# Visual Navigation for Tiny Drones

Tom van Dijk

# Visual Navigation for Tiny Drones

# Visual Navigation for Tiny Drones

**Proefschrift**

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof.dr.ir. T.H.J.J. van der Hagen;
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op vrijdag 15 november 2024 om 12:30 uur

door

**Josephus Cornelis van Dijk**

Master of Science in Systems & Control,
Technische Universiteit Delft,

Master of Science in Mechanical Engineering,
Technische Universiteit Delft,

geboren te Tilburg, Nederland.

Dit proefschrift is goedgekeurd door de promotoren.

| | |
|---|---|
| *Trefwoorden:* | Micro Aerial Vehicles, visual navigation, obstacle avoidance, route following, depth perception |
| *Gedrukt door:* | Ipskamp Printing |
| *Omslag:* | T. van Dijk |

ISBN 978-94-6384-675-2

Een elektronische versie van dit proefschrift is beschikbaar op
`http://repository.tudelft.nl/`.

“The enemy of art
is the absence of limitations”

*– Orson Welles*

# Contents

**Summary** **ix**

**1 Introduction** **1**
1.1 Tiny drones . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2 Visual navigation . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Research questions . . . . . . . . . . . . . . . . . . . . . . . . 4
1.4 Dissertation outline . . . . . . . . . . . . . . . . . . . . . . . 5
References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**2 A (p)review of visual collision avoidance** **9**
2.1 Problem statement . . . . . . . . . . . . . . . . . . . . . . . . 10
2.2 Literature review . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.2.1 Obstacle Avoidance . . . . . . . . . . . . . . . . . . . . . 11
2.2.2 Deep Learning for depth perception . . . . . . . . . . . . 31
2.3 Preliminary results . . . . . . . . . . . . . . . . . . . . . . . . 34
2.3.1 Monocular depth perception. . . . . . . . . . . . . . . . . 34
2.3.2 Flight tests . . . . . . . . . . . . . . . . . . . . . . . . . 42
References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

**3 Monocular depth perception** **53**
3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
3.2 Related work. . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
3.3 Position vs. apparent size . . . . . . . . . . . . . . . . . . . . 56
3.3.1 Evaluation method . . . . . . . . . . . . . . . . . . . . . 59
3.3.2 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
3.4 Camera pose: constant or estimated?. . . . . . . . . . . . . . . 60
3.4.1 Camera roll. . . . . . . . . . . . . . . . . . . . . . . . . . 62
3.5 Obstacle recognition . . . . . . . . . . . . . . . . . . . . . . . 65
3.5.1 Color and Texture . . . . . . . . . . . . . . . . . . . . . . 67
3.5.2 Shape and contrast . . . . . . . . . . . . . . . . . . . . . 67
3.6 Conclusions and future work . . . . . . . . . . . . . . . . . . . 71
References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

**4 Visual route following** **75**
4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
4.1.1 Efficient visual route following . . . . . . . . . . . . . . . 79

4.2 Results . . . 79
4.2.1 Selected homing algorithms for comparison . . . 79
4.2.2 Comparison of holistic visual homing algorithms . . . 82
4.2.3 Visual homing and odometry . . . 84
4.2.4 Route following with minimal memory . . . 86
4.3 Discussion . . . 88
4.4 Materials and methods . . . 91
4.4.1 Hardware . . . 91
4.4.2 Image processing . . . 91
4.4.3 Homing implementation . . . 91
4.4.4 Route memory . . . 92
4.4.5 Experimental setup . . . 93
4.4.6 Simulation setup . . . 94
References . . . 94

**5 Conclusion** **99**
5.1 Perception . . . 100
5.2 Avoidance . . . 102
5.3 Route following . . . 104
5.4 Navigation . . . 104
5.5 Outlook . . . 105
References . . . 105

**A Supplementary materials for Chapter 4: Visual route following** **109**
A.1 Presence of texture in different environments . . . 109
A.2 Route-following experiments in simulation . . . 111
A.3 Theoretical model for spacing snapshots . . . 117
A.3.1 Impact of model refinements . . . 118
A.4 Position plots over time for flight experiments . . . 120

**Acknowledgements** **123**

**Curriculum Vitæ** **125**

**List of Publications** **127**

# Summary

In recent years, the use of drones in practical applications has seen a rapid increase, for instance in inspection, agriculture or environmental research. Most of these drones have a span in the order of tens of centimeters and a weight of half a kilogram or more. Smaller drones offer advantages in terms of safety and cost. However, their reduced payload capacity makes it difficult to carry the sensors and computers required for autonomous operation.

One of the most essential tasks an autonomous drone needs to perform is navigation. Here, navigation is defined as the ability to move towards a specified location while avoiding obstacles along the way. Ideally, the drone should also remember traveled routes, to make the return journey more efficient. However, on tiny drones (palm-size or smaller) the on-board processing power is often limited to a single microcontroller and the choice of sensors is limited. Cameras are popular sensors for tiny drones, because they're small, lightweight and passive, although they do require some processing power to produce useful results. The goal of this dissertation is to find a new, visual navigation strategy that fits within the constraints of these tiny drones.

First, existing work in terms of visual perception and avoidance is reviewed. Multiple options exist for visual perception: stereo vision, optical flow and monocular vision. All of these options are discussed and compared, leading to the conclusion that stereo vision performs best at shorter distances albeit at the cost of an additional camera, while monocular vision performs better at longer distances. Optical flow is ruled out for avoidance, as it has excessively large errors precisely in the direction of movement. For avoidance, the options in terms of motion planning, map types and odometry are discussed. Perhaps unsurprisingly, the optimal choice is found to be dependent on the application. For computational efficiency on tiny drones, the most important choice is whether multiple measurements should be fused into a single map, or if individual percepts are good enough for avoidance. The latter is significantly less computationally demanding. For visual odometry, the depth information should be used if available, and the IMU can provide efficiency benefits in feature tracking. At Preliminary results are shown for monocular vision, visual odometry and obstacle avoidance.

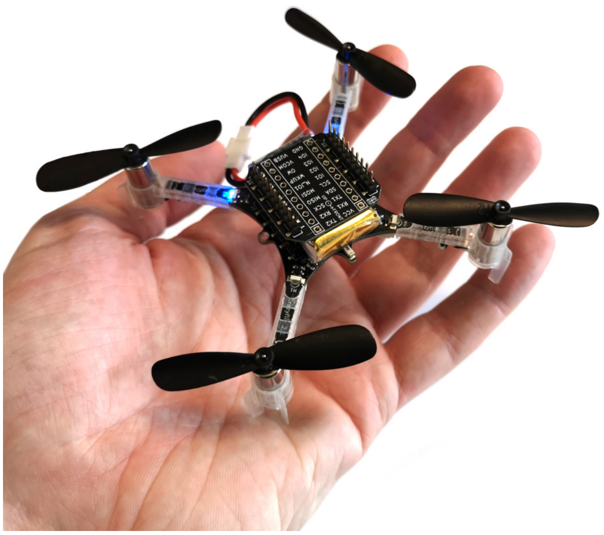
Secondly, the dissertation takes a deeper dive into monocular depth estimation. Monocular depth estimation has the advantage that it only needs a single camera – which saves valuable weight on tiny drones – but its processing is more complex. The goal of this chapter is to analyze the learned behavior of neural networks for monocular depth perception, to see if this can be distilled into simple, lightweight algorithms. Using experiments based on data augmentation, it is shown that all four of the analyzed networks rely on the vertical position of objects in the image to estimate their depth. While this cue would be simple to replicate, it does depend on a known pose of the camera. Further investigation shows that the networks have a strong prior 'assumption' about this pose, which may make transfer to drones more difficult. Finally, the networks need to have some sense of an 'object'. In this case, it is shown that various shapes are recognized as an object provided that they have contrasting outlines and a dark shadow at the bottom. While this last feature is clearly present in the car-based KITTI dataset, it may not transfer directly to other environments. However, the vertical position cue can likely be used to provide monocular depth estimates to resource-limited systems such as tiny drones.

Thirdly, the remembering of traveled routes is investigated. Traditional mapping strategies from robotics would quickly run out of memory on microcontrollers, especially over longer trajectories. Instead, inspiration for a memory-efficient route-following strategy is found in nature. Here, insects are able to remember and follow remarkably long routes despite their tiny brains. Their strategy is often broken up into a few components, most notably path integration (odometry in robotics) and visual homing. We implement a novel strategy based on these components on a 56-gram drone. Here, the focus lies on traveling long distances using odometry, while periodically using visual homing to return to known locations to counteract odometric drift. The proposed strategy is demonstrated over multiple experiments, where the most efficient run required only 0.65 kilobytes to remember a route of 56 meters. This shows that tiny drones can retrace known paths by combining odometry with periodic homing maneuvers to counteract drift.

Finally, the avoidance of obstacles is discussed in the conclusion of this dissertation. This research has been performed by MSc students under my supervision, who have found and demonstrated that bug algorithms are an effective navigation strategy in three-dimensional, limited-field-of-view applications and provide a lightweight goal-oriented avoidance strategy that is suitable for tiny drones.

By combining all of the above results, a full navigation strategy for tiny drones can be proposed: tiny drones can visually navigate by using lightweight monocular vision algorithms to perceive obstacles, three-dimensional bug algorithms to avoid them while moving to new locations, and odometry and visual homing to retrace known paths.

# 1 Introduction



Bitcraze Crazyflie 2.1

92 x 92 x 29 mm
27 grams

On-board processor:
STM32F405
(ARM Cortex-M4, 168 MHz,
192 kB RAM, 128 kB Flash)

Max payload weight:
15 grams

Approximate cost: $225

Figure 1.1: An example of a ‘tiny drone’: the palm-sized Bitcraze Crazyflie 2.1.

## 1.1. Tiny drones

Drones are an upcoming technology that find more and more applications nowadays. For instance, drones can be used to inspect powerlines [1], industrial sites [2] or railways [3]. Precision agriculture is another widely-discussed application, where drones can capture high-quality data and measurements in a relatively short time [4]. Drones can also interact with their environment, for instance by collecting water- or atmospheric samples [5, 6], environmental DNA [7] or acoustic measurements[1]. In indoor environments such as greenhouses, drones may have an advantage over regular robots since they require less infrastructure.

Current-day drones are typically quadrotors or multirotors with a span in the order of tens of centimeters and a weight of half a kilogram or more. Tiny drones such as the Crazyflie shown in Fig. 1.1 do not see much practical use yet. This is unfortunate, since such drones offer a lot of advantages compared to their larger cousins. Firstly, tiny drones are very safe and unlikely to cause serious harm [8]. Their mass is significantly smaller, reducing the impulse and energy on collision. Additionally, their rotors have less mass and inertia, and are therefore less likely to seriously harm any person they come in contact with. As a result, tiny drones could be used near humans or in otherwise sensitive environments, where regular drones would be too dangerous. A second advantage of smaller drones is their lower cost. Even if the drone is completely destroyed, their replacement cost will likely be less than that of a regular-sized quadrotor. As a direct consequence, tiny drones could take a lot more risk during their mission. Alternatively, tiny drones could be used in larger numbers, in which they could speed up tasks by doing them in parallel or add redundancy in numbers [9–11].

With all that said, there are of course also disadvantages to tiny drones compared to

[1]https://www.tudelft.nl/en/2023/lr/flying-robots-survey-biodiversity-and-climate-inside-tropical-rainforests

larger ones. The most obvious and important one is their small payload capacity. While larger drones can carry a variety of sensors – either for the mission, navigation, or both – tiny drones need to make a selection in what they carry. Some sensors like laser scanners (such as the 830-gram Velodyne Puck[2] or the 190-gram Slamtec RPlidar A1[3]) cannot even be carried at all because of their weight. Besides sensors, the onboard computer and battery capacity are also limited by the payload capability. Unfortunately, this makes it really difficult to make tiny drones fully autonomous.

## 1.2. Visual navigation

To make tiny drones more useful, it is important that they can find their way between different places. They should be able to find their way to new locations, while avoiding obstacles along the way. To achieve this, the drones will first need to detect obstacles, and then to maneuver such that a collision is avoided while still moving towards the target position. When routes are traversed more often, the efficiency of navigation can be improved by following routes that are already known to be safe, rather than re-discovering a new route each time.

