ALMA MATER STUDIORUM

UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE

AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Smart Vehicular Systems

GATES N' POSES: ON THE AI-WAY TO SPEED OF PERCEPTION IN AUTONOMOUS RACING DRONES

CANDIDATE

Enrico Pallotta

SUPERVISOR

Prof. Giovanni Pau

Academic year 2022-2023

Session 2nd

Abstract

This thesis explores the application of AI in the realm of autonomous racing drones. The racing industry has historically driven the evolution of robust and efficient systems, with the recent focus shifting towards self-driving mechanisms. The challenges in this domain are complex, requiring precise control and perception systems with minimal reaction times.

The research conducted at the Autonomous Robotics Research Center of the Technology Innovation Institute is presented, with a particular emphasis on perception systems for autonomous racing drones. The study introduces an innovative high-speed dataset and a pioneering approach for gate pose estimation powered by state-of-the-art keypoints detection neural network.

For the purpose of evaluation, the research also presents an algorithm capable of reconstructing the map of an unknown track, leveraging the results provided by the gate pose estimation. This work contributes significantly to the field, enhancing the accuracy of perception systems and significantly lowering their computational complexity.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The racing world has always been a fundamental catalyst for the development of increasingly efficient and safe technologies. The extreme conditions to which vehicles are subjected, tends to amplify all the critical issues that can normally arise in a vehicle. This, over time, has enabled the development of increasingly efficient, effective and robust systems, significantly revolutionizing not only the world of racing but also the technologies that surround us in our everyday lives. In recent years, the need to develop self-driving systems has led to the emergence of races for autonomous vehicles such as cars and drones. Here the challenges become even more complicated, as it is necessary to develop control and perception systems that are able to operate very precisely and with very short reaction times.

In this master thesis, the research conducted at the Autonomous Robotics Research Center of the Technology Innovation Institute will be presented, focusing on the development of perception systems for autonomous racing drones. The primary emphasis will be on both the groundbreaking of an unseen high-speed dataset and a novel AI-powered approach for pose estimation of gates. The results of this work are extremely important for the future research, development and evaluation of new learning-based control and perception systems.

In order to have a meaningful evaluation of the gate pose estimation method,

an algorithm will be presented. This algorithm utilizes the results generated by the gate corner detector to reconstruct the map of an unknown track.

1.1 Drone racing

Drone racing is a high-speed, exhilarating sport where pilots operate small, remotely controlled drones through intricate and challenging courses. These drones are equipped with cameras that provide a real-time first-person view (FPV) to the pilots. With years of training, human pilots are able to navigate in complex environments at incredible speeds, reaching peacks of 50 m/s. It comes naturally that an interest has arisen in transferring, and possibly exceeding, those capabilities into self-driving drones. Such improvements in technology, would be very useful in a lot of application, like drones operating in emergency scenarios, for surveillance, search-and-rescue operations and many more.

Over the last five years, a few competitions have been organized, the most important ones being in correspondence with four editions of the *International Conference on Intelligent Robots and Systems* (IROS 2016-2018) and AlphaPilot [15] sponsored by Lockheed Martin. Similarly to races of human piloted drones, in autonomous competitions the drone has to go through gates, with a specific order, in the shortest time possible. In order to be able to operate autonomously, drones are equipped with a computation unit (usually an Nvidia Jetson GPU) and multiple onboard sensors such as IMU, cameras, rangefinders. Given the limited computational power and all the problems that affect these sensors at high-speeds (motion blur, light changes, motors vibrations), being able to emulate human performances, arises several challenges in perception, planning and control.

Chapter 2

Background

Since the first autonomous drone race at IROS 2016, several datasets and methods for gate detection have been proposed. In this chapter, a summary of widely recognized publicly available datasets and their distinguishing features will be provided. This overview will underscore the rationale behind the introduction of a new and expanded dataset. Subsequently, an examination of prior approaches employed by both researchers and race competitors in the context of gate detection will be conducted. This analysis aims to demonstrate the various ways in which such visual information can be leveraged.

2.1 Datasets

In the field of AI for perception systems in autonomous racing drones, the availability of suitable datasets plays a critical role in advancing research and development. Over the years, several datasets have emerged, each catering to specific aspects of this domain. In this section, a comprehensive review of existing datasets is conducted, discussing their characteristics and emphasizing the distinctive contributions of the newly created dataset.

Among the earlier datasets for multi-rotor Visual-Inertial Odometry (VIO) and Simultaneous Localization and Mapping (SLAM), the 2016 EuRoC [5]

and the 2017 Zurich Urban micro aerial vehicle MAV [28] datasets are notable. However, these datasets were primarily characterized by comparatively lower speeds and lower image capture frequencies. Such limitations proved inadequate for addressing the demanding requirements of drone racing, where high-speed maneuvering and real-time perception are paramount.

Recognizing the need for datasets that align with the aggressive flight nature of drone racing, recent years have witnessed a resurgence in dataset creation. The UPenn dataset released in 2018 [37] aimed to validate stereo VIO methods for fast autonomous flight but did not incorporate the presence of racing gates.

In 2019, Lockheed Martin contributed to the landscape by releasing an image dataset for Test #2 of its AlphaPilot challenge's virtual qualifiers [15]. However, this dataset lacked essential drone state information, which limited its applicability to comprehensive perception tasks. Concurrently, researchers recognized the need for benchmarks that focused on faster trajectories, as observed in the dataset presented in [12].

The Blackbird dataset [1], introduced in 2020, was designed to facilitate perception research in aggressive indoor flight scenarios. Notably, it combined real-world inertial data with high-resolution images generated in simulation environments.

Despite these efforts, a dataset that encompasses the full spectrum of high-speed autonomous flights, offers high-resolution, high-frequency RGB mono-camera images (comparable to what human pilots rely on), and provides comprehensive annotations, including precise gate corner labels, was notably absent until the creation of our dataset. Our dataset distinguishes itself from [12] by not only including piloted but also autonomous high-speed flights, thereby reflecting the diverse operational modes of racing drones. Moreover, it offers higher-resolution images captured under varying light conditions, rendering it more representative of the challenging real-world scenarios encountered in drone racing. Most importantly, the dataset encompasses both high-frequency

motion capture data and pixel-level annotations of gates and their corners, ensuring accurate and comprehensive ground truth information for research purposes.

Recent additions to the dataset landscape include [31] and [2], both specialized datasets catering to drone racing and aggressive multi-rotor flight. Additionally, [16] introduced an open-source, open-hardware racing drone, further enhancing the accessibility and reproducibility of research in this domain.

Compared to the existing literature [17], our dataset stands out in several key aspects. First, it represents the fastest autonomously flown dataset, pushing the boundaries of speed and maneuverability. Second, it offers high-resolution, high-frequency image data, capturing the nuances of different lighting conditions that are crucial for real-world racing scenarios. Lastly, our dataset is fully annotated, going down to the granularity of individual gate corners, which empowers researchers to delve into various aspects of autonomous drone racing, including Visual-Inertial Odometry (VIO), gate pose estimation [13, 21], and end-to-end control. In summary, our dataset provides a robust foundation for advancing research in AI-driven perception systems for high-speed autonomous racing drones.

2.2 Gate detection

Human pilots rely on gates as a key position reference, gates allow them to understand which are the next waypoints but also to have a relative estimate of their position in the 3D space. Indeed, one of the key requirements of an autonomous racing drone, is being able to detect gates. This can be done in several ways, at different semantic levels, depending on which is the objective. For example, if one has a good state estimation module, then relying on the simple bounding box detection (Section 2.2.1) of the gate can be enough to pass through the next waypoints. In most situations however, particularly

in drone races, drone cannot rely on an accurate estimate of the state. When an external motion capture or tracking system is not available, the common solution is to use measurements of the IMU coupled with the optical flow obtained by a stereo system, to perform VIO and estimate the state of the drone. Unfortunately, VIO is well known to be affected drift, which is particularly intensified at high speeds where the noise from the IMU increases and the optical flow struggles due to motion blur in images. The knowledge about shape, size and perhaps position of gates, can be leveraged to get an estimate of the drone's state, allowing to correct the drift. Thus, for the purpose of improving localization abilities, more complicated solution that aim to extract the pose of gates seen from the camera needs to be employed (Sections 2.2.2, 2.2.3).

2.2.1 Simple gate detection

In the early stages of autonomous drone racing navigation, the primary approach involved using the visibility of gates in the camera image stream to perform reactive control towards approximated waypoints. Many competitors focused on straightforward gate detection using either classical computer vision techniques or neural networks for object detection.

