude, which therefore is directly affected by the over-imposed position and velocity control. Direct control over the attitude itself is therefore lost and the demanded acceleration is subject to several limitations in change rate, magnitude, and direction.

On the other hand, many approaches on LQR [233] linearize the system at a given stable state [195] or use a precomputed library of LQR gains [194]. None of these approaches is capable of adjusting to the full state space, varying state or input costs, or changing system parameters at execution time. Moreover, they often separate orientation and position control, similarly to the aforementioned geometric control. Also for Model Predictive Control (MPC), this decoupling is used to simplify the problem [191, 109].

A reason for this is surely the difficulty of applying an LQR scheme to a system where inputs (body-frame) and states (world-frame) do not share the same reference coordinate frame. In the example of most MAVs, this means that the attitude radically changes the linearization, even in the simplified case of a quadrotor where often only the yaw state is concerned. This necessitates the recomputation of the LQR whenever an attitude change occurs, or in simplified cases at a certain rate, to handle this state-dependent linearization.

All of these methods make use of the under-actuation property of the system, splitting the control problem into multiple subproblems. Especially leveraging the desired acceleration to define the attitude of the vehicle leads to complications, since it disregards the attitude dynamics and therefore requires planned smooth trajectories which must also account for actuator saturations. One specific problem arising from this control scheme is the difficulty to distinguish between actually desired negative downward acceleration (wanted), and erroneously large downward acceleration from large position errors (unwanted). This makes it difficult to use one single control algorithm for many different maneuvers, while keeping the interface and tuning possibilities simple and intuitive for the user. Neither could varying system-and-control parameters be handled without significant complications.

H.1.3 Contribution

We propose a linear quadratic regulator that unifies the control of translational and rotational dynamics of a quadrotor and mitigates many of the aforementioned shortcomings. It vastly simplifies the tuning of the system, since the tuning parameters are costs that directly relate to state errors, weighting *all* system states in relation to each other. To

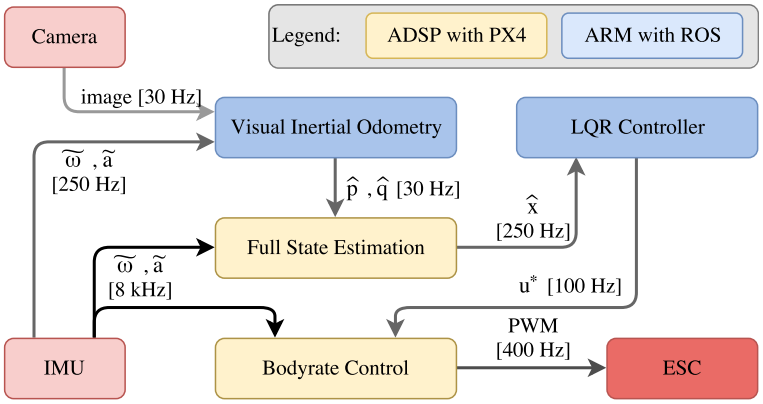


Figure H.2: Overeview of the control architecture implementation on the Snapdragon Flight with measured acceleration $\tilde{\boldsymbol{a}}$ and bodyrates $\tilde{\boldsymbol{\omega}}$, estimated position $\hat{\boldsymbol{p}}$, orientation $\hat{\boldsymbol{q}}$ and full state $\hat{\boldsymbol{x}}$, and desired inputs $\boldsymbol{u}^*$

achieve this, we leverage the formulation of dynamics, reference states, trajectories, and costs to implement a state-dependent LQR. This provides a robust way to control such a MAV not only in the reduced static state space (hover) but also along dynamic state sequences (trajectories). The shown formulation of the dynamics can be extended with a more detailed state description, for example the dynamics of inputs or even virtual states, like the integral of position error. We focus here on the discrete infinite time horizon LQR solution, but our linearization method translates without restrictions to iterative LQR or other model predictive control schemes.

In multiple experiments, we show the robustness of our controller in various situations, such as disturbance from hover, reference jumps, tracking of feasible as well as infeasible trajectories and disturbance during such tracking. We use a small MAV as visible in Fig. H.1 with all computation running onboard.

H.1.4 Structure of the Paper

We first give an overview of the control architecture, in Sec. H.2.1, since this defines our system dynamics formulation, which is described in the following Sec. H.2.2. The LQR control principle is outlined in Sec. H.2.3 with a detailed explanation of the linearization of this system in Sec. H.2.4. We explain the tracking scheme used for the experiment in Sec. H.2.5. The experiment setup, platform and control architecture is specified in Sec. H.3. We show our results in Sec. H.4 and comment our findings the discussion Sec. H.5 followed by the conclusion Sec. H.6.

H.2 Methodology

H.2.1 Control Hierarchy

Most quadrotors are equipped with an Inertial Measurement Unit (IMU) to get high-frequency intrinsic measurements of the rotational velocity and linear acceleration to stabilize the rotational dynamics, and additional extrinsic information to stabilize the translation dynamics. This extrinsic information often comes from GPS, offboard motion capture systems or nowadays also from onboard visual (-inertial) odometry, where we will focus on the last. While the inertial sensors often work at very high frequencies, of up to 8 kHz, to stabilize the fast rotational dynamics, visual odometry operates at a slower rate (typically 30 Hz, visualized in Fig. H.2 and explained in Sec. H.3.2). This is still enough to estimate position and velocity since state-of-the-art visual inertial odometry fuses measurements from the IMU and the camera to output a higher frequency pose estimate.

This choice of sensors is fitted specifically to multirotors. Since they produce thrust forces and drag torques with each rotor, as long as all rotors lie in one plane, one can simplify these forces to a collective thrust and a torque around each body axis. To control acceleration, velocity and position, the quadrotor must adjust its orientation since the collective thrust is always aligned with the body z-axis.

We can now split the control scheme into two domains of fast and slow dynamics. The fast dynamics contains the bodyrates that can be measured directly by the gyroscope in the IMU. The slow dynamics contains the orientation—which is the integral state of the body rates—and the position and velocity—which are integral states of the acceleration—depending on collective thrust, orientation, and gravity. Since the bodyrates are the integral state of the torque produced by different single motor thrusts, they can be controlled with simple feedback loops using the directly available bodyrate measurement. From a control perspective, we intuitively separate these two domains, where the output of the slow dynamics domain are the desired bodyrates and collective thrust. The fast dynamics domain controls the single rotor thrusts to achieve the desired values. This cascade vastly simplifies the control architecture but our approach fully extends to a system where both domains are fused and controlled together. Note that this does not imply decoupling of the rotational and translational dynamics, but using the first derivative of the rotation as an input to the system.

H.2.2 System Dynamics

The system dynamics of a quadrotor can be described as a single rigid-body system. Based on the assumption that the low-level bodyrate controller is faster then our LQR controlled state, we use the bodyrates as input to our system. The bodyrates ${}_B\boldsymbol{\omega}$ are defined in the body-fixed frame, as well as the collective normalized thrust ${}_B\boldsymbol{c} = [0, 0, c]$ whereas gravity $\boldsymbol{g} = [0, 0, g]^T$ is defined in world frame. Furthermore the state $\boldsymbol{x}$ consists

of position $\boldsymbol{p}$, orientation $\boldsymbol{q}$ and velocity $\boldsymbol{v}$.

$$\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}]^T = [p_x, p_y, p_z, q_w, q_x, q_y, q_z, v_x, v_y, v_z]^T \tag{H.1}$$

$$\boldsymbol{u} = [{}_B\boldsymbol{\omega}_{des}, {}_B\, c_{des}]^T = [\omega_{des,x}, \omega_{des,y}, \omega_{des,z}, c_{des}] \tag{H.2}$$

We use quaternions to represent the orientation of the quadrotor to avoid singularities (gimbal lock) due to angle representation. We will drop the prefix ${}_B[\ \ ]$ for the body frame from now on. The system dynamics can then be described in a simplified form as:

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \qquad \dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \qquad \dot{\boldsymbol{v}} = \boldsymbol{g} + \boldsymbol{q} \odot \boldsymbol{c}, \tag{H.3}$$

where $\otimes$ is the quaternion multiplication and $\odot$ is the quaternion rotation as:

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = \boldsymbol{Q}^{\times}(\boldsymbol{q}_1)\boldsymbol{q_2} = \bar{\boldsymbol{Q}}^{\times}(\boldsymbol{q}_2)\boldsymbol{q_1} \tag{H.4}$$

$$\boldsymbol{q}_1 \odot \boldsymbol{v} = \boldsymbol{R}^{\times}(\boldsymbol{q})\boldsymbol{v}. \tag{H.5}$$