This behavior is already possible on larger drones. On tiny drones, however, it is still an open problem. Two main factors come into play here: firstly, tiny drones have significantly less payload capacity (Fig. 1.1). As a result, it is not possible to carry most of the sensors that are available on larger drones. Either the sensor itself is too heavy, or the power they require causes an excessive increase in battery weight. However, one sensor that seems especially suitable for tiny systems, is the camera. The camera is a passive sensor and therefore requires relatively little power. At the same time, it is highly versatile and provides rich information about the drone's immediate environment. This is also the major downside of cameras and vision: significant amounts of processing are often required to extract useful information.

The processing of sensor data is the second hurdle for navigation on tiny drones. On larger drones and robots, Simultaneous Localization and Mapping (SLAM) is the go-to approach for navigation from local sensor data. Using SLAM, the robot can construct a map of its environment that maintains consistency over multiple measurements, despite errors in e.g. the measured displacement between positions [12–14]. However, SLAM tends to be a computationally intensive process which requires a powerful processor and high memory capacity. And while steps are taken to reduce these requirements, state-of-the-art methods in this direction still require a mobile GPU or FPGA [15] rather than a simple microcontroller. One of the smallest examples of SLAM, 'tinySLAM' [16], is able to run on a microcontroller, but can only map relatively small areas and is not shown to maintain consistency at greater distances. Even without the processing requirements of SLAM, just keeping a map in memory can already be challenging for tiny drones, which often only have a few kilobytes of microcontroller memory to work with. Ideally, navigation should be performed without a map altogether. But is this actually possible?

Luckily, insect navigation results from biology show us that such a feat is indeed possible. Insects such as ants and bees are able to traverse remarkable distances despite their

[2] https://velodynelidar.com
[3] https://www.slamtec.ai/home/rplidar_a1/

tiny brains. Rather than building a detailed, geometrical map, navigation strategies in nature seem far more ‘behavior-based’, leading to robust results with remarkable little data. The navigation strategies of insects have been broken down by biologists into three major components: Path Integration, Visual Guidance and Route Following [17]. These will provide a great source of inspiration when implementing navigation for tiny drones.

The observations of biology have also spawned new research topics in robotics. Of note here is the development of Bug Algorithms [18, 19]. Bug algorithms use very simple behavior rules to let a robot traverse a complex, obstacle-ridden environment towards a target position. The main challenge of these algorithms is to prevent the robot from getting stuck in loops. Not only do these algorithms manage to do this, they can even do so without a map! Therefore, bug algorithms form a great complement to the techniques mentioned above. If bug algorithms, path integration and visual guidance could be combined with a lightweight visual perception algorithm, this would pave the way towards visual navigation for tiny drones.

## 1.3. Research questions

As has become apparent from the above sections, the goal of this thesis is to bring visual navigation to tiny drones. How to achieve this, is the main research question:

**Main Research Question**

How do we bring visual navigation to tiny drones?

To make this question as specific as possible, I define navigation as *the ability to purposefully move through an environment without collision to reach a given point of interest*. The drone must move purposefully, rather than moving randomly until it eventually encounters the point of interest, because battery capacity is limited. Collisions with the environment should be avoided, because for most drones this leads to failure or damage to the environment. (Collision-proof concepts such as caged drones are not considered in this thesis.) Exploration is not considered to be a part of navigation, but as a higher-level task that provides target positions to the navigation system. On the other hand, moving towards such a position that has not been visited before, but whose position is precisely specified, *is* considered part of navigation.

To safely move towards new locations, the drone should be able to perceive and avoid obstacles along the way. As discussed above, the hardware that the drone can bring along for this task is severely restricted. Therefore, the first research question is how tiny drones can perceive obstacles, taking the payload and processing limitations into account:

**RQ 1**

How can tiny drones perceive obstacles?

After perceiving an obstacle, the drone must find a way around it. For random exploration, any action that avoids a collision would be sufficient, but when moving towards a goal (as defined above), the drone should also take the position of its intended target into account. The question of how to weigh the different movement options, how to take the target location into account, and how to do all of this as efficiently as possible, forms the basis of research question 2:

**RQ 2**

How can tiny drones avoid obstacles while moving towards a specific point?

With the answers to these two questions, it is possible to implement a navigation systems that allows the drone to move between points. However, it is still somewhat lacking, as for each movement the drone has to re-discover the route. While this is fine in unexplored environments, there are better ways to do this in environments that are traversed multiple times. By remembering the routes between important points, the drone can skip the search for a new route, and instead follow the same route as it did before. This might save valuable time and reduce the chance of a collision or getting stuck. Returning to the start location is also part of nearly all missions. Therefore, the following research question serves to fully 'complete' the navigation task:

**RQ 3**

How can tiny drones retrace known paths?

While solutions are already available for larger drones, the main challenge here will be to find suitable algorithms and strategies that can be used on microcontroller-equipped tiny drones.

## 1.4. Dissertation outline

This dissertation is structured as follows. Chapter 2 will take a significantly deeper dive into the existing literature than was possible in this introduction. It is primarily focused on the two first research questions: how can obstacles be perceived and how can they be avoided? Besides the literature overview, it presents some very early results of a practical collision avoidance system. Additionally, it presents the initial experiments on monocular depth estimation that led to the research of Chapter 3.

Chapter 3 takes a closer look at perception (RQ 1). Based on Chapter 2's findings, it assumes the use of a camera for obstacle avoidance. Furthermore, it assumes monocular vision, as the stereo baseline on tiny drones will be a strong limitation for stereo vision on these platforms. At the time of writing, monocular depth perception is primarily done through deep learning. However, there was no research into how these networks actually achieved this task. Chapter 3 aims to experimentally find the underlying mechanisms for monocular depth perception. While it does not directly lead to a lightweight implementation, it gives useful insights that might inspire such an algorithm in the future. At the same time, it provides a critical view of the safety of current depth estimation applications and strongly advocates the use of more varied training and testing datasets in the process.

Research question 2 is discussed in Chapter 2, but will not be treated in further detail in this dissertation. The research into this question has been performed by MSc students that I have supervised during my time at the MAVLab. I am happy to report that their work shows that mapless avoidance of obstacles while moving towards a goal is not only possible, but can even be performed with a high success rate. The results will be discussed in the conclusion chapter to answer this question.

In Chapter 4, the following of known routes will be investigated (RQ 3). Since nearly all map structures quickly become too large for the kilobyte-size memory of microcontrollers, a novel approach inspired by insect navigation is developed and evaluated on a 56-gram microdrone. The results show that routes can be followed using less than 20 bytes of memory per meter, and that the strategy is therefore highly suitable for tiny systems.

Finally, in Chapter 5 I will collect the results from the main chapters to answer the research questions posed above. Using the conclusions of Chapter 5, it will be possible to bring visual navigation to tiny drones, thereby achieving the main goal of this thesis.

## References

[1] S. Jordan, J. Moore, S. Hovet, J. Box, J. Perry, K. Kirsche, D. Lewis, and Z. T. H. Tse, *State-of-the-art technologies for uav inspections,* IET Radar, Sonar & Navigation **12**, 151 (2018).

[2] P. Nooralishahi, C. Ibarra-Castanedo, S. Deane, F. López, S. Pant, M. Genest, N. P. Avdelidis, and X. P. Maldague, *Drone-based non-destructive inspection of industrial sites: A review and case studies,* Drones **5**, 106 (2021).

[3] T. Askarzadeh, R. Bridgelall, and D. D. Tolliver, *Systematic literature review of drone utility in railway condition monitoring,* Journal of Transportation Engineering, Part A: Systems **149**, 04023041 (2023).

[4] A. Rejeb, A. Abdollahi, K. Rejeb, and H. Treiblmaier, *Drones in agriculture: A review and bibliometric analysis,* Computers and electronics in agriculture **198**, 107017 (2022).

[5] H. Lally, I. O'Connor, O. Jensen, and C. Graham, *Can drones be used to conduct water sampling in aquatic environments? a review,* Science of the total environment **670**, 569 (2019).

[6] J. Jońca, M. Pawnuk, Y. Bezyk, A. Arsen, and I. Sówka, *Drone-assisted monitoring of atmospheric pollution—a comprehensive review,* Sustainability **14**, 11516 (2022).

[7] E. Aucone, S. Kirchgeorg, A. Valentini, L. Pellissier, K. Deiner, and S. Mintchev, *Drone-assisted collection of environmental dna from tree branches for biodiversity monitoring,* Science Robotics **8**, eadd5762 (2023).

[8] Z. Svatỳ, L. Nouzovskỳ, T. Mičunek, and M. Frydrỳn, *Evaluation of the drone-human collision consequences,* Heliyon **8** (2022).

[9] K. N. McGuire, *Indoor swarm exploration with Pocket Drones*, Ph.D. thesis (2019).

[10] M. Coppola, *Automatic Design of Verifiable Robot Swarms*, Ph.D. thesis (2021).

[11] S. Li, *Autonomous Swarms of Tiny Flying Robots*, Ph.D. thesis (2021).

[12] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics* (MIT Press, Cambridge, Mass., 2005).

[13] T. Taketomi, H. Uchiyama, and S. Ikeda, *Visual slam algorithms: A survey from 2010 to 2016,* IPSJ Transactions on Computer Vision and Applications **9**, 1 (2017).

[14] A. Macario Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, *A comprehensive survey of visual slam algorithms,* Robotics **11**, 24 (2022).

[15] M. Abouzahir, A. Elouardi, R. Latif, S. Bouaziz, and A. Tajer, *Embedding slam algorithms: Has it come of age?* Robotics and Autonomous Systems **100**, 14 (2018).

[16] B. Steux and O. El Hamzaoui, *tinyslam: A slam algorithm in less than 200 lines c-language program,* in *2010 11th International Conference on Control Automation Robotics & Vision* (IEEE, 2010) pp. 1975–1979.

[17] X. Sun, S. Yue, and M. Mangan, *A decentralised neural model explaining optimal integration of navigational strategies in insects,* Elife **9**, e54026 (2020).

[18] V. Lumelsky and A. Stepanov, *Dynamic path planning for a mobile automaton with limited information on the environment,* IEEE transactions on Automatic control **31**, 1058 (1986).

[19] K. N. McGuire, G. C. de Croon, and K. Tuyls, *A comparative study of bug algorithms for robot navigation,* Robotics and Autonomous Systems **121**, 103261 (2019).

# 2
# A (p)review of visual collision avoidance

With a growing number of Unmanned Aerial Vehicles (UAVs), the risk of collisions increases. A Collision Avoidance System (CAS) is required to keep UAV operation safe. A major challenge for CASs on smaller UAVs is the limited payload capacity, which drastically limits the sensors that can be carried. Here, cameras can provide rich information despite their low weight.

Visual collision avoidance starts with seeing potential obstacles. Stereo vision works well to estimate depth at short range, but requires an additional camera. Optical flow is less useful for avoidance, since the optical flow converges to zero near the focus-of-expansion, i.e. precisely in the direction of movement. Finally, appearance cues – such as the height of objects in the field-of-view – provide an extra sense of depth while requiring only one camera. These cues are difficult to implement by hand. However, rapid developments in deep learning have created new opportunities to learn these cues instead. Once an obstacle is sensed, it must be avoided. An overview of maps and strategies is presented, but the optimal choice depends strongly on the application.

This chapter ends with preliminary results on the above-mentioned topics. It turns out that appearance-based depth prediction using neural networks might not transfer well to UAVs. However, since these networks are black boxes, this is difficult to predict or debug. An experimental approach is used to gain an insight into the inner workings of the neural network under evaluation. Other parts of a potential collision avoidance system are implemented on a real-world UAV. Using stereo vision in combination with visual odometry, the UAV can control its trajectory in indoor and outdoor environments and is able to stop in front of obstacles.

The contents of this chapter have been published as:

**van Dijk, T.** (2020). *Self-Supervised Learning for Visual Obstacle Avoidance*. TU Delft. (Conceptualization: Tom van Dijk, Guido de Croon)

## 2.1. Problem statement

With a growing number of drones, the risk of collision with other air traffic or fixed obstacles increases. New safety measures are required to keep the operation of Unmanned Aerial Vehicles (UAVs) safe. One of these measures is the use of a *Collision Avoidance System* (CAS), a system that helps the drone autonomously detect and avoid obstacles.

The design of a Collision Avoidance System is a complex task with many smaller subproblems, as illustrated by Albaker and Rahim [1]. How should the drone sense nearby obstacles? When is there a risk of collision? What should the drone do when a conflict is detected? All of these questions need to be answered to develop a functional Collision Avoidance System. However, all of these subproblems – except the sensing of obstacles – only concern the *behavior* of the vehicle. They can be solved independently of the target platform as long as it can perform the required maneuvers; it does not matter whether it is a UAV or a larger vehicle.

