Delft University of Technology

# Visual Navigation and Optimal Control for Autonomous Drone Racing

Li, S.

**DOI**
10.4233/uuid:07641fdf-5c87-4417-ad7c-e4232bd49570

**Publication date**
2020

**Document Version**
Final published version

**Citation (APA)**
Li, S. (2020). *Visual Navigation and Optimal Control for Autonomous Drone Racing*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:07641fdf-5c87-4417-ad7c-e4232bd49570

**Important note**
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

**Copyright**
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

**Takedown policy**
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

*This work is downloaded from Delft University of Technology.*
*For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.*

# Visual Navigation and Optimal Control for Autonomous Drone Racing

Shuo Li

# VISUAL NAVIGATION AND OPTIMAL CONTROL FOR AUTONOMOUS DRONE RACING

# VISUAL NAVIGATION AND OPTIMAL CONTROL FOR AUTONOMOUS DRONE RACING

## Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus Prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates,
to be defended publicly on Thursday 12 November 2020 at 12:30 o'clock

by

**Shuo LI**

Master of Engineering in Navigation, Guidance and Control,
Northwestern Polytechnical University, China
born in Shaanxi, China.

This dissertation has been approved by the promotors.

| | |
|---|---|
| Composition of the doctoral committee: | |
| Rector Magnificus | chairperson |
| Prof. dr. G. C. H. E. de Croon | Delft University of Technology, promotor |
| Dr. ir. C. C. de Visser | Delft University of Technology, copromotor |
| | |
| Independent members: | |
| Dr. J. Martinez-carranza | INAOE, Mexico |
| Prof. D. H. Shim | KAIST, South Korea |
| Prof. dr. P. Campoy | U. Politechnica de Madrid, Spain |
| Dr. J. C. van Gemert | Delft University of Technology |
| Prof. dr. ir. T. Keviczky | Delft University of Technology |
| | |
| Reserve member: | |
| Prof. dr. ir. M. Mulder | Delft University of Technology |

*Keywords:* Autonomous drone racing, visual navigation, nonlinear model-predictive control

*Printed by:* Ipskamp

*Front & Back:* Yuwen Liu.

Copyright © 2020 by S.Li

ISBN 978-94-6384-175-7

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

*This book is for my beloved parents*
*who are my powerful and secure backing...*

# CONTENTS

Summary 1

Samenvatting 3

1 Introduction 5
- 1.1 Background . . . . . 5
- 1.2 Challenges and Previous research . . . . . 8
  - 1.2.1 Visual navigation and sensor fusion . . . . . 8
  - 1.2.2 Guidance and control . . . . . 10
- 1.3 Research questions . . . . . 11
- 1.4 Outline . . . . . 13
- References . . . . . 14

2 Autonomous drone race: A computationally efficient vision-based navigation and control strategy 21
- 2.1 Introduction . . . . . 22
- 2.2 System overview . . . . . 23
- 2.3 Vision navigation . . . . . 25
  - 2.3.1 Gate detection . . . . . 25
  - 2.3.2 Pose estimation . . . . . 30
  - 2.3.3 Vision-IMU state estimation . . . . . 36
- 2.4 Control strategy . . . . . 40
- 2.5 Full track experiment setup and result . . . . . 45
- 2.6 Conclusion and future work. . . . . . 47
- References . . . . . 48

3 In-flight Model Parameter and State Estimation using Gradient Descent for High-speed Flight 51
- 3.1 Introduction . . . . . 52
- 3.2 Related work . . . . . 53
- 3.3 Quadrotor model . . . . . 54
  - 3.3.1 Dynamic model of quadrotor . . . . . 54
  - 3.3.2 IMU misalignment. . . . . . 56
  - 3.3.3 Aerodynamic model . . . . . 56
  - 3.3.4 AHRS bias model. . . . . . 57
  - 3.3.5 Full model . . . . . 59
- 3.4 State estimation. . . . . . 59
  - 3.4.1 Vision-based Extended Kalman Filter . . . . . 59
  - 3.4.2 Vision-based gradient descent method . . . . . 61

3.5 Experiment setup and result . . . 63
3.5.1 Experiment setup . . . 63
3.5.2 Analysis of Vision-based gradient descent method (VGD) . . . 65
3.5.3 Comparison between Vision-based EKF (VEKF), Vision-based gradient descent method (VGD) and Vision-based gradient descent method with kinematic model(VGD-kinematic) . . . 67
3.6 Conclusion . . . 69
References . . . 71

**4 Visual Model-predictive Localization for Computationally Efficient Autonomous Racing of a 72-gram Drone 75**
4.1 Introduction . . . 76
4.2 Problem Formulation and System Description . . . 79
4.2.1 Problem Formulation . . . 79
4.2.2 System Overview . . . 79
4.3 Robust Visual Model-predictive Localization (VML) and Control . . . 81
4.3.1 Gate assignment . . . 82
4.3.2 Visual Model-predictive Localization (VML) . . . 85
4.3.3 Flight Plan and High Level Control . . . 92
4.4 Simulation Experiments . . . 93
4.4.1 Simulation Setup . . . 93
4.4.2 Simulation result and analysis . . . 94
4.5 Real-world Experiments . . . 100
4.5.1 Processing time of each component . . . 100
4.5.2 Flying experiment without gate displacement . . . 102
4.5.3 Flying experiment with gate displacement . . . 103
4.5.4 Flying experiment with different altitude and moving gate . . . 104
4.6 Discussion . . . 106
4.7 Conclusion . . . 107
References . . . 107

**5 Aggressive Online Control of a Quadrotor via Deep Network Representations of Optimality Principles 115**
5.1 INTRODUCTION . . . 116
5.2 DESIGN OF THE G&CNET . . . 117
5.2.1 The dynamical system . . . 117
5.2.2 The optimisation problem . . . 118
5.2.3 Network architecture and training . . . 118
5.3 SIMULATION RESULT AND ANALYSIS . . . 119
5.3.1 Stability of Neural Network Controller . . . 119
5.3.2 Differential flatness based aggressive trajectory generation and control (DiffG&C) . . . 120
5.3.3 Simulation of the G&CNet Controller . . . 121
5.3.4 Comparison between DiffG&C and G&CNet . . . 121

5.4 EXPERIMENT SETUP AND RESULT . . . . 123
5.4.1 Experiment Setup . . . . 123
5.4.2 Experiment Result . . . . 124
5.5 CONCLUSIONS . . . . 126
References . . . . 126

**6 Conclusion** **129**
6.1 Research questions . . . . 129
6.2 Discussion . . . . 131
6.3 Future work . . . . 132

**Acknowledgements** **135**

**Curriculum Vitæ** **139**

**List of Publications** **141**

# SUMMARY

Drones, especially quadrotors, have shown their great value for applications like aerial photography, object delivery and warehouse inspection. At the same time, with the development of Artificial Intelligence (AI), computers can replace humans and even perform better than humans in some areas where it was impossible before like the AI program Alpha Go which beat the human world champion in Go matches and Alpha star which was rated above 99.8% human players in the real-time strategy game StarCraft II. Concerning drones, the question is whether they can fly races completely by themselves and if they can fly even faster than human pilots' racing drones?

Although there exist many technologies for drones to fly autonomously in terms of navigation, guidance and control, autonomous drone racing still sets an enormous challenge for the robotics community. For example, the most commonly used vision camera based navigation technologies such as Simultaneous Localization and Mapping (SLAM) and Visual Inertial Odometry (VIO) suffer motion blur when the drone moves fast and high computational demand which is scarce onboard the drone. Moreover, the commonly used PID controller has no guarantees of optimality while much parameter tuning is required. Many other challenges like these require new technologies to satisfy more complex and challenging flying scenarios to challenge humans in drone races.

This thesis attempts to answer the question mentioned above. First of all, this thesis presents 2 systematic solutions for autonomous drone racing including navigation, guidance and control techniques. The solutions are computationally so efficient that they can run on board of a Bebop 1 quadrotor (made in 2014) without using the GPU and a cheap 72-gram quadrotor called the 'Trashcan'. With the constraints of the processing power and cheap onboard sensors, the Bebop can fly through 15 gates in a complex scenario with an average speed of $1.5m/s$ and the Trashcan can fly through a 4-gate racing track for 3 laps with an average speed of $2m/s$. Both solutions helped the MAVLab, TU Delft, participate in the IROS autonomous drone racing in 2017 and 2018.

In terms of visual navigation, a computationally efficient gate detection method 'snake gate' is developed to detect the racing gate during the flight. Together with a revised version of Perspective-3-Point (P3P) method, the detection results are used to provide location information for the drone. A Kalman filter is developed to fuse these detections with the onboard IMU readings. Unlike the traditional Kalman filter, this version deduces the velocity from the accelerometers readings by a linear drag model approximation instead of integrating the accelerometers. In this way, the Kalman filter has a faster convergence rate. Another filtering method, Visual Model-predictive Localization (VML), is also developed to fuse the vision detections and onboard attitude estimation. The simulation and real-world flight results show that the VML is more robust to outliers than the commonly used Kalman filter especially when there are invalid measurements. Also, the VML is more efficient than the Kalman filter in handling measurement delays. At last, a gradient descent based parameter estimation method is developed to estimate

the quadrotor's aerodynamics coefficients and the Attitude and heading reference system (AHRS) biases using the visual measurements and the onboard state predictions. With the estimated parameters, the quadrotor can have a better state prediction when no visual measurement is available in some time.

In terms of guidance and control, a novel neural network based nonlinear optimal controller, G&CNet, is developed to steer the drone to the target with the minimum time. This G&CNet moves the time-consuming nonlinear controller onboard and can be run at 200HZ to map the current states and the optimal control policy calculated offboard. The simulation results show that the flying result is very close to the theoretical nonlinear optimal control solution. Both simulation and real-world flying results show that it has faster flights than a commonly used polynomial based trajectory generation and tracking method.

Last but not least, the methods provided can be generalized to other applications. For example, for the outdoor flight where the Global Positioning System (GPS) is available for navigation, the vision measurements can be directly replaced by the GPS signals in the proposed navigation strategies and they should work directly. For the proposed G&CNet, it should work in all scenarios where the guidance and control modules are needed to move the drone from one point to another point. In this way, the proposed methods allow drones to move faster in a robust way, extending their mission capabilities.

# SAMENVATTING

Drones – en in het bijzonder drones met vier propellors - vormen een grote belofte voor toepassingen zoals luchtfotografie, het afleveren van pakketjes, en het in kaart brengen van de voorraad in magazijnen. Tegelijkertijd vinden er stormachtige ontwikkelingen plaats op het gebied van kunstmatige intelligentie (KI), waarbij computers even goed of beter aan het worden zijn als mensen, ook als het gaat om taken die voorheen als zeer complex geacht werden voor de computer. Voorbeelden zijn het bordspel "Go", waar het computerprogramma AlphaGo de wereldkampioen heeft verslagen, en het computerspel "StarCraft II", waar het computerprogramma Alpha Star sterker genoteerd staat dan 99.8

Alhoewel er veel technieken ontwikkeld zijn voor drones om autonoom te vliegen, en zowel hun eigen toestand te schatten, hun pad te bepalen en dat pad uit te voeren met besturingssystemen, representeert drone racen een enorme uitdaging voor de roboticagemeenschap. Bijvoorbeeld, veelgebruikte technieken zoals visuele inertiële odometrie (Visual Inertial Odometry, VIO, in het Engels) en gelijktijdig localiseren en in kaart brengen (Simultaneous Localization And Mapping, SLAM) hebben veel last van de waas die ontstaat als een drone sneller gaat vliegen. Bovendien vereisen beide technieken veel rekenkracht die nou juist schaars is op kleine drones. Voor wat betreft de besturing zijn veelgebruikte technieken zoals een Proportionele, Integrerende en Differentiërende (PID) besturing niet zo geschikt; Ze geven geen garantie van een optimaliteit en vereisen veel moeite om af te stellen. Vele soortgelijke uitdagingen resulteren erin dat we voor drone racen nieuwe technologieën moeten ontwikkelen, vooral als we naar meer en meer complexe racescenario's gaan.

Dit proefschrift probeert bovenstaande vragen te beantwoorden. Ten eerste presenteer ik twee systematische oplossingen voor autonoom drone racen, die alle elementen bevatten van toestandschatting tot het bepalen en uitvoeren van een pad met besturingstechnieken. De oplossingen zijn zo efficiënt in termen van rekenkracht, dat ze kunnen werken aan boord van een Bebop 1 quadrotor (gemaakt in 2014), zonder de grafische computer processor (GPU) te gebruiken, en aan boord van een goedkope drone die "Trashcan" heet en slechts 72 gram weegt. Ondanks de beperkingen aangaande rekenkracht, energie, en de lage kwaliteit van de goedkope sensoren, kan de Bebop door 15 poortjes vliegen in een complex scenario met een gemiddelde snelheid van 1.5 meter / seconde. De TrashCan kan drie rondjes vliegen door een circuit bestaande uit vier poortjes met een gemiddelde snelheid van 2 meter / seconde. Beide oplossingen hebben aan de basis gestaan van de deelname van het MAVLab van de TU Delft aan de IROS drone races in 2017 en 2018.

Voor wat betreft visuele navigatie, heb ik een computationeel efficiënt algoritme, "snake gate", ontworpen om poortjes te detecteren tijdens de vlucht. Samen met een aangepaste versie van P3P leidt dit algoritme tot poortdetecties die op hun beurt locatie-informatie verschaffen aan de drone. Een Kalman filter is ontwikkeld om de detecties te combineren met metingen van de inertiële meeteenheid (Inertial Measurement Unit, IMU). In tegenstelling tot traditionele Kalman filters in dit gebied, worden de versnellingsmetingen niet direct geïntegreerd om snelheden en posities te schatten, maar dienen ze om de luchtweerstand te schatten (met een lineair model). Zo convergeert het Kalman filter sneller. Ik heb ook een andere filtermethode ontwikkeld, waarin visuele perceptie gecombineerd wordt met een model, en "Visual Model-predictive Localization" (VML) geheten. Simulate en echte vluchtresultaten laten zien dat VML robuuster is tegen foute metingen dan een standaard Kalman filter. Ook gaat VML efficiënter om met het modelleren van metingsvertragingen. Tenslotte is er een methode ontwikkeld die parameters van de drone schat, zoals de luchtweerstandconstanten en de constante afwijkingen van de versnellingsmeters en toestandschatting. De methode gebruikt de gradiënt om schattingsfouten te minimaliseren. Het schatten van deze parameters helpt om de toestand van de drone (zoals de snelheid) beter te schatten wanneer er geen poortjes in zicht zijn.

Aangaande besturingssystemen is er een nieuwe optimale besturing ontwikkeld gebaseerd op neurale netwerken. De methode, genaamd "GCNet" is ontwikkeld om de quadrotor – onderhevig aan nonlineaire dynamica – in een minimale tijd naar een doelpositie toe te sturen. Waar optimale besturing doorgaans veel rekenkracht en rekentijd vereist, is dit niet het geval voor GCNet: Dit kan op 200Hz gedraaid worden op de Bebop 1 drone, om de huidige toestand om te zetten in de optimale stuurcommando's. Het neuraal netwerk representeert een optimale besturing, die verkregen is door voor de vlucht uitgebreid te trainen in een computersimulator. De simulatieresultaten laten zien dat de tijd van het traject van de drone het theoretisch optimale resultaat zeer goed benaderen. Zowel simulatieresultaten als vluchten in de echte wereld laten zien dat de aanpak resulteert in snellere vluchten dan met een veelvuldig gebruikte methode die polynomen gebruikt voor het genereren en volgen van een snel pad.

Tenslotte is het belangrijk aan te geven dat de ontwikkelde methoden ook toepasbaar zijn in een andere context. Zo kunnen voor buitenvluchten de metingen van navigatiesatellieten gebruikt worden (zoals het Global Positioning System, GPS). De voorgestelde filters en besturingsmethoden hoeven dan niet aangepast te worden: de metingen die in de drone race van beeldherkenning komen, kunnen namelijk zonder problemen verplaatst worden door de snelheids- en positiemetingen van GPS. Ook GCNet kan zonder moeite op andere problemen en systemen worden toegepast, zo lang de drone van één locatie naar een andere moet bewegen. Op deze manier draagt het proefschrift bij aan het sneller bewegen en robuuster toestandschatten van drones. Dit breidt de mogelijkheden van drones nog verder uit.

# 1

# INTRODUCTION

## 1.1. BACKGROUND

In recent years, quadrotors have gradually shown their value for applications such as obtaining agriculture images [1, 2], picking up and delivering objects [3, 4], power line inspection [5] and warehouse inspection [6]. Commercial companies like DJI [7] and Parrot [8] launched their customer products, which are smart and user-friendly to users who have no background in controlling quadrotors. Since then, quadrotors were no longer expensive tools in specific industry areas or flying robots only in laboratories. They entered people's lives as aerial photography tools and electronic toys.

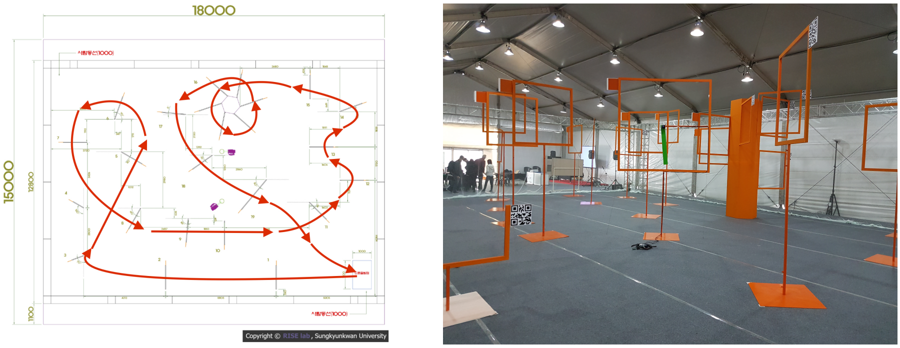
Before the quadrotors entered people's lives, there was a minority group of enthusiasts who developed the quadrotors to powerful, agile and sturdy flying platforms (Figure 1.1). Due to the quadrotor's agile nature, these drones could perform high-speed flights, aggressive turns and freestyle flips under the control of the pilots. Subsequently, competitions between these enthusiasts appeared. The rules of drone racing are simple: the pilots steer their drones to fly through windows, gates or avoid obstacles in a certain sequence and the fastest wins. The pilots wear First-person View (FPV) goggles to receive the images transmitted from the onboard camera via radio waves and steer the drone accordingly. In most cases, in order to fly the drone aggressively with as little delay as possible, the pilots directly give thrust, roll, pitch and yaw rate commands to the drone via transmitters. The onboard autopilot will allocate the revolution to each rotor according to the received commands. In fact, controlling the quadrotor in this way is not straightforward for humans because when the sticks are in neutral, the drone keeps its current attitude instead of going back to level. Thus, to fly a racing drone properly, a lot of training is required. That is also why commercial companies like DJI provide their customers with high-level interfaces by which the users can directly control the speed of the drone or even the position of the drone, which in turn, requires more sensors, high-performance onboard computing and also loses the flexibility to perform aggressive maneuvers. With the development of drone racing, more and more organized races show up and become annual e-sport events such as the one held by Drone Racing League (DRL).

Human pilots spend time mainly on training their flying skills, designing the hardware and tuning the parameters to make their flight faster. Initially, these races were seen as a new popular e-sport but they did not attract much attention of academia.

Figure 1.1: Racing drone, by Richard Bramlette via flickr.com [9]

With the advent of advanced computing abilities and Artificial Intelligence (AI), people started to realize that computers can replace humans and even perform better in areas where it was not possible before, such as Alpha Go [10] which beat the human world champion in Go matches and Alpha Star [11]which was rated above 99.8% human players in the real-time strategy game StarCraft II. The games in which humans have been beaten by AI programs were all closed, simulated environments. The next frontier consists of challenges in robotics, dealing with open, real-world settings. In fact, some years ago the question arose whether drones can fly races completely by themselves? And can they fly even faster than human pilots? These questions piqued roboticists' interest. If we look back at how human pilots steer the drone through the racing track, we can find that they use their eyes to sense the environment, the position, velocity and attitude of the racing drones which is called navigation in the context of control theory. During the flight, the pilots keep replanning the trajectories that guide the drone through the gates or avoid the obstacles, which is called guidance. At last, the pilots steer the sticks on the transmitters to execute the planned trajectories which is called control. These three parts together make the control loop closed and were studied for many years both in academia and industry. Hence, robotics scientists thought it was possible to challenge human pilots who can achieve speeds up to $190km/h$, although there were many challenges to the existing technologies.

To answer the first question mentioned above, the first autonomous drone race was held by the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) in 2016, South Korea [12]. (Figure 1.2) The rules were the same as for human drone racing except for that humans could not interfere with the flight, no external equipment was allowed and all computations had to be done onboard. It was the first attempt of autonomous drone racing and no participant finished the whole track. The first-place team, USRG, KAIST and the second-place team MAVLab, TU Delft both passed 10 gates. But the winner, KAIST, had a faster speed of $0.6m/s$ [13]. This slow speed, partly due to the tortuous track and partly due to the still lacking technologies and algorithms, is far lower than for human racing drones. However, from the positive side, this autonomous racing drone has demonstrated the possibility of racing autonomously and pointed out a new research direction of flying drone race tracks autonomously using only onboard resources and even faster than human pilots.



(a) The layout of IROS 2016 autonomous drone race track

(b) Picture of IROS 2016 autonomous drone race track

Figure 1.2: The layout and a picture of IROS 2016 autonomous drone race track

From this milestone, more research related to autonomous drone racing appeared and the IROS autonomous drone race has become an annual event for research groups to compete and communicate their latest research results. Most of them were not robust enough, computationally expensive or had low flying speed. One of the fastest examples was created by NASA's JPL lab who reported their autonomous drone racing solution obtaining speeds very close to human pilots in their lab [14]. In their research, a visual-inertial localization and mapping system was used for navigation and an aggressive trajectory connecting waypoints was generated to finish the track [15]. Gao et al. reported a teach-and-repeat method for autonomous drone race [16]. In their approach, a global map was built and a time-optimal trajectory was designed a priori. The drone then tracked the designed trajectories with the state estimation provided by VIO. For both systems, although the speed is high, when the environments are changed, such as gate displacement, the systems have to be manually set again. Other studies have some adaptability to the gate displacement, but the speed is much slower. For example, the winner of IROS 2017 autonomous drone race, INAOE, flew through 9 gates with the speed of $0.7m/s$ which is slightly faster than the previous year's winner [17]. They used SLAM for navigation and the detections of the gates were used to correct the drift. Jung et al. developed a convolution neural network (CNN) to detect the racing gates which is more robust than the classic color-filter based approach [18]. Although no exact flying speed was reported, from the released flying experiment video, it can be seen that the drone still had a low speed. Loquercio et al. [19] finished the track with the speed of $3m/s$. Instead of using CNN to detect the racing gate, they used CNN to map onboard images to desired waypoints and speed. A visual-inertial odometry was used for state estimation in their apporoach. In IROS 2018 Autonomous drone race, the winner, RPG lab from UZH, pushed the boundary of the autonomous drone race's average speed to

$2m/s$ [20]. They used a CNN to predict the position of the gate and based on the prediction, a model predictive controller (MPC) was employed to steer the drone through the gate with fast speed. Although their approach was faster and adaptive to the gate displacement, the algorithms had to be run on high-performance and relatively heavy platforms such as Intel UpBoard and Qualcomm Snapdragon Flight platform. As one of the few early research groups focusing on autonomous drone race, the MAVLab, TU Delft also developed an approach addressing robustness, flying speed and computational efficiency. My thesis work consists of contributions to this approach, as further laid out in this thesis. For example, a commercial Bebop 1 quadrotor flew through 15 gates with a speed of $1.5m/s$ in a complex environment. Also, we created a 72g quadrotor able to fly the race track with an average speed of $2m/s$ and the maximum speed of $2.6m/s$.

## 1.2. CHALLENGES AND PREVIOUS RESEARCH

Similar to flying a drone autonomously in other scenarios, navigation, guidance and control are all essential to autonomous racing drones. But there are three challenges to this scenario. Firstly, in the indoor environment, the racing drone needs to navigate without a Global Positioning System (GPS) signal while detecting the racing gates is also required. Secondly, all the computation has to be done onboard while the weight of the racing drone should be limited to have fast and safe flights, which in turn, requires efficient navigation algorithms with light-weight sensors. Thirdly, to achieve fast flying speed, optimal guidance methods, advanced control techniques and accurate quadrotor aerodynamics models are needed. This section will discuss the previous work on these three aspects.

### 1.2.1. VISUAL NAVIGATION AND SENSOR FUSION

Navigation in the context of control is defined by the establishment of the current and future states including position, velocity and attitude, etc. In the outdoor environment, GPS is the most commonly used navigation equipment for drones [21–24]. However, most indoor environments and many parts of the urban canyon remain without access to GPS [25]. Thus, motion capture systems such as VICON and Opti-track systems, which can provide position measurements with the precision of millimeters, are often used to replace GPS in the indoor environment [26–29]. However, the high price, occlusion of the obstacles in the flying area and complex installation/calibration procedure limit the use of such systems. An alternative solution is to use ultra-wideband (UWB) range measurements to replace the motion capture systems [30–32]. This approach is cheap and has the precision of centimeters but still only gives positioning in a limited, known area. For many applications, it is unacceptable to rely on the installation of such external equipment.

In terms of navigation using only onboard sensors, simultaneous localization and mapping (SLAM) and visual inertial odometry (VIO) are the most commonly used methods. In general, SLAM jointly estimates the feature positions and the camera/IMU pose that together form the state vector, whereas VIO does not include the features in the state but still utilizes the visual measurements to impose motion constraints between the camera/IMU poses [33]. In addidion, compared to SLAM, VIO does not perform loop closure. SLAM/VIO can be used with different sensor setups, for example, laser scanners. Bachrach et al. [34] used a laser rangefinder to map the environment and navigate the drone while the computations were done off-board. Shen et al. [35] implemented SLAM with a laser range finder using only onboard resources. Bry et al. [36] first collected a 3D map of the environment before the flight and the map was then used for the navigation of the drone. Although laser scanners can provide satisfying navigation results, their heavy weight, fragile components [37] and high price make them unsuitable for the drone race scenario. Alternative sensors that can be used for SLAM/VIO are cameras, including stereo cameras and monocular cameras. Fraundorfer et al. [38] used a stereo camera onboard to build a map and explore the unknown environment. Weiss et al. [39] used a single camera to build a 3D map of the environment and navigate the drone. Although SLAM/VIO with onboard cameras have been widely used for indoor navigation, they are still not ideal for autonomous drone racing because of standard cameras' relative long latency, blur when performing aggressive maneuver and most importantly, high demand of onboard computational resources, which requires a heavier computational unit and subsequently limits the speed of the racing drone. Event cameras are an alternative sensor to classic visual navigation, which output brightness changes in the form of a stream of asynchronous “events" instead of intensity frames[40]. They have a very high dynamic range, do not suffer from motion blur, and provide measurements with a latency as low as one microsecond [41]. At the same time, due to its different sensing technique with conventional cameras, traditional image processing methods can not be directly used. They also have more significant noise compared to conventional cameras [42]. With these advantages and challenges, event camera based SLAM/VIO has been studied by multiple research groups and is still active [43–46].

SLAM/VIO is designed to work in general environments with the cost of heavy computational demand but this generality is not always necessary. In some situations, detecting specific features can significantly decrease the computational burden for visual navigation. Also, detecting targets for navigation is required in some applications such as autonomous landing on a known landing pad. Li et al. [47] used a downward-looking monocular camera to detect two LED markers to navigate a quadrotor to land on a moving ground robot. Eberli et al. [48] also used a downward-looking camera to detect a circular blob for taking off. Wenzel et al. [49] used an infrared (IR) camera to track a T-shaped 3D-pattern of infrared lights for navigation. Meier et al. [50] used a marker board full of unique markers whose positions are known prior for navigation. Although these methods require much less computational resources, the simple features of the markers and the static background make the detection less challenging. At the same time, downward-looking cameras with static makers can limit the flying range of the drone. Falanga et al. [51] used a forward-looking camera to detect a narrow gap and then the detections were used to navigate the drone through the gap with high speed. However, the narrow gap had a black-and-white rectangular pattern which made the detection easier.

In terms of sensor fusion on drones, in most cases, the position measurement from only one source, like onboard cameras or laser range finders, is not enough. They are either in low frequency or noisy. In worse cases, they have outliers which may lead to unstable behaviors of the drone. On the other hand, onboard IMUs have less noise and can measure some drone states in high frequency. However, they have biases that lead to state prediction divergence in a short time. Thus, sensor fusion or state estimation techniques are needed to take advantage of both sensors to provide high frequency, smooth and unbiased state estimation. It is indisputable that the Kalman filter and its variants are the most commonly used sensor fusion/state estimation method in the fields of robotics and aerospace. For nonlinear systems, the Extended Kalman filter (EKF) [52] is the standard technique for state estimation [53–55]. However, the EKF uses first-order linearization to approximate the nonlinear system which introduces large errors. Thus, the Unscented Kalman Filter (UKF) was developed to address this problem by using a deterministic sampling approach [56] and it is also widely used on quadrotors [57–60]. The UKF has, however, the limitation that it does not apply to general non-Gaussian distributions [61]. Another commonly used filter is the particle filter which increases estimating accuracy with a much higher computational expense. Due to the limited computation capacity, it is not commonly used onboard drones in real-time applications.

### 1.2.2. GUIDANCE AND CONTROL

As stated before, guidance and control are essential for quadrotors. Guidance is defined by generating reference trajectories on a high level. In this thesis, to make it more specific, we interpret the word *guidance* as generating the waypoints or position/velocity trajectory references that guide the quadrotor to the desired position. To some extent, reference trajectories can be seen as moving waypoints. Currently, real-time trajectory generation, the feasibility and the optimality of the trajectories are still the main topics of the research of guidance in robotics. Control is usually defined by generating actuators' commands to track the desired reference trajectories or arrive at the desired targets based on current states. These two terms are always discussed together. However, in traditional approaches, they are usually designed separately, that is, reference trajectories are first generated and then the controllers are designed to tracked the generated trajectories. In recent years, optimal control theory based methods, which can design both reference trajectories and control input signals simultaneously, gradually emerged due to the increase in computation power. Some of them have already shown their feasibility in the real world. However, there is still much space for improvement and it is still an active area in the robotics community.

The simplest guidance method is just a waypoint in 3D space followed by a pure feedback controller to steer the drone to the target [22, 28]. This way is straightforward and simple. In fact, it sends a step reference to the controller. In many cases, it can be competent for the task, especially for hover tasks or the tasks where the maneuver is not aggressive. However, it does not consider any dynamics property of the drones and transfers all the burden to the controller. If the controller is not well designed or tuned, the drone may experience actuator saturation, overshoot, oscillation and even loose stability. Furthermore, this strategy has no guarantee of the time/energy optimality. Other methods like generating trajectories by a low-pass filter or connecting points by polynomials can produce continuous and smooth trajectories. However, lacking dynamics models in the trajectory generation phase makes these methods have no essential difference with the 'waypoint' method. To track the reference trajectory, a multiple-loop cascaded structure is always used. For the attitude loop, a large number of controllers were developed. For example, linear controllers such as PID and LQR controllers [62, 63] have been proved to be efficient and stable by many real world flights. Nonlinear attitude controllers like the backstepping controller and sliding-mode controller were also developed to address the nonlinearity of the quadrotor [64]. Smeur et al. [23, 65] developed an adaptive incremental nonlinear dynamic inversion (INDI) controller to reduce the dependency on the drone's model. To track the desired trajectories, outer-loop controllers usually receive the difference between the desired position and current estimated position and output the desired attitude or acceleration signal to the inner-loop controller. Similar to the inner-loop controllers, PID [60], backstepping [66] and INDI [23] controllers, etc. are also commonly employed to track the reference.

Another commonly used guidance and control method for drones is the differential flatness based method which explores the drone's dynamics model while designing the trajectory [26]. Basically, the quadrotor's inputs can be written as nonlinear functions of the position trajectories, heading trajectories and their derivatives. In other words, if a drone's trajectories are determined, its inputs (four rotors' speed for quadrotors) to execute these trajectories can be calculated analytically. Thanks to this property, during the trajectory generation phase, the inputs of the drones can be minimized and guaranteed to be within the constraints. At the same time, time can be allocated for each segmentation on the trajectories to adjust the aggressiveness of the flight. Mellinger et al. [26] and Bry et al. [36] designed the trajectories based on this theory, which can guide the drone to fly through a thrown circular hoop and a complex indoor environment aggressively. Faessler et al. [67] took the quadrotor's aerodynamics model into consideration to design the trajectories and also used the calculated inputs as feed-forward terms to the controller. Then, a feedback controller was added to compensate for model inaccuracy. It turned out that their method could track the aggressive reference trajectories with higher accuracy. Although the differential flatness based method can guide the drone to fly with high speed, it still cannot guarantee time optimality. In other words, the drones are still not flying at their limits.

In terms of optimality (minimum time or minimum energy), the essential way to solve this guidance and control problem is to use optimal control theory, which considers the drone's model and outputs the optimal trajectories and inputs at the same time. However, finding a theoretical optimal control solution of a non-linear system is infeasible and finding a suboptimal solution numerically requires a considerable amount of time [68, 69] and outputs a feed-forward solution, which makes it unsuitable for real-time implementations. There are two directions to apply this optimal control theory onboard. The first one is to decouple and linearize the system, which can result in analytical optimal solutions [70, 71] but can lead to an inaccurate representation of the system, which may lead to a suboptimal solution. The other direction is to use the suboptimal solution directly. For example in Geisert's work [72], they used the result of the first iteration of the optimization, which is suboptimal but fast to compute. It is noticeable that both ways sacrifice the optimality to reduce the calculation time.

## 1.3. RESEARCH QUESTIONS

Before the start of this work, there was no research on autonomous drone racing. Although a large body of research has focused on navigation, guidance and control of the quadrotor, they still have their limitations for autonomous drone races which require robust and stable systems to fly fast in complex environments using only onboard resources. This leads to the first research question:

**RESEARCH QUESTION 1**

How does a drone fly through a racing track fully autonomously using only onboard resources?

During the flight, the drone may lose position measurements for a short time. A common approach is to use pure state prediction to provide state estimation. However, due to the inaccuracy of the drone's model, the prediction will diverge quickly. To keep the drone flying during the measurement loss, it is important to estimate an accurate aerodynamic model and the sensors' error model to minimize the divergence. Nevertheless, different drones have different aerodynamic models and one sensor may behave differently at different time within a day and even during the flight. Modeling the aerodynamics and the sensors before take off is inconvenient and unrealistic especially in the competition arena. Hence, the second research question can be raised:

**RESEARCH QUESTION 2**

How well can the drone estimate its aerodynamics parameters together with its AHRS biases during flight with the help of the vision information?

In complex racing environments, vision measurements are not always reliable. Vision measurements are usually in low frequency compared to the onboard IMU measurements, which can lead to slow convergence of the Kalman filter. Detection outliers are also not a small probability event especially in the racing track where gates are similar. A classic Kalman filter does not have the mechanism to handle these outliers properly which can lead to the states "jumping" to the outliers. Mahalanobis distance is always used as Kalman filter's outlier rejection index. It works well in most cases. However, when the estimated states' variances are large and an outlier appears, for example, after a long time prediction, the Kalman filter has a high chance to jump to the outlier. The worst case is that the Kalman filter will reject the subsequent true positive detections and then diverge. Also, vision algorithms always have significant delays. To have better state estimation results, compensating for the delay is also necessary. The common delay compensation method used by Kalman filter requires substantial computational resources which are scarce on the drone. This result in the third research question:

RESEARCH QUESTION 3

How does a moving horizon based method compare to a Kalman filter in the drone race setting?

If the above questions are answered, we should push the boundary one step further that is to increase the racing drones' speed since the final target of this research is to demonstrate the technology that the drone can not only finish the track autonomously but also with fast speed and even faster than human pilots. This then culminates in the last research question:

RESEARCH QUESTION 4

What are the properties of a neural-network-based imitation of a (close-to) time-optimal control policy?

## 1.4. OUTLINE

To answer the questions raised above, this thesis has 4 chapters to address each question respectively and 1 chapter to conclude them. They are organized as follows.

In Chapter 2, we present a full system strategy of flying a Bebop 1 commercial drone through the racing track autonomously with an average speed of $1.5m/s$. In this chapter, we will first discuss a novel gate detection method 'Snake gate detection' for detecting the racing gates. Next, a Kalman filter is introduced to fuse the detection results with an onboard IMU to provide state estimation. Next, two control strategies used in the race will be discussed. At last, an experiment in which the drone flew through 15 gates in a complex environment will be presented. In fact, this work is one of the earliest research in the autonomous drone race field. It was the lightest and one of the fastest autonomous racing drone at that time.

Chapter 3 will discuss a gradient-descent based method which estimates the quadrotor's aerodynamic parameters and onboard AHRS biases during the flight. We first discuss the reason for the AHRS biases and introduce its model. Then, a gradient descent method is used to minimize the error between the predicted states and the observations. At last, the estimated result is used to compensate for the subsequent pure state prediction. The result shows that the proposed method can significantly increase the accuracy of the prediction. This method can be used in the autonomous drone race to increase the state prediction's accuracy when the gate detections are not available.

In Chapter 4, we present another systematic strategy of the autonomous drone race including a new localization method called visual model-predictive localization (VML) and the control strategy. The proposed VML is proved to be more robust to the outliers than a commonly used Kalman filter by simulation. It is also capable of handling vision delays with high efficiency. At last, we also show the real-world flight of a $72g$ tiny drone which flew through the racing track with an average speed of $2m/s$ and the maximum speed of $2.6m/s$. It is the smallest and one of the fastest autonomous racing drones in the world. The proposed VML method helped the MAVLab, TU Delft win the first prize (1 million dollars) in the first Alpha Pilot autonomous drone race in 2019.

Chapter 5 addresses the onboard optimal control problem of the quadrotor. As it was stated above, classic optimal control methods are too heavy to run onboard. To solve this problem, we designed an optimal trajectory library consisting of hundreds of thousands of trajectories and corresponding control policies with different initial states. Then, a neural network was trained to map the states and the optimal control policy. The trained neural network could run onboard to calculate the optimal control policy to steer the drone to the target. In this way, we moved the nonlinear optimal controller onboard with high update frequency. The simulation results show that the resulting flying trajectories were very close to the theoretical optimal result. The real-world flying trajectories were also very close to the simulated optimal trajectories.

In Chapter 6, we conclude our work that it is possible the drones can race by themselves in complex racing environments with high speed with the proposed navigation and control methods. However, the speed of the autonomous racing drone is still slower than human pilots. At the end, we point out the future possible directions for further speed improvement.

## REFERENCES

[1] A. Barrientos, J. Colorado, J. d. Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente, *Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots,* Journal of Field Robotics **28**, 667 (2011).

[2] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss, *Uav-based crop and weed classification for smart farming,* in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017) pp. 3024–3031.

[3] D. Mellinger, M. Shomin, N. Michael, and V. Kumar, *Cooperative grasping and transport using multiple quadrotors,* in *Distributed autonomous robotic systems* (Springer, 2013) pp. 545–558.

[4] S. Kim, S. Choi, and H. J. Kim, *Aerial manipulation using a quadrotor with a two dof robotic arm,* in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2013) pp. 4990–4995.

[5] G. Zhou, J. Yuan, I.-L. Yen, and F. Bastani, *Robust real-time uav based power line detection and tracking,* in *2016 IEEE International Conference on Image Processing (ICIP)* (IEEE, 2016) pp. 744–748.

[6] A. Eudes, J. Marzat, M. Sanfourche, J. Moras, and S. Bertrand, *Autonomous and safe inspection of an industrial warehouse by a multi-rotor mav,* in *Field and Service Robotics* (Springer, 2018) pp. 221–235.

[7] A. Schroder, M. Renker, U. Aulenbacher, A. Murk, U. Boniger, R. Oechslin, and P. Wellig, *Numerical and experimental radar cross section analysis of the quadrocopter dji phantom 2,* in *2015 IEEE Radar Conference* (IEEE, 2015) pp. 463–468.

[8] P.-J. Bristeau, F. Callou, D. Vissiere, and N. Petit, *The navigation and control technology inside the ar. drone micro uav,* IFAC Proceedings Volumes **44**, 1477 (2011).

[9] R. Bramlette, *Dsc_0352r,* (2016), online; accessed December 14, 2019.

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, *Mastering the game of go with deep neural networks and tree search,* nature **529**, 484 (2016).

[11] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, *Grandmaster level in starcraft ii using multi-agent reinforcement learning,* Nature **575**, 350 (2019).

[12] H. Moon, Y. Sun, J. Baltes, and S. J. Kim, *The IROS 2016 Competitions [Competitions],* IEEE Robotics & Automation Magazine **24**, 20 (2017).

[13] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, *A direct visual servoing-based framework for the 2016 iros autonomous drone racing challenge,* Journal of Field Robotics **35**, 146 (2018).

[14] A. Good, *Drone race: Human versus artificial intelligence,* `https://www.nasa.gov/feature/jpl/drone-race-human-versus-artificial-intelligence` (2017).

[15] B. Morrell, M. Rigter, G. Merewether, R. Reid, R. Thakker, T. Tzanetos, V. Rajur, and G. Chamitoff, *Differential flatness transformations for aggressive quadrotor flight,* in *Robotics and Automation (ICRA), 2018 IEEE International Conference on Robotics and Automation* (IEEE, 2018) pp. 5204–5210.

[16] F. Gao, L. Wang, K. Wang, W. Wu, B. Zhou, L. Han, and S. Shen, *Optimal trajectory generation for quadrotor teach-and-repeat,* IEEE Robotics and Automation Letters (2019).

[17] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, *et al.*, *Challenges and implemented technologies used in autonomous drone racing,* Intelligent Service Robotics , 1 (2019).

[18] 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**, 2539 (2018).

[19] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, *Deep drone racing: Learning agile flight in dynamic environments,* arXiv preprint arXiv:1806.08548 (2018).

[20] 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,* in *2019 International Conference on Robotics and Automation (ICRA)* (IEEE, 2019) pp. 690–696.

[21] C.-S. Y. C.-S. Yoo and I.-K. A. I.-K. Ahn, *Low cost gps/ins sensor fusion system for uav navigation,* in *Digital Avionics Systems Conference, 2003. DASC'03. The 22nd,* Vol. 2 (IEEE, 2003) pp. 8–A.

[22] C. De Wagter, R. Ruijsink, E. J. Smeur, K. G. van Hecke, F. van Tienen, E. van der Horst, and B. D. Remes, *Design, control, and visual navigation of the delftacopter vtol tail-sitter uav,* Journal of Field Robotics **35**, 937 (2018).

[23] E. J. Smeur, M. Bronz, and G. C. de Croon, *Incremental control and guidance of hybrid aircraft applied to a tailsitter unmanned air vehicle,* Journal of Guidance, Control, and Dynamics , 1 (2019).

[24] J. L. Crassidis, *Sigma-point kalman filtering for integrated gps and inertial navigation,* IEEE Transactions on Aerospace and Electronic Systems **42**, 750 (2006).

[25] A. Bachrach, S. Prentice, R. He, and N. Roy, *Range–robust autonomous navigation in gps-denied environments,* Journal of Field Robotics **28**, 644 (2011).

[26] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors,* in *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (IEEE, 2011) pp. 2520–2525.

[27] D. Mellinger, N. Michael, and V. Kumar, *Trajectory generation and control for precise aggressive maneuvers with quadrotors,* The International Journal of Robotics Research **31**, 664 (2012).

[28] E. J. Smeur, G. C. de Croon, and Q. Chu, *Cascaded incremental nonlinear dynamic inversion for mav disturbance rejection,* Control Engineering Practice **73**, 79 (2018).

[29] G. Tang, W. Sun, and K. Hauser, *Learning trajectories for real-time optimal control of quadrotors,* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018) pp. 3620–3625.

[30] M. W. Mueller, M. Hamer, and R. D'Andrea, *Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation,* in *2015 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2015) pp. 1730–1736.