One noteworthy example is the work of Jung et al. [20], who achieved victory in the first autonomous drone racing competition at IROS 2016. They employed a color-based approach that effectively worked for that specific race, as the gates were monochromatic and bright orange. However, this approach tends to falter in the face of challenges such as varying light conditions and motion blur induced by high-speed flight. To address these limitations, Jung et al. proposed "ADRNet," a modified Single Shot MultiBox Detector (SSD) [25]. ADRNet's task was to predict a bounding box around gates, and their control module aimed to align the image frame center with the center of this bounding box. However, a weakness of this approach lies in the strong assumption that the center of the bounding box corresponds to the center of

the real gate. While this assumption holds true when the camera's optical axis is perpendicular to the gate and there is no lens distortion, in most real-world situations, these conditions are not met.

A significant advancement in harnessing the information that a visible gate can provide was made by [30]. They initially proposed an object detector very similar to ADRNet, utilizing a lightweight version of SSD with MobileNetV2 [35] as the backbone. This detector was employed to predict only the closest gate. Subsequently, they processed the grayscale version of the bounding box crop using another convolutional neural network (CNN) in conjunction with a fully connected layer to regress the distance to the gate. However, this approach did not yield high accuracy, as they reported an average error of 0.66 meters in distance prediction, even when the drone was operating at low speeds (maximum 2 meters per second) and the entire perception module was running off-board.

2.2.2 Gate Corners Detection

While simple gate detection methods can suffice when the drone operates at low speeds and benefits from a robust state estimation system, exploiting the presence of gates in images can provide invaluable information for mapping or state estimation in case of known tracks. In such contexts, estimating the relative pose of visible gates becomes crucial. Given the well-known shape and size of gates, along with the camera's intrinsic parameters, detecting the corners of gates allows for accurate pose estimation.

Over the years, various approaches have been employed to detect gate corners, depending on race settings and hardware constraints. Some of these methods rely on classical computer vision techniques, as seen in "Snake-gate" [22] and "Lines" [36]. These methods are particularly suitable for drones with limited computational power. However, they still face common challenges posed by noise, motion blur, and varying light conditions.

Schmidt [36] proposed solutions based on object detection models, such as YOLOv5 [33] and Faster-RCNN [34], and semantic segmentation of corners. While this paper offers a valuable performance comparison of different approaches, these methods were applied out of context on the Alphapilot dataset [15]. Consequently, issues related to partial occlusion, corner-to-gate correspondence and computational constraints were not fully addressed.

In the Alphapilot 2019 competition [15], the winning team and the second-place team both relied on deep neural networks for gate detection and pose estimation. The winning team employed ”GateNet” [13], a UNet architecture trained on a private manually labeled dataset consisting of 2336 images. They employed a ”de-rotation” step and subsequently applied the Snake-gate algorithm [22] to locate the corners of gates. In contrast, the second-place team [15] introduced an effective method for solving the corner-to-gate correspondence problem when multiple gates were present in the image. They employed a UNet to predict both corner masks and Part Affinity Fields (PAFs), adapting a technique originally proposed for human pose estimation [6]. Utilizing this information, they tackled a K-dimensional matching problem [39] to construct coherent ”bodies” of visible gates.

2.2.3 End-to-end gate pose estimation

A significant milestone in the field of autonomous racing drones was achieved at IROS 2018 when Kaufmann et al. [21] secured victory using the first end-to-end image-to-pose approach. They employed a CNN with two separate Multi-Layer Perceptron (MLP) heads. One head was dedicated to regressing the relative pose of the closest gate, while the other provided an estimate of uncertainty. This uncertainty information was particularly valuable, as it was used to update an extended Kalman filter, maintaining a global estimate of the race track. The network was trained on a private dataset that featured only a single non-squared gate in various environments. This choice was rationalized

by the perception system's primary objective of estimating the pose of the next gate. However, as noted in [15], the neural network struggled when multiple gates were simultaneously visible.

Following in the footsteps of the IROS 2018 winners, Pham et al. [32] presented a similar approach. They proposed a CNN with a single fully connected layer on top, responsible for regressing various parameters, including the confidence value, gate center pixel coordinates, distance, and orientation of visible gates. Their work demonstrated that integrating such detections into multiple Extended Kalman Filters (EKFs), one per gate, allowed for a precise estimate of the racing track.

These end-to-end approaches mark a significant advancement in gate pose estimation, as they aim to provide a holistic understanding of the drone's position and orientation relative to the race track based solely on visual input without any further processing step. However, ongoing research continues to address challenges associated with handling multiple visible gates and enhancing the robustness of end-to-end pose estimation in complex racing environments.

Chapter 3

High-Speed dataset

In this chapter, the methodologies and processes employed in collecting and annotating the high-speed dataset are explored. It begins with an introduction to the drone platform, its hardware components, and sensor systems, followed by an exploration of the software stack. The focus will then move to the indoor arena setup, a critical environment for our experiments. The heart of this chapter lies in our detailed examination of the collected visual data and the precise labeling methods employed, which are essential for our research. The culmination of the efforts in creating this dataset has led to the submission of a paper to *IEEE Robotics and Automation Letters* (RA-L). While this thesis primarily concentrates on the visual data collected by the drone, for an extensive understanding of the entire data collection process and guidance on utilizing the dataset, it is recommended to refer to our original paper that will be soon released. Some information, figures, and tables presented in this thesis have been directly sourced from the mentioned paper.

3.1 Drone Setup

To facilitate the collection of our dataset, a custom quadrotor, as depicted in Figure 3.1, was built. Its design is centered around a 5” carbon-fiber frame with a propeller-to-propeller diagonal spanning 215mm. The fully-assembled

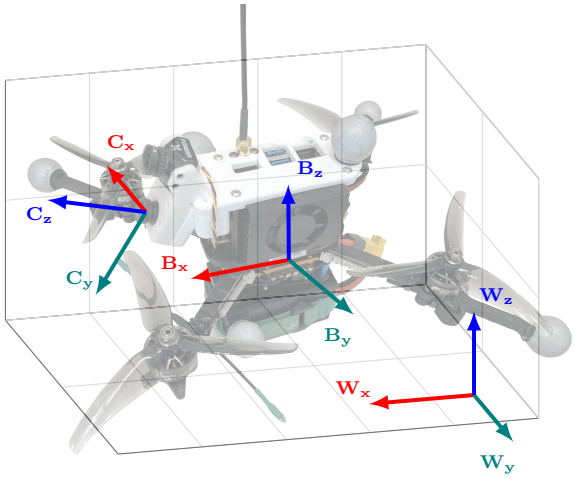


Figure 3.1: The drone platform used to record the dataset, the body frame B has its origin at the FCU's IMU location, the camera frame C is located where the bottom lens is (the top lens being the one of the FPV system).

drone, inclusive of the battery, possesses a weight of approximately 870g, while boasting an impressive maximum speed of 179km/h. This high-speed capability is essential for executing the aggressive maneuvers demanded in drone racing.

One noteworthy feature of our design is its adaptability, which enables it to function seamlessly as both an autonomous racing drone and a human-piloted First-Person View (FPV) racing drone. This dual-purpose configuration serves a critical purpose: it establishes an authentic testing environment for benchmarking the performance of autonomous drone racing against human counterparts.

3.1.1 Design

The quadrotor comprises three main sub-systems: *(i)* quadrotor electronics, *(ii)* an autonomous module, and *(iii)* a First-Person-View (FPV) system. These sub-systems are integrated using the frame and fasteners. The assembled system includes two cameras: one digital camera connected to the autonomous module and one analog camera used by the human pilot in the FPV system.

Both cameras share the same mount, with the FPV camera positioned above the digital one.

The quadrotor electronics *(i)* consist of the electronic speed controller (ESC), the Kakute H7 v1 flight controller unit (FCU), the radio controller (RC) receiver, and the battery. These components are mounted underneath the frame, with aluminum standoffs connecting the frame and a 3D-printed custom battery cage. The FCU hosts an STM32H7 microcontroller and is capable of running various firmware, including Ardupilot and PX4.

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

The FPV system *(iii)*, independent from the autonomous module and designed for human piloting, includes an analog camera, a video transmitter, and their respective antennas, all placed above the frame.

3.1.2 Sensors

The quadrotor is equipped with three primary sensors for autonomous and piloted aggressive flight: *(i)* an IMU, *(ii)* an RGB camera, and *(iii)* an FPV camera.

(i) The IMU, an InvenSense MPU6000, is embedded into the FC and provides precise real-time tri-axis angular rate sensor (gyroscope) data and accurate tri-axis accelerometer data.

(ii) The camera, an Arducam IMX219 8MP RGB Bayer, is part of the

autonomous module and captures 640x480 pixel frames at 120Hz with a diagonal field-of-view (FOV) of 175°. The image is converted to BGR format for processing and analysis. This conversion is performed by the NVIDIA GStreamer plugin on the NVIDIA companion computer.

(iii) The FPV camera is a Foxeer T-Rex Mini 1500TVL with 6ms of latency. It is used by a human pilot in conjunction with a pair of 1280x960 OLED Fat Shark HDO2 goggles.