The *sensing of the environment*, on the other hand, is the only subproblem that places requirements on the hardware, specifically the sensors that should be carried by the UAV. It is the hardware that sets UAVs apart from other vehicles. Unlike autonomous cars, other ground-based vehicles or larger aircraft, UAVs have only a small payload capacity. It is therefore not practical to carry large or heavy sensors such as LIDAR or radar for obstacle avoidance. Instead, obstacle avoidance on UAVs requires clever use of lightweight sensors: cameras, microphones or antennae. This literature survey will therefore focus on the *sensing* of the environment.

Out of the sensors mentioned above – cameras, microphones and antennae – cameras are the only ones that can detect nearly all ground-based obstacles and other air traffic; microphones and antennae are limited to detection of sources of noise or radio signals[1]. Therefore, this review will focus on the *visual detection of obstacles*.

The field of computer vision is well-developed; it may already be possible to find an adequate solution for visual obstacle detection using existing stereo vision methods like Semiglobal Matching (SGM) [2]. These methods, however, only use a fraction of the information present in the images to estimate depth – the *disparity*. Other cues such as the apparent size of known objects are completely ignored. The use of *appearance cues* for depth estimation is a relatively new development driven largely by the advent of Deep Learning, which allows these cues to be learned from large, labeled datasets. As long as the UAV's operational environment is similar to this training dataset it should be possible to use appearance cues in a CAS. However, this is difficult to guarantee and may require a prohibitively large training set.

*Self-Supervised Learning* may provide a solution to this problem. After training on an initial dataset, the UAV will continue to collect new training samples during operation. This allows it to 'adapt' to its operational environment and to learn new depth cues that are relevant in that environment. Self-Supervised Learning for depth map estimation is a young field, the first practical examples started to appear around 2016 (e.g. [3]). Most of the current literature is focused on automotive applications or on datasets captured at

[1]They could be used to detect *reflections* of sound or radio waves – this is the working principle behind ultrasonic ranging and radar – but since these are active measurements the power consumption is assumed to be too large for use on UAVs. Additionally, in the case of ultrasonic measurements the range might be too short.

eye-level. It is still an open question whether Self-Supervised Learning techniques can be used for visual obstacle avoidance on UAVs.

## 2.2. Literature review

This section presents an overview of relevant literature for visual obstacle avoidance. The review consists of two parts: subsection 2.2.1 presents an overview of obstacle avoidance systems and their components, paying special attention to the visual detection of obstacles. Then, subsection 2.2.2 takes a closer look at the use of neural networks for depth estimation.

### 2.2.1. Obstacle Avoidance

Practical Collision Avoidance Systems (CAS) need to solve a number of subproblems in order to detect and avoid obstacles. An overview of the tasks involved and possible solutions is given in [1, 4]. In general, a CAS contains the following elements:

- Sensing of the environment
- Conflict detection
- Avoidance maneuver: planning and execution

The system should have some way to *sense* potential obstacles in its environment. In this review, sensing will be primarily performed through vision, but other sensors could also be used to detect obstacles. Communication with other aircraft also falls under this element. *Conflict detection* is used to decide whether an evasive maneuver should be performed. It usually requires a method to predict future states of the UAV and of detected obstacles or aircraft and a threshold or minimum safe region that should stay free of obstacles. When the conflict detection indicates that a collision is imminent, an *escape maneuver* has to be performed to avoid this potential collision. Depending on the method, this maneuver can be performed using simple rules, planned by optimizing some cost function or even performed in collaboration with other aircraft (e.g. TCAS).

The elements listed above are typical for the avoidance of other vehicles but can also be used for the avoidance of static obstacles. In this case, the conflict detection is often skipped or simplified since only the UAV itself is moving; instead it is often performed implicitly during the planning of the escape maneuver around the obstacle.

Subsections 2.2.1 and 2.2.1 take a closer look at the sensing of the environment and planning of escape maneuvers. Conflict detection is not considered for now as this review is primarily aimed at the avoidance of static obstacles. Subsection 2.2.1 will briefly highlight the literature (or lack thereof) on the performance evaluation of Collision Avoidance Systems.

#### Sensing

The goal of sensing is to detect and locate nearby obstacles. The range of sensors that could be used for obstacle detection is large, but a number of these can be ruled out for usage on UAVs because of their weight or power requirements. This subsection will focus on the visual detection of obstacles.

The localization of obstacles through vision can be split into two parts: estimation of the bearing towards the obstacle and estimation of the distance. As long as the obstacle

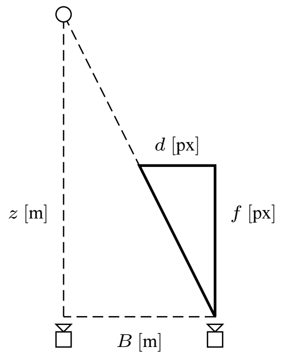


Figure 2.1: Stereo vision uses the disparity $d$ to estimate the depth $z$ towards obstacles. The camera baseline $B$ and focal length $f$ are constant and obtained through calibration.

can be reliably found in the image, estimation of the *bearing* is fairly straightforward. The position of the obstacle in the image is a direct result of its bearing relative to the camera and this relation can be inverted.

Estimation of the *distance* towards the obstacle is more complicated. Since the obstacle is projected onto the image plane, the depth information is lost. Other cues need to be used to estimate the distance towards the obstacle. These cues can be broadly split into three categories:

- Stereo vision
- Optical flow
- Appearance

Stereo vision uses two images taken *at the same time* from different locations, optical flow uses two images taken *at different times* and appearance is based on *single* images. The next subsections take a closer look at these depth estimation methods.

**Stereo vision** Stereo vision uses images taken at the same time from different viewpoints to estimate depth. The difference in viewpoints causes the obstacle to appear in different positions in the images. The difference in these positions – the *disparity* – is inversely related to the depth of the object.

An example of depth estimation using stereo vision is shown in Figure 2.1 for an obstacle at distance $z$ observed using a stereo camera with focal length $f$ and baseline $B$. Using equal triangles, the disparity of the obstacle is:

$$d = B\ f\ z^{-1} \tag{2.1}$$

This equation can be solved for $z$ to find a distance estimate $\hat{z}$ given a disparity $d$:

$$\hat{z} = B\ f\ d^{-1} \tag{2.2}$$

The camera parameters $B$ and $f$ are found beforehand through calibration.

The main challenge of stereo vision is to find this disparity; it is often difficult to find out which pixels in the images belong to the same point in the world. A first way to categorize stereo vision algorithms is to make a distinction between sparse and dense algorithms. *Sparse* algorithms estimate the disparity of a small number of highly recognizable points in the images. The disparity accuracy tends to be good as these points are easy to match, but because only a small number of points is considered the resulting depth map can contain large holes, especially in environments with little texture. Sparse stereo algorithms are therefore a poor choice for obstacle detection, but they sometimes appear as part of Visual Odometry (VO) or Simultaneous Localization and Mapping (SLAM) algorithms. *Dense* algorithms, on the other hand, estimate the depth for the entire image. They should therefore be able to estimate the distance towards all obstacles in view.

In [5], Scharstein and Szeliski present an extensive taxonomy of dense stereo vision algorithms. According to the authors, most dense stereo vision algorithms perform the following steps to find a disparity map:

1. Matching cost computation
2. Cost aggregation
3. Disparity optimization
4. Disparity refinement

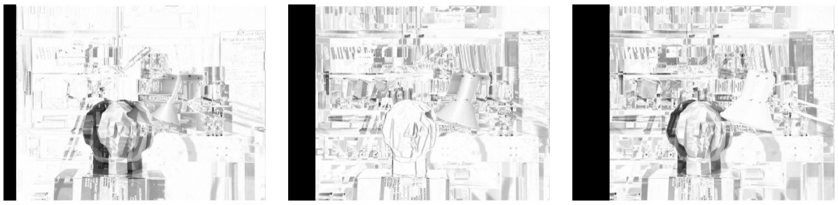
An example of these four steps is shown in Figure 2.2 for the *block matching* algorithm.

An important distinction can be made between global and local algorithms, which differ in the way the disparity optimization is performed. *Global* algorithms try to optimize a single cost function that depends on all pixel disparities. These algorithms can produce accurate depth maps even for scarcely textured scenes, but tend to be slower than local algorithms. *Local* algorithms independently optimize the disparities of pixels or small regions. These algorithms are easier to parallelize and typically faster, but less accurate.

An in-depth review of stereo vision methods is out of scope for this chapter. While it is important to understand the way these stereo vision algorithms work, their run-time performance and accuracy are perhaps more relevant for their use on UAVs. These are difficult to predict from first principles and are instead measured on benchmarks, of which the Middlebury Stereo benchmark[2] [5] and the KITTI Stereo benchmark[3] [6] are commonly-used examples.

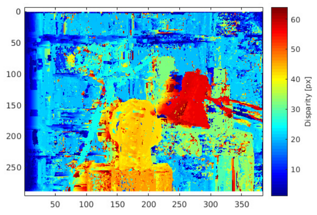
In [7], Tippetts *et al.* perform an extensive review of stereo vision algorithms for resource-limited systems. The authors collected run-time and accuracy measurements for a large number of algorithms and use these to produce scatterplots of their performance. Where possible, the run-times were normalized based on the hardware for which they were reported. The article provides an excellent starting point for the selection of stereo algorithms, its only downside being that it was written in 2012 and that it is therefore not fully up-to-date.

[2] http://vision.middlebury.edu/stereo/
[3] http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo

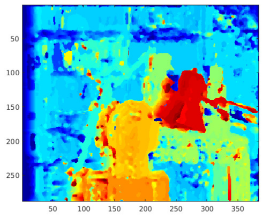


(a) *Matching cost computation*. The matching cost is calculated per pixel for all disparities under consideration. In this example the pixel difference is used as matching cost. Shown are difference images at three different disparities, where white indicates a low matching cost and black a high cost. In the left image, the disparity is roughly equal to the true disparity of the background: the background has a low matching cost (white). In the middle image, the disparity is close to that of the head; in the right image it is close to that of the lamp.

(b) *Cost aggregation*. Sometimes individual pixels can be hard to match. In this case, information on neighboring pixels can make the matching task easier. In this example, the matching cost images are convolved with a $3 \times 3$ averaging filter to take nearby pixels into account.



(c) *Disparity optimization*. Using the aggregated matching cost, the per-pixel disparity can be found through optimization. In this example, the per-pixel argmax over the disparities is used. This is a form of *local* optimization as the pixel disparities can be found independently.



(d) *Disparity refinement*. Post-processing is used to clean up the disparity map from the previous step. In this example, a median filter is used to remove outliers.

Figure 2.2: Example of the *block matching* stereo algorithm broken down into the four steps described by Scharstein and Szeliski [5]. Input image: *Tsukaba*, Middlebury Stereo Vision Dataset 2001 [5] courtesy of the University of Tsukuba.

A similar review was performed for this literature study, so that algorithms published after 2012 could also be included. Run-time and accuracy measures were obtained from the Middlebury and KITTI benchmarks. Run-time figures were not normalized, as the majority of methods are evaluated on similar platforms (CPU-based methods on an unspecified 2.5 GHz processor, GPU-based methods on an NVIDIA Titan X). The main focus of this comparison is on algorithms for which code is publicly available. The results are shown in Figure 2.3.

The following conclusions are drawn from these results: first of all, there exist close-to-optimal stereo vision algorithms for which code is publicly available. This means that it is not necessary to write an own implementation of a state-of-the-art algorithm. Secondly: from the CPU-based methods, ELAS [8] and SGM/SGBM variants [2] are still among the best performers. The inclusion of SGBM in OpenCV makes this an ideal algorithm for initial development. Thirdly: the use of a GPU can significantly increase performance, mainly in terms of accuracy. However, it is currently unclear how this performance improvement weighs up against the increase in weight and power consumption of such a platform. The 250 W power required by the NVIDIA Titan X is quite high for a UAV, and the performance benefit seen in the benchmarks might be significantly smaller on an embedded GPU.

On a higher level, stereo vision has the advantage over other depth cues that its depth estimate is based on the *baseline* between the two cameras. This is an advantage because the baseline is constant and easy to measure or calibrate. In comparison, the distance between successive images for optical flow is often unknown; it has to be estimated and therefore leads to more uncertainty in the distance estimate. Appearance cues have a similar disadvantage, the size of certain cues in the environment is not exactly known, also leading to uncertainty in the depth estimate.