[31] D. Hoeller, A. Ledergerber, M. Hamer, and R. D'Andrea, *Augmenting ultra-wideband localization with computer vision for accurate flight,* IFAC-PapersOnLine **50**, 12734 (2017).

[32] A. K. Raja and Z. Pang, *High accuracy indoor localization for robot-based fine-grain inspection of smart buildings,* in *Proceedings of the IEEE International Conference on Industrial Technology,* Vol. 2016-May (2016) pp. 2010–2015.

[33] G. Huang, *Visual-inertial navigation: A concise review,* in *2019 International Conference on Robotics and Automation (ICRA)* (IEEE, 2019) pp. 9572–9582.

[34] A. Bachrach, R. He, and N. Roy, *Autonomous flight in unknown indoor environments,* International Journal of Micro Air Vehicles **1**, 217 (2009).

[35] S. Shen, N. Michael, and V. Kumar, *Autonomous multi-floor indoor navigation with a computationally constrained mav,* in *2011 IEEE International Conference on Robotics and Automation* (IEEE, 2011) pp. 20–25.

[36] A. Bry, C. Richter, A. Bachrach, and N. Roy, *Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments,* The International Journal of Robotics Research **34**, 969 (2015).

[37] S. Hrabar, *An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance,* Journal of Field Robotics **29**, 215 (2012), arXiv:10.1.1.91.5767 .

[38] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, *Vision-based autonomous mapping and exploration using a quadrotor mav,* in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2012) pp. 4557–4564.

[39] S. Weiss, D. Scaramuzza, and R. Siegwart, *Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments,* Journal of Field Robotics **28**, 854 (2011).

[40] D. Gehrig, M. Gehrig, J. Hidalgo-Carrió, and D. Scaramuzza, *Video to events: Bringing modern computer vision closer to event cameras,* arXiv preprint arXiv:1912.03095 (2019).

[41] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, *High speed and high dynamic range video with an event camera,* IEEE Transactions on Pattern Analysis and Machine Intelligence (2019).

[42] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, *et al.*, *Event-based vision: A survey,* arXiv preprint arXiv:1904.08405 (2019).

[43] D. Weikersdorfer, R. Hoffmann, and J. Conradt, *Simultaneous localization and mapping for event-based vision systems,* in *International Conference on Computer Vision Systems* (Springer, 2013) pp. 133–142.

[44] H. Rebecq, T. Horstschäfer, G. Gallego, and D. Scaramuzza, *Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time,* IEEE Robotics and Automation Letters **2**, 593 (2016).

[45] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, *The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam,* The International Journal of Robotics Research **36**, 142 (2017).

[46] H. Kim, S. Leutenegger, and A. J. Davison, *Real-time 3d reconstruction and 6-dof tracking with an event camera,* in *European Conference on Computer Vision* (Springer, 2016) pp. 349–364.

[47] W. Li, T. Zhang, and K. Kühnlenz, *A vision-guided autonomous quadrotor in an air-ground multi-robot system,* in *2011 IEEE International Conference on Robotics and Automation* (IEEE, 2011) pp. 2980–2985.

[48] D. Eberli, D. Scaramuzza, S. Weiss, and R. Siegwart, *Vision based position control for mavs using one single circular landmark,* Journal of Intelligent & Robotic Systems **61**, 495 (2011).

[49] K. E. Wenzel, A. Masselli, and A. Zell, *Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle,* Journal of intelligent & robotic systems **61**, 221 (2011).

[50] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, *Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision,* Autonomous Robots **33**, 21 (2012).

[51] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, *Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision,* in *Robotics and Automation (ICRA), 2017 IEEE International Conference on* (IEEE, 2017) pp. 5774–5781.

[52] B. Anderson and J. B. Moore, *Optimal filtering,* (1979).

[53] S. Weiss, M. W. Achtelik, M. Chli, and R. Siegwart, *Versatile distributed pose estimation and sensor self-calibration for an autonomous mav,* in *2012 IEEE International Conference on Robotics and Automation* (IEEE, 2012) pp. 31–38.

[54] A. Santamaria-Navarro, G. Loianno, J. Solà, V. Kumar, and J. Andrade-Cetto, *Autonomous navigation of micro aerial vehicles using high-rate and low-cost sensors,* Autonomous robots , 1 (2018).

[55] K. D. Sebesta and N. Boizot, *A real-time adaptive high-gain ekf, applied to a quadcopter inertial navigation system,* IEEE Transactions on Industrial Electronics **61**, 495 (2014).

[56] E. A. Wan and R. Van Der Merwe, *The unscented kalman filter for nonlinear estimation,* in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)* (Ieee, 2000) pp. 153–158.

[57] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, *Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu,* IEEE Robotics and Automation Letters **2**, 404 (2017).

[58] R. He, R. He, S. Prentice, S. Prentice, N. Roy, and N. Roy, *Planning in information space for a quadrotor helicopter in a GPS-denied environments,* in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on,* 2007 (2008) pp. 1814–1820.

[59] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, *Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor.* in *Robotics: Science and Systems*, Vol. 1 (Citeseer, 2013).

[60] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, *et al.*, *Fast, autonomous flight in gps-denied and cluttered environments,* Journal of Field Robotics **35**, 101 (2018).

[61] R. Van Der Merwe, A. Doucet, N. De Freitas, and E. A. Wan, *The unscented particle filter,* in *Advances in neural information processing systems* (2001) pp. 584–590.

[62] S. Bouabdallah, A. Noth, and R. Siegwart, *Pid vs lq control techniques applied to an indoor micro quadrotor,* in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, Vol. 3 (IEEE, 2004) pp. 2451–2456.

[63] S. Tijmons, M. Karásek, and G. de Croon, *Attitude control system for a lightweight flapping wing mav,* Bioinspiration & biomimetics **13**, 056004 (2018).

[64] S. Bouabdallah and R. Siegwart, *Backstepping and sliding-mode techniques applied to an indoor micro quadrotor,* in *Proceedings of the 2005 IEEE international conference on robotics and automation* (IEEE, 2005) pp. 2247–2252.

[65] E. J. Smeur, Q. Chu, and G. C. de Croon, *Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles,* Journal of Guidance, Control, and Dynamics (2015).

[66] P. Adigbli, J.-b. Mouret, and S. Doncieux, *Nonlinear attitude and position control of a micro quadrotor using sliding mode and backstepping techniques,* (2007).

[67] M. Faessler, A. Franchi, and D. Scaramuzza, *Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories,* IEEE Robotics and Automation Letters **3**, 620 (2017).

[68] F. Morbidi, R. Cano, and D. Lara, *Minimum-energy path generation for a quadrotor uav,* in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2016) pp. 1492–1498.

[69] M. Hehn, R. Ritz, and R. D'Andrea, *Performance benchmarking of quadrotor systems using time-optimal control,* Autonomous Robots **33**, 69 (2012).

[70] M. Hehn and R. D'Andrea, *Quadrocopter trajectory generation and control,* IFAC proceedings Volumes **44**, 1485 (2011).

[71] O. Santos, H. Romero, S. Salazar, O. García-Pérez, and R. Lozano, *Optimized discrete control law for quadrotor stabilization: Experimental results,* Journal of Intelligent & Robotic Systems **84**, 67 (2016).

[72] M. Geisert and N. Mansard, *Trajectory generation for quadrotor based systems using numerical optimal control,* in *2016 IEEE international conference on robotics and automation (ICRA)* (IEEE, 2016) pp. 2958–2964.

# 2

# AUTONOMOUS DRONE RACE: A COMPUTATIONALLY EFFICIENT VISION-BASED NAVIGATION AND CONTROL STRATEGY

---

This chapter is based on the following article:
**Li S**, Ozo MM, De Wagter C, de Croon GC. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. Robotics and Autonomous Systems. 2020 Nov 1;133:103621.

## 2.1. INTRODUCTION

First person view (FPV) drone racing has been a popular sport in recent years, where the pilots have to control the drones to fly through gates decorated by LED lights at fast speed. In the field of robotics, drone racing has raised the question: how can drones be designed to fly races by themselves, possibly faster than human pilots? To start answering this question, the world's first autonomous drone race was held in 2016 [1]. In this drone race, the drones were asked to fly through square, orange gates in a predefined sequence using onboard resources. To increase the level of challenge for gate detection, unlit gates were used in the race. The rules were simple: the one who flies furthest will win the race, and if two drones arrive at the same gate or complete the full track, the fastest time counts. The winner of the 2016 race (the team from KAIST) flew through 10 gates (the distance is around 50$m$) within 86s [2] and the winner of the 2017 race (the team from INAOE) flew through 9 gates (60$m$) within 194s, which are much slower than the FPV drone race players. Compared to the FPV drone race, the task of autonomous drone race is more challenging because the drone has to navigate, perceive, plan and control all by itself using only scarce onboard resources, representing a considerable challenge for areas such as artificial intelligence and control.

Autonomous drone racing can be seen in the more general context of high-speed flight. In fact, before the autonomous drone race, there are several on flying through circles or gaps. To the best of our knowledge, the first research on quadrotor's flying through circles is [3]. In their work, the drone can fly through a thrown circle and three fixed circles with fast speed. In [4], the drone can fly through a tilted narrow gap. In both studies, a VICON motion capture system is used to provide the state estimation for the drone and the position of the gap or circles is known a priori. Lyu et al. [5] use an onboard camera to detect the gap and the drone could navigate itself through the gate. But the image processing is done off-board. In their experiment, the background of the gap is a white wall which makes the gap to be detected relatively easily. Loianno et al. [6] for the first time use onboard resources to detect a window, plan the trajectory and control the drone to fly through a window. In their work, visual inertial odometry (VIO), which is computationally quite expensive for our drone, is used to provide the state estimation to the drone. In Falanga et al.'s [7] work, a drone with a fish-eye camera can detect a black and white gap and design a trajectory through the gap using only onboard resources. In [8], deep-learning-based optical flow is used to find any arbitrary shaped gap with an NVIDIA Jetson TX2 GPU. But the drone has to execute a fixed sideways translational motion to detect the gap before going through it, which slows down the drone. The studies above aim at motion planning, object detection or onboard perception, so in most of these studies only one gap is flown through and there is no solution on how to fly through the next gate after passing through the previous one.

Multiple studies have focused directly on autonomous drone racing, designing a strategy that will allow to fly an entire trajectory. In [9], a simulated drone learns how to minimize the time spent to finish the race track, by learning from two different PID controllers. Although an interesting approach, it ignores several of the real-world aspects of drone racing, such as restricted onboard computation or how to deal with accelerometer biases. NASA's Jet Propulsion Laboratory has developed an autonomous racing drone controlled by AI, which can fly almost as fast as the racing drones controlled by expert human FPV pilots.[10, 11] They use VIO for navigation which is computationally relatively expensive. Kaufmann et al. develop a strategy that combines a convolutional neural network (CNN) and minimum jerk trajectory generation.[12] In their work, an in-house quadrotor with an Intel UpBoard and a Qualcomm Snapdragon Flight Kit which is used for VIO, is used as the platform. In [2], a systematic solution for the IROS autonomous drone race 2016 is presented. In their work, an NVIDIA Jetson TK1 single-board computer and a stereo camera are used for a visual servoing task. They finally passed through 10 gates within 86s and won the race. We will use their result as a benchmark to compare our research result.

In this paper, we present a solution for autonomous drone racing, which is computationally more efficient than the solutions discussed above. For the gate detection, a novel light-weight algorithm, “snake gate detection”, is described and analyzed in detail in Section 2.3. Instead of using a common, purely vision-based perspective-n-point (PnP) algorithm, we combine the onboard attitude estimate with the gate detection result to determine the position of the drone. We show that this is more robust than the PnP method. Then, a novel Kalman filter is introduced that uses a straightforward drag model to estimate the velocity of the drone. Two control strategies to control the drone to go through the gate and find the next gate are discussed in Section 2.4. In Section 2.5, flight tests are performed with a Parrot Bebop 1 drone, by replacing the Parrot firmware with our Paparazzi autopilot code. All algorithms run in real-time on the limited Parrot P7 dual-core CPU Cortex A9 processor, and no hardware changes are required as the vision algorithms use the frontal camera and other sensors already present in the Bebop. The flight experiments are done in a complex and narrow environment (a showroom displaying aircraft components in the basement of Aerospace Engineering, TU Delft).[1] The result shows that the drone can fly through a sequence of 15 gates autonomously using only onboard resources in a very complex environment with a velocity of up to $1.5m/s$.

## 2.2. SYSTEM OVERVIEW

The quadrotor hardware used as experiment platform in this work is a commercially available Parrot Bebop 1 (Figure 2.1). However, all Parrot software was replaced by own computer vision, own sensor drivers and own navigation and control using the Paparazzi-UAV open-source autopilot project [13]. Only the Linux operating system was kept. The most important characteristics are listed in Table 2.1. It should be noted that the image from the front camera as used by our autopilot in this work is only $160 \times 350$ pixels and all the processing for the drone race takes place on the Parrot P7 dual-core CPU Cortex 9 (max 2GHz), although the Bebop is equipped with a quad core GPU.

The structure of the system is shown in Figure 2.2. For visual navigation, a novel algorithm, snake gate detection, is implemented to detect the gates. It outputs the coordinates of detected gates’ corners, which are then sent to the pose estimation block. In pose estimation block, the coordinates of the gate corners on the image plane would be projected to 3D space, which provides the relevant position between the drone and the gate. For attitude and heading reference system (AHRS), a classic complementary filter [14] is employed. At last, the position measured by the front camera, attitude estimation

---

[1] The video of the experiment is available at: https://youtu.be/bwF0TAjC8iI

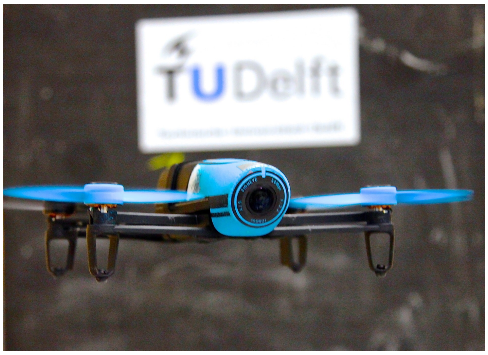


Figure 2.1: The Parrot Bebop 1 is used as experiment platform. The software is replaced by the Paparazzi UAV open-source autopilot project

Table 2.1: List of onboard sensors used in the experiment

| | |
|---|---|
| camera | a 6 optical elements and 14 Mega pixels sensor |
| | a vertical stabilization camera (not used in this work) |
| processor | Parrot P7 dual-core CPU cortex 9 (max 2GHz) |
| IMU | MPU 6050 |
| sonar | $< 8m$ |

from AHRS and IMU measurement are fused by a Kalman filter to provide a position estimate.

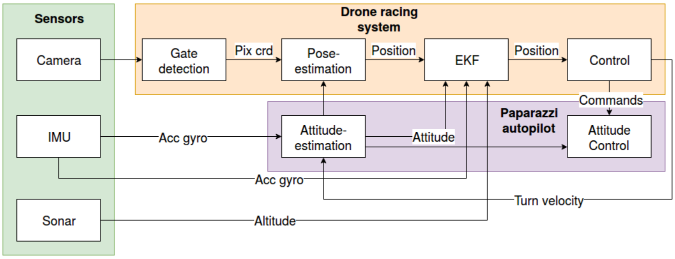


Figure 2.2: The structure of the autonomous system

In terms of control, when the target gate is in the field of view, a PD controller (Control block in Figure 2.2) is used to steer the drone to align with the center of the gate. After passing through the gate or there is no gate in the field of view, a prediction-based feedforward control scheme is employed to steer the drone to the next gate, which will be further explained in Section 2.4. An adaptive incremental nonlinear dynamic inversion (INDI) controller is used as low-level attitude controller [15].

The race track can be divided into two parts. The first part is the approaching gate part where the target gate can be used by the drone for navigation. The other one is after gate part, which starts from the point where the drone passing through the gate and ends at the point where the drone can see the next gate. The different race tracks can be seen as the different combination of these two parts. Thus, at first , due to the space restriction of our experimental environment, we simplify the race track to a two gates track which can be seen in Figure 2.3. Most of our experiments are done and analyzed in this simplified race track with the ground truth measurement provided by Opti-track. At last, the system is moved to a more complex and realistic drone race track to be verified.

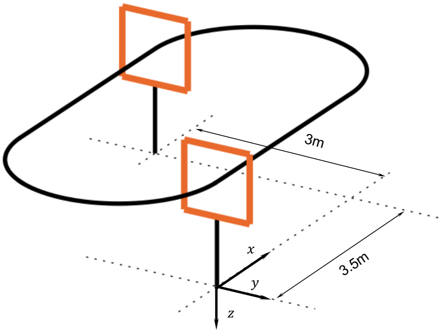


Figure 2.3: A simplified race track

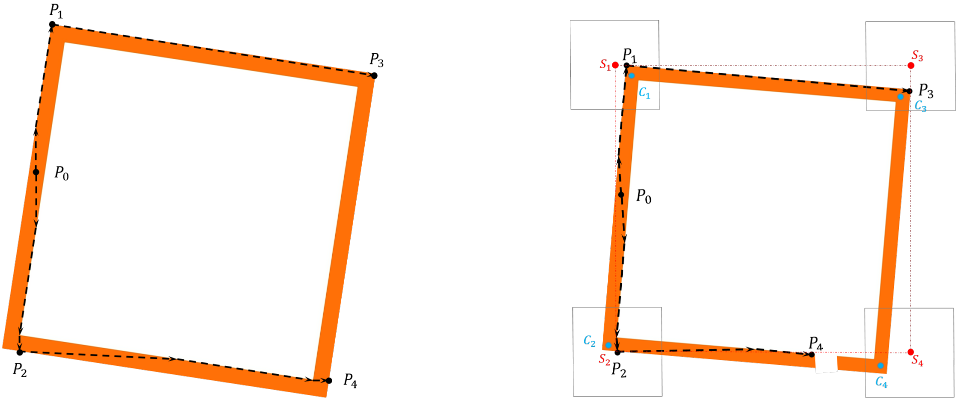
## 2.3. VISION NAVIGATION

In the FPV drone race, gates are usually decorated with LEDs in order to be easily recognized by drone pilots. Drone pilots can then use the gates to navigate themselves to approach the gates. Inspired by FPV drone race, in our research, we also use gates for navigation since their simple shape and relatively large size make them relatively easy to be extracted and their projection on the image plane can provide information such as position and attitude of the drone. In this section, we first present an efficient gate detection method to extract the four corners of the gate on the image plane. Next, the position of the four corners of the gate is projected to 3D space combining AHRS reading. At last, a Kalman filter providing position estimation by fusing the vision measurement, the IMU measurement and the onboard AHRS reading is discussed.

### 2.3.1. GATE DETECTION

Gate detection can be accomplished by multiple different computer vision methods, such as Viola and Jones[16], Hough transform[17] and deep learning[18, 19]. In this article, we propose a novel gate detection algorithm called snake gate detection which is lightweight and easy to be implemented onboard.

We search the gates based on their colors on an distorted image because the undistortion procedure for each image can slow down the whole detection procedure (Figure 2.4). Luckily, our detection method can still work properly on this distorted image. The search starts by randomly sampling [20] in the original image. If a random point $\mathbf{P}_0$ hits the target color (gate's color), we continue searching 'up and down' to find points $\mathbf{P}_1$ and $\mathbf{P}_2$. It should be noted that this search can search along the edge of the oblique bar of the gate (Figure 2.4). To prevent that the algorithm may find some small color blocks which have the same color as the gate, we introduce a threshold, which is called the minimum length threshold $\sigma_L$. If $\|\mathbf{P}_1 - \mathbf{P}_2\| < \sigma_L$, this search would be terminated. Then, we use $\mathbf{P}_1$ and $\mathbf{P}_2$ as start points respectively to search 'left and right' to find $\mathbf{P}_3$ and $\mathbf{P}_4$. Similar to the vertical search, the horizontal search can also search along the oblique bar and the result would be checked by $\sigma_L$ to ensure that the detection is not too small and hence unlikely to be a gate. The algorithm can be found in Algorithm 1. It should be noted that while small $\sigma_L$ may lead to acceptance of some small detections which in most cases are false positive detections, large $\sigma_L$ can lead to the result that some gate in the image are rejected. The selection of $\sigma_L$ will be discussed later in this section.



(a) If the gate is continuous on image plane, snake gate detection algorithm should find all four corners $P_1$, $P_2$, $P_3$ and $P_4$

(b) When the gate is not continuous on image plane, first a square $S_1$, $S_2$, $S_3$ , $S_4$ with minimum length including $P_1$, $P_2$, $P_3$ , $P_4$ is found. Four small squares centering at $S_i$ are then found. In these small squares, a histogram analysis helps to refine our estimate of the gate's corners in the image

Figure 2.4: An example of snake gate detection.

If the gate's image is continuous in the image plane and the gates' edges are smooth, snake gate detection should find all four points (Figure 2.4.(a)). However, due to varying light conditions, some parts of the gate may get overexposure or underexposed which may lead to color deviation. For example, in Figure 2.4.(b), part of the lower bar gets overexposed. In this case, $\mathbf{P}_4$ will not reach the real gate's corner. Hence, a refining process is employed to find the real gate's corner. To refine the detection, a square with minimum length including four points is firstly obtained (Red square in Figure 2.4). Then four small squares centering at $S_i$ are found (Four gray square in Figure 2.4.(b)). The raw detection is refined by finding the centroid of the patch around each rough corner.

In one image, in most cases, the number of detected gates $N_d$ is larger than the number of real gates in the image $N_g$. It can be caused by duplicated samples on the same gate, which are true positive detections and do not affect the performance of navigation. The other reason for $N_d > N_g$ is the false positive detections, which affects the accuracy of navigation significantly and should be eliminated. Here, another threshold, color fitness threshold $\sigma_{cf}$, is introduced to help decrease the number of false positive detection.

$$cf = \frac{N_c}{N} \tag{2.1}$$

where $N_c$ is the total number of pixels on the polygon whose color is target color and $N$ is the number of pixels on the polygon.

Only the gates whose $cf > \sigma_{cf}$ will be accepted as detected gates. Similar to minimum length threshold $\sigma_L$, the selection of $\sigma_{cf}$ also affects the detection accuracy significantly.

**Algorithm 1** snake gate detection

```
procedure SnakeGateDetection($image$)
    for i = 1:maxSample do
        $\mathbf{P}_0$ = randomPoint()
        if isTargetColor($\mathbf{P}_0$,$image$) then
            [$\mathbf{P}_1$,$\mathbf{P}_2$] = searchUpDown($\mathbf{P}_0$,$image$)
            if $\|\mathbf{P}_1 - \mathbf{P}_2\| > \sigma_L$ then
                $\mathbf{P}_3$ = searchLeftRight($\mathbf{P}_1$,$image$)
                $\mathbf{P}_4$ = searchLeftRight($\mathbf{P}_2$,$image$)
                if $\|\mathbf{P}_1 - \mathbf{P}_3\| > \sigma_L$ OR $\|\mathbf{P}_2 - \mathbf{P}_4\| > \sigma_L$ then
                    [$\mathbf{S}_1$,$\mathbf{S}_2$,$\mathbf{S}_3$,$\mathbf{S}_4$] = findMinimalSquare($\mathbf{P}_1$,$\mathbf{P}_2$,$\mathbf{P}_3$,$\mathbf{P}_4$)
                    detectedGate = refineCorner($\mathbf{S}_1$,$\mathbf{S}_2$,$\mathbf{S}_3$,$\mathbf{S}_4$)
                    if checkColorFitness(detectedGate) > $\sigma_{cf}$ then
                        return $detectedGate$
                    end if
                end if
            end if
        end if
    end for
end procedure
```

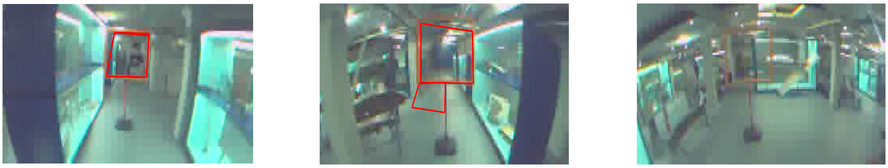
To evaluate the performance of the snake gate detection algorithm, 600 onboard images with/without gate are used to test the algorithm (Figure 2.5). The ROC curve with varying $\sigma_L$ is shown in Figure 2.6. It should be noted that the detection is done 10 times with one $\sigma_L$ to obtain the statistical result. The x-axis of ROC curve is average of false positive detection per image and the y-axis is true positive rate. To make the trend in Figure 2.6(a) clearer, we enlarge local part of the ROC curve by using logarithm coordinate system in Figure 2.6(b). From ROC curve, it can be seen that when $\sigma_L$ is small ($\sigma_L < 15$), the number of the false positive detections decreases significantly while $\sigma_L$ increases without sacrificing TPR.That is because $\sigma_L$ helps to reject the small detections caused by small color blocks of the environment. When $\sigma_L > 35$, however, TPR decreases sharply, the reason is that $\sigma_L$ is too large to accept true positive detections. $\sigma_L = 25$ can give the optimal option with low FPs/image and almost highest TPR. Then, with $\sigma_L = 25$ , we draw another ROC curve with varying $\sigma_{cf}$, which is shown in Figure 2.7. It can be

**Algorithm 2** search in vertical direction (search in horizontal direction is similar)

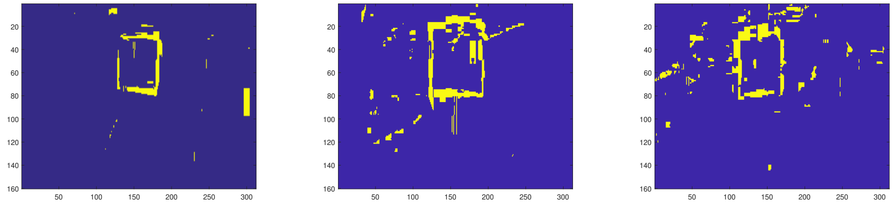
```
procedure SEARCHUPANDDOWN($\mathbf{P0}, image$)
    $\mathbf{P}_1 = \mathbf{P}_0$, $\mathbf{P}_2 = \mathbf{P}_0$, $done = false$
    while $!done$ do
        if isTargetColor($\mathbf{P}_1.x, \mathbf{P}_1.y - 1$) then
            $\mathbf{P}_1.y = \mathbf{P}_1.y - 1$
        else if isTargetColor($\mathbf{P}_1.x - 1, \mathbf{P}_1.y - 1$) then
            $\mathbf{P}_1.x = \mathbf{P}_1.x - 1$
            $\mathbf{P}_1.y = \mathbf{P}_1.y - 1$
        else if isTargetColor($\mathbf{P}_1.x + 1, \mathbf{P}_1.y - 1$) then
            $\mathbf{P}_1.x = \mathbf{P}_1.x + 1$
            $\mathbf{P}_1.y = \mathbf{P}_1.y - 1$
        else
            $done = true$
        end if
    end while
    $done = false$
    while $!done$ do
        if isTargetColor($\mathbf{P}_2.x, \mathbf{P}_1.y + 1$) then
            $\mathbf{P}_2.y = \mathbf{P}_2.y + 1$
        else if isTargetColor($\mathbf{P}_2.x - 1, \mathbf{P}_1.y + 1$) then
            $\mathbf{P}_2.x = \mathbf{P}_2.x - 1$
            $\mathbf{P}_2.y = \mathbf{P}_2.y + 1$
        else if isTargetColor($\mathbf{P}_1.x + 1, \mathbf{P}_1.y + 1$) then
            $\mathbf{P}_2.x = \mathbf{P}_2.x + 1$
            $\mathbf{P}_2.y = \mathbf{P}_2.y + 1$
        else
            $done = true$
        end if
    end while
    return $\mathbf{P}_1$, $\mathbf{P}_2$
end procedure
```

seen that with increasing $\sigma_{cf}$, false positive detections decrease without significantly decreasing of TPR.

In autonomous drone race 2017, we tuned $\sigma_L$ through experimental trial-and-error and accept the detection with highest color fitness, from which, the ROC point is plotted by red circle in Figure 2.6 and Figure 2.7. It is remarkably close to the optimal thresholds one would pick, given this more extended analysis. Please note that the algorithm used in the 2017 drone race only accepted the gate with the highest color fitness, and not every gate that was over the color threshold. It should also be noted that the method described above can be used for tuning $\sigma_L$ and $\sigma_{cf}$ automatically. However, manually labeling and running snake gate detection on the dataset for each set of parameters is time-consuming especially when the drone needs to be deployed in a new racing track



(a) Ture positive detection (b) Ture positive detection and false positive detection (c) False negative detection



(d) (e) (f)

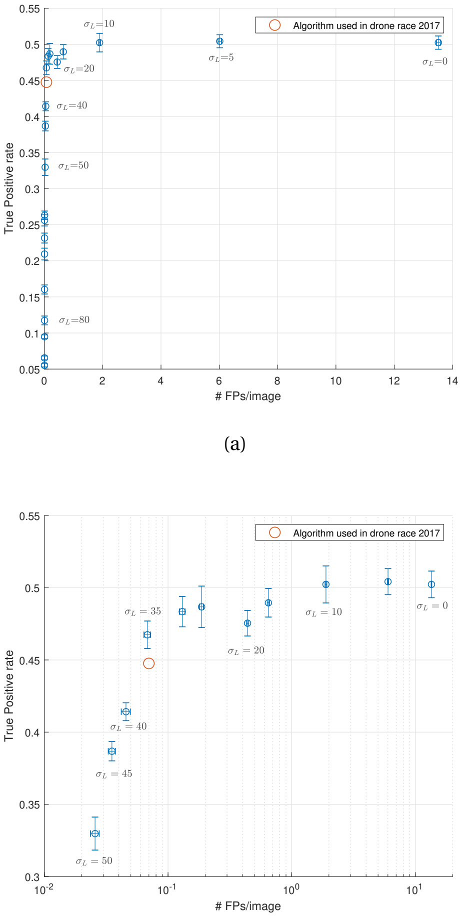
Figure 2.5: Examples of the snake gate detection results. The first row are original onboard images with detection results. The second row are corresponding masks

with limited preparing time.

It should be noted that the true positive rate in above figures is the statistical result on the entire dataset. In order to evaluate how good or bad a true positive rate of 0.46 is, one has to take additional factors into account. Importantly, the distance between the drone and the gate can significantly affect the detection. Figure 2.8 shows how the true positive rate changes with the change of distance between the gate and the drone. It is very clear that when the drone gets closer to the gate, the snake gate detection has a higher true positive rate, reaching 70% at close distances.

Figure 2.9 shows the detection result while the drone approaches the gate. In the beginning, the distance between the drone and the gate is large which leads to false negative detections. Once the drone starts detecting the gate, it can detect the gate most of times. However, there still exist some false negative detections. But these false negative detections could be handled by filters which will be explained in details next section.

When the drone is close to the gate ($< 1m$), only part of the gate can be seen. In this scenario, snake gate detection will not detect the gate. A second detection called histogram gate side detection is employed to replace snake gate detection when the position estimate from the Kalman filter is < 1m (Figure 2.10). This detection algorithm accumulates the number of target color pixels by each column. Then two peaks of the histogram which represent two sidebars of the gate can be found. Later, the position of these two bars can be used by pose estimation to extract relative position between the gate and the drone.



(a)

(b)

Figure 2.6: The ROC curve with the change of $\sigma_L$.

### 2.3.2. POSE ESTIMATION

When a gate with known geometry is detected, its image can provide the pose information of the drone. The problem of determining the position and orientation of a camera given its intrinsic parameters and a set of $n$ correspondences between 3D points and their 2D projections is called Perspective-n-Point (PnP) problem [21]. In our case, 4 coplanar control points (gate corners) are available which leads to a unique solution [22]. However, PnP is sensitive to the mismatches of 3D points and 2D points which in our case is inevitable because the vibration and complex environment. Therefore, these methods are usually combined with RANSAC scheme to reject noise and outliers. Unfortunately, the fact that only four corner points are available on one gate limits the effectiveness of such a scheme. In this section, a novel algorithm combining gate detec-

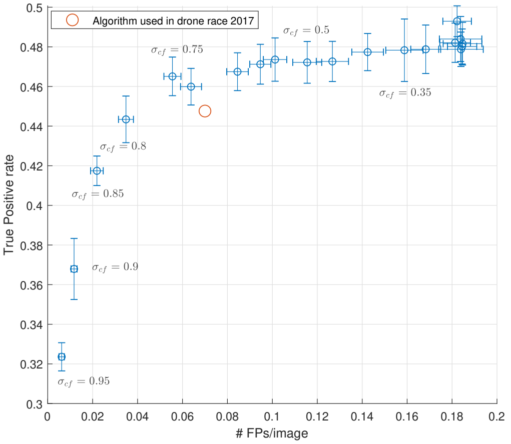


Figure 2.7: ROC curve with $\sigma_L = 25$ and varying $\sigma_{cf}$

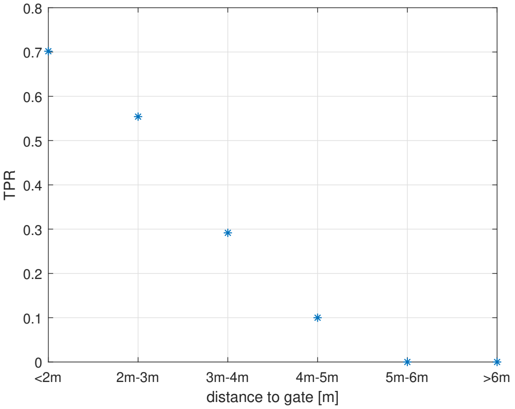


Figure 2.8: When the drone approaches the gate, true positive rate becomes larger because of larger and clearer gate on image plane

tion result and the onboard AHRS attitude estimation will be derived to provide the pose estimation of the drone.

Since we are using a fish-eye camera, a calibration procedure should be done first [23]. Then, the camera can be simplified as a pinhole camera model (Figure 2.11). According to the similar triangle principle, we have

$$\begin{bmatrix} x^C_{\mathbf{P}'} \\ y^C_{\mathbf{P}'} \end{bmatrix} = \begin{bmatrix} f & 0 \\ 0 & f \end{bmatrix} \begin{bmatrix} \frac{x^C_{\mathbf{P}}}{z^C_{\mathbf{P}}} \\ \frac{y^C_{\mathbf{P}}}{z^C_{\mathbf{P}}} \end{bmatrix} \tag{2.2}$$

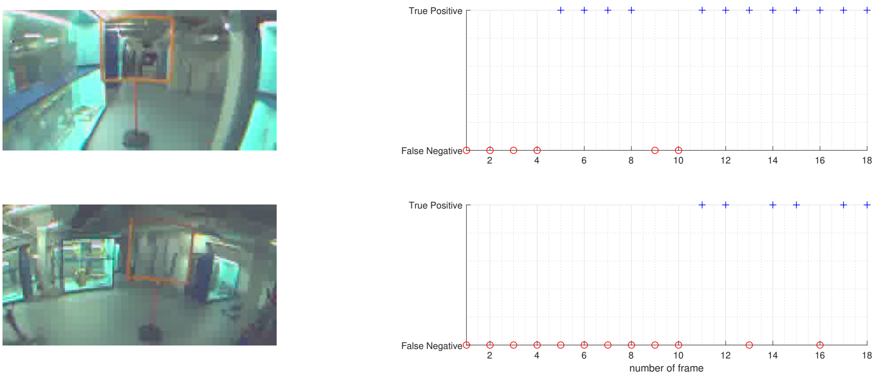


Figure 2.9: While the drone approaches the gate, there still exist some false negatives which may caused by light condition and distortion.

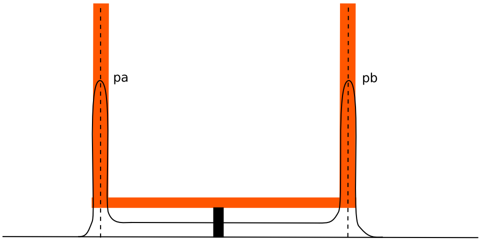


Figure 2.10: When the drone is close to the gate, only a part of the gate can be seen. A histogram of target color in x axis is employed. Two side bars can be found by the two peaks of the histogram

Assume that each pixel's size is $d_x$ and length $d_y$ and the principle points' coordinate is $(C_x, C_y)$, we could transfer the pinhole model 2.2 to

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{f}{d_x} & 0 \\ 0 & \frac{f}{d_y} \end{bmatrix} \begin{bmatrix} \frac{x^c_{\mathbf{P}}}{z^c_{\mathbf{P}}} \\ \frac{y^c_{\mathbf{P}}}{z^c_{\mathbf{P}}} \end{bmatrix} + \begin{bmatrix} C_x \\ C_y \end{bmatrix} \tag{2.3}$$

To write the pinhole model 2.3 in homogeneous coordinates, we have

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x^c_{\mathbf{P}}}{z^c_{\mathbf{P}}} \\ \frac{y^c_{\mathbf{P}}}{z^c_{\mathbf{P}}} \\ 1 \end{bmatrix} \tag{2.4}$$

where $f_x = f/d_x$, $f_y = f/d_y$. $u$, $v$,$C_x$ and $C_y$ are in pixel unit. From Figure 2.11, it can be seen that the 3D point $P$, the image point $P^{'}$ and the focal point $O_c$ are on one line. Thus, the direction of the light ray from $O_c$ to $P$ can be described by a bearing vector $\mathbf{v}$

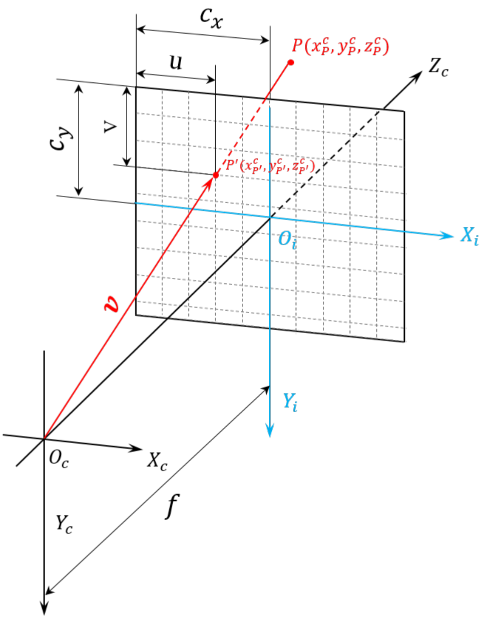


Figure 2.11: A pinhole camera model. $O_c$ is the focal point and the origin of camera frame $O_c X_c Y_c Z_c$. $f$ is the focus. $O_i X_i Y_i Z_i$ is image frame. $\mathbf{P}$ is a 3D point in space and $\mathbf{P}^{'}$ is its image point on image plane

which can be expressed in camera frame by

$$\mathbf{v} = \begin{bmatrix} v_x^c \\ v_y^c \\ v_z^c \end{bmatrix} = \begin{bmatrix} (u - C_x)/f_x \\ (v - C_y)/f_y \\ 1 \end{bmatrix} \tag{2.5}$$

To express vector $\mathbf{v}$ in earth frame, we introduce 2 rotation matrices $\Re_C^B$ and $\Re_B^E$. $\Re_C^B$ is the rotation matrix from camera frame $C$ to body frame $B$ which is a fixed matrix. $\Re_C^B$ is the rotation matrix from body frame $B$ to earth frame $E$ consist of three Euler angle $\psi$, $\theta$ and $\phi$, which can be measured from onboard AHRS system. Thus, bearing vector $\mathbf{v}$ could be expressed in the Earth frame $E$ by

$$\mathbf{v} = \begin{bmatrix} v_x^E \\ v_y^E \\ v_z^E \end{bmatrix} = \Re_B^E \Re_C^B \begin{bmatrix} v_x^c \\ v_y^c \\ v_z^c \end{bmatrix} \tag{2.6}$$

A line passing through point $\mathbf{P}$ with direction $\mathbf{v}$ can be written as

$$L(\mathbf{p}, \mathbf{v}) = \mathbf{p} + \lambda \mathbf{v}, \lambda \in [-\infty, +\infty] \tag{2.7}$$

The perpendicular distance $D(\mathbf{t}; \mathbf{p}, \mathbf{v})$ of a point $\mathbf{t}$ to line $L(\mathbf{p}, \mathbf{v})$ is

$$D(\mathbf{t}; \mathbf{p}, \mathbf{v}) = \left\| (\mathbf{p} - \mathbf{t}) - ((\mathbf{p} - \mathbf{t})^T \mathbf{v}) \mathbf{v} \right\|_2 \tag{2.8}$$

According to the pinhole model, 4 light rays with bearing vectors $\mathbf{v}_i$ from four corners of the gate should intersect at the focal point $\mathbf{t}$ ( Figure 2.12), which is the position of the drone. The bearing vectors can be calculated by the four points' images on the image plane and camera's intrinsic parameters. This intersection point could be calculated analytically. However, due to the detection error of the gate's corners, bearing vectors can be wrongly calculated, for example, in Figure 2.12 four light rays do not intersect at one point.(gray line) Thus, there is no analytical solution of camera's position. Instead of finding analytical solution of camera's position, a numerical solution is found that finds a point whose distance to the four light rays is minimum. Hence, estimating the position of the drone can be converted to an optimization problem that finds an optimal point $\mathbf{t}$ which has minimal distance to 4 light rays, which can be expressed mathematically by

$$\min_{\mathbf{t}} \sum_{i=1}^{4} D(\mathbf{t};\mathbf{p}_i,\mathbf{v}_i) \tag{2.9}$$

which is a least squares problem.

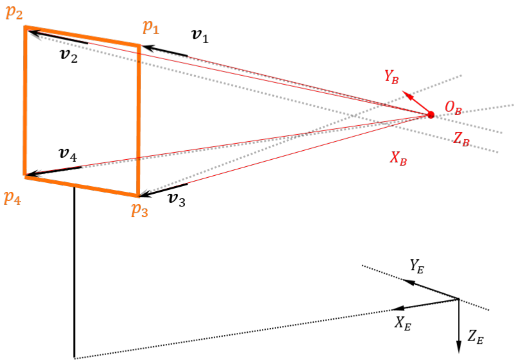


Figure 2.12: Four light rays from four corners of the gate with bearing vector $\mathbf{v}_i$, which could be calculated by four corner's images on image plane and camera's intrinsic parameters, should intersect at focal points. (red line) However, wrong bearing vectors from wrong detections could make the light rays not intersect at one point.

When the drone is close to the gate, only two sidebars can be detected by the histogram method. With the position of bars on the image plane, the pose of the drone can be estimated by geometrical principle. In Figure 2.13, $\alpha_1$ and $\alpha_2$ are calculated by the position of the image of two bars on image plane and intrinsic parameters. Then we have

$$\begin{aligned}
\gamma &= \frac{\pi}{2} - \alpha_2 \\
\frac{r_1}{\sin\gamma} &= \frac{g_s}{\sin(\alpha_1+\alpha_2)} \\
x_h &= r_1 \cos\alpha_1 \\
y_h &= \frac{g_s}{2} - r_1 \sin\alpha_1
\end{aligned} \tag{2.10}$$

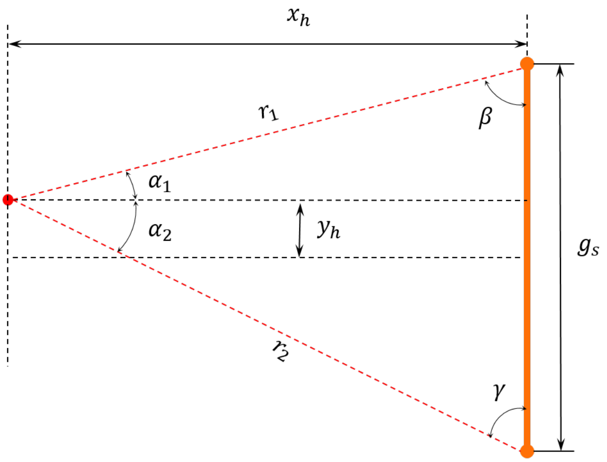
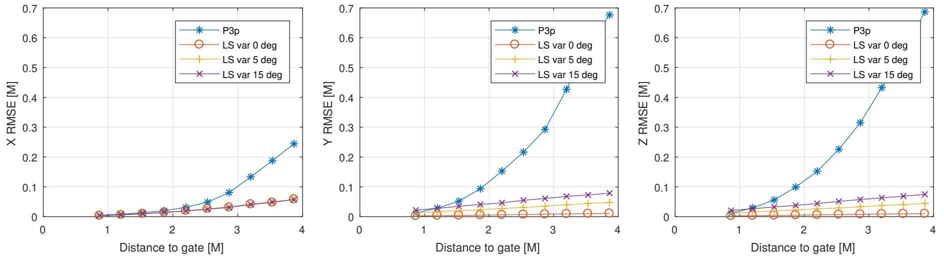


Figure 2.13: The top view of the position of the drone and the gate

where $g_s$ is the length of the gate. Hence, based on the detection of the histogram peaks in the image (corresponding to $\alpha_1$ and $\alpha_2$), we can deduce the lateral position of the camera with respect to the gate ($x_h$ and $y_h$).