$\boldsymbol{Q}^{\times}(\boldsymbol{q})$ is the quaternion multiplication matrix and $\boldsymbol{R}^{\times}(\boldsymbol{q})$ is a rotation matrix with

$$\boldsymbol{Q}^{\times}(\boldsymbol{q}) = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix} \tag{H.6}$$

$$\bar{\boldsymbol{Q}}^{\times}(\boldsymbol{q}) = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & q_z & -q_y \\ q_y & -q_z & q_w & q_x \\ q_z & q_y & -q_x & q_w \end{bmatrix} \tag{H.7}$$

$$\bar{\boldsymbol{Q}}^{\times}(\boldsymbol{q}) = \boldsymbol{Q}^{\times}(\bar{\boldsymbol{q}}) \tag{H.8}$$

$$\bar{\boldsymbol{Q}}^{\times T}(\boldsymbol{q})\boldsymbol{Q}^{\times}(\boldsymbol{q}) = \begin{bmatrix} 1 & \not\vdash \\ \not\vdash & \boldsymbol{R}^{\times}(\boldsymbol{q}) \end{bmatrix}. \tag{H.9}$$

This results in the full equations for velocity and orientation as:

$$\dot{\boldsymbol{v}} = \boldsymbol{g} + \boldsymbol{R}^{\times}(\boldsymbol{q})\boldsymbol{c} = \begin{bmatrix} 2(q_w q_y + q_x q_z)c \\ 2(q_y q_z - q_w q_x)c \\ -g + (1 - 2q_x^2 - 2q_y^2)c \end{bmatrix} \tag{H.10}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\bar{\boldsymbol{Q}}^{\times}\left(\begin{bmatrix}0\\ \boldsymbol{\omega}\end{bmatrix}\right)\boldsymbol{q} = \frac{1}{2}\begin{bmatrix}-\omega_x q_x - \omega_y q_y - \omega_z q_z\\ \omega_x q_w + \omega_z q_y - \omega_y q_z\\ \omega_y q_w - \omega_z q_x + \omega_x q_z\\ \omega_z q_w + \omega_y q_x - \omega_x q_y\end{bmatrix} \quad . \tag{H.11}$$

H.2.3 Linear Quadratic Regulator

A Linear Quadratic Regulator provides the optimal solution for a given linear time-invariant system

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{H.12}$$

at the reference state $\boldsymbol{x}_0$ and input $\boldsymbol{u}_0$ with respect to a user-defined quadratic cost given by the two positive-definite cost matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. We define the cost-to-go for the infinite-time solution as:

$$\boldsymbol{J}(\boldsymbol{x}, \boldsymbol{u}) = \int_0^{\infty} \tilde{\boldsymbol{x}}^T \boldsymbol{Q}\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{u}}^T \boldsymbol{R}\tilde{\boldsymbol{u}}\ dt \tag{H.13}$$

with the errors $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_0$ and $\tilde{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u}_0$. Assume that the optimal cost-to-go is of the quadratic form $\boldsymbol{J}^*(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}$. The solution for $\boldsymbol{P}$ can be found using the Continuous Algebraic Riccati Equation

$$0 = \boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^T\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{Q} \tag{H.14}$$

and results in

$$\boldsymbol{u} = \boldsymbol{u}_0 + \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{x}_0) \qquad \text{with } \boldsymbol{K} = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} \tag{H.15}$$

as the optimal feedback-policy. To solve the problem, we use a dynamic programming approach as described in [233].

H.2.4 Linearization

To calculate the LQR, the full linearization of the system is needed. Instead of just calculating all partial derivatives, we see from (H.3) that most entries of $\boldsymbol{A}$ and $\boldsymbol{B}$ are

independent:

$$\boldsymbol{A}(\boldsymbol{q},\boldsymbol{u}) = \frac{\partial}{\partial \boldsymbol{q}}\boldsymbol{f}(\boldsymbol{q},\boldsymbol{u}) = \begin{bmatrix} \not\vdash & \not\vdash & \frac{\partial}{\partial \boldsymbol{v}}\dot{\boldsymbol{p}} \\ \not\vdash & \frac{\partial}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} & \not\vdash \\ \not\vdash & \frac{\partial}{\partial \boldsymbol{q}}\dot{\boldsymbol{v}} & \not\vdash \end{bmatrix} \tag{H.16}$$

$$\boldsymbol{B}(\boldsymbol{q},\boldsymbol{u}) = \frac{\partial}{\partial \boldsymbol{u}}\boldsymbol{f}(\boldsymbol{q},\boldsymbol{u}) = \begin{bmatrix} \not\vdash & 0 \\ \frac{\partial}{\partial \boldsymbol{\omega}}\dot{\boldsymbol{q}} & 0 \\ \not\vdash & \frac{\partial}{\partial c}\dot{\boldsymbol{v}} \end{bmatrix}. \tag{H.17}$$

Therefore, we calculate only the required derivatives in the following sections.

Preface: Partial Derivatives w.r.t. Unit Quaternions

Since the orientation is represented with a unit quaternion , this induces a constraint on the respective states so that $\|\boldsymbol{q}\| = 1$. When deriving a function of this unit quaternion, the constraint does no longer hold for the derivative. To make the constraint generally valid, we discard it for the general quaternion $\boldsymbol{q}$ and enforce a specific unit quaternion $\boldsymbol{q}_u = \boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1}$. The derivation then yields:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{q}} f\left(\boldsymbol{q}_u\right) &= \frac{\partial}{\partial \boldsymbol{q}_u} f\left(\boldsymbol{q}\right) \cdot \frac{\partial}{\partial \boldsymbol{q}} \left(\boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1}\right) \\ \frac{\partial}{\partial \boldsymbol{q}} \left(\boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1}\right) &= \frac{\partial}{\partial \boldsymbol{q}} \boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1} + \boldsymbol{q} \cdot \frac{\partial}{\partial \boldsymbol{q}} \|\boldsymbol{q}\|^{-1} \\ &= \|\boldsymbol{q}\|^{-1} \cdot \not\Vdash - \boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-2} \cdot \frac{\partial}{\partial \boldsymbol{q}} \|\boldsymbol{q}\| \\ &= \|\boldsymbol{q}\|^{-1} \left(\not\Vdash - \|\boldsymbol{q}\|^{-2} \boldsymbol{q}\boldsymbol{q}^T\right) \end{aligned} \tag{H.18}$$

Position

As in (H.3), the position is the integral of the velocity in the same frame, all partial derivatives are zero except for the

$$\frac{\partial}{\partial \boldsymbol{v}}\dot{\boldsymbol{p}} = \not\Vdash. \tag{H.19}$$

Orientation

From (H.3), we can see that it only depends on the orientation $\boldsymbol{q}$ and the bodyrates $\boldsymbol{\omega}$. From this, we get for the partial derivative with respect to the orientation:

$$\frac{\partial}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} = \frac{1}{2}\begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \cdot \frac{\not\vdash - \|\boldsymbol{q}\|^{-2}\boldsymbol{q}\boldsymbol{q}^T}{\|\boldsymbol{q}\|} \tag{H.20}$$

and similar for the derivation with respect to bodyrates:

$$\frac{\partial}{\partial \boldsymbol{\omega}}\dot{\boldsymbol{q}} = \frac{1}{2}\begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & -q_y \\ q_z & q_w & q_x \\ -q_y & q_x & q_w \end{bmatrix}. \tag{H.21}$$

Velocity

From (H.3), we also get the partial derivatives from the quaternion rotation function and the thrust:

$$\frac{\partial}{\partial \boldsymbol{q}}\dot{\boldsymbol{v}} = 2c\begin{bmatrix} q_y & q_z & q_w & q_x \\ -q_x & -q_w & q_z & q_y \\ q_w & -q_x & -q_y & q_z \end{bmatrix} \cdot \frac{\not\vdash - \|\boldsymbol{q}\|^{-2}\boldsymbol{q}\boldsymbol{q}^T}{\|\boldsymbol{q}\|} \tag{H.22}$$

$$\frac{\partial}{\partial c}\dot{\boldsymbol{v}} = \begin{bmatrix} (q_w q_y + q_x q_z) \\ (q_y q_z - q_w q_x) \\ (q_w^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix}. \tag{H.23}$$

H.2.5 Trajectory Tracking Scheme

To track the reference state along a predefined trajectory, one could use multiple tracking schemes, for example:

- **temporal tracking** where the reference state $\boldsymbol{x}_0(t)$ is chosen based on the passed time,
- **spatial tracking** where the reference state $\boldsymbol{x}_0(\boldsymbol{p})$ closest to the position of the system is chosen.