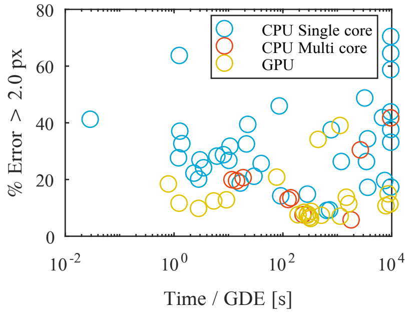
Stereo vision also has limitations. First of all it requires two or more cameras. The resulting weight will be larger for this setup than for depth estimation based on optical flow or appearance cues.

Secondly, the range of stereo vision is limited, although not as badly as commonly thought [9]. As the distance to obstacles increases, the disparity decreases inversely (see Figure 2.4). This means that for far-away objects the disparity hardly changes with distance. As a result, the sensitivity to measurement errors $\mathrm{d}\hat{z}/\mathrm{d}d$ increases with distance until it becomes impractically large:

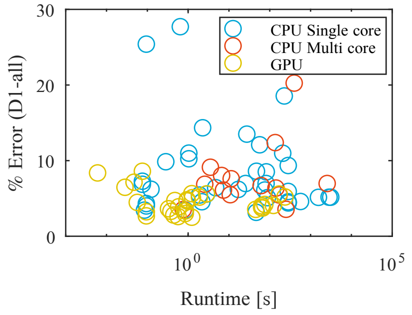
$$\frac{\mathrm{d}\hat{z}}{\mathrm{d}d} = -B\ f\ d^{-2} \tag{2.3}$$

$$= -\frac{z^2}{B\ f} \tag{2.4}$$

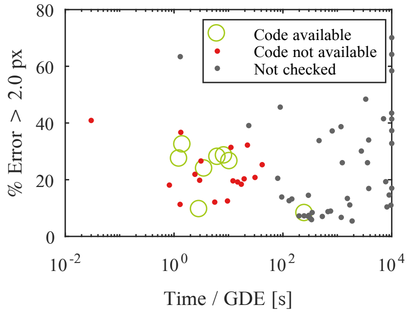
This growing uncertainty limits the maximum range of stereo vision. The disparity errors are the result of incorrect matching of pixels in the input images and are typically independent of distance. If the stereo algorithm only searches for discrete disparities, these errors will be in the order of 0.5 px at best. Stereo algorithms for long-range distances therefore need to estimate subpixel disparities. According to Pinggera *et al.*, it is possible to reach a consistent error limit of 0.1 px under real-world conditions [9]. The sensitivity



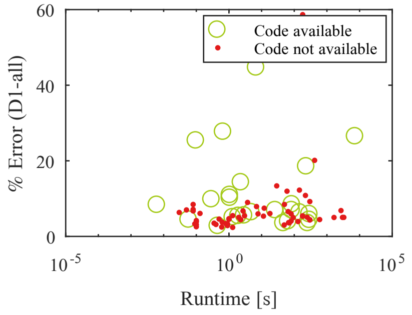
(a) Platforms and performance on the Middlebury benchmark.



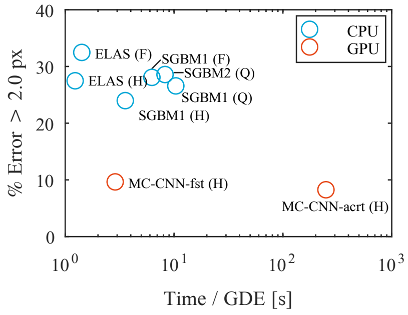
(b) Platforms and performance on the KITTI benchmark.



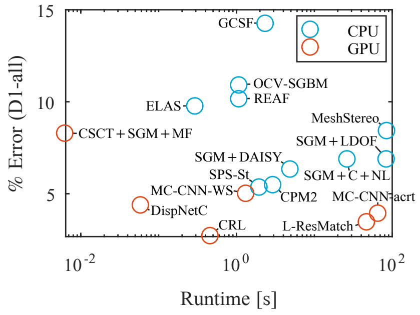
(c) Code availability on the Middlebury benchmark.



(d) Code availability on the KITTI benchmark.

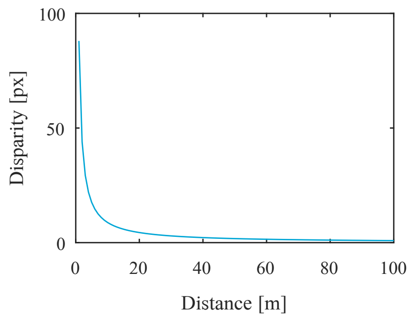


(e) Best performing methods on the Middlebury benchmark for which code is available.

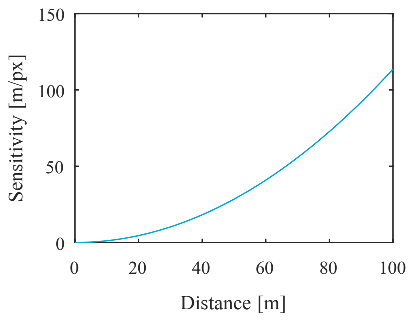


(f) Best performing methods on the KITTI benchmark for which code is available.

Figure 2.3: Scatterplots of accuracy versus runtime performance on the Middlebury[1] and KITTI[2] stereo vision benchmarks. Runtime on the Middlebury benchmark is measured in seconds per Giga Disparity Evaluations (GDE), which is found by multiplying the image width, height and maximum number of disparities. Data obtained on 27/11/2017. *a, b*: Methods running on the GPU tend to perform better than those running on the CPU. On the Middlebury benchmark they perform better in terms of accuracy, while on the KITTI benchmark they also outperform CPU methods in terms of runtime – perhaps because runtime performance is more important for automotive applications than for the static pictures of Middlebury. *c, d*: While code is not available for every method, there are enough close-to-optimal algorithms for which source code has been published. *e, f*: These methods should be considered first when choosing a stereo vision algorithm, as they perform well and their code is publicly available. Popular choices are ELAS [8] and SGM/SGBM [2]; the latter is also included in OpenCV.
[1]: https://vision.middlebury.edu/stereo/ [5]
[2]: http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo (CC BY-NC-SA 3.0)



(a) Disparity vs. distance. As the distance increases, the disparity converges to zero. For faraway objects the disparity hardly changes with distance anymore.



(b) Sensitivity vs. distance. The sensitivity is defined as $-\mathrm{d}z/\mathrm{d}d$, i.e. the distance error for a 1 px error in the disparity estimate.

Figure 2.4: Maximum range of stereo vision. As the distance increases, the sensitivity to stereo matching errors increases quadratically. Example plots generated for a camera with baseline $B = 20\,\mathrm{cm}$ and focal length $f = 400\,\mathrm{px}$. Best-case disparity errors are in the order of $0.5\,\mathrm{px}$ to $0.1\,\mathrm{px}$ [9] depending on the algorithm.

to measurement errors can also be reduced by increasing the baseline $B$ or focal length $f$ of the cameras.

Finally, the matching of features between the input images is often a weak point of stereo vision. As a result, it may perform badly with the following obstacles: textureless surfaces, finely or repetitively textured surfaces, textures oriented parallel to the baseline, reflections and transparency. Furthermore, depending on the algorithm, slanted surfaces and occlusions can be problematic.

**Optical flow** Optical flow tracks the movement of image features over time. In a static environment, the shift of these features depends on the movement of the camera and the distance to the features; in general, features further away from the camera will move less than those that are nearby. If the movement of the camera is known, the distance to the features can be obtained. When only the rotation is known the distance cannot be found; however it is still possible to estimate the time-to-contact, which is sufficient for some forms of obstacle avoidance.

Figure 2.5 shows an example of optical flow and its use for depth estimation. The example assumes forward motion[4] at a known velocity without rotation of the camera. Given the obstacle's position $(x, z)$ and the camera's focal length $f$, its image position $u$ can be found using equal triangles:

$$u = x\ f\ z^{-1} \tag{2.5}$$

Taking the time derivative produces the instantaneous optical flow $\dot{u}$ of the obstacle or

[4] Optical flow from sideways or vertical motion has slightly different characteristics, but will not be explained here to keep the explanation short. A forward-facing camera is the most relevant example for obstacle avoidance.

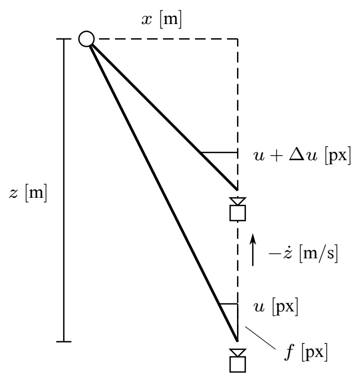


Figure 2.5: Optical flow for forward motion. The image position $u$ of an obstacle located at $(x, z)$ changes as the UAV moves forward with a velocity of $-\dot{z}$.

feature:

$$\begin{aligned} \dot{u} &= -x\ f\ z^{-2}\ \dot{z} && (2.6) \\ &= -u\ z^{-1}\ \dot{z} && (2.7) \end{aligned}$$

In practice, however, the optical flow is estimated between two images separated by a time interval $\Delta t$. The result is a shift in position $\Delta u$ instead of the flow $\dot{u}$:

$$\begin{aligned} \Delta u &\approx \dot{u}\ \Delta t && (2.8) \\ &\approx -u\ z^{-1}\ \dot{z}\ \Delta t && (2.9) \end{aligned}$$

The depth $\hat{z}$ can be found by solving this equation for $z$:

$$\hat{z} = -u\ \dot{z}\ \Delta t\ \Delta u^{-1} \quad \forall \Delta u \neq 0 \implies \forall u \neq 0 \tag{2.10}$$

and, if velocity $\dot{z}$ is not available, the time-to-contact $\tau$ is found using:

$$\begin{aligned} \tau &= \hat{z}/\dot{z} && (2.11) \\ &= -u\ \Delta t\ \Delta u^{-1} && (2.12) \end{aligned}$$

Note, however, that from Equation 2.7 and 2.9 it follows that the flow $\dot{u}$ and shift $\Delta u$ will be zero in the center of the image where $u$ is zero (the Focus-of-Expansion). It is therefore not possible to estimate depth at the Focus-of-Expansion as the result is undefined.

The main problem of optical flow is not the estimation of depth but the tracking of features between images. It is therefore very similar to stereo vision. The main difference, however, is that stereo vision only searches for matches along one dimension, while optical flow is two-dimensional. Optical flow is therefore more difficult to compute.

As in stereo vision, a distinction can be made between sparse and dense optical flow algorithms. *Sparse* algorithms track highly recognizable points, typically corners. Sparse tracking is frequently found in VO or SLAM. Like sparse stereo vision, sparse optical flow is not suitable for obstacle detection as it may leave large holes in the depth map. *Dense* algorithms estimate optical flow for the complete image and are therefore better suited for obstacle detection.

An overview of optical flow techniques is presented in [10]. The survey is similar to [5] in that it breaks down the algorithms into a few key components. According to Baker *et al.*, most dense optical flow algorithms perform a global optimization (i.e. for all pixels at the same time) of the following energy function: $E_{\text{data}} + \lambda E_{\text{prior}}$, where the *data term* $E_{\text{data}}$ follows from the content of the images (similar to the matching cost in stereo vision) and the *prior term* $E_{\text{prior}}$ encodes assumptions of the flow field such as its smoothness [10]. The final component of a dense optical flow algorithm is the *optimization algorithm*.

An in-depth overview of optical flow algorithms is again beyond the scope of this chapter. Instead, existing optical flow algorithms are compared by benchmark results. The results are obtained from Baker *et al.*, 2011 [10] (the more up-to-date Middlebury website[5] unfortunately does not report run-times) and from the KITTI optical flow 2015 benchmark[6] [6]. The results are shown in Figure 2.6.

The KITTI results show that code is available for fast and accurate optical flow estimation on GPUs. Code for the best performing CPU-based algorithms is not available; SPyNet [11] might be used as an equally fast alternative, but it has a higher error percentage than the best-performing algorithms. The other CPU-based algorithms for which code is available have run-times larger than one second. While it may be possible to reduce their run-times by, for instance, lowering the resolution of the images, there is no guarantee that they will run fast enough for practical use in obstacle avoidance. From the results of Baker *et al.*, 2011 only FOLKI has a run-time of one second, while the others are in the order of ten seconds or more.

Compared to stereo vision, the main advantage of optical flow is that it only requires a single camera, which saves weight. However, optical flow also has a number of disadvantages. First of all, if a metric depth estimation is required, the velocity of the UAV should be known. Estimation of this velocity is not trivial and uncertainties in this estimate are an additional source of error for depth estimation.

