

# Race Against the Machine: a Fully-annotated, Open-design Dataset of Autonomous and Piloted High-speed Flight

Michael Bosello, Davide Aguiari, Yvo Keuter, Enrico Pallotta, Sara Kiade, Gyordan Caminati, Flavio Pinzarrone, Junaid Halepota, Jacopo Panerati, and Giovanni Pau

**Abstract**—Unmanned aerial vehicles, and multi-rotors in particular, can now perform dexterous tasks in impervious environments, from infrastructure monitoring to emergency deliveries. Autonomous drone racing has emerged as an ideal benchmark to develop and evaluate these capabilities. Its challenges include accurate and robust visual-inertial odometry during aggressive maneuvers, complex aerodynamics, and constrained computational resources. As researchers increasingly channel their efforts into it, they also need the tools to timely and equitably compare their results and advances. With this dataset, we want to (i) support the development of new methods and (ii) establish quantitative comparisons for approaches originating from the broader robotics and artificial intelligence communities. We want to provide a one-stop resource that is comprehensive of (i) aggressive autonomous and piloted flight, (ii) high-resolution, high-frequency visual, inertial, and motion capture data, (iii) commands and control inputs, (iv) multiple light settings, and (v) corner-level labeling of drone racing gates. We also release the complete specifications to recreate our flight platform, using commercial off-the-shelf components and the open-source flight controller Betaflight, to democratize drone racing research. Our dataset, open-source scripts, and drone design are available at [github.com/tii-racing/drone-racing-dataset](https://github.com/tii-racing/drone-racing-dataset).

## I. INTRODUCTION

Unmanned aerial vehicles (UAVs) and multi-rotor drones have become ubiquitous robotic platforms, supporting a wide array of industries from video-making to warehouse monitoring, to surveillance and inspection of energy and transport infrastructure. Drone racing, in particular, has emerged as the go-to benchmark problem to measure the advances made by researchers in the quest to surpass human-level, autonomous performance in fast and aggressive flight [1]–[4]. Yet, drone racing competitions, equipment, and venues can still be difficult and expensive to access.

The last decade of machine learning progress has shown how datasets, open standards, and open-source code help scientific progress and transparency, shaping the entire scientific fields [5]. One of the fundamental advantages of datasets is to greatly simplify and shorten the development pipeline of new methods by allowing researchers to re-use the tried and tested data collection and consolidation work of others. Datasets allow researchers from different parts of the world and disciplines to work on common problems.

Today, several conferences and journals, including NeurIPS, the IEEE Robotics and Automation Letter, and the International Journal of Robotics Research explicitly solicit data and benchmark papers as a way to increase the number and visibility of peer-reviewed datasets [12].

All the authors are with the Autonomous Robotics Research Center of the Technology Innovation Institute, Abu Dhabi, United Arab Emirates. Michael Bosello and Giovanni Pau also are with the University of Bologna, Bologna, Italy. E-mails: {`firstname.lastname`}@tii.ae

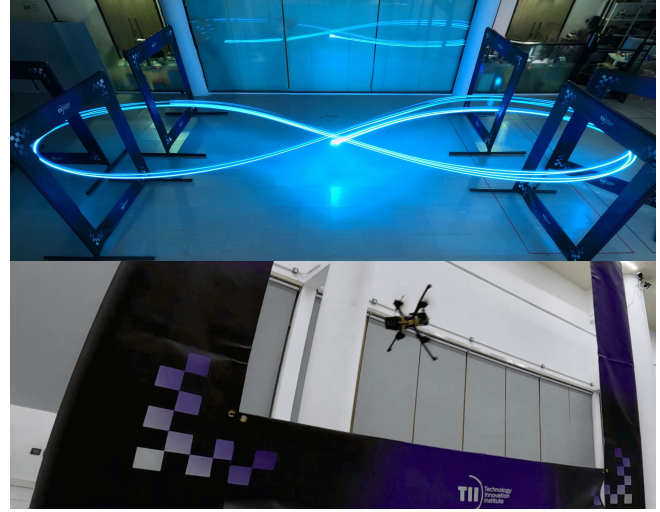


Fig. 1: Long exposure, low-light capture of the open-design racing drone used to collect the dataset (top), and an aggressive maneuver through one of the labeled gates (bottom).

Robotics datasets and benchmarks are as important and beneficial to the community as they are challenging to create—because of the idiosyncrasies of robotic hardware and real-world systems. Notable robotics datasets have focused on vision problems, e.g., the KITTI Vision Benchmark Suite [13], including data from stereo cameras, GPS, and laser scanners for tasks such as object detection, tracking, and visual-inertial odometry (VIO) [9], or the use of special, novel sensors and instruments [14]. Early drone racing datasets [6] also focused on scene understanding and gate pose estimation problems, while more recent datasets have put a greater emphasis on the coupling with on-board inertial data, ground truth information, and controls [15], [16].

Our dataset builds upon these and aims to be a one-stop resource for researchers to simultaneously pursue multiple lines of work, including semantic scene understanding, VIO, mapping and planning, and data-based system identification for fast and aggressive multi-rotor flight. As for a benchmark to be successful, it must be effectively and easily repeatable, in Section III, we release the complete design specifications of the drone used to collect the dataset.

The main contributions of our work are as follows.

- The public release of a dataset for drone racing (inclusive of open-source code for visualization and post-processing) that is characterized by:
  - fast ( $>20\text{m/s}$ ), aggressive flight, both autonomous and human-piloted, on multiple trajectories (including a complex 3D racing track);

TABLE I: Comparison of multi-rotor and drone racing datasets for visual-inertial odometry, scene understanding, and control