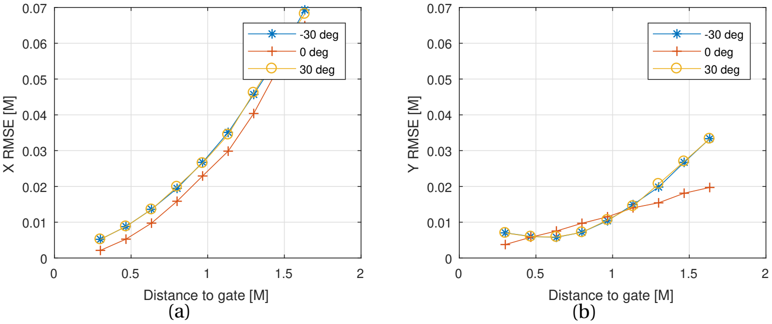
A simulation is done to test the performance of our algorithm and compare it with a standard PnP method. For simulation, artificial gates are created, which are projected onto a virtual pinhole camera image. Since gate detections contain image noise and outliers, a set of real gate detections are compared with ground truth data. Based on this test the vision method experiments will therefore contain image noise with a standard deviation of 3.5 pixels. The Root Mean Squared Error (RMSE) is used to evaluate the performance of both algorithms. The result is shown in Figure 2.14 where each point represents a thousand trails of the position estimation algorithm in the presence of pixel noise. It can be seen that the error varies mainly as a function of distance to the gate. The LS method uses prior knowledge of the attitude and heading of the vehicle to obtain a more accurate position estimate. To study the effect of attitude error, noise with a variance of 0, 5 and 15 degrees is added to the attitude and heading estimates. It is clear from the figure that the LS method has far higher accuracy in RMSE compared to the PnP method, even in the presence of relatively large noise in the attitude estimate.

Also, the histogram position estimation method is evaluated in simulation. Similar to the LS method, pixel noise with a standard deviation of 3.5 is introduced. Figure 2.15 shows the results of the position RMSE in the horizontal plane in x and y-direction. The experiment is performed with a heading angle of -30, 0 and 30 degrees. From the figure, it can be observed that the position error of this method is relatively low. However, in reality, the method is only effective up to a maximum distance of 1.5 meters, due to the possible background color leading to spurious histogram peaks that are hard to filter out.



(a) (b) (c)

Figure 2.14: Simulation result of P3P and LS method. With the incrementation of the distance between the drone and the gate, both methods' error increase. However, LS method has much less error than P3P.



(a) (b)

Figure 2.15: X and Y histogram position RMSE as function of distance to the gate

### 2.3.3. VISION-IMU STATE ESTIMATION

In order to close the control loop, state estimation is essential since the measurements (in our case, distance from vision, acceleration and angular velocity from IMU) are inevitably noisy and biased. A common approach for state estimation is the Kalman filter and its variants such as extended Kalman filter(EKF), Unscented Kalman Filter(UKF) and Particle filtering. In the field of UAVs, 15-states (position $\mathbf{x}$, velocity $\mathbf{v}$, attitude $\phi$ and IMU bias $\mathbf{b}$) Kalman filter is used commonly in many scenarios. It first integrates angular rate to gain rotation matrix from body to earth $\Re_B^E$. Next, $\Re_B^E$ is used to rotate acceleration measured by the accelerometer to earth frame. Then, the acceleration will be integrated twice to gain the position. And finally, position measurement will be used to correct the position prediction. Usually, UAVs' onboard IMUs are low-cost MEMS which suffer from biases and noise severely. During the prediction phase, the bias of accelerometer is integrated twice which may cause the prediction to deviate from the real position over time. If the position measurement has a relatively high frequency, the deviation of the position prediction could be corrected before it diverges. At the same time, the bias of IMU could also be estimated as states in the system and it should converge in short time. However, in our case, position measurements come from onboard image processing which has a low rate of around 20 HZ and the drone may cover significant durations without vision measurements. In this case, position prediction may deviate largely before new position measurement comes. Thus, the bias estimation converges slowly. In this section, we adopt the drone's aerodynamics model to the prediction model in Kalman filter which has a better performance than classic 15-states Kalman filter.

The kinematics of the drone can be described by

$$\dot{\mathbf{X}} = \mathbf{V} \tag{2.11}$$

To express **V** in body frame, we have

$$\begin{bmatrix} \dot{x}^E \\ \dot{y}^E \\ \dot{z}^E \end{bmatrix} = \Re_B^E(\phi,\theta,\psi) \begin{bmatrix} v_x^B \\ v_y^B \\ v_z^B \end{bmatrix} \tag{2.12}$$

where,$x^E, y^E, z^E$ are the drone's position in earth frame $E$. $v_x^B, v_y^B, v_z^B$ are the drone's velocity in body frame $B$. One property of the onboard accelerometer is that it measures specific force $\mathbf{F}_s$ in body frame $B$ instead of vehicle's acceleration. The specific force in $\mathbf{Z}^B$ direction is mainly caused by thrust $\mathbf{T}$ under the assumption that the thrust of quadrotor is aligned with $\mathbf{Z}^B$. The force acting on $\mathbf{X}^B$ and $\mathbf{Y}^B$ can be caused by many factors, for instance, blade flapping, profile drag, and translational drag. But they could be approximated as a linear function, assuming that the indoor environment has no wind [24]:

$$\begin{bmatrix} a_x^B \\ a_y^B \end{bmatrix} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \begin{bmatrix} v_x^B \\ v_y^B \end{bmatrix} \tag{2.13}$$

where $k_x$ and $k_y$ are drag coefficient which could be identified off-line. With this property, the accelerometer can actually provide the information of velocity of the drone by

$$\begin{bmatrix} v_x^B \\ v_y^B \end{bmatrix} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}^{-1} \begin{bmatrix} a_x^m - b_a^x \\ a_y^m - b_a^y \end{bmatrix} \tag{2.14}$$

where $a_x^m$ and $a_y^m$ are the measurement of accelerometer. $b_a^x$ and $b_a^y$ are the bias of accelerometer. Combine equation 2.12 and equation 2.14, we have

$$\begin{bmatrix} \dot{x}^E \\ \dot{y}^E \\ \dot{z}^E \end{bmatrix} = \Re_B^E(\phi,\theta,\psi) \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} a_x^m - b_a^x \\ a_y^m - b_a^y \\ v_z^B \end{bmatrix} \tag{2.15}$$

In equation 2.15, the bias only needs to be integrated once to predict the position of the drone instead of being integrated twice in original 15-states Kalman filter, which could help to decrease the error of prediction.

As mentioned above, the onboard AHRS system is a complementary filter, which on a low level fuses accelerometer and gyro data to estimate the attitude of the drone. It can directly provide the attitude estimation to the outer loop. The AHRS fusing only IMU data may introduce a bias to the attitude estimation. In this paper, we assume that this low level attitude estimation bias can be neglected. Hence, AHRS and accelerometer reading can be used as inputs to propagate the prediction model 2.15.

According to Newton's laws of motion, the motion of the drone can be described as

$$\begin{bmatrix} \dot{v}_x^B \\ \dot{v}_y^B \\ \dot{v}_z^B \end{bmatrix} = \Re_E^B \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \begin{bmatrix} a_x^m - b_a^x \\ a_y^m - b_a^y \\ a_z^m - b_a^z \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} v_x^B \\ v_y^B \\ v_z^B \end{bmatrix} \tag{2.16}$$

where $g$ is gravity factor and $p, q, r$ are angular velocity in body frame $B$ measured by the gyro. Since in equation 2.15, body velocity has already had measurements from the accelerometer, in equation 2.16, we omit the first 2 equations and only leave the last equation combining with 2.14, which results

$$\dot{v}_z^B = a_z^m - b_a^z + g\cos\theta\cos\phi + q\frac{a_x^m - b_a^x}{k_x} - p\frac{a_y^m - b_a^y}{k_y} \tag{2.17}$$

With the assumption that gyro's bias is small, which can be neglected and the accelerometer's bias changes slowly,

$$\dot{\mathbf{b}}_a = \begin{bmatrix} \dot{b}_a^x \\ \dot{b}_a^y \\ \dot{b}_a^z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.18}$$

Combining equation 2.15, equation 2.17 and equation 2.18, we have the process model for EKF as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{2.19}$$

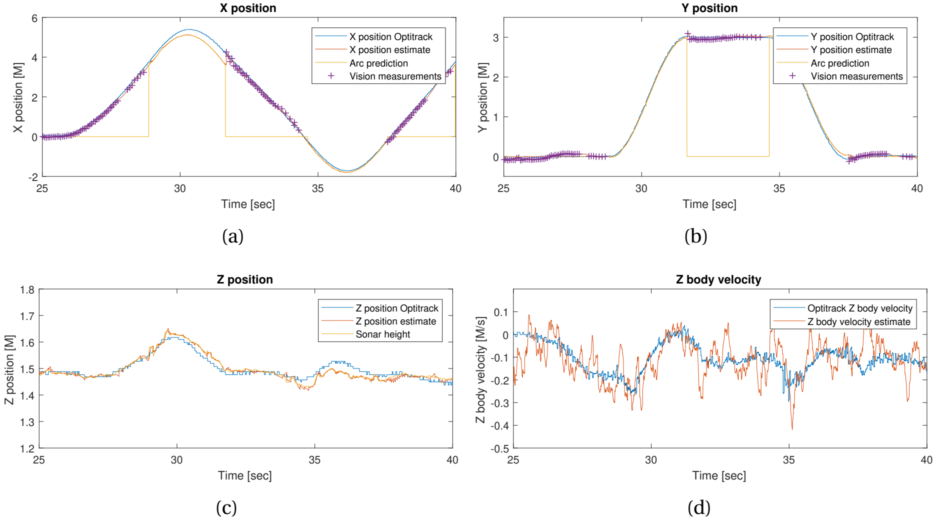
with states and inputs defined by

$$\mathbf{x} = [x_E, y_E, z_E, v_z^B, b_a^x, b_a^y, b_a^z]^T \tag{2.20}$$

$$\mathbf{u} = [\phi, \theta, \psi, a_x^m, a_y^m, a_z^m, p, q]^T \tag{2.21}$$

Then, a standard EKF predict/update procedure will be done to estimate the states, which can be found in Appendix.

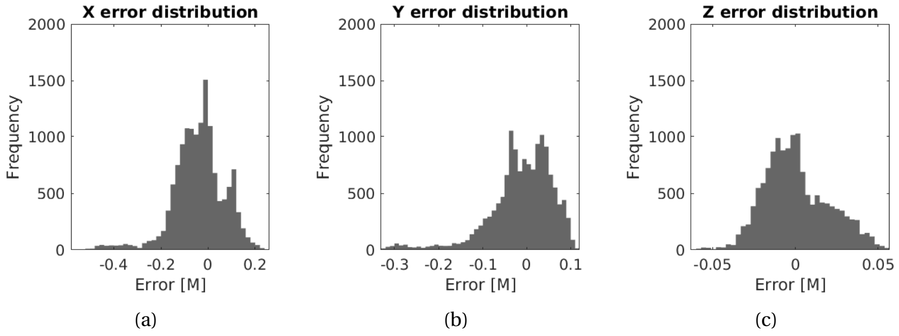
To evaluate the performance of the visual navigation method described in this section, a flight test with a simplified two-gates track where the drone flies through two gates cyclically is done (Figure 2.3). A first experiment aims to gather onboard data to be analyzed off-line. Hence, Opti-track system is used to provide accurate position measurements to make the loop closed. It should be noted that only in straight parts, the gate is in the drone's filed of view and the snake gate detection algorithm is done onboard, while the pose estimation and EKF are done off-board. The outer-loop controller is a PD controller combining Opti-track measurements to steer the drone to align with the center of the gate. In the arc parts, the gates are no longer available for navigation and the drone navigates itself to fly along an arc only by state prediction without the involvement of Opti-track, which will be explained in details in next section. The filtering result is shown in Figure 2.16. During the straight part (purple vision measurements), the EKF runs state prediction and measurement update loop and the estimated states curves (red) coincide with ground truth curves (blue) well. The error distributions between estimated states and ground-truth states are shown in Figure 2.17. All histograms are centered around 0 error. But there are still a few estimation errors above 0.2$m$ in both x error and y error distribution which explains the fact that a few arcs end up at points which are more than 0.5m from target endpoint, which could be seen in next section. To make readers clearer to the experiment set up and result, a 3D ground truth and estimation result can be found in Figure 2.18



(a) (b)

(c) (d)

Figure 2.16: Extended Kalman filter result. The straight part flight is done with Opti-track. The vision pose estimation is done onboard. The arc part is done only by state prediction without the involvement of Opti-track.



(a) (b) (c)

Figure 2.17: Extended Kalman filter error distribution

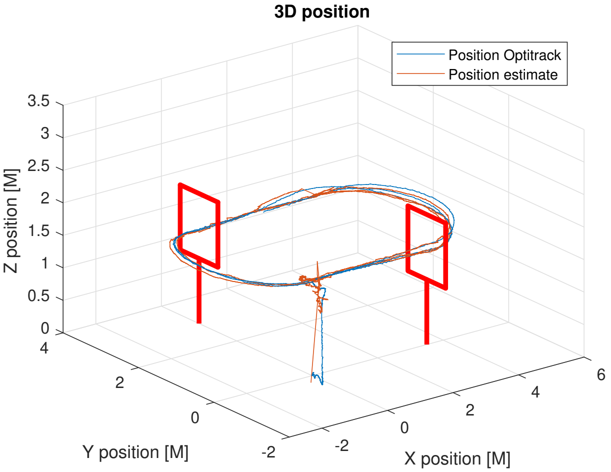


Figure 2.18: Experiment setup. The drone takes off from the ground and flies an oval cyclically. In straight parts, Opti-track is used to help the drone align with the gates, while vision detection is done onboard for logging. In the arc parts, a feed-forward control with state prediction is employed, which explains the reason the arcs end up at slightly different points.

## 2.4. CONTROL STRATEGY

Like classic control strategy of quadrotor, our control system is also divided into a inner-loop controller which stabilizes attitude of the quadrotor and a outer-loop controller which steers the quadrotor along the desired trajectory. For the inner-loop controller, an INDI controller is employed on-board[15].

For outer-loop control, we have two different control strategies for straight parts and arc parts respectively (Figure 2.19). During the straight part where the drone faces the gate and the gate is available for visual navigation, a PD controller is used to command a roll maneuver to steer the drone to align with the center of the gate while the pitch angle is fixed to a certain degree $\theta_0$ and the heading is fixed to the same direction as the gate.

$$\begin{cases} \phi_c = -k_p \hat{y} - k_d \dot{\hat{y}} \\ \theta_c = \theta_0 \\ \psi_c = 0^\circ \end{cases} \tag{2.22}$$

where subscript $c$ means command and position $y$ is defined in local frame whose origin is fixed at the center of the gate.

At the point the drone flies through the gate, no position measurement is available. Thus, the outer-loop controller has to be switched to a pure feed-forward controller relying on state prediction to turn a coordinated arc which ends in front of the next gate. To derive the control law in the arc, we first introduce body fixed earth frame $F$ (Figure 2.20) whose origin $O^F$ is at the mass point of the drone, $X^F$ is along the heading of the drone, $Z^F$ points to the earth. In other words, the only non-zero Euler angle from $E$ to $F$ is yaw which is the same with the drone's yaw angle. To express Newton second law in $F$ we have

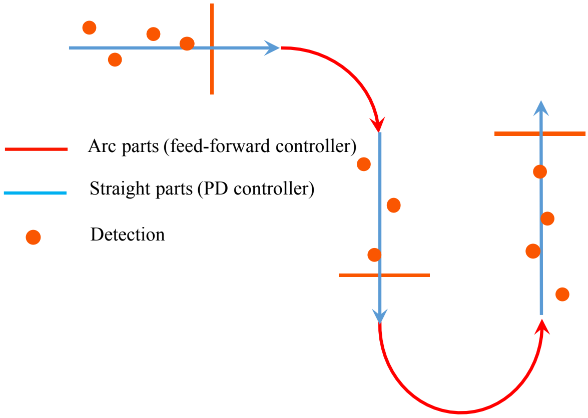

Arc parts (feed-forward controller)
Straight parts (PD controller)
Detection

Figure 2.19: Two control strategies used in the experiment. When the drone faces the gates (straight parts), A PD controller combined with the Kalman filter is used to steer the drone to align with the gate. After passing through the gate, the drone switches to a feed-forward controller to fly an arc which ends in front of the next gate.

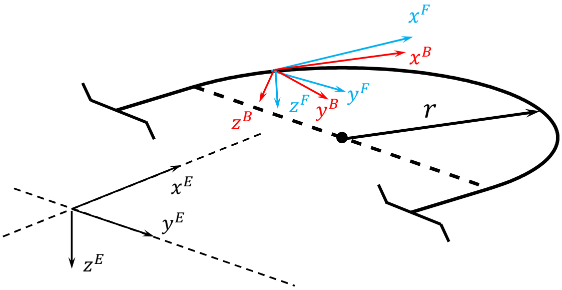


Figure 2.20: Body fixed earth frame $F$ whose origin $O^F$ is at the mass point of the drone, $X^F$ is along the heading of the quadrotor, $Z^F$ points to the earth. The rotation matrix from $E$ to $F$ is $\Re^F_E(\psi)$. The rotation matrix from $B$ to $F$ is $\Re^F_B(\phi,\theta)$

$$\left.\frac{\partial \mathbf{v}}{\partial t}\right|_F + \Omega \times \mathbf{v} = \mathbf{F} \tag{2.23}$$

where $\left.\frac{\partial \mathbf{v}}{\partial t}\right|_F$ is the derivative of $\mathbf{v}$ in $F$, $\mathbf{F}$ is the force acting on the drone and $\Omega$ is angular velocity of Frame $F$ with respect to earth frame $E$. During the arc, the drone's heading is supposed to be tangent to the arc to maintain a zero sideslip turn, the angular velocity of $F$ with respect to $E$ should be

$$\Omega = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{v_x^F}{r} \end{bmatrix} \tag{2.24}$$

To express equation 2.23 in scalar form, we have

$$\begin{bmatrix} \frac{\partial v_x^F}{\partial t} \\ \frac{\partial v_y^F}{\partial t} \\ \frac{\partial v_z^F}{\partial t} \end{bmatrix} = \Re_E^F \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \Re_B^F \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} + \begin{bmatrix} a_x^F \\ a_y^F \\ a_z^F \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \frac{v_x^F}{r} \end{bmatrix} \times \begin{bmatrix} v_x^F \\ v_y^F \\ v_z^F \end{bmatrix} \tag{2.25}$$

where $T$ is the thrust of the drone and

$$\begin{bmatrix} a_x^F \\ a_y^F \\ a_z^F \end{bmatrix} = \Re_B^F \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} \Re_F^B \begin{bmatrix} v_x^F \\ v_y^F \\ v_z^F \end{bmatrix} \tag{2.26}$$

During the arc, we would like to keep the altitude not changed, which in this frame means at the same height as at the start of the arc. Thus to make $\frac{\partial v_z^F}{\partial t} = 0$ in equation 2.25, we can have

$$T = \frac{-g - a_z^F}{\cos\theta \cos\phi} \tag{2.27}$$

In the arc, $\frac{\partial v_y^F}{\partial t}$ should be enforced to 0, substitute equation 2.27 to the second line of equation 2.25, we have,

$$\phi_c = \tan^{-1} \frac{(a_y^F - \frac{v_x^{F2}}{r}) \cos\theta}{-g - a_z^F} \tag{2.28}$$

Similar to the straight part, pitch command $\theta_c$ is also fixed to a certain value. To conclude, during the arc maneuver, the control inputs are

$$\begin{cases} \psi_c(t) = \int_0^t \frac{v_x^F(t)}{r} dt \\ \phi_c(t) = \tan^{-1} \frac{(a_y^F(t) - \frac{v_x^F(t)^2}{r}) \cos\theta_c(t)}{-g - a_z^F(t)} \\ \theta_c(t) = \theta_0 \end{cases} \tag{2.29}$$

The flight test result can be found in Figure 2.21. The drone enters the arc at red points and starts feed-forward control with the control strategy in equation 2.29. In a feed-forward arc maneuver, $\theta_c = -5°$, $r = 1.5m$ and each arc takes around $2s$. Before entering the arc, the drone is steered by the feedback control strategy in equation 2.22. At the same time, visual navigation is running to estimate the states of the drone which also tells the drone where to start to turn an arc. Thus in each lap, red points are slightly different from each other which is caused by filtering error. It could also be seen that the endpoints (yellow points) of arc maneuver has a distribution with larger variance compared to that at entry points. It is mainly because that state prediction in principle is an integration based method, which may be highly affected by the accuracy of initial states. In table 2.2, it is clear that the error at entry point in the x direction is much less than the one in the y direction. As a result, the error in the y axis at the endpoints is larger than that in the x axis. This error can also be caused by model inaccuracy and the disturbance during the arcs. Thus, the pure feed-forward control strategy is only

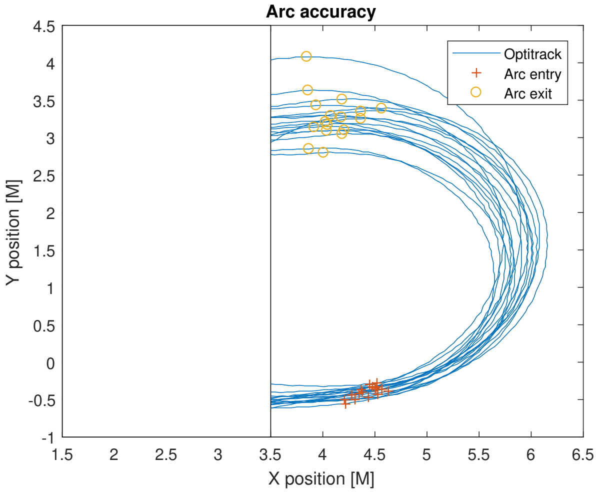


Figure 2.21: The flight test result of feed-forward control.The start points of the arcs (red points) slightly differ from each other because of the filter error. The end points of the arcs have a larger variance because the arc maneuvers are based on state prediction which is affected by model accuracy and initial state estimation.

Table 2.2: Feed-forward control accuracy distribution

| Axis | Entry speed variance $\sigma_v$ | Position error variance $\sigma_x$ |
|---|---|---|
| X | $0.0043 m/s$ | $0.0296 m$ |
| Y | $0.0106 m/s$ | $0.8087 m$ |

effective for short time durations. In our case, $2s$ is enough to steer the drone to the next gate where visual navigation is available and feedback control strategy can be switched on again.

After the arc, the drone will detect the gate again and the detection will correct the filtering error. Thus, there will be a jump in the filtering result (Figure 2.22). For the feedback controller, the control target is to steer the drone to $y = 0$. In fact, this is a simple step signal tracking or a way-point tracking problem. Simulations are done to check the feasibility of the proposed controller to steer the drone through the gate. The simplified drone model is

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{v}_y = g \tan\phi + k_y v_y \end{cases} \tag{2.30}$$

where $x$ and $y$ are the position of the drone and $v_y$ is the velocity of the drone in $y$ direction. In this model, we neglect $z$ because in the real-world flight, the altitude is controlled by a separate controller which can keep the altitude to be a constant. $v_x$ is the input of the model because in our real-world experiment setup, $\theta$ is set to be a constant which leads to a constant velocity in $x$ direction. $\phi$ is another input of the model. A PD controller is employed to steer the drone to $y = 0$ by $\phi = k_v(k_p(0 - y) - v_y)$, where $k_p = 1$

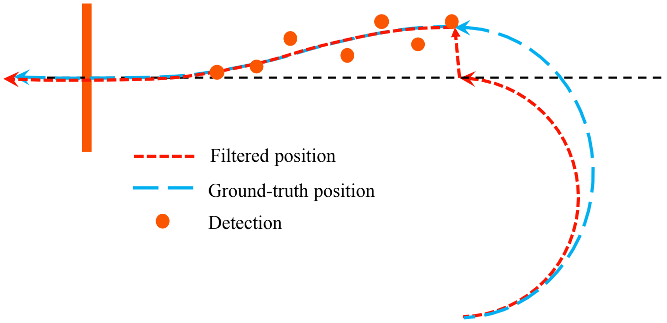
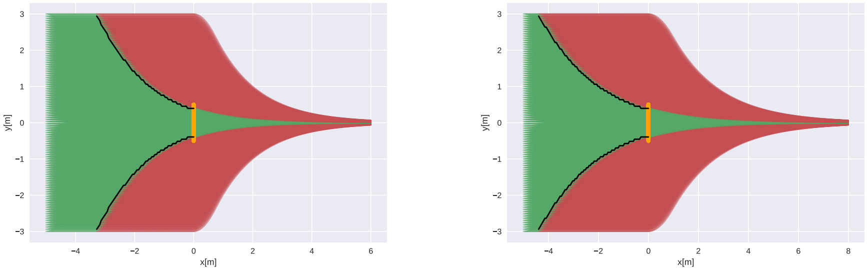


Figure 2.22: During the arc maneuver, the drone will not detect the gate. Thus, the state estimation is purely based on the prediction (red arc). However, due to the model inaccuracy and the sensors' bias, the predicted trajectory will diverge from the ground-truth trajectory (blue curve). After the turn, the drone will detect the gate again and the estimated position will jump to the ground-truth position. In fact, although there is a jump in the state estimation (red curve), the real-world trajectory should be continued (blue curve).

and $k_v = 2$. The simulation result can be found in Figure 2.23



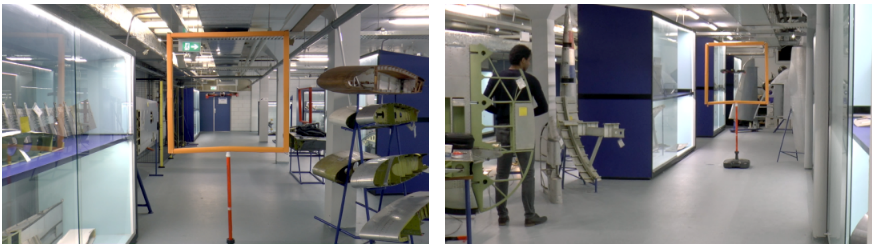
(a) $v_x = 1.5m/s$ (b) $v_x = 2m/s$

Figure 2.23: The simulation result of the drone's passing through the gate. Whether the drone can pass through the gate depends on its initial position $x_0, y_0$ and its forward speed $v_x$. In each figure, 10000 simulations are done with different initial points $x_0 \in [-5m, 0m]$, $y_0 \in [-3m, 3m]$. The area to the left of the black curve is the set of the points, from which the drone can pass through the gate. Obviously, when the forward speed gets larger, the feasible initial points become less.

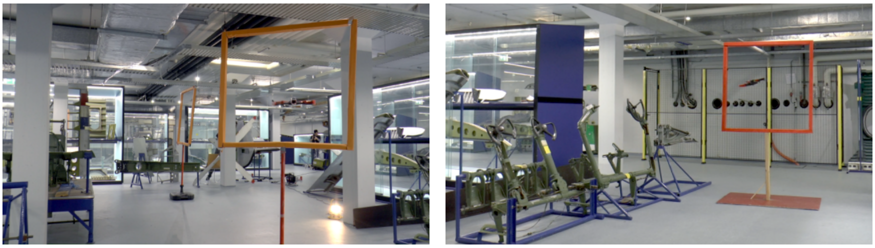
Figure 2.23 is the simulation result with the forward speed $v_x = 1.5m/s$ and $v_x = 2m/s$. In each figure, 10,000 trajectories are simulated with their own initial points $x_0 \in [-5m, 0m]$, $y_0 \in [-3m, 3m]$. The points to the left of the black curves are the initial points from which the drone can pass through the gates. It can be seen that when the drone's speed gets higher, the number of the feasible initial points gets smaller. In other words, the drone needs more distance to adjust its position to pass through the gate. In our real-world experiment set up, for example, the forward speed is around $1.5m/s$ and the position error in $y$ axis is $0.8m$ as shown in Table 2.2. The drone needs a margin of $2m$ in $x$ direction to steer itself through the gate safely.

## 2.5. FULL TRACK EXPERIMENT SETUP AND RESULT

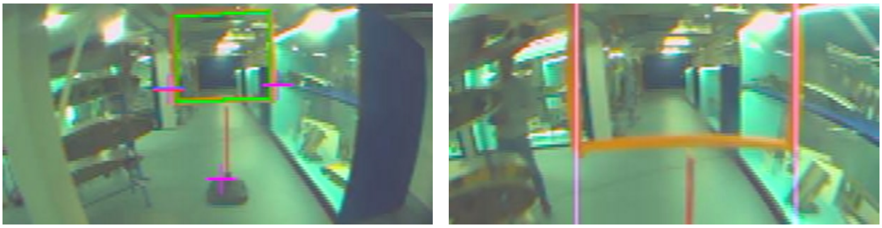
In the previous sections, we have discussed the proposed visual navigation method and control strategies and the results of the experiments designed to verify our method in laboratory environment. In this section, we integrate all subsystems and move to a more challenging and realistic environment, a showroom in the basement of the Faculty of Aerospace Engineering, TU Delft where many aircraft components are displayed, to test the performance of our method. In this showroom, we placed five $1m \times 1m$ gates in the corridor which is surrounded by dense showcases and aircraft components such as aircraft flaps, rudders, yokes and so on. The five gates are shown in Figure 2.24. Compared to the IROS 2017 autonomous drone racing, this track has smaller gates, much denser obstacles and the background of the gates is complex which in all put many challenges for the drone to fly the whole track fully autonomously.



(a) The first gate of the track (b) The second gate of the track



(c) The third and fourth gate of the track (d) The fifth gate of the track



(e) Onboard snake gate detection (f) Onboard histogram detection

Figure 2.24: Five gates are placed in a dense obstacle track. The gates are placed in narrow corridors and are surrounded by dense obstacles such as aircraft flaps, rudders and yokes. The first two row images are the environment around the gates and the last row are the onboard images with detection results.

In this track, the drone takes off from ground and flies through the whole track with $\theta = -5°$ or $\theta = -7°$, which lead to the forward speed to be around $1.5m/s$ and $1.8m/s$ respectively, which is faster than the winner in autonomous drone race in 2016 who flew through 10 gates with $86s$ [2], whose velocity is around $0.5m/s$. The onboard images and the flight result can be found in Figure 2.24 and Figure 2.25.

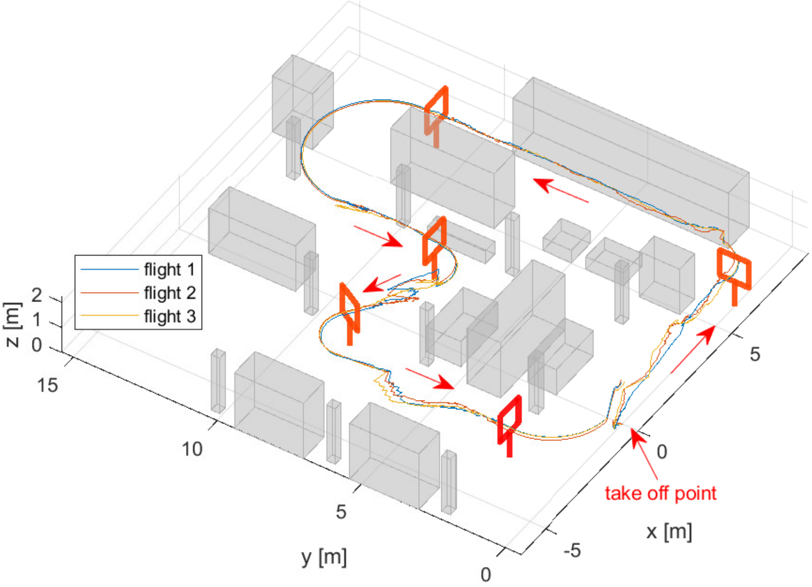


Figure 2.25: 3 independent flight trajectories in the basement. It should be noted that these trajectories are the position estimation result of the flight instead of ground-truth trajectories.

The environment is not equipped with a ground truth position system, therefore only estimated data is available. However, analyzing the estimated trajectory does give an insight of the flight and estimation performance in general. It can be observed that during some parts of the track some rapid changes in position occur. These jumps in position estimate occur once the next gate is first detected after a long period without seeing a gate. During this period the position estimation only relies on the integration of the drag based velocity. Errors in this prediction introduce an accumulating drift in the position estimate, which is corrected when a gate detection is available again. After the correction, the lateral position controller has enough time to steer the drone through the gate.

During the experiments, although in most cases the drone can pass through the gate, there are still some failure cases (the drone crashes to the gate). They are caused by non-detection of the gates or very late detection when the drone is already very close to the gate. In these two scenarios, the drone has to control itself purely based on prediction or the drone has no time to adjust its position. In our basement experiment, the poor quality of the onboard images leads to these non-detection problems. In terms of the open-loop control strategy, with the estimated linear aerodynamic model, we find that the control performance is very accurate in a short time. For example, after the second gate, there is a pole that is close to the arc (Figure 2.25), but the drone never crashed into this pole.

## 2.6. CONCLUSION AND FUTURE WORK

In this paper, we present a systematic scheme to accomplish the task of autonomous drone racing, as held by IROS in 2017. In our work, a novel and computationally efficient gate detection method is implemented onboard a Parrot Bebop 1 drone with all algorithms executed at 20 HZ frequency. With the detected gates, we employ a pose estimation scheme combining onboard AHRS estimation, which has higher accuracy than the commonly used P3P method. Then a more efficient Kalman filter is implemented onboard which converges faster than a traditional 15-states Kalman filter. In terms of the control strategy, a prediction-based feed-forward control strategy is used to control the drone to fly in the short time intervals without position measurements. And finally, the whole system is tested in a showroom with dense showcases and aircraft components. In this flight test, the average speed reached $1.5m/s$ which is higher than the speeds exhibited at the autonomous drone races in 2016 and 2017.

There are multiple directions for future work. For instance, the visual process is essentially based on color detection. Higher robustness in the visual processing may be reached by employing machine learning methods in computer vision. Also, a PD-controller is used to steer the drone through the gate, which makes the trajectory sub-optimal and can on the long term lead to overshoot. This can be improved, e.g., by utilizing optimal control methods. We hope that such future improvements will allow further augmenting the flight speed, hopefully approaching human pilot performance.

## APPENDEX: EXTENDED KALMAN FILTER

(1) Predict states based on equation 2.19

$$\hat{\mathbf{x}}_{k|k-1} = \hat{\mathbf{x}}_{k-1} + \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1})\mathrm{T} \tag{2.31}$$

(2) Linearize and discretize the system

$$\begin{aligned} \mathbf{F}_{k-1} &= \frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_{k-1}} \\ \Phi_{k|k-1} &\approx \mathbf{I} + \mathbf{F}_{k-1}\mathrm{T} \\ \mathbf{H}_k &= \frac{\partial}{\partial \mathbf{x}} \mathbf{h}(\mathbf{x}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_{k|k-1}} \end{aligned} \tag{2.32}$$

(3) Calculate prediction covariance matrix $\mathbf{P}_{k|k-1}$

$$\mathbf{P}_{k|k-1} = \Phi_{k|k-1}\mathbf{P}_{k-1}\Phi^{\mathrm{T}}_{k|k-1} + \mathbf{Q}_{k-1} \tag{2.33}$$

where $\mathbf{Q}_{k-1}$ is system noise covariance matrix.

(4) Calculate Kalman gain and update prediction.

$$\begin{aligned} \delta\hat{\mathbf{x}}_k &= \mathbf{K}_k\left\{\mathbf{Z}_k - \mathbf{h}[\hat{\mathbf{x}}_{k|k-1}, k]\right\} \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}[\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k]^{-1} \\ \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k|k-1} + \delta\hat{\mathbf{x}}_k \end{aligned} \tag{2.34}$$

where $\mathbf{R}_k$ is sensor noise covariance matrix.

(5) Update the covariance matrix of state estimation error

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k/k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^{\mathrm{T}} + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^{\mathrm{T}} \tag{2.35}$$

## REFERENCES

[1] H. Moon, Y. Sun, J. Baltes, and S. J. Kim, *The IROS 2016 Competitions [Competitions],* IEEE Robotics & Automation Magazine **24**, 20 (2017).

[2] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, *A direct visual servoing-based framework for the 2016 iros autonomous drone racing challenge,* Journal of Field Robotics **35**, 146 (2018).

[3] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors,* in *2011 IEEE International Conference on Robotics and Automation* (IEEE, Shanghai, 2011) pp. 2520–2525.

[4] D. Mellinger, N. Michael, and V. Kumar, *Trajectory generation and control for precise aggressive maneuvers with quadrotors,* The International Journal of Robotics Research **31**, 664 (2012).

[5] E. Lyu, Y. Lin, W. Liu, and M. Q.-H. Meng, *Vision based autonomous gap-flying-through using the micro unmanned aerial vehicle,* in *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)* (IEEE, 2015) pp. 744–749.

[6] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, *Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu,* IEEE Robotics and Automation Letters **2**, 404 (2017).

[7] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, *Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision,* in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017) pp. 5774–5781.

[8] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos, *Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight,* IEEE Robotics and Automation Letters **3**, 2799 (2018).

[9] G. Li, M. Mueller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, *Teaching uavs to race with observational imitation learning,* arXiv preprint arXiv:1803.01129 (2018).

[10] A. Good, *Drone race: Human versus artificial intelligence,* `https://www.nasa.gov/feature/jpl/drone-race-human-versus-artificial-intelligence` (2017).

[11] B. Morrell, M. Rigter, G. Merewether, R. Reid, R. Thakker, T. Tzanetos, V. Rajur, and G. Chamitoff, *Differential flatness transformations for aggressive quadrotor flight,* in *Robotics and Automation (ICRA), 2018 IEEE International Conference on Robotics and Automation* (IEEE, 2018) pp. 5204–5210.

[12] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, *Deep drone racing: Learning agile flight in dynamic environments,* arXiv preprint arXiv:1806.08548 (2018).

[13] B. Gati, *Open source autopilot for academic research-the paparazzi system,* in *American Control Conference (ACC), 2013*, IEEE (IEEE, Washington, DC, USA, 2013) pp. 1478–1481.

[14] M. Euston, P. Coote, R. Mahony, J. Kim, and T. Hamel, *A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV,* (2008) pp. 22–26.

[15] E. J. Smeur, G. C. De Croon, and Q. Chu, *Gust disturbance alleviation with incremental nonlinear dynamic inversion,* in *IEEE International Conference on Intelligent Robots and Systems*, Vol. 2016-Novem (2016) pp. 5626–5631.

[16] P. Viola and M. Jones, *Rapid object detection using a boosted cascade of simple features,* Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001 **1**, I (2001).

[17] J. Illingworth and J. Kittler, *A survey of the hough transform,* Computer Vision, Graphics and Image Processing **44**, 87 (1988).

[18] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks,* Nips , 91 (2015).

[19] S. Jung, H. Lee, S. Hwang, and D. H. Shim, *Real time embedded system framework for autonomous drone racing using deep learning techniques,* in *2018 AIAA Information Systems-AIAA Infotech@ Aerospace* (2018) p. 2138.

[20] G. De Croon, C. De Wagter, B. Remes, and R. Ruijsink, *Sub-sampling: real-time vision for micro air vehicles,* Robotics and Autonomous Systems **60**, 167 (2012).

[21] M. Bujnak, Z. Kukelova, and T. Pajdla, *A general solution to the p4p problem for camera with unknown focal length,* in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (IEEE, 2008) pp. 1–8.

[22] M. A. Abidi and T. Chandra, *A new efficient and direct solution for pose estimation using quadrangular targets: Algorithm and evaluation,* IEEE transactions on pattern analysis and machine intelligence **17**, 534 (1995).

[23] P. Dhane, K. Kutty, and S. Bangadkar, *A generic non-linear method for fisheye correction,* International Journal of Computer Applications **51** (2012).

[24] J. Svacha, K. Mohta, and V. Kumar, *Improving quadrotor trajectory tracking by compensating for aerodynamic effects,* in *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on* (IEEE, 2017) pp. 860–866.

# 3

# IN-FLIGHT MODEL PARAMETER AND STATE ESTIMATION USING GRADIENT DESCENT FOR HIGH-SPEED FLIGHT

---

This chapter is based on the following article:

**Li S**, De Wagter C, de Visser CC, Chu QP, de Croon GC. In-flight model parameter and state estimation using gradient descent for high-speed flight. International Journal of Micro Air Vehicles. 2019 Mar;11:1756829319833685.

## 3.1. INTRODUCTION

Quadrotors have received considerable attention in recent years thanks to their mechanical simplicity and good maneuverability combined with hover properties. They have offered new possibilities in a variety of fields like aerial photography, inspection and even transportation. With recent advances in on-board computation and sensor technology, aggressive maneuvering has come within reach of many applications. To further stimulate aggressive and fast flight, autonomous drone racing is gaining interest. The first ever autonomous drone race was held by the International Conference on Intelligent Robots and Systems (IROS) in 2016 [1]. A track consisting of gates had to be flown autonomously in a pre-specified order. The robot had to achieve this as fast as possible, while only relying on onboard sensors and processing. Figure 3.1 illustrates the setup of the 2016 indoor track.

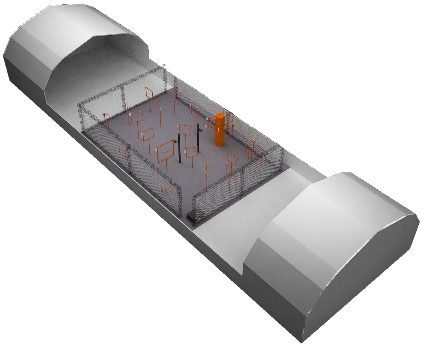


Figure 3.1: The map of the IROS 2016 drone race. In this drone race, the UAVs have to fly through orange gates in a pre-specified order as fast as possible.

Autonomous indoor drone racing brings many new challenges to the fields of quadrotor navigation and control. One initial challenge is the navigation without any external positioning system like Vicon, Optitrack or GPS. Typical approaches to this problem make use of on-board cameras and use Visual-Inertial Odometry to integrate position. This type of algorithms rely on integrating inertial information, tracking visual features over several frames and solving an optimization problem to retrieve the most likely solution. In autonomous drone racing, on top of this position estimation algorithm, gate detection is often needed when the position of gates is not precisely known, or when gates contain moving parts—as is the case in the IROS competitions. With the limited computational resources of small indoor drones, to achieve the fast speeds needed in drone racing, this paper proposes a navigation solution based solely on gate detection, augmented with inertial measurements and an aerodynamic model. To cope with the sometimes sparse and noisy non-Gaussian visual observations, we formulate the navigation solution as an optimization problem. We then solve it using a gradient descent method. The resulting method provides online estimation of the quadrotor position, velocity and inertial biases using less computational resources than traditional Visual Inertial Odometry. The proposed approach also estimates aerodynamic properties of the quadcopter—which become increasingly important in the case of fast aggressive control. Finally, the approach scales favorably with increasing flight speeds as it keeps performing well even with very few position updates. As a comparison, we use the Kalman filter, which is currently still the default choice for navigation. Since the Extended Kalman filter is significantly less computationally complex than the Unscented Kalman filter [2], in this paper we select the Extended Kalman filter as a benchmark. We compare the results with Extended Kalman filter, which is shown to be much more sensitive to visual outliers or other non-Gaussian effects.