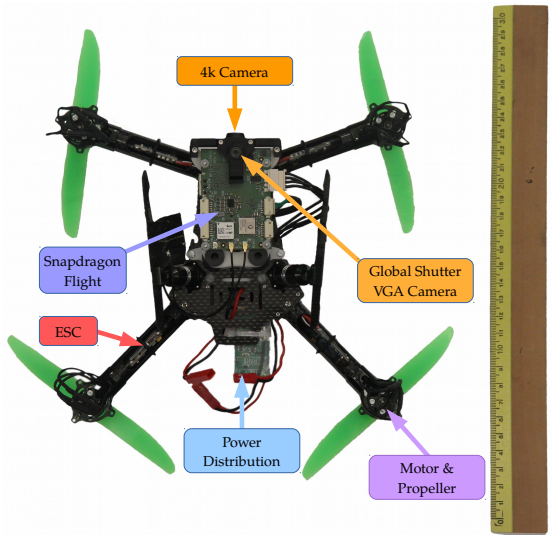


Figure H.3: Our Quadrotor seen from below with a 30 cm ruler. The Snapdragon Flight is visible on top with the global shutter camera. Four "DYS SN20A" ESCs are placed in the arms of the "RKH 250" frame to feed the "DYS BX1306-3100kV" motors with "Gemfan 5030" propellers.

Since temporal tracking requires the system to follow the trajectory with exact computed timing, it is often less robust against disturbances and model uncertainties. Therefore, we chose spatial tracking, similarly to the approach in [7]. In each control loop, we search on the remaining trajectory for the reference state that minimizes the spatial distance to the actual vehicle position.

H.3 Experimental Setup

H.3.1 Experiment Platform

Our experiment platform is designed to be lightweight and easy to operate. We achieved this at a total take-off weight of 255 g. The combination of such low inertia and a carbon fiber frame make it extremely robust. It consist of consumer-grade electronic speed controller (ESC), motors, propellers, and a frame with 250 mm diagonal motor distance. Using this components, it can deliver a maximal thrust-to-weight ratio of ~ 2. As flight controller, we use a Snapdragon Flight[1] single-board solution, visible in Fig. H.3. It includes a computational unit, IMU, front-facing color camera at a 4k-resolution and down-facing global-shutter gray-scale camera at VGA resolution.

[1] Qualcomm: developer.qualcomm.com/software/machine-vision-sdk

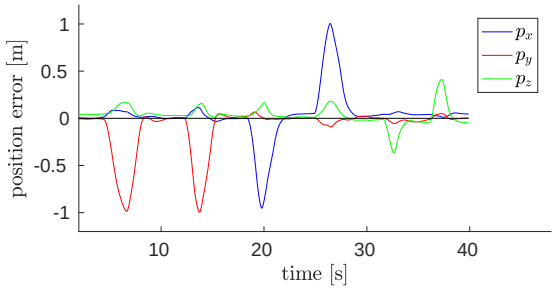


Figure H.4: Position error during a hover task. The vehicle was disturbed by pulling it away from its reference point as seen in the accompanying video.

H.3.2 Control Architecture

The main software is split into a fast low-level controller for the bodyrates and our superimposed high-level LQR controller, as already described in Sec H.2.1. This architecture is illustrated in Fig. H.2. On the Snapdragon Flight's Application Digital Signal Processor (aDSP); the low-level controller runs in the form of the PX4[2] flight stack. This controller reads the IMU onboard the Snapdragon Flight with 8 kHz and computes the PWM signal for the ESC from the desired bodyrates and collective thrust.

The high-level controller runs on the main processor (2.26 GHz quad-core ARM) using Linaro Linux[3] provided by Intrinsyc[4] with ROS[5]. To get an accurate state estimate, we use a Visual Inertial Odometry (VIO) by Qualcomm. The IMU readings are down-sampled and forwarded to the VIO pipeline at 250 Hz together with a VGA gray-scale image from the down-facing global-shutter camera at 30 Hz. The pose estimate after each camera frame is fed back to the PX4 flight stack, which fuses it together with IMU signals to get a low drift, high frequency state estimate at 250 Hz.

Our proposed LQR controller uses this estimate to compute the target bodyrates and collective thrust at 100 Hz. Furthermore, it recalculates the cost-optimal gain matrix $\boldsymbol{K}$ from (H.15) at 10 Hz using the linearization of the system at the most recent estimated state. To chose the tracking reference, we use the the spatial tracking scheme from Sec. H.2.5 due to better handling of model uncertainties and disturbances. All experiments were done with the quadratic cost terms set to $\boldsymbol{Q} = diag([100, 100, 100, 1, 1, 1, 1, 10, 10, 10])$ and $\boldsymbol{R} = diag([1, 5, 5, 0.1])$, while the inputs were limited at $\|{}_B\boldsymbol{\omega}_{des}\| \leq 2\,\mathrm{rad\,s^{-1}}$ and $2\,\mathrm{m\,s^{-2}} \leq_B c_{des} \leq 18\,\mathrm{m\,s^{-2}}$ for all experiments.

[2]PX4 flight stack: www.px4.io
[3]Linaro: www.linaro.org
[4]Intrinsic: www.intrinsyc.com
[5]Robot Operating System: www.ros.org

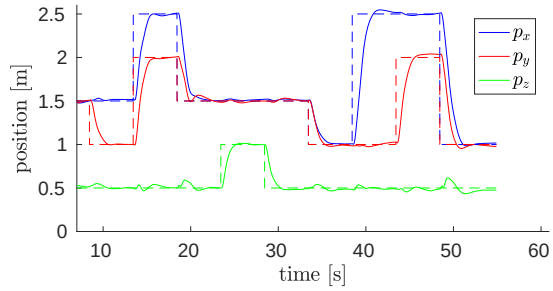


Figure H.5: Position during step changes in the reference all 5 seconds. The dashed line marks the reference, while the full line marks the estimate.

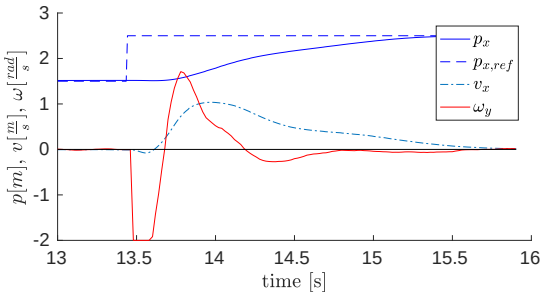


Figure H.6: Input ω_y with reference $p_{x,ref}$, position p_x and velocity v_x during step changes in the reference.

H.3.3 Experiment Description

We perform four different experiments to show basic stability of our controller: hover with reference jumps and disturbances (H.3.3); tracking a feasible minimum-snap trajectory (H.3.3); tracking an infeasible trajectory (H.3.3); disturbing the vehicle during tracking a feasible trajectory (H.3.3). For all experiments, we use onboard VIO only, without any external motion capture system. Because the floor and walls provide practically no texture, we place some textured carpets in the room.

Hover with Reference Jumps and Disturbances

In this experiment we let the vehicle hover and apply step-changes to the reference and disturbances by pulling the vehicle from its hover position. While the reference is momentarily constant, we still apply the state-dependent LQR to adjust to the estimated vehicle state. No trajectory or feasibility checks are needed, since the pure state-feedback is enough to stabilize the vehicle around and towards the reference.

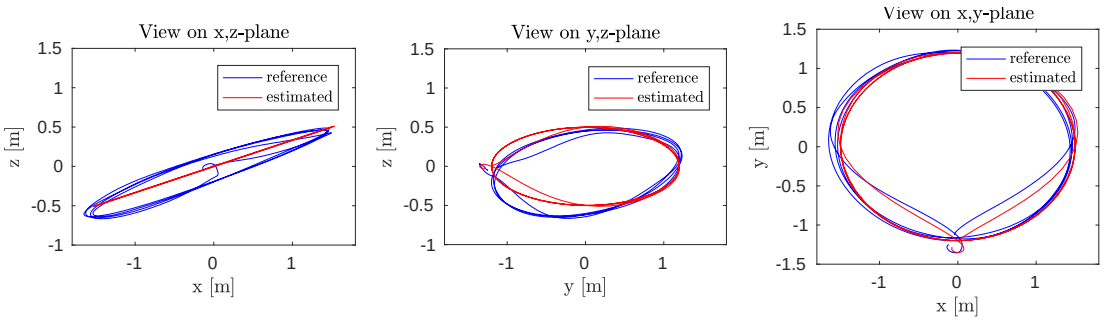


Figure H.7: View from each world axis on a feasible minimum-snap trajectory with $v_{max} = 3.0\,\mathrm{m\,s^{-1}}$ executed using only VIO and our state-dependent LQR. Note the error in z-direction due to thrust uncertainties.