Ref.	Time & Distance	Data Coll.	Conditions		Gates		Top Speed	Vision/Camera Specifications				Pose/Inertial Data		Control Inputs		Battery Voltage	Data Formats	
			Scene	Lighting	Pose	Labels		Resolution/Freq.	Color	FoV	Stereo	Event	IMU	MoCap	CTBR			Motor
Ours TIL-RATM	~29' ~7km	Real	Indoor	3 Levels Labeled	✓	✓ <sup>†</sup>	9.5m/s <sup>◊</sup> 21.8m/s <sup>*¶</sup>	640x480@120Hz	RGB	D 175°	✗	✗	@500Hz	@275Hz	@100Hz	@100Hz	@50Hz	rosbag, CSV, JPEG
[3] UZH-FPV	~24' ~11km	Real	Indoor; Outdoor	Multiple, Unlabel.	✗	✗	26.8m/s <sup>¶</sup> 23.4m/s <sup>◊</sup>	848x800@30Hz 346x260@50Hz 640x480@30Hz*	Grayscale Grayscale Grayscale	D 163° 120° 186°	✓	✓	@200Hz @500Hz @1000Hz	@20Hz <sup>§</sup>	✗	✗	✗	rosbag, TXT, PNG
[6] AlphaPilot	n/a <sup>‡</sup>	Real	Indoor; 1 Gate	Multiple, Unlabel.	✗	✓ <sup>  </sup>	n/a	1296x864	RGB	n/a	✗	✗	✗	✗	✗	✗	✗	JSON, JPEG
[7] Blackbird	~10h ~100km	Real + Synth.	Indoor, 5 Scenes	Multiple, Unlabel.	✗	✗	7m/s	1024x768@120Hz <sup>††</sup> 1024x768@360Hz <sup>††</sup>	Grayscale RGB	V 60°	✓	✗	@100Hz	@360Hz	✗	@190Hz	✗	rosbag, CSV, MP4, PNG Depth
[8] EuRoC	~22' ~1km	Real	Indoor, 2 Scenes	Multiple, Unlabel.	✗	✗	2.3m/s	752x480@20Hz	Grayscale	H 115°	✓	✗	@200Hz	@20Hz <sup>§</sup> @100Hz	✗	✗	✗	CSV, PLY, PNG
[9] GRASP	~10' ~3km	Real	Outdoor; 1 Scene	Multiple, Unlabel.	✗	✗	17.5m/s	960x800@40Hz	Grayscale	n/a	✓	✗	@200Hz	✗	✗	✗	✗	rosbag
[10] EyeGaze	~300' ~100km	Synth.	Indoor, 2 Scenes	Multiple, Unlabel.	✓	✓ <sup>¶¶</sup>	13.8m/s	800x600@60Hz	RGB	120°	✗	✗	✗	@500Hz <sup>**</sup>	@500Hz	✗	✗	CSV, MP4
[11] NeuroBEM	~75'	Real	n/a	✗	✗	✗	18m/s	✗	✗	✗	✗	✗	@1000Hz	@400Hz	✗	@1000Hz	@400Hz	CSV

<sup>†</sup>Bounding boxes, top-bottom left-right corners. <sup>◊</sup>Piloted. <sup>¶</sup>Autonomous. <sup>\*</sup>Stereo. <sup>§</sup>Leica laser tracker. <sup>‡</sup>9300 frames.  
<sup>||</sup>Internal corners. <sup>††</sup>Synthetic camera images. <sup>¶¶</sup>Area of interest of the gaze. <sup>\*\*</sup>Simulated.

- high-resolution, high-frequency ( $\sim 10^2$ Hz) collection of visual, inertial, and motion capture data;
- versatility—our dataset includes drone racing gates fully labeled to the level of individual corners [17] (for VIO, self-localization, scene understanding, etc.), information about commands, control inputs, and battery voltages (for estimation problems, etc.), as well as lighting and sensor settings metadata.
- The open design (with commercial off-the-shelf (COTS) components) of the racing drone used to collect the data. For direct comparisons, the same design allows, without modifications, both autonomous and piloted flight.

## II. RELATED WORK

In Table I, we summarize the—all very recent—datasets for vision-based flight, drone racing, and aggressive quadrotor control related to our own. Earlier datasets for multi-rotor VIO and simultaneous localization and mapping (SLAM) included the 2016 EuRoC [8] and the 2017 Zurich Urban [18] micro aerial vehicle (MAV) datasets. However, these were characterized by comparatively lower speeds and frequencies of images and collected data than those needed for drone racing.

The last five years have seen a renewed interest in aggressive flight [15]. In [9], a new dataset was introduced to validate stereo VIO methods for fast autonomous flight, although without the inclusion of racing gates. The Blackbird dataset [7] was proposed as an aggressive indoor flight dataset for agile perception. While inertial data were collected in the real world, its high-resolution images were generated in simulation. In 2019, Lockheed Martin released an image dataset for Test#2 of its AlphaPilot challenge’s virtual qualifiers [6], that did not include drone state information (while Test#3 consisted of control in simulation).

The work closest to ours is the dataset presented in [3]. It also observes that previous benchmarks for drone VIO had focused on too slow trajectories. Our dataset differs from [3] in that it provides higher-frequency RGB mono-camera images (the information used by human pilots), it

includes piloted and autonomous flights on identical tracks, and it is fully annotated with high-frequency motion capture data as well as the gates’ corner labels.

Other recent, specialized datasets for drone racing and aggressive multi-rotor flight include [10], [11]. Another open-source, open-hardware racing drone was proposed in [19]. In comparison, our design has a more powerful companion computer and it can be seamlessly used by human pilots.

Compared to the existing literature (Table I) [15], our dataset (*i*) includes high-resolution, high-frequency images captured in different light settings and (*ii*) it is fully annotated, down to the gates’ individual corners. It can support research in all aspects of autonomous drone racing from VIO, to gate pose estimation [20], [21] and data-driven control.

## III. AN OPEN-DESIGN QUADROTOR FOR AUTONOMOUS DRONE RACING AND DATA COLLECTION

To collect this dataset, we designed a new custom quadrotor (Figure 2). Our design is based on a 5” carbon-fiber frame with a propeller-to-propeller diagonal of 215mm. The fully-assembled drone, including the battery, weighs  $\sim 870$ g, has a thrust-to-weight ratio of 7.5 (load cell), and can reach a maximum speed (measured outdoors) of 179km/h—allowing for the aggressive maneuvers required in drone racing. The top linear acceleration and angular velocity in the dataset are  $69.85\text{m/s}^2$  ( $>7g$ ’s) and  $20.01\text{rad/s}$ . Importantly, our design can be used, without modifications, as both an autonomous and a human-piloted FPV racing drone. We do this to create a true test bench to benchmark drone racing autonomy against human performance. Our dataset includes both autonomous and piloted flights. Its components are listed in Table II.