In Related work, an overview of studies on aerodynamics modeling and state estimation methods is given. Section Quadrotor model, will describe the quadrotor model parameters that will be solved. Section State estimation proposes two different approaches for the the visual state estimation. First a classic 15-state Extended Kalman Filter (EKF) is developed as benchmark. Then the novel FMINCON-based gradient descent optimization method is proposed to solve the model parameters and states. In Section Experiment setup and result, both algorithms are compared on flight test data and Section Conclusion summarizes the conclusions.

## 3.2. RELATED WORK

Several researchers have already proposed aerodynamics models for quadrotors [3–7]. The main object of their studies is to derive a nonlinear quadrotor aerodynamics model to improve the control performance by compensating for the nonlinear terms. In some studies, a detailed aerodynamic model is analyzed through theory and fitted by experimental data [3]. Simplified aerodynamic models are also established from experiments [5, 6]. It should be noted that their models are all obtained off-line using external measurements, such as GPS, VICON and thrust test beds. Aerodynamic models can also be combined with on-board measurements, for instance from computer vision [8], in order to better estimate the velocity of the drone on-line. In this article, we employ a simplified aerodynamic model in the trajectory estimation exactly for this purpose.

Quadrotor control heavily relies on attitude estimation from an attitude and heading reference system (AHRS). This system is typically based on inertial sensors (accelerometers and gyroscopes), but also relies on orientation sensors (magnetometer) and/or positioning sensors (GPS, Vicon) to estimate inertial sensor biases and compensating for long term drift. Sensor biases become increasingly important as the drone will have to fly longer or temporarily perform feedforward control maneuvers in the absence of sensor measurements. Hence, for drone racing, it is important to estimate them accurately. Here we briefly discuss the sensors and then the filtering employed in estimating both attitude and position or velocity on MAVs.

Most systems intended for outdoor environments utilize the magnetometer and GPS-measurements [9–13]. The indoor equivalent is the use of a motion tracking system such as Vicon or Optitrack [14]. In many applications - like autonomous drone racing, it is required to have accurate state estimation without the help of external systems. The necessary position or velocity measurements can be obtained from multiple sensors. One early option is to use laser scanners [15, 16]. But a laser scanner contains sensitive optics and mirrors, which are susceptible to shock and vibration problems [17]. Another choice for on-board navigation is RGB-D devices [18–20]. The main drawback of these RGB-D devices is that their maximum depth perception range is limited to a few meters [21, 22]. This is why light-weight and inexpensive on-board cameras which are more robust to vibration and shock, have attracted interest of researchers for the navigation of drones. Generally, visual odometry (VO) algorithms [23] using a stereo camera or monocular camera are used for estimation of the MAV's translation and rotation between frames [24–28]. However, generic visual odometry approaches necessitate detecting features, matching corresponding features and estimating motions, which leads to a heavy demand for on-board computational resources and low-frequency estimation. In the meantime, aggressive maneuvers may introduce blur into generic visual odometry and seriously affect the accuracy of estimation. Moreover, in complicated environments like drone racing, dynamic spectators may also interfere visual odometry. Less generic but computationally efficient methods are employed in some specific environments, for instance, using detection of known visual markers to determine position [29, 30]. However, these methods can not cope with other generic environments.

Concerning filtering, with white-Gaussian position measurement, Kalman filter and its variants are widely used. It is well-known that nonlinearities in the state update or observation equations can be handled by an Extended Kalman Filter (EKF) [11, 12, 22, 31] and that heavy nonlinearities are often handled better by an Unscented Kalman Filter (UKF) [30, 32, 33]. Also, there are factor graph based smoothing methods which can handle nonlinearity and allows multi-rate, asynchronous, and possibly delayed measurements, which have similar performance with an EKF [34, 35]. We hypothesize that when these measurements get sparser, and their noise distribution moves further away from the Gaussian distribution, it will be better to estimate the attitude, heading, and trajectory in general as an optimization problem that uses more data at a time. In particular, we want to optimize the trajectory and parameters such as sensor biases, given a specified time-window with the corresponding sensor measurements, control inputs, and knowledge of the aerodynamic model. Our approach will be explained below, starting with our dynamic quadrotor model.

## 3.3. QUADROTOR MODEL

### 3.3.1. DYNAMIC MODEL OF QUADROTOR

Before deriving the dynamic model for quadrotor, two reference frames are introduced. (Figure 3.4)

- Earth frame $E$. The origin of the local tangent earth frame is on the ground, the x-axis $x^E$ points to north, the y-axis $y^E$ points east and the z-axis $z^E$ points down.
- Body frame $B$. The origin of the body frame is at the center of mass. Its x-axis $x^B$ is in the symmetry plane of the drone and points forward. Its z-axis $z^B$ also lies in the symmetry plane and points downward. The y-axis $y^B$ is directed to the right, perpendicular to the symmetry plane.

The relative relation between two frames can be expressed by three successive rotations along three axes. In this paper, we use $z-y-x$ sequence to rotate one frame to the other. The corresponding angle of rotation is defined by $\phi_E^B$,$\theta_E^B$ and $\psi_E^B$ which are also called Euler angles. Given the Euler angles between the two frames, the rotation matrix between two frames can be expressed by

$$\Re_E^B = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ S_\phi S_\theta C_\psi - C_\phi S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & S_\phi C_\theta \\ C_\phi S_\theta C_\psi + S_\phi S_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi & C_\phi C_\theta \end{bmatrix} \tag{3.1}$$

in which $C_X$ and $S_X$ denote the cosine and sine of $X$ respectively The control of the quadrotor is often divided in to two loops which can be independently developed; namely a high level translation loop and a faster low-level attitude loop. For the attitude loop, the inputs of the system are the four rotor speeds and the output consists of the three Euler angles. For the translation loop, the inputs of system are three Euler angles and the output is position. Since quadrotor attitude control is a well developed topic, in this work we only derive the translational model and have used INDI from [36] as inner-loop.

According to Newton's laws of motion, the motion of quadrotor can be described as

$$m\dot{\mathbf{V}} = m\mathbf{g} + \mathbf{F} \tag{3.2}$$

where $m$ is mass of the drone, $\mathbf{g}$ is gravity vector and $\mathbf{F}$ is the specific force vector. The change in position can be described by the kinematic equation:

$$\dot{\mathbf{X}} = \mathbf{V} \tag{3.3}$$

In equation 3.2, the specific force $\mathbf{F}$ can be expressed in Body frame $B$ as

$$\mathbf{F}^B = \begin{bmatrix} F_x^B \\ F_y^B \\ F_z^B \end{bmatrix} \tag{3.4}$$

Gravity acting on the center of mass and expressed in Earth frame is

$$m\mathbf{g}^E = m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{3.5}$$

Combining all forces yields the equations of motion in inertial frame

$$\begin{gathered} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\ \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \Re_B^E \begin{bmatrix} a_x^B \\ a_y^B \\ a_z^B \end{bmatrix} \end{gathered} \tag{3.6}$$

where

$$\begin{bmatrix} a_x^B \\ a_y^B \\ a_z^B \end{bmatrix} = \begin{bmatrix} F_x^B \\ F_y^B \\ F_z^B \end{bmatrix} / m \tag{3.7}$$

In the system above, we have six states $\mathbf{x} = [x, y, z, v_x, v_y, v_z]^\mathrm{T}$ and four inputs $\mathbf{u} = [\phi, \theta, \psi, a_z^B]^\mathrm{T}$. In equation 3.6, the specific force is a nonlinear function of velocity, attitude, angular rates and other factors. It can be expressed as $\mathbf{F} = \mathbf{f}_a(\mathbf{V}, \phi, \theta, \psi, ...)$. This system is a multiple input multiple output nonlinear system.

### 3.3.2. IMU MISALIGNMENT

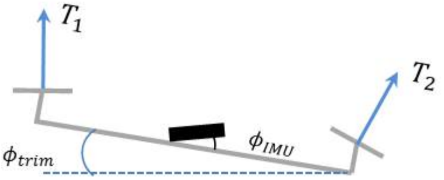


Figure 3.2: When a quadrotor hovers, usually the average attitude of the quadrotor and reading of the AHRS are not zero. This is caused by the misalignment of both the IMU and the rotors

Equation 3.6 reveals that rotation matrix $\Re_E^B$ is an essential part of the model. However, in the real world, many aspects can contribute to attitude estimation errors. A first reason is the misalignment of the IMU (See Figure 3.2). Assembly inaccuracy can cause the measurements of the IMU to differ from the real states in body frame. Rotor misalignment can also affect the performance of quadrotor. In an ideal quadrotor, the four rotors should be perpendicular to $x_B O y_B$ plane. In practice however, due to installation errors or deformation of rotors or axes, the thrust produced by the rotors is not perfectly perpendicular to the $x_B O y_B$ plane.

Both factors lead to non-zero required attitude during hover: $\phi_E^B \neq 0°$ and $\theta_E^B \neq 0°$. In order to model this misalignment error, we introduce a new frame. The IMU frame $I$ is an orthogonal frame whose three axis coincide with three axes of the accelerometers. The rotation between the IMU frame $I$ and the body frame $B$ can be described by Euler angles $\Phi_I^B = [\phi_I^B, \theta_I^B, \psi_I^B]^\mathrm{T}$. The rotation matrix between the IMU frame $I$ and the body frame $B$ is $\Re_I^B(\Phi_I^B)$. Since the IMU frame is physically attached to the body frame, we have the assumption

$$\begin{cases} \dot{\phi}_I^B(t) = 0 \\ \dot{\theta}_I^B(t) = 0 \\ \dot{\psi}_I^B(t) = 0 \end{cases} \tag{3.8}$$

### 3.3.3. AERODYNAMIC MODEL

There are many factors that can affect the quadrotor's aerodynamics. Some examples are the quadrotor's velocity $\mathbf{V}$, its angle of attack $\alpha$, the thrust $\mathbf{T}$, the rotor speed $\omega$, the angular velocity $\mathbf{q}$ and so on. Accurate and complete quadrotors models can be complicated and nonlinear [37–40]. Moreover, accurate modeling also requires many more parameters to be estimated and this leads to heavier computations. In the context of autonomous drone racing we opted for a faster approach using a minimal model that covers the most important aerodynamic effects which is a simplified linear drag model, hereby maximizing the yield for a given computational load. In particular, many drag factors—such as induced drag, translation drag and blade flapping drag—can be approximated as linear functions of body velocity $v_x^b$ and $v_y^b$ with the assumption that wind is still and the velocity is below $4m/s$ [41, 42]. This results in the following simple lumped parameter model:

$$\begin{cases} a_x^B = K_x v_x^B \\ a_y^B = K_y v_y^B \end{cases} \tag{3.9}$$

where

$$\begin{bmatrix} v_x^B \\ v_y^B \end{bmatrix} = \Re_E^B(3;3) \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{3.10}$$

$a_x^B, a_y^B$ are the acceleration caused by drag in the body frame. $K_x$, $K_y$ are first-order drag coefficients in body frame coordinates $B$ and have units $1/s$.

### 3.3.4. AHRS BIAS MODEL

When positioning information is available, the mainstream approach for estimating attitude is merging information from gyro, accelerometer and the positioning system. For instance, the classic 15 state Kalman filter uses accelerometer and gyro measurements to predict states along with GPS measurement updates. It can provide non-biased optimal attitude by estimating the gyro and accelerometer biases as states.

When no continuous external positioning information is available, like in our experiment, a compromise is to neglect kinematic accelerations in the attitude filter. In this case, the biases of accelerometers can not be estimated.

In the case of attitude determination with constant sensor biases and small angles, the Kalman gain in the Kalman filter typically converges to an almost constant value. To avoid the computational overhead of computing the Kalman gain, complementary filters can be used with very similar results. The structure of the complementary attitude determination filter implemented in this work can be found in Figure 3.3. In Figure 3.3, $\Omega_m = [p_m, q_m, r_m]$ are the gyro measurements. $\mathbf{a}_m = [a_x^m, a_y^m, a_z^m]$ contains the accelerometer measurements and

$$\mathbf{R}' = \begin{pmatrix} 1 & \tan\theta\sin\phi & \tan\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{pmatrix} \tag{3.11}$$

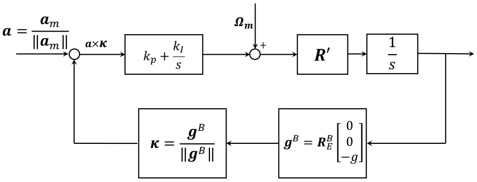

$\boldsymbol{a} = \frac{\boldsymbol{a}_m}{\|\boldsymbol{a}_m\|}$
$\boldsymbol{a}\times\boldsymbol{\kappa}$
$k_p + \frac{k_I}{s}$
$\boldsymbol{\Omega}_m$
+
$\boldsymbol{R}'$
$\frac{1}{s}$
$\boldsymbol{\kappa} = \frac{\boldsymbol{g}^B}{\|\boldsymbol{g}^B\|}$
$\boldsymbol{g}^B = \boldsymbol{R}_E^B \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$

Figure 3.3: Complementary filter for attitude determination.

Figure 3.3 shows that the gyroscopes are integrated and the accelerometer is used as feedback to determine attitude. The high frequency vibrations and centripetal forces which are measured by the accelerometers cancel out on the long term when no constant non-zero accelerations are present. On the long term, the resulting attitude estimation therefore converges to:

$$\begin{bmatrix} \hat{\phi}_a(t) \\ \hat{\theta}_a(t) \end{bmatrix} = \begin{bmatrix} \arctan \frac{-a_x^m}{-a_z^m} \\ \arctan \frac{-\cos\hat{\phi}_a(t) a_x^m}{-a_z^m} \end{bmatrix} \tag{3.12}$$

where $a_x^m$, $a_y^m$ and $a_z^m$ are measurements of the accelerometer in three axes.

The gyroscopes measure angular velocity in the three axes of the body frame.Because they are integrated, even small biases cause drift over time, and in this filter the gyro biases $\mathbf{b}_g = [b_p, b_q, b_r]^T$ are accounted for by the $k_I/s$ term in the filter.

Accelerometers unfortunately also suffer from biases, which is denoted by $\mathbf{b}_a = [b_{a_x}, b_{a_y}, b_{a_z}]^T$, for instance caused by temperature changes. Fortunately, the biases of the accelerometers only change slowly. Everything combined, the AHRS has an erroneous representation of where earth is, which is referred to as coordinate frame $E^{'}$ and is shown in Figure 3.4. The AHRS attitude is then defined as the rotation between $E^{'}$ and $I$ and is denoted as $\Phi_{E'}^{I} = [\phi_{E'}^{I}, \theta_{E'}^{I}, \psi_{E'}^{I}]^{\mathrm{T}}$. The corresponding rotation matrix is written as $\Re_{E'}^{I}(\Phi_{E'}^{I})$.

The rotation between the real earth $E$ and $E^{'}$ can be expressed by three Euler angles $\Phi_E^{E^{'}} = [\phi_E^{E^{'}}, \theta_E^{E^{'}}, \psi_E^{E^{'}}]^{\mathrm{T}}$. Based on the assumption that the AHRS error changes slowly, we can assume

$$\begin{cases} \dot{\phi}_E^{E^{'}}(t) \approx 0 \\ \dot{\theta}_E^{E^{'}}(t) \approx 0 \\ \dot{\psi}_E^{E^{'}}(t) \approx 0 \end{cases} \tag{3.13}$$

With this assumption, on the short term the rotation matrix $\Re_E^{E^{'}}(\Phi_E^{E^{'}})$ is a constant matrix.

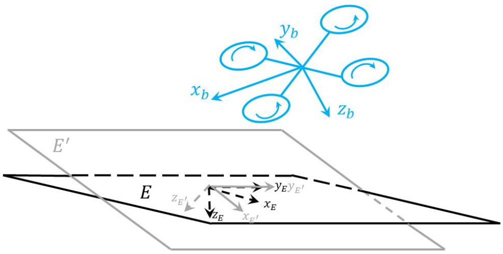


Figure 3.4: AHRS estimation errors can be represented by an erroneous Earth reference frame $E^{'}$.

Four reference frames have been introduced, namely $E$, $E^{'}$, $I$ and $B$. The rotation matrix $\Re_E^B$ in equation 3.6 can now be expressed as

$$\Re_E^B(\Phi_E^B) = \Re_I^B(\Phi_I^B)\Re_{E'}^{I}(\Phi_{E'}^{I})\Re_E^{E^{'}}(\Phi_E^{E^{'}}) \tag{3.14}$$

where $\Re_I^B(\Phi_I^B)$ and $\Re_E^{E'}(\Phi_E^{E'})$ are constant matrices and $\Re_{E'}^I(\Phi_{E'}^I)$ represents the attitude as determined by the AHRS.

### 3.3.5. FULL MODEL

Combining equation 3.6, equation 3.9 and equation 3.14 we obtain the full model as

$$\dot{\mathbf{x}} = \begin{cases} v_x \\ v_y \\ v_z \\ \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \Re_B^E \begin{bmatrix} 0 \\ 0 \\ a_z^B \end{bmatrix} + \begin{bmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & 0 \end{bmatrix} \Re_E^B \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \end{cases} \tag{3.15}$$

$$\begin{aligned} &\Re_E^B(\Phi_E^B) = \Re_I^B(\Phi_I^B)\Re_{E'}^I(\Phi_{E'}^I)\Re_E^{E'}(\Phi_E^{E'}) \\ &\Re_B^E = \Re_E^{B\,\mathrm{T}} \\ &a_z^B = a_z^m - b_{a_z} \end{aligned}$$

The model in equation 3.15 contains the following parameters, which are assumed to be constant over short periods of time:

$$\Theta = [K_x, K_y, b_{a_z}, \phi_E^{E'}, \theta_E^{E'}, \psi_E^{E'}, \phi_I^B, \theta_I^B, \psi_I^B]^\mathrm{T} \tag{3.16}$$

## 3.4. STATE ESTIMATION

To estimate the states of the model from the Quadrotor model section, two approaches are derived. As a benchmark, an Extended Kalman filter (EKF) is developed. Secondly, a novel gradient descent based optimization method to estimate the states is proposed.

### 3.4.1. VISION-BASED EXTENDED KALMAN FILTER

The attitude determination Kalman filter uses the inertial sensors as inputs to predict the states of the system, then uses different observations to revise the predictions. When the system is linear, observable and the noise is white Gaussian, then it can be mathematically proven that the Kalman filter provides the optimal solution. If the system is nonlinear, it can be linearized at every time step, which is referred to as the Extended Kalman filter. A classic 15-state EKF is implemented as found in Gross's work,[43] the difference being that we use vision measurements instead of GPS as positioning information. The following states are used:

$$\begin{aligned} \mathbf{X} &= [x, y, z]^\mathrm{T} \\ \mathbf{V} &= [v_x, v_y, v_z]^\mathrm{T} \\ \Phi &= [\phi, \theta, \psi]^\mathrm{T} \\ \mathbf{b}_a &= [b_{a_x}, b_{a_y}, b_{a_z}]^\mathrm{T} \\ \mathbf{b}_g &= [b_p, b_q, b_r]^\mathrm{T} \end{aligned} \tag{3.17}$$

with as inputs

$$\begin{aligned} \Omega_m &= [p^m, q^m, r^m]^{\mathrm{T}} \\ \mathbf{a}_m &= [a_x^m, a_y^m, a_z^m]^{\mathrm{T}} \end{aligned} \tag{3.18}$$

and as observation

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.19}$$

The process equation is

$$\begin{cases} \dot{\mathbf{X}} = \mathbf{V} \\ \dot{\mathbf{V}} = \mathbf{g} + \Re_B^E(\mathbf{a}_m + \mathbf{b}_a) \\ \dot{\Phi} = \Re'(\Omega_m + \mathbf{b}_g) \\ \dot{\mathbf{b}}_a = \mathbf{0} \\ \dot{\mathbf{b}}_g = \mathbf{0} \end{cases} \tag{3.20}$$

This forms a standard nonlinear system expression

$$\dot{\mathbf{x}}' = \mathbf{f}(\mathbf{x}', \mathbf{u}) \tag{3.21}$$

where $\mathbf{x}' = [\mathbf{X}, \mathbf{V}, \Phi, \mathbf{b}_a, \mathbf{b}_g]^{\mathrm{T}}$ and

$$\mathbf{f}(\mathbf{x}', \mathbf{u}) = \begin{bmatrix} \mathbf{V} \\ \mathbf{g} + \Re_E^B(\mathbf{a}_m + \mathbf{b}_a) \\ \Re'(\Omega_m + \mathbf{b}_g) \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{3.22}$$

The Extended Kalman filter follows 5 steps:
(1) Predict the states based on equation 3.20

$$\hat{\mathbf{X}}_{k|k-1} = \hat{\mathbf{X}}_{k-1} + \mathbf{f}(\hat{\mathbf{X}}_{k-1}, \mathbf{u}_{k-1})\mathrm{T} \tag{3.23}$$

where T is sampling time.
(2) Linearize and discretize the system

$$\begin{aligned} \mathbf{F}_{k-1} &= \frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_{k-1}} \\ \Phi_{k|k-1} &\approx \mathbf{I} + \mathbf{F}_{k-1}\mathrm{T} \\ \mathbf{H}_k &= \frac{\partial}{\partial \mathbf{x}} \mathbf{h}(\mathbf{x}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_{k-1}} \end{aligned} \tag{3.24}$$

(3) Propagate the covariance matrix $\mathbf{P}_{k|k-1}$

$$\mathbf{P}_{k|k-1} = \Phi_{k|k-1}\mathbf{P}_{k-1}\Phi^{\mathrm{T}}_{k|k-1} + \mathbf{Q}_{k-1} \tag{3.25}$$

where $\mathbf{Q}_{k-1}$ is system noise covariance matrix.

(4) Calculate the Kalman gain and update the prediction.

$$\begin{aligned}
\delta\hat{\mathbf{X}}_k &= \mathbf{K}_k\left\{\mathbf{Z}_k - \mathbf{h}[\hat{\mathbf{X}}_{k|k-1}, k]\right\} \\
\mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}[\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k]^{-1} \\
\hat{\mathbf{X}}_k &= \hat{\mathbf{X}}_{k|k-1} + \delta\hat{\mathbf{X}}_k
\end{aligned} \tag{3.26}$$

where $\mathbf{R}_k$ is sensor noise covariance matrix.

(5) Update the covariance matrix of the state estimation error

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k/k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^{\mathrm{T}} + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^{\mathrm{T}} \tag{3.27}$$

### 3.4.2. VISION-BASED GRADIENT DESCENT METHOD

According to the gate detection algorithm we used in IROS 2016 autonomous drone race, the vision-based position used as observation in the Kalman filter (Equation 3.19) has very non-Gausian noise, which can significantly affect the estimation accuracy of Kalman filters. The vision measurement model will be discussed later. Therefore the state prediction is rewritten as a parameter optimization problem in the form of a trajectory matching problem.

Unlike the Kalman filter which estimates continuously varying states like pitch and roll for any moment in time, the proposed gradient descent using the model from equation 3.15 in essence estimates corrections on top of attitude estimates provided by an external complementary attitude filter.

Since most model parameters like drag and AHRS error are integrated twice to arrive at position, observing the trajectory over a period of time allows for extremely fine observations of these parameters. For instance, a sub degree attitude error is hard to identify in noisy raw accelerometer measurements. However integrating the consequence of this small angle error, which causes a percentage of gravity to be erroneously double-integrated in the lateral position after several seconds, becomes very easily observable.

The observed trajectory is obtained from the vision pipeline and expressed as a list of $n$ noisy measurements. The predicted trajectory is based on integrating the model presented in equation 3.15 using attitude from the AHRS and given a set of model parameters $\hat{\Theta}$. The resulting trajectory becomes:

$$\mathbf{F}(\Theta) = \int_0^t \mathbf{f}(\Theta, \mathbf{u}(t), t)\,dt = \begin{bmatrix} \hat{x}(\Theta, \mathbf{u}(t), t) \\ \hat{y}(\Theta, \mathbf{u}(t), t) \\ \hat{z}(\Theta, \mathbf{u}(t), t) \\ \hat{v}_x(\Theta, \mathbf{u}(t), t) \\ \hat{v}_y(\Theta, \mathbf{u}(t), t) \\ \hat{v}_z(\Theta, \mathbf{u}(t), t) \end{bmatrix} \tag{3.28}$$

The error between the predicted integrated trajectory and the vision measurements is found as

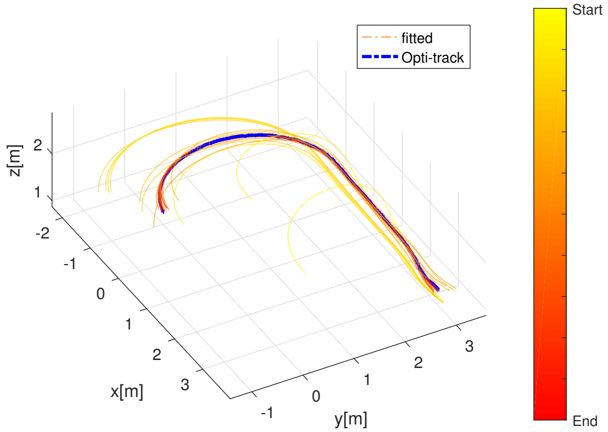


Figure 3.5: A gradient descent method optimizes a set of parameters Θ to best fit a predicted trajectory through a measured trajectory (blue). During the fitting phase, the gradient descent method converges to the ground-truth trajectory.

$$J(\Theta) = \sum_{i=i}^{n} \left\| \begin{bmatrix} \hat{x}(\Theta, \mathbf{u}(t_i), t_i) \\ \hat{y}(\Theta, \mathbf{u}(t_i), t_i) \\ \hat{z}(\Theta, \mathbf{u}(t_i), t_i) \end{bmatrix} - \begin{bmatrix} x_i^m \\ y_i^m \\ z_i^m \end{bmatrix} \right\| \tag{3.29}$$

where $x_i^m, y_i^m, z_i^m$ are position measurements obtained from onboard computer vision. Now the state estimation has become a nonlinear parameter optimization problem that finds a set of optimal parameters $\Theta^\star$ to minimize the value of $J(\Theta)$ which can be expressed as

$$\begin{aligned} &\min_{\Theta} J(\Theta) \\ s.t. \quad &\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \end{aligned} \tag{3.30}$$

To solve the problem formulated by equation 3.30, we can apply many types of nonlinear optimization methods to find the optimal parameters $\Theta^\star$. In this paper, we propose the gradient descent method, which is iteratively searching for optimal values in negative gradient direction until it finds the minimum point:

$$\Theta_{k+1} = \Theta_k + \alpha \nabla J(\Theta_k) \tag{3.31}$$

where $\alpha$ is learning rate and

$$\nabla J(\Theta_k) = \left[ \frac{\partial}{\partial \Theta_1} J(\Theta) \quad \cdots \quad \frac{\partial}{\partial \Theta_n} J(\Theta) \right]^{\mathrm{T}} |_{\Theta=\Theta_k} \tag{3.32}$$

is the gradient of $J(\Theta)$.

Figure 3.5 shows an example of the gradient descent approach. The propagation in time of the model from equation 3.15 for various parameters Θ is compared to the ground-truth measured by a passive external positioning system. The gradient descent starts with an initial guess of $\Theta_0$, and gradually gets the predicted trajectory closer to the real trajectory until an optimal set $\Theta^\star$ is found. In this example, we directly use Opti-track data as measurements which better illustrate how the predicted trajectories converge to the ground-truth trajectory (measured by Opti-track).

## 3.5. EXPERIMENT SETUP AND RESULT

### 3.5.1. EXPERIMENT SETUP

In order to study the performance of state estimation methods, a hippodrome shaped track is used with end circles with radius of 1.5$m$ and straights of 3$m$ as shown in Figure 3.6. Onboard flight data is recorded while flying without computer vision but based on optitrack position. The data is then analyzed in MATLAB. A Bebop 1 (Figure 3.7) from Parrot is used as experiment platform. It is equipped with three gyros, three accelerometers, one sonar, one barometer, a front camera and a bottom camera. Only the front camera and IMU are used and the original stock flight-code in the drone is replaced by open-source software from the Paparazzi-UAV project [44]. The AHRS runs on-board and consists of the complementary filter discussed in previous section. The flight time of the test runs is about 100$s$ and the average flight velocity is about 1.8$m/s$, resulting in about 15 circles of the hippodrome. An overview of data gathered is presented in Table 3.1.

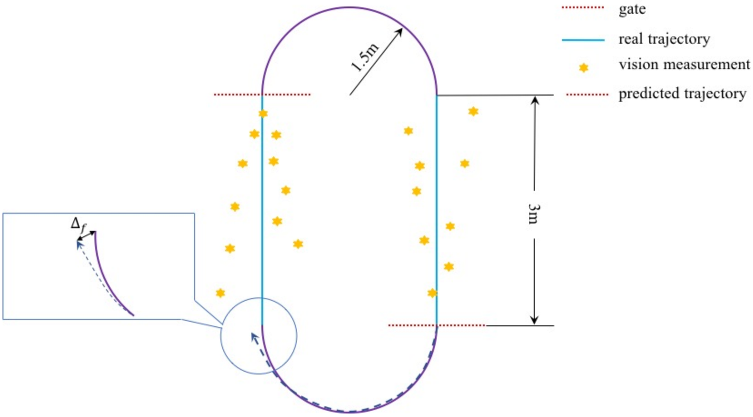


Figure 3.6: The top view of the experiment track.

During the IROS 2016 autonomous drone race, we used the bebop 1 onboard camera to detect the gates and provide the position measurements for navigation. In this work, however, noisy vision measurements are generated simulating on-board vision-based gate detections with various levels of accuracy. Along the straight part trajectory, $n$ random points $\mathbf{P}_i$ are randomly sampled ($15 < n < 20, i \in [1, n]$). For each sampled point $\mathbf{P}_i$, we calculate the distance between $\mathbf{P}_i$ and the gate which is denoted by $\hat{x}_i - x_g$. Then, the noise $\Delta\mathbf{P}_i$ is generated depending on the distance to the gate that $\Delta\mathbf{P}_i$ is larger when the gate is further away. Finally, $\Delta\mathbf{P}_i$ is added to $\mathbf{P}_i$ to get the simulated measurements $\mathbf{P}_i^v$.

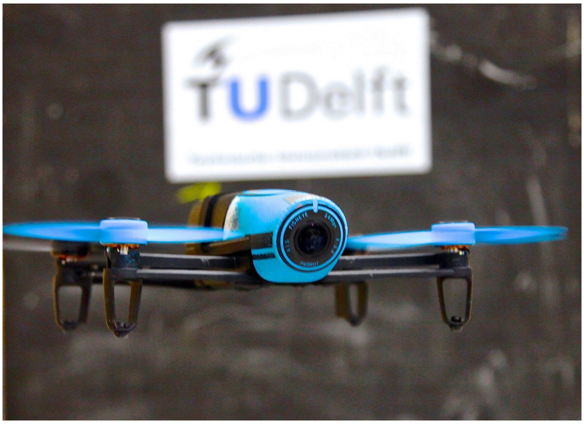


Figure 3.7: The Parrot Bebop 1 hardware is used as experiment platform. All flight code is replaced with open-source Paparazzi-UAV flight code.

Table 3.1: Data gathered during the experiment

| Parameter | Symbol | Frequency (Hz) | Source |
|---|---|---|---|
| acceleration | $\hat{\mathbf{a}}^m$ | 512 | IMU |
| angular velocity | $\hat{\mathbf{p}}^m$ | 512 | IMU |
| attitude | $\hat{\Phi}^m$ | 512 | AHRS |
| position | $\hat{\mathbf{x}}^m$ | 120 | Opti-track |
| velocity | $\hat{\mathbf{v}}^m$ | 120 | Opti-track |
| altitude | $z^m$ | 512 | sonar |

This process can be described by equation 4.34

$$\begin{aligned}\mathbf{P}_i^v &= \mathbf{P}_i^m + \Delta\mathbf{P}_i \\ \Delta\mathbf{P}_i &\sim \mathbf{N}(\mathbf{0}, \mathbf{S}_i) \\ \mathbf{S}_i &= \begin{bmatrix} \sigma_i^2 & 0 & 0 \\ 0 & \sigma_i^2 & 0 \\ 0 & 0 & \sigma_i^2 \end{bmatrix} \\ \sigma_i &= 0.1(\hat{x}_i - x_g)\end{aligned} \tag{3.33}$$

The test flights consist of two distinct phases which are shown in Figure 3.6.

- During the *straight part* (blue line), the gates are in the field of view of the quadrotor and vision-based position measurements are available. The vision-based EKF can run both prediction and update loops. The vision-based gradient descent method searches for parameters $\Theta$ that make the prediction best fit the noisy measurements.
- During the arc (purple line), no position measurements are available but an open-loop coordinated turn is performed. The vision-based EKF can only rely on model

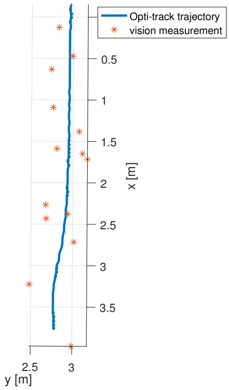


Figure 3.8: Based on the vision measurement model (equation 4.34), simulated vision measurement points (red) are generated around the real trajectory (blue). During the autonomous drone race, only the visual measurement points are available.

prediction and the gradient descent method uses the last estimated parameters and on-board inertial data to propagate the states of the quadrotor. This phase must be limited in time as the open-loop integration is diverging as can be seen in Figure 3.9.

The test track is designed to resemble an autonomous drone race track, where it is not possible to keep gates in sight at all times. When using fast gate detection as sole means of position information, some maneuvers need to be performed open-loop. But even when gates are in-sight, better model prediction allows the estimation of more accurate trajectories through the noisy visual data. Therefore, as a performance index we selected the prediction error $\Delta_f$ at the final point of the open-loop arc to evaluate the performance of both algorithms.

$$\Delta_f = \left\| \begin{bmatrix} x_f \\ y_f \end{bmatrix} - \begin{bmatrix} \hat{x}_f \\ \hat{y}_f \end{bmatrix} \right\| \tag{3.34}$$

where $x_f$ and $y_f$, which are from Opti-track, form the ground truth of the end point of the arc, while $\hat{x}_f$ and $\hat{y}_f$ are the filter prediction of the end point.

### 3.5.2. ANALYSIS OF VISION-BASED GRADIENT DESCENT METHOD (VGD)

In this section, we use the on-board flight data and generated vision measurements to analyze the Vision-based gradient descent method (VGD) using a MATLAB implementation of gradient descent, FMINCON.

The performance of the gradient descent method is affected by the size of the training data. It is important to investigate how the size of the dataset used to search for $\Theta^\star$ affects the estimation performance. We use the notation $\gamma$ ($1 \leqslant \gamma \leqslant 5$) to represent the size of the history used by FMINCON. In other words, $\gamma$ is the number of straight lines whose corresponding vision measurement is used by FMINCON. Too short $\gamma$ will contain very few visual measurements and the approach is at risk of over-fitting the gate

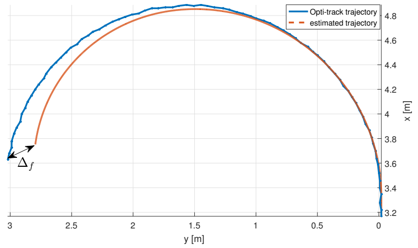


Figure 3.9: When vision measurements are not available, the quadrotor can only rely on model predictions based on model information and inertial data. This prediction will diverge in time. The better the model prediction is, the smaller the end point prediction error $\Delta_f$ becomes.

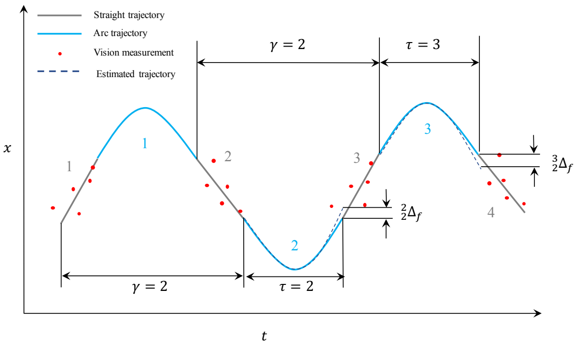


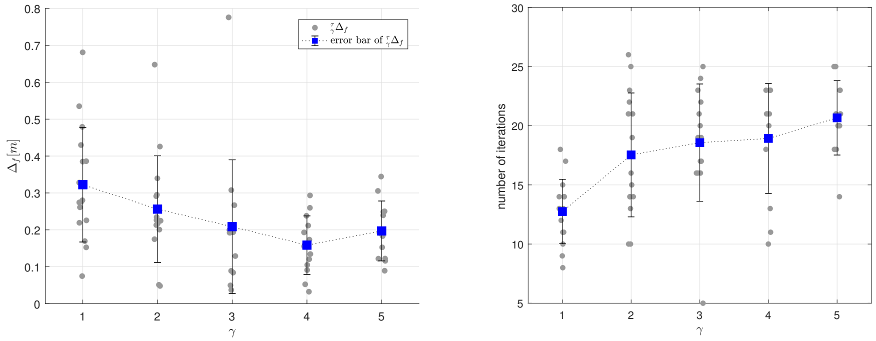
Figure 3.10: Example test flight data showing the $x$ position in function of time and illustrating the prediction strategy when $\gamma = 2$. First, the data of straight lines 1 and 2 is used to estimate $\Theta^\star$. Then the identified model parameters are used to predict the second turn. Finally, the final point error after the second arc $^{2}_{2}\Delta_f$ is calculated. Here, subscript 2 means the data from 2 straight lines is used and superscript 2 means second arc's trajectory prediction is used. This procedure is repeated by using data of straight lines 2 and 3 and predicting the trajectory of third arc and so forth.

detection noise. Too long $\gamma$ will violate the constant parameter constraint like for instance equation 3.13. Figure 3.10 shows an example where $\gamma = 2$. For each step, we use an array of flight data and vision measurements of size $\gamma$ in FMINCON to search for $\Theta^\star$. Then, $\Theta^\star$ is used to estimate the trajectory of next arc, which is given by id $\tau$ $(1 \leqslant \tau \leqslant 15)$. Finally the final point error $^{\tau}_{\gamma}\Delta_f$ can be calculated using equation 3.34:

$$^{\tau}_{\gamma}\Delta_f = \left\| \begin{bmatrix} x_f \\ y_f \end{bmatrix} - \begin{bmatrix} \hat{x}_f \\ \hat{y}_f \end{bmatrix} \right\| \tag{3.35}$$

The stopping criteria used in the FMINCON optimization is:

$$\frac{\| J(\Theta_k) - J(\Theta_{k-1}) \|}{\| J(\Theta_k) \|} \leqslant 10^{-4} \tag{3.36}$$



(a) Final point error $^\tau_\gamma\Delta_f$ in function of $\gamma$ for various parts of the run $\tau$

(b) Number of FMINCON iterations based on stopping criteria (equation 3.36) in function of $\gamma$ for various parts of the run $\tau$

Figure 3.11: Influence of the history length $\gamma$ on the prediction accuracy $^\tau_\gamma\Delta_f$ and required number of iterations.

With different combinations of $\tau$ and $\gamma$, a set of 70 $^\tau_\gamma\Delta_f$ is gathered. The prediction accuracy results, $^\tau_\gamma\Delta_f$, and the number of iterations based on the stopping criteria from equation 3.36 are shown in Figure 3.11.

Figure 3.11 (a) shows the prediction accuracy $^\tau_\gamma\Delta_f$ as a function of the history length $\gamma$. Each gray dot represents an individual arc estimation $\tau$ on another part of the data while the blue dots give the average for a given $\gamma$. Similarly, in Figure 3.11 (b) the required number of iterations based on the stopping criteria is shown. The figures show that the prediction error $^\tau_\gamma\Delta_f$ keeps decreasing up to $\gamma = 4$. This means that fitting more than one straight part helps improving the accuracy of state estimation. Figure 3.11 (b) shows that the average number of iterations is about 19 and the maximum is only 25, which means this vision-based gradient descent method quickly converges and is not very computationally expensive.

### 3.5.3. COMPARISON BETWEEN VISION-BASED EKF (VEKF), VISION-BASED GRADIENT DESCENT METHOD (VGD) AND VISION-BASED GRADIENT DESCENT METHOD WITH KINEMATIC MODEL(VGD-KINEMATIC)

In this section, in order to show the different performance of the gradient descent between the kinematic model and model from equation 3.15, we introduce a new method called Vision-based gradient descent method with kinematic model (VGD-kinematic). This method has the same principle as VGD except that it is using a kinematic model 3.37 as prediction model.

$$\begin{cases} \dot{\mathbf{X}} = \mathbf{V} \\ \dot{\mathbf{V}} = \mathbf{g} + \Re_B^E(\mathbf{a}_m + \mathbf{b}_a) \\ \dot{\Phi} = \Re'(\Omega_m + \mathbf{b}_g) \end{cases} \tag{3.37}$$

In this case, the parameters to be estimated are the bias of accelerometers and gyros, which can be written as

$$\Theta = [b_{a_x}, b_{a_y}, b_{a_z}, b_p, b_q, b_r]^{\mathrm{T}} \tag{3.38}$$

To compare the performance of the of three methods, all three methods are tested using the same on-board data and the same generated vision measurements. In both VGD and VGD-kinematic, $\gamma$ was set to 3, which means that the flight data of the last 3 straights is used in the estimation of $\Theta^{\star}$. Note that during the first two arcs of the flight, there is not yet enough flight data, and $\gamma$ will be smaller than 3.

The resulting full flight is shown in Figure 3.13. In Figure 3.13, the orange dots are the generated vision measurements from the straight parts of the track. The magenta curve is the estimation result of the VEKF. In the VEKF, $\mathbf{R} = diag([2.5^2, 2.5^2, 2.5^2]), \mathbf{Q} = diag([(2e-6, 2e-6, 5e-6, e-5, 5e-6, 3e-5, 3e-8, 3e-9, 3e-9, 0, 0, 0, 0, 0, 0])$ and $\mathbf{P}_0 = 10 \times I_{15\times 15}$. The blue curve is the estimation result of the VGD and the red curve is the result of VGD-kinematic. To test the sensitivity of the VGD and the VGD-kinematic algorithm, the initial parameters $\Theta_0$ are selected randomly within some ranges which can be found in Table 3.2. It can be seen that while the VEKF clearly converges to the measurements. The long prediction horizon combined with few and noisy measurement updates challenges the filter to its limit. On the other hand, the VGD managed to find parameters that fit the model very well through the noisy measurements and is not sensitive to the initial parameters. Even large measurement noise does not affect the prediction too much as the dynamics of the quadrotor can not explain them.

Table 3.2: The range of $\Theta_0$ in VGD and VGD-kinematic

| $\Theta_0$ | Range | $\Theta_0$ | Range |
|---|---|---|---|
| $K_*^0$ | $[-1, 0]$ | $\phi_*^{*0}$ | $[-3°, 3°]$ |
| $\mathbf{b}_a^0$ | $[-1m/s^2, 1m/s^2]$ | $\theta_*^{*0}$ | $[-3°, 3°]$ |
| $\mathbf{b}_g^0$ | $[-3°/s, 3°/s]$ | $\psi_*^{*0}$ | $[-3°, 3°]$ |

The final point prediction error ${}_{\gamma}^{\tau}\Delta_f$ after each turn of the three algorithms is shown in Figure 3.12. The VEKF requires several laps ($3^{th}$ arc, or about $20s$ of flight) to converge to sub-meter prediction accuracy. During the rest of the flight, the EKF can predict the 180 degree turns with a final point prediction error of around $0.5m$. The VGD-kinematic uses the derived kinematic model as prediction model and utilizes multiple vision measurements for parameter estimation. It has similar performance when compared with the VEKF. Overall, the VGD, which uses the same measurements as the VGD-kinematic but performs a bias and aerodynamics model estimation, is shown to find the best estimates of all parameters. It even find good model parameters for the first arc, using only 1 straight line's flight data. During the whole flight, ${}_{\gamma}^{\tau}\Delta_f$ of the VGD is kept around $0.2m$.