3.1.3 Software

The FC firmware used is Betaflight 4.3.1 [4], with proportional-integral-derivative controller (PID) tuning performed by a human pilot. The companion computer communicates with the FC using Multiwii Serial Protocol (MSP) to send commands and read sensor data. To ensure safety, Betaflight's `MSP_OVERRIDE` feature is activated, allowing the human pilot to override motor commands with an RC controller for emergency disarming.

On the Orin NX module, NVIDIA JetPack 5.1.1 is installed, which includes Jetson Linux 35.3.1 Board Support Package (BSP) with Linux Real-Time Kernel 5.10 and an Ubuntu 20.04-based root file system with CUDA 11.4 support. The Robot Operating System 2 (ROS2) [27] Humble distribution is used as the middleware for communication between the perception, planning, and control modules.

3.2 Arena Setup

The dataset was recorded in TII's indoor arena (Figure 3.2), measuring 24 x 9.7 x 7 meters. The arena is equipped with a Qualisys motion capture system (MoCap) comprising 32 Arqus A12 cameras, which track the 6DoF poses of defined rigid bodies with millimeter accuracy at 275Hz.

To prepare the motion capture system for use, calibration is required. An L-shaped arrangement of markers is placed in the center of the arena, defining

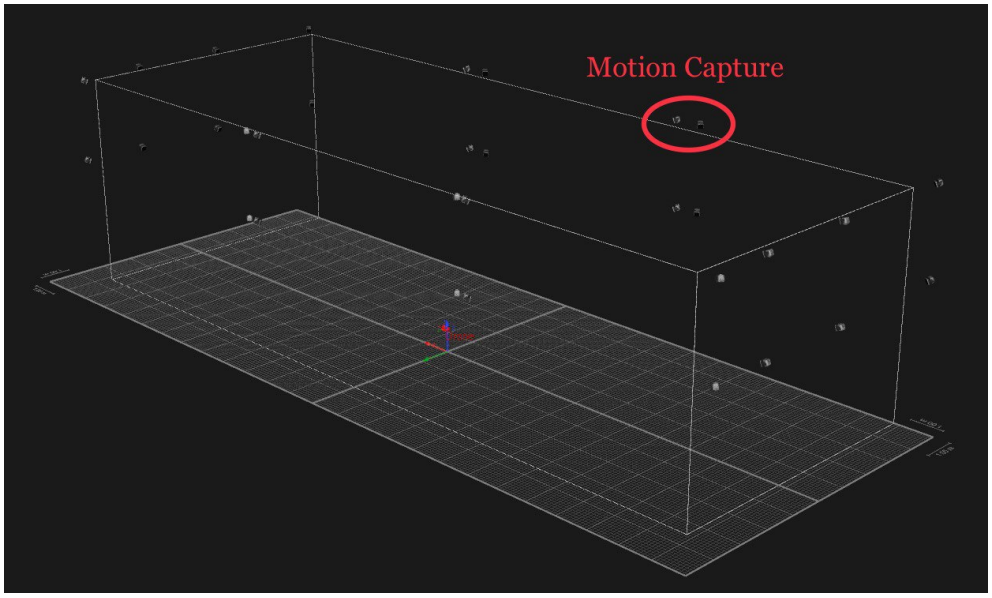


Figure 3.2: 3D view of the arena from Qualysis software, the 3d bounding box determines the area covered by the motion capture system. The position of all the 32 cameras is visible.

the axes' direction and the motion capture reference frame's center position. The x-axis aligns with the arena's longest direction, while the z-axis points upward. Calibration involves recording a sequence of movements using a T-shaped wand with markers on the sides.

3.3 Data collection

The dataset contains a total of 24 flights (Table 3.1): 12 *human-piloted* and 12 *autonomous* ones. In either case, two shapes—*ellipse* and *lemniscate* (Figure 3.3)—have been executed 6 times, using the same gate configuration in each run. In the course of the six repetitions, various settings of light conditions and motion blur can be observed, as indicated in Table 3.2. These settings are determined by the combination of lighting (on/off), blinds (up/down), and exposure time (auto/fixed). Recording at 120 fps, resulted in a high number of images : 177030 in total. Using different combination of exposure and lights resulted in a large variability in the recorded images (Figure 3.4).

Control	Trajectory	Top Speed	Time	Distance
Autonomous	Ellipse	**21.83 m/s**	149.32 s	526.15 m
	Lemniscate	10.22 m/s	155.32 s	480.67 m
Piloted	Ellipse	9.50 m/s	575.62 s	3355.67 m
	Lemniscate	8.93 m/s	**594.60 s**	**3577.65 m**

Table 3.1: Summary of the flights recorded in the dataset

Position, orientation and velocity of the drone were recorded using Qualisys, while linear and angular acceleration were collected by the onboard IMU. In the end, all these information were synchronized according to the timestamps, and, beside the ROS bags and raw data, a `csv` file was generated for each flight. In order to use computer vision methods for objects pose estimation, the camera calibration results (intrinsic matrix and distortion parameters) and the set of images used were released too. Using Qualisys, by placing a marker in place of the camera, an estimate of the translation between the drone and camera pose was computed. Regarding the rotation, we assumed them to be aligned, except for the x axis, as the camera was placed at $40°$ and $50°$ during lemniscate and ellipse trajectory, in order to compensate for the high pitch, due to high-speed, and make gates more visible. For each flight, the position and orientation of the gates was also recorded, in order to allow the development and evaluation of any perception module.

3.4 Labeling

As far as our research has revealed, there has been no previous dataset that encompasses images captured from the First-Person View (FPV) of a real drone operating at high speeds, featuring comprehensive labeling of all gates and their associated corners, even when occluded. My primary contribution to this dataset centered around the meticulous labeling of images, involving the delineation of bounding boxes around each gate and the assignment of corners

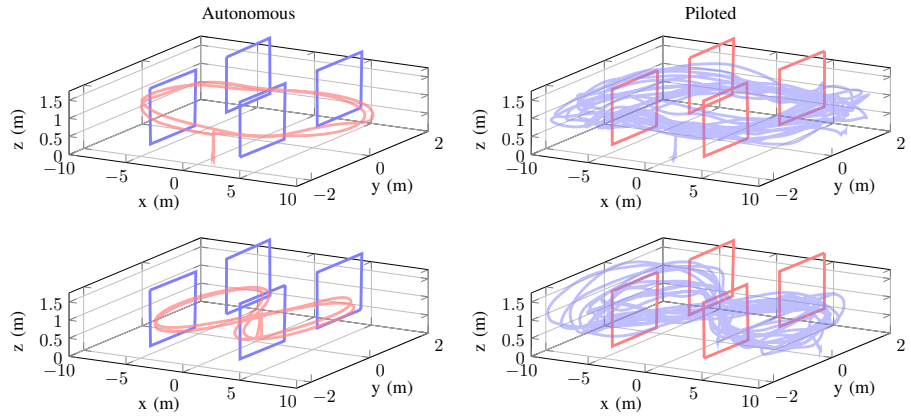
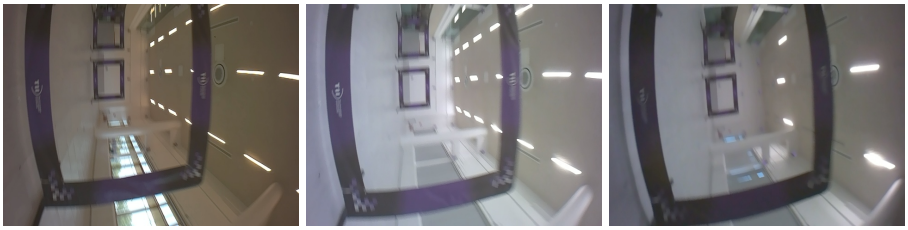


Figure 3.3: Examples of recorded trajectories on a 4-gate track: an ellipse (top) and a lemniscate (bottom).

Flight	Trajectory	Lighting	Motion Blur	Autonomous	Piloted
1	Ellipse	high	high	2699	12666
2	Ellipse	high	normal	2944	11316
3	Ellipse	medium	high	2903	10821
4	Ellipse	medium	normal	3494	12811
5	Ellipse	low	high	2773	10071
6	Ellipse	low	normal	3040	11201
7	Lemniscate	high	high	3692	12831
8	Lemniscate	high	normal	3651	10461
9	Lemniscate	medium	high	3001	11224
10	Lemniscate	medium	normal	3314	11201
11	Lemniscate	low	high	2668	12861
12	Lemniscate	low	normal	2806	12581
			Total	36985	140045

Table 3.2: Summary of the images recorded in each flight

(a) Flight 01 (b) Flight 04 (c) Flight 06



(d) Flight 07 (e) Flight 09 (f) Flight 11

Figure 3.4: Examples of recorded images during autonomous flights

to their respective gates.