### A. Design Overview

The quadrotor has three main sub-systems: (*i*) the quadrotor electronics, (*ii*) the autonomous module, and (*iii*) the First-Person-View (FPV) system. These sub-systems are combined by means of the frame and fasteners. The assembled system comprises two cameras. One digital, connected to the autonomous module, and one analog, used by the human pilot in the FPV system. The two cameras share the

TABLE II: Off-the-shelf components needed to re-create the open-design racing drone used to collect the dataset

Component	Producer	Model
ESC	T-MOTOR	F55A PROII 6S 4IN1
FCU	Holybro	Kakute H7 v1.3
RC Receiver	Team BlackSheep	CROSSFIRE NANO RX (SE) LONG RANGE
Battery	Tattu	R-Line v5.0 1400mAh 22.2V 150C 6S1P LiPo
Computer	NVIDIA	Orin NX 16GB Module
Carrier board	Seed Studio	A203 (Version 2)
BEC	Matek	BEC12S-PRO
Camera	Arducam	B0179 8MP IMX219
FPV Camera	Foxeer	T-Rex Mini 1500TVL
Frame	Pyrodrone	Hyperlite Floss 3.0 Race Frame 5"
Motors	T-MOTOR	F60 PRO V 2020KV
Propellers	T-MOTOR	T5147

same mount, and the FPV camera is placed above the digital one (Figure 2).

1) *Quadrotor electronics*: These are (i) the electronic speed controller (ESC), (ii) the Kakute H7 v1 flight controller unit (FCU), (iii) the radio controller (RC) receiver, and (iv) the battery. They are mounted underneath the frame. These components are protected by the aluminum standoffs connecting the frame and a 3D-printed custom battery cage. The FCU hosts an STM32H7 microcontroller and it is capable of running multiple firmware, including Betaflight, Ardupilot, and PX4.

2) *Autonomous module*: It comprises (i) an NVIDIA Orin NX (hosted on the A203v2 carrier board with SSD and wireless card), (ii) the battery eliminator circuit (BEC) powering it, and (iii) an Arducam RGB camera. These components are placed above the frame and are secured by two 3D-printed plates, connected by aluminum standoffs. The top plate provides the mount for the cameras. A MIPI CSI-2 ribbon cable connects the companion board with the Arducam. The FC is connected via a serial port, using a shielded cable. This connection is used to both control the drone and read FC’s sensors.

3) *FPV system*: Independent from the autonomous module and used for human piloting instead, it comprises an FPV analog camera, a video transmitter, and its antennas, all placed above the frame.

The CAD models of all the 3D printed parts and the bill of materials of all other COTS components are available online<sup>1</sup>, with a video tutorial on how to re-create our drone.

## B. Sensors

Our quadrotor is equipped with multiple sensors for autonomous and piloted aggressive flight:

<sup>1</sup>[github.com/tii-racing/drone-racing-dataset/tree/main/quadrotor](https://github.com/tii-racing/drone-racing-dataset/tree/main/quadrotor)

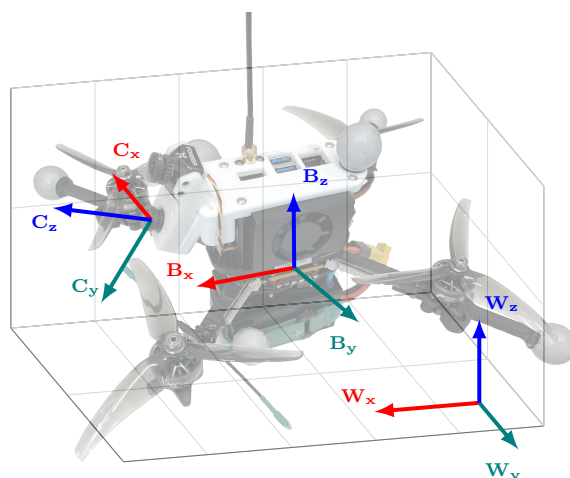


Fig. 2: The drone platform used to record the dataset, the body frame  $B$  has its origin at the FCU’s IMU location, the camera frame  $C$  is located where the bottom lens is (the top lens being the one of the FPV system).

1) *InvenSense MPU6000 IMU*: Part of the quadrotor electronics (III-A.1), it is embedded into the FC. This module has two functions: delivering precise, real-time tri-axis angular rate sensor (gyroscope) data, as well as accurate tri-axis accelerometer data. The raw IMU data are read by the companion computer in a demand/response exchange fashion, using the Multiwii Serial Protocol (MSP) [22].

2) *Arducam B0179 IMX219 8MP RGB Bayer camera*: Part of the autonomous module (III-A.2), it captures  $640 \times 480$  pixel frames at 120Hz with a diagonal field-of-view (FOV) of  $175^\circ$  and a horizontal field-of-view (HFOV) of  $155^\circ$ . Its readout speed is  $3.22 \times 10^{-5}$ s, computed as the line length divided by the pixel rate before re-scaling, i.e.,  $3560\text{px}/(1280 \times 720 \times 120\text{Hz})$ . It is one of the most used lightweight embedded cameras on the market and it is fully supported by NVIDIA with dedicated MIPI CSI-2 drivers. The image YUV frames are captured in NV12 format. In the NV12 format, each pixel in the luminance (Y) component is represented by a single byte. The chrominance (UV) components are interleaved and share memory locations. The image is then converted to BGR, a common color format suitable for processing and analysis. Eventually, the image is saved as JPEG. This pipeline is accomplished by the NVIDIA GStreamer plugin on the NVIDIA companion computer.

3) *Foxeer T-Rex Mini 1500TVL*: It is the low latency (6ms) camera in the FPV system (III-A.3). For the sake of the data collection in this letter, this was used by a human pilot in conjunction with a pair of 1280x960 OLED Fat Shark HDO2 goggles.

## C. Software

1) *Quadrotor electronics (III-A.1) software*: The FC firmware we used is Betaflight 4.3.1 [23], whose proportional-integral-derivative controller (PID) was tuned by a human pilot. The companion computer uses MSP to both send commands to the FC and read its sensors. Accordingly, we activate Betaflight’s MSP\_OVERRIDE feature to



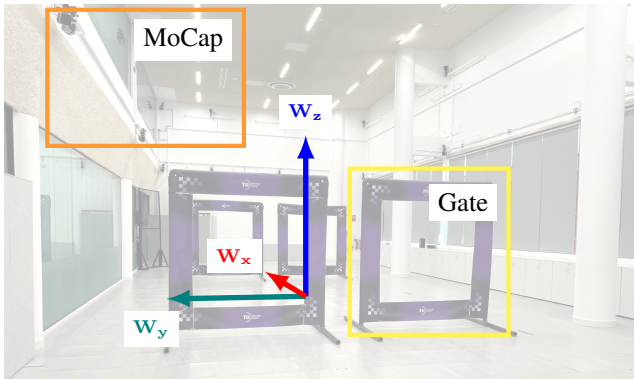


Fig. 3: The 25×9.7×7 meters indoor arena, instrumented with 32 Qualisys MoCap cameras and equipped with four 5×5 feet racing gates used to record the dataset.

bypass the RC controller’s commands. An `MSP_OVERRIDE` channels mask limits to override the motor commands: for safety reasons, a human can always disarm the drone with an RC controller.