Tracking a Feasible Trajectory

We generate a feasible trajectory using the minimum-snap approach from [148]. For this, we set four waypoints at positions $\boldsymbol{p}_{ms0} = [0, -1.2, 0]^T$, $\boldsymbol{p}_{ms1} = [1.5, 0, 0.5]^T$, $\boldsymbol{p}_{ms2} = [0, 1.5, 0]^T$, $\boldsymbol{p}_{ms3} = [-1.5, 0, 0.5]^T$ and compute a minimum-snap trajectory sampled with $dt = 0.1\,\mathrm{s}$. We limit the velocity to $3.0\,\mathrm{m\,s^{-1}}$. This results in a trajectory close to an ellipse, except for the start and end phase.

Tracking an Infeasible Trajectory

In this experiment, we show the stability of the controller with respect to infeasible references. We generate a perfectly elliptical trajectory, parametrized with respect to the angle α by $\boldsymbol{p}_{ell}(\alpha) = [r_x \sin(\alpha), -r_y \cos(\alpha), r_z \sin(\alpha)]^T$ with $r_x = 1.5\,\mathrm{m}$, $r_y = 1.2\,\mathrm{m}$, $r_z = 0.5\,\mathrm{m}$ and a sample with $d\alpha = 2\pi/36$. The velocity is set to point from one reference point to the next one with a magnitude of $2.0\,\mathrm{m\,s^{-1}}$. The reference orientation is perfectly horizontal as in hover. Note that, therefore, the velocity is discontinuous, the acceleration is infeasible, and no feed-forward part exists.

Tracking with Disturbance

We generate a feasible trajectory as in H.3.3 but with a lower maximal velocity of $1.0\,\mathrm{m\,s^{-1}}$. During execution, we stop the vehicle by holding or pulling it away from the trajectory. Because of the spatial tracking, it continues along the trajectory without skipping any segments, which instead could happen in temporal tracking as mentioned earlier (H.2.5).

H.4 Results

All experiments are visible in the accompanying video at: https://youtu.be/8OVsJNgNfa0

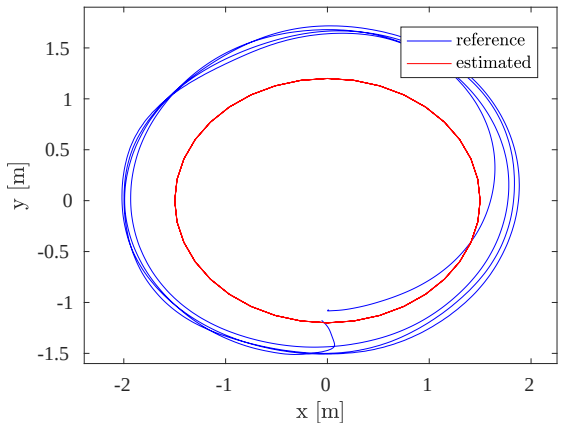


Figure H.8: Stabilizing the vehicle along an unfeasible trajectory. Note the deviation outward of the ellipse due to unfeasible velocity and attitude reference, resulting in a error of ∼0.5 m.

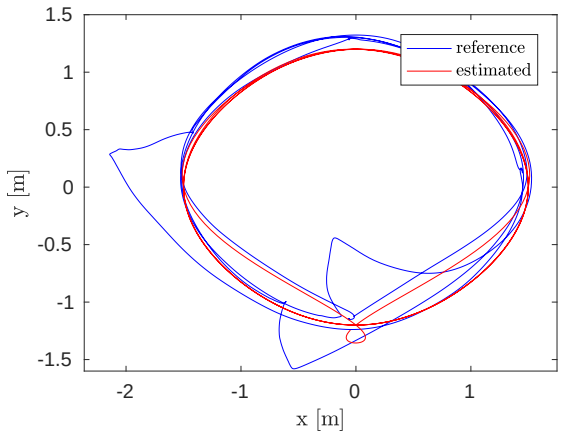


Figure H.9: Trajectory tracked with a human operator disturbing (pulling) the vehicle. Note the convergence back to the trajectory without skipping segments due to spatial tracking.

H.4.1 Hover with Reference Jumps and Disturbances

First, we note the non-zero static error in position tracking, which results from model uncertainties and the lack of an integrative part in the controller. In Fig. H.4, we can see the position error due to a large disturbance by a human. Note how the disturbance is rejected with very little overshoot. Another property is the bounded velocity due to position- and velocity-error saturation. In Fig. H.5, we depict the reaction to jumps in the reference. Note how small deviations from the z-axis setpoint are caused by large jumps in x and y as a trade-off to lower the quadratic state cost as fast as possible. Additionally, we show the an input in Fig. H.6 during a reference jump. Note that the input jumps due to the instantaneous jump in the reference, but otherwise is smooth even over the updates of the LQR gains at 10 Hz.

H.4.2 Tracking a Feasible Trajectory

While tracking a feasible trajectory introduces no more complexity than our hover experiment except for a moving reference state, we expect the same results as in H.4.1. In fact, we can again observe some small offset from the reference trajectory, as depicted in Fig. H.7, resulting from model uncertainties. The biggest influence is the uncertainty of the collective thrust, since this can change with battery voltage and is not accounted for in the low level controller, resulting in the observable deviation in z-direction (calibrated to minimal z-error in hover at 70% charging state). We can observe this deviations in the first and second plot in Fig. H.7, as well as the non symmetry of it due to gravity. This results in a too low position when moving upward, and a too high position when moving downward. The last plot of Fig. H.7 shows the x, y-plane depicting the horizontal tracking performance with little position error.

H.4.3 Tracking an Infeasible Trajectory

With this experiment, we want to show stability of the controller when the reference is infeasible. We can see in Fig. H.8 how the vehicle deviates outward from the reference trajectory due to the lack of a correct reference attitude and infeasible velocity changes. Still, the controller manages to follow the trajectory and stabilize the vehicle, with a trade-off in positional accuracy.

H.4.4 Tracking with Disturbance

In our last experiment, we show how the spatial tracking performs under significant disturbances during execution, depicted in Fig. H.9. While a time-based tracker would not account for the disturbance and possibly skip segments after a disturbance, our spatial tracker is not influence by catching, holding and pulling the vehicle from the trajectory. The algorithm for spatial tracking simply keeps the reference at the last sample point while we hold the vehicle back. As soon as we release the vehicle, it converges back onto the trajectory and continues tracking without skipping any segments, as visible in our video. Note that this adds to the robustness of the implementation and covers many failure scenarios introduced by disturbance as well as model uncertainties.

H.4.5 Computation Time

Our algorithm computes the solution for the state-dependent LQR at 10 Hz with a mean computation time of $\bar{t}_c = 36\,\text{ms}$ with a standard deviation of $\sigma_c = 11\,\text{ms}$ in an average of $\bar{n}_i = 41$ iterations.

H.5 Discussion

In this work, we showed the fundamental applicability of state-dependent and, therefore, time-varying LQR control on a MAV with real experiments, where other approaches only show LQR control for attitude (linearized around one reference state) or only deliver simulation results. While we did not directly compare to other approaches, we intended to mitigate problems of hierarchical control schemes and show the feasibility of the state-dependent LQR approach using our implementation, proven by our experiments. Two topics remain up for discussion: (H.5.1) optimality under uncertainties due to state estimation and (H.5.2) extensions of our approach.

H.5.1 Separation Principle

Since modern MAVs use visual integral state estimation as in our approach, it is interesting to discuss the stability of our controller in conjunction with such an estimator. Many of these estimators are based on a Kalman filter as in [157, 57]. Due to the separation principle, the combination of such an estimator and an LQR will still be stable and optimal if both individual components are stable. Therefore, this approach is valid for a wide variety of state estimation pipelines.

H.5.2 Extension of our Approach

The true advantage of our controller and linearization is its possible extensions, such as: (i)) integral states to counteract steady-state errors; (ii)) implementation of an actuator dynamics model to cover uncertainties and actuator limitations; (iii)) development of a iterative LQR for planning and model predictive control; (iv)) predictive control for cost-field-based obstacle avoidance; (v)) extension to other platforms. We will build upon our findings and investigate the implementation of these extensions to develop a more elaborate control pipeline.

