Autonomous Robots (2022) 46:307–320
https://doi.org/10.1007/s10514-021-10011-y

AlphaPilot: autonomous drone racing

Philipp Foehn[1] · Dario Brescianini[1] · Elia Kaufmann[1] · Titus Cieslewski[1] · Mathias Gehrig[1] · Manasi Muglikar[1] · Davide Scaramuzza[1]

Received: 8 February 2021 / Accepted: 19 July 2021 / Published online: 19 October 2021

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*.

Keyword Drone racing . Agile flight . Aerial vehicles

1 Introduction

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 Moon et al. (2017, 2019) 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

Video of our approach:https://youtu.be/DGjwm5PZQT8
Talk at RSS 2020:https://youtu.be/k6vGEj1ZZWc
RSS paper:http://rpg.ifi.uzh.ch/docs/RSS20_Foehn.pdf

Philipp Foehn, Dario Brescianini, and Elia Kaufmann have contributed equally.

All authors are with the Robotics and Perception Group.

✉ Philipp Foehn
foehn@ifi.uzh.ch

[1] Dep. of Informatics, Dep. of Neuroinformatics, University of Zurich and ETH, Zurich, Switzerland

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

[2] https://www.nytimes.com/2019/03/26/technology/alphapilot-ai-drone-racing.html

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

raises fundamental challenges in real-time state estimation, perception, planning, and control.

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) Cadena et al. (2016). There exists a variety of VIO algorithms, e.g., Mourikis and Roumeliotis (2007); Bloesch et al. (2015); Qin et al. (2018); Forster et al. (2017a), that are based on feature detection and tracking that achieve very good results in general navigation tasks Delmerico and Scaramuzza (2018). 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 Delmerico et al. (2019).

To overcome this difficulty, several approaches exploiting the structure of drone racing with gates as landmarks have been developed, e.g., Li et al. (2019); Jung et al. (2018); Kaufmann et al. (2018), where the drone locates itself relative to gates. In Li et al. (2019), 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 Jung et al. (2018), 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 Kaufmann et al. (2018) also relies on relative gate data but has the advantage that it works even when no gate is visible. In particular, it uses a CNN to directly infer relative gate poses from images and fuse the results with a VIO state estimate. However, the CNN does not perform well when multiple gates are visible as it is frequently the case for drone racing.

Assuming knowledge of the platform state and the environment, there exist many approaches which can reliably generate feasible trajectories with high efficiency. The most prominent line of work exploits the quadrotor's underactuated nature and the resulting differentially-flat output states Mellinger et al. (2012); Mueller et al. (2015), where trajectories are described as polynomials in time. Other approaches additionally incorporate obstacle avoidance Zhou et al. (2019); Gao et al. (2019) or perception constraints Falanga et al. (2018); Spasojevic et al. (2020). 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 Hehn et al. (2012); Loock et al. (2013); Ryou et al. (2020); Foehn and Scaramuzza (2020). However, while Hehn et al. (2012); Loock et al. (2013) are limited to 2D scenarios and only find trajectories between two given states, Ryou et al. (2020) requires simulation and real-world data obtained on the track, and the method of Foehn and Scaramuzza (2020) is not applicable due to computational constraints.

1.3 Contribution

The approach contributed herein builds upon the work of Kaufmann et al. (2018) 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 Li et al. (2019); Jung et al. (2018); Kaufmann et al. (2018), 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., LaValle (2006), 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 Foehn et al. (2020) by including a more detailed elaboration on our gate corner detection in Sect. 4 with an ablation study in Sect. 8.1, further details on the fusion of VIO and gate detection in Sect. 5, and a description of the path parameterization in Sect. 6, completed by an ablation study on the planning horizon length in Sect. 8.3.

2 AlphaPilot race format and drone

2.1 Race format

From more than 400 teams that participated in a series of qualification tests including a simulated drone race Guerra et al. (2019), 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.