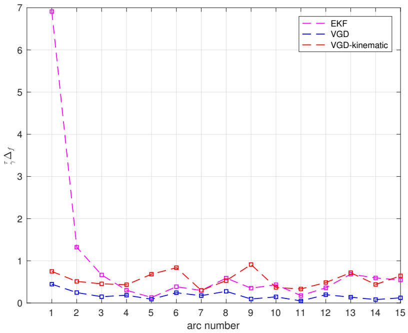
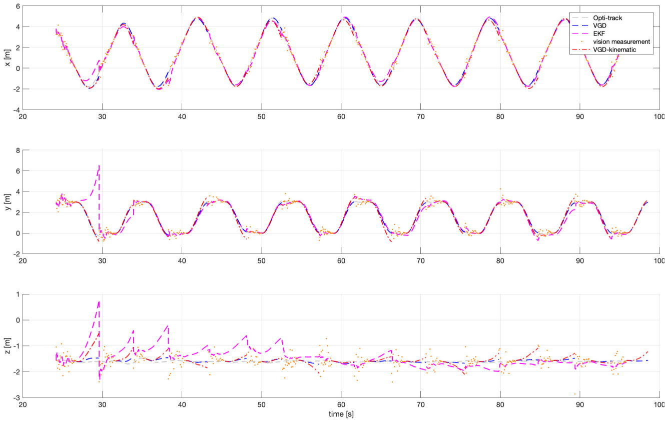


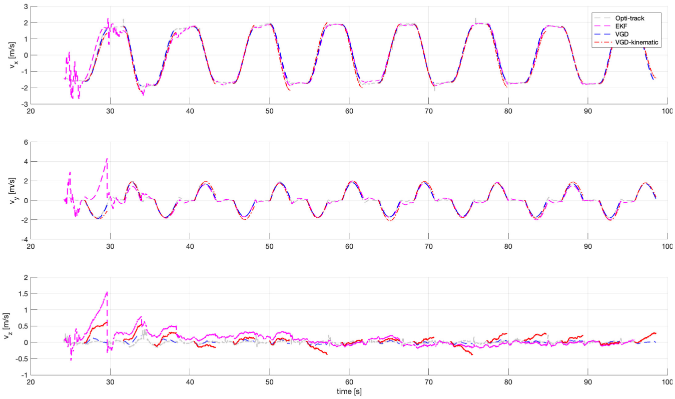
Figure 3.12: The final point error $^{\tau}_{\gamma}\Delta_f$ when using the VEKF, VGD and VGD-kinematic. The VGD has the most stable performance and least $^{\tau}_{\gamma}\Delta_f$ compared to the EKF and VGD-kinematic

## 3.6. CONCLUSION

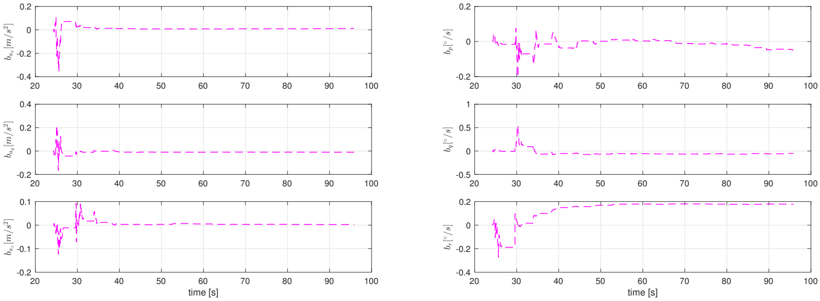
Accurate state and parameter estimation is essential for quadrotor control, especially when they perform aggressive maneuver. However, in the environment where only sparse and noisy position measurements are available, a classic Kalman filter can struggle to provide accurate state and model parameter estimation results. In this paper, we presented a novel method that only uses sparse vision measurements to estimate the AHRS error and select aerodynamic parameters of the quadrotor using a gradient descent method. The experiment result shows that our vision-based gradient descent method could increase the accuracy of state estimation when compared to a classic Kalman filter in environments where only sparse noisy position measurements are available.



(a) Position estimation



(b) Velocity estimation



(c) Estimation of accelerometer bias (d) Estimation of gyro bias

Figure 3.13: Comparison of the position and velocity estimation results of the EKF and the FMINCON based gradient descent method using inertial sensors and discrete low frequency noise vision based position measurements from gate detections.

## REFERENCES

[1] H. Moon, Y. Sun, J. Baltes, and S. J. Kim, *The IROS 2016 Competitions [Competitions],* IEEE Robotics & Automation Magazine **24**, 20 (2017).

[2] J. N. Gross, Y. Gu, M. B. Rhudy, S. Gururajan, and M. R. Napolitano, *Flight-test evaluation of sensor fusion algorithms for attitude estimation,* IEEE Transactions on Aerospace and Electronic Systems **48**, 2128 (2012).

[3] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, *Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment,* in *AIAA Guidance, Navigation and Control Conference and Exhibit* (American Institute of Aeronautics and Astronautics, Reston, Virigina, 2007).

[4] Haomiao Huang, G. Hoffmann, S. Waslander, and C. Tomlin, *Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering,* in *2009 IEEE International Conference on Robotics and Automation* (IEEE, 2009) pp. 3277–3282.

[5] M. Bangura and R. Mahony, *Nonlinear Dynamic Modeling for High Performance Control of a Quadrotor,* in *Proceedings Australasian Conference on Robotics and Automation* (2012).

[6] J. Svacha, K. Mohta, and V. Kumar, *Improving quadrotor trajectory tracking by compensating for aerodynamic effects,* in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)* (IEEE, 2017) pp. 860–866.

[7] J.-M. Kai, G. Allibert, M.-D. Hua, and T. Hamel, *Nonlinear feedback control of Quadrotors exploiting First-Order Drag Effects,* in *IFAC World Congress* (Toulouse, France, 2017) in IFAC world congress, Toulouse 2017.

[8] P.-J. Bristeau, F. Callou, D. Vissiere, and N. Petit, *The navigation and control technology inside the ar. drone micro uav,* IFAC Proceedings Volumes **44**, 1477 (2011).

[9] C.-S. Y. C.-S. Yoo and I.-K. A. I.-K. Ahn, *Low cost gps/ins sensor fusion system for uav navigation,* in *Digital Avionics Systems Conference, 2003. DASC'03. The 22nd*, Vol. 2 (IEEE, 2003) pp. 8–A.

[10] A. K. Brown, *Gps/ins uses low-cost mems imu,* IEEE Aerospace and Electronic Systems Magazine **20**, 3 (2005).

[11] E. Shi, *An improved real-time adaptive kalman filter for low-cost integrated gps/ins navigation,* in *Measurement, Information and Control (MIC), 2012 International Conference on*, Vol. 2 (IEEE, 2012) pp. 1093–1098.

[12] H. Lopes, E. v. Kampen, and Q. Chu, *Attitude determination of highly dynamic fixed-wing uavs with gps/mems-ahrs integration,* in *AIAA Guidance, Navigation, and Control Conference* (2012) p. 4460.

[13] D. B. Kingston and R. W. Beard, *Real-time attitude and position estimation for small uavs using low-cost sensors,* in *AIAA 3rd unmanned unlimited technical conference, Workshop and exhibit* (sn, 2004) pp. 2004–6488.

[14] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors,* in *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (IEEE, 2011) pp. 2520–2525.

[15] A. Bry, C. Richter, A. Bachrach, and N. Roy, *Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments,* The International Journal of Robotics Research **34**, 969 (2015).

[16] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. Grixa, F. Ruess, M. Suppa, and D. Burschka, *Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue,* IEEE Robotics and Automation Magazine **19**, 46 (2012).

[17] S. Hrabar, *An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance,* Journal of Field Robotics **29**, 215 (2012), arXiv:10.1.1.91.5767 .

[18] R. G. Valenti, I. Dryanovski, C. Jaramillo, D. P. Ström, and J. Xiao, *Autonomous quadrotor flight using onboard rgb-d visual odometry,* in *Robotics and Automation (ICRA), 2014 IEEE International Conference on* (IEEE, 2014) pp. 5233–5238.

[19] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, *Estimation, planning, and mapping for autonomous flight using an RGB-D camera in GPS-denied environments,* The International Journal of Robotics Research **31**, 1320 (2012).

[20] C. Sampedro, H. Bavle, A. Rodríguez-Ramos, A. Carrio, R. A. S. Fernández, J. L. Sanchez-Lopez, and P. Campoy, *A fully-autonomous aerial robotic solution for the 2016 international micro air vehicle competition,* in *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on* (IEEE, 2017) pp. 989–998.

[21] C. Mostegel, A. Wendel, and H. Bischof, *Active monocular localization: Towards autonomous monocular exploration for multirotor MAVs,* in *2014 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2014) pp. 3848–3855.

[22] S. Huh, D. H. Shim, and J. Kim, *Integrated navigation system using camera and gimbaled laser scanner for indoor and outdoor autonomous flight of uavs,* in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on* (IEEE, 2013) pp. 3158–3163.

[23] D. Nistér, O. Naroditsky, and J. Bergen, *Visual odometry,* in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, Vol. 1 (Ieee, 2004) pp. I–I.

[24] F. Andert, N. Ammann, J. Puschel, and J. Dittrich, *On the safe navigation problem for unmanned aircraft: Visual odometry and alignment optimizations for uav positioning,* in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on* (IEEE, 2014) pp. 734–743.

[25] R. Strydom, S. Thurrowgood, and M. V. Srinivasan, *Visual odometry: autonomous uav navigation using optic flow and stereo,* in *Australasian Conference on Robotics and Automation (ACRA)* (Australian Robotics and Automation Association, 2014) pp. 1–10.

[26] I. F. Mondragón, M. A. Olivares-Méndez, P. Campoy, C. Martínez, L. Mejias, I. F. Mondragón, M. A. Olivares-Méndez, P. Campoy, ·. C. Martínez, and L. Mejias, *Unmanned aerial vehicles UAVs attitude, height, motion estimation and control using visual systems,* Auton Robot **29**, 17 (2010).

[27] L. Rodolfo García Carrillo, A. Enrique Dzul López, R. Lozano, C. Pégard, L. R. García Carrillo, R. Lozano, and A. E. Dzul López, *Combining Stereo Vision and Inertial Navigation System for a Quad-Rotor UAV,* J Intell Robot Syst **65**, 373 (2012).

[28] J. Martínez-Carranza and A. Calway, *Efficient visual odometry using a structure-driven temporal map,* in *Robotics and Automation (ICRA), 2012 IEEE International Conference on* (IEEE, 2012) pp. 5210–5215.

[29] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, *Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision,* in *Robotics and Automation (ICRA), 2017 IEEE International Conference on* (IEEE, 2017) pp. 5774–5781.

[30] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, *Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu,* IEEE Robotics and Automation Letters **2**, 404 (2017).

[31] J. S. Jang and D. Liccardo, *Small uav automation using mems,* IEEE Aerospace and Electronic Systems Magazine **22**, 30 (2007).

[32] J. L. Crassidis, *Sigma-point kalman filtering for integrated gps and inertial navigation,* IEEE Transactions on Aerospace and Electronic Systems **42**, 750 (2006).

[33] P. Zhang, J. Gu, E. E. Milios, and P. Huynh, *Navigation with imu/gps/digital compass with unscented kalman filter,* in *Mechatronics and Automation, 2005 IEEE International Conference*, Vol. 3 (IEEE, 2005) pp. 1497–1502.

[34] S. Lange, N. Sünderhauf, and P. Protzel, *Incremental smoothing vs. filtering for sensor fusion on an indoor uav,* in *Robotics and Automation (ICRA), 2013 IEEE International Conference on* (IEEE, 2013) pp. 1773–1778.

[35] V. Indelman, S. Williams, M. Kaess, and F. Dellaert, *Factor graph based incremental smoothing in inertial navigation systems,* in *Information Fusion (FUSION), 2012 15th International Conference on* (IEEE, 2012) pp. 2154–2161.

[36] E. J. Smeur, Q. Chu, and G. C. de Croon, *Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles,* Journal of Guidance, Control, and Dynamics (2015).

[37] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, *Quadrotor helicopter flight dynamics and control: Theory and experiment,* in *Proc. of the AIAA Guidance, Navigation, and Control Conference*, Vol. 2 (2007) p. 4.

[38] M. Bangura, R. Mahony, *et al.*, *Nonlinear dynamic modeling for high performance control of a quadrotor,* in *Australasian conference on robotics and automation* (2012) pp. 1–10.

[39] S. Sun, C. C. de Visser, and Q. Chu, *Quadrotor gray-box model identification from high-speed flight data,* Journal of Aircraft **56**, 645 (2019).

[40] S. Sun and C. de Visser, *Aerodynamic model identification of a quadrotor subjected to rotor failures in the high-speed flight regime,* IEEE Robotics and Automation Letters **4**, 3868 (2019).

[41] J. Svacha, K. Mohta, and V. Kumar, *Improving quadrotor trajectory tracking by compensating for aerodynamic effects,* in *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on* (IEEE, 2017) pp. 860–866.

[42] M. Faessler, A. Franchi, and D. Scaramuzza, *Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories,* IEEE Robotics and Automation Letters **3**, 620 (2017).

[43] J. N. Gross, Y. Gu, M. B. Rhudy, S. Gururajan, and M. R. Napolitano, *Flight-test evaluation of sensor fusion algorithms for attitude estimation,* IEEE Transactions on Aerospace and Electronic Systems **48**, 2128 (2012).

[44] B. Gati, *Open source autopilot for academic research-the paparazzi system,* in *American Control Conference (ACC), 2013*, IEEE (IEEE, Washington, DC, USA, 2013) pp. 1478–1481.

# 4

# VISUAL MODEL-PREDICTIVE LOCALIZATION FOR COMPUTATIONALLY EFFICIENT AUTONOMOUS RACING OF A 72-GRAM DRONE

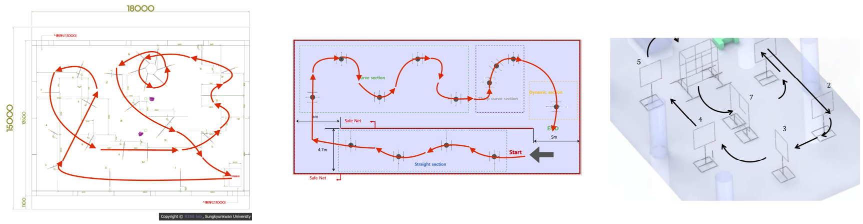
---

This chapter is based on the following article:
**Li S**, van der Horst E, Duernay P, De Wagter C, de Croon GC. Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone. Journal of Field Robotics. 2020 Jun 1.

## 4.1. INTRODUCTION

Drones, especially quadrotors, are transformed by enthusiasts in spectacular racing platforms. After years of development, drone racing has become a major e-sports, where the racers fly their drones in a preset course at high speed. It was reported that an experienced first person view (FPV) racer can achieve speeds up to 190$km/h$ when sufficient space is available. The quadrotor itself uses an inertial measurement unit (IMU) to determine its attitude and rotation rates, allowing it to execute the human's steering commands. The human mostly looks at the images and provides the appropriate steering commands to fly through the track as fast as possible. The advance in research areas such as computer vision, artificial intelligence and control raises the question: would drones not be able to fly faster than human pilots if they flew completely by themselves? Until now, this is an open question. In 2016, the world's first autonomous drone race was held at IROS 2016 [1], which became an annual event trying to answer this question (Figure 4.1).

We focus on developing computationally efficient algorithms and extremely light weight autonomous racing drones that have the same or even better performance than currently existing larger drones. We believe that these drones may be able to fly faster, as the gates will be relatively larger for them. Moreover, a cheap, light-weight solution to drone racing would allow many people to use autonomous drones for training their racing skills. When the autonomous racing drone becomes small enough, people may even practice with such drones in their own home.



(a) IROS 2016 drone race track (b) IROS 2017 drone race track (c) IROS 2018 drone race track

Figure 4.1: The IROS autonomous drone race track over the years 2016 - 2018 (a-c). The rules have always been the same. Flight is to be fully autonomous, so there can be no human intervention. The drone that passes through most subsequent gates in the track wins the race. When the number of passed gates is the same, or the track is fully completed, the fastest drone wins the race.

Autonomous drone racing is indebted to earlier work on agile flight. Initially, quadrotors made agile maneuvers with the help of external motion capture systems [2, 3]. The most impressive feats involved passing at high speeds through gaps and circles. More recently, various researchers have focused on bringing the necessary state estimation for these maneuvers onboard. Loianno et al. plan an optimal trajectory through a narrow gap with difficult angles while using Visual Inertial Odometry (VIO) for navigation [4]. The average maximum speed of their drone can achieve 4.5$m/s$. However, the position of the gap is known accurately a priori, so no gap detection module is included in their research. Falanga et al. have their research on flying a drone through a gap aggressively by detecting the gap with fully onboard resources [5]. They fuse the pose estimation from the detected gap and onboard sensors to estimate the state. In their experiment, the platform with a forward-facing fish-eye camera can fly through the gap with $3m/s$. Sanket et al. develop a solution for a drone to fly through arbitrarily shaped gaps without building an explicit 3D model of a scene, using only a monocular camera [6].

Drone racing represents a larger, even more challenging problem than performing short agile flight maneuvers. The reasons for this are that: (1) all sensing and computing has to happen on board, (2) passing one gate is not enough. Drone races can contain complex trajectories through many gates, requiring good estimation and (optimal) control also on the longer term, and (3) depending on the race, gate positions can change, other obstacles than gates can be present, and the environment is much less controlled than an indoor motion tracking arena.

One category of strategies for autonomous drone racing is to have an accurate map of the track, where the gates have to be in the same place. One of the participants of the IROS 2017 autonomous drone race, the Robotics and Perception Group, reached gate 8 in $35s$. In their approach, waypoints were set using the pre-defined map and VIO was used for navigation. A depth sensor was used for aligning the track reference system with the odometry reference system. NASA's JPL lab report in their research results that their drone can finish their race track in a similar amount of time as a professional pilot. In their research, a visual-inertial localization and mapping system is used for navigation and an aggressive trajectory connecting waypoints is generated to finish the track [7]. Gao et al. come up with a teach-and-repeat solution for drone racing [8]. In the teaching phase, the surrounding environment is reconstructed and a flight corridor is found. Then, the trajectory can be optimized within the corridor and be tracked during the repeating phase. In their research, VIO is employed for pose estimation and the speed can reach $3m/s$. However, this approach is sensitive to changing environments. When the position of the gate is changed, the drone has to learn the environment again.

The other category of strategies for autonomous drone race employs coarser maps and is more oriented on gate detection. This category is more robust to displacements of gates. The winner of IROS 2016 autonomous drone race, Unmanned Systems Research Group, uses a stereo camera for detecting the gates [9]. When the gate is detected, a waypoint will be placed in the center of the gate and a velocity command is generated to steer the drone to be aligned with the gate. The winner of the IROS 2017 autonomous drone race, the INAOE team, uses metric monocular SLAM for navigation. In their approach, the relative waypoints are set and the detection of the gates is used to correct the drift of the drone [10]. Li et al. combine gate detection with onboard IMU readings and a simplified drag model for navigation [11]. With their approach, a Parrot Bebop 1 ($420g$) can use its native onboard camera and processor to fly through 15 gates with $1.5m/s$ along a narrow track in a basement full of exhibits. Kaufmann et al. use a trained CNN to map the input images to the desired waypoint and the desired speed to approach it [12]. With the generated waypoint, a trajectory through the gate can be determined and executed while VIO is used for navigation. The winner of the IROS 2018 autonomous drone race, the Robotics and Perception Group, finished the track with $2m/s$ [13]. During the flight, the relative position of the gates and a corresponding uncertainty measure are predicted by a Convolutional Neural Network (CNN). With the estimated position of the gate, the waypoints are generated, and a model predictive controller (MPC) is used to control the drone to fly through the waypoints while VIO is used for navigation.

From the research mentioned above, it can be seen that many of the strategies for autonomous drone racing are based on generic, but computationally relatively expensive navigation methods such as VIO or SLAM. These methods require heavier and more expensive processors and sensors, which leads to heavier and more expensive drone platforms. Forgoing these methods could lead to a considerable gain in computational effort, but raises the challenge of still obtaining fast and robust flight.

In this paper, we present a solution to this challenge. In particular, we propose a Visual Model-predictive Localization (VML) approach to autonomous drone racing. The approach does not use generic vision methods such as VIO and SLAM and is still robust to gate changes, while reaching speeds competitive to the currently fastest autonomous racing drones. The main idea is to rely as much as possible on a predictive model of the drone dynamics, while correcting the model and localizing the drone visually based on the detected gates and their supposed positions in the global map. To demonstrate the efficiency of our approach, we implement the proposed algorithms on a cheap, commercially available smart-camera called “Jevois” and mount it on the “Trashcan” racing drone. The modified Trashcan weighs only $72g$ and is able to fly the race track with the speed up to $2.6m/s$. The vision-based navigation and high-level controller run on the Jevois camera while the low-level controller provided by the open source Paparazzi autopilot [14, 15] runs on the Trashcan. To the best of our knowledge, the presented drone is the smallest and one of the fastest autonomous racing drone in the world. Figure 4.2 shows the weight and the speed of our drone in comparison to the drones of the winners of the IROS autonomous drone races.

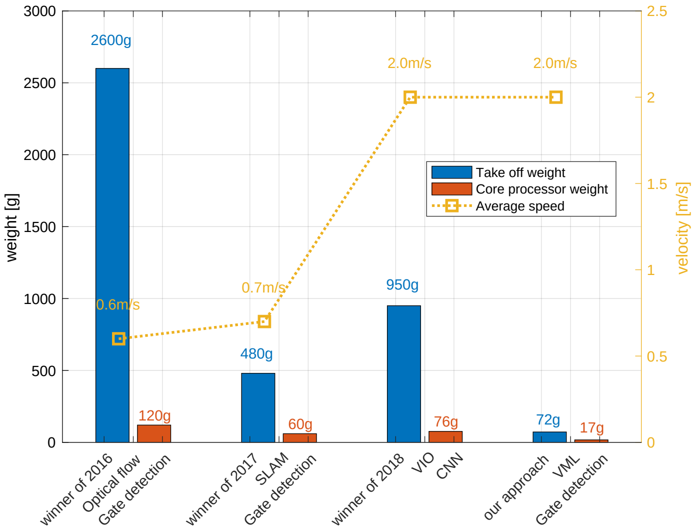


Figure 4.2: The weight and the speed of the approach proposed in this article and the winners' of IROS autonomous drone race. All weights are either directly from the articles or estimated from online specs of the used processors.

## 4.2. PROBLEM FORMULATION AND SYSTEM DESCRIPTION

### 4.2.1. PROBLEM FORMULATION

In this work, we will develop a hardware and a software system that the flying platform can fly through a drone race track fully autonomously with high speed using only onboard resources. The racing track setup can be changed and the system should be adaptive to this change autonomously.

For visual navigation, instead of using SLAM or VIO, we directly use a computationally efficient vision algorithm for the detection of the racing gate to provide the position information. However, implementing such a vision algorithm on low-grade vision and processing hardware results in low frequency, noisy detections with occasional outliers. Thus, a filter should be employed to still provide high frequency and accurate state estimation. In Section 4.3, we first briefly introduce the 'Snake Gate Detection' method and a pose estimation method used to provide position measurements. Then, we propose and analyze the novel visual model-predictive localization technique that estimates the drone's states within a time window. It fuses the low-frequency onboard gate detections and high-frequency onboard sensor readings to estimate the position and the velocity of the drone. The control strategy to steer the drone through the racing track is discussed. The simulation result in Section 4.4 shows the comparison between the proposed filter and the Kalman filter in different scenarios with outliers and delay. In Section 4.5, we will introduce the flying experiment of the drone flying through a racing track with gate displacement, different altitude and moving gate during the flight. In Section 4.6, the generalization and the limitation of the proposed method are discussed. Section 4.7 concludes the article.

### 4.2.2. SYSTEM OVERVIEW

To illustrate the efficiency of our approach, we use a small racing drone called Trashcan (Figure 4.3). This racing drone is designed for FPV racing with the Betaflight flight controller software. In our case, to fly this Trashcan autonomously, we replaced Betaflight by the Paparazzi open source autopilot for its flexibility of adding custom code, stable communication with the ground for testing code and active maintenance from the research community. In this article, the Paparazzi software only aims to provide a low level controller. The main loop frequency is $2k$Hz. We employ a basic complementary filter for attitude estimation and the attitude control loop is a cascade control including a rate loop and an attitude loop. For each loop, a P-controller is used. The details of Trashcan's hardware can be found in Table 4.1

Table 4.1: The specifications of Trashcan's hardware

| | |
|---|---|
| Weight | $48g$ (with the original camera) |
| Size | $98mm \times 98mm \times 36mm$ |
| Motor | TC0803 KV15000 |
| MCU | STM32F4 (100MHZ) |
| Receiver | FrSky D16 |

For the high level vision, flight planning and control tasks, we use a light-weight

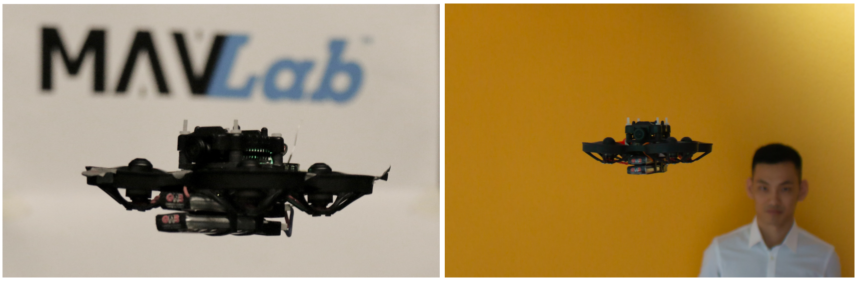
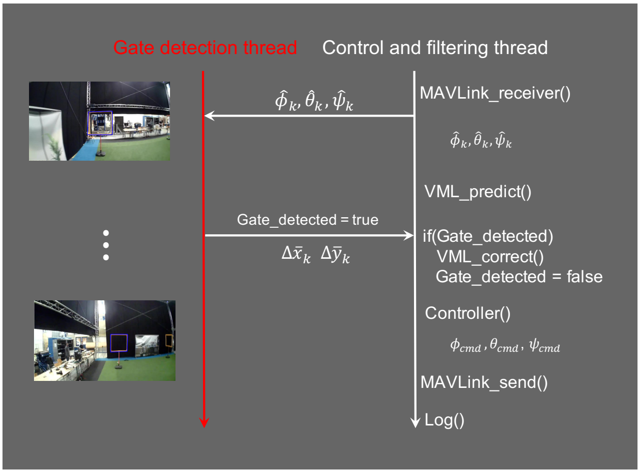

(a) (b)

Figure 4.3: The flying platform. The Jevois is mounted on the Trashcan. The Trashcan provides power to the Jevois and they communicate with each other by the MAVLink protocol. The weight of the whole platform is only 72$g$.

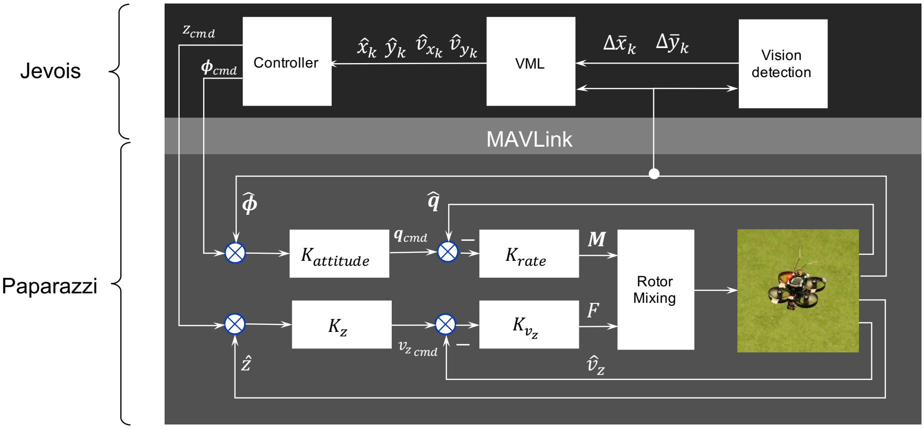
smart camera (17$g$) called Jevois, which is equipped with a quad core ARM Cortex A7 processor and a dual core Mali-400 GPU. In our experiment, there are two threads running on the Jevois, one of which is for vision detection and the other one is for filtering and control (Figure 4.4(a)). In our case, the frequency of detecting gates ranges from 10HZ to 30HZ and the frequency of filtering and control is set to 512HZ. The Gate detection thread processes the images in sequence. When it detects the gate it will send a signal telling the other thread a gate is detected. The control and filtering thread keeps predicting the states and calculating control command in high frequency. It uses a novel filtering method, explained in Section 4.3, for estimating the state based on the IMU and the gate detections.

The communication between the Jevois and Trashcan is based on the MAVLink protocol with a baud rate of 115200. Trashcan sends the AHRS estimation with a frequency of 512HZ. And the Jevois sends the attitude and altitude commands to Trashcan with a frequency of 200HZ. The software architecture of the flying platform can be found in Figure 4.4(b).

In Figure 4.4(b), the Gate detection and Pose estimation module first detects the gate and estimates the relative position between the drone and the gate. Next, the relative position will be sent to the Gate assignment module to be transferred to global position. With the global position measurements and the onboard AHRS reading, the proposed VML filter fuses them together to have accurate position and velocity estimation. Then, the Flight plan and high level controller will calculate the desired attitude commands to steer the drone through the whole track. These attitude commands will be sent to the drone via MAVLink protocol. On the Trashcan drone, Paparazzi provides the low level controller to stabilize the drone.


Gate detection thread
Control and filtering thread
$\hat{\phi}_k, \hat{\theta}_k, \hat{\psi}_k$
MAVLink_receiver()
$\hat{\phi}_k, \hat{\theta}_k, \hat{\psi}_k$
VML_predict()
Gate_detected = true
$\Delta\bar{x}_k \ \Delta\bar{y}_k$
if(Gate_detected)
VML_correct()
Gate_detected = false
Controller()
$\phi_{cmd}, \theta_{cmd}, \psi_{cmd}$
MAVLink_send()
Log()

(a) The two threads structure running on Jevois. For the gate detection thread, the frequency of gate detection ranges from 10HZ to 30HZ while the frequency of control and filtering thread is 512HZ



(b) The software architecture of the UAV platform. The vision detection, filtering and control are all running on Jevois. Paparazzi provides the low level controller to stabilize the drone

Figure 4.4: The architectures of the software on Jevois and the software of the whole flying platform

## 4.3. Robust Visual Model-predictive Localization (VML) and Control

State estimation is an essential part of drones' autonomous navigation. For outdoor flight, fusing a GPS signal with onboard inertial sensors is a common way to estimate the pose of the drone [16]. However, for indoor flight, a GPS signal is no longer available. Thus, off-board cameras [17], Ultra Wide Band Range beacons [18] or onboard cameras [19] can be used to provide the position or velocity measurements for the drone. The accuracy and time-delay of these types of infrastructure setups differ from each other. Hence, the different sensing setups have an effect on what type of filtering is best for each situation. The most commonly used state estimation technique in robotics is the Kalman filter and its variants, such as the Extended Kalman filter [20–22]. However, the racing scenario has properties that make it challenging for a Kalman filter. Position measurements from gate detections often are subject to outliers, have non-Gaussian noise, and can arrive at a low frequency. This makes the typical Kalman filter approach unsuitable because it is sensitive to outliers, is optimal only for Gaussian noise, and can converge slowly when few measurements arrive. In this section, we will propose a visual model-predictive localization technique which is robust to low-frequency measurements with significant numbers of outliers. Subsequently, we will also present the control strategy for the autonomous drone race.

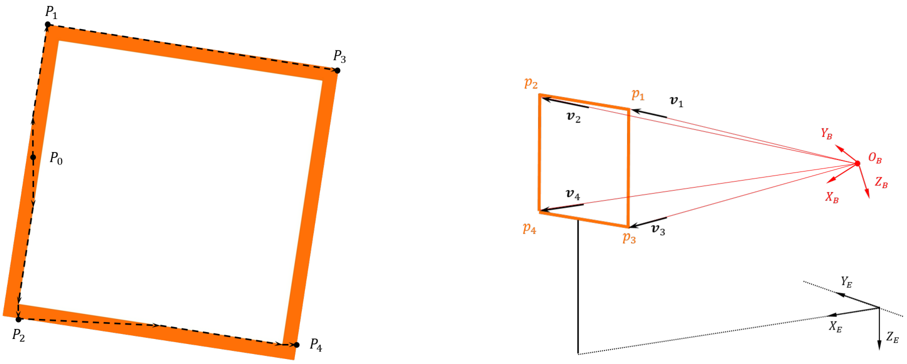
### 4.3.1. GATE ASSIGNMENT

In this article, we use the "snake gate detection" and pose estimation technique as in Li et al. [11]. The basic idea of snake gate detection is searching for continuing pixels with the target color to find the four corners of the gate. Subsequently, a perspective $n$-point (PnP) problem is solved, using the position of the four corners in the image plane, the camera's intrinsic parameters, and the attitude estimation to solve the relative position between the drone and the $i^{th}$ gate at time $k$, $\Delta\bar{\mathbf{x}}^i_k = [\Delta\bar{x}^i_k, \Delta\bar{y}^i_k]$. Figure 4.5 shows this procedure, which is explained more in detail in [11]. In most cases, when the light is even and the camera's auto exposure works properly, the gate in the image is continuous and the Snake gate detection algorithm can detect the gate correctly. However, after an aggressive turn, such as a turn to a window, the camera cannot adapt to the new light condition immediately. In this case, Snake gate detection usually cannot detect the gate. Another failure case is that due to the uneven light condition or the similar color in the background, Snake gate detection may get interfered with. These situations make the searching stop in the middle of the bar or stop at the background pixels. Although we have some mechanism to prevent these false positive detections, there is still a small chance that a false positive happens. The negative effect is that outliers may appear which leads to a challenge for the filter and the controller.

Since for any race a coarse map of the gates is given a priori (cf. Figure 4.1), the position and the heading of gate $i$, $\mathbf{x}^i_g = [x^i_g, y^i_g, \psi^i_g]$ can be known roughly (Figure 4.6). We use the gates' positions to transfer the relative position $\Delta\bar{\mathbf{x}}^i_k$ measured by camera to a global position $\bar{\mathbf{x}}_k = [\bar{x}_k, \bar{y}_k]$ by equation 4.1. In equation 4.1, $x^i_g$, $y^i_g$ and $\psi^i_g$ are the position of the gate $i$ which are known from the map.

$$\begin{bmatrix}\bar{x}_k\\ \bar{y}_k\end{bmatrix} = \begin{bmatrix}x^i_g\\ y^i_g\end{bmatrix} + \begin{bmatrix}\cos\psi^i_g & -\sin\psi^i_g\\ \sin\psi^i_g & \cos\psi^i_g\end{bmatrix}\begin{bmatrix}\Delta\bar{x}^i_k\\ \Delta\bar{y}^i_k\end{bmatrix} \tag{4.1}$$

Here, we assume that the position of the gate is fixed. Any error experienced in the observations is then assumed to be due to estimation drift on the part of the drone. Namely, without generic VIO, it is difficult to make the difference between drone drift and gate displacements. If the displacements of the gates are moderate, this approach will work: after passing a displaced gate, the drone will see the next gate, and correct its position again. We only need a very rough map with the supposed global positions of



(a) Snake gate detection. From one point on the gate $P_0$, the Snake gate detection method first searches up and down, then left and right to find all the four corners of the gate

(b) When the four points of the gate are found, The relative position between the drone and the gate is calculated with the points' position, the camera's intrinsic parameters and the current attitude estimation

Figure 4.5: The Snake gate detection method and pose estimation method [11]

the gates (Figure 4.6). Gate displacements only become problematic if after passing gate $i$ the gate $i+1$ would not be visible when following the path from the expected positions of gate $i$ to gate $i+1$.

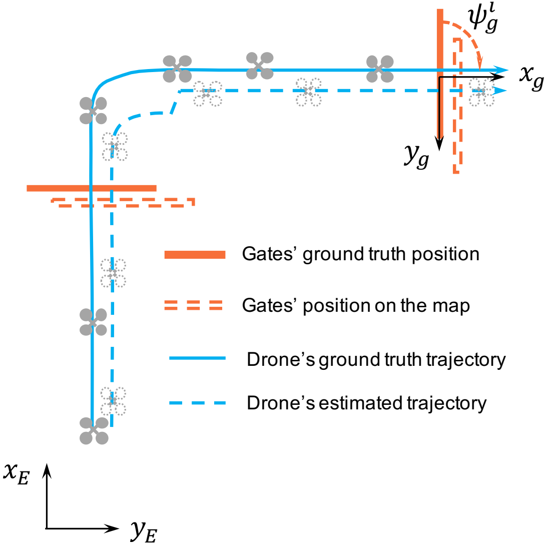

$\psi_g^i$
$x_g$
$y_g$
Gates' ground truth position
Gates' position on the map
Drone's ground truth trajectory
Drone's estimated trajectory
$x_E$
$y_E$

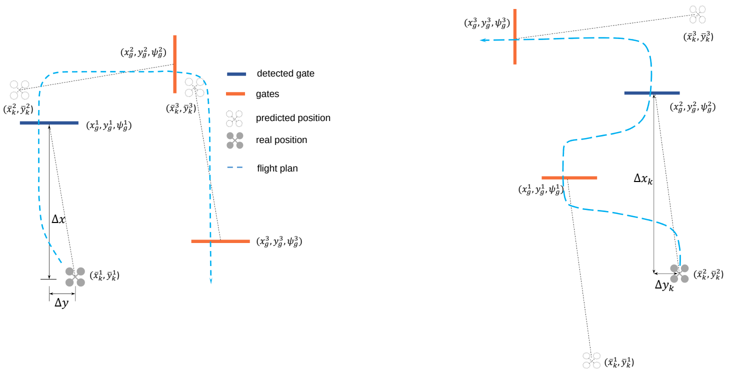
Figure 4.6: The gates are displaced. The drone uses the gate's position on the map to navigate. After passing through the first gate, it will use the second gate's position on the map for navigation. After seeing the second gate, the position of the drone will be corrected.

At the IROS drone race, gates are identical, so for our position to be estimated well, we need to assign a detection to the right gate. For this, we rely on our current estimated global position $\hat{\mathbf{x}}_k = [\hat{x}_k, \hat{y}_k]$. When a gate is detected, we go through all the gates on the map using equation 4.1 to calculate the predicted position $\bar{\mathbf{x}}_k^i = [\bar{x}_k^i, \bar{y}_k^i]$. Then, we calculate the distance between the predicted drone's position $\bar{\mathbf{x}}_k^i$ and its estimated position $\hat{\mathbf{x}}_k$ at time $t_k$ by

$$\Delta d_k^i = \left\| \bar{\mathbf{x}}_k^i - \hat{\mathbf{x}}_k \right\|_2 \tag{4.2}$$

After going through all the gates, the gate with the predicted position closest to the estimated drone position is considered as the detected gate. At time $t_k$, the measurement position is determined by

$$\begin{aligned} j &= \underset{i}{\operatorname{argmin}} \, \Delta d_k^i \\ \bar{\mathbf{x}}_k &= \bar{\mathbf{x}}_k^j \end{aligned} \tag{4.3}$$


$(x_g^2, y_g^2, \psi_g^2)$
$(\bar{x}_k^2, \bar{y}_k^2)$
$(\bar{x}_k^3, \bar{y}_k^3)$
$(x_g^1, y_g^1, \psi_g^1)$
$\Delta x$
$(x_g^3, y_g^3, \psi_g^3)$
$(\bar{x}_k^1, \bar{y}_k^1)$
$\Delta y$
detected gate
gates
predicted position
real position
flight plan
$(x_g^3, y_g^3, \psi_g^3)$
$(\bar{x}_k^3, \bar{y}_k^3)$
$(x_g^2, y_g^2, \psi_g^2)$
$(x_g^1, y_g^1, \psi_g^1)$
$\Delta x_k$
$(\bar{x}_k^2, \bar{y}_k^2)$
$\Delta y_k$
$(\bar{x}_k^1, \bar{y}_k^1)$

(a) It iterates through all gates, evaluating where the drone would be if it was observing those gates. The position closest to the current global position is chosen as the right observation.

(b) The drone detects other gate instead of the one to be flew through. This still helps state estimation, as the observed gate indeed gives an estimate closest to the current estimated global position.

Figure 4.7: In most cases the drone will detect the next gate in the race track. However, the proposed gate assignment strategy also allows to exploit detections of other gates.

The gate assignment technique can help us obtain as much information on the drone's position as possible when a gate is detected. Namely, it can also use detections of other gates than the next gate, and allows to use multiple gate detections at the same time in order to improve the estimation. Still, this procedure will always output a global coordinate for any detection. Hence, false positive or inaccurate detections can occur and have to be dealt with by the state estimation filter.

### 4.3.2. VISUAL MODEL-PREDICTIVE LOCALIZATION (VML)

The racing drone envisaged in this article has a forward-looking camera and an Inertial Measurement Unit (IMU). As explained in the previous section, the camera is used for localization in the environment, with the help of gate detections. Using a typical, cheap CMOS camera will result in relatively slow position updates from the gate detection, with occasional outliers. The IMU can provide high-frequency, and quite accurate attitude estimation by means of an Attitude and Heading Reference System (AHRS). The accelerations can also be used in predicting the change in translational velocities of the drone. In traditional *inertial* approaches, the accelerations would be integrated. However, for smaller drones the accelerometer readings become increasingly noisy, due to less possible damping of the autopilot. Integrating accelerometers is 'acceleration stable', meaning that a bias in the accelerometers that is not accounted for can lead to unbounded velocity estimates. Another option is to use the accelerometers to measure the drag on the frame, which - assuming no wind - can be easily mapped to the drone's translational velocity (cf. [11]). Such a setup is 'velocity stable', meaning that an accelerometer offset of drag model error would lead to a proportional velocity offset, which is bounded. On really small vehicles like the one we will use in the experiments, the accelerometers are even too noisy for reliably measuring the drag. Hence, the proposed approach uses a prediction model that only relies on the attitude estimated by the AHRS which is an indirect way of using the accelerometer. It uses the attitude and a constant altitude assumption to predict the forward acceleration, and subsequently velocity of the drone. The model is corrected from time to time by means of the visual localization. Although the IMU is used for estimating attitude, it is not used as an inertial measurement for updating translational velocities. This leads to the name of the method; Visual Model-predictive Localization (VML), which will be explained in detail in this subsection.

#### PREDICTION ERROR MODEL

As mentioned above, the attitude estimated from the AHRS is used in the prediction of the drone's velocity and position. However, due to the AHRS bias and the model inaccuracy, the prediction will diverge from the ground truth over time. Fortunately, we have visual gate detections to provide position information. This *vision-based localization* will not integrate the error over time but it has a low frequency. Figure 4.8 is a sketch of what the onboard predictions and the vision measurements look like. The red curve is the prediction result diverging from the ground truth curve because of AHRS biases. The magenta dots are the low frequency detections which distribute around the ground truth. The error between the prediction and measurements can be modeled as a linear function of time which will be explained later in this section. When the error model is estimated correctly, it can be used to compensate for the divergence of the prediction to obtain accurate state estimation.