Prior datasets were primarily designed for applications involving simple object detection, semantic segmentation, or pixel-wise methods. Consequently, none of these datasets included labels for occluded gates. Thus, the question arises: How was the annotation of all these images managed?

One straightforward solution would have been to utilize perspective projection. Given the recorded positions of both the drone and the gates through the motion capture system, and the knowledge of camera intrinsics and lens distortion parameters, one could intuitively project the four corners of each gate back into the image frame. However, practical implementation presented challenges in the context of real-time data acquisition. Issues such as network instability leading to packet loss from the motion capture system, small tracking inaccuracies in the recorded objects, and the complexities of synchronizing images with their corresponding poses collectively contributed to deviations between the projected points and their actual pixel positions.

The alternative approach adopted, involved the creation of a synthetic dataset, followed by training a robust and high-performance model using this synthetic data. This model was subsequently employed for automated labeling

after a fine-tuning on manually labelled real data. The auto-labeling pipeline we devised can be summarized as follows:

1. Creation of a synthetic dataset using Blender[9].
2. Pretraining of a two-stage top-down keypoint detector [8, 10].
3. Fine-tuning of the detector using a small set of manually labeled real-world images.
4. Iterative review, correction, and fine-tuning of the generated labels.

The last step was an ongoing process, meticulously refined until all generated labels met the requisite quality standards.

3.4.1 Data Generation Pipeline

To construct a 3D model of the racing gates and render them in random positions, Blender[9] was utilized in conjunction with a Python script plugin known as [7], which facilitated automated rendering.

Within the Blender environment, a world configuration was established, comprising five gates, four distinct light sources, and a camera sharing identical image resolution (640x480) and intrinsic parameters as those estimated from the real camera through calibration.

The Python rendering script systematically applied random translations and rotations to all elements in the world, ensuring that gates did not intersect with one another. To impose some degree of coherence and prevent entirely random camera poses, a constraint was placed on the camera to track one of the gates.

A total of 50,000 images were generated, each accompanied by 4x4 transformation matrices (with respect to the world reference frame) for all objects in the scene, which were stored in a numpy file.

For each rendered image, initially in PNG format with an alpha channel, a random background from the COCO[24] 5k images validation set was applied.

To simulate the motion blur characteristic of real flight images, random sampled kernels of varying dimensions (3x3, 5x5, or 7x7) were applied through convolutions.

By employing the stored pose information of each object alongside the camera's intrinsic parameters, the corners of each gate were projected into the rendered image frame, and their corresponding bounding boxes were extracted.

Concerning the labeling format, adherence was maintained to the conventions outlined by COCO[23] for keypoint detection and the standard format for bounding boxes. Each gate label encompassed the following attributes:

- Bounding box: c_x, c_y, w, h
- The four inner corners: $tl_x, tl_y, tl_v, tr_x, tr_y, tr_v, br_x, br_y, br_v, bl_x, bl_y, bl_v$

In this notation, 'c' represents the center, 't' signifies the top, 'b' denotes the bottom, 'l' represents the left, and 'r' stands for the right. Additionally, each keypoint 'k' featured a third value, 'k_v', indicating its visibility. As per COCO conventions, visibility values carried the following interpretations:

- 0: Keypoint is out of the image.
- 1: Keypoint is occluded.
- 2: Keypoint is clearly visible.

The rendering plugin also facilitated the storage of segmentation masks for each gate, allowing to distinguish between visible and occluded keypoints.

3.4.2 Auto-labeller Pre-training

A top-down keypoints detector was used to automate the labeling process for all images. The decision of using such type of model is well detailed in Section 4.1. This choice implied to utilize the widespread Pytorch-based framework developed by Open-MMLab for training both the object detection [8]

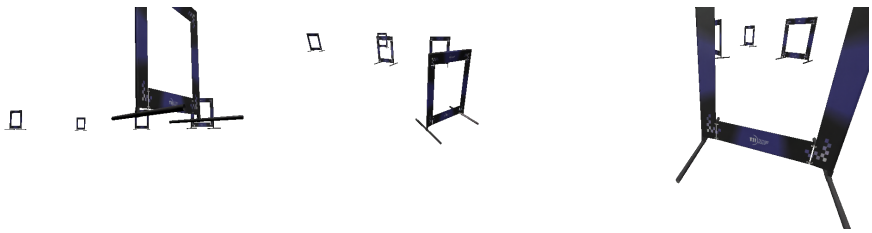


Figure 3.5: Examples of synthetic images generated in Blender

and keypoint detection [10] networks.

This modular framework facilitated the design of a top-down keypoints detector consisting of two independently trained networks. Users can select from a variety of available models, and our selection leaned towards the highest performing models based on the Average Precision (AP) scores provided by the MM library. Due to computational constraints on an 80GB Nvidia A100 GPU, the choice was made to employ the large versions of the object detector RTMDET [26] and keypoints detector RTMPOSE [18]. These models were initially trained separately. However, during inference for auto-labeling, they operated in a cascade mode. The bounding box detected by the first model defined the region in which the second model sought to locate keypoints.

As first training session, the synthetic dataset was employed, training on the 80% of the generated images and using the rest for validation.

3.4.3 Auto-labeller Fine-tuning

In computer vision applications for robotics, there is often a demand for high-quality pixel-level annotations. Manual labeling of such data is an extremely time-consuming endeavor. In situations where employing specialized sensors to collect this information, such as LIDAR or stereo systems for obtaining depth maps, is impractical, a common solution involves generating synthetic images and their corresponding annotations using computer graphics and simulation tools.

Even when ultra-realistic graphics engines are employed, the generated

data may not closely resemble the distribution of real-world data. This disparity is commonly referred to as the ”sim-to-real” gap, leading to the potential underperformance of models trained solely on synthetic data when tested on real data.

To address this challenge, a widely adopted strategy is to collect a relatively small sample of labels for real images and employ them to fine-tune the pre-trained model. In line with this approach, 5000 images randomly selected from the entire dataset were manually annotated. These images were subsequently divided into training and test sets (80-20%). Fine-tuning sessions for both the object and keypoints detectors were conducted.

The fine-tuning process resulted in a highly performing model capable of detecting gates and their corners even when they were partially occluded. Initially, this model was used to automatically generate labels for the 12 autonomous flights. Following meticulous validation and any necessary manual corrections, the model underwent further fine-tuning, ultimately leading to improved performance. This enhanced model significantly streamlined the labeling process for the remaining piloted flights.

Chapter 4

Pose estimation

As discussed in Section 2.2, the estimation of drone racing gate poses primarily falls into two categories: corner detection followed by Perspective-n-Point (PnP) utilizing camera parameters, or training an end-to-end neural network. The latter approach, while potentially offering faster inference times, was deemed impractical due to its limitations in handling multiple gates and adapting to different environments.

Many proposed methods have focused on corner detection by relying on full gate segmentation masks, whether based on color [22] or UNet-based [13] approaches, followed by post-processing to identify gate corners. Phoen et al. [15] adopted a UNet model to directly predict the segmentation masks of corners and Part Affinity Fields (PAFs), which were subsequently used in post-processing to establish corner-to-gate associations.

However, these methods encounter two main challenges: they struggle in scenarios with multiple overlapping gates, potentially with corners obscured by other gates, and they require post-processing steps to handle segmentation masks and solve corner-to-gate correspondence.

In this chapter, the focus will be on how state-of-the-art keypoint detection models can be utilized to precisely identify the pixel coordinates of gate corners, effectively addressing the challenges mentioned earlier. Furthermore, an algorithm will be introduced that utilizes PnP and the drone's state estimate

(position and orientation) for accurate reconstruction of the racing track.

4.1 Keypoint Detection

The employed approach to address both of these challenges in gate pose estimation was to treat it as a keypoints detection task. Many solutions have been developed for detecting keypoints, primarily designed for human pose estimation following the COCO [23] standard, which defines 17 keypoints for the human body. These solutions can generally be categorized as top-down and bottom-up methods.

Top-down methods involve a two-stage process where object detection is performed first, followed by keypoints detection for each detected object. However, this approach, due to its reliance on two networks and increasing complexity with the number of detected objects, is not suitable for real-time applications and can fail if the bounding box detection is imprecise. In contrast, bottom-up approaches aim to detect keypoints for all objects in the scene simultaneously and then employ post-processing methods to group keypoints and construct the structure of each object. The approach proposed by [15] can be considered a bottom-up keypoints detection method.

A solution that combines the advantages of both top-down and bottom-up approaches is the one presented by Maji et al. [29] for efficient multi-person pose estimation in YOLO-Pose. The following method for gate pose estimation relies on a variant of this keypoint detector, available within the YOLOv8 [19] framework.

4.2 YOLO

You Only Look Once (YOLO) is one of the most prominent and widely adopted single-stage object detection systems. Introduced by Joseph Redmon et al. [33] in 2016, YOLO has undergone numerous iterations and improvements over

the years by different research groups, focusing on enhancing its capabilities while maintaining a strong emphasis on the speed-accuracy trade-off. Today, it stands as the de facto standard solution for many real-time computer vision applications.