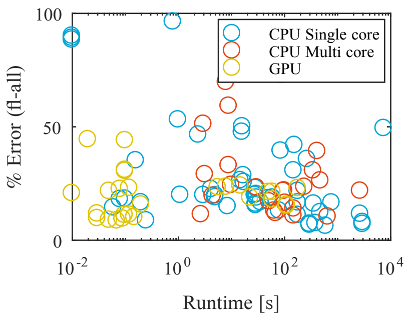
A second problem is that the optical flow approaches zero near the FoE. By definition the FoE lies in the direction of travel, exactly the place where obstacles *should* be detected. Since the flow needs to be inverted to estimate distance, this makes the depth estimate extremely sensitive to measurement errors in shift $\Delta u$. This is demonstrated with the sensitivity $\mathrm{d}\hat{z}/\mathrm{d}\Delta u$, i.e. the error in the distance estimate for a 1 px error in $\Delta u$, which is found by differentiating Equation 2.10 with respect to $\Delta u$:

$$\frac{\mathrm{d}\hat{z}}{\mathrm{d}\Delta u} = u\ \dot{z}\ \Delta t\ \Delta u^{-2} \tag{2.13}$$

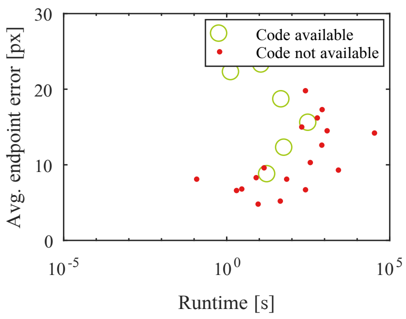
$$= \frac{z^2}{\dot{z}\ u\ \Delta t} \tag{2.14}$$

For reference, the best average end-point errors in the KITTI optical flow 2012 benchmark[7] [13] lie in the order of 1 px. The expected flow and sensitivity are shown in Figure 2.7 for a

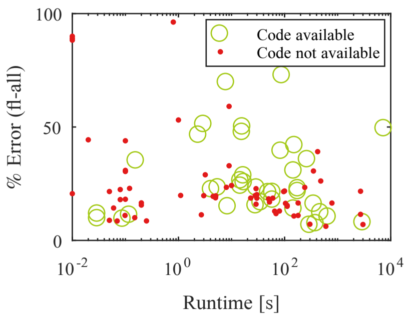
[5] http://vision.middlebury.edu/flow/
[6] http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=flow
[7] http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=flow



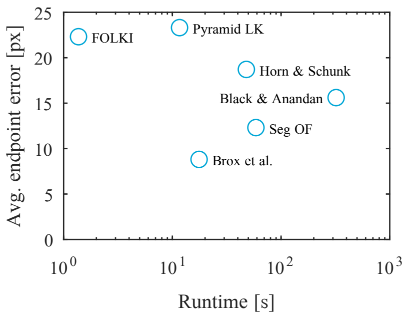
(a) Platform and performance on the KITTI benchmark.



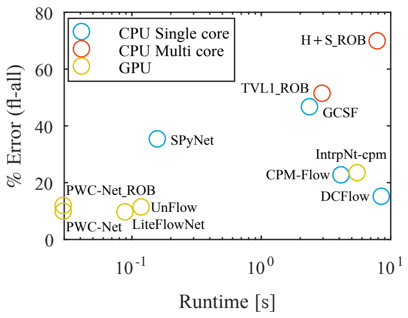
(b) Code availability of the methods reviewed in Baker *et al.*, 2011.



(c) Code availability on the KITTI benchmark.

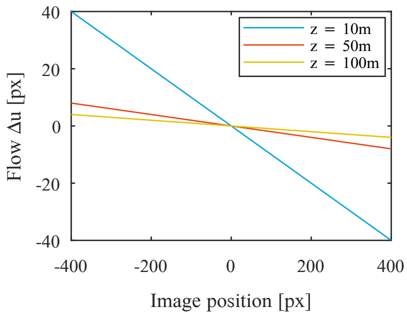


(d) Best performing methods in Baker *et al.*, 2011 for which code is available.

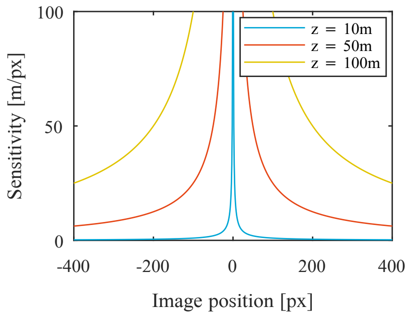


(e) Best performing methods on the KITTI benchmark for which code is available.

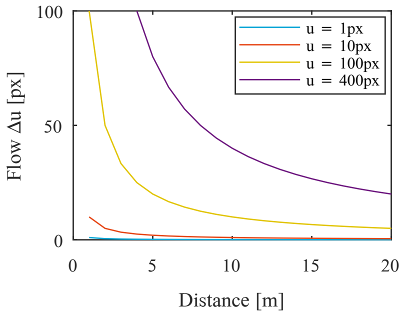
Figure 2.6: Scatterplots of dense optical flow estimation accuracy and run-time performance. Data is obtained from Baker *et al.*, 2011$^1$ [10] and the KITTI optical flow 2015 benchmark$^2$ [6] (data obtained on 28/08/2018). *a*: GPU-based methods tend to have lower runtimes and error percentages than CPU-based algorithms. (Platform information is not available for Baker *et al.*, 2011). *b, c*: Of the methods listed in Baker *et al.*, 2011, source code is not available for the best performing ones. This is slightly better for the KITTI benchmark. *d, e*: These are the best performing methods for which code is publicly available. GPU-based methods perform significantly better than CPU-based ones. There is little overlap between Baker *et al.*, 2011 and KITTI in terms of algorithms, but note that there is a 7-year gap between the two benchmarks. The algorithm by Brox *et al.* is included in OpenCV [12]. For the other methods code is available, but it might take more work to integrate these into research code.
$^1$: Baker *et al.*, 2011 [10] (CC BY-NC 2.0)
$^2$: `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=flow` (CC BY-NC-SA 3.0)



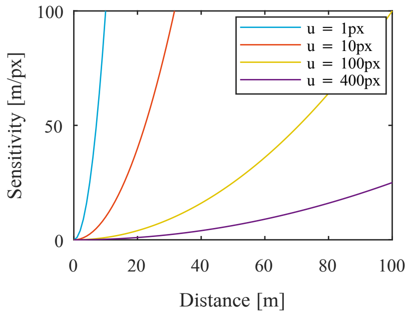
(a) Expected optical flow $\Delta u$ as a function of image position $u$ for three obstacle distances $z$.



(b) Sensitivity to measurement errors in $\Delta u$ as a function of image position $u$ for three obstacle distances $z$.



(c) Expected optical flow $\Delta u$ as a function of obstacle distance $z$ for four image positions $u$.



(d) Sensitivity to measurement errors in $\Delta u$ as a function of obstacle distance $z$ for four image positions $u$.

Figure 2.7: Example of expected optical flow and sensitivity to measurement errors in $\Delta u$. Data generated for a camera traveling at $10\,\mathrm{m/s}$ with an optical flow algorithm running at $10\,\mathrm{Hz}$ ($\Delta t = 0.1\,\mathrm{s}$). Plot *b* shows that the sensitivity to errors strongly increases near the center of the image and approaches infinity at the Focus-of-Expansion (FoE). Plot *d* shows that obstacles near the FoE ($u = 1\,\mathrm{px}$) can only be detected at short ranges where the sensitivity to measurement errors is low, while the range is significantly larger near the edge of the image ($u = 400\,\mathrm{px}$).

drone traveling at $10\,\mathrm{m/s}$. The conclusion drawn from this figure is that it may be difficult to get an adequate measurement range near the FoE, as the sensitivity to measurement errors rapidly increases for $|u| < 100\,\mathrm{px}$.

Equation 2.14 suggests a few ways to reduce the sensitivity to errors. First of all, the UAV can fly faster; this results in larger flow vectors relative to the measurement error. Secondly, the frame rate can be *reduced*, this will also increase the size of the flow vectors. Note, however, that there is an upper limit to $\Delta t$ as the resulting $\Delta u$ should remain small enough that features remain in view. The frame rate should also remain high enough to detect obstacles in time. Finally, the sensitivity can be reduced by using a higher-resolution camera or a zoom lens, as $u$ will be larger (note that the sensitivity does not depend on the camera's focal length).

The final disadvantage of optical flow is that it requires sufficient texture to match pixels between successive images. Like stereo vision, it can produce incorrect results for textureless surfaces, finely or repetitively textured surfaces, reflections and transparency.

Not mentioned in this review is *scene flow*, the 3D equivalent of optical flow. The result of scene flow is a 3-dimensional velocity vector for each pixel, together with a depth or disparity. A review of this field is left for future work.

**Appearance** Unlike stereo vision or optical flow, appearance cues can be found inside a *single* image. As humans we are already familiar with appearance-based cues because we use them all the time, such as when looking at photographs. Photographs do not contain disparities since they are flat, nor do they produce optical flow as they do not move. Still, it is possible to estimate depth from these images; this is the field of *monocular depth estimation*.

'Appearance' is not really a single cue, as is the case for stereo vision which relies entirely on disparities or optical flow which results only from the flow vectors. Instead, appearance cues are a collection of image features that depend in one way or another on depth. An extensive treatment of depth cues used by humans can be found in [14]. The following is a non-exhaustive list of appearance cues:

- Occlusion. Nearby objects cover those further away.
- Image size of known objects. Using the focal length of the camera, this can be transformed back into a distance estimate.
- Different image sizes of similar objects. The objects that appear smaller in the image are further away.
- Perspective. Parallel lines in the environment appear to converge in the image, their distance provides an indication of depth.
- Vertical image position. Objects that appear higher in the image are further away.
- Texture gradient. Surface textures will appear more fine-grained if they are further away.
- Light and shadow. This cue is especially relevant for surface relief. Light typically comes from above, brighter regions are assumed to face upwards.

- Atmospheric haze. Far-away objects take on a blue-ish tint.
- Sky segmentation. The sky is infinitely far away.

Most of these cues require knowledge about the environment, such as the presence of a flat ground or parallel lines, knowledge about the size of objects, and so on. This makes appearance-based depth estimation more difficult to implement than stereo vision or optical flow. If it is even possible to implement some of these cues, this quickly leads to rather ad-hoc solutions. For this reason, appearance-based cues have seen relatively little use in computer vision until recently.

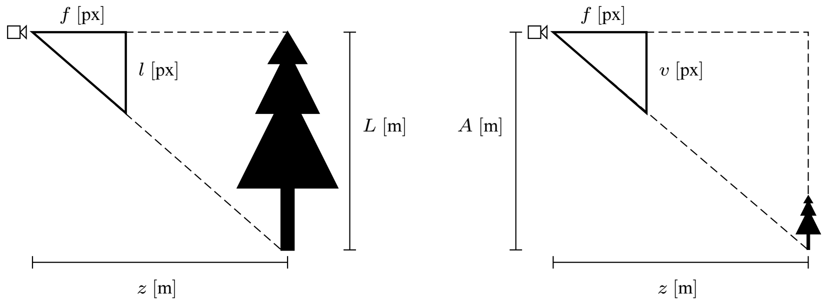
One of the first practical examples of monocular depth estimation for arbitrary outdoor images is Saxena *et al.*'s *Make3D* [15, 16], first published in 2006. The system relies on a combination of superpixel segmentation and hand-crafted features. These are fed into a Markov Random Field (MRF) to model the relations between the regions in the image. Another example of monocular depth estimation using classical machine learning methods, this time based on the texture gradient cue, is given in [17].

The field of monocular depth estimation really took off with the arrival of Deep Learning. Using Convolutional Neural Networks (CNNs), it is no longer necessary to develop feature descriptors by hand. Instead, these features and the relations between them are learned from a large dataset of example images. Eigen *et al.* are the first to use a CNN for monocular depth estimation in [18, 19]. Their network is trained on color images labeled with the true depth map obtained with a Kinect (NYU Depth v2) or LIDAR (KITTI). The first example of *Self-Supervised Learning* for depth estimation is published in 2016 by Garg *et al.* [3]. Instead of training to predict a depth map, their CNN is trained to predict the other image in a stereo pair. Deep learning has made it possible to use appearance for depth estimation by taking away the need to manually implement an estimator for these cues. Section 2.2.2 will go into more detail on Deep Learning for depth estimation.