Assuming that there is no wind, and knowing the attitude, we can predict the acceleration in the $x$ and $y$ axis. Figure 4.9 shows the forces the drone experiences. $T_*^*$ denotes the acceleration caused by the thrust of the drone. It provides the forward acceleration together with the pitch angle $\theta$. $D_*^*$ denotes the acceleration caused by the drag which is simplified as a linear function of body velocity. [23]

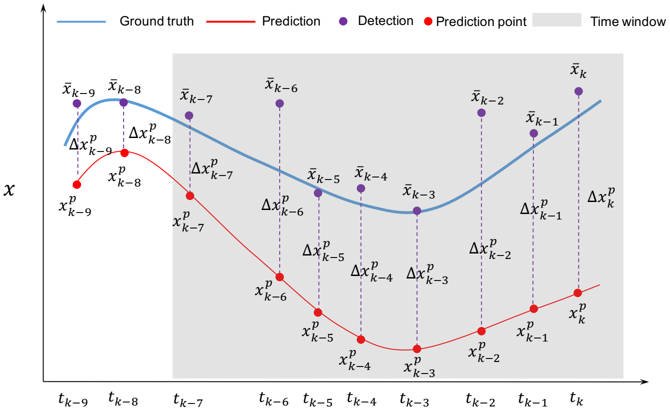


Figure 4.8: Illustrative sketch of the time window $t \in [t_{k-q}, t_k]$. At the beginning of this time window, the difference between the ground truth and the prediction is $\Delta x_{k-q}$ and $\Delta v_{k-q}$. The prediction can be done with high frequency AHRS estimates. The vision algorithm outputs low frequency unbiased measurements. The prediction curve deviates more and more from the ground truth curve over time because of the AHRS bias and model inaccuracy.

$$\begin{cases} D_x^B &= c_x v_x^B \\ D_y^B &= c_y v_y^B \end{cases} \tag{4.4}$$

where $c_*$ is the drag coefficient.

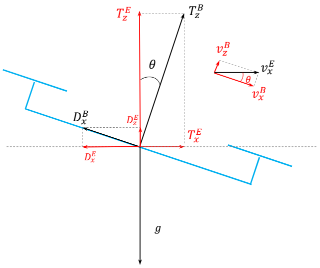


Figure 4.9: Free body diagram of the drone. $v_*^*(t)$ is the velocity of the drone. The superscript $E$ denotes north-east-down (NED) earth frame while $B$ denotes body frame. $T_*^*$ is the acceleration caused by thrust and $D_*^*$ is the acceleration caused by the drag, which is a linear function of the body velocity. $g$ is the gravity factor and $c$ is the drag factor which is positive. $\theta(t)$ is the pitch angle of the drone. It should be noted that since we use NED frame, $\theta < 0$ when the drone pitches down.

According to Newton's second law in $xoz$ plane,

$$\begin{bmatrix} a_x^E(t) \\ a_z^E(t) \end{bmatrix} = \begin{bmatrix} 0 \\ g \end{bmatrix} + \Re_B^E(\theta) \begin{bmatrix} 0 \\ T_z^B(t) \end{bmatrix} + \Re_B^E(\theta)\mathbf{D}\Re_E^B(\theta) \begin{bmatrix} v_x^E(t) \\ v_z^E(t) \end{bmatrix} \tag{4.5}$$

Expand equation 4.5, we have

$$\begin{cases} a_x^E(t) = \sin\theta(t)T_z^B(t) - v_x^E(t)c \\ a_z^E(t) = \cos\theta(t)T_z^B(t) + g - v_z^E(t)c \end{cases} \tag{4.6}$$

where $\Re_E^B(\theta)$ is the rotation matrix and $\mathbf{D} = \begin{bmatrix} -c & 0 \\ 0 & -c \end{bmatrix}$ is the drag coefficient matrix. If the altitude is kept the same as in the IROS drone race, we have

$$\begin{cases} T_z^B(t) = \frac{-g}{\cos\theta(t)} \\ a_x^E(t) = -g\tan\theta(t) - v_x^E(t)c \end{cases} \tag{4.7}$$

Since the model in the $y$ axis has the same form as in the $x$ axis, the dynamic model of the quadrotor can be simplified as

$$\begin{cases} \dot{x}(t) & = v_x^E(t) \\ \dot{y}(t) & = v_y^E(t) \\ \dot{v}_x^E(t) & = -g\tan\theta(t) - v_x^E(t)c \\ \dot{v}_y^E(t) & = g\tan\phi(t) - v_y^E(t)c \end{cases} \tag{4.8}$$

where $x(t)$ and $y(t)$ are the position of the drone, and $\phi$ is the roll angle of the drone. In equation 4.8, the movement in $x$ and $y$ axis is decoupled. Thus we only analyze the movement in the $x$ axis. The result can be directly generalized to the $y$ axis. The nominal model of the drone in $x$ axis can be written by

$$\dot{\mathbf{x}}^n(t) = \mathbf{A}\mathbf{x}^n(t) + \mathbf{B}u^n(t) \tag{4.9}$$

where $\mathbf{x}^n(t) = \begin{bmatrix} x^n(t) \\ v_x^n(t) \end{bmatrix}$,$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & -c \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ -g \end{bmatrix}$ and $u^n = \tan(\theta)$. The superscript $n$ denotes the nominal model. Similarly, with the assumption that the drag factor is accurate, the prediction model can be written as

$$\dot{\mathbf{x}}^p(t) = \mathbf{A}\mathbf{x}^p(t) + \mathbf{B}u^p(t) \tag{4.10}$$

where $\mathbf{x}^p(t) = \begin{bmatrix} x^p(t) \\ v_x^p(t) \end{bmatrix}$ and $u^p = \tan(\theta + \theta_b)$. $\theta_b$ is the AHRS bias and is assumed to be a constant in short time. Consider a time window $t \in [t_{k-q}, t_k]$, the states of nominal model at time $t_k$ are

$$\mathbf{x}_k^n = (\mathbf{I} + \mathbf{A}T_s)^q \mathbf{x}_{k-q}^n + \sum_{i=1}^{q} (\mathbf{I} + \mathbf{A}T_s)^{i-1} \mathbf{B}T_s u_{k-i}^n \tag{4.11}$$

where $T_s$ is the sampling time. The predicted states of model 4.10 are

$$\mathbf{x}_k^p = (\mathbf{I} + \mathbf{A}T_s)^q \mathbf{x}_{k-q}^p + \sum_{i=1}^{q} (\mathbf{I} + \mathbf{A}T_s)^{i-1} \mathbf{B}T_s u_{k-i}^p \tag{4.12}$$

Thus, the error between the predicted model and nominal model can be written as

$$\Delta \mathbf{x}_k^p = (\mathbf{I}+\mathbf{A}T_s)^q \left[\mathbf{x}_{k-q}^p - \mathbf{x}_{k-q}^n\right] + \sum_{i=1}^{q} (\mathbf{I}+\mathbf{A}T_s)^{i-1} T_s \mathbf{B} u_b \tag{4.13}$$

where $u_b = (u_{k-i}^p - u_{k-i}^n)$ is the input bias which can be considered as a constant in a short time. In equation 4.13,

$$(\mathbf{I}+\mathbf{A}T_s)^i = \begin{bmatrix} 1 & T_s \sum_{j=1}^{i} (1-cT_s)^{j-1} \\ 0 & (1-cT_s)^i \end{bmatrix} \tag{4.14}$$

Since the sampling time $T_s$ is small, ($T_s = 0.002s$ in our case), we can assume

$$(\mathbf{I}+\mathbf{A}T_s)^i \approx \begin{bmatrix} 1 & iT_s \\ 0 & 1 \end{bmatrix} \tag{4.15}$$

Hence, equation 4.13 can be approximated by

$$\Delta \mathbf{x}_k^p \approx (\mathbf{I}+\mathbf{A}T_s)^q \left[\mathbf{x}_{k-q}^p - \mathbf{x}_{k-q}^n\right] + \sum_{i=1}^{q} \begin{bmatrix} 1 & iT_s \\ 0 & 1 \end{bmatrix} T_s \mathbf{B} u_b \tag{4.16}$$

$$\approx \begin{bmatrix} 1 & qT_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_k^p \\ \Delta v_k^p \end{bmatrix} + \begin{bmatrix} q & \frac{q(q+1)}{2} T_s \\ 0 & q \end{bmatrix} T_s \mathbf{B} u_b \tag{4.17}$$

Expanding equation 4.17, we have

$$\begin{cases} \Delta x_k^p \approx \Delta x_{k-q}^p + qT_s \Delta v_{x\,k-q}^p - \frac{q(q+1)}{2} T_s^2 g u_b \\ \Delta v_k^p \approx \Delta v_{x\,k-q}^p - qT_s g u_b \end{cases} \tag{4.18}$$

Actually, $qT_s = t_k - t_{k-q}$ is the time span of the time window. If we neglect $T_s^2$ term, we can have the prediction error at time $t_k$

$$\Delta x_k^p \approx \Delta x_{k-q}^p + (t_k - t_{k-q}) \Delta v_{k-q}^p \tag{4.19}$$

Thus, within a time window, the state estimation problem can be transformed to a linear regression problem with model equation 4.19, where $\hat{\beta} = [\Delta x_{k-q}^p, \Delta v_{k-q}^p]^{\mathrm{T}}$ are the parameters to be estimated. From equation 4.19, we can see that in a short time window, the AHRS bias does not affect the prediction error. The error is mainly caused by the initial prediction error $\Delta x_{k-q}^p$. Furthermore, velocity error $\Delta v_{k-q}^p$ can cause the prediction error to diverge over time. If the time window is updated frequently, model 4.19 can remain accurate enough. Hence, in this work, we focus on the main contributors to the prediction error and will not estimate the bias term. The next step is how to efficiently and robustly estimate $\Delta x_{k-q}^p$ and $\Delta v_{k-q}^p$.

In this simplified linear prediction error model, we use the constant altitude assumption to approximate the thrust $T_z^B$ on the drone, which may lead to inaccuracy of the model. During the flight, this assumption may be violated by aggressive maneuvers in $z$ axis. However, if the maneuver in $z$ axis is not very aggressive and the time window is small (in our case less than $2s$), the prediction error model's inaccuracy level can be kept in an acceptable range. In the simulation and the real-world experiment shown later, we will show that although the altitude of the drone changes $1m$ in $2s$, the proposed filter can still have very high accuracy with this assumption. Another way to improve the model accuracy is to estimate the thrust by fusing the accelerometer readings and rotor speed together, which needs the establishment of the rotors' model. It should also be noted that we neglect $T_s^2$ term in equation 4.18 to have a linear model. To increase the model accuracy, the prediction error model can be a quadratic model. In our case, since the time window is small, the linear model is accurate enough.

### PARAMETER ESTIMATION METHOD

The classic way for solving the linear regression problem based on equation 4.19 is to use the Least Square Method (LS Method) with all data within the time window and estimate the parameters $\hat{\beta}$.

$$\hat{\beta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y} \tag{4.20}$$

where

$$\hat{\beta} = \begin{bmatrix} \Delta x^p_{k-q} & \Delta v^p_{k-q} \end{bmatrix}^{\mathrm{T}}, \mathbf{X} = \begin{bmatrix} 1 & t_{k-q} - t_{k-q} \\ 1 & t_{k-q+1} - t_{k-q} \\ \vdots & \vdots \\ 1 & t_k - t_{k-q} \end{bmatrix}, \mathbf{Y} = \begin{bmatrix} x^p_{k-q} - \bar{x}_{k-q} \\ x^p_{k-q+1} - \bar{x}_{k-q+1} \\ \vdots \\ x^p_k - \bar{x}_k \end{bmatrix}$$

The LS Method in equation 4.20 can give optimal unbiased estimation under the assumption of zero-mean Gaussian residuals. However, if there exist outliers in the time window $t \in [t_{k-q}, t_k]$, they will be considered equally during the estimation process. These outliers can significantly affect the estimation result. Thus, to exclude the outliers, we employ random sample consensus (RANSAC) to increase the performance [24]. In a time window $t \in [t_{k-q}, t_k]$, we first calculate the prediction error $\Delta\mathbf{x}^p_{k-q,k} = \{\Delta x^p_{k-q+i} | \Delta x^p_{k-q+i} = x^p_{k-q+i} - \bar{x}_{k-q+i}, 0 \leq i \leq q\}$ and time difference $\Delta\mathbf{t} = \{\Delta t_i | \Delta t_i = t_i - t_{k-q}, 0 \leq i \leq q\}$. For each iteration $i$, the subsets of $\Delta\mathbf{x}^p_{k-q,k}$ and $\Delta\mathbf{t}_{k-q,k}$ are randomly selected, which are denoted by $\Delta\mathbf{x}^s_{k-q,k}$ and $\Delta\mathbf{t}^s_{k-q,k}$. The size of the subset $n^s$ can be calculated by $n^s = q\sigma_s$, where $\sigma_s$ is the ratio of sampling. We use subsets $\Delta\mathbf{x}^s_{k-q,k}$ and $\Delta\mathbf{t}^s_{k-q,k}$ to estimate the parameters $\hat{\beta}_i$ (Figure 4.10).

When $\hat{\beta}_i$ is estimated, it will be used to calculate the total prediction error $\varepsilon_i$ of the all the data in the time window $t_i \in [t_{k-q}, t_k]$ by

$$\varepsilon_i = \sum_{j=k-q}^{k} \epsilon_j \tag{4.21}$$

where

$$\epsilon_j = \begin{cases} \left\| \Delta v^p_{k-q_i}(\Delta t_j - \Delta t_{k-q}) + \Delta x^p_{k-q_i} - \Delta x^p_j \right\|_2, & \text{if } \epsilon_j < \sigma_{th} \\ \sigma_{th}, & \text{otherwise} \end{cases} \tag{4.22}$$

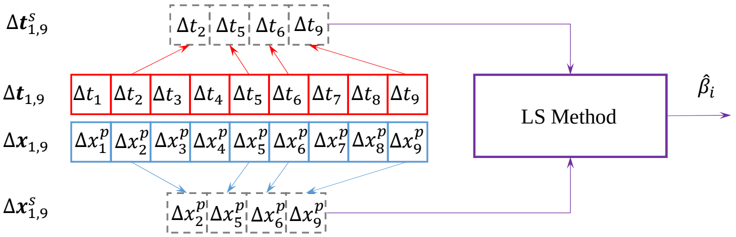

$\Delta\boldsymbol{t}^s_{1,9}$
$\Delta t_2$ $\Delta t_5$ $\Delta t_6$ $\Delta t_9$
$\Delta\boldsymbol{t}_{1,9}$
$\Delta t_1$ $\Delta t_2$ $\Delta t_3$ $\Delta t_4$ $\Delta t_5$ $\Delta t_6$ $\Delta t_7$ $\Delta t_8$ $\Delta t_9$
$\Delta\boldsymbol{x}_{1,9}$
$\Delta x^p_1$ $\Delta x^p_2$ $\Delta x^p_3$ $\Delta x^p_4$ $\Delta x^p_5$ $\Delta x^p_6$ $\Delta x^p_7$ $\Delta x^p_8$ $\Delta x^p_9$
$\Delta\boldsymbol{x}^s_{1,9}$
$\Delta x^p_2$ $\Delta x^p_5$ $\Delta x^p_6$ $\Delta x^p_9$
LS Method
$\hat{\beta}_i$

Figure 4.10: In the $i^{th}$ iteration, the data in the time window $t \in [t_1, t_9]$ will be randomly sampled into $\Delta\mathbf{t}^s_{k-q,k}$ and $\Delta\mathbf{x}^s_{k-q,k}$. Then LS Method (equation 4.20) will be used to estimate the parameters $\hat{\beta}_i$. In this example, $\sigma_s = 0.4$, which means that $n^s = 9 \times 0.4 \approx 4$ samples should be sampled.

In the process of equation 4.21, if $\epsilon_j$ is larger than a threshold $\sigma_{th}$, it counts the threshold as the error. After all the iterations, the parameters $\hat{\beta}_i$ which has the least prediction error will be selected to be the estimated parameters for this time window $t_i \in [t_{k-q}, t_k]$. The pseudo-code of this Basic RANSAC Fitting (BRF) method can be found in Algorithm 4.

With the Basic RANSAC Fitting (BRF) method, the influence of the outliers is reduced, but it has no mechanism to handle the peak in velocity estimation (PiVE). For example, in time window $t_i \in [t_{k-q}, t_k]$, BRF can estimate the optimal parameters $\hat{\beta}$ with the minimal error. However, sometimes it will set $\Delta v^p_{k-q}$ to unrealistically high values. This happens when there are few detections in the time window, which may result in the inaccurate estimation of the parameters. In reality, the drone flies at maximum speed $3m/s$, so the velocity prediction error at the start of time window $t_{k-q}$ should not be too large. To avoid PiVE, we add a penalty factor/prior matrix $\mathbf{P}$ to limit $\Delta v^p_{k-q}$ in the fitting process. The loss function can be written as

$$J(\hat{\beta}) = \left\|\mathbf{X}\hat{\beta} - \mathbf{Y}\right\|_2^2 + \hat{\beta}^{\mathrm{T}}\mathbf{P}\hat{\beta} \tag{4.23}$$

where

$$\mathbf{P} = \begin{bmatrix} p_x & 0 \\ 0 & p_v \end{bmatrix} \tag{4.24}$$

is the penalty factor/prior matrix. To minimize the loss function, we take derivatives of $J(\hat{\beta})$ and let it be 0

$$\frac{\partial J(\hat{\beta})}{\partial \hat{\beta}} = 2\mathbf{X}^{\mathrm{T}}\mathbf{X}\hat{\beta} - 2\mathbf{X}^{\mathrm{T}}\mathbf{Y} + \mathbf{P}\hat{\beta} + \mathbf{P}^{\mathrm{T}}\hat{\beta} = 0 \tag{4.25}$$

Then we have the estimated parameters by

$$\hat{\beta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \mathbf{P})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y} \tag{4.26}$$

We call the use of equation 4.26 inside the RANSAC fitting the Prior RANSAC fitting (PRF). Compared to equation 4.20, PRF has the penalty factor/prior matrix $\mathbf{P}$ in it. By tuning matrix **P** we can add the prior knowledge to the parameter distribution. For example, in our case $\Delta v^p_{k-q}$ should not be high. Thus, we can increase $p_v$ in **P** to limit the value of $\Delta v^p_{k-q}$.

To conclude, in this part we propose 3 methods for estimating the parameters $\hat{\beta}$. The first one is the LS Method which considers all the data in a time window equally. The second method is Basic RANSAC Fitting method (BRF), which has the mechanism to exclude the outliers. And the third one is Prior RANSAC Fitting method (PRF), which can not only exclude the outliers but also take into account the prior knowledge to avoid PiVE. In the next section, we will discuss and compare these 3 methods in simulation to see which one is the most suitable for our drone race scenario.

### PREDICTION COMPENSATION

After the error model (equation 4.19) is estimated in time window $k$, the error model can be used to compensate the prediction by

$$\begin{bmatrix}\hat{x}_{k+i}\\ \hat{v}_{k+i}\end{bmatrix} = \begin{bmatrix}x^p_{k+i}\\ v^p_{k+i}\end{bmatrix} - \begin{bmatrix}1 & t_{k+i}-t_{k-q}\\ 0 & 1\end{bmatrix}\begin{bmatrix}\Delta x^p_{k-q}\\ \Delta v^p_{k-q}\end{bmatrix} \tag{4.27}$$

Also, at each prediction step, the length $\Delta T = t_k - t_{k-q}$ of the time window will be checked, since the simplified model 4.19 is based on the assumption that the time span of the time window $\Delta T$ is small. If $\Delta T$ is larger than the allowed maximum time window size $\Delta T_{max}$, the filter will delete the oldest elements until $\Delta T < \Delta T_{max}$. The pseudo-code of the proposed VML with LS Method can be found in Algorithm 5 and Algorithm 6.

### COMPARISON WITH KALMAN FILTER

When it comes to state estimation or filtering technique, it is inevitable to mention the Kalman filter which is the most commonly used state estimation method. The basic idea of the Kalman filter is that at time $t_{k-1}$, it first predicts the states at time $t_k$ with its error covariance $\mathbf{P}_{k|k-1}$ to have prior knowledge of the states at $t_k$.

$$\begin{aligned}
&\hat{\mathbf{X}}_{k|k-1} = \hat{\mathbf{X}}_{k-1} + \mathbf{f}(\hat{\mathbf{X}}_{k-1}, \mathbf{u}_{k-1})\mathrm{T_s}\\
&\mathbf{F}_{k-1} = \frac{\partial}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_{k-1}}\\
&\Phi_{k|k-1} \approx \mathbf{I} + \mathbf{F}_{k-1}\mathrm{T}\\
&\mathbf{H}_k = \frac{\partial}{\partial \mathbf{x}}\mathbf{h}(\mathbf{x}(t))|_{\mathbf{x}(t)=\hat{\mathbf{x}}_k}\\
&\mathbf{P}_{k|k-1} = \Phi_{k|k-1}\mathbf{P}_{k-1}\Phi^{\mathrm{T}}_{k|k-1} + \mathbf{Q}_{k-1}
\end{aligned} \tag{4.28}$$

When an observation arrives, the Kalman filter uses an optimal gain $\mathbf{K}_k$ which is a combination of the prior error covariance $\mathbf{P}_{k+1|k}$ and the observation's covariance $\mathbf{R}_k$ to compensate the prediction, which as a result, leads to the minimum error covariance $\mathbf{P}_k$.

$$\begin{aligned}
&\delta\hat{\mathbf{X}}_k = \mathbf{K}_k\left\{\mathbf{Z}_k - \mathbf{h}[\hat{\mathbf{X}}_{k|k-1}, k]\right\}\\
&\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^{\mathrm{T}}_k[\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}^{\mathrm{T}}_k + \mathbf{R}_k]^{-1}\\
&\hat{\mathbf{X}}_k = \hat{\mathbf{X}}_{k|k-1} + \delta\hat{\mathbf{X}}_k\\
&\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k/k-1}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^{\mathrm{T}} + \mathbf{K}_k\mathbf{R}_k\mathbf{K}^{\mathrm{T}}_k
\end{aligned} \tag{4.29}$$

According to [25], a Kalman filter is a least square estimation made into a recursive process by combining prior data with coming measurement data. The most obvious difference between the Kalman filter and the proposed VML is that VML is not a recursive method. It does not estimate the states at $t_k$ only based on the last step states $\hat{\mathbf{x}}_{k-1}$. It estimates the states considering the previous prediction and observations in a time window.

In the VML approach, we use least square method within a time window, which looks similar to the least square estimation method. However, there are two major differences between the two methods. The first one is that in the proposed VML, the prediction information is fused to the VML. Secondly and most importantly, we estimate the prediction error model $\hat{\beta}$ instead of estimating all the states in the time window as in the least square method. In Section 4.4, we will introduce Kalman filter's different variants for outliers and delay and compare them with VML in estimation accuracy and computation load in detail.

### 4.3.3. FLIGHT PLAN AND HIGH LEVEL CONTROL

With the state estimation method explained above, to fly a racing track, we employ a flight plan module which sets the waypoints that guide the drone through the track and a two-loop cascade P-controller to execute the reference trajectory (Figure 4.11).

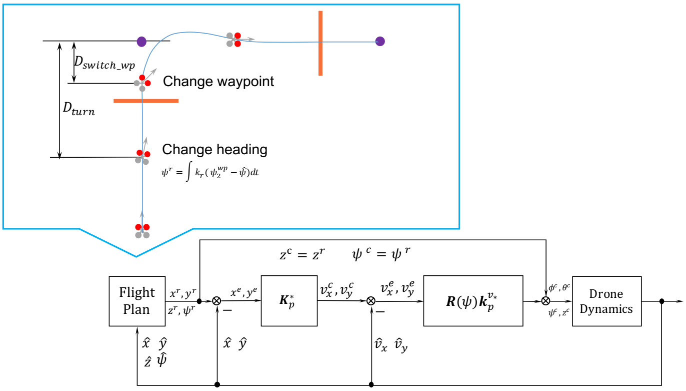


Figure 4.11: The Flight plan module generates the waypoints for the drone to fly the track. When the distance between the drone and the current waypoint $d < D_{turn}$, the drone starts to turn to the next waypoint while still approaching the current waypoint. When $d < D_{switch\_wp}$, the drone switches the current waypoint to the next one. The cascade P-controller is used for executing the reference trajectory from the flight plan module. The attitude and rate controllers are provided by the Paparazzi autopilot. $k_r$ is a positive constant to adjust the speed of the drone's yawing to the setpoint. In the real world experiment and simulation, we set $k_r = 1$.

Usually, the waypoint is just behind the gate. When the distance between the drone and the waypoint is less than a threshold $D_{turn}$, the gate can no longer be detected by our method, and we set the heading of the drone to the next waypoint. This way, the drone will start turning towards the next gate before arriving at the waypoint. When the distance between the drone and the waypoint is within another threshold $D_{switch\_wp}$, the waypoint switches to the next point. With this strategy, the drone will not stop at one waypoint but already start accelerating to the next waypoint, which can help to save time. The work flow of flight plan module can be found in Algorithm 7.

We employ a two-loop cascade P controller (equation 4.30) to control the drone to reach the waypoints and follow the heading reference generated from the flight plan module. The altitude and attitude controllers are provided by the Paparazzi autopilot, and are both two-loop cascade controllers.

$$\Phi^c(k) = \mathbf{R}_\psi \mathbf{K}_v (\mathbf{K}_x (\mathbf{x}^r(k) - \hat{\mathbf{x}}(k)) - \hat{\mathbf{v}}(k)) \tag{4.30}$$

where $\Phi^c(k) = [\theta^c(k), \phi^c(k)]^{\mathrm{T}}$, $\mathbf{R}_\psi = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix}$, $\mathbf{K}_v = \begin{bmatrix} k_{vx} & 0 \\ 0 & k_{vy} \end{bmatrix}$, $\mathbf{K}_x = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$, $\mathbf{x}^r(k) = [x^r(k), y^r(k)]^{\mathrm{T}}$, $\hat{\mathbf{x}}(k) = [\hat{x}(k), \hat{y}(k)]^{\mathrm{T}}$, $\hat{\mathbf{v}}(k) = [\hat{v}_x(k), \hat{v}_y(k)]^{\mathrm{T}}$.

## 4.4. SIMULATION EXPERIMENTS

### 4.4.1. SIMULATION SETUP

To verify the performance of VML in the drone race scenario, we first test it in simulation and then use an Extended Kalman filter as benchmark to compare both filters to see which one is more suitable in different operation points. We first introduce the drone's dynamics model used in the simulation.

$$\begin{gathered}
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\
\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \Re_B^E \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} + \Re_B^E \mathbf{K} \Re_E^B \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{T} \end{bmatrix} = \begin{bmatrix} k_\phi(\phi^c - \phi) \\ k_\theta(\theta^c - \theta) \\ k_\psi(\psi^c - \psi) \\ k_T(T^c - T) \end{bmatrix}
\end{gathered} \tag{4.31}$$

where $(x, y, z)$ is the position of the drone in the Earth frame. $v_*$ is the velocity of the drone. $g$ is the gravity factor. $T$ is the acceleration caused by the thrust force. $\phi$, $\theta$, $\psi$ are the three Euler angles of the body frame. And $\Re_B^E$ is the rotation matrix from the Body frame to the Earth frame. $\mathbf{K} = diag([-0.5, -0.5, 0])$ is the simplified first order drag matrix, where the values are based on a linear fit of the drag based on real-world data with the Trashcan drone. $\Re_B^E \mathbf{K} \Re_E^B [v_x v_y v_z]^{\mathrm{T}}$ is the acceleration caused by other aerodynamics. The last four equations are the simplified first order model of the attitude controller and thrust controllers where the proportional feedback factors are $k_\phi = 6$, $k_\theta = 6, k_\psi = 5, k_T = 3$. Thus, the model 4.31 in the simulation is a 10 states $\mathbf{x} = [x, y, z, v_x, v_x, v_x, \phi, \theta, \psi, T]^{\mathrm{T}}$ and 4 inputs $\mathbf{u} = [\phi^c, \theta^c, \psi^c, T^c]^{\mathrm{T}}$ nonlinear system. In this simulation, we use the same flight plan module and high-level controllers discussed in Section 4.3 (Figure 4.11) to generate a ground truth trajectory through a 4-gate square racing track. In this track, we use different height to test if the altitude change affects the accuracy of the VML.

Table 4.2: The map of the simulated racing track

| Gate ID | $x[m]$ | $y[m]$ | $z[m]$ | $\psi[°]$ |
|---|---|---|---|---|
| 1 | 4 | 0 | −1.5 | 0 |
| 2 | 4 | 4 | −2.5 | 90 |
| 3 | 0 | 4 | −1.0 | 180 |
| 4 | 0 | 0 | −1.5 | 270 |

With the ground truth states, next step is to generate the sensor reading. In the real world, AHRS estimation outputs biased attitude estimation because of the accelerator's bias. To model AHRS bias, we have a simplified AHRS bias model

$$\begin{bmatrix}\phi_b\\ \theta_b\end{bmatrix}=\begin{bmatrix}\cos\psi & \sin\psi\\ -\sin\psi & \cos\psi\end{bmatrix}\begin{bmatrix}B_N\\ B_E\end{bmatrix} \tag{4.32}$$

where $\phi_b$ and $\theta_b$ are the AHRS biases on $\phi$ and $\theta$. $B_N$ and $B_E$ are the north and east bias caused by the accelerometer bias, which can be considered as constants in short time. From real-world experiments, they are less than 3°. Thus, the AHRS reading can be modelled by

$$\begin{bmatrix}\bar{\phi}_k\\ \bar{\theta}_k\end{bmatrix}=\begin{bmatrix}\phi_k\\ \theta_k\end{bmatrix}+\begin{bmatrix}\cos\psi & \sin\psi\\ -\sin\psi & \cos\psi\end{bmatrix}\begin{bmatrix}B_N\\ B_E\end{bmatrix}+\begin{bmatrix}\epsilon_\phi\\ \epsilon_\theta\end{bmatrix} \tag{4.33}$$

where $\epsilon_* \sim N(0,\sigma_*)$ is the AHRS noise and in our simulation we will set $\sigma_* = 0.5°$, $B_N = -2°$, $B_E = 1°$. For vision measurements generation, we first determine the segment $[u, v]$ of the trajectory where the drone can detect the gate. Then, we calculate the number of the detection by $n_v = \frac{t_u - t_v}{f_v}$, where $f_v$ is the detection frequency. Next, we randomly select $n_v$ points between $u$ and $v$ to be vision points. For these points, we generate detection measurement by

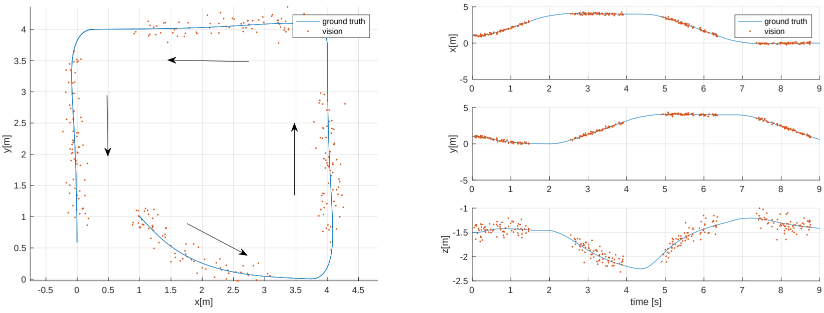
$$\begin{bmatrix}\bar{x}_k\\ \bar{y}_k\end{bmatrix}=\begin{bmatrix}x_k\\ y_k\end{bmatrix}+\begin{bmatrix}\epsilon_x\\ \epsilon_y\end{bmatrix} \tag{4.34}$$

In equation 4.34, $\epsilon_* \sim N(0,\sigma_*)$ is the detection noise and $\sigma_* = 0.1m$ In these $n_v$ vision points, we also randomly select a few points as outlier points, which have the same model with equation 4.34 but $\sigma_* = 3m$. In the following simulations, the parameters are the same with the value mentioned in this section if there is no statement. The simulated ground truth states and sensor measurements are shown in Figure 4.12.

### 4.4.2. SIMULATION RESULT AND ANALYSIS

#### COMPARISON BETWEEN EKF, BRF AND PRF WITHOUT OUTLIERS

We employ an EKF as benchmarks to compare the performance of our proposed filter. The details of the EKF can be found in the Appendix. We first do the simulation in only one operation point, where $f_v = 30$HZ, $\sigma_* = 0.1m$ and the probability of outliers $P_{out} = N_{outliers}/N_{detection} = 0$. At this operation point, three filters are run separately. The result is shown in Figure 4.13.

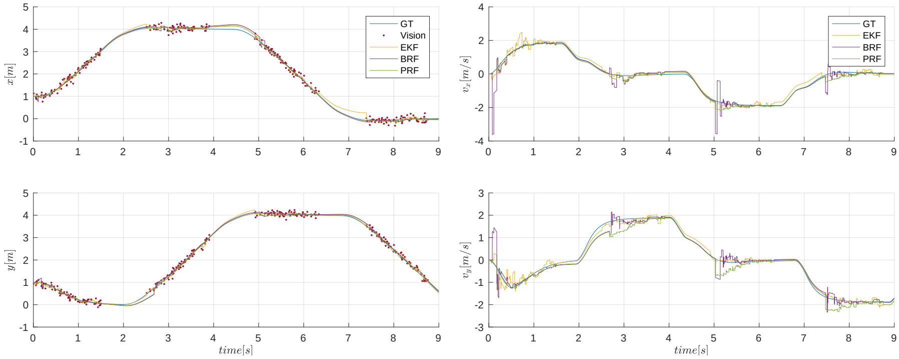


(a) Generated ground truth states and vision measurements in $x - y$ plane

(b) Generated ground truth position and vision measurements

Figure 4.12: In the simulation, the ground truth states are first generated (blue curve). Then, vision measurements and AHRS readings are generated. It can be seen clearly that the bias of AHRS readings changes with the heading, as on a real drone. Namely, the offset in $\phi$ and $\theta$ changes when the $\psi$ changes. This phenomenon is modeled by equation 4.32. In this simulation $f_v = 30\text{HZ}, \sigma_x = \sigma_y = \sigma_z = 0.1m$

.



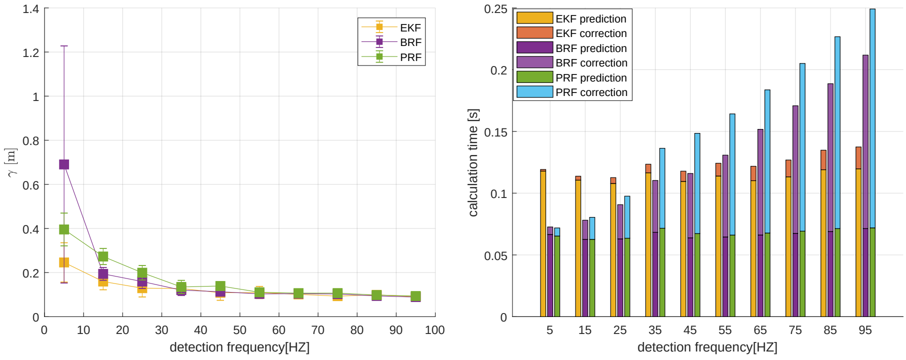
(a) Position estimation of EKF, BRF and PRF

(b) Velocity estimation of EKF, BRF and PRF

Figure 4.13: The filtering result of EKF, BRF and PRF. $f_v = 50HZ$ and $\sigma_x = \sigma_y = 0.1$. When there are no outliers, EKF, BRF and PRF's estimating result all converge to ground truth value. In velocity estimation, however, EKF has longer startup period than VML and BRF shows peaks. To limit this overfitting, in PFR, we add a prior matrix $\mathbf{P} = \begin{bmatrix} 0 & 0 \\ 0 & 0.3 \end{bmatrix}$ and the velocity's peak is significantly smoothed and is closer to the ground truth velocity.

When there are no outliers, all three filters can converge to the ground truth value. However, the EKF has a longer startup period and BRF overfits after turning, leading to unlikely high velocity offsets (the peaks in Figure 4.13b)). This is because, after the turn, the RANSAC buffer is empty. When the first few detections come into the buffer, the

RANSAC has a larger chance to estimate inaccurate parameters. In PRF, however, we add a prior matrix $\mathbf{P} = \begin{bmatrix} 0 & 0 \\ 0 & 0.3 \end{bmatrix}$ to limit the value of $\Delta v$ and the number of the peaks in the velocity estimation is significantly decreased. At the same time, the velocity estimation is closer to the ground truth value.



(a) Estimation error of the filters with different detection frequencies.

(b) Calculation time of the filters.

Figure 4.14: The simulation result of the filters. It can be seen that when the detection frequencies are below $20HZ$, the EKF performs better than BRF and PRF. However, when the detection frequencies are higher than $20HZ$, BRF and PRF start performing better than the EKF. In terms of computation time, the EKF is affected by the detection frequency slightly while the computation load of BRF and PRF increase significantly higher detection frequencies

To evaluate the estimation accuracy of each filter, we first introduce a variable, average estimation error $\gamma$, to be an index of the filter's performance:

$$\gamma = \sqrt{\frac{\sum_{i=1}^{N}(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2}{N}} \tag{4.35}$$

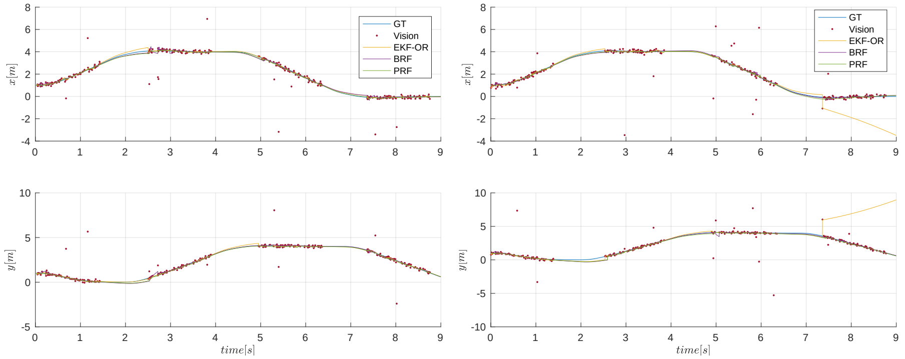
where $N$ is the number of the sample points on the whole trajectory. $\hat{x}$ and $\hat{y}$ are the estimated states by the filter. $x$ and $y$ are the ground truth positions generated by the simulation. $\gamma$ captures how much the estimated states deviate from the ground truth states. A smaller $\gamma$ indicates a better filtering result.

We use running time to evaluate the computation efficiency of each filter. It should be noted that since we need to store all the simulation data for visualization and MATLAB has no mechanism of passing pointers, data accessing can take much computation time. Thus, we only count the running time of the core parts of the filters, which are the prediction and the correction.

The results are shown in Figure 4.14. In the simulation, the time-window in BRF and PRF is set to be $1s$ and 5 iterations are performed in the RANSAC procedure. For each frequency, the filters are run 10 times separately and their average $\gamma$ and running time are calculated. It can be seen in Figure 4.14(a) that when the detection frequency is larger than 30 HZ, BRF and PRF perform close to the EKF. In terms of calculation time, the EKF is heavier than BRF and PRF when the frequency is lower than $40HZ$. It is because that during the prediction phase, the EKF not only predicts the states but also calculates the Jacobian matrix and the prior error covariance $\mathbf{P}_{k|k-1}$ by high frequency while BRF and PRF only do the state prediction. However, when the detection comes, the EKF does the correction by several matrix operations while BRF and PRF do the RANSAC which is much heavier. This explains why the EKF's computation load is only slightly affected by the detection frequency but BRF and PRF's computation load increases significantly with higher detection frequency.

### COMPARISON BETWEEN EKF, BRF AND PRF WITH OUTLIERS

When outliers appear, the regular EKF can be affected significantly. Thus, outlier rejection strategies are always used within an EKF to increase its robustness. A commonly used method is using Mahalanobis distance between the observation and its mean as an index to determine whether an observation is an outlier [26, 27]. Thus, in this section, we implement an EKF with outlier rejection (EKF-OR) as a benchmark to compare the outlier rejection performance of BRF and PRF. The basic idea for the EKF-OR is that the square of the observation's Mahalanobis distance is Chi-square distributed. Hence, when the observation arrives, its Mahalanobis distance will be calculated and checked whether it is within a threshold $\chi_\alpha$. If it is not, this observation will be rejected.



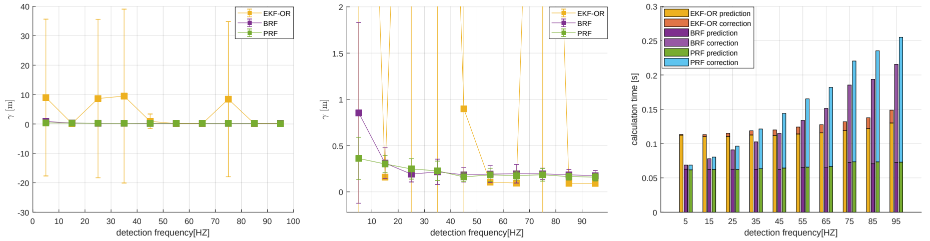
(a) When outliers appear, EKF-OR, BRF and PRF can reject them.

(b) After a long time of pure prediction, EKF-OR has large error covariance. Once it meets an outlier, it has a high chance to jump to it. As a consequence, the later true positive detections are beyond the threshold $\chi_\alpha$ and EKF-OR will treat them as outliers

Figure 4.15: In most cases, EKF-OR, BRF and PRF can reject the outliers. But after a long time of pure prediction, EKF-OR is very vulnerable to the outliers while BRF and PRF still perform well.

Two examples of the filters' rejecting outliers are shown in Figure 4.15. The first figure shows a common case that the three filters can reject the outliers successfully. However, in some special cases, EKF-OR is vulnerable to the outliers. In Figure 4.15(b), for instance, after a long time of pure prediction, the error covariance $\mathbf{P}_{k|k-1}$ becomes large.

Once EKF-OR meets an outlier, it has a high chance to jump to it. The subsequent true positive detections will be treated as outliers and EKF-OR starts diverging. At the same time, BRF and PRF are more robust to the outliers. The essential reason is that for EKF-OR, it depends on its current state estimation (mean and error covariance) to identify the outliers. When the current state estimation is not accurate enough, like the long-time prediction in our case, EKF-OR loses its ability to identify outliers. In other words, it tends to trust whatever it meets. The worse situation is that after jumping to the outlier, its error covariance become smaller which, as a consequence, leads to the rejection of the coming true positive detections. However, for BRF and PRF, outliers are determined in a time window including history. Thus, after long time of prediction, when BRF and PRF meet an outlier, they will judge it considering the detections in the past. If there is no other detection in the time window, they will wait for enough detections to make a decision. With this mechanism, BRF and PRF become more robust than EKF-OR especially when EKF-OR's estimation is not accurate.



(a) Estimation error of the EKF-OR, BRF and PRF with different detection frequencies

(b) Partial enlarged drawing of (a)

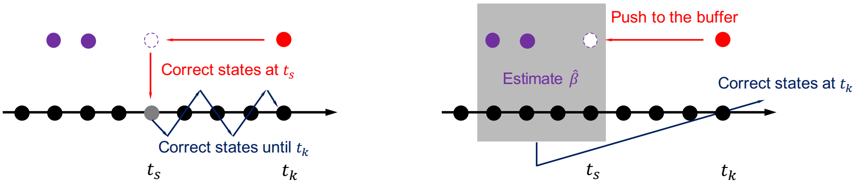
(c) Calculation time of the filters

Figure 4.16: The estimation error of EKF-OR, BRF and PRF and their calculation time with outliers. EKF-OR has some chance (15%) to diverge, which leads to the high estimation error.

Figure 4.16 shows the estimation error and the calculation time of the three filters. As we stated before, although EKF-OR has the mechanism of dealing with the outliers, it still can diverge due to the outliers in some special cases. Thus, in Figure 4.16(a) EKF-OR has large estimation error when the detection frequency is both low and high. In terms of calculation time, it can be seen that it has no significant difference with the non-outlier case.