2) *Autonomous module (III-A.2) software*: On the Orin NX module, we installed NVIDIA JetPack 5.1.1. The JetPack includes Jetson Linux 35.3.1 Board Support Package (BSP) with Linux Real-Time Kernel 5.10, an Ubuntu 20.04-based root file system with CUDA 11.4 support. We use the Humble (current LTS) distribution of the Robot Operating System 2 (ROS2) as the middleware for communication between the perception, planning, and control modules.

#### IV. DATA COLLECTION PROTOCOL

##### A. Flight Arena, Racing Gates, and Motion Capture System

Our dataset was recorded in a 25 (L) by 9.7 (W) by 7 (H) meters indoor flying arena. The arena is equipped with a 32-camera Arqus A12 Qualisys Motion Capture (MoCap) system, tracking 6DoF poses of defined rigid bodies with millimeter accuracy at 275Hz. The drone design from Section III is equipped with six 25mm markers defining a single rigid body. The markers are mounted on the top plate, battery cage, and 48.2mm arm extensions, preventing the markers from being occluded by the propellers (Figure 2). The origin of the drone’s rigid body was placed at the location of the FCU’s IMU, as shown in Figure 2. The IMU was also calibrated using the RC before each take-off [23].

We use a minimum of 4 and up to 7 gates to create 2D and 3D racing tracks. The 7-gate track is inspired by the one in [1] but shrunken to meet our arena size constraints (Figure 3). This track features challenging maneuvers like the Split-S and sharp turns, and it evolves over the  $z$ -axis for 5 meters. The gates are made of PVC pipes covered by printed fabric banners. They measure 7 by 7 feet (213.36 cm) and have an internal opening of 5 by 5 feet (152.4 cm), similar to those used in major drone racing leagues [24]. Each racing gate’s rigid body is defined by four markers placed in its inner corners.

##### B. Flight Program

Our dataset contains a total of 30 flights (Table III): 12 *human-piloted* and 18 *autonomous* ones. In either case, two shapes—*ellipse* and *lemniscate* (Figure 4)—have been

TABLE III: Summary of the flights recorded in the dataset

Control	Shape	Top Speed	Time	Distance
Autonomous	Ellipse <sup>†</sup>	<b>21.83 m/s</b>	149.08 s	455.27 m
	Lemniscate <sup>†</sup>	10.22 m/s	155.08 s	359.63 m
	Race Track <sup>‡</sup>	21.39 m/s	278.05 s	1161.51 m
Piloted	Ellipse <sup>§</sup>	9.50 m/s	575.38 s	2586.3 m
	Lemniscate <sup>§</sup>	8.93 m/s	<b>593.63 s</b>	<b>2593.47 m</b>

<sup>†</sup>Flown twice in 6 flights (3 brightness × 2 camera settings). <sup>‡</sup>Flown three times in 6 flights (3 brightness × 2 camera settings). <sup>§</sup>Flown as many times as possible in 6 flights (3 brightness × 2 camera settings).

executed 6 times. The 6 additional autonomous flights are collected on a 3D race track. The 6 repetitions correspond to the combination of 3 different illumination conditions and 2 camera settings.

1) *Brightness levels*: We created three levels of brightness for data collection. The *high* brightness level is achieved with both natural and artificial light (max./avg./min. measured illuminance in the arena of 2480 lx, 1500 lx, and 254 lx); the *medium* level has controlled light, and it is obtained by turning on all the artificial lights in the arena but keeping natural light out using the blinds (955 lx/672.3 lx/254 lx). The *low* brightness condition is obtained by turning off most of the lights in the arena and keeping the blinds down, (216 lx/72 lx/34.2 lx).

2) *Camera settings*: We used two different `nvguscamera` settings, *auto* exposure time and gains, and *fixed* exposure time (2.5ms), analog gain (2), and digital gain (1). In the *auto* setting, the image is brighter but suffers from higher motion blur. In *fixed*, the image is darker, with limited motion blur.

##### C. Human-piloted and Autonomous Control

In autonomous mode, each flight contains exactly two (ellipse, lemniscate) or three (race track) laps, lasting ~25 and ~45s, respectively. The human-piloted flights last between 84 and 108s, during which the pilot tries to achieve the maximum number of laps possible on a single battery charge. We did not clip the human-piloted flights as they exhibit higher variability and cover a larger state space.

In the autonomous flights, we use a Proportional Derivative (PD) controller based on [25] for the 2D tracks and a Model Predictive Controller (MPC) based on [26] for the 3D track. The motion capture system feeds the current quadrotor pose into the controller at 275Hz through WiFi. Control commands are sent to the FCU at the same update frequency of 275Hz by the PD controller, and 160Hz by the MPC. A Python implementation of the PD controller and its tuned gains are available on GitHub<sup>2</sup>.

The human pilot used a camera angle (for both their FPV camera and the recorded Arducam) of 30°. The autonomous runs were recorded with a camera angle of 40° for the lemniscate and 50° for the ellipse and race track, empirically chosen to maximize the gates’ visibility. FPV pilots choose the camera angles according to the speed they plan to reach<sup>3</sup>.

<sup>2</sup>[github.com/tii-racing/drone-racing-dataset/blob/main/scripts/reference.controller.py](https://github.com/tii-racing/drone-racing-dataset/blob/main/scripts/reference.controller.py)

<sup>3</sup>[www.getfpv.com/learn/fpv-essentials/fpv-camera-angle-full-throttle-flight/](http://www.getfpv.com/learn/fpv-essentials/fpv-camera-angle-full-throttle-flight/)

We used the same principle to choose the camera angle for the autonomous mode, making the gates visible at high speed.