For an in-depth review and detailed explanation of YOLO's inception, evolution, and contributions from various research groups, readers can refer to an extensive resource [38].

One notable group that has played a significant role in advancing YOLO and making it accessible to a wider audience is Ultralytics. Ultralytics, the developers behind YOLOv5, introduced a PyTorch-based version of YOLO, along with a user-friendly Python library that simplifies the training, validation, and deployment pipeline for various YOLO variants. Building on their extensive experience with YOLOv5, Ultralytics released YOLOv8 [19] in the most recent development. This version incorporates several innovations from intermediate YOLO iterations, including the transition to an anchor-free model (based on the TOOD implementation [14]) and the use of decoupled heads for different tasks.

In addition to its traditional capabilities in image classification and object detection, YOLOv8 extends its utility to segmentation, object tracking, and, as of April 2023, pose estimation tasks. For the latter, Ultralytics drew inspiration from the work of Maji et al. [29] on human pose estimation, particularly human keypoints detection.

Ultralytics adopted a straightforward approach to keypoints detection: similar to bounding box predictions, adedicated head directly regresses the coordinates of 17 keypoints on a human body. This implementation utilizes the same model architecture as YOLOv8 for object detection but includes an additional and decoupled head specialized in predicting keypoints. Importantly, this approach ensures that the model's performance in detecting objects and keypoints remains largely independent, allowing for accurate keypoints localization even when bounding box predictions are imprecise.

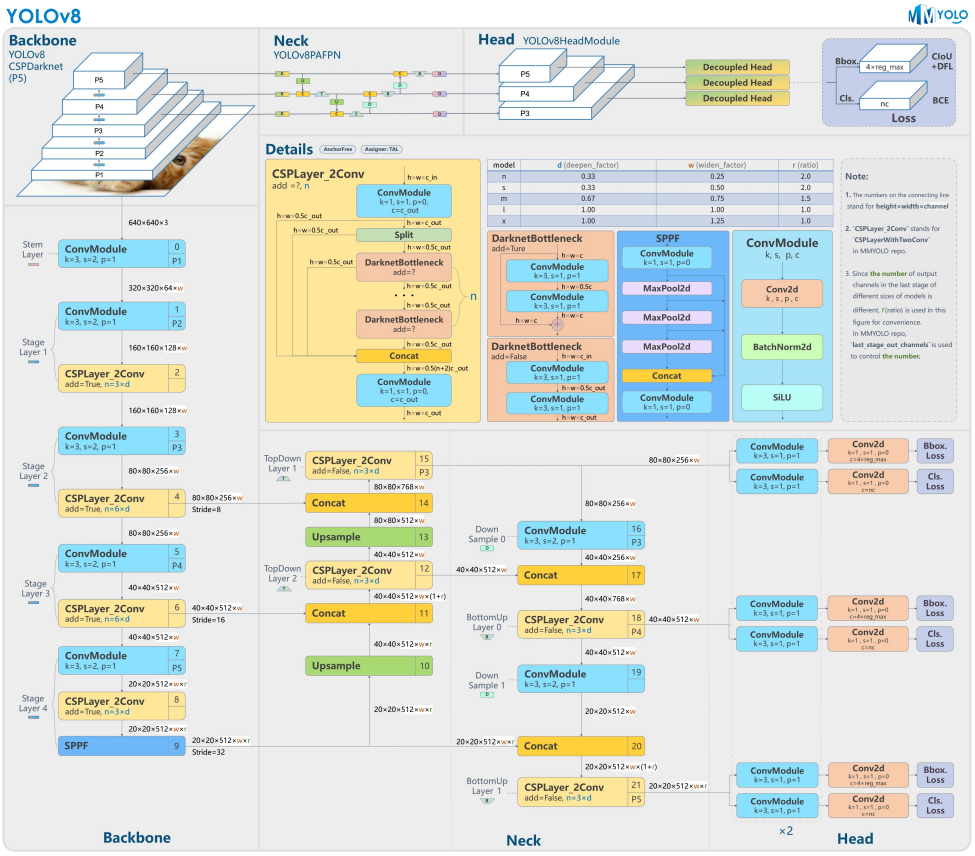


Figure 4.1: Detailed YOLOv8 architecture [11]. Only the object detection head is shown, however the network is the same for the keypoint detection task.

4.2.1 Metric and Loss

A notable innovation in the keypoint detector lies in its loss function, which is directly tied to the target metric. The COCO dataset [23] introduced not only a human keypoint dataset but also defined the evaluation criteria for model performance. In particular, COCO introduced the Objects Keypoint Similarity (OKS) metric, analogous to Intersection over Union (IoU), enabling the use of classic Average Precision (AP) and Average Recall (AR) metrics for keypoints. For an object with N keypoints, the OKS is defined as follows:

$$OKS = \frac{\sum_i^N \left[\exp\left(-\frac{d_i^2}{2s^2\kappa_i^2}\right)\delta(v_i > 0)\right]}{\sum_i^N \delta(v_i > 0)} \in [0,1];\ \kappa_i = 2\sigma_i \tag{4.1}$$

In this equation:

- d_i represents the Euclidean distance between the ground truth and detected keypoints.
- v_i is the ground truth visibility flag for each keypoint.
- The distance is transformed by an unnormalized Gaussian distribution with a standard deviation of $s\kappa_i$, where s represents the size of the object (determined by the bounding box).
- κ_i essentially controls the size of the Gaussian distribution around a specific keypoint, defined as $2\sigma_i$.

The choice of σ values strongly influences the OKS outcome. Smaller σ values demand higher precision for a given keypoint, making it more challenging to achieve a high similarity value.

In YOLO, OKS is not solely used as a similarity measure for AP/AR computation; it is directly incorporated into the keypoint loss function:

$$\mathcal{L}_{kpts} = 1 - OKS \tag{4.2}$$

For model evaluation, the mean average precision (mAP) is computed as the average of results obtained using multiple threshold values of OKS. Specifically, the considered interval will be the common $[0.5, 0.95]$ with a step of 0.05.

4.3 Training

Ultralytics has released multiple versions of the model at different scales, ranging from the ”nano” version with 3.3 million parameters to the ”xlarge” version with 69.4 million parameters. Given the computational constraints of the onboard device and the desire for the network to perform at or above the camera's frame rate (120Hz), the choice was made to utilize the smaller version of YOLOv8. Consequently, the pre-trained model, `yolov8n-pose.pt` (nano), was chosen, and its default 17-keypoints detection head was replaced with one designed for detecting four gate corners.

Training experiments were conducted at different input resolutions, including 640^2, 448^2, 224^2, and 192^2, to find the right trade-off between accuracy and inference time. The choice to work with these input resolutions, despite the camera recording at 640×480, arises from the YOLO framework's requirement for squared input shapes when using multi-GPU training and certain data augmentations.

Several data augmentations were applied during training, including $\pm 10\%$ resize and translation, HSV color transformations, and random left-right flips. In the initial epochs, mosaic augmentation was also employed. `AdamW` optimizer along with a linear learning rate scheduler were chosen for the training procedure.

The setting of the σ values, as explained in Section 4.2.1, plays a crucial role since OKS is directly used in the loss function. While COCO [23] assigns low values (0.025) to small body parts like ears and nose and higher values (0.107) to larger body parts like hips, estimating the pose of the gates requires

high precision in the detection of corners. Therefore, all the four σ values were set to 0.025.

Attempting to train with such low σ values initially did not yield high mAP scores (see Figure 4.2). Training a model from scratch directly on a challenging task is known to be suboptimal. To tackle this issue, the concept of "Curriculum Learning" as proposed by Y. Bengio et al. [3] was adopted. In this approach, the training data is structured in a curriculum or sequence, with each step representing varying levels of task complexity. The model is trained on simpler examples first and is progressively exposed to more challenging ones.

Therefore, a training pipeline with multiple steps was defined, at each step a lower value for the σ values is used. The pipeline consisted of three training steps with σ values of 0.25, 0.05, and finally 0.025. In each step, the model pretrained in the previous step was fine-tuned for 50 epochs starting from a lower learning rate.

During the authorship of this thesis, labels for all the human-piloted recorded flights were still under review. As a result, only the autonomous flights were utilized for training and evaluating the performance of YOLO and the mapping algorithm. The evaluation was conducted on two flights, one per trajectory, with flights 6 and 9 serving as the test/validation set, while the remaining flights were used for training.

4.4 Evaluation

The models were tested on two different trajectories: an ellipse trajectory recorded under low-light conditions and a lemniscate trajectory with normal lighting but high motion blur.