### FILTERING RESULT WITH DELAYED DETECTION

Image processing and visual algorithms can be very computationally expensive for running onboard a drone, which can lead to significant delay [20, 28]. Many visual navigation approaches ignore this delay and directly fuse the visual measurements with the onboard sensors, which sacrifices the accuracy of the state estimation. A commonly used approach for compensating this vision delay is a modified Kalman filter proposed by Weiss et al. [20]. The main idea of this approach, called EKF delay handler (EKF-DH), is having a buffer to store all sensor measurements within a certain time. At time $t_k$, a vision measurement corresponding to the states at earlier time $t_s$ arrives. It will be used to correct the states at time $t_s$. Then, the states will be propagated again from $t_s$ to $t_k$ (Figure 4.17(a)). Although updating the covariance matrix is not needed according to [20], this approach still requires updating history states whenever a measurement arrives, which can be computationally expensive especially when the delay and the measurement frequency get larger. In our case, we need to use the error covariance for outlier rejections, it is necessary to update the history error covariance matrices, which in turn increases the computation load further. At the same time, for VML, when the measurement arrives, it will first be pushed into the buffer. Then, the error model will be estimated within the buffer/time window. With the estimated parameter $\hat{\beta}$, the prediction at $t_k$ can be corrected directly without the need of correcting all the states between $t_s$ and $t_k$ (Figure 4.17(b)). Thus, the computational burden will not increase when the delay exists.



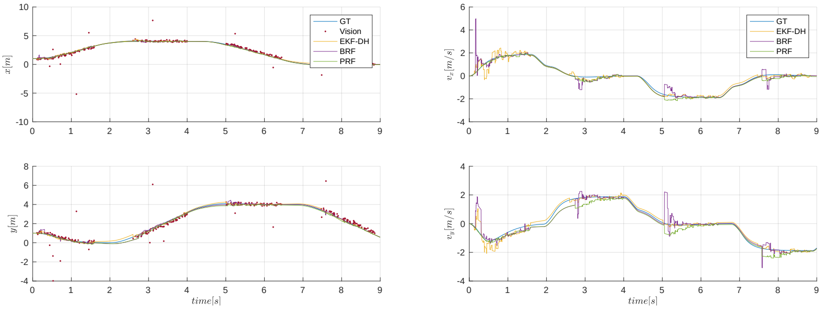
(a) The sketch of the EKF-DH proposed in [20]. When the measurement arrives at $t_k$, EKF-DH first corrects the corresponding states at $t_s$ and then updates the states until $t_k$.

(b) The sketch of VML's mechanism of handling delay. When the measurement arrives, it will be pushed to the buffer with the corresponding states. Then, the error model will be estimated by the RANSAC approach. At last, the estimated model will be used to compensate the prediction at $t_k$. There is no need to update all the states between $t_s$ and $t_k$

Figure 4.17: The sketches of EKF-DH and VML's handling delay mechanism.

Figure 4.18 shows an example of the simulation result of the three filters when both outliers and delay exist. In this simulation, the visual delay is set to be 0.1$s$. It can be seen that although there is a lag between the vision measurements and the ground-truth, all the filters can estimate accurate states. However, EKF-DH requires much more computation effort. Figure 4.19 shows the estimation error and the computation time of the three filters.

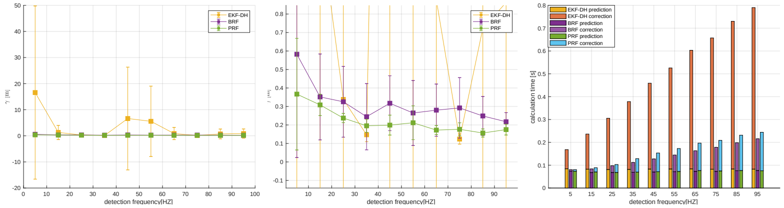
In Figure 4.19, we can see that the computation load of EKF-DH increases significantly due to its mechanism of handling delay. Unsurprisingly, EKF-DH is still sensitive to some outliers while BRF and PRF can handle the outliers.



(a) Position estimation of the three filters with outliers and delay

(b) Velocity estimation of the three filters with outliers and delay

Figure 4.18: An example of the performance of the three filters when outliers and delay exist.



(a) Estimation error of the EKF-DH, BRF and PRF with different detection frequencies

(b) Partial enlarged drawing of (a)

(c) Calculation time of the filters

Figure 4.19: The estimation error of EKF-DH, BRF and PRF and their calculation time with outliers and delay.

## 4.5. REAL-WORLD EXPERIMENTS

### 4.5.1. PROCESSING TIME OF EACH COMPONENT

Before testing the whole system, we first test on the ground how much time the Snake gate detection, the VML-prediction, the VML-correction and the controller take when running on a Jevois smart camera. On the ground, we set an orange gate in front of a Jevois camera and calculate the time that each component takes. For each image, for example, we start timing when a new image arrives and the Snake gate detection is run. Then, we stop timing when the snake gate finishes. For the VML and the controller, we use the same strategy to calculate processing time. In this test, the vision detection frequency is 15HZ and the number of RANSAC iterations in VML is set to 5. Figures 4.20 shows the timing statistics for each component on the Jevois. It can be seen that outliers happen during the testing process, which can be caused by system interrupts. Thus, we first exclude the outliers by the Interquartile Range Method [29] and then provide the statistics for each component. The result can be found in Figure 4.21 and Table 4.3.

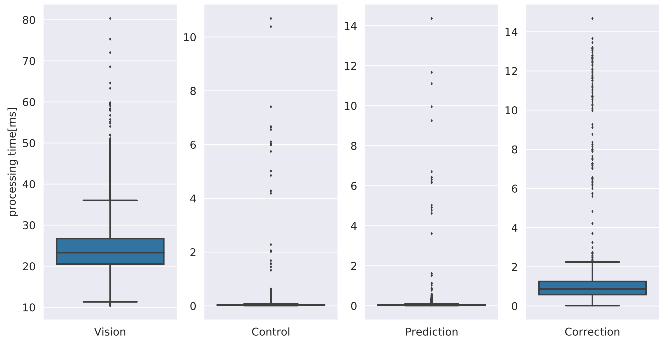
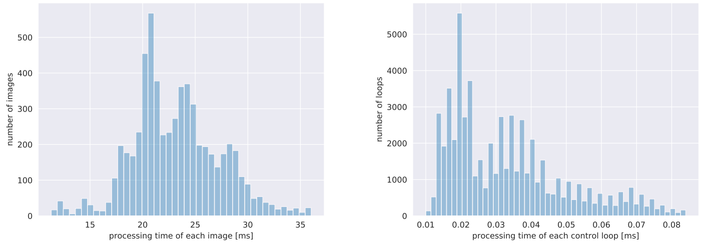
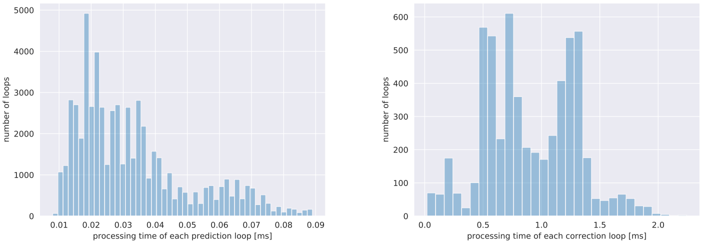


Figure 4.20: The statistical result of the four components' (vision, control, prediction and correction) processing time



(a) Histogram of the vision's processing time (b) Histogram of the controller's processing time



(c) Histogram of the VML-prediction's processing time (d) Histogram of the VML-correction's processing time

Figure 4.21: The histograms of each component's processing time without outliers

Table 4.3: Statistics for VML's components' processing time without outliers

| | mean[ms] | std[ms] | max[ms] | min[ms] | outlier rate |
|---|---|---|---|---|---|
| Vision | 23.2 | 4.2 | 36.0 | 11.3 | 6.7% |
| Controller | 0.03 | 0.02 | 0.08 | 0.01 | 6.4% |
| VML-prediction | 0.03 | 0.02 | 0.09 | 0.01 | 7.5% |
| VML-correction | 0.90 | 0.39 | 2.25 | 0.02 | 2.1% |

From Table 4.3, it can be seen that vision takes much more time than the other three parts. Please note though that the snake gate computer vision detection algorithm is already a very efficient gate detection algorithm. In fact, it has tunable parameters, i.e., the number of samples taken per image for the detection (3000 in the current setup), which allow the algorithm to run even much faster at the cost of having less accuracy (see [11] for more details). The main gain in time in the approach presented in this article is that we do not employ VIO and SLAM, which would take substantially more processing. However, as the Snake gate detection provides relatively low-frequency and noisy position measurements, the VML needs to run in high frequency and cope with the detection noise to still provide accurate estimation for the controller.

### 4.5.2. FLYING EXPERIMENT WITHOUT GATE DISPLACEMENT

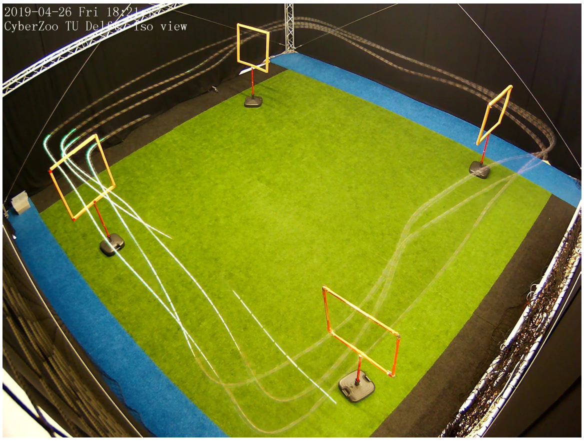


Figure 4.22: The picture of the Trashcan flying the track where the gates are displaced. The average speed is $2m/s$ and the maximum speed is $2.6m/s$.

Figure 4.23 shows the flying result of the drone flying the track without gate displacement. The position of the 4 gates is listed in Table 4.4. In Table 4.4, $x_g$ and $y_g$ are the position of the gates in the real world and $\tilde{x}_g$ and $\tilde{y}_g$ are their position on the map. In this situation, they are the same. The aim of this experiment is to test the filter's performance with sufficient detections. Thus, the velocity is set to be $1.5m/s$ to give the drone more time to detect the gate. In Figure 4.23, the blue curve is the ground truth data from Optitrack motion capture system and the yellow curves are the filtering results. From the flying result, it can be seen that the filtered results are smooth and coincide with the ground truth position well. During the period when the detections are not available, the state prediction is still accurate enough to navigate the drone to the next gate. When the drone detects the next gate, the filter will correct the prediction. In this situation, the divergence of the states is only caused by the prediction drift. It should also be noted that when the outliers appears at $84s$, the filter is not affected by them because of the RANSAC technique in the filter.

Table 4.4: The position of the gates without displacement

| gate ID | $x_g[m]$ | $y_g[m]$ | $\tilde{x}_g[m]$ | $\tilde{y}_g[m]$ |
|---|---|---|---|---|
| 1 | 5 | 0 | 5 | 0 |
| 2 | 6.5 | 5 | 6.5 | 5 |
| 3 | 1 | 7 | 1 | 7 |
| 4 | 0 | 1 | 0 | 1 |

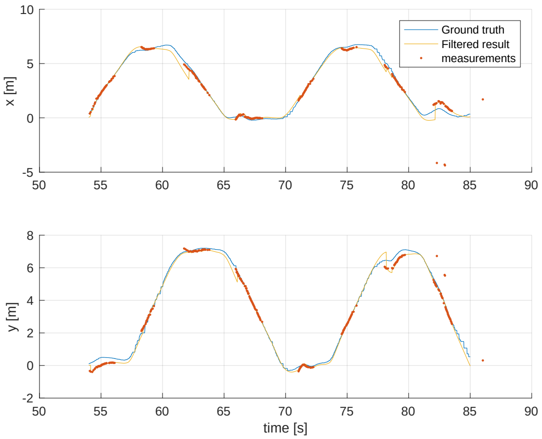
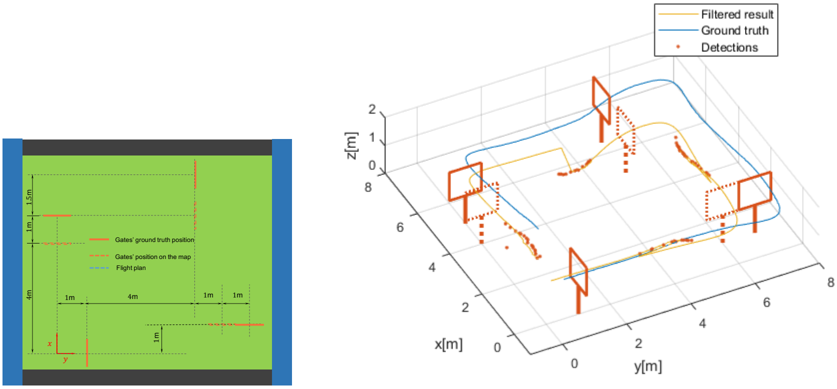


Figure 4.23: The flying result of the drone flying the track without the gate displacement.

### 4.5.3. FLYING EXPERIMENT WITH GATE DISPLACEMENT

In this section, we test our strategy under a difficult condition where the drone flies faster, the gates are displaced and the detection frequency is low. The real gate positions and their position on the map are listed in Table 4.5 and shown in Figure 4.24(a). Gates are displaced between 0 and 1.5m from their supposed positions. The dashed orange lines in Figure 4.24(a) denote the gate positions on the map while the solid orange lines denote the real gate positions which are displaced from the map. Figure 4.24(b) shows the flight data of the first lap. The orange solid gates are the ground truth positions of the gates. The yellow curve is the filtered position based on the gates' positions on the map (orange dashed gates). In other words, the yellow curve is where the drone thinks it is based on the knowledge of the map. After passing through one gate, when the drone detects the next gate, the filter will start correcting the filtering error from the prediction error and the gate displacement.

The whole flight result is shown in Figure 4.25. From the result, it can be seen that the drone can fly the track for 3 laps with an average speed of $2m/s$ and a maximum speed of $2.6m/s$ while an experienced pilot flies the same drone in the same track with an average speed of $2.7m/s$ after several runs of training. Figure 4.25(a) is the filtering result of the position. It should be noted that the filtering result does not coincide with the ground truth curve because of the displacement of the gates. The pose estimation is based on the gates' position on the map. When the gates are displaced, the drone still thinks they are at the position which the map indicates. After the turn, when the drone sees the next gate, which is displaced, it will attribute the misalignment to the prediction error and correct the prediction by means of new detections. With this strategy, our algorithm is robust to the displacement of the gates.



(a) The map of the race track where the gates in the real world are displaced.

(b) The flying data of the first lap.

Figure 4.24: The experiment where the gates are displaced. When the drone sees the next gate after passing through one gate, the filter will start correcting the error caused by the prediction drift and the gate's displacement. Thus, there is a jump in the filtering result.

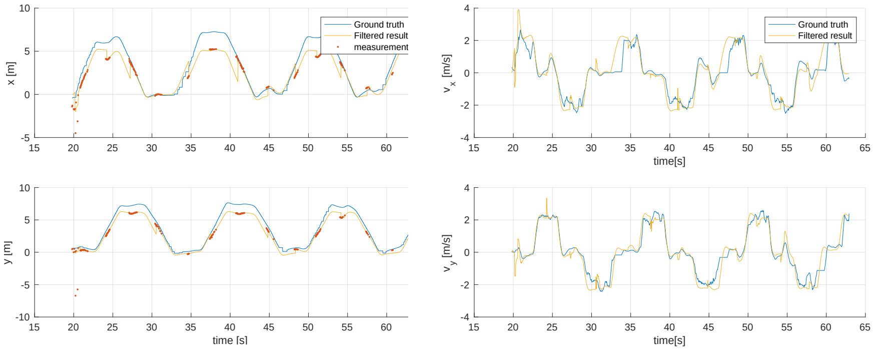
Table 4.5: The position of the gates with displacement

| gate ID | $x_g[m]$ | $y_g[m]$ | $\tilde{x}_g[m]$ | $\tilde{y}_g[m]$ |
|---|---|---|---|---|
| 1 | 5 | 0 | 4 | 0 |
| 2 | 6.5 | 5 | 5 | 5 |
| 3 | 1 | 7 | 1 | 6 |
| 4 | 0 | 1 | 0 | 1 |

### 4.5.4. FLYING EXPERIMENT WITH DIFFERENT ALTITUDE AND MOVING GATE

We also show a more challenging trace track where the height of the gates varies from $0.5m$ to $2.5m$. Also, during the flight, the position of the second gate ($2.5m$) is changed after the drone passes through it. In the next lap, the drone can adapt to the changing position of the gate (Figure 4.26).

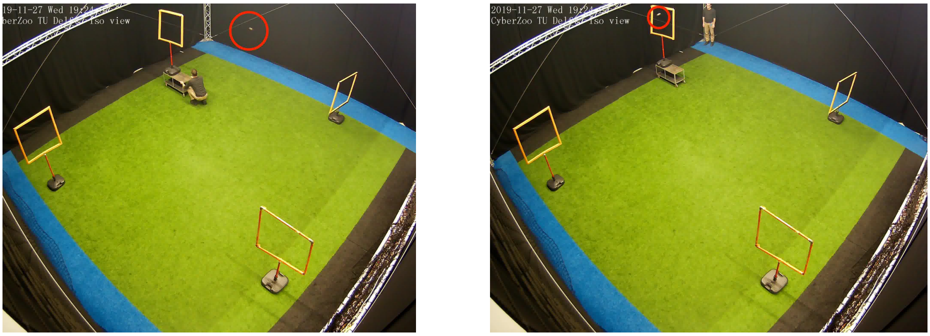
The flight result is shown in Figure 4.27. In this flight, the waypoints are not changed and the gates are deployed without any ground truth measurement. Thus, the estimated position does not coincide with the ground-truth position. It should be noted that the



(a) The position estimation result. It should be noted that the position estimation curve does not coincide with the ground truth curve coming from our motion capture system because the gate displacements.

(b) The velocity estimation result of VML

Figure 4.25: The result of flying the track with the gate displacement.



(a) After the drone passes through the second gate, the gate is moved.

(b) In the next lap, the drone can adapt to the changing position of the gate and fly through it.

Figure 4.26: The flying experiment where the heights of the gates vary from $0.5m$ to $2.5m$. During the flight, the position of the second gate is changed.

height difference between the second gate and the third gate is $2m$. With this altitude change which violates the constant altitude assumption for the prediction error model, the proposed VML is still accurate enough to navigate the drone through the gate.

From the real flight result, we can see that the VML performs well and can navigate the drone through the racing track with high speed even though the gates are displaced. Also, this strategy does not need computationally expensive methods like generic VIO and SLAM. This allows it to be run on a very light-weight flying platform.

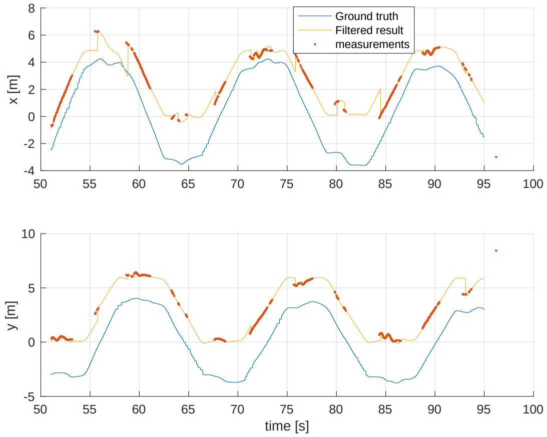


Figure 4.27: The flying result of the drone flying the track with different height and the gate's position changing.

## 4.6. DISCUSSION

In this paper, we proposed a novel state estimation method called Visual Model-predictive Localization which provides navigation information for a 72 gram autonomous racing drone. The algorithm's properties were thoroughly studied in simulation and the feasibility of real-world implementation was shown in challenging real world experiments. Although in this paper VML is used for a specific drone race scenario, this method can be directly used for navigation in other more general scenarios where the sensors have low frequency, temporary failure, outliers and delays. For example, our approach can be directly adopted into an outdoor environment where position measurements are provided by a GPS signal that has a delay, temporary failures and outliers. Just as in our drone race experiments, the proposed approach should be more reliable than a Kalman filter. For indoor flight, we used a common linear drag model for state prediction which does not need a lot of effort and precise equipment to identify. Outdoor flight would require adaptations to this model, for instance such as the ones explained in, e.g., [30].

We implemented our approach by adding a cheap smart camera Jevois to a tiny racing drone Trashcan. With very limited carrying capacity and more complex aerodynamics properties, it is still demonstrated that this light-weight flying platform has the ability to finish the drone race task autonomously. Compared to a regular size racing drone, the Trashcan has more complex aerodynamics and is more sensitive to disturbances. On the other hand, it has faster dynamics which can make maneuvers more agile. More importantly, it is much safer than a regular size racing drone, which may even allow for flying at home. In any case, the present approach represents another direction of the autonomous drone race, which does not need high performance and heavy onboard computers. Also, without computationally expensive navigation methods such as SLAM and VIO, the proposed approach is still able to make the drone navigate autonomously with relatively high speed.

However, the proposed approach still has its limitations. First of all, in this approach, we don't estimate the thrust. Instead, we use a non-changing altitude assumption to approximate the thrust to derive the prediction error model. The simulation and real world experiments have shown that violating this assumption still results in accurate estimation. However, when the racing track will contain more considerable height changes, it may become desirable to estimate the thrust with a model, in order to have a more accurate error model and increase the estimation accuracy, especially in more aggressive flight.

Secondly, the current detection method is sensitive to light conditions. Most failures are caused by the non-detection of the gate. This is a major bottleneck of increasing the speed of the flight. In the future, we will design a gate detection method using deep learning methods to detect the gate in a more complex environment. This deep net can then run on the GPU of the Jevois. Also, higher speeds could be attainable.

Thirdly, in this paper, we mainly focus on the navigation part of the drone. The guidance is only a way-point based method and the controller is a PID controller. To make the drone fly faster, optimal guidance and control methods are needed [31–33]. Another direction is to explore joint estimation for navigation. This will become very useful when one assumes that gates are mostly not displaced. Then, over multiple laps, the drone can get a better idea of where the gates are.

In the future, with the high speed development of computational capacity, when more reliable gate detection and online optimal control are implemented onboard, the speed of this autonomous racing drone can certainly be increased significantly. Compared to regularly sized drones, this tiny flying platform will be able to perform faster and more agile flight. At that time, the proposed VML approach will still be suitable for providing stable state estimation for the drone.

## 4.7. CONCLUSION

In this paper, we presented an efficient Visual Model-predictive Localization (VML) approach to autonomous drone racing. The approach employs a velocity-stable model that predicts lateral accelerations based on attitude estimates from the AHRS. Vision is used for detecting gates in the image, and - by means of their supposed location in the map - for localizing the drone in the coarse global map. Simulation and real-world flight experiments show that VML can provide robust estimates with sparse visual measurements and large outliers. This robust and computationally very efficient approach was tested on an extremely lightweight flying platform, i.e., a Trashcan racing drone with a Jevois camera. In the flight experiments, the Trashcan flew a track of 3 laps with an average speed of $2m/s$ and a maximum speed of $2.6m/s$. To the best of our knowledge, it is the world's smallest autonomous racing drone with a weight 6 times lighter than the currently lightest autonomous racing drone setup, while its velocity is on a par with the currently fastest autonomously flying racing drones seen at the latest IROS autonomous drone race.

## REFERENCES

[1] H. Moon, Y. Sun, J. Baltes, and S. J. Kim, *The iros 2016 competitions [competitions]*, IEEE Robotics & Automation Magazine **24**, 20 (2017).

[2] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors,* in *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (IEEE, 2011) pp. 2520–2525.

[3] D. Mellinger, N. Michael, and V. Kumar, *Trajectory generation and control for precise aggressive maneuvers with quadrotors,* The International Journal of Robotics Research **31**, 664 (2012).

[4] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, *Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu,* IEEE Robotics and Automation Letters **2**, 404 (2017).

[5] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, *Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision,* in *Robotics and Automation (ICRA), 2017 IEEE International Conference on* (IEEE, 2017) pp. 5774–5781.

[6] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos, *Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight,* IEEE Robotics and Automation Letters **3**, 2799 (2018).

[7] B. Morrell, M. Rigter, G. Merewether, R. Reid, R. Thakker, T. Tzanetos, V. Rajur, and G. Chamitoff, *Differential flatness transformations for aggressive quadrotor flight,* in *Robotics and Automation (ICRA), 2018 IEEE International Conference on Robotics and Automation* (IEEE, 2018) pp. 5204–5210.

[8] F. Gao, L. Wang, K. Wang, W. Wu, B. Zhou, L. Han, and S. Shen, *Optimal trajectory generation for quadrotor teach-and-repeat,* IEEE Robotics and Automation Letters (2019).

[9] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, *A direct visual servoing-based framework for the 2016 iros autonomous drone racing challenge,* Journal of Field Robotics **35**, 146 (2018).

[10] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, *et al.*, *Challenges and implemented technologies used in autonomous drone racing,* Intelligent Service Robotics , 1 (2019).

[11] S. Li, M. Ozo, C. De Wagter, and G. de Croon, *Autonomous drone race: A computationally efficient vision-based navigation and control strategy,* arXiv preprint arXiv:1809.05958 (2018).

[12] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, *Deep drone racing: Learning agile flight in dynamic environments,* arXiv preprint arXiv:1806.08548 (2018).

[13] 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,* in *2019 International Conference on Robotics and Automation (ICRA)* (IEEE, 2019) pp. 690–696.

[14] B. Gati, *Open source autopilot for academic research-the paparazzi system,* in *American Control Conference (ACC), 2013*, IEEE (IEEE, Washington, DC, USA, 2013) pp. 1478–1481.

[15] G. Hattenberger, M. Bronz, and M. Gorraz, *Using the paparazzi uav system for scientific research,* in *International Micro Air Vehicle Competition and Conference 2014*, edited by G. de Croon, E. van Kampen, C. D. Wagter, and C. de Visser (Delft, The Netherlands, 2014) pp. 247–252.

[16] L. V. Santana, A. S. Brandao, and M. Sarcinelli-Filho, *Outdoor waypoint navigation with the ar. drone quadrotor,* in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)* (IEEE, 2015) pp. 303–311.

[17] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, *A platform for aerial robotics research and demonstration: The flying machine arena,* Mechatronics **24**, 41 (2014).

[18] M. W. Mueller, M. Hamer, and R. D'Andrea, *Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation,* in *2015 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2015) pp. 1730–1736.

[19] K. McGuire, G. De Croon, C. De Wagter, K. Tuyls, and H. Kappen, *Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone,* IEEE Robotics and Automation Letters **2**, 1070 (2017).

[20] S. Weiss, M. W. Achtelik, M. Chli, and R. Siegwart, *Versatile distributed pose estimation and sensor self-calibration for an autonomous mav,* in *2012 IEEE International Conference on Robotics and Automation* (IEEE, 2012) pp. 31–38.

[21] A. Santamaria-Navarro, G. Loianno, J. Solà, V. Kumar, and J. Andrade-Cetto, *Autonomous navigation of micro aerial vehicles using high-rate and low-cost sensors,* Autonomous robots , 1 (2018).

[22] J. N. Gross, Y. Gu, M. B. Rhudy, S. Gururajan, and M. R. Napolitano, *Flight-test evaluation of sensor fusion algorithms for attitude estimation,* IEEE Transactions on Aerospace and Electronic Systems **48**, 2128 (2012).

[23] M. Faessler, A. Franchi, and D. Scaramuzza, *Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories,* IEEE Robotics and Automation Letters **3**, 620 (2017).

[24] M. A. Fischler and R. C. Bolles, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,* Communications of the ACM **24**, 381 (1981).

[25] G. T. Diderrich, *The kalman filter from the perspective of goldberger—theil estimators,* The American Statistician **39**, 193 (1985).

[26] G. Chang, *Robust kalman filtering based on mahalanobis distance as outlier judging criterion,* Journal of Geodesy **88**, 391 (2014).

[27] Z. Li, G. Chang, J. Gao, J. Wang, and A. Hernandez, *Gps/uwb/mems-imu tightly coupled navigation with improved robust kalman filter,* Advances in Space Research **58**, 2424 (2016).

[28] E. van Horssen, J. van Hooijdonk, D. Antunes, and W. Heemels, *Event-and deadline-driven control of a self-localizing robot with vision-induced delays,* IEEE Transactions on Industrial Electronics (2019).

[29] G. Upton and I. Cook, *Understanding statistics* (Oxford University Press, 1996).

[30] L. Sikkel, G. de Croon, C. De Wagter, and Q. Chu, *A novel online model-based wind estimation approach for quadrotor micro air vehicles using low cost mems imus,* in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2016) pp. 2141–2146.

[31] D. Tailor and D. Izzo, *Learning the optimal state-feedback via supervised imitation learning,* Astrodynamics **3**, 361 (2019).

[32] S. Li, E. Ozturk, C. De Wagter, G. C. de Croon, and D. Izzo, *Aggressive online control of a quadrotor via deep network representations of optimality principles,* arXiv preprint arXiv:1912.07067 (2019).

[33] G. Tang, W. Sun, and K. Hauser, *Learning trajectories for real-time optimal control of quadrotors,* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018) pp. 3620–3625.

# APPENDEX

## KALMAN FILTER'S PREDICTION MODEL

$$\left\{\begin{aligned}
\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} &= \begin{bmatrix}v_x\\ v_y\end{bmatrix}\\
\begin{bmatrix}\dot{v}_x\\ \dot{v}_y\end{bmatrix} &= \begin{bmatrix}\cos\psi^m & -\sin\psi^m\\ \sin\psi^m & \cos\psi^m\end{bmatrix}\{\begin{bmatrix}-g\sin\theta\\ g\cos\phi\end{bmatrix} + \begin{bmatrix}k_x & 0\\ 0 & k_y\end{bmatrix}\begin{bmatrix}\cos\psi^m & \sin\psi^m\\ -\sin\psi^m & \cos\psi^m\end{bmatrix}\begin{bmatrix}v_x\\ v_y\end{bmatrix}\}\\
\begin{bmatrix}\dot{B}_N\\ \dot{B}_E\end{bmatrix} &= \begin{bmatrix}0\\ 0\end{bmatrix}\\
\begin{bmatrix}\phi\\ \theta\end{bmatrix} &= \begin{bmatrix}\phi^m\\ \theta^m\end{bmatrix} + \begin{bmatrix}\cos\psi^m & \sin\psi^m\\ -\sin\psi^m & \cos\psi^m\end{bmatrix}\begin{bmatrix}B_N\\ B_E\end{bmatrix}
\end{aligned}\right. \tag{4.36}$$

The inputs of the system 4.36 is the AHRS reading $\mathbf{u} = [\phi^m, \theta^m, \psi^m]^\mathrm{T}$. The states of the Extended Kalman filter are $\mathbf{X} = [x, y, v_x, v_y, B_N, B_E]^\mathrm{T}$. With the standard Extended Kalman filter procedure list below, the states of the system can be estimated.

## PSEUDOCODES

**Algorithm 3** gate_assignment

```
**Input:**$\Delta\bar{x}_k, \Delta\bar{y}_k$
**Output:**$\bar{x}_k, \bar{y}_k$
**for** $i = 1; i <= gate\_numbers; i++$ **do**
    $\bar{x}^i_k = \cos\psi^i_g \Delta\bar{x}_k - \sin\psi^i_g \Delta\bar{y}_k + x^i_g$
    $\bar{y}^i_k = \sin\psi^i_g \Delta\bar{x}_k + \cos\psi^i_g \Delta\bar{y}_k + y^i_g$
    $\Delta d^i_k = (\bar{x}^i_k - \hat{x}_k)^2 + (\bar{y}^i_k - \hat{y}_k)^2$
**end for**
$j = \mathrm{argmin}_i \, \Delta d^i_k$
$\bar{x}_k = \bar{x}^j_k$
$\bar{y}_k = \bar{y}^j_k$
```

**Algorithm 4** Basic RANSAC Fitting

```
**Input:**$\Delta\mathbf{x}^p_{k-q,k}$, $\Delta\mathbf{t}$
**output:**$\hat{\beta} = \left[\Delta x^p_{k-q}, \Delta v^p_{k-q}\right]$
**for** $i = 1; i <= iterations; i++$ **do**
    **sample_id** $= random\_integers(k-q, k, n^s)$
    $\Delta\mathbf{t}^s = \Delta\mathbf{t}[\mathbf{sample\_id}]$
    $\Delta\mathbf{x}^s_{k-q,k} = \Delta\mathbf{x}^p_{k-q,k}[\mathbf{sample\_id}]$
    $[\Delta x^p_{k-q_i}, \Delta v^p_{k-q_i}] = linear\_regression(\Delta\mathbf{t}^s, \Delta\mathbf{x}^s_{k-q,k})$
    **for** $j = k-q; j < k; j++$ **do**
        $\epsilon_j = \left\| \Delta v^p_{k-q_i}(\Delta t_j - \Delta t_{k-q}) + \Delta x^p_{k-q_i} - \Delta x^p_j \right\|_2$
        **if** $\epsilon_j > \sigma_{th}$ **then**
            $\varepsilon_i += \sigma_{th}$
        **else**
            $\varepsilon_i += \epsilon_j$
        **end if**
    **end for**
    **if** $\varepsilon_i < \varepsilon_{min}$ **then**
        $\varepsilon_{min} = \varepsilon$
        $\Delta x^p_{k-q} = \Delta x^p_{k-q_i}$
        $\Delta v^p_{k-q} = \Delta v^p_{k-q_i}$
    **end if**
**end for**
```

**Algorithm 5** Visual Model-predictive Localization

```
while true do
    $t_{k+i} = current\_time$
    $x_k^p += v_{x\,k}^p T_s$
    $y_k^p += v_{y\,k}^p T_s$
    $v_{x\,k}^p += (-g\tan\theta - c v_{x\,k}^p) T_s$
    $v_{y\,k}^p += (g\tan\phi - c v_{y\,k}^p) T_s$
    $clear\_old\_elements\_in\_queue()$
    if flagNewPoseEstimation then
        $queue.front{+}{+}$
        $queue.\mathbf{time}[queue.front] = t_{k+i}$
        $queue.\mathbf{x}_k^p[queue.front] = x_k^p$
        $queue.\mathbf{y}_k^p[queue.front] = y_k^p$
        $queue.\bar{\mathbf{x}}[queue.front] = \bar{x}_k$
        $queue.\bar{\mathbf{y}}[queue.front] = \bar{y}_k$
        $queue.size{+}{+}$
        if $queue.size > N_{fit}$ then
            $[\Delta x_{k-q}^p, \Delta v_x{}_{k-q}^p, \Delta y_{k-q}^p, \Delta v_y{}_{k-q}^p] = filter\_correct()$
            $t_k = t_{k+i}$
        end if
    end if
    $\hat{x}_k = x_k^p - \Delta x_{k-q}^p + (t_{k+i} - t_k)\Delta v_x{}_{k-q}^p$
    $\hat{y}_k = y_k^p - \Delta y_{k-q}^p + (t_{k+i} - t_k)\Delta v_y{}_{k-q}^p$
    $\hat{v}_{x\,k} = v_{x\,k}^p - \Delta v_x{}_{k-q}^p$
    $\hat{v}_{y\,k} = v_{y\,k}^p - \Delta v_y{}_{k-q}^p$
end while
```

**Algorithm 6** filter_correct

```
Output: $\Delta x^p_{k-q}, \Delta {v_x}^p_{k-q}, \Delta y^p_{k-q}, \Delta {v_y}^p_{k-q}$
for $i = 1; i <= queue.size; i++$ do
    $\Delta \mathbf{t}_i = queue.\mathbf{time}.[newest\_item\_id] - queue.\mathbf{time}[i]$
    $\Delta \mathbf{x}^p_i = queue.\mathbf{x}^p[i] - queue.\bar{\mathbf{x}}[i]$
    $\Delta \mathbf{y}^p_i = queue.\mathbf{y}^p[i] - queue.\bar{\mathbf{y}}[i]$
end for
$[\Delta x^p_{k-q}, \Delta {v_x}^p_{k-q}] = linear\_regression(\Delta \mathbf{t}, \Delta \mathbf{x}^p)$
$[\Delta y^p_{k-q}, \Delta {v_y}^p_{k-q}] = linear\_regression(\Delta \mathbf{t}, \Delta \mathbf{y}^p)$
```

**Algorithm 7** $flight\_plan$

```
Output: $x^r, y^r, z^r, \psi^r$
if $(\hat{x}_k - \mathbf{wp}_x[waypoint\_id])^2 + (\hat{y}_k - \mathbf{wp}_y[waypoint\_id])^2 < D_{switch\_wp}$ then
    $waypoint\_id++$
end if
if $(\hat{x}_k - \mathbf{wp}_x[waypoint\_id])^2 + (\hat{y}_k - \mathbf{wp}_y[waypoint\_id])^2 < D_{turn}$ then
    $\psi_{sp} = \mathbf{wp}_\psi[waypoint\_id + 1])$
    $r_{cmd} = k_r(\psi_{sp} - \psi^r)$
    $\psi^r += r_{cmd}$
end if
$x^r = \mathbf{wp}_x[waypoint\_id])$
$y^r = \mathbf{wp}_y[waypoint\_id])$
$z^r = \mathbf{wp}_z[waypoint\_id])$
$\psi^r = \psi_{ref}$
```

# 5

# AGGRESSIVE ONLINE CONTROL OF A QUADROTOR VIA DEEP NETWORK REPRESENTATIONS OF OPTIMALITY PRINCIPLES

---

This chapter is based on the following article:
**Li S**, Öztürk E, De Wagter C, de Croon GC, Izzo D. Aggressive online control of a quadrotor via deep network representations of optimality principles. In2020 IEEE International Conference on Robotics and Automation (ICRA) 2020 May 31 (pp. 6282-6287). IEEE.

## 5.1. INTRODUCTION

A major challenge in the field of drone control is to achieve aggressive autonomous flight. In terms of control, much research focuses on designing controllers which can track a reference guidance trajectory also when considering unmodeled dynamics, nonlinearities and disturbances which become significant when the maneuver of the drone gets aggressive [1–4]. In terms of guidance, multiple methods varying from a simple setpoint to high order polynomial trajectory generation methods have shown their feasibility in guiding a quadrotor to the desired target including some time optimality principles.

Two fundamentally different approaches are used to obtain aggressive quadrotor trajectories. The first one is differential flatness based trajectory generation and control [5, 6]. This method is able to generate aggressive trajectories for quadrotors (based on a minimum-time polynomial guidance), and hence it is widely used in real quadrotor flights. However, the resulting trajectory can be far from being truly time optimal.

The second approach uses optimal control theory to find and fly a trajectory that incorporates the required optimality principles. Due to the time-consuming nature of this calculation, this method is unsuitable for an online implementation [7, 8]. Several methods have been proposed to address this, where the most common is to represent the system dynamics as a series of simpler linear systems with analytical solutions [9, 10]. Unfortunately, this simplification can lead to an inaccurate representation of the nonlinear response of the system and can thus negatively impact performance. An alternative approach is to find and use, on-board, a sub-optimal solution instead. For example, by using the result of the first iteration of a nonlinear programming (NLP) solver [11] which, although incomplete, is faster to compute.

In recent years, leveraging significant advances in machine learning techniques and in particular in artificial neural networks, a number of new methods have been proposed relevant to the aggressive control of quadrotor trajectories. Reference trajectories have been optimized using DNNs [12], waypoint tracking has been achieved by means of reinforcement learning [13] and trajectory tracking using RBFNN [14]. Tang et al. [15] combine both optimal control and machine learning. Their experimental results have shown that a trained neural network can predict an optimal trajectory to the target point, which can then be tracked using PID control. This work is an important step towards online optimal control, however the main computation is done on a workstation (i.e. not onboard) and, since a PID controller is introduced to track the reference, there are delays during the tracking as a result of which the controls may violate the constraints due to the feedback term. In a different context (i.e. spacecraft landing and mass optimal control) Sanchez et al. [16] successfully introduced the use of imitation learning of optimal controls to train DNNs capable of safely steering the system to desired target positions. Following that work, Tailor and Izzo [17] made an extensive study of the technique on simulated drone dynamics and Izzo et al. [18] introduced the term G&CNets (guidance and control networks) to refer to these networks and showed how to study the stability of the resulting trajectories analytically via differential algebraic techniques.

In this chapter, extending previous work on G&CNets, we present an approach for the on-board optimal control problem of a quadrotor that does not need a PID controller to track the trajectory and we test it during real flights. In our approach, 250,000 optimal trajectories are generated offline. Then, a G&CNet—which is a neural network trained to learn this dataset—is computed. Instead of predicting an optimal trajectory as the work in [15], G&CNet predicts the required optimal thrust directly which will be transferred to the optimal pitch rate acceleration and sent to the controller, and thus can be seen in the context of non-Linear MPC. Since the work of [15] is difficult to reproduce, we made the comparison between G&CNet and the differential flatness based trajectory generation and control (DiffG&C) in simulation. The simulation results show that the proposed G&CNet can guide the drone to the target points much faster while satisfying optimality principles. Finally, the developed G&CNet and DiffG&C controllers are verified in real-world flight tests where the results show that the on-board G&CNet can guide the drone to the target with a resulting real-time trajectory that is very close to the theoretical optimal solution.

## 5.2. DESIGN OF THE G&CNET

### 5.2.1. THE DYNAMICAL SYSTEM

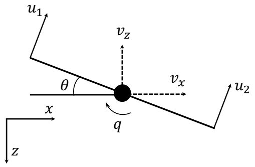


Figure 5.1: Axis definition

Specifying the state of a quadrotor in the $xoz$ plane as

$$\mathbf{x} = \begin{bmatrix} x & z & v_x & v_z & \theta & q \end{bmatrix} \tag{5.1}$$

as defined in Figure 5.1, the dynamical model for which we compute the optimal control is:

$$\mathbf{f}(\mathbf{x},\mathbf{u}) = \begin{bmatrix} \dot{x} = v_x \\ \dot{z} = v_z \\ \dot{v}_x = -\left[u_\Sigma \frac{\Delta F}{m} + 2\frac{\underline{F}}{m}\right]\sin\theta - \beta v_x \\ \dot{v}_z = -\left[u_\Sigma \frac{\Delta F}{m} + 2\frac{\underline{F}}{m}\right]\cos\theta + g_0 - \beta v_z \\ \dot{\theta} = q \\ \dot{q} = \frac{L}{I_{yy}}\Delta F(u_2 - u_1) \end{bmatrix} \tag{5.2}$$

where $\Delta F = \overline{F} - \underline{F} = 0.59$ N is the range of the thrust magnitude, $\overline{F} = 2.35$ N is the maximum thrust, $\underline{F} = 1.76$ N is the minimum thrust, $\beta = 0.5$ is the drag coefficient, $m = 0.389$ kg is the quadrotor mass, $L = 0.08$ m is the length of the quadrotor, $I_{yy} = 0.001242$ kg m$^2$ is the moment of inertia about the y-axis, $\mathbf{u} = [u_1, u_2] \in [0,1]$ are the left and right throttles respectively, and $u_\Sigma = (u_1 + u_2)$.