#### D. Image Labeling

Often being the only well-known object in a racing environment, gates are key to relative localization and next-waypoint detection. In this dataset, we provide image labels in the form of bounding boxes and keypoints associated with the inner corners of all visible gates. Coupled with the drone’s inertial data and the ground truth from the motion capture system, our dataset allows to reproduce and benchmark the state-of-the-art in gate pose estimation [27] as well as to develop new methods.

Labeling was first performed automatically, using a top-down keypoints detector [28], [29] trained on a synthetic dataset and fine-tuned on 5’000 manually labeled images. All the images were then labeled through an iterative process of manual review and re-training. Finally, all labels were eventually manually reviewed. Bounding boxes and corner positions are provided, based on the auto-labeling results and a human estimate, even for partially occluded gates. However, no distinction between occluded and fully visible keypoints is made. Thus, the only visibility values we provide are 0 (outside the image boundaries) and 2 (inside the image boundaries) as per COCO format definition [30]. As a convention, gates are not labeled when there are no visible corners.

#### E. Time Synchronization

We record three separate data streams during each flight: (i) a `rosbag` containing the FCU readings and the autonomous control setpoints, (ii) the on-board camera images, and (iii) the Qualisys motion capture measurements.

For the FCU data, we use custom ROS2 messages and the Real-Time Kernel to limit the jitter in the sensors’ readings. Furthermore, the GStreamer pipeline allows us to save the images and timestamp them using the frame acquisition time.

In the end, two different clocks are involved: the real-time clock of the drone’s companion computer, and the clock of the Qualisys workstation. Both were synced with a Network Time Protocol (NTP) server placed inside the facility, before each flight. The clock offsets w.r.t. the NTP server of the two machines were recorded before and after each flight to compute the jitter that occurred during the flight. The drone achieved a microsecond clock accuracy with Chrony, while the Qualisys workstation recorded a millisecond accuracy. The total jitter from start to end of a trajectory never exceeded 3ms.

#### F. Data Post-processing

The motion capture data were converted to CSV, and the clock offset with the onboard computer was removed. The ROS2 bags were also dumped into CSVs. All the data were then trimmed to remove pre-take-off and post-landing records. We used an open-source script for data alignment (Snipped VI) to produce user-friendly comprehensive CSVs. The alignment is achieved through linear interpolation for all the fields with the exception of the rotation matrices, for which spherical linear interpolation [31] is used instead.

## V. DATA FORMAT

For each flight, we recorded data from four different sources: (i) the motion capture system’s ground truth (drone and gates poses), (ii) the Arducam IMX219 Bayer camera (images), (iii) the flight controller (IMU, battery, motors, RC), and (iv) companion computer (autonomous control reference and control inputs).

All data collected is timestamped (Unix epoch time) with a microsecond resolution. In every flight folder, a YAML metadata file summarizes the camera and light setting, along with the type of track. The total time of flight and meters traveled are also included. The folder and file structure of the dataset are shown in Figure 5

#### A. Camera-aligned and Uniform-sampling CSVs

For each flight, we pre-compiled two easy-to-use, comprehensive CSV files that include all the data detailed in the subsequent sections, aligned via interpolation. In each `[FLIGHT]_cam_ts_sync.csv` file, we use the timestamps from the camera frames, and all other data points are linearly interpolated to align with these timestamps. Conversely, in each `[FLIGHT]_500hz_freq_sync.csv` file, timestamps are sampled at a uniform 500Hz frequency between the first and last camera timestamps. All numerical data are again interpolated and aligned with timestamps, and each row references the file name of the camera frame with the closest timestamp. All the columns in each of these CSVs are presented in Table IV.

#### B. Drone and Gates’ Poses from Motion Capture

For each flight, files `gate_corners_[FLIGHT].csv` contain the timestamped x, y, and z coordinates of all the gates’ markers, in meters. Files `mocap_[FLIGHT].csv` contain the poses of all the rigid bodies, i.e., the drone and the gates. Each pose comprises x, y, z, roll, pitch, yaw, measurement residual, and the orientation described by a 3×3 column-major order matrix. Poses are in meters and radians.

#### C. Camera Frames

The images from the Arducam IMX219 Bayer camera are recorded at 120 FPS with a resolution of 640×480px and saved as JPEG files. They are provided as a ZIP file along with a `camera_[FLIGHT].csv` which contains the timestamps of the acquisition of the frame and the name of the corresponding JPEG file.

#### D. Image Labels

For each image, the gates’ bounding boxes and internal corner labels are given as a TXT file with the same name as the JPEG file. Each line in a TXT file represents a single gate in the form:

$$0 \ c_x \ c_y \ w \ h \ tl_x \ tl_y \ tl_v \ tr_x \ tr_y \ tr_v \ br_x \ br_y \ br_v \ bl_x \ bl_y \ bl_v$$

where 0 is the class label for a gate (the only class in our dataset);  $c_x, c_y, w, h \in [0, 1]$  are its bounding box center’s coordinates, width, and height, respectively; and  $tl_x, tl_y \in [0, 1], tl_v \in [0, 2]$  are the coordinates and visibility (0 outside the image boundaries; 2 inside the image boundaries) of the top-left internal corner. Similarly for  $tr, bl$ ,