H.6 Conclusion

We proposed a state-dependent LQR which couples the translational and rotational states of the vehicle, and provides the dynamics and linearization needed. We implemented this approach on a lightweight quadrotor using a low-power Qualcomm ARM platform and run all state estimation, LQR computation and controller onboard. Our implementation is efficient enough to update the LQR gains at 10 Hz and our experiments prove feasibility and robustness with this control approach.

Bibliography

[1] D. Abeywardena, Z. Wang, G. Dissanayake, S.L. Waslander, and S. Kodagoda. "Model-aided state estimation for quadrotor micro air vehicles amidst wind disturbances". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2014.

[2] 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.

[3] 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.

[4] 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.

[5] Gokul Anandayuvaraj. *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.

[6] 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).

[7] Gianluca Antonelli, Elisabetta Cataldi, Filippo Arrichiello, Paolo Robuffo Giordano, Stefano Chiaverini, and Antonio Franchi. "Adaptive Trajectory Tracking for Quadrotor MAVs in Presence of Parameter Uncertainties and External Disturbances". In: *IEEE Trans. Control Sys. Tech.* 26.1 (2018), pp. 248–254. DOI: 10.1109/TCST.2017.2650679.

[8] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman. "The Blackbird Dataset: A large-scale dataset for UAV perception in aggressive flight". In: *Int. Symp. Experimental Robotics (ISER)*. 2018.

[9] Amado Antonini. "Pre-integrated dynamics factors and a dynamical agile visual-inertial dataset for UAV perception". MA thesis. Massachusetts Institute of Technology, 2018.

[10] Amado Antonini, Winter Guerra, Varun Murali, Thomas Sayre-McCord, and Sertac Karaman. "The Blackbird UAV dataset". In: *The International Journal of Robotics Research* 39.10-11 (2020), pp. 1346–1364. DOI: 10.1177/0278364920908331.

[11] F. Augugliaro and R. D'Andrea. "Admittance control for physical human - quadrocopter interaction". In: *IEEE Eur. Control Conf. (ECC)*. 2013.

[12] F. Augugliaro, A. P. Schoellig, and R. D'Andrea. "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2012. DOI: 10.1109/IROS.2012.6385823.

[13] 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.

[14] S. Baker, R. Gross, and I. Matthews. "Lucas-Kanade 20 Years On: A Unifying Framework: Part 3". In: *Int. J. Comput. Vis.* (2003).

[15] S. Baker and I. Matthews. "Lucas-Kanade 20 Years On: A Unifying Framework: Part 1". In: *Int. J. Comput. Vis.* (2002).

[16] M. Bangura and R. Mahony. "Real-time Model Predictive Control for Quadrotors". In: *IFAC World Congress* (2014). DOI: 10.3182/20140824-6-za-1003.00203.

[17] 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.

[18] 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.

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

[20] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. "Speeded-Up Robust Features (SURF)". In: *Comput. Vis. Image. Und.* (2008). DOI: 10.1016/j.cviu.2007.09.014.

[21] Richard Bellman. "Dynamic programming". In: *Science* 153.3731 (1966), pp. 34–37.

[22] Richard Bellman. "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society* 60.6 (1954), pp. 503–515.

[23] 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.

[24] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. Athena scientific Belmont, MA, 1995.

[25] 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.

[26] H.G. Bock and K.J. Plitt. "A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems". In: *IFAC World Congress* 17.2 (1984), pp. 1603–1608. DOI: 10.1016/S1474-6670(17)61205-9.

[27] Samit Bouabdallah, Pierpaolo Murrieri, and Roland Siegwart. "Design and Control of an Indoor Micro Quadrotor". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2004. DOI: 10.1109/ROBOT.2004.1302409.

[28] Samit Bouabdallah, Andre North, and Roland Siegwart. "PID vs LQ control techniques applied to an indoor micro quadrotor". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2004. DOI: 10.1109/IROS.2004.1389776.

[29] Samit Bouabdallah and Roland Siegwart. "Full control of a quadrotor". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2007. DOI: 10.1109/IROS.2007.4399042.

[30] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[31] Dario Brescianini and Raffaello D'Andrea. "Tilt-prioritized quadrocopter attitude control". In: *IEEE Transactions on Control Systems Technology* (2018). URL: https://ieeexplore.ieee.org/document/8556372.

[32] Dario Brescianini, Markus Hehn, and Raffaello D'Andrea. "Quadrocopter Pole Acrobatics". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2013. DOI: 10.1109/IROS.2013.6696851.

[33] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. "The EuRoC micro aerial vehicle datasets". In: *Int. J. Robot. Research* 35 (10 2015), pp. 1157–1163.

[34] 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).

[35] 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.

[36] Bárbara Barros Carlos, Tommaso Sartor, Andrea Zanelli, Gianluca Frison, Wolfram Burgard, Moritz Diehl, and Giuseppe Oriolo. "An Efficient Real-Time NMPC for Quadrotor Position Control under Communication Time-Delay". In: *IEEE Int. Conf. Cont. Autom. Robot. Vis. (ICARCV)*. IEEE. 2020, pp. 982–989. DOI: 10.1109/ICARCV50220.2020.9305513.

[37] 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. eprint: https://robotics.sciencemag.org/content/5/38/eaay1246.full.pdf. URL: https://robotics.sciencemag.org/content/5/38/eaay1246.

[38] 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.

[39] Connect Tech Inc. *Quasar Carrier Board*. https://connecttech.com/product/quasar-carrier-nvidia-jetson-tx2/. Accessed: 2021-7-20. Aug. 2019.

[40] CORDIS - European Comission. *AgileFlight*. https://cordis.europa.eu/project/id/864042. Accessed: 2021-7-30.

[41] G. Costante, J. Delmerico, M. Werlberger, P. Valigi, and D. Scaramuzza. "Exploiting Photometric Information for Planning under Uncertainty". In: *Proc. Int. Symp. Robot. Research (ISRR)* (2018), pp. 107–124. DOI: 10.1007/978-3-319-51532-8_7.

[42] Richard W. Cottle and George B. Dantzig. "Complementary pivot theory of mathematical programming". In: *Linear Algebra and its Applications* (1968). DOI: 10.1016/0024-3795(68)90052-9.

[43] *CUDA C++ Programming Guide*. NVIDIA Corporation. 2019.

[44] 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.

[45] Mike Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359.

[46] Frank Dellaert and Michael Kaess. "Factor Graphs for Robot Perception". In: (2017).

[47] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza. "Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2019.

[48] J. Delmerico and D. Scaramuzza. "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots". In: *IEEE Int. Conf. Robot. Autom. (ICRA)* (May 2018).

[49] 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.

[50] *Developer Blog - Faster Parallel Reductions on Kepler*. https://devblogs.nvidia.com/faster-parallel-reductions-kepler/. NVIDIA Corporation. 2014.

[51] 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.

[52] Moritz Diehl, Hans Georg Bock, and Johannes P Schlöder. "A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control". In: *SIAM J. on Control and Optimization* 43.5 (2005), pp. 1714–1736. DOI: 10.1137/S0363012902400713.

[53] DJI. *DJI Digital FPV System*. https://www.dji.com/fpv. Accessed: 2021-7-20.

[54] 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.

[55] J. Dupeyroux, J. Hagenaars, 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.

[56] J. Engel, V. Koltun, and D. Cremers. "Direct Sparse Odometry". 2016. URL: http://arxiv.org/pdf/1607.02565.pdf.

[57] 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 Micro Aerial Vehicle". In: *J. Field Robotics* 33.4 (2016), pp. 431–450.

[58] 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.

[59] 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.

[60] 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.

[61] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. "Dynamic obstacle avoidance for quadrotors with event cameras". In: *Science Robotics* 5.40 (2020).

[62] 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.

[63] 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.

[64] 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.

[65] Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Bock, and Moritz Diehl. "qpOASES: A parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* (Dec. 2014). DOI: 10.1007/s12532-014-0071-1.

[66] 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.

[67] 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.

[68] 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). URL: https://link.springer.com/article/10.1007/s11370-018-00271-6.

[69] 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.

[70] Philipp Foehn, Davide Falanga, Naveen Kuppuswamy, Russ Tedrake, and Davide Scaramuzza. "Fast Trajectory Optimization for Agile Quadrotor Maneuvers with a Cable-Suspended Payload". In: *Robotics: Science and Systems (RSS)*. 2017.

[71] 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. URL: https://robotics.sciencemag.org/content/6/56/eabh1221.

[72] Philipp Foehn and Davide Scaramuzza. "Onboard State Dependent LQR for Agile Quadrotors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018. DOI: 10.1109/ICRA.2018.8460885.