2.2 Drone specifications

All teams were provided with an identical race drone (Fig. 1) that was approximately 0.7 m in diameter, weighed 3.4kg, and had a thrust-to-weight ratio of 1.4. The drone was equipped with a NVIDIA Jetson Xavier embedded computer for interfacing all sensors and actuators and handling all computation for autonomous navigation onboard. The sensor suite included two $\pm 30°$ forward-facing stereo camera pairs (Fig. 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 1. The drone was equipped with a flight controller that controlled the total thrust f along the drone's z-axis (see Fig. 2) and the angular velocity, $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$, in the body-fixed coordinate frame $\mathcal{B}$.

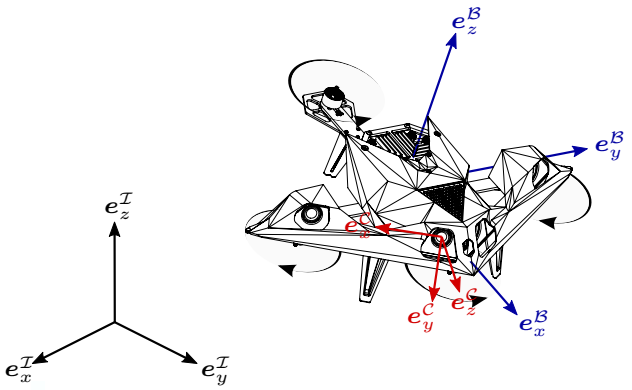

Fig. 2 Illustration of the race drone with its body-fixed coordinate frame $\mathcal{B}$ in blue and a camera coordinate frame $\mathcal{C}$ in red

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 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 Kai et al. (2017). 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. 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 $\mathcal{I}$ Shuster (1993). 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 $\boldsymbol{q} = (q_w, \tilde{\boldsymbol{q}})$ Shuster (1993). The drone's equations of motion are

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

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

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 $\mathcal{B}$, $\boldsymbol{D} = \mathrm{diag}(d_x, d_y, 0)$ 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 Shuster (1993). The drag coefficients

Table 1 Sensor specifications

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

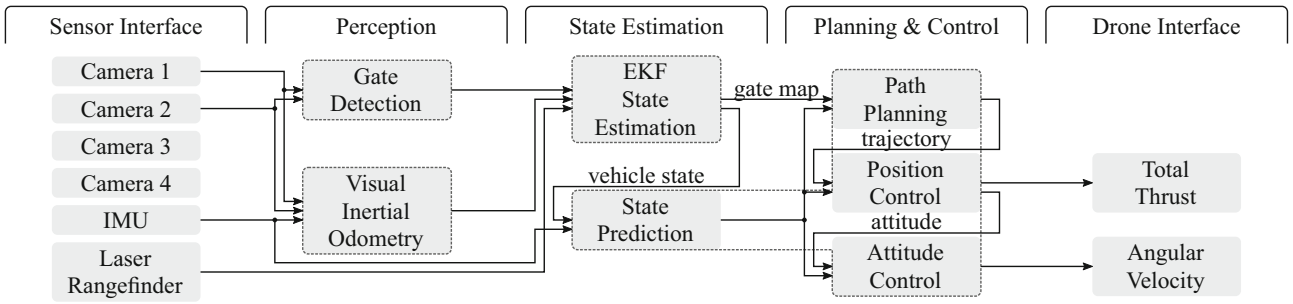

Fig. 3 Overview of the system architecture and its main components. All components within a dotted area run in a single thread

were identified experimentally to be $d_x = 0.5\,\mathrm{kg/s}$ and $d_y = 0.25\,\mathrm{kg/s}$.

3 System overview

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

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 Sect. 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 Bloesch et al. (2015) 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.

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 Sect. 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.

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 Sect. 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 Sect. 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.

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 60Hz, whereas the VIO thread only achieves approximately 35Hz. 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 50Hz. To achieve this, the planning and control thread includes the state prediction which compensates for latencies introduced by the VIO.

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. 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. 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 Cao et al. (2017). These are vector fields, which, in our case, are defined along the edges of the gates, and point from one corner to the next corner of the same gate, see column three in Fig. 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.

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 Fig. 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.

4.1.1 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

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

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.