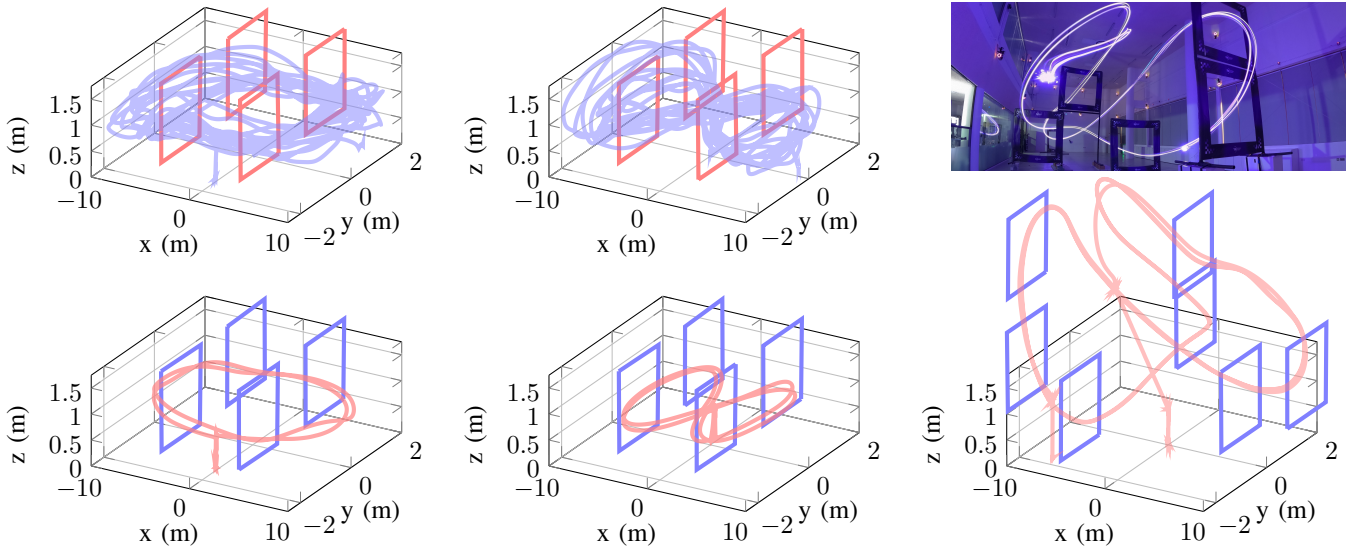


Fig. 4: Examples of recorded trajectories: piloted ellipse flight-01p-ellipse (top-left), piloted lemniscate flight-07p-lemniscate (top-center), autonomous ellipse flight-01a-ellipse (bottom-left), autonomous lemniscate flight-07a-lemniscate (bottom-center), and autonomous race track flight-13a-trackRATM (bottom-right).

br, the top-right, bottom-left, and bottom-right corners. All values are in pixel coordinates normalized with respect to image size. The keypoints label format follows the COCO definition [30]. The labels are provided as a ZIP file in `label_[FLIGHT].zip`.

#### E. On-board Data from the Quadrotor Electronics

From the FCU in Subsubsection III-A.1, we record the following measurements: battery’s voltage (Volts), at 50Hz (`battery_[FLIGHT].csv`); IMU, i.e., accelerometer ( $m/s^2$ ) and gyroscope ( $rad/s$ ) x, y, and z axes in the East-North-Up (ENU) board frame, at 500Hz (`imu_[FLIGHT].csv`); single motor thrust feedback, normalized between 0 and 1, for all four motors, at 100Hz (`motors_thrust_[FLIGHT].csv`); RC’s channels, i.e. roll, pitch, thrust, yaw, aux1, aux2, aux3, and aux4 values of the sticks between 1000 and 2000, at 100Hz (`channels_[FLIGHT].csv`, only for human-piloted flights).

#### F. On-board Data from the Autonomous Module

From the NVIDIA Orin in Subsubsection III-A.2, we record the following measurements: the drone state, i.e., position ( $m$ ), orientation (quaternion), velocity ( $m/s$ ), and angular velocity ( $rad/s$ ), at 275Hz, sent as ground truth from the MoCap system to the drone, with the time delay of the communication channel (`drone.state_[FLIGHT].csv`).

Furthermore, only for the autonomous flights, we also record: the controller’s reference, i.e., position ( $m$ ), orientation (quaternion), linear ( $m/s$ ) and angular velocity ( $rad/s$ ), acceleration ( $m/s^2$ ), jerk ( $m/s^3$ ), heading ( $rad$ ), and heading rate ( $rad/s$ ), computed at 100Hz for the ellipse and lemniscate, and 500Hz for the race track (`reference_[FLIGHT].csv`); the collective thrust and body rates (CTBR) computed by the autonomous controller, i.e., the normalized thrust ( $N/kg$  i.e.  $m/s^2$ ), roll, pitch, and yaw rates ( $rad/s$ ), at 275Hz for the PD controller, and 160Hz for the MPC, (`ctbr_[FLIGHT].csv`); and the RC’s channels sent to the FCU calculated from the CTBR commands, i.e.,

roll, pitch, thrust, yaw, aux1, aux2, aux3, and aux4 values of the sticks, at 275Hz and 160Hz for PD and MPC respectively, (`channels_[FLIGHT].csv`).

TABLE IV: Data available in the precompiled `cam.ts_sync.csv` and `500Hz_freq_sync.csv` (Sec. V-A)

Column Number and Quantity Name	Unit	Data Type
0. elapsed.time	s	float
1. timestamp	$\mu s$	int
2. img.filename	n/a	string
3. accel.[x y z]	$m/s^2$	float
6. gyro.[x y z]	$rad/s$	float
9. thrust[0-3]	1	float $\in [0, 1]$
13. channels.[roll pitch thrust yaw]	1	int $\in [1000, 2000]$
17. aux[1-4]	1	int $\in [1000, 2000]$
21. vbat	V	float
22. drone.[x y z]	m	float
25. drone.[roll pitch yaw]	rad	float
28. drone.velocity.linear.[x y z]	$m/s$	float
31. drone.velocity.angular.[x y z]	$rad/s$	float
34. drone.residual	m	float
35. drone.rot[[0-8]]	1	float
44. gate[1-7].int.[x y z]	m	float
56. gate[1-7].int.[roll pitch yaw]	rad	float
68. gate[1-7].int.residual	m	float
72. gate[1-7].int.rot[[0-8]]	1	float
108. gate[1-7].marker[1-4].[x y z]	m	float

## VI. VISUALIZATION AND POST-PROCESSING SCRIPTS

Along with the dataset, we provide an installable<sup>4</sup> open-source Python repository comprising of a set of processing scripts to visualize and manipulate the data.