Appearance-based depth estimation has the advantage that it only requires a single camera. Unlike optical flow, however, it can work without an estimate of the UAV's velocity. Secondly, appearance-based depth estimation relies on different features than stereo vision and optical flow. As a result, appearance cues may work better for obstacles where the previous algorithms are likely to fail. Appearance-based depth estimation could therefore be a valuable addition for depth estimation, but this is not yet proven. Whether obstacle avoidance will truly benefit from appearance cues is still an open question.

The main disadvantage of appearance-based depth perception is that it is inaccurate, especially with regards to scale. Monocular depth perception lacks a reliable reference length by which the scene can be scaled. In stereo vision this is provided by the baseline between the cameras; in optical flow by the distance between the two images. In monocular depth estimation, the only obvious source of this information is the known size of objects, but this has to be learned from the training set and may vary between different object instances.

The depth scale, however, is not the only problem of monocular depth estimation. The relative depth between objects also suffers from large inaccuracies. This is effectively demonstrated by Smolyanskiy *et al.* in [20]. The authors show that the depth map produced by *MonoDepth* [21] looks visually correct; however, an overhead view of the



(a) Depth estimation using the known size $L$ of an object and its image size $l$.

(b) Depth estimation using the vertical image position $v$ and the altitude $A$ of the UAV.

Figure 2.8: Two examples of depth estimation based on appearance features.

resulting point cloud shows that this is clearly not the case. It is not clear whether this is a limitation of MonoDepth or its training set, or a more fundamental issue with monocular depth estimation.

Estimating the sensitivity to measurement errors of appearance cues is a bit more difficult than for stereo vision or optical flow as the cues are not always clearly defined or based on simple geometry. An attempt is made to model the uncertainty of the two depth cues: the size of known objects in the image and the vertical position of objects in the image. These examples are shown in Figure 2.8. Using equal triangles, the image size $l$ of an object can be found as follows:

$$l = f\ L\ z^{-1} \tag{2.15}$$

Similarly, given the drone's height $A$ above the terrain, the vertical position $v$ in the image is found using:

$$v = f\ A\ z^{-1} \tag{2.16}$$

Note that these equations are exactly the same when $L = A$ and $l = v$. For brevity only the first cue will be discussed in more detail, the results also apply to the second case.

Equation 2.15 can be solved for $z$ to produce a depth estimate $\hat{z}$:

$$\hat{z} = f\ L\ l^{-1} \tag{2.17}$$

There are two sources of uncertainty in this equation. First of all, there may be a small error in the length measurement $l$ in the image. Sensitivity to this error is found to be:

$$\frac{\partial \hat{z}}{\partial l} = -f\ L\ l^{-2} \tag{2.18}$$

$$= \frac{z^2}{f\ L} \tag{2.19}$$

| | | | |
|---|---|---|---|
| $B$ | 20 cm | $\epsilon_d$ | 0.1 px |
| $f$ | 400 px | $\epsilon_{\Delta u}$ | 1.0 px |
| $\dot{z}$ | 10 m/s | $\epsilon_l$ | 2 px |
| $\Delta t$ | 0.1 s | $\epsilon_L$ | 3 m |
| $L$ | 10 m | $\epsilon_A$ | 10 m |
| $A$ | 100 m | | |

Table 2.1: Parameters used to generate Figure 2.9. Error bounds $\epsilon_l$, $\epsilon_L$ and $\epsilon_A$ are an educated guess, the bounds $\epsilon_d$ and $\epsilon_{\Delta u}$ are based on literature and the KITTI benchmark.

While this sensitivity also increases quadratically with distance, its magnitude remains relatively small compared to the errors of stereo vision or optical flow: when observing an object with size $L = 10\,\text{m}$ (e.g. a tree, or the length or wingspan of a Cessna 172) at a distance of 100 m with a focal length of $f = 400\,\text{px}$, the sensitivity to length measurement errors is only 2.5 m/px, compared to $\sim$ 100 m/px for a stereo camera with baseline $B = 20\,\text{cm}$ and the same focal length.

The second source of error is uncertainty about the object's true size $L$. Sensitivity to these errors is found as follows:

$$\frac{\partial \hat{z}}{\partial L} = f\, l^{-1} \tag{2.20}$$

$$= \frac{z}{L} \tag{2.21}$$

Note that unlike all error sensitivities found before, this one only grows linearly with distance. This suggests that appearance-based depth estimation might have an advantage over stereo vision or optical flow at longer distances, as long as the error in the image length measurement $l$ remains sufficiently small.

Sensitivity to errors in the image length measurement (Equation 2.19) can be reduced with a larger focal length $f$. There is, however, no way to reduce the sensitivity to errors in $L$ (Equation 2.21), as $L$ mainly depends on the object that happens to be in front of the UAV.

In the example of the vertical image position, however, $L$ is equal to the altitude of the drone. This altitude is mostly likely larger than the size of objects the drone will encounter, which means this depth estimation method will be more accurate than using the size of the object. Secondly, the sensitivity to errors can in this case be reduced by flying higher, thereby increasing $L$.

This section on sensing is concluded with a comparison of the expected errors of stereo vision, optical flow and appearance-based depth estimation. The expected error is calculated by multiplying the sensitivity (e.g. $\mathrm{d}\hat{z}/\mathrm{d}d$) with an estimated upper bound on said error (e.g. $\epsilon_d = 0.1\,\text{px}$ for stereo vision with subpixel disparities). Note that this is only a first-order approximation of the error, the results may not be realistic as the expected error approaches or exceeds the true distance $z$. A comparison chart of the depth estimation methods is shown in Figure 2.9. The parameters used to generate this chart are listed in Table 2.1.

While the results should be taken with a grain of salt, they do highlight the trends found in this literature review. The error of optical flow is prohibitively large near the

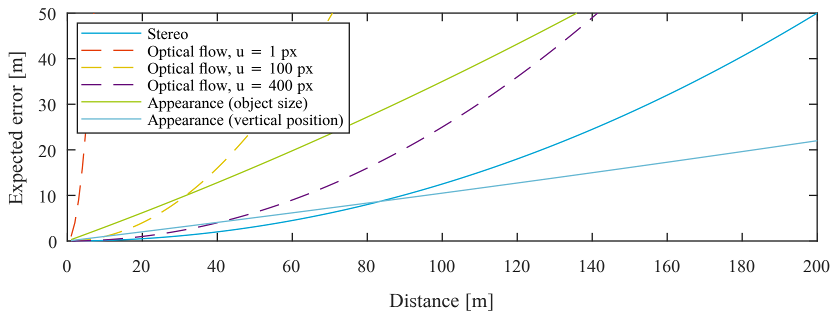


Figure 2.9: Comparison of expected error bounds for selected depth estimation methods.

center of the image ($u = 1\,\text{px}$ and $100\,\text{px}$), but comparatively decent near the edge of the image ($u = 400\,\text{px}$). The error could be reduced by flying faster, a speed of $10\,\text{m/s}$ was assumed for this comparison. Stereo vision appears to be the best choice in this scenario for obstacles up to a distance of $\sim 80\,\text{m}$. Unlike optical flow, however, this depth estimate should also be accurate near the center of the image. The result plotted here is based on a stereo vision algorithm that can estimate subpixel disparities. Finally, at larger distances the depth estimate based on the vertical position of objects performs best, due to its predominantly linear increase in sensitivity to measurement errors.

## Avoidance

When an obstacle is detected along the UAV's direction of travel, it should perform an avoidance maneuver to prevent a collision. There are different ways to handle this, from the very simple and lightweight reflexive behaviors, to high-level planning in maze-like environments.

The execution of an avoidance maneuver typically requires the following components: 1) *motion planning*, which determines the actions the UAV should take; 2) a *map*, a representation of the obstacles in the vicinity of the UAV and 3) *odometry*, which is often required to accurately perform the planned maneuver. These components will be briefly discussed in the following subsections.

**Motion planning** Motion planning determines the action the UAV should take to avoid collisions while moving towards its goal location. An overview of motion planning and obstacle avoidance algorithms can be found in [22, 23].

Minguez *et al.* [23] make a distinction between global planning and local planning (called 'motion planning' and 'obstacle avoidance' in their article, these terms will not be used here to avoid confusion with the overall task of obstacle avoidance). *Global planning* assumes that the location of all obstacles is known, the goal is to find a trajectory that optimizes a given performance measure. *Local planning* assumes that only obstacles detected by the UAV's sensors are known. The goal here is to adapt the current trajectory of the UAV to avoid a collision with nearby obstacles. Local planning has the disadvantage that it can get trapped in certain situations (mazes for example, but these situations are unlikely in outdoor flight). However, unlike global planning it can function in unknown environments. Local planning is therefore the most relevant for UAV obstacle avoidance.

Motion planning algorithms can be broadly divided into the following classes: reactive planning, planning without dynamics and planning with dynamics. *Reactive planning* refers to a class of algorithms that prescribe a control input or motion based directly on the presence of obstacles. An example is the use of potential fields to determine the direction of travel of the UAV: detected obstacles 'repel' the drone, preventing a collision. In *planning without dynamics* the goal is to find a path for the UAV that guides it past the detected obstacles. This path should also minimize a cost function, setting these algorithms apart from reactive planning. Once a path is found, it is left to a lower-level controller to actually follow it. An example is [24] where a Rapidly-exploring Random Tree (RRT) is used to plan a path through a forest. *Planning with dynamics* also optimizes a cost function, but includes a dynamic model of the UAV. Model Predictive Control (MPC) is an example of this. The inclusion of dynamics ensures that the maneuver can actually be performed, but requires a dynamic model of the UAV to be available. The use of dynamics is particularly suitable for high-performance manneuvers (e.g. drone racing), while planning without dynamics is more suitable for general-purpose applications as it does not require a model.

For brevity this section only lists examples of algorithms. The reader is referred to the cited reviews for a more extensive overview of methods.

**Maps** Motion planning requires a map, but the exact function of the map differs per algorithm. At the very least, the map serves to document the location of nearby obstacles; even reactive planning will need this information. For more complicated algorithms, the map allows the planning of an avoidance maneuver around the obstacles. Finally, a map allows multiple observations of obstacles to be combined, which is the basic idea behind SLAM.

Maps can be made at different levels of detail. Ground robots and autonomous cars often create highly detailed maps of their immediate surroundings. These types of maps are also applicable to UAVs flying at low altitudes or indoors, but their creation is computationally intensive. An example of less detailed maps for aircraft is the Enhanced Ground Proximity Warning System (EGPWS), which uses relatively coarse-scaled static maps to prevent terrain collisions on passenger aircraft. Such a map could also be used on UAVs as a form of geofencing, but this would primarily apply to cruise flight as such a static map is difficult to keep up to date at a high enough level of detail for take-offs and landings.

Table 2.2 lists map types that could be used to model the immediate surroundings of the UAV during flight. The maps are divided into three classes: image-space maps, discretized space maps and continuous space maps. *Image space* maps are essentially the same as depth maps: they consist of pixels for which the distance towards the first obstacle is stored. *Discretized space* maps split the surroundings of the drone into a collection of discrete cells that can be free or occupied. These maps are commonly used for range-sensor-based SLAM on indoor robots. Finally, *continuous space* maps do not discretize the space around the UAV, but store a continuous position estimate for each measurement point. A point cloud is a typical example of this map, but it is also possible to track the position of entire objects.

The table furthermore lists the following properties: *computational complexity* gives an

| | Image-space | Discretized space (voxels) | | Continuous space | |
|---|---|---|---|---|---|
| | | Cartesian | Polar | Point cloud | Obstacle positions |
| Computational complexity | Low | High | High | High | Low |
| Volumetric | 2.5D[a] | Yes | Yes | No[b] | No[c] |
| Probabilistic | No | Occupancy | Occupancy | Position | Position |
| Dynamic | No | No | No | Yes | Yes |
| Single-frame | Yes | Yes | Yes | Yes | Yes |
| Multi-frame | No | Yes | No | Yes | Yes |
| Reference frame | Body | World | Body | Any | Any |

[a]Volumetric in horizontal and vertical directions but not in depth.
[b]It is possible to fit a mesh on the point cloud or assume a small, fixed volume around each point.
[c]A fixed volume can be assumed for the obstacle, if known.