4.1.2 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

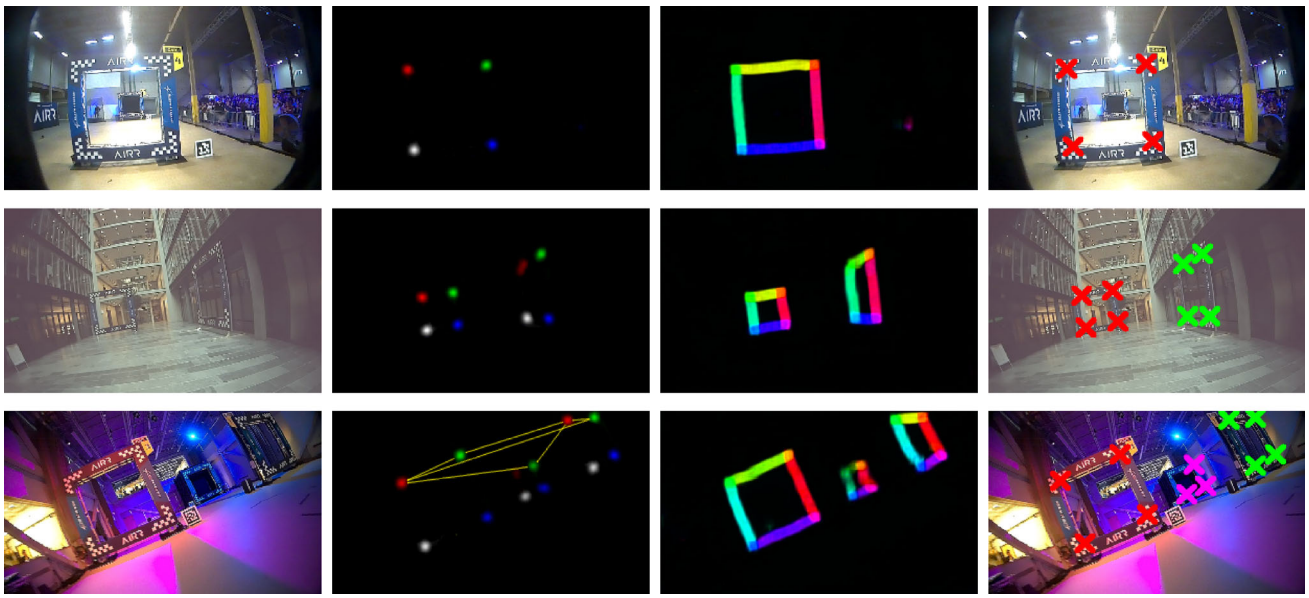

Fig. 4 The gate detection module returns sets of corner points for each gate in the input image (fourth column) using a two-stage process. In the first stage, a neural network transforms an input image, $\boldsymbol{I}_{w\times h\times 3}$ (first column), into a set of confidence maps for corners, $\boldsymbol{C}_{w\times h\times 4}$ (second column), and Part Affinity Fields (PAFs) Cao et al. (2017), $\boldsymbol{E}_{w\times h\times(4\cdot 2)}$ (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 Sect. 4.2. Best viewed in color (Color figure online)

gate. Then, the part affinity field, $\boldsymbol{E}^*_{(k,l)}(\boldsymbol{s})$, is defined as:

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

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.

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. 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

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

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 Cao et al. (2017), finding this optimal assignment corresponds to a K-dimensional matching problem that is known to be NP-Hard West (2001). Following Cao et al. (2017), 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):

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

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

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

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 (7) and (8) enforce that no two edges share the same corner. Above optimization problem can be solved using the Hungarian method Kuhn (1955), 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.

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 changes in brightness, hue, contrast and saturation.

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 Ronneberger et al. (2015) 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.

5 State estimation

The nonlinear measurement models of the VIO, gate detection, and laser rangefinder are fused using an EKF Kalman (1960). 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

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

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

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

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:

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

where $\boldsymbol{Q}$ is the random walk process noise. For each measurement 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

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

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, $\|\boldsymbol{q}_{\mathcal{IV}}\| \overset{!}{=} 1$. 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 Forster et al. (2017b). 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

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

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

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

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

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

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

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.

5.1.1 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 Szeliski (2010) 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

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

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

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

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

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

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

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. 2). The Jacobian with respect to the EKF's state space is derived using the chain rule,

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

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 (14).