<sup>4</sup>[github.com/tii-racing/drone-racing-dataset/blob/main/README.md](https://github.com/tii-racing/drone-racing-dataset/blob/main/README.md)

https://github.com/tii-racing/drone-racing-dataset/data/

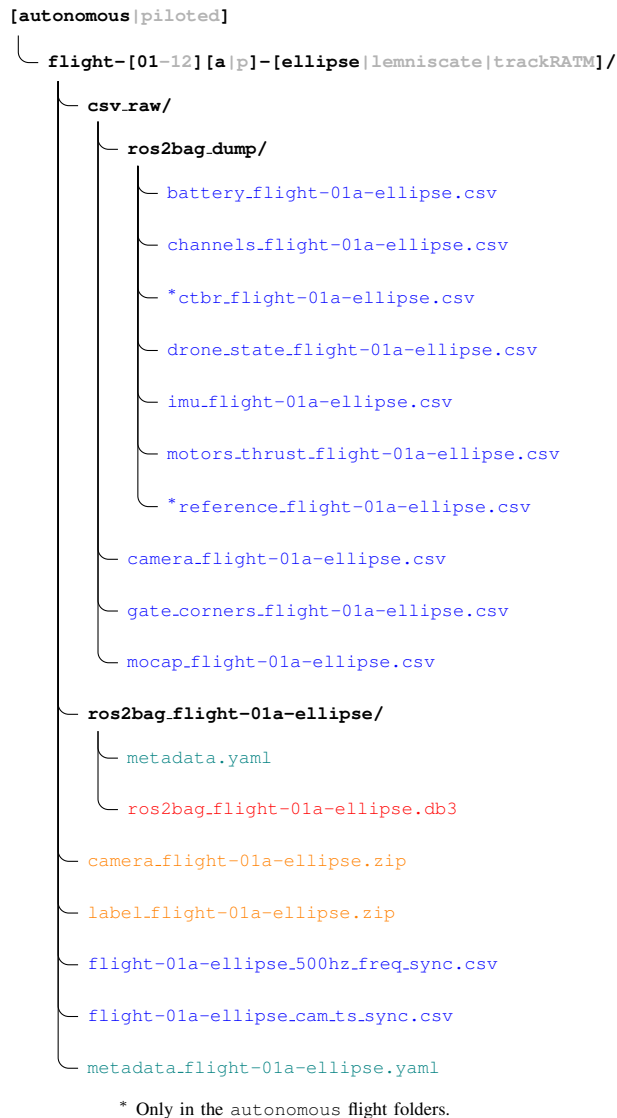


Fig. 5: Folder and file structure of the dataset.

`data_interpolation.py` can be used to re-sample and align the data at an arbitrary frequency using linear interpolation. Its output is a CSV file containing the aligned, re-sampled data from all the sources in Section V. First, one selects a flight by passing its id, e.g., `flight-01a-ellipse` as an argument. Then, one can choose which synchronization option to use. In option 1, one must choose a new frequency for which timestamps are generated, using the first and last camera timestamp as a window. All the data are then interpolated to the generated timestamps. For the camera frames, the one with the closest timestamp is referred to by file name. In option 2, the timestamps of the camera CSV are used as a reference instead.

```
$ python3 data_interpolation.py \
> --flight flight-01a-ellipse \
> --sync-option 1 --freq 200
Loading and pre-processing CSVs...
Using frequency 200 to interpolate all data
Syncing dataframes...
Sync complete.
Saving final CSV...
```

Final CSV saved.

`data_plotting.py` is a script to simultaneously visualize multiple sensor data using customizable subplots. One can select which subplots to include by means of command-line arguments. The script facilitates quick insights into the drone's flight behavior and performance, such as its 3D trajectory and the gate positions, as well as pose, velocities, accelerations, battery voltage, RC channels, etc.; an example of the script output is shown in Figure 6.

```
$ python3 data_plotting.py \
> --csv-file flight-01a-ellipse_cam_ts_sync.csv \
> --subplots 3d
```

`label_visualization.py` is the script to visualize the label annotations on the images. It allows one to skim the images of a flight, read the associated YOLO-style label file, and plot the gates' bounding boxes and internal corner keypoints. An individual frame example is shown in Figure 7.

```
$ python3 label_visualization.py \
> --flight flight-01a-ellipse
```

`create_std_bag.py` is a utility script that reads all the data from a user-specified flight and creates a new ROS2 bag with the same data (including images) but only using standard messages from the ROS2 `msgs` library. All the messages are timestamped in nanoseconds. It may take several GBs.

```
python3 create_std_bag.py \
> --flight flight-01a-ellipse
```

## VII. CONCLUSIONS

The development of autonomous racing drones requires simultaneously tackling challenging perception, state estimation, and control tasks—in real time—under limited computational resources. With this dataset, we created a one-stop resource to develop and evaluate new algorithms for autonomous drone racing. Our work is comprehensive of both aggressive autonomous and piloted flight; high-resolution, high-frequency visual, inertial, and motion capture data; commands and control inputs; multiple light settings; and corner-level labeling of drone racing gates. Along with the data, we open-sourced the scripts used to parse and visualize them. To further democratize autonomous drone racing research, we also released the parts list and instructions to recreate our flight platform, using commercial off-the-shelf components, hoping to see it re-created and used by researchers around the world.

## REFERENCES

- [1] E. Kaufmann, *et al.*, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, Aug 2023.
- [2] A. Romero, *et al.*, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3340–3356, 2022.
- [3] J. Delmerico, *et al.*, "Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6713–6719.
- [4] Y. Song, *et al.*, "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning," *Science Robotics*, vol. 8, no. 82, 2023.
- [5] J. Deng, *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.