Table 2.2: Overview of common properties of map types. This table only lists the typical properties of these maps; exceptions can likely be found for many entries in this table. Note that combinations of these maps are possible (e.g. a cartesian voxel map for static obstacles and an EKF for the positions of other aircraft).

indication of the amount of processing power and memory required to build and maintain the map. Single-frame maps are relatively lightweight, as are filter-based maps if they have small covariance matrices (e.g. an EKF of obstacle positions). Maps that are constructed from multiple image frames tend to require more processing and memory. The next properties describe the information a map may contain: *volumetric* maps describe non-zero volumes of space, such as voxels. On the other hand, non-volumetric maps such as point clouds model obstacles using infinitesimally small points. Additional processing (e.g. mesh- or surface fitting, or expanding the points by a certain volume) of these maps is required before they can be used to test for collisions. *Probabilistic* maps can explicitly model uncertainty; either in the occupancy of parts of the environment, or in the position estimate of points or obstacles. *Dynamic* maps can also model the velocities of obstacles. Continuous-space maps are particularly suitable for this, as they can be updated over multiple frames without introducing quantization errors by storing intermediate states into a discretized map.

In principle all of the maps in Table 2.2 can successfully be used on UAVs, but it depends on the application which map is the most suitable. The most important decision is whether the map should combine multiple measurements (*multi-frame*) or represent only a single measurement (*single-frame*). Combining multiple measurements allows the drone to map large and complex environments; it is therefore particularly suited for indoor operations but its use is limited to larger drones as the underlying algorithms can be computationally intensive. Cartesian voxel maps are a common choice for this application (e.g. [25, 26]). If the environment is simple enough that it can be captured in a single measurement, then image-space maps are a logical choice as these require very little processing to create and because other map types do not provide additional advantages if they do not fuse multiple measurements. An example of the use of an image-space map for UAV obstacle avoidance is found in [24]. For the avoidance of other aircraft, a continuous-space map is a good choice as such a map is easy to update and can also model the velocity of the other aircraft. An Extended Kalman Filter (EKF) with the states of the detected aircraft is an example of such a map.

**Odometry** To perform all but the most basic avoidance maneuvers, the UAV will need an estimate of its velocity. Outdoors GPS is often available, but reflections can make it inaccurate in densely built areas. Indoors, GPS is not available for navigation so a different solution needs to be found.

A common solution for GPS-less flight is Visual Odometry (VO), where a camera is used to estimate the velocity of the drone. The simplest methods directly transform the optical flow from a bottom-facing camera into a velocity estimate; this is commonly combined with sonar measurements to provide a sense of scale. More complex VO algorithms are closely related to SLAM but lack loop closure capabilities. These algorithms often estimate the UAV's pose relative to a *keyframe*. The use of a keyframe instead of the integration of velocities prevents drift over time; errors only accumulate when new keyframes are created.

VO algorithms can be separated into dense and sparse algorithms, and direct and indirect methods. A good description of these categories is provided in the introduction of [27]. The dense and sparse attributes are similar to those in stereo vision and optical flow: sparse algorithms only track a small number of keypoints, while dense algorithms use the entire input image. Direct and indirect refers to the way that keypoints are matched or tracked: direct methods rely only on the intensities of neighboring pixels, while indirect methods first need to construct feature descriptors.

Both monocular, stereo and RGB-D vision can be used for VO. Stereo and RGB-D have the advantage that the map can be initialized from a single observation; this is not the case for monocular VO as one observation can only provide the bearing of the keypoints. Since a depth map is already required for obstacle avoidance, it should also be used for VO.

Another design consideration is the use of an Inertial Measurement Unit (IMU). IMUs measure accelerations and angular velocities, which can be integrated to track the drone's pose. Additionally, it can provide an estimate of the gravity vector. The IMU typically has a higher update rate than the camera and is also not sensitive to the appearance of the environment. The integration of small measurement errors, however, causes the pose estimate to drift over time – especially in the horizontal plane [28]. It is therefore not practical to rely solely on the IMU, it needs to be fused with other measurements like VO. There are two approaches to the fusion of IMU data with VO: tight coupling and loose coupling. With tight coupling, the IMU measurements are used in the same filter that performs the visual pose estimation, for instance in the update step of an EKF. With loose coupling, the vision-based pose estimate is calculated separately, after which a second filter is used to fuse it with the IMU measurement. Tight coupling produces more accurate results, but loose coupling might be easier to implement with existing autopilot filters. A second use of the IMU is in feature tracking. The IMU can be used to predict the next position of keypoints; this estimate can reduce the search space for tracking. An example of this principle can be found in [29].

An overview of VO methods is presented in Table 2.3. UAV obstacle avoidance applications should prefer methods that use stereo or RGB-D input together with the IMU.

### Performance evaluation

Once a Collision Avoidance System (CAS) has been implemented, its performance should be evaluated. The literature review on this subject can be kept brief: hardly any literature

| Method | Camera | IMU | S/D | Descriptor | Code available | Platform |
|---|---|---|---|---|---|---|
| DSO [27] | Mono | – | Sparse | Direct | https://github.com/JakobEngel/dso | CPU, laptop 5× real-time |
| SVO2 [30] | Mono (incl. fisheye, catadioptric), stereo | Yes (optional) | Sparse | Direct | Binary only, incl. armhf. http://rpg.ifi.uzh.ch/svo2.html | Laptop, smartphone (100fps), MAV |
| ORB-SLAM2 [31] | Mono, stereo, RGB-D | – | Sparse | ORB | https://github.com/raulmur/ORB_SLAM2 | Core i7 real-time |
| Usenko *et al.* [32] | Stereo | Yes | Semi | Direct | – | – |
| OKVIS [33] | Mono, stereo | Yes | Sparse | BRISK | https://github.com/ethz-asl/okvis | Real-time (platform not mentioned) |
| ORB-SLAM [34] | Mono | – | Sparse | ORB | https://github.com/raulmur/ORB_SLAM | – |
| ROVIO [35] | Mono | Yes | Sparse | Direct | https://github.com/ethz-asl/rovio | CPU 1-core 33fps |
| SVO [36] | Mono | No | Sparse | Direct | https://github.com/uzh-rpg/rpg_svo | Embedded on MAV 55fps, Laptop 300fps |
| Schmid *et al.* [25] | Stereo | Yes | Sparse | Direct | – | MAV |
| LSD-SLAM [37, 38] | Mono (incl. fisheye, catadioptric), stereo | No | Dense | n/a | https://github.com/tum-vision/lsd_slam (without stereo support) | CPU 40× real-time |
| eVO [29] | Stereo | Gyro only | Sparse | Direct | – | CPU 1.86 MHz 2-core 12 – 56 ms per frame |
| DVO [39] | RGB-D | Yes (optional) | Dense | n/a | https://github.com/tum-vision/dvo_slam | CPU 1-core 30fps |
| MSCKF 2.0 [40] | Mono, stereo | Yes | Sparse | Direct | 3rd-party implementations on github | CPU i7 2.66 GHz 1-core 10 ms per frame |
| DTAM [41] | Mono | – | Dense | n/a | openDTAM | GPU |
| PTAM [42] | Mono | – | Sparse | Direct | http://www.robots.ox.ac.uk/~gk/PTAM/ | – |

Table 2.3: Overview of VO algorithms.

exists on this topic. Most articles on obstacle avoidance demonstrate their method in an example application, but there is no common benchmark on which they can be compared.

A first step towards such a benchmark was taken in [43]. One of the core ideas of this paper is that the obstacle avoidance task can be split into smaller subtasks that can be evaluated independently. For instance, the accuracy of obstacle detection can be evaluated independently from the UAV's motion planning algorithm or state estimator.

The main difficulty in the development of a benchmark is to find suitable metrics to describe the avoidance problem: the metrics should be chosen such that different environments with the same metrics (e.g. obstacle density, typical obstacle size) result in the same behavior. It should be possible to predict the performance of an obstacle avoidance system when all relevant metrics of the target environment are available. Such a benchmark would be extremely valuable both for UAVs and other types of robots.

A similar lack of benchmarks exists for robot navigation. A proposal for a navigation benchmark is found in [44], perhaps this paper can also serve as inspiration for an obstacle avoidance benchmark.

### 2.2.2. Deep Learning for depth perception

Because of strict weight constraints, UAV obstacle detection is strongly dependent on vision. While earlier vision algorithms had to be designed and tuned by hand, the arrival of Deep Learning allows depth estimation to be learned from large datasets. This section presents an overview of recent literature and developments in the field of depth perception. Since the first application of a Convolutional Neural Network (CNN) for depth perception in 2014 [18] this field has been rapidly evolving. This is also illustrated by the articles cited in this section, as the majority of them were uploaded to ArXiv between June 2018 and now. Each month, roughly ten new relevant papers appear on ArXiv.

Section 2.2.2 describes different depth perception tasks. Section 2.2.2 will discuss the training of these networks including a brief overview of commonly used datasets. Finally, section 2.2.2 presents some works on the analysis of networks after they have been trained.

#### Problems in depth perception

While the goal of depth perception is clear – the estimation of a depth map from input images – there are a few ways this problem is formulated in literature. The most common problem that is solved in literature is *depth prediction*: generating a depth map using only one or more input images. A second problem in literature is that of *depth completion*. In this case, a partial depth map is already available, such as the depth towards VO or SLAM keypoints. The goal of the neural network is then to fill in the missing parts of the depth map. Finally, recent literature has shifted towards the *combination of depth perception with other tasks* in the same network. For instance, a single network performs both depth estimation and object segmentation. The next subsections look more closely at these problems.

**Depth prediction** The goal of depth prediction is the estimation of a depth map given only one or more RGB images. The field of *monocular depth prediction* uses only one image for its depth estimate. The first CNN for monocular depth prediction was presented in 2014 by Eigen *et al.* [18]. This network was trained on images labeled with true depth maps. Because these maps are hard to obtain, Garg *et al.* [3] developed the first network that used unsupervised learning. Training is performed by predicting the *other* image from a stereo pair; it is no longer necessary to collect true depth maps. Godard *et al.* [21] proposed further improvements to this technique. The recently published *PyD-Net* (July 2018) can run at $\sim 2$ Hz on a Raspberry Pi 3 CPU and still produce competitive results [45].

While methods [3, 21] do not require true depth labels, they still need to use a stereo camera to collect training data for monocular vision. As a result, these methods cannot be used for on-board training of monocular vision. An alternative to these approaches is to train on monocular image sequences. Examples of this approach are [46–48].

While it may seem redundant at first, it is also possible to perform depth prediction on stereo images. The advantage of this over 'normal' stereo vision methods such as SGM is that the neural network can also learn to include appearance cues. These provide additional depth information that is not provided by just the disparities. An example of deep learning for stereo vision is found in [49], where Self-Supervised Learning (SSL) is used to learn stereo vision from scratch. After training, the network can compete with existing state-of-the-art algorithms.

Compared to monocular vision, stereo vision has the advantage that a reliable reference distance is available: the baseline between the two cameras. As a result, depth estimates from stereo vision are more accurate than those from monocular vision. This point is strongly argued by Smolyanskiy *et al.*, who state that any application that relies on accurate depth estimates and that can carry more than one camera should do so [20]. The use of a stereo camera should be possible on all UAVs as even the $\sim 20$ g DelFly can carry a small stereo camera. The only reason the preliminary work in section 2.3 still looks at monocular vision is that this allows appearance cues to be examined in isolation from disparity or flow cues.

**Depth completion** Where depth prediction uses only RGB images, *depth completion* assumes that some sparse depth information is available. This information can come, for instance, from the depth of VO keypoints. In literature, LIDAR is also commonly mentioned as a source of sparse depth measurements.

Ma and Karaman [50] implement a network that uses sparse depth information and then compare its performance to monocular depth estimation. They come to the interesting conclusion that even a depth map generated from only 20 sparse depth measurements *without RGB images* already has a higher accuracy than the monocular depth estimation networks of [19, 51]. Note that this comparison is based on scale-aware metrics; the scale-invariant error [18] is not reported so it is not possible to say whether the relative distances are incorrect or that the monocular methods only suffer from a scaling error. Nevertheless, the experiments show that sparse depth measurements can be a valuable addition for depth estimation. The authors also check whether the use of RGB images in addition to sparse depth estimates leads to further accuracy improvements: this is primarily the case for low numbers of depth measurements, at higher numbers there is also an increase in accuracy but it is small. The work of Ma and Karaman is continued in [52]; other recent examples of depth completion are [53–55]. No examples were found where sparse depth completion is combined with or compared to stereo vision.

The good results from depth completion lead to an interesting design choice: is it better to perform depth prediction and use the results for VO, or to use VO to collect sparse measurements and use these to estimate a depth map? A third option has also appeared in recent literature: use a single network to predict both depth and pose from image sequences.