First, let's discuss the benefits of the curriculum training strategy employed. A comparison between a model trained at an input resolution of 640^2

with and without our custom training pipeline reveals the significant difference. Results from straightforward training (Figure 4.2) demonstrate that the model struggles to accurately localize keypoints when its head is trained from scratch. In contrast, Figure 4.3 shows results obtained by the model fine-tuned in the last curriculum training step. Thanks to the previous training at $\sigma = 0.05$, it starts from a lower loss and achieves a 98.86 mAP. Table 4.1 illustrates how using a lower input image resolution decreases the performance in terms of mAP. The 3% difference between 640^2 and 192^2 resolutions, suggests that the model's performance is relatively stable even with coarse images.

While standard metrics are valuable for model selection and hyperparameter tuning, they may not provide meaningful insights for deployment in robotic applications. In robotics, it's common to conduct a final evaluation using metrics directly related to the task at hand. For this purpose, inference results from models trained at different input resolutions were used to construct a map of the race track using the algorithm (1) described in Section 4.6. Consequently, the mean translation error of the gates in centimeters serves as the final evaluation metric. To establish a baseline, the ground-truth labels provided in the dataset were used, resulting in baseline errors of 13 cm and 28 cm for flights 6 and 9, respectively (see the "Label" column in Tables 4.2 and 4.3). These errors stem from various noise sources and problems detailed in Section 4.6. Additionally, synchronization delays between the Qualisys poses and recorded images introduce inaccuracies, with a more pronounced impact on the lemniscate flight due to its higher angular acceleration that imply high orientation variance in subsequent time units.

Taking the baseline errors into account, it's clear that the performance trends seen in Table 4.1 are reflected in the mapping results: models using high-resolution images exhibit lower mean translation errors.

Finally, an evaluation in terms of inference speed is crucial, particularly in the context of drone races. After optimization (explained in Section 4.5), an inference speed test was conducted with each model. The results in Figure

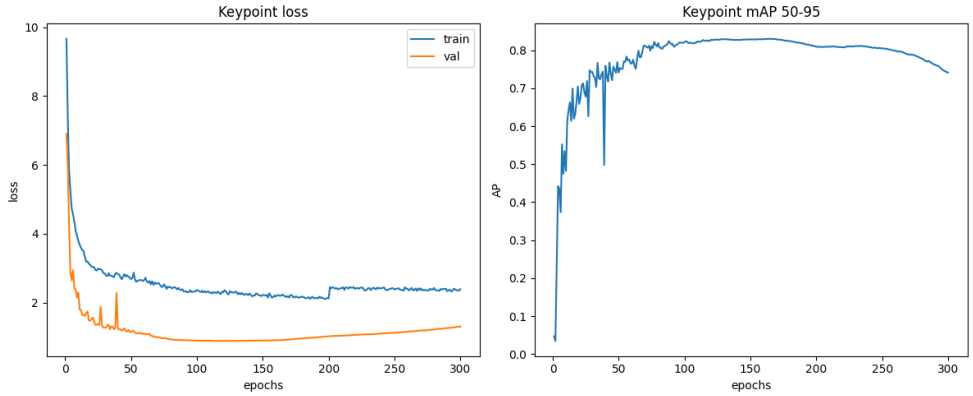


Figure 4.2: Straight forward training with $\sigma = 0.025$ (640^2).

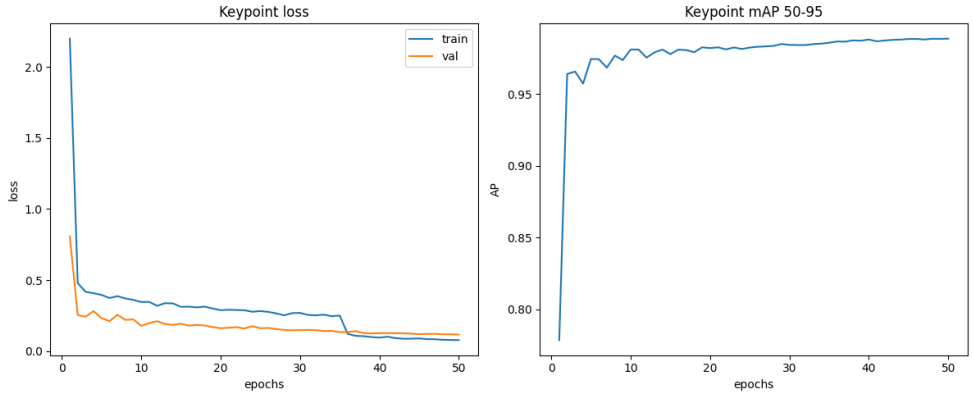


Figure 4.3: Last step of curriculum training with $\sigma = 0.025$ (640^2).

4.4 reveal that the Nvidia Jetson Orin NX is capable of running the optimized TensorRT engine at very high frequencies. However, there is a significant gap between the model with an input size of 448^2 and the one with 224^2. The choice of which model to use for online gate pose estimation depends on the specific race setting. Considering performance in terms of both accuracy and inference speed, the model using images at 224^2 likely strikes the best balance. Even though the camera captures images at 120 FPS, the ability to rapidly process this information for mapping or state estimation is critical for the drone's control module, which must make quick decisions. Moreover, the onboard computation unit must manage various processes and sensors, so a lighter model will have a smaller overall computational impact.

Image size	mAP
640^2	**98.86**
448^2	98.44
224^2	96.74
192^2	95.88

Table 4.1: mAP [.50:.95:.05] on the test/val set

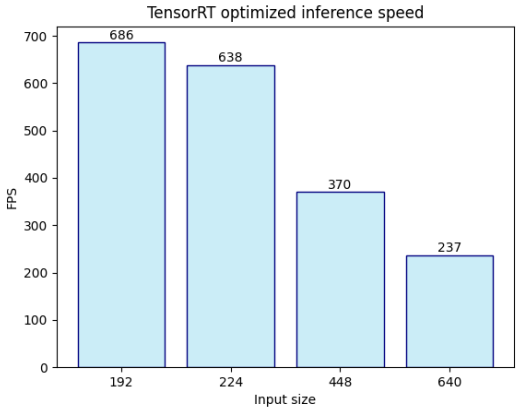


Figure 4.4: Inference speed-test with TensorRT optimized models, comparison at different image input size.

Image size	Label	Prediction
640^2	13.02 ± 5.48	**14.95** ± 9.45
448^2	13.02 ± 5.48	18.77 ± 6.53
224^2	13.02 ± 5.48	15.35 ± 6.35
192^2	13.02 ± 5.48	18.27 ± 8.11

Table 4.2: Mapping mean translation error (cm) - Flight 06 (autonomous)

Image size	Label	Prediction
640^2	28.05 ± 20.73	**27.77** ± 20.36
448^2	28.05 ± 20.73	30.00 ± 22.05
224^2	28.05 ± 20.73	32.44 ± 22.94
192^2	28.05 ± 20.73	35.10 ± 24.05

Table 4.3: Mapping mean translation error (cm) - Flight 09 (autonomous)

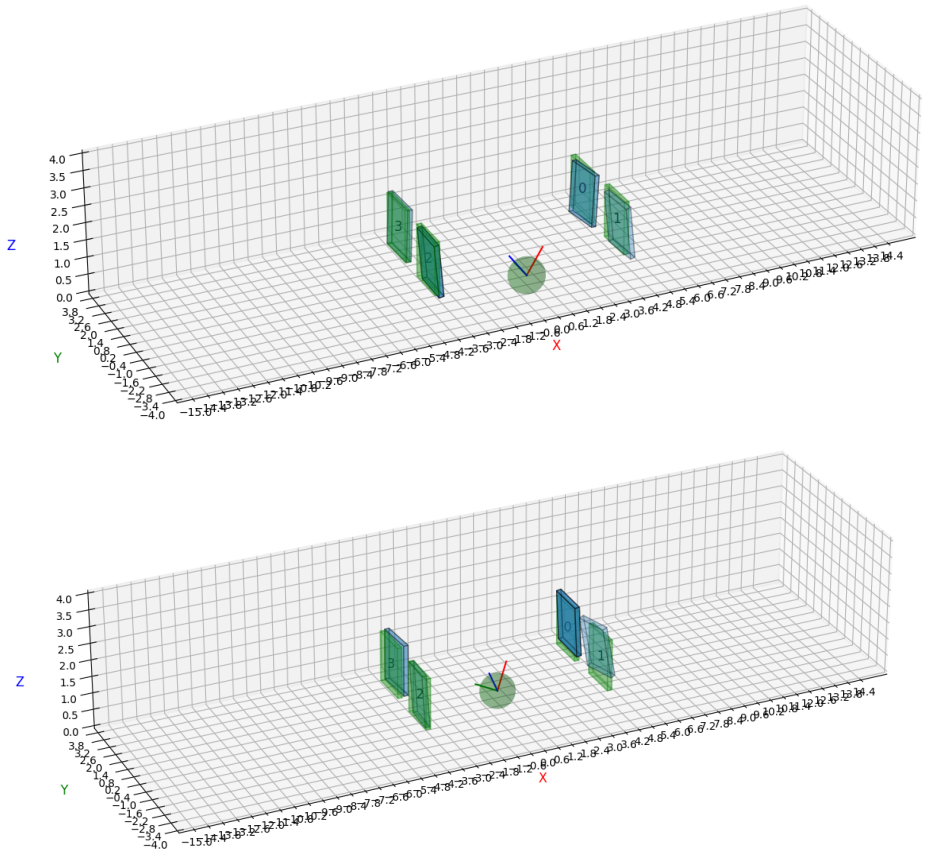


Figure 4.5: Results of mapping using ground truth keypoints labels for Flight 06 (top) and Flight 09 (bottom). In green gates' poses as from MoCap, in blue the estimated ones. In the middle the starting pose of the drone.