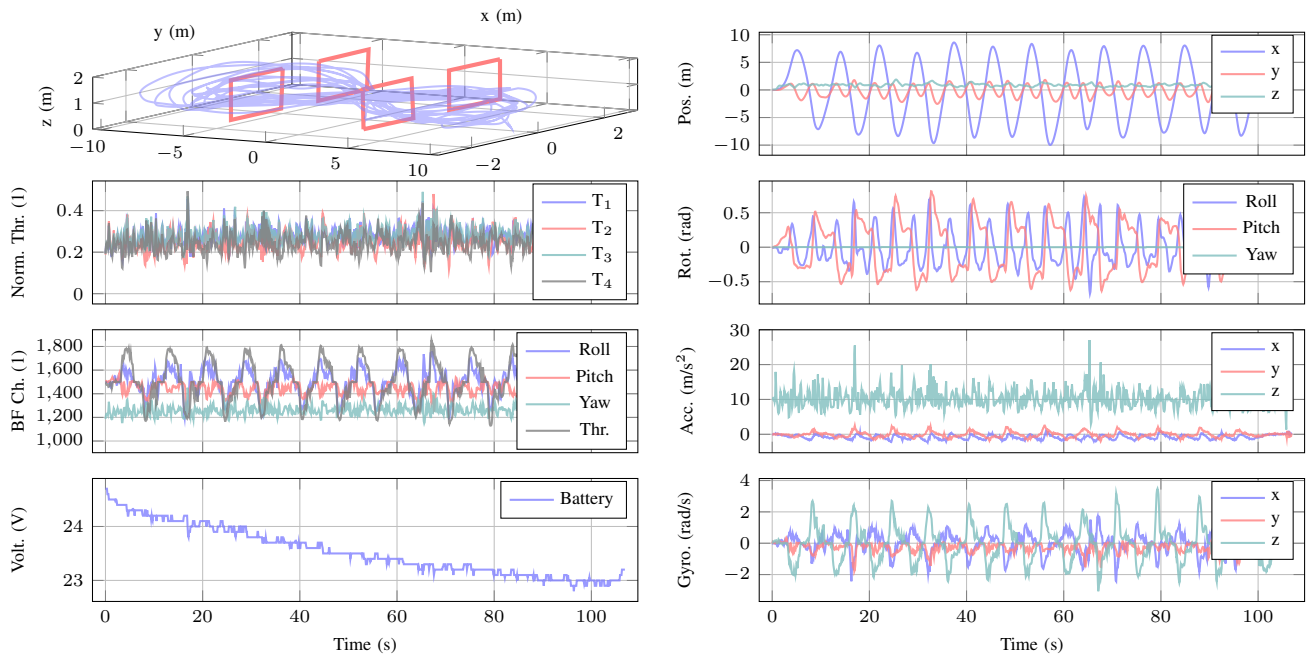


Fig. 6: Sample output of the utility script `data_plotting.py` (Section VI), plotting 21 different data points for `flight-07p-lemniscate`, a piloted, 13-lap flight on the lemniscate trajectory.

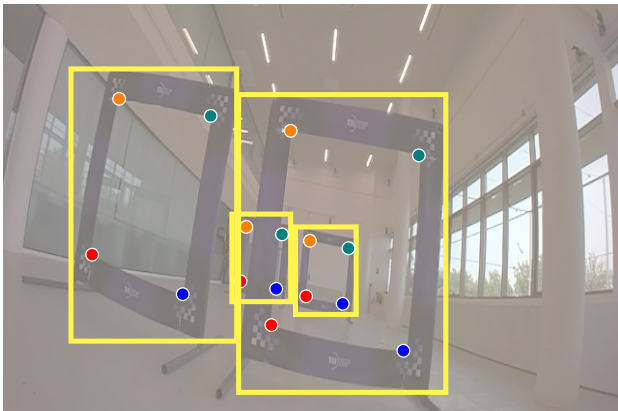


Fig. 7: Sample output of script `label_visualization.py` from Section VI, overlaying the bounding boxes and keypoints (top-left, top-right, bottom-right, and bottom-left corners) for all obstructed and unobstructed gates in one of the frames from `flight-02a-ellipse`.

[6] “AlphaPilot – Lockheed Martin AI Drone Racing Innovation Challenge,” <https://www.herox.com/alphapilot/teams>, 2019, [Online].

[7] A. Antonini, et al., “The blackbird dataset: A large-scale dataset for uav perception in aggressive flight,” in *Proceedings of the 2018 International Symposium on Experimental Robotics*, J. Xiao, et al., Eds. Cham: Springer International Publishing, 2020, pp. 130–139.

[8] M. Burri, et al., “The EuRoC micro aerial vehicle datasets,” *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[9] K. Sun, et al., “Robust stereo visual inertial odometry for fast autonomous flight,” *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[10] C. Pfeiffer et al., “Human-piloted drone racing: Visual processing and control,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3467–3474, 2021.

[11] L. Bauersfeld, et al., “NeuroBEM: Hybrid aerodynamic quadrotor model,” in *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, jul 2021.

[12] P. Newman et al., “Data papers—peer reviewed publication of high quality data sets,” pp. 587–587, 2009.

[13] A. Geiger, et al., “Vision meets robotics: The kitti dataset,” *International Journal of Robotics Research (IJRR)*, 2013.

[14] W. Zhao, et al., “UTIL: An ultra-wideband time-difference-of-arrival indoor localization dataset,” 2022.

[15] D. Hanover, et al., “Autonomous drone racing: A survey,” 2023.

[16] A. Loquercio, et al., “Learning high-speed flight in the wild,” *Science Robotics*, vol. 6, no. 59, 2021.

[17] P. Foehn, et al., “Alphapilot: autonomous drone racing,” *Autonomous Robots*, vol. 46, no. 1, pp. 307–320, Jan 2022.

[18] A. L. Majdik, et al., “The Zurich urban micro aerial vehicle dataset,” *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 269–273, 2017.

[19] P. Foehn, et al., “Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight,” *Science Robotics*, vol. 7, no. 67, p. eabl6259, 2022.

[20] C. de Wager, et al., “The artificial intelligence behind the winning entry to the 2019 ai robotic racing competition,” *ArXiv*, vol. abs/2109.14985, 2021.

[21] E. Kaufmann, et al., “Beauty and the beast: Optimal methods meet learning for drone racing,” *2019 International Conference on Robotics and Automation (ICRA)*, pp. 690–696, 2018.

[22] MultiWii, “Multiwii serial protocol,” [http://www.multiwii.com/wiki/index.php?title=Multiwii\\_Serial\\_Protocol](http://www.multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol).

[23] Betaflight, “The betafight open source flight controller firmware project,” <https://github.com/betaflight/betaflight>.

[24] MultiGP, “Multigp drone racing gate,” <https://www.multigp.com/product/drone-racing-gate-bundle/>.

[25] M. Faessler, et al., “Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories,” *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2018.

[26] D. Falanga, et al., “Pampc: Perception-aware model predictive control for quadrotors,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8.

[27] H. X. Pham, et al., “Deep learning for vision-based navigation in autonomous drone racing,” *Deep Learning for Robot Perception and Cognition*, 2022.

[28] K. Chen, et al., “Mmdetection: Open mmlab detection toolbox and benchmark,” 2019.

[29] M. Contributors, “Openmmlab pose estimation toolbox and benchmark,” <https://github.com/open-mmlab/mmpose>, 2020.

[30] “COCO - Common Objects in Context - Data format,” [Accessed 30. Aug. 2023]. [Online]. Available: <https://cocodataset.org/#format-data>

[31] K. Shoemake, “Animating rotation with quaternion curves,” in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’85. New York, NY, USA: Association for Computing Machinery, 1985, p. 245–254.