### 5.2.2. THE OPTIMISATION PROBLEM

The cost function we need to minimise for the optimal controls is:

$$J(\epsilon, t_f, \mathbf{u}(t)) = (1-\epsilon)t_f + \epsilon \int_0^{t_f} (u_1(t)^2 + u_2(t)^2)\mathrm{d}t \tag{5.3}$$

where $\epsilon \in [0,1]$ is a hybridisation parameter. When $\epsilon = 0$, the cost function is exactly time-optimal, and when $\epsilon = 1$, the cost function is exactly power-optimal. With this parameter we are able to generate datasets from time-optimal to power-optimal continuously. Similar to the weighting factor of [19], we set $\epsilon$ close to zero ($\epsilon = 0.2$) to improve the numerical convergence of the problem and avoid difficult to track control profiles. We trained two networks for $\epsilon = 0.5$ and $\epsilon = 0.2$ in order to compare how well the quadrotor is able to track and execute the optimal controls with differing degrees of aggressiveness. As we are more interested in time-optimal guidance and control, the dataset and training details focus only on the $\epsilon = 0.2$ controller, but the same arguments and methods apply to the $\epsilon = 0.5$ controller.

$$\begin{aligned} \underset{\mathbf{u}, t_f}{\text{minimize}} \quad & J(\epsilon, t_f, \mathbf{u}(t)) \\ \text{subject to} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \ \forall t \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(t_f) = \mathbf{0} \end{aligned} \tag{5.4}$$

Using a direct transcription and collocation method (Hermite-Simpson transcription), the trajectory optimisation problem is transformed into an NLP problem [19]. The AMPL modelling language was used to specify the NLP problem which was then solved via SNOPT, an SQP NLP solver. Solving for 250,000 trajectories with initial states, $\mathbf{x}_0$, drawn uniformly from $x_0 \in [-10, 10]$ m, $z_0 \in [-10, 10]$ m, $v_{x0} \in [-5, 5]$ m s$^{-1}$, $v_{z0} \in [-5, 5]$ m s$^{-1}$, $\theta_0 \in [-\pi/3, \pi/3]$ rad, and $\omega_0 \in [-0.01, 0.01]$ rad s$^{-1}$, we obtain a database of state-control pairs of the form:

$$\begin{aligned} \kappa_i = \left(\mathbf{x}_j^{(i)}, \mathbf{u}_j^{(i)}\right)_{j=1}^{K} \quad & \text{where} \\ \mathbf{x}_1^{(i)} = \mathbf{x}_0^{(i)}, \mathbf{x}_J^{(i)} = \mathbf{0} \quad & i = 1, \dots, M \end{aligned} \tag{5.5}$$

where $i$ indexes the trajectories and $K = 81$ is the number of grid points in the Hermite-Simpson transcription [19]. We solved for 250,000 trajectories of which 214,210 converged, and following an 80-10-10 split, these trajectories were split into training, validation and test sets. Overall, this translates to 13,880,808 state-control pairs that the network was trained on, and 1,735,101 that the network was tested on.

### 5.2.3. NETWORK ARCHITECTURE AND TRAINING

We construct neural network architectures in the same manner as [19] with 3 layers, 100 hidden units with softplus activation functions, and sigmoid activation functions for the output controls.

Thus we train on the loss function:

$$l = \left\| \mathcal{N}(\mathbf{x}) - \mathbf{u}^* \right\|^2 \tag{5.6}$$

with a minibatch size of 256 and a starting learning rate of $10^{-3}$ using the Adam optimizer. For further details on network training and construction, refer to [19]. From this training, the $\epsilon = 0.2$ network was able to achieve a mean absolute error (MAE) of 0.0105 for $u_1$ and 0.0107 for $u_2$ on the training set, and a MAE of 0.0108 for $u_1$ and 0.0109 for $u_2$ on the test set.

## 5.3. SIMULATION RESULT AND ANALYSIS

In this section, we analyse the theoretical performance of the proposed optimal controller. First we discuss the simulated stability characteristics of the G&CNet($\epsilon = 0.2$) controller. Then we introduce the aforementioned DiffG&C as a benchmark controller. Finally, we detail the simulation of both methods and present a comparison between simulations.

### 5.3.1. STABILITY OF NEURAL NETWORK CONTROLLER

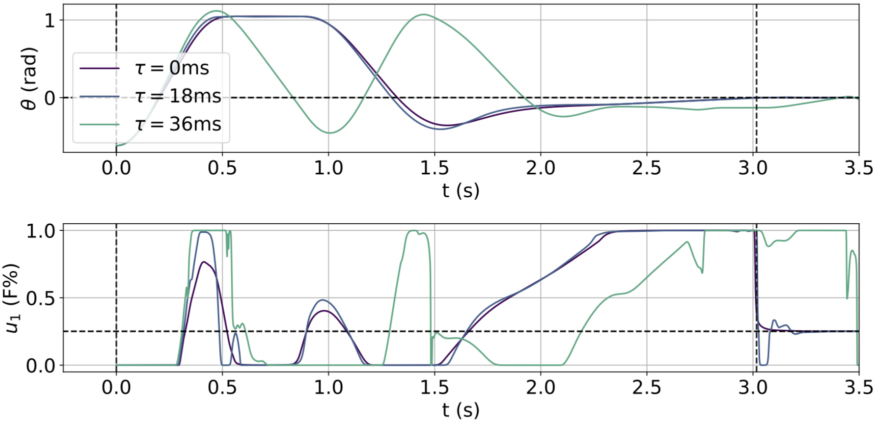


Figure 5.2: Pitch (top) and the left thrust (bottom) during a G&CNet driven trajectory simulated with control delays of $0ms$, $18ms$ and $36ms$. The vertical dashed lines show the initial and final time of the true optimal trajectory. The horizontal dashed lines show the target final states: $\theta(t_f) = 0.0$ and $u_1(t_f) = u_{\text{hover}}$.

One of the foremost important things is the stability of any controller used on the quadrotor as an unstable controller can lead to failure. The primary stability concerns arise due to the fact that in a real quadrotor there is a measurable delay between the computation of the controls, the state given to the controller and the controller response which arises due to factors such as the time taken to compute the state, and the inertia of the rotors. This delay can be modeled by a fixed time between the command and the execution of the control command:

$$\mathbf{u}(t) = \mathcal{N}(\mathbf{x}(t-\tau)) \tag{5.7}$$

where $\tau$ is the time delay. Using the methods developed in [18], we find that the stability margin of the G&CNet($\epsilon = 0.2$) controller is $\tau_s = 63.8ms$. Although this stability margin is high, it mostly provides information as to the hovering stability of the quadrotor, but we are more interested in the general stability during flight. Figure 5.2 shows the effect of an increasing time delay on the G&CNet($\epsilon = 0.2$) controller left thrust and pitch for delays of $\tau = 0ms$, $\tau = 18ms$ and $\tau = 36ms$. Here we see that, as the delay increases, the controller becomes increasingly unstable up to the point where it is no longer able to track the optimal trajectory nor hover in the final state.

### 5.3.2. DIFFERENTIAL FLATNESS BASED AGGRESSIVE TRAJECTORY GENERATION AND CONTROL (DIFFG&C)

A commonly used aggressive trajectory generation method is to use high order polynomials $P(t) = \mathbf{p}^{\mathrm{T}}\mathbf{t}$ to connect the initial point, the waypoints and the final point [5, 6]. Thanks to the differential flatness properties of the quadrotor, the thrust on each rotor can be directly related to the $4^{th}$ order derivative of the position curves $\mathbf{u} = \mathbf{f}(\mathbf{p}, t)$ [5, 20]. In particular, in this method, we use the same kinematics model as the reference [20] with Bebop's drag coefficient ($\mathbf{D} = \mathrm{diag}(-\beta, -\beta, -\beta)$), mass $m = 0.389$ kg and length $L = 0.08$ m.

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{v} \\ \dot{\mathbf{v}} = \mathbf{g} + \mathbf{T} + \mathbf{R}^{\mathrm{T}}\mathbf{D}\mathbf{R}\mathbf{v} \\ \dot{\Phi} = \mathbf{R}'\mathbf{q} \end{cases} \tag{5.8}$$

where thrust $\mathbf{T} = [0, 0, T]^{\mathrm{T}}$ and body rate $\mathbf{q} = [p, q, r]^{\mathrm{T}}$ are the inputs of the system with the assumption that the low-level acceleration controller and rate controller can track the reference well. Equation 5.9 is used to check the feasibility of the thrust each rotor can provide.

$$\dot{\mathbf{q}} = \mathbf{I}^{-1}(\tau - \mathbf{q} \times \mathbf{I}\mathbf{q}) \tag{5.9}$$

From the computed polynomial trajectory, the body rate $\mathbf{q}$ and the rotor thrusts can be determined. For a given arrival time $t_f$, the best trajectory connecting two states is the one with minimal snap. By decreasing the arrival time $t_f$ until the constraints are violated, the polynomial trajectory with minimum arrival time and minimal snap can be found.

$$\min_{t_f}\left\{\min_{\mathbf{p}} \int_0^{t_f} P^{(4)}(t)\mathrm{d}t\right\} = \min_{t_f}\{\min_{\mathbf{p}} \mathbf{p}^{\mathrm{T}}\mathbf{Q}\mathbf{p}\} \tag{5.10}$$

$$\text{s.t. } \mathbf{A}\mathbf{p} = \mathbf{b} \tag{5.11}$$

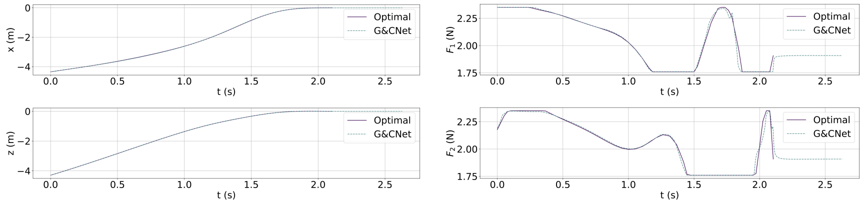
$$\mathbf{f}(\mathbf{p}, t) < \mathbf{c} \tag{5.12}$$

where (5.10) is the optimization target, the integral of the $4^{th}$ order derivative of the polynomial which can be written as a quadratic form. Equation 5.11 is the constraints of the polynomial and (5.12) gives the input constraints. The readers are referred to [6] for the detailed derivation of matrix $\mathbf{Q}$. The algorithm is listed below

The feed-forward control inputs are computed from the polynomial trajectories and a feedback PID controller is used to compensate for disturbance. The readers are referred to [20] for further details on the controller implementation.

**Algorithm 8** The pseudocode of DiffG&C

```
procedure DIFF_CONTROL_GUIDANCE($t_f$, $\mathbf{b}$, $\mathbf{c}$)
    while $\mathbf{f}(\mathbf{p}, t) < \mathbf{c}$ do                    ▷ check feasibility
        $\mathbf{p}^* = \mathbf{p}$
        $t_f = t_f - \Delta t$                                           ▷ minimise time
        $\min_{\mathbf{p}} \mathbf{p}^T \mathbf{Q} \mathbf{p}$   s.t. $\mathbf{A}\mathbf{p} = \mathbf{b}$    ▷ gradient descent
    end while
return $t_f$, $\mathbf{p}^*$
end procedure
```



(a) A simulated trajectory using G&CNet($\epsilon = 0.1$). (b) Force of the front rotors and the rear rotors along the simulated trajectory.

Figure 5.3: An example simulation of G&CNet($\epsilon = 0.2$). In each simulation step, the controller receives $x$,$z$,$v_x$,$v_z$,$\theta$,$q$ and outputs the thrust command of the front rotors and the rear rotors. The desired total thrust $T$ and rate acceleration $\dot{q}$ are calculated by (5.13) and sent to model 5.2 for integration.

We only investigate the movement in the $xoz$ plane by setting any movement in the $y$ direction to 0. This way, the model given by (5.8) can be simplified to the model in (5.2).

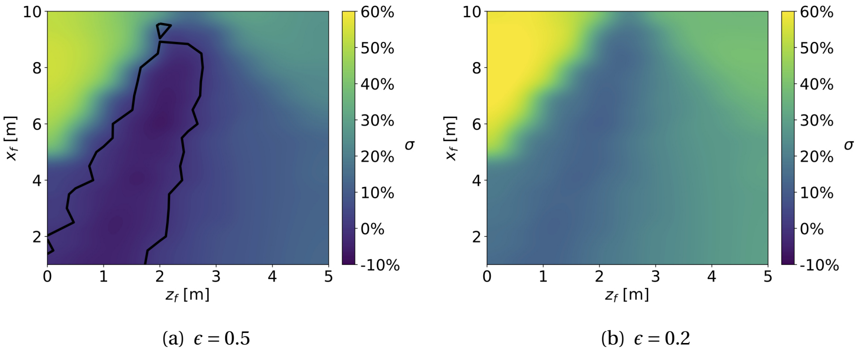
### 5.3.3. SIMULATION OF THE G&CNET CONTROLLER

In this simulation, we use the model from (5.2) as our dynamical model with the rate acceleration $\dot{q}$ and total thrust $T$ as the inputs. The reason is that on the real drone, there are different low-level controllers which can track the thrust and the rate acceleration accurately, one of which is the incremental nonlinear dynamic inversion controller (INDI) [1]. We calculate the desired thrust and rate acceleration command from the G&CNet controller outputs using Eq. (5.13)

$$\begin{cases} \dot{q}_{cmd} = \frac{(u_2 - u_1)\Delta F L}{I_{yy}} \\ T_{cmd} = \frac{(u_1 + u_2)\Delta F + 2\underline{F}}{m} \end{cases} \tag{5.13}$$

### 5.3.4. COMPARISON BETWEEN DIFFG&C AND G&CNET

In this section, a comparison is made in simulation between DiffG&C and G&CNet. The time required by the drone to reach the target is used to derive a performance index. In each trial, the initial position of the drone is set to be $[x_0, z_0] = [0m, 2.5m]$ and the same target $x_f \in [1, 10]$, $z_f \in [0, 5]$ is set for both controllers. To quantify the performance of a



(a) $\epsilon = 0.5$ (b) $\epsilon = 0.2$

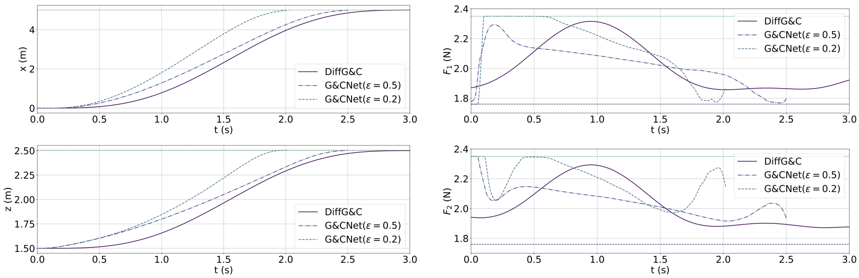
Figure 5.4: Comparison of arrival time between DiffG&C and G&CNet. Despite power optimality being weighted equally to time optimality, G&CNet($\epsilon = 0.5$) can, in most cases, steer the drone to the target points in less time than DiffG&C (the black line shows the region border where G&CNet outperforms DiffG&C). On the other hand, G&CNet($\epsilon = 0.2$) is always faster than DiffG&C.

method, we introduce an index $\sigma$:

$$\sigma = \frac{t_f^{DiffG\&C} - t_f^{G\&CNet}}{t_f^{DiffG\&C}} \tag{5.14}$$

where $t_f^*$ is the arrival time of each controller. When $\sigma > 0$, the G&CNet controller is faster than DiffG&C and vice versa. Figure 5.4 gives the simulation results of multiple target points with $\epsilon = 0.2$ and $\epsilon = 0.5$. From Figure 5.4(a), it can be seen that, in most cases, G&CNet($\epsilon = 0.5$) has a shorter arrival time than DiffG&C outside the region delineated by the black border, and in this region the arrival time is within 10% of DiffG&C. As seen in Figure 5.4(b), with G&CNet($\epsilon = 0.2$), the arrival time is always shorter and up to 60% faster than DiffG&C.

Figure 5.5 shows a comparison plot of the trajectories and controls of DiffG&C, G&CNet ($\epsilon = 0.5$) and G&CNet($\epsilon = 0.2$). It can be seen that all three controllers reach the target, but the control profiles and arrival times differ significantly. With DiffG&C, due to the polynomial representation of trajectories, the quadrotor inputs cannot be fully utilised, and thus the time-optimality cannot be guaranteed. On the other hand, G&CNets are able to saturate the inputs and arrive at a similar or smaller time.



(a) (b)

Figure 5.5: An example of comparison between DiffG&C and G&CNet($\epsilon = 0.5$) when $x_f = 5$, $z_f = 2.5$.

# 5.4. EXPERIMENT SETUP AND RESULT

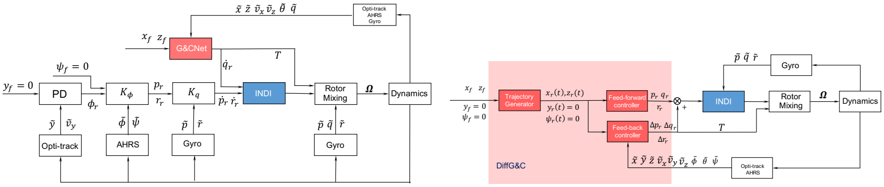
In this section we show the experimental setup for real-world flights and the flight performance of each method.

## 5.4.1. EXPERIMENT SETUP

To verify the proposed G&CNet, we use a commercial Parrot Bebop 1 as our flying platform (Figure 5.6). The software is fully replaced by an open-source autopilot, Paparazzi-UAV. This autopilot provides full access to the raw sensor data and rotor commands. In this experiment, the position and velocity feedback are from Opti-track motion capture system. The attitude estimation is from an on-board complementary filter, which is inevitably biased. The angular rate estimation is from the on-board gyroscope. The control architecture is shown in Figure 5.7. For G&CNet, the lateral movement and heading are controlled by the original outer-loop PID controller and inner-loop INDI controller to keep $y = 0$ and $\psi = 0°$. The maneuver on the vertical plane is taken over by the proposed G&CNet. In each control update, G&CNet receives the state estimations and outputs the desired pitch acceleration $\dot{q}$ and the thrust $T$. For the benchmark DiffG&C, after the trajectory is generated, the desired angular rate $\mathbf{q}$ can be directly calculated. Then a feedback controller is used to compensate the deviation caused by the model inaccuracy, and the state estimation bias.

In the real-world flight tests, we test 3 controllers which are DiffG&C, G&CNet($\epsilon = 0.5$) and G&CNet($\epsilon = 0.2$) respectively. For each controller, the start position is set to be $\mathbf{x}_0 = [0m, 1.5m]^{\mathrm{T}}$ and 3 targets which are $\mathbf{x_f^1} = [5m, 2.5m]^{\mathrm{T}}$,$\mathbf{x_f^2} = [5m, 1.5m]^{\mathrm{T}}$ and $\mathbf{x_f^3} =$

Figure 5.6: A Parrot Bebop 1 is used as the flying platform. The original autopilot is fully replaced by an open-source autopilot called Paparazzi UAV.



(a) The control structure of G&CNet. A PD controller and an INDI controller are used to keep the quadrotor at $y = 0m$ and $\psi = 0°$.

(b) The control structure of DiffG&C. The feed-forward signal is directly computed from generated trajectories. A feedback controller is used to correct for deviations.

Figure 5.7: The control structure of the proposed G&CNet and the benchmark DiffG&C.

$[5m, 0.5m]^{\mathrm{T}}$ are set to be tested. For each target, we have 10 independent flights. To evaluate the performance of one controller, we have 2 indices which are average arrival time $\Delta\bar{t}_*$ and average tracking error $\Delta\bar{x}_*$ defined by

$$\Delta\bar{t}_* = \frac{\sum_i^N \Delta t_*^i}{N} \tag{5.15}$$

$$\Delta\bar{x}_* = \frac{\sum_{i=1}^N \sum_{j=1}^{n_i} \left\| \hat{\mathbf{x}}_*^{i,j} - \mathbf{x}_r^{i,j}{}_* \right\|}{\sum_{i=1}^N n_i} \tag{5.16}$$

where $\Delta t_*^i$ is the arrival time of $i^{th}$ flight of method $*$. $N$ is the number of the flight of one controller, which is 10 in our case. $\hat{\mathbf{x}}_*^{i,j}$ is the position of $i^{th}$ flight's $j^{th}$ sample measured by the Opti-track system. $\mathbf{x}_r^{i,j}{}_*$ is the corresponding position reference. It should be noted that in DiffG&C, $\mathbf{x}_r$ is the reference trajectory while in G&CNet, it is the simulated trajectory.

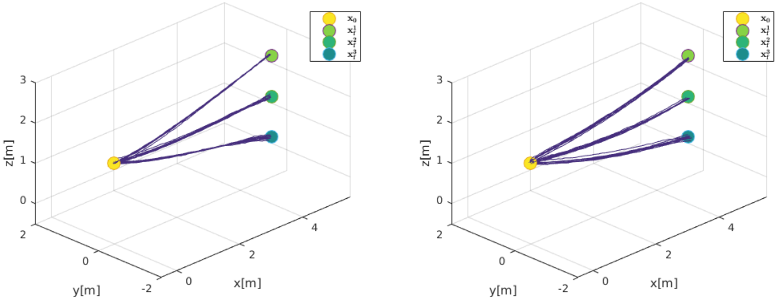
### 5.4.2. EXPERIMENT RESULT

The experiment is set up as described in the previous section and we have 90 flights in total (3 controllers × 3 targets × 10 flights, depicted in Figure 5.8). The average arrival time is listed in Table 5.1 and the average tracking error is listed in Table 5.2.

Table 5.1: Average arrival time $\Delta\bar{t}_*$ to targets $\mathbf{x}_\mathbf{f}^\mathbf{i}$

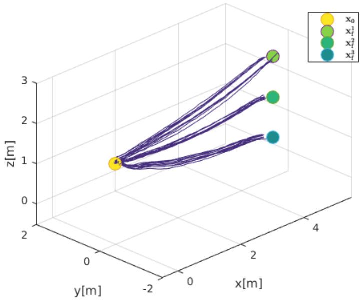
| Controller | $\mathbf{x}_\mathbf{f}^\mathbf{1}$ | $\mathbf{x}_\mathbf{f}^\mathbf{2}$ | $\mathbf{x}_\mathbf{f}^\mathbf{3}$ |
|---|---|---|---|
| DiffG&C | $2.63s \pm 0.05s$ | $2.18s \pm 0.02s$ | $2.10s \pm 0.04s$ |
| G&CNet(0.5) | $2.36s \pm 0.02s$ | $2.20s \pm 0.02s$ | $2.13s \pm 0.01s$ |
| G&CNet(0.2) | $1.96s \pm 0.03s$ | $1.88s \pm 0.03s$ | $1.91s \pm 0.04s$ |

From Table 5.1, it can be seen that when the target is set to $\mathbf{x}_\mathbf{f}^\mathbf{1}$, G&CNet($\epsilon = 0.5$) reaches the target in a shorter time DiffG&C, whereas for targets $\mathbf{x}_\mathbf{f}^\mathbf{2}$ and $\mathbf{x}_\mathbf{f}^\mathbf{3}$, it is on par with the benchmark. On the other hand, G&CNet($\epsilon = 0.2$) always reaches the target in



(a) DiffG&C

(b) GCNet($\epsilon = 0.5$)



(c) GCNet($\epsilon = 0.2$)

Figure 5.8: The real-world flight data of different controllers to different targets

faster time. These experimental results confirm the simulation results that were obtained in Section 5.3.

In terms of tracking error, DiffG&C has the smallest tracking error $\Delta\bar{x}$ followed by G&CNet($\epsilon = 0.5$), and finally G&CNet($\epsilon = 0.2$). We find that G&CNet($\epsilon = 0.5$) outperforms G&CNet($\epsilon = 0.2$) in terms of the tracking error. This can be attributed to the fact that a lower $\epsilon$ corresponds to a more aggressive trajectory and, in turn, a high-frequency high amplitude changes of the inputs. As mentioned in Section 5.2, this is difficult for the quadrotor to track due to the inertial properties of its rotors.

Table 5.2: Average tracking error $\Delta\bar{x}_*$ to targets $\mathbf{x_f^i}$

| **Controller** | $\mathbf{x_f^1}$ | $\mathbf{x_f^2}$ | $\mathbf{x_f^3}$ |
|---|---|---|---|
| DiffG&C | $0.06m$ | $0.07m$ | $0.07m$ |
| G&CNet($\epsilon = 0.5$) | $0.13m$ | $0.09m$ | $0.10m$ |
| G&CNet($\epsilon = 0.2$) | $0.17m$ | $0.15m$ | $0.28m$ |

## 5.5. CONCLUSIONS

We have proposed G&CNet as a novel online optimal controller for quadrotors that removes the need for expensive real-time optimal trajectory generation by learning a deep neural representation of the optimal state-control mapping. We have demonstrated, both in simulation and with real-world flight tests, that G&CNets are not only feasible for this purpose, but also competitive with a commonly used method, DiffG&C. Our results indicate that a G&CNet weighting equally power and time optimality ($\epsilon = 0.5$) is, at worst, 10% slower than DiffG&C and faster most of times while a G&CNet aggressively biased towards time optimality ($\epsilon = 0.2$) is always considerably faster by up to 60%.

There are many avenues of exploration available. Future work can focus on adding the actuator model into the optimal control problem thus eliminating the issue of difficult to track bang-bang controls for the rotors. A further extension of our work would be to implement the optimal control problem in the full 3-dimensional model thus potentially adding more interesting manoeuvre capabilities to the quadrotor. Additionally, the network could be trained to achieve a nonzero velocity in the final state in preparation for consecutive manoeuvres.

## REFERENCES

[1] E. J. Smeur, Q. Chu, and G. C. de Croon, *Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles,* Journal of Guidance, Control, and Dynamics (2015).

[2] E. Tal and S. Karaman, *Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness,* in *2018 IEEE Conference on Decision and Control (CDC)* (IEEE, 2018) pp. 4282–4288.

[3] S. Sun, C. C. de Visser, and Q. Chu, *Quadrotor gray-box model identification from high-speed flight data,* Journal of Aircraft **56**, 645 (2019).

[4] S. Sun and C. de Visser, *Aerodynamic model identification of a quadrotor subjected to rotor failures in the high-speed flight regime,* IEEE Robotics and Automation Letters **4**, 3868 (2019).

[5] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors,* in *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (IEEE, 2011) pp. 2520–2525.

[6] A. Bry, C. Richter, A. Bachrach, and N. Roy, *Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments,* The International Journal of Robotics Research **34**, 969 (2015).

[7] M. Hehn, R. Ritz, and R. D'Andrea, *Performance benchmarking of quadrotor systems using time-optimal control,* Autonomous Robots **33**, 69 (2012).

[8] F. Morbidi, R. Cano, and D. Lara, *Minimum-energy path generation for a quadrotor uav,* in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2016) pp. 1492–1498.

[9] M. Hehn and R. D'Andrea, *Quadrocopter trajectory generation and control,* IFAC proceedings Volumes **44**, 1485 (2011).

[10] O. Santos, H. Romero, S. Salazar, O. García-Pérez, and R. Lozano, *Optimized discrete control law for quadrotor stabilization: Experimental results,* Journal of Intelligent & Robotic Systems **84**, 67 (2016).

[11] M. Geisert and N. Mansard, *Trajectory generation for quadrotor based systems using numerical optimal control,* in *2016 IEEE international conference on robotics and automation (ICRA)* (IEEE, 2016) pp. 2958–2964.

[12] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, *Deep neural networks for improved, impromptu trajectory tracking of quadrotors,* in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017) pp. 5183–5189.

[13] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, *Control of a quadrotor with reinforcement learning,* IEEE Robotics and Automation Letters **2**, 2096 (2017).

[14] S. Li, Y. Wang, J. Tan, and Y. Zheng, *Adaptive rbfnns/integral sliding mode control for a quadrotor aircraft,* Neurocomputing **216**, 126 (2016).

[15] G. Tang, W. Sun, and K. Hauser, *Learning trajectories for real-time optimal control of quadrotors,* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018) pp. 3620–3625.

[16] C. Sánchez-Sánchez and D. Izzo, *Real-time optimal control via deep neural networks: study on landing problems,* Journal of Guidance, Control, and Dynamics **41**, 1122 (2018).

[17] D. Tailor and D. Izzo, *Learning the optimal state-feedback via supervised imitation learning,* Astrodynamics (2019), 10.1007/s42064-019-0054-0.

[18] D. Izzo, D. Tailor, and T. Vasileiou, *On the stability analysis of deep neural network representations of an optimal state-feedback,* arXiv e-prints , arXiv:1812.02532 (2018), arXiv:1812.02532 [cs.NE] .

[19] D. Tailor and D. Izzo, *Learning the optimal state-feedback via supervised imitation learning,* arXiv e-prints , arXiv:1901.02369 (2019), arXiv:1901.02369 [cs.LG] .

[20] M. Faessler, A. Franchi, and D. Scaramuzza, *Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories,* IEEE Robotics and Automation Letters **3**, 620 (2017).

# 6

# CONCLUSION

## 6.1. RESEARCH QUESTIONS

Recall the first research question raised in the introduction:

> RESEARCH QUESTION 1
>
> How does a drone fly through a racing track fully autonomously using only onboard resources?

In Chapter 2, a systematic strategy for autonomous drone racing was presented and tested in real world. A commercial Bebop 1 quadrotor was used as the flying platform to demonstrate the proposed strategy. Firstly, a gate detection method 'snake gate' was developed and an analysis of the Snake gate detection's performance was discussed in detail. Next, a Kalman filter fusing the visual detections and onboard IMU readings was developed. The flying results showed that the proposed Kalman filter can provide accurate state estimation. Then, two controllers were designed for the drone to fly through the racing track. The first one was a basic PD controller which was used when there are gates in the drone's field of view. The other controller was used to steer the drone to fly an arc to approach the next gate. Finally, the whole system was tested in a showroom full of aircraft components. The drone could successfully fly through 15 gates with an average speed of $1.5 m/s$ in such a complex environment.

The second research question is proposed as

**RESEARCH QUESTION 2**

How well can the drone estimate its aerodynamics parameters together with its AHRS biases during flight with the help of the vision information?

In Chapter 3, a parameter estimation method was proposed to estimate the aerodynamic coefficients and AHRS biases during the flight. Firstly, the cause of AHRS biases was discussed in detail and then a model of the AHRS biases was established. Next, a gradient descent method was developed to minimize the error between the predicted trajectory and the visual detections. In this way, the AHRS biases and the aerodynamics coefficients can be estimated at the same time. Finally, the proposed method was tested with real flight data. The result showed that with 19 iterations, the proposed method had the final point prediction error of $0.2m$ while the EKF had the final point prediction error of $0.5m$.

The third research question is

**RESEARCH QUESTION 3**

How does a moving horizon based method compare to a Kalman filter in the drone race setting?

In Chapter 4, a 72-gram autonomous racing drone was developed as the flying platform for autonomous drone racing. It is a combination of a cheap smart camera Jevois and a light-weight quadrotor Trashcan. In this chapter, a novel state estimation method VML was proposed. It estimates the error model between the model prediction and visual measurements in a short-time window (moving horizon). The model is then used to compensate for the model prediction in the future to provide accurate state estimation. The simulation result showed that when outliers existed, a commonly used Kalman filter had 15% chance to diverge while the proposed VML did not diverge. And in terms of handling vision delays, the proposed VML needed less than $0.2s$ for one simulation while the Kalman filter needed up to $0.8s$ in the MATLAB implementation. Finally, the proposed method and hardware were tested in the real world. The drone flew through a four-gate racing track by 3 laps with an average speed of $2m/s$ and a peak speed of $2.6m/s$.

Finally, the last research question is raised as

**RESEARCH QUESTION 4**

What are the properties of a neural-network-based imitation of a (close-to) time-optimal control policy?

In Chapter 5, a neural network based optimal controller, G&CNet was developed to control the quadrotor to the target point with minimum time. The G&CNet was trained based on $250,000$ optimal trajectories. It maps the current states of the drone and the corresponding optimal control policy. The simulation result showed that the flying result of the G&CNet is very close to the theoretical optimal solution. Both simulation and real-world flying results showed that it has faster flight than a commonly used polynomial based trajectory generation and tracking method.

## 6.2. DISCUSSION

At the beginning of this work, autonomous drone racing had never been studied by any research group. Thanks to the first autonomous drone racing, the IROS 2016 autonomous drone racing, this topic has attracted many researchers' attention. After several year of development, it gradually has become a new research topic in robotics community and multiple groups have performed research on it. The work presented in this thesis is the four years' research result on this topic since the first autonomous drone race. It covers the most essential parts of flying a drone through racing tracks autonomously including vision, navigation and control. This research aims to provide a solution for autonomous drone racing with robust, accurate and efficient navigation guidance and control algorithms. Compared to other work in the drone race filed, the methods proposed in this thesis are so computationally efficient that can run on a tiny quadrotor without the need of high-performance computers and expensive sensors. Thus, the proposed methods in this thesis can decrease the cost of autonomous racing drones significantly, which is more friendly to users. In addition, as demonstrated in Chapter 4, the proposed methods are able to run on a cheap $72g$ tiny quadrotor, which is much safer than common racing drones and even allows for flying at home. In fact, the techniques presented in this thesis are not only for such a specific target. They can be transferred to other applications easily.

In Chapter 2, we presented a light-weight visual navigation algorithm. One essential part of this navigation algorithm is the detection method called 'Snake gate' which is used to detect the colored squares. This algorithm is so computationally efficient that it can run on a Bebop 1 (made in 2014) without any help of the GPU to provide position measurements. The other essential part of the navigation algorithm is a novel Kalman filter which uses the accelerometers' reading to deduce the quadrotor's velocity. Together with the proposed control methods, the proposed visual navigation method can be used not only for autonomous drone racing but also in other industrial applications like navigation in a warehouse. In this way, the high cost of installing motion capture systems like VICON can be saved.

In Chapter 3, an onboard parameter estimation method was proposed. In fact, AHRS biases exist on each quadrotor while their aerodynamics differs from each other. It is unwise to calibrate the IMUs frequently and estimate the aerodynamics for each quadrotor. Because, on one hand, this process is time-consuming. If more accurate calibration or model is required, some other external equipment like a precise turntable or motion tracking system is needed, which is not always available in the flying area. On the other hand, the biases of IMUs vary due to multiple factors like temperature, humidity and air pressure. They can even change during one flight. Thus, estimating the AHRS biases and aerodynamics online can be a more efficient way to increase the state estimation accuracy and control performance. The proposed method can be used together with other position measurements instead of the detection of the gates in our case. For example, in outdoor flights, GPS measurements can be used to replace the vision detections in the proposed method to estimate the AHRS bias and the quadrotor's dynamics.

A novel state estimation method, VML, is proposed in Chapter 4. The proposed algorithm is more robust to outliers and more efficient in handling vision delays compared to the commonly used Kalman filters, especially when there is a short time when no measurement is available. It is easy to see that the proposed method can be generalized to other navigation scenarios where there are delays, temporary failures and outliers in the measurements. For example, in outdoor flight, the vision measurements can be replaced by GPS measurements directly, which also has delays, outliers and temporary failures. The proposed VML should work as well to provide robust and stable state estimation as in the drone race scenario.

Last but not least, the proposed G&CNet in Chapter 5 moves nonlinear optimal control problem onboard. Obviously, this controller is not only designed for autonomous drone racing. It is a general control method which guides and steers a quadrotor from any point to the target point. This G&CNet is lightweight enough to run on a Bebop 1 quadrotor. Thus, there is no doubt that it can also run on other more powerful drones. Furthermore, it should be able to use in any other quadrotor applications where the guidance and control modules are needed to steer the quadrotor from one point to another point when the accurate state estimation is provided by the navigation module.

## 6.3. FUTURE WORK

Although the work presented in this thesis provides a systematic solution for autonomous drone racing as one of the most computationally efficient and fastest autonomous drone racing solutions, it is still not as fast as human pilots' racing drones. There is still much space for improvement.

Regarding visual detection, although the proposed Snake gate detection is successfully used for four years' autonomous drone racing, it still has its shortcomings. For example, when the light conditions change, the color threshold has to be retuned manually, which is not very straight-forward and time-consuming. Also, when the light is uneven which leads to incorrect exposure, the pixel searching in the Snake gate detection can stop at the middle of the bar, which leads to false negative detections. At last, the snake gate detection does not work when part of the gate is obscured. Thus, a more robust gate detection technique should be developed. One research direction can be deep learning based object recognition techniques. With enough training data (labeled images), the neural network should be able to detect the racing gate onboard regardless of what the light condition is. It should also be able to detect the gates when parts of them are obscured.

In terms of guidance and control, the proposed G&CNet only works in the vertical plane and the final target is to hover at one point. In order to use the G&CNet in real applications, a 3D version has to be developed. Thus, the optimal policy dataset has to be regenerated with a full state model. Furthermore, the final target point should not be velocity zero. There should be more final states so that the quadrotor can change its target smoothly during the flight. What follows is how to increase the optimization speed and avoid local minima. Moreover, when the new dataset is generated, the structure of the neural network may need to be adjusted to learn the dataset accurately.

Finally, the proposed VML and G&CNet are verified separately. It will be very interesting to combine them together in a real drone race scenario once the 3D version of the G&CNet is established. At the same time, robust, close-to-time-optimal flight would then get in reach even of small, cheap quadrotors.

# ACKNOWLEDGEMENTS

This thesis is the result of my four years' work at the MAVLab of Control and Simulation department at TU Delft. It was a great honor for me to be a part of this fantastic team. At the MAVLab, I grew up from a student who lived in the simulation world to a researcher who could make his ideas fly in the air. It is obvious that this thesis could never have come out without the help from my colleagues including my supervisors, my PhD fellows, the engineers at the lab and the master students who worked with me. I would like to thank you all for the help, the strict requirements and the encouragement you gave me.

First I would like to thank my promotor Guido de Croon. I can still remember the sunny afternoon you asked me if I wanted to join your drone race team. From that moment, we worked together to win the IROS autonomous drone races although I haven't brought you a gold medal. But you taught me a lot on how to do research, how to read papers and how to solve problems. The countless scenes you worked with me late at the Cyberzoo until being kicked out, debugged the code in hotel rooms, fixed the code until the last minute of the competitions always inspired me to bravely face the challenges both in my academic life and daily life. Thank you for teaching me how to be an independent scientist, inspiring me with new ideas and encouraging me when I got stuck in my research.

My thanks also go to my co-promotor Coen de Visser. Although my research didn't progress in the direction you originally designed, it was you who introduced me to the area of drones, which opened a door for me to a new fantasy world. Without your patient guidance, I could not have entered the robotics world so smoothly. I still remember that you told me 'We are colleagues.' which helped me discuss or argue on equal terms with my supervisors or other researchers including you. Thank you for your kindness which made my first far-from-home year easy and smooth.

To Christophe De Wagter, you are such a magic Doraemon for me! You always had ideas from your magic pocket. It seems that whenever we have questions, we only need to shout out 'We have our Christophe!' and you will jump out to save us. I remember I once asked you how you knew so much. Your answer was 'I keep learning!' which inspires me until now. Thank you for your help during my PhD. I hope one day I can be others' Doraemon like you.

I sincerely thank Max Mulder and Qiping Chu. Thank you for interviewing me and giving me such a valuable chance to have research here. To Max, thank you for monitoring my progress which ensured I was always on the correct track and thank you for teaching me sailing although I still cannot do it well. To Ping, thank you for your advice on my research and thank you for the random talk at the coffee corner which made me feel like talking to elders at home.

To Erik van der Horst, I sincerely express my thanks to you. Your restless working guaranteed my tiny drone to fly in the air. We spent endless evenings at the lab soldering chips, testing drones and eating delivery food while discussing the true meaning of life. I am so honoured to have you be the witness of the tiny drone flying through the gate for the first time. I can not imagine how my research life would have been without such an experienced engineer.

I want to thank Dario Izzo, Ekin Öztürk and Dharmesh Tailor at the European Space Agency (ESA). Thank you for collaborating with me on our project. I value the experience working with the ESA and I appreciate your receptions at the canteen and bar.

To Michaël Ozo, you joined our team at a hard time. Thank you for your patience in making the optical flow code back to life, which helped us continue on drone racing. I felt so lucky to work with you. Your kindness and patience always inspire me to keep going forward. I still remember our adventures in Daejeon, South Korea and Vancouver where we debugged the code in a luxury hotel in the city center of Vancouver. I will not forget the fantasy self-driving tour in Vancouver just two days after we lost the competition.

I also want to thank the people at the MAVLab who are a crowd of such amazing people. We, PhD students at the lab, have seen too many failures and frustrated faces. But that is one important part of our lives, isn't it? I have also witnessed countless success moments like HannWoei's autonomous landing, Sjoerd's autonomous Delfly, Ewoud's INDI controller, Kirk's event-based landing, Kimberly's swarm pocket drones, Mario's leader and follower, Diana's flapping wing, Tom's obstacle avoidance, Shushuai's onboard localization, Federico's drone racing and Yingfu's deep learning based visual navigation, etc. To Sven, I left the lab shortly after you joined, I didn't witness your flying by myself but I believe you will have your shining time shortly. I am so proud of us that we, as scientists, push the boundary of human knowledge a little forward and I believe contributions like ours will finally improve people's quality of life. My thanks also go to the researchers and engineers at the lab who are Matěj Karásek, Julien Dupeyroux, Bart Remes, Nilay Sheth, Freek van Tienen, Roland Meertens, Kevin van Hecke, Bart Slinger, etc. Thank you for the help and the fun you gave me.

I enjoyed my four years time in the Netherlands where I had a lot of friends around me. Feijia Yin and Xiaodong Guo, I don't know how to express my thanks to you. But thank you for your advice, help and the happy time we spent together. Linfeng Gou and Chuan Lin, we became good friends since we took the same flight to the Netherlands. It was our first time going abroad and throwing ourselves into such a strange place totally different from home. We helped each other in the Netherlands and I believe we will never forget this one year. Hongxiao Guo, my flatmate, I still remember two frustrated PhDs cheered each other up in the living room after the failures in their respective labs. But fortunately, we have gone through it! Ye Zhang and Yingzhi Huang, I cannot forget you served me my first dinner when I was far from home for the first time, which made me relax and feel at home. Xuerui Wang and Sihao Sun, thank you for inviting me to join your hotpot many times which I appreciate. My thanks also go to other friends who are Lei Yang, Ye Zhou, Zhou Nie, Bo Sun, Ying Yu, Cheng Liu, Peng Lu, Junzi Sun, Wei Fu, etc.

Please let me thank my PhD fellows at Control  Simulation group who are Annemarie Landman, Dyah Jatiningrum, Tommaso Mannucci, Sophie Armanini, Daniel Friesen, Jaime Junell, Sarah Barendswaard, Dirk van Baelen, Kasper van der El, Emmanuel Sunil, Paolo Scaramuzzino, etc. Thank you for the interesting talk at the coffee corner!

Last but not least, I sincerely express my thanks to my parents, Yuanye Li and Yaping

Liu. Thank you for your selfless love and support. Without you, I would never have thought about doing a PhD abroad. Thank you for the long phone calls when I was stuck in my research for a long time. And thank you for your support which gave me the courage to move forward. You are always my powerful backing!

# Curriculum Vitæ

## Shuo Li

| | |
|---|---|
| 11-09-1990 | Born in Shaanxi, China. |

## Education

| | |
|---|---|
| 2009–2013 | BSc in College of Astronautics<br>Northwestern Polytechnical University |
| 2013–2016 | MSc in College of Astronautics<br>Northwestern Polytechnical University |
| 2015–2020 | Ph.D. in Aerospace Engineering<br>Delft University of Technology<br>*Thesis:* Visual Navigation and Optimal Control for Autonomous Drone Racing<br>*Promotor:* Prof. dr. G. C. H. E. de Croon |

## Awards

| | |
|---|---|
| 2016 | Second prize in IROS autonomous drone race |

## ACADEMIC ACTIVITIES

Reviewer for Journal of Field Robotics

Reviewer for International Conference on Robotics and Automation (ICRA)

Reviewer for IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

# LIST OF PUBLICATIONS

5. **Li S.**, van der Horst, E., Duernay, P., De Wagter, C., de Croon, G. C, *Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone*, Journal of Field Robotics (2020).

4. **Li, S.**, Ozo, M. M., De Wagter, C., de Croon, G. C., *Autonomous drone race: A computationally efficient vision-based navigation and control strategy*, Robotics and Autonomous Systems (2020).

3. **Li, S.**, Öztürk, E., De Wagter, C., de Croon, G. C., Izzo, D., *Aggressive online control of a quadrotor via deep network representations of optimality principles*, 2020 IEEE International Conference on Robotics and Automation (ICRA) 6282-6287 (2020).

2. **Li, S.**, De Wagter, C., de Visser, C. C., Chu, Q. P., de Croon, G. C. H. E. *In-flight model parameter and state estimation using gradient descent for high-speed flight*, International Journal of Micro Air Vehicles (2019).

1. Moon, H., Martinez-Carranza, J., Cieslewski, T., Faessler, M., Falanga, D., Simovic, A., Scaramuzza, D., **Li, S.**, Ozo, M., De Wagter, C. and de Croon, G., *Challenges and implemented technologies used in autonomous drone racing*, Intelligent Service Robotics (2019).