5.1.2 Gate correspondences

The gate detection (see Fig. 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 (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

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

5.1.3 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

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

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 (14).

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.

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 Bertsekas (1995), it can be shown that the optimal control input is bang-bang in acceleration, i.e., has the form

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

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 Maurer (1977)

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

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 (23) or (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 (23) or (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. 5 depicts the position and velocity of an example time-optimal motion primitive.

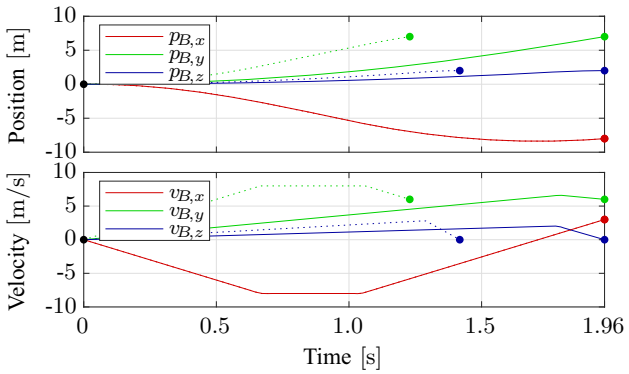

Fig. 5 Example time-optimal motion primitive starting from rest at the origin to a random final position with non-zero final velocity. The velocities are constrained to $\pm 7.5\,\mathrm{m/s}$ and the inputs to $\pm 12\,\mathrm{m/s^2}$. The dotted lines denote the per-axis time-optimal maneuvers

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 exis-

tence 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 Bertsekas (1995) 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.

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 (1) and (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.,

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

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

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

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

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

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

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 (28), yielding

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

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

7 Control

This section presents a control strategy to track the near time-optimal path from Sect. 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.

7.1 Position control

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

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

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

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

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 (31) into the translational dynamics (1), the total thrust, f, is computed to be

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

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 nonlinear attitude controller similar to Brescianini and D'Andrea (2018) 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:

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

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.

8 Results

The proposed system was used to race in the *2019 AlphaPilot* championship race. The course at the championship race

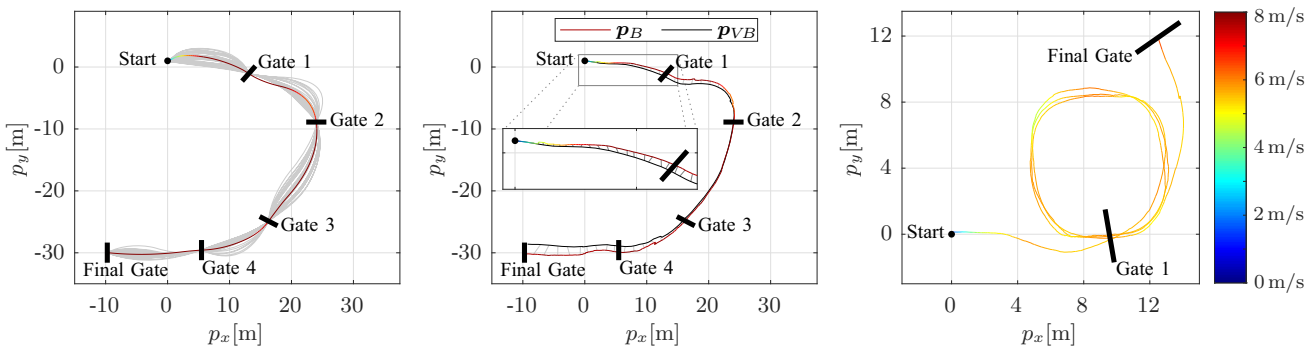

Fig. 6 Top view of the planned (left) and executed (center) path at the championship race, and an executed multi-lap path at a testing facility (right). Left: Fastest planned path in color, sub-optimal sampled paths in gray. Center: VIO trajectory as $\boldsymbol{p}_{VB}$ and corrected estimate as $\boldsymbol{p}_B$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

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. 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. 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. 6, right), albeit the course consisted of only two gates. The drone was commanded to pass 4 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. 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.