4.5 Model deployment

In the deployment phase of the research, efforts were made to leverage the computational capabilities of the NVIDIA Jetson Orin NX. This involved an initial step of converting the trained YOLO model to the **ONNX** (Open Neural Network Exchange) format. ONNX served as a bridge, facilitating interoperability between various deep learning frameworks and paving the way for seamless integration with **TensorRT**, NVIDIA's high-performance deep learning inference optimizer and runtime.

TensorRT played a pivotal role in achieving significantly faster inference times on the Jetson Orin NX. This optimization process is performed automatically and involves a range of techniques, including layer fusion, precision calibration, and dynamic tensor memory management, all of which geared towards maximizing the hardware capabilities of the target GPU. At the end of this process, there was the creation of a highly efficient TensorRT engine, custom-tailored for our specific YOLO model and the Jetson Orin NX. Specifically, the precision calibration step entails selecting between `FP32`, `FP16`, and `INT8`. The decision was made to utilize `FP16` to prevent a decrease in accuracy.

In order to run onboard and communicate with all the software stack of the drone, the code for inference and mapping had to be realized as ROS2 nodes. Particularly, a node (`YOLO_NODE`) was developed to intake input images from the camera and publishes a list of gate detections. For each gate, bounding box, confidence score, keypoints, and the relative pose of the gate with respect to the camera are specified. The relative pose is obtained using PnP and is represented by two `std::array float[3]` for translation and rotation. Thus, the second ROS2 node (`MAPPER`) takes in input the results of the `YOLO_NODE` and the state estimation provided by Qualysis, uses the mapping algorithm (2) to update the current belief of the track and publishes it as a list of gate poses in the world frame. The inference code, implemented in C++ utilizing the CUDA API, leveraged the produced engine, facilitating real-time keypoint detection

with impressive speed (Table 4.4), thus demonstrating the tangible benefits of TensorRT for edge computing applications.

4.6 Mapping

Being their shape and dimension well known, gates represents a key landmark in the context of drone races, indeed also human pilots rely on them to understand the track and have an estimate of their position. With the availability of camera intrinsic matrix, distortion coefficient, size of gates and a model that is able to detect gates' corners, all the necessary elements are in place to estimate the position and orientation of gates in the camera frame through PnP.

Perpsective-n-Point (PnP) is a robust and widely used algorithm in computer vision that problem of estimating the 3D pose (position and orientation) of an object or a set of points in space, relative to the camera that observes them. The core idea behind PnP is to establish correspondences between 2D image points (the keypoints detected by YOLO) and their corresponding 3D points in the real world. These 3D points may be defined in a local or global coordinate system, depending on the application. The PnP algorithm works by solving a system of equations derived from the geometric relationship between the known camera intrinsics, the 2D image points, and the 3D world points. It employs techniques such as the Direct Linear Transform (DLT) or more robust nonlinear solvers like Levenberg-Marquardt to minimize the error between the projected 3D points and the detected 2D keypoints.

While PnP is a powerful and widely used tool in computer vision for estimating the 3D pose of objects or points in space, it worth to note its limitations and potential sources of noise that can affect its precision. Noises can indeed arise from different sources:

- Inaccurate camera intrinsic parameter matrix and distortion coefficients due to imprecision in the camera calibration process. These parameters are one of the fundamental inputs of the PnP algorithm, the entire set of

equations indeed is built on top of them.

- Wrong assumptions and approximations of the 3D world coordinates. Gates are assumed to be perfectly squared objects of 1524 cm with 4 co-planar inner corners, in reality the shape of gates can vary, especially because they are not made of rigid materials.

- Inaccuracies in the detected gate corner coordinates. Even relying on human labels, the given pixel coordinate will never be perfect since light condition, low resolution and motion blur make it impossible to determine the exact location of the corner.

Considering all of these challenges, it becomes clear why even when using ground-truth labels, we observe a baseline error in Tables 4.2 and 4.3.

To evaluate our model's real-world performance, an algorithm was developed for mapping the racing track (Algorithm 1). This algorithm leverages the current drone state estimate, including position and orientation, to project the gates' pose estimates relative to the camera back into the world reference frame. The world pose of the detected gate (`gate_world`) is then matched with the closest gate, up to a given threshold, in the current map estimate. In the event of a match, the new detection contributes to updating the pose of the matched gate.

Before processing a new image, a refinement step is applied. In this step, a check is performed, ensuring that no pair of gates in the map is too close. In such case, the two identified gates are merged using a weighted average based on their number of detections (`hits`).

In the end, the `map` array will contain the estimated poses of gates. Additionally, with the `map_hits`, one can filter out gates that fall below a certain threshold of detections. As mentioned in the previous section (4.5), a ROS2 node in C++ was implemented to execute the mapping algorithm. However, due to time constraints, its online performance was not thoroughly tested.

Algorithm 1 Mapping algorithm

map ← [] ▷ Stores pose matrices of detected gates
map_hits ← [] ▷ # detections per mapped gate
$\alpha \leftarrow 0.7$ ▷ Update coefficient
thr ← 2.0 ▷ Match threshold (m)
for image in flight_images **do**
    map, map_hits ← map_refinement(map, map_hits, thr)
    camera_world ← state_estimate() ▷ Camera pose in world system
    for detection in YOLO(image) **do**
        kpts ← detection$[5:]$ ▷ Discard bounding box
        if num_visible_points(kpts) is 4 **then**
            gate_camera ← pose_estimation(kpts) ▷ Result of PnP
            gate_world ← gate_camera×camera_world
            match ← closest_gate(map, gate_world, thr)
            if match **then**
                map[match] ← α∗map[match]+$(1-\alpha)$∗gate_world
                map_hits[match]++
            else ▷ No match or empty map
                map.append(gate_world)
                map_hits.append(1)
            end if
        end if
    end for
end for

Algorithm 2 Refinement of the map

Require: $N \geq 2$
Ensure: map : too close gates are merged
function map_refinement(map[N], map_hits[N], thr)
    for $i \leftarrow 1$ to $N-2$ **do**
        match ← closest_gate(map[$i+1$:], map[i], thr)
        if match **then**
            match ← match+$i+1$
            total_hits ← map_hits[i] + map_hits[match]
            $w_i \leftarrow$ map_hits[i] / total_hits
            $w_m \leftarrow$ map_hits[match] / total_hits
            map[i] ← w_i∗map[i] +w_m∗map[match]
            map_hits[i] ← map_hits[i] + map_hits[match]
            map.remove(match)
            map_hits.remove(match)
        end if
    end for
    return map, map_hits
end function

Chapter 5

Conclusion and Future works

Throughout this master thesis, the primary focus has been on the development of perception systems for autonomous racing drones. The objective was to tackle the challenges associated with operating drones with exceptional precision and minimal reaction times in the context of autonomous racing.

To address these challenges, a high-speed dataset was recorded in an indoor arena using a custom quadrotor drone. This dataset encompasses flight trajectories, sensor data and labeling information. The process of labeling required and extensive work: tools for 3D rendering were used to produce synthetic images, a two-stage keypoint detection model was first pretrained on the generated images and then finetuned of a few manually labelled ones. The entire process for labeling automatization was important to drastically reduce the effort required to manually label hundreds of thousand of images. The creation of such a comprehensive dataset is a significant achievement as it provides a valuable resource for both training and evaluating perception algorithms specifically tailored for high-speed racing scenarios, and for developing classic or reinforcement learning based control systems.

For this purpose, one of the key contributions of this thesis lies in the development of a keypoints detection model using the YOLOv8 architecture. This model was trained on the high-speed dataset to detect racing gates, using

a method that can be seen as a form of *"Curriculum learning"*. The performance of the model was evaluated from different perspectives: canonical metrics (mAP) were used to find the best training recepy, translation error in the map reconstruction was used to verify the effective capability of such model, demonstrating promising results in accurately detecting and localizing gates in real-time scenarios. Consequently, a mapping algorithm was developed that utilizes the outputs of the keypoints detection model to reconstruct the map of an unknown racing track. By leveraging the gate detections, this algorithm estimates the pose of the racing gates, enabling the creation of a comprehensive map of the track. The accurate mapping of the track is essential for autonomous racing drones as it provides them with a detailed understanding of the environment, allowing them to plan optimal trajectories and make informed decisions during the race.