[73] Philipp Foehn et al. "Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight". In: *Science Robotics* (2021). under review.

[74] 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.

[75] 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.

[76] 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.

[77] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. "Appearance-based Active, Monocular, Dense Depth Estimation for Micro Aerial Vehicles". In: *Robotics: Science and Systems (RSS)*. 2014. DOI: 10.15607/RSS.2014.X.029.

[78] W. Förstner and E. Gülch. "A Fast Operator for Detection and Precise Location of Distinct Point, Corners and Centres of Circular Features". In: *Proceedings of the ISPRS Conference on Fast Processing of Photogrammetric Data*. 1987. URL: http://www.ipb.uni-bonn.de/pdfs/Forstner1987Fast.pdf.

[79] F. Fraundorfer and D. Scaramuzza. "Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications". In: *IEEE Robot. Autom. Mag.* (2012). DOI: 10.1109/MRA.2012.2182810.

[80] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. "RotorS—A Modular Gazebo MAV Simulator Framework". In: *Studies in Computational Intelligence*. Springer, 2016, pp. 595–625.

[81] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. "RotorS - A modular gazebo MAV simulator framework". In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[82] Guillermo Gallego et al. "Event-based Vision: A Survey". In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2020).

[83] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. "Learning to fly by crashing". In: *International Conference on Intelligent Robots and Systems, IROS*. 2017, pp. 3948–3955.

[84] 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.

[85] M. Geisert and N. Mansard. "Trajectory generation for quadrotor based systems using numerical optimal control". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016. DOI: 10.1109/ICRA.2016.7487460.

[86] Wojciech Giernacki, Mateusz Skwierczyński, Wojciech Witwicki, Pawel Wroński, and Piotr Kozierski. "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.

[87] Alessandro Giusti et al. "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots". In: *IEEE Robot. Autom. Lett.* 1.2 (2016).

[88] 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 International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[89] Drew Hanover, Elia Kaufmann, Philipp Foehn, 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.

[90] C. R. Hargraves and S. W. Paris. "Direct Trajectory Optimization Using Nonlinear Programming and Collocation". In: *J. Guidance, Control, and Dynamics* (1987). DOI: 10.2514/3.20223.

[91] C. Harris and M. Stephens. "A combined corner and edge detector". In: *In Proc. of Fourth Alvey Vision Conference*. 1988.

[92] 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.

[93] Stephen Hawking, Eddie Redmayne, Kip S. Thorne, and Lucy Hawking. *Brief Answers to the Big Questions: The final book from Stephen Hawking*. John Murray, 2020. ISBN: 9781473695993.

[94] 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.

[95] Markus Hehn and Raffaello D'Andrea. "Quadrocopter trajectory generation and control". In: *IFAC World Congress*. Vol. 18. 1. 2011, pp. 1485–1491.

[96] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis. "Camera-IMU-based localization: Observability analysis and consistency improvement". In: *Int. J. Robot. Research* 33.1 (2014), pp. 182–201.

[97] 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.

[98] B. Houska, H.J. Ferreau, and M. Diehl. "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization". In: *Optimal Control Applications and Methods* 32.3 (2011). DOI: 10.1002/oca.939.

[99] B. Houska, H.J. Ferreau, and M. Diehl. "An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range". In: *Automatica* 47.10 (2011). DOI: 10.1016/j.automatica.2011.08.020.

[100] M. Hwangbo, J. Kim, and T. Kanade. "Inertial-aided KLT feature tracking for a moving camera". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2009. DOI: 10.1109/IROS.2009.5354093.

[101] 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.

[102] 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).

[103] 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).

[104] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. "Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning". In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 2539–2544. DOI: 10.1109/LRA.2018.2808368.

[105] 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.

[106] R. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *J. Basic Eng.* 82 (1 1960), pp. 35–45.

[107] Rudolf Emil Kalman et al. "Contributions to the theory of optimal control". In: *Bol. soc. mat. mexicana* 5.2 (1960), pp. 102–119.

[108] 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.

[109] Mina Kamel, Kostas Alexis, Markus Achtelik, and Roland Siegwart. "Fast nonlinear model predictive control for multicopter attitude tracking on SO(3)". In: *IEEE Conf. Control Appl. (CCA)*. 2015, pp. 1160–1166. DOI: 10.1109/CCA.2015.7320769.

[110] 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. URL: https://doi.org/10.1109/ICRA.2019.8793631.

[111] 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.

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

[113] 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. URL: https://doi.org/10.1109/TNN.2011.2162110.

[114] Dong Ki Kim and Tsuhan Chen. "Deep neural network for real-time autonomous indoor navigation". In: *arXiv:1511.04668* (2015).

[115] J.S. Kim, M. Hwangbo, and T. Kanade. "Realtime affine-photometric KLT feature tracker on GPU in CUDA framework". In: *Int. Conf. Comput. Vis. Workshops (ICCVW)*. 2009. DOI: 10.1109/ICCVW.2009.5457608.

[116] 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.

[117] M. Koliraov, D. Ta, and F. Dellaert. "Differential Dynamic Programming for Optimal Estimation". In: *IEEE Int. Conf. Robot. Autom. (ICRA)* (2015).

[118] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[119] 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.

[120] *Laird Connectivity*. https://www.lairdconnect.com/. Accessed: 2021-7-20.

[121] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006. URL: http://planning.cs.uiuc.edu.

[122] T. Lee, M. Leoky, and N.H. McClamroch. "Geometric tracking control of a quadrotor UAV on SE(3)". In: *IEEE Conf. Decision Control (CDC)*. 2010, pp. 5420–5425. DOI: 10.1109/CDC.2010.5717652.

[123] 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).

[124] 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).

[125] 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).

[126] H. Liu, M. Chen, G. Zhang, H. Bao, and Y. Bao. "ICE-BA: Incremental, Consistent and Efficient Bundle Adjustment for Visual-Inertial SLAM". In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2018. DOI: 10.1109/CVPR.2018.00211.

[127] S. Liu, K. Mohta, N. Atanasov, and V. Kumar. "Search-Based Motion Planning for Aggressive Flight in SE(3)". In: *IEEE Robot. Autom. Lett.* (2018). DOI: 10.1109/LRA.2018.2795654.

[128] 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).

[129] 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.

[130] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 404–411.

[131] 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.

[132] 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.

[133] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. "Agile Autonomy: Learning High-Speed Flight in the Wild". In: *Science Robotics* (2021). under review.

[134] 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.

[135] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vis.* (2004). DOI: 10.1023/B:VISI.0000029664.99615.94.

[136] B. Lucas and T. Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Int. Joint Conf. Artificial Intell. (IJCAI)*. 1981.

[137] Martin Luessi. *radix*. en. https://www.brainfpv.com/product/radix-fc/. Accessed: 2021-7-20. Dec. 2017.

[138] T. Lupton and S. Sukkarieh. "Visual-Inertial-Aided Navigation for High-Dynamic Motion in Built Environments Without Initial Conditions". In: *IEEE Trans. Robot.* (2012).

[139] R. Madaan et al. "AirSim Drone Racing Lab". In: *PMLR post-proceedings of the NeurIPS 2019's Competition Track* (2020).

[140] Ratnesh Madaan et al. "Airsim drone racing lab". In: *NeurIPS 2019 Competition and Demonstration Track*. PMLR. 2020.

[141] 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.

[142] A. L. Majdik, C. Till, and D. Scaramuzza. "The Zurich Urban Micro Aerial Vehicle Dataset". In: *Int. J. Robot. Research* 36.3 (2017), pp. 269–273.

[143] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[144] 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.

[145] C. D. McKinnon and A. P. Schoellig. "Unscented external force and torque estimation for quadrotors". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2016.

[146] 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.

[147] 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.

[148] 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.

[149] D. Mellinger, N. Michael, and V. Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *Int. J. Robot. Research* (2012). DOI: 10.1177/0278364911434236.

[150] Daniel Mellinger, Nathan Michael, and Vijay Kumar. "Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors". In: *Int. Symp. Experimental Robotics (ISER)*. 2010.

[151] 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.

[152] Kartik Mohta et al. "Fast, autonomous flight in GPS-denied and cluttered environments". In: *J. Field Robot.* 35.1 (2018), pp. 101–120. DOI: 10.1002/rob.21774.

[153] H. Moon et al. "Challenges and implemented technologies used in autonomous drone racing". In: *J. Intell. Service Robot.* (2019). DOI: 10.1007/s11370-018-00271-6.

[154] 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. URL: https://ieeexplore.ieee.org/document/7886372.

[155] 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.

[156] 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).

[157] 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.