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 Ronneberger et al. (2015) are compared. Table 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.

Table 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 352 x 592 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

Deployment Even in instances of strong changes in illumination, the gate detector was able to accurately identify the gates in a range of $2-17$ m. Fig. 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 Sect. 4, one simultaneous inference for the left- and right-facing camera requires computing 3.86GFLOPS (40kFLOPS per pixel). By implementing the network in TensorRT and performing infer-

Table 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	9.5935 s	1.66 ms 2.35%
2	9.2913 s	18.81 ms 26.56%
3	9.2709 s	35.74 ms 50.47%
4	9.2667 s	53.00 ms 74.84%
5 (full track)	9.2622 s	70.81 ms 100%
CPC Foehn and Scaramuzza (2020) (full track)	6.520 s	$4.62 \cdot 10^5$ ms 6524%

ence 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.

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 Li et al. (2019); Jung et al. (2018); Kaufmann et al. (2018), 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. 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. 6 (center).

8.3 Planning and control

Figure 6 (left) shows the nominally planned path for the *AlphaPilot* championship race, where the coloured line depicts the fastest path along all the sampled paths depicted in gray. In particular, a total of $M = 150$ different states are sampled at each gate, with the velocity limited to 8 m/s and the angle between the velocity and the gate normal limited to $\varphi_{\max} = 30°$. During flight, the path is re-planned in a receding horizon fashion through the next $N = 3$ gates (see Fig. 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 3 presents the trade-offs between total flight time and computation cost for different horizon lengths N for the track shown in Fig. 6 (left). In addition, Table 3 shows the flight and computation time of the time-optimal trajectory generation from Foehn and Scaramuzza (2020), 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. Figure 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.

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.

Acknowledgements This work was supported by the Intel Network on Intelligent Systems, the Swiss National Science Foundation (SNSF) through the National Center of Competence in Research (NCCR) Robotics, the SNSF-ERC Starting Grant, and SONY.

Funding Open Access funding provided by Universitàt Zürich.

References

Bertsekas, Dimitri P. (1995). *Dynamic programming and optimal control* (Vol. 1). MA: Athena scientific Belmont.

Bloesch, Michael, Omari, Sammy, Hutter, Marco, & Siegwart, Roland. (2015). Robust visual inertial odometry using a direct EKF-based approach.

Brescianini, D., & DâAndrea, R., (2018). Tilt-prioritized quadrocopter attitude control. *IEEE Transactions on Control Systems Technology*, *28*(2), 376–387.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., et al. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics, 32*(6), 1309–1332.

Cao, Zhe., Simon, Tomas., Wei, Shih-En., and Sheikh, Yaser .(2017). Realtime multi-person 2d pose estimation using part affinity fields. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7291–7299.

Delmerico J, & Scaramuzza D. (2018). A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots.

Delmerico J, Cieslewski T, Rebecq H, Faessler M, & Scaramuzza D. (2019). Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset.

Douglas Brent West. (2001). *Introduction to graph theory* (Vol. 2). NJ: Prentice hall Upper Saddle River.

Elia K, M Gehrig, P Foehn, R Ranftl, A Dosovitskiy, V Koltun, and D Scaramuzza. (2018) Beauty and the beast: Optimal methods meet learning for drone racing. *IEEE International Conference on Robotics and Automation (ICRA)*, pp 690–696.

Falanga, Davide., Foehn, Philipp., Lu, Peng., and Scaramuzza, Davide .(2018). Pampc: Perception-aware model predictive control for quadrotors. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8.

Foehn, Philipp and Scaramuzza, Davide (2020) CPC: Complementary progress constraints for time-optimal quadrotor trajectories. Preprint arXiv:2007.06255.

Foehn, P., Brescianini, D., Kaufmann, E., Cieslewski, T., Gehrig, M., Muglikar, M., and Scaramuzza. D.(2020). Alphapilot: Autonomous drone racing. In *Robotics: Science and Systems (RSS)*. 10.15607/RSS.2020.XVI.081.