Furthermore, the successful deployment of the trained keypoints detection model on the Nvidia Jetson Orin NX, a powerful embedded platform, exemplifies the feasibility of running the perception system on resource-constrained hardware. This achievement is of paramount importance for real-world applications, where computational resources are often limited.

While this thesis has made significant strides in the field of perception systems for autonomous racing drones, several exciting avenues for future research remain unexplored. One potential direction is to delve into advanced deep learning techniques to further enhance gate detection performance. This could involve exploring novel architectures, incorporating temporal information, or leveraging additional sensor modalities to improve the robustness and accuracy of the perception system. Furthermore, integrating perception systems with planning and control algorithms is another avenue for future exploration. While this thesis focused primarily on perception, the ultimate goal of autonomous racing drones is to navigate the race track efficiently and safely. By integrating perception with planning and control algorithms, it is possible to develop end-to-end autonomous racing systems that can make real-time

decisions based on the perception of the environment.

In conclusion, this master thesis has made substantial contributions to the development of perception systems for autonomous racing drones. The creation of a high-speed dataset, the development of a keypoints detection model, and the successful deployment on resource-constrained hardware have paved the way for further advancements in autonomous racing technology. These achievements serve as a solid foundation for future research endeavors in the exciting intersection of AI, computer vision, and robotics, with the potential to revolutionize not only the world of racing but also various other domains that rely on autonomous systems.

Bibliography

[1] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman. The blackbird dataset: a large-scale dataset for uav perception in aggressive flight. In J. Xiao, T. Kröger, and O. Khatib, editors, *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 130–139, Cham. Springer International Publishing, 2020. isbn: 978-3-030-33950-0.

[2] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza. NeuroBEM: hybrid aerodynamic quadrotor model. In *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, July 2021. doi: 10.15607/rss.2021.xvii.042.

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, Montreal, Quebec, Canada. Association for Computing Machinery, 2009. isbn: 9781605585161. doi: 10.1145/1553374.1553380. url: https://doi.org/10.1145/1553374.1553380.

[4] Betaflight. The betaflight open source flight controller firmware project. https://github.com/betaflight/betaflight.

[5] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016. doi: 10.1177/0278364915620033.

[6] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*:1302–1310, 2016. url: https://api.semanticscholar.org/CorpusID:16224674.

[7] J. Cartucho, S. Tukra, Y. Li, D. S. Elson, and S. Giannarou. Visionblender: a tool to efficiently generate computer vision datasets for robotic surgery. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*:1–8, 2020.

[8] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin. Mmdetection: open mmlab detection toolbox and benchmark, 2019. arXiv: 1906.07155 [cs.CV].

[9] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2022. url: http://www.blender.org.

[10] M. Contributors. Openmmlab pose estimation toolbox and benchmark. https://github.com/open-mmlab/mmpose, 2020.

[11] M. Contributors. MMYOLO: OpenMMLab YOLO series toolbox and benchmark. https://github.com/open-mmlab/mmyolo, 2022.

[12] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza. Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6713–6719, 2019. doi: 10.1109/ICRA.2019.8793887.

[13] C. de Wagter, F. Paredes-Vallés, N. Sheth, and G. C. de Croon. The artificial intelligence behind the winning entry to the 2019 ai robotic racing competition. *ArXiv*, abs/2109.14985, 2021. url: https://api.semanticscholar.org/CorpusID:238227128.

[14] C. Feng, Y. Zhong, Y. Gao, M. R. Scott, and W. Huang. Tood: task-aligned one-stage object detection, 2021. arXiv: 2108.07755 [cs.CV].

[15] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza. Alphapilot: autonomous drone racing. *Autonomous Robots*, 46:307–320, 2020. url: https://api.semanticscholar.org/CorpusID:218889286.

[16] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, and D. Scaramuzza. Agilicious: open-source and open-hardware agile quadrotor for vision-based flight. *Science Robotics*, 7(67):eabl6259, 2022. doi: 10.1126/scirobotics.abl6259. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abl6259.

[17] D. Hanover, A. Loquercio, L. Bauersfeld, A. Romero, R. Pěnika, Y. Song, G. Cioffi, E. Kaufmann, and D. Scaramuzza. Autonomous drone racing: a survey. *ArXiv*, abs/2301.01755, 2023. url: https://api.semanticscholar.org/CorpusID:255440725.

[18] T. Jiang, P. Lu, L. Zhang, N. Ma, R. Han, C. Lyu, Y. Li, and K. Chen. Rtmpose: real-time multi-person pose estimation based on mmpose, 2023. arXiv: 2303.07399 [cs.CV].

[19] G. Jocher. *YoloV8*. Ultralytics. 2023. url: https://github.com/ultralytics/ultralytics.

[20] S. Jung, S. Hwang, H. Shin, and D. H. Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 3(3):2539–2544, 2018. doi: 10.1109/LRA.2018.2808368.

[21] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Beauty and the beast: optimal methods meet learning for drone racing. *2019 International Conference on Robotics and*

Automation (ICRA):690–696, 2018. url: https://api.semanticscholar.org/CorpusID:53114276.

[22] S. Li, M. M. O. I. Ozo, C. de Wagter, and G. C. de Croon. Autonomous drone race: a computationally efficient vision-based navigation and control strategy. *Robotics Auton. Syst.*, 133:103621, 2018. url: https://api.semanticscholar.org/CorpusID:52284283.

[23] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick. Coco - keypoint evaluation. 2014. url: https://cocodataset.org/#keypoints-eval.

[24] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. arXiv: 1405.0312. url: http://arxiv.org/abs/1405.0312.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg. Ssd: single shot multibox detector. In *European Conference on Computer Vision*, 2015. url: https://api.semanticscholar.org/CorpusID:2141740.

[26] C. Lyu, W. Zhang, H. Huang, Y. Zhou, Y. Wang, Y. Liu, S. Zhang, and K. Chen. Rtmdet: an empirical study of designing real-time object detectors, 2022. arXiv: 2212.07784 [cs.CV].

[27] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot operating system 2: design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074. url: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[28] A. L. Majdik, C. Till, and D. Scaramuzza. The Zurich urban micro aerial vehicle dataset. *The International Journal of Robotics Research*,

36(3):269–273, 2017. doi: 10.1177/0278364917702237. eprint: https://doi.org/10.1177/0278364917702237.

[29] D. Maji, S. Nagori, M. Mathew, and D. Poddar. Yolo-pose: enhancing yolo for multi person pose estimation using object keypoint similarity loss. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*:2636–2645, 2022. url: https://api.semanticscholar.org/CorpusID:248177719.

[30] T. Morales, A. Sarabakha, and E. Kayacan. Image generation for efficient neural network training in autonomous drone racing. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. doi: 10.1109/IJCNN48605.2020.9206943.

[31] C. Pfeiffer and D. Scaramuzza. Human-piloted drone racing: visual processing and control. *IEEE Robotics and Automation Letters*, 6(2):3467–3474, 2021. doi: 10.1109/LRA.2021.3064282.

[32] H. X. Pham, H. I. Ugurlu, J. L. Fevre, D. Bardakci, and E. Kayacan. Deep learning for vision-based navigation in autonomous drone racing. *Deep Learning for Robot Perception and Cognition*, 2022. url: https://api.semanticscholar.org/CorpusID:246777980.

[33] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*:779–788, 2015. url: https://api.semanticscholar.org/CorpusID:206594738.

[34] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015. url: https://api.semanticscholar.org/CorpusID:10328909.

[35] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: inverted residuals and linear bottlenecks. *2018 IEEE/CVF*

Conference on Computer Vision and Pattern Recognition:4510–4520, 2018. url: https://api.semanticscholar.org/CorpusID:4555207.

[36] A. Schmidt. A comparison of gate detection algorithms for autonomous racing drones. *2022 IEEE Aerospace Conference (AERO)*:1–13, 2022. url: https://api.semanticscholar.org/CorpusID:251472236.

[37] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018. doi: 10.1109/LRA.2018.2793349.

[38] J. R. Terven and D. M. C. Esparza. A comprehensive review of yolo: from yolov1 to yolov8 and beyond. *ArXiv*, abs/2304.00501, 2023. url: https://api.semanticscholar.org/CorpusID:257913644.

[39] D. B. West. Introduction to graph theory. In 1995. url: https://api.semanticscholar.org/CorpusID:118323960.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Giovanni Pau, and the Technology Innovation Institute for the opportunity to undertake this internship.

I'm grateful to the entire "Drone racing team" for their support, collaboration and friendship in these months of hard work.