[158] M. Mueller, S. Lupashin, and R. D'Andrea. "Quadrocopter ball juggling". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2011, pp. 4972–4978. DOI: 10.1109/IROS.2012.6385963.

[159] M. W. Mueller, M. Hehn, and R. D'Andrea. "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2013. DOI: 10.1109/iros.2013.6696852.

[160] Mark Wilfried Mueller, Markus Hehn, and Raffaello D'Andrea. "A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation". In: *IEEE Trans. Robot.* 31.6 (2015), pp. 1294–1310.

[161] Matthias Müller, Vincent Casser, Neil Smith, Dominik L Michels, and Bernard Ghanem. "Teaching UAVs to Race Using UE4Sim". In: *arXiv:1708.05884* (2017).

[162] 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.

[163] 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).

[164] T. Nägeli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges. "Real-time Motion Planning for Aerial Videography with Dynamic Obstacle Avoidance and Viewpoint Optimization". In: *IEEE Robot. Autom. Lett.* 2.3 (2017). DOI: 10.1109/LRA.2017.2665693.

[165] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges. "Real-time Planning for Automated Multi-View Drone Cinematography". In: *SIGGRAPH*. 2017.

[166] Balazs Nagy, Philipp Foehn, and Davide Scaramuzza. "Faster than FAST: GPU-Accelerated Frontend for High-Speed VIO". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2020.

[167] Fang Nan, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. "Nonlinear MPC for Quadrotor Fault-Tolerant Control". In: *IEEE Robot. Autom. Lett.* 2022.

[168] A. Neubeck and L. Van Gool. "Efficient Non-Maximum Suppression". In: *IEEE Int. Conf. Pattern Recog. (ICPR)*. 2006. DOI: 10.1109/ICPR.2006.479.

[169] 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.

[170] Newswire. *Global Drone Service Market Report 2019: Market is Expected to Grow from USD 4.4 Billion in 2018 to USD 63.6 Billion by 2025, at a CAGR of 55.9%*. Apr. 2019. URL: https://markets.businessinsider.com/news/stocks/global-drone-service-market-report-2019-market-is-expected-to-grow-from-usd-4-4-billion-in-2018-to-usd-63-6-billion-by-2025-at-a-cagr-of-55-9-1028147695.

[171] 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 [cs.RO].

[172] 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.

[173] Barza Nisar, Philipp Foehn, Davide Falanga, and Davide Scaramuzza. "VIMO: Simultaneous Visual Inertial Model-based Odometry and Force Estimation". In: *Robotics: Science and Systems (RSS)*. 2019.

[174] D. Nister, O. Naroditsky, and J. Bergen. "Visual odometry". In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2004.

[175] David A Nix and Andreas S Weigend. "Estimating the mean and variance of the target probability distribution". In: *IEEE Int. Conf. Neural Netw.* 1994.

[176] 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.

[177] 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.

[178] K. Omar. *KFAST: vectorized x86 CPU implementation of the FAST feature detector*. 2006. URL: https://github.com/komrad36/KFAST (visited on 02/27/2020).

[179] D. Oro, C. Fernández, X. Martorell, and J. Hernando. "Work-efficient parallel non-maximum suppression for embedded GPU architectures". In: *Int. Conf. Acoust., Speech, Signal Proc. (ICASSP)*. 2016. DOI: 10.1109/ICASSP.2016.7471831.

[180] D. Palossi et al. "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.

[181] 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.

[182] Adam Paszke et al. "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.

[183] B. Penin, R. Spica, P. Robuffo Giordano, and F. Chaumette. "Vision-Based Minimum-Time Trajectory Generation for a Quadrotor UAV". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017. DOI: 10.1109/IROS.2017.8206522.

[184] Christian Pfeiffer and Davide Scaramuzza. "Human-Piloted Drone Racing: Visual Processing and Control". In: *IEEE Robot. Autom. Lett.* (2021). DOI: 10.1109/LRA.2021.3064282.

[185] Tuan Q. Pham. "Non-maximum Suppression Using Fewer than Two Comparisons per Pixel". In: *Advanced Concepts for Intelligent Vision Systems ACIVS*. 2010. DOI: 10.1007/978-3-642-17688-3_41.

[186] Lev Semenovich Pontryagin, EF Mishchenko, VG Boltyanskii, and RV Gamkrelidze. *The mathematical theory of optimal processes*. Wiley, 1962.

[187] Michael Posa, Cecilia Cantu, and Russ Tedrake. "A direct method for trajectory optimization of rigid bodies through contact". In: *Int. J. Robot. Research* (2014). DOI: 10.1177/0278364913506757.

[188] C. Potena, D. Nardi, and A. Pretto. "Effective Target Aware Visual Navigation for UAVs". In: *Eur. Conf. Mobile Robots (ECMR)*. 2017.

[189] T. Qin, P. Li, and S. Shen. "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator". In: *IEEE Trans. Robot.* July 2018.

[190] 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.

[191] Guilherme V. Raffo, Manuel G. Ortega, and Francisco R. Rubio. "MPC with Nonlinear H_∞ Control for Path Tracking of a Quad-Rotor Helicopter". In: *IFAC World Congress* 41.2 (2008), pp. 8564–8569. DOI: 10.3182/20080706-5-kr-1001.01448.

[192] Suchithra Rajendran and Sharan Srinivas. "Air taxi service for urban mobility: A critical review of recent developments, future challenges, and opportunities". In: *Transportation Research Part E: Logistics and Transportation Review* 143 (Nov. 2020), p. 102090. DOI: 10.1016/j.tre.2020.102090.

[193] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*. Vol. 2. Nob Hill Publishing Madison, WI, 2017.

[194] P. Reist and R. Tedrake. "Simulation-Based LQR-Trees With Input and State Constraints". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2010, pp. 5504–5510. DOI: 10.1109/ROBOT.2010.5509893.

[195] Elias Reyes-Valeria, Rogerio Enriquez-Caldera, Sergio Camacho-Lara, and Jose Guichard. "LQR control for a quadrotor using unit quaternions: Modeling and simulation". In: *Int. Conf. on Electronics, Communications and Computing (CONIELECOMP)*. 2013, pp. 172–178. DOI: 10.1109/conielecomp.2013.6525781.

[196] A. Richards and J. P. How. "Aircraft trajectory planning with collision avoidance using mixed integer linear programming". In: *IEEE Am. Control Conf. (ACC)*. 2002. DOI: 10.1109/ACC.2002.1023918.

[197] 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 [cs.RO].

[198] 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.

[199] 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.

[200] E. Rosten and T. Drummond. "Machine learning for high-speed corner detection". In: *Eur. Conf. Comput. Vis. (ECCV)*. 2006. DOI: 10.1007/11744023_34.

[201] E. Rosten, R. Porter, and T. Drummond. "FASTER and better: A machine learning approach to corner detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2010). DOI: 10.1109/TPAMI.2008.275.

[202] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. "ORB: An Efficient Alternative to SIFT or SURF". In: *Int. Conf. Comput. Vis. (ICCV)*. 2011. DOI: 10.1109/ICCV.2011.6126544.

[203] F. Ruggiero, J. Cacace, H. Sadeghian, and V. Lippiello. "Impedance control of VToL UAVs with a momentum-based external generalized forces estimator". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014.

[204] G. Ryou, E. Tal, and S. Karaman. "Multi-Fidelity Black-Box Optimization for Time-Optimal Quadrotor Maneuvers". In: *Robotics: Science and Systems* (June 2020).

[205] Jianbo S. and Carlo T. "Good features to track". In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)* (1994). DOI: 10.1109/CVPR.1994.323794.

[206] 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.

[207] 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.

[208] Parrot Drone SAS. *Parrot ANAFI Ai*. https://www.parrot.com/en/drones/anafi-ai. Accessed: 2021-7-20.

[209] D. Scaramuzza, F. Fraundorfer, and R. Siegwart. "Real-time Monocular Visual Odometry for On-road Vehicles with 1-point RANSAC". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2009. DOI: 10.1109/ROBOT.2009.5152255.

[210] Jonas Schlagenhauf, Peter Hofmeier, Thilo Bronnenmeyer, Reinhart Paelinck, and Moritz Diehl. "Cascaded Nonlinear MPC for Realtime Quadrotor Position Tracking". In: *IFAC World Congress* 53.2 (2020), pp. 7026–7032. DOI: 10.1016/j.ifacol.2020.12.444.

[211] 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).

[212] *Secondary navigation*. Aug. 2021. URL: https://www.faa.gov/uas/resources/by_the_numbers/.

[213] 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.