Forster, Christian, Zhang, Zichao, Gassner, Michael, Werlberger, Manuel, & Scaramuzza, Davide. (2017a). SVO: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics, 33*(2), 249–265. https://doi.org/10.1109/TRO.2016.2623335

Forster, Christian, Carlone, Luca, Dellaert, Frank, & Scaramuzza, Davide. (2017b). On-manifold preintegration for real-time visual-inertial odometry. *IEEE Transactions on Robotics, 33*(1), 1–21. https://doi.org/10.1109/TRO.2016.2597321

Gao, Fei, William, Wu., Gao, Wenliang, & Shen, Shaojie. (2019). Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics*. https://doi.org/10.1002/rob.21842

Guerra, Winter., Tal, Ezra., Murali, Varun., Ryou, Gilhyun., and Karaman, Sertac.(2019). Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. arXiv:1905.11377.

Hehn, M., Ritz, R., & DâAdrea, R., (2012). Performance benchmarking of quadrotor systems using time-optimal control. *Autonomous Robots*,. https://doi.org/10.1007/s10514-012-9282-3.

Kai, J. M., Allibert, G., Hua, M. D., & Hamel, T. (2017). Nonlinear feedback control of quadrotors exploiting first-order drag effects. *IFAC-PapersOnLine, 50*(1), 8189–8195.

Kalman, Rudolf E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering, 82*(1), 35–45.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*(1–2), 83–97.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge Cambridge Cambridge Cambridge: Cambridge University Press.

Li, Shuo., van der Horst, Erik.,Duernay, Philipp., De Wagter, Christophe., and de Croon, Guido.(2019). Visual model-predictive localization for computationally efficient autonomous racing of a 72-gram drone. arXiv:1905.10110.

Loock, W. Van., Pipeleers, G., and Swevers, J.(2013). Time-optimal quadrotor flight. In *IEEE European Control of Conference (ECC)*. 10.23919/ECC.2013.6669253.

Maurer, H. (1977). On optimal control problems with bounded state variables and control appearing linearly. *SIAM Journal on Control and Optimization, 150*(3), 345–362.

Mellinger, D., Michael, N., & Kumar, V. (2012). Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*. https://doi.org/10.1177/0278364911434236

Moon, H., Sun, Y., Baltes, J., & Kim, S. J. (2017). The IROS 2016 competitions. *IEEE Robotics Automation Magazine, 24*(1), 20–29.

Moon, H., Martinez-Carranza, J., Cieslewski, T., Faessler, M., Falanga, D., Simovic, A., et al. (2019). Challenges and implemented technologies used in autonomous drone racing. *Intelligent Service Robotics, 12*, 137–148.

Mourikis, Anastasios I., and Roumeliotis, S.I.(April 2007). A multi-state constraint Kalman filter for vision-aided inertial navigation.

In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3565–3572.

Mueller MW., Hehn M., and D'Andrea R.(2015). A computationally efficient motion primitive for quadrocopter trajectory generation. IEEE Transactions on Robotics. doi: https://doi.org/10.1109/TRO.2015.2479878.

Qin, Tong, Li, Peiliang, & Shen, Shaojie. (2018). VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics, 34*(4), 1004–1020. https://doi.org/10.1109/TRO.2018.2853729

Ronneberger, Olaf., Fischer, Philipp., and Brox, Thomas .(2015). U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer.

Ryou, G., Tal, E., and Karaman, S.(2020). Multi-fidelity black-box optimization for time-optimal quadrotor maneuvers. *Robotics: Science and Systems (RSS)*. 10.15607/RSS.2020.XVI.032.

Shuster, Malcolm D. (1993). Survey of attitude representations. *Journal of the Astronautical Sciences, 410*(4), 439–517.

Spasojevic, Igor, Murali, Varun, & Karaman, Sertac. (2020). Joint feature selection and time optimal path parametrization for high speed vision-aided navigation.

Sunggoo, J., Sunyou, H., Heemin, S., & Shim, D. H. (2018). Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters, 3*(3), 2539–2544. https://doi.org/10.1109/LRA.2018.2808368

Szeliski, Richard .(2010). *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer. ISBN 9781848829343.

Zhou, B., Gao, F., Wang, L., Liu, C., & Shen, S. (2019). Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*. https://doi.org/10.1109/LRA.2019.2927938