[214] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[215] M. Sheckells, G. Garimella, and M. Kobilarov. "Optimal Visual Servoing for differentially flat underactuated systems". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2016.

[216] Malcolm D. Shuster. "Survey of attitude representations". In: *Journal of the Astronautical Sciences* 41.4 (Oct. 1993), pp. 439–517.

[217] Skydio. *Skydio*. July 26, 2021.

[218] 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.

[219] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. "Flightmare: A Flexible Quadrotor Simulator". In: *Conference on Robot Learning*. 2020.

[220] 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.

[221] 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.

[222] S. Spedicato and G. Notarstefano. "Minimum-Time Trajectory Generation for Quadrotors in Constrained Environments". In: *IEEE Transactions on Control Systems Technology* (2018). DOI: 10.1109/TCST.2017.2709268.

[223] R. Spica, P. Robuffo Giordano, and F. Chaumette. "Coupling Active Depth Estimation and Visual Servoing via a Large Projection Operator". In: *Int. J. Robot. Research* 36.11 (2017).

[224] K. Su and S. Shen. "Catching a Flying Ball with a Vision-Based Quadrotor". In: *Int. Symp. Experimental Robotics (ISER)*. 2016.

[225] 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.

[226] 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.

[227] 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.* (2021). arXiv: 2011.11104 [cs.RO].

[228] 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.

[229] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010. ISBN: 9781848829343.

[230] D. Ta, M. Kobilarov, and F. Dellaert. "A factor graph approach to estimation and model predictive control on Unmanned Aerial Vehicles". In: *IEEE Int. Conf. Unmanned Aircraft Syst. (ICUAS)*. 2014.

[231] A. Tagliabue, M. Kamel, S. Verling, R. Siegwart, and J. Nieto. "Collaborative transportation using MAVs via passive force control". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.

[232] 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.

[233] Asma Al-Tamimi, Frank L. Lewis, and Murad Abu-Khalaf. "Discrete-Time Nonlinear HJB Solution Using Approximate Dynamic Programming: Convergence Proof". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.4 (2008), pp. 943–949. DOI: 10.1109/TSMCB.2008.926614.

[234] S. Tang and V. Kumar. "Mixed integer Quadratic Program Trajectory Generation for a Quadrotor With a Cable-Suspended Payload". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2015, pp. 2216–2222. DOI: 10.1109/ICRA.2015.7139492.

[235] The Apache Software Foundation. *NuttX*. https://nuttx.apache.org/. Accessed: 2021-7-20.

[236] The Betaflight Open Source Flight Controller Firmware Project. *Betaflight*. https://github.com/betaflight/betaflight. Accessed: 2021-7-20.

[237] T. Tomic and S. Haddadin. "A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2014.

[238] 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.

[239] Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza. "Data-driven mpc for quadrotors". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3769–3776.

[240] Unity. *Unity Asset Store*. https://assetstore.unity.com/. Accessed: 2021-08-02.

[241] Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. "Direct visual-inertial odometry with stereo cameras". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016.

[242] 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.

[243] Antonio Vitale1, 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.

[244] A. Waechter and L. Biegler. "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming". In: *Mathematical programming* (2006). DOI: 10.1007/s10107-004-0559-y.

[245] D. J. Webb and J. van den Berg. "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2013. DOI: 10.1109/ICRA.2013.6631299.

[246] Douglas Brent West. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, NJ, 2001.

[247] *World's Fastest Drone: Drone Racing League*. https://www.youtube.com/watch?v=dPpQWIA59Wo. 2019.

[248] Andrew Wynn, Milan Vukov, and Moritz Diehl. "Convergence Guarantees for Moving Horizon Estimation Based on the Real-Time Iteration Scheme". In: 59.8 (2014), pp. 2215–2221. DOI: 10.1109/TAC.2014.2298984.

[249] P. Yalamanchili, U. Arshad, Z. Mohammed, P. Garigipati, P. Entschev, B. Kloppenborg, J. Malcolm, and J. Melonakos. *ArrayFire - A high performance software library for parallel computing with an easy-to-use API*. 2015. URL: https://github.com/arrayfire/arrayfire.

[250] B. Yüksel, C. Secchi, H. Bülthoff, and A. Franchi. "A nonlinear force observer for quadrotors and application to physical interactive tasks". In: *IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*. 2014.

[251] C. Zach, D. Gallup, and J. Frahm. "Fast gain-adaptive KLT tracking on the GPU". In: *IEEE Conf. Comput. Vis. Pattern Recog. Workshops (CVPRW)*. 2008. DOI: 10.1109/CVPRW.2008.4563089.

[252] Andrea Zanelli, Greg Horn, Gianluca Frison, and Moritz Diehl. "Nonlinear Model Predictive Control of a Human-sized Quadrotor". In: *IEEE Eur. Control Conf. (ECC)*. Limassol, Cyprus, 2018, pp. 1542–1547. DOI: 10.23919/ECC.2018.8550530.

[253] Andrea Zanelli, Quoc Tran-Dinh, and Moritz Diehl. "A Lyapunov function for the combined system-optimizer dynamics in inexact model predictive control". In: *Automatica* 134 (2021).

[254] 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.

[255] 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.

[256] Z. Zhang and D. Scaramuzza. "A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2018, pp. 7244–7251.

[257] Zichao Zhang and Davide Scaramuzza. "Perception-aware Receding Horizon Navigation for MAVs". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018, pp. 2534–2541.

[258] 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.

[259] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. "Robust and efficient quadrotor trajectory generation for fast autonomous flight". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3529–3536.

[260] Boyu Zhou, Jie Pan, Fei Gao, and Shaojie Shen. "RAPTOR: Robust and Perception-aware Trajectory Replanning for Quadrotor Fast Flight". In: *IEEE Transactions on Robotics* (2021).

[261] A. Z. Zhu, D. Thakur, T. Ozaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. "The Multivehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception". In: *IEEE Robot. Autom. Lett.* 3.3 (July 2018), pp. 2032–2039.

[262] John G Ziegler, Nathaniel B Nichols, et al. "Optimum settings for automatic controllers". In: *trans. ASME* 64.11 (1942).

Philipp Föhn

Date of Birth: 19. July 1991

Education

April 2017–February 2022 **Doctoral Program**, *at the University of Zurich, Department of Informatics*, Robotics and Perception Group, Zurich, Switzerland

September 2015–March 2017 **MSc Robotics, Systems and Control**, *ETH Zurich*, Switzerland

Focus on system modelling, control, algorithmics and implementation (summa cum laude), under tuitition of Prof. Roland Siegwart

Master Thesis supervised by Prof. Davide Scaramuzza:
Fast Trajectory Optimization for Agile Quadrotor Maneuvers with a Cable-Suspended Payload

Semester Thesis supervised by Prof. Davide Scaramuzza:
Impedance Control for Physical Interaction with Quadrotors

September 2011–August 2015 **BSc Mechanical Engineering**, ETH Zurich, Switzerland

Focus on control systems

August 2006–July 2010 **Grammar School**, Kantonsschule Kollegium Schwyz, Switzerland

Focus on physics and mathematics

August 1998–July 2006 **Primary and Secondary School**, Mittelpunktschule Schwyz, Switzerland

Work Experience

September 2016–June 2017 **Coaching of Student Project**, Autonomous Systems Lab, ETH Zurich, Switzerland

Coach of a one-year project with 12 students developing an autonomous race car, supervised by Prof. Roland Siegwart

April 2016–July 2016 **Conceptual Research**, Autonomous Systems Lab, ETH Zurich, Switzerland

Conceptual research on an electric drivetrain for agricultural industry partner with recommendation for the management board, supervised by Prof. Marco Hutter

February 2015–May 2015 **Internship**, ALSTOM Inspection Robotics Zurich, Switzerland

Work on control, planning and user interface for inspection robots

September 2013–August 2015 **AMZ Racing Team**, Academic Motorsport Association Zurich, Switzerland

Development of an electric race car for the biggest international engineering competition *Formula Student*, achieving 1^{st} place in world ranking and a Guiness World Record

September 2014–
August 2015 Chief Technical Officer of electronic systems and team leadership

September 2013–
August 2014 Responsible for vehicle controls

January 2008–December 2014 **Multiple Temporary Jobs**, elmor AG, Schwyz, Switzerland

Work on high-precision counter and scale manufacturing

January 2011 **Workshop Internship**, Victorinox AG, Ibach, Switzerland

Internship with education in basic metal works and workshop practices

August 2010–January 2011 **Temporary Job**, Trafo Betschart AG, Ingenbohl, Switzerland

Manufacturing of high-voltage transformers