**Combined tasks: depth, pose, flow, segmentation, ...** Recently a growing number of articles is appearing on networks that combine depth estimation with other tasks. Common combinations are depth with pose, segmentation and/or optical flow. There are a few potential advantages to combining these techniques in a single network: if filters can be shared between tasks, this might lead to a lower total number of parameters. Secondly, combining multiple tasks can potentially improve learning as the depth estimation is encouraged to use the (intermediate) results of, for instance, object segmentation and vice versa. A review of these networks is left for future work; it is therefore not possible to confirm these advantages in this chapter.

### Training

Training is an essential component of Deep Learning. For depth estimation, two types of training are common in literature: supervised and unsupervised (also called self-supervised). Earlier examples of monocular depth estimation (e.g. [18]) rely on *supervised learning*. The network is trained to replicate a true depth map that belongs to the input image. This depth map is typically obtained using a LIDAR sensor or an RGB-D sensor (e.g. Microsoft's Kinect). An advantage of supervised learning is that in most cases a true depth value is available for every pixel. The major disadvantage, however, is that it requires an additional sensor to capture the true depth of the scene. For this reason, supervised learning cannot be used on-board a UAV; all training has to be performed offline.

In *unsupervised learning*, the true depth map is not available. Instead, unsupervised learning often depends on a reconstruction error, where for instance the other images in a stereo pair are predicted and compared to the true images (e.g. [3]). The advantage of unsupervised learning is that the training data is easier to collect. Since no additional sensor is required, learning can also be performed online, allowing the UAV to adapt to its environment during operation. 'Unsupervised learning' is a bit of a misnomer as the methods primarily rely on supervised training methods. The argument to call them unsupervised is that no labeled data has to be provided from an external source. Unsupervised learning is the same as *Self-Supervised Learning (SSL)*, but this term does not introduce ambiguity about the learning method, while it still makes it sufficiently clear that the supervision is already provided by the input data. Therefore, only 'SSL' will be used in this chapter.

Recent articles have started to use Generative Adversarial Networks (GANs) for depth perception (e.g. [56, 57]). In the GAN framework, a second network (the *discriminator*) is trained to distinguish the network's output from the training label. The depth perception network (the *generator*) and discriminator are trained in alternation. In this framework the discriminator essentially replaces the loss function, but unlike the loss function it is trained specifically for the (last version of) the generator network and can therefore provide a more precise measure of its performance.

GANs can be used in both supervised and self-supervised settings. In the former ([57]), it compares the generated and true depth maps; in the latter ([56]) it compares reconstructed and true images. In both papers the accuracy exceeds that of common benchmark papers.

The (off-line) training of a neural network requires an appropriate dataset. The use of publicly available dataset also allows a quantitative comparison between methods. Commonly used datasets are the KITTI stereo dataset[8] and the NYUv2 dataset[9] [58]. The KITTI dataset is aimed at automotive applications; the images are obtained from a stereo camera and LIDAR mounted on the front of a car. The NYUv2 dataset contains RGB-D images captured in indoor environments. Other frequently-used datasets are Make3D[10] [15, 16, 59] and the Cityscapes dataset[11] [60].

Instead of using data captured in the real world, it is also possible to generate these from a simulation. Examples of generated datasets are vKITTI[12] [61] and Synthia[13]. Training data can also be generated during closed-loop simulation. An example of this is Microsoft's AirSim[14] for UAVs and autonomous cars. An advantage of simulation is that the actual depth of all pixels is directly available. The disadvantage is that the generated images differ from those captured in the real world – the *reality gap*. In [62] Zheng *et al.* propose to use a GAN to reduce the difference between real and simulated images. The resulting network can outperform [18] but not [3, 21] when subsequently evaluated on real datasets.

#### Analysis of trained networks

While there are many articles on deep learning for depth perception, no articles were found on *how* the trained networks perform this task. There is a small number of articles that focus on the analysis of CNNs in general. In [63] Zeiler and Fergus use unpooling and deconvolution operations to map neuron activities back to the input space. Given an input image, this technique produces an image that highlights the regions that cause a strong activation of a selected neuron. This technique focuses on single neurons, but note that [64] argues that it is the space spanned by multiple neurons can be more informative than individual neuron activations. In a more recent paper Olah *et al.* [65] present a highly detailed (interactive) overview of visualization techniques. This article provides a good starting point for further research into neural network visualization.

The cited papers examine generic CNNs at a rather low level. No articles were found that examine the high-level behavior of networks for depth perception. How exactly do these networks estimate depth? This information is essential in order to predict the behavior of these networks on other platforms – UAVs in this case. Therefore, section 2.3 presents the first steps towards a high-level understanding of these networks.

## 2.3. Preliminary results

### 2.3.1. Monocular depth perception

The goal of this research is to use SSL to improve obstacle avoidance on UAVs. SSL will be used for depth estimation as the need to use vision to sense the environment sets

[8] http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo
[9] https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html
[10] http://make3d.cs.cornell.edu/data.html
[11] https://www.cityscapes-dataset.com/
[12] http://www.europe.naverlabs.com/Research/Computer-Vision/Proxy-Virtual-Worlds
[13] http://synthia-dataset.net/
[14] https://github.com/Microsoft/AirSim

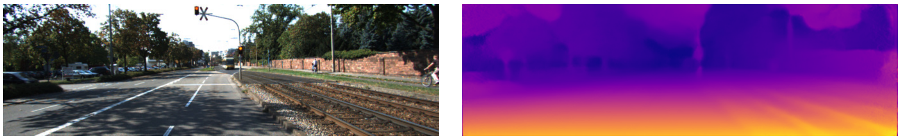


Figure 2.10: Monocular depth estimation with *MonoDepth* [21]. Input image: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

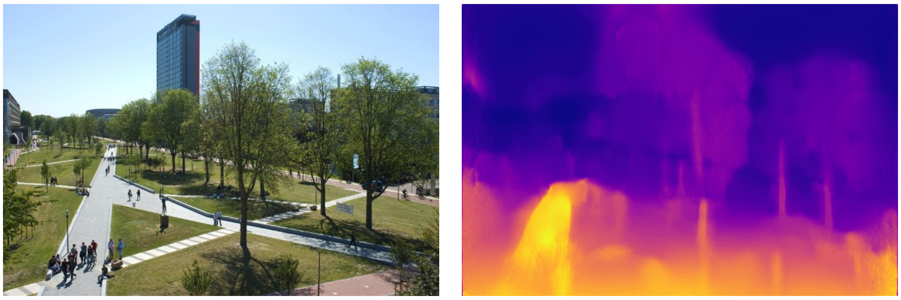


Figure 2.11: The MonoDepth network does not transfer well to different viewpoints. The road on the left is seen as a nearby obstacle. The high-rise building of EWI appears closer than the trees in front of it. Input image: *Aanzicht Mekelpark met studenten* by D. Brinkman, © Delft University of Technology, https://repository.tudelft.nl/view/MMP/uuid:cb952de1-34e8-4db3-a577-9177a62581ed?fullscreen=1.

UAVs apart from other vehicles. The first question to be asked is whether SSL-based depth estimation can actually be used on a UAV. At first sight this may seem obvious: why would it not work on a UAV? However, results indicate that this might not be as simple as it appears.

This section presents experiments performed on the *MonoDepth* network [21]. MonoDepth is a Self-Supervised monocular depth estimation network that is trained on the KITTI stereo vision dataset. The network predicts disparities such that these minimize a reconstruction error between two images of a stereo pair. On images in the KITTI dataset the network performs quite well (Figure 2.10). However, when the network is used on images taken from a different viewpoint (Figure 2.11), the accuracy of the depth map quickly degrades.

Clearly, a network trained on a dataset of automotive images cannot be transferred directly to a UAV. Most likely it is possible to get MonoDepth to work on a UAV by training it on a suitable dataset. However, that does not explain why the network trained on KITTI fails. The results on KITTI images show that the network *can* estimate depth, but apparently it does so using image features that do not work on UAVs. To guarantee correct behavior it is important to know what these features are and under what circumstances they are learned.

While there is a large and increasing number of articles on monocular depth perception, there is not a single paper that analyses what these networks actually learn. This experiment is a first step towards an *understanding* of monocular depth perception as learned by neural networks. The goals of this experiment are:

- Provide insight into monocular vision. While useful for UAVs, this insight will also be extremely valuable for automotive applications. With an understanding of the inner workings of monocular depth perception, it becomes easier to predict its behavior and to make guarantees about its correctness.

- Provide insight into the use of monocular vision on UAVs. The results should explain why the network trained on KITTI does not transfer well. The same experiments can then be performed on a network trained on a UAV dataset, and the differences can be compared.

- The features learned by MonoDepth might be replicated using simpler, lightweight algorithms. This would enable the use of monocular vision on embedded hardware and tiny UAVs, perhaps even the DelFly.

The problem with neural networks like MonoDepth is that they are black boxes. It is difficult to analyze what is going on inside them. In that respect there is some overlap with human depth perception, which is also difficult to take apart piece-by-piece. For neural networks there are techniques to visualize the functions of individual neurons, but this is a very low-level form of analysis. Instead, the experiments presented here will analyze the behavior of the entire network using the time-tested scientific method: coming up with *hypotheses* on how the network could estimate depth, then designing experiments that *test* whether these hypotheses are true.

So what would be a good hypothesis on the inner workings of MonoDepth? The experiments performed in this section test the following points:

- MonoDepth assumes there is a flat ground in front of the vehicle.

- MonoDepth uses color to detect the sky.

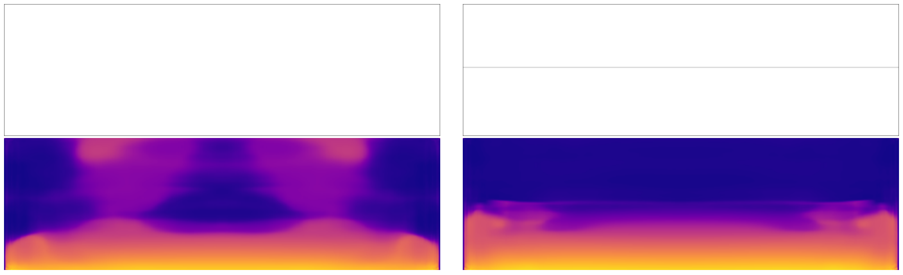
- MonoDepth estimates the distance towards obstacles using one or more of the following features:

  - Apparent size of known obstacles.
  - Vertical position in the image.

MonoDepth is trained on the KITTI dataset, which contains images taken from a forward-facing camera on a car. The camera has a fixed attitude and height and in the majority of the images there is a free section of road in front of the car. Rather than detecting the road, MonoDepth can just assume it is there as this will be true for nearly all images in the training set. It is hypothesized that MonoDepth assumes the presence of a flat ground rather than detecting it. The ground's depth estimate will be 'overwritten' by any obstacles it detects.

The upper half of the image consists of sky and obstacles. The sky seems easy to detect using its color or brightness and would result in a large number of correct pixels; it is therefore assumed that MonoDepth does just that, possibly combined with a prior expectation of seeing the sky in the upper half of the image.

With the ground and sky detected, the rest of the depth map consists of obstacles. Following the list of appearance cues in section 3.5, likely candidates for depth estimation towards obstacles are their apparent size and vertical position as both seem relatively easy



(a) MonoDepth's response to a completely white input image. Although there is a hint of a ground surface, the depth map (especially the top half) contains a lot of garbage.

(b) Addition of a single thin line is enough to make MonoDepth detect a floor and sky, or at least assume they are there.

Figure 2.12: MonoDepth has strong prior expectations about the presence of a flat ground in the lower half of the image and sky in the upper half. (Outlines added for visibility, these are not part of the input images.)

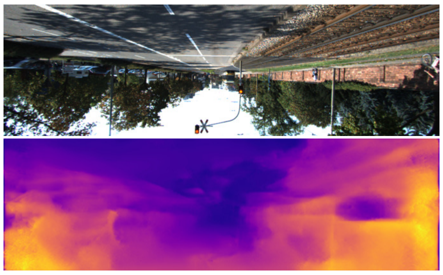


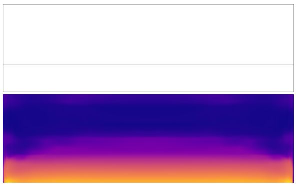
Figure 2.13: When the image is flipped vertically, the trees and sky in the lower half of the image are assumed to be closer than the road in the upper half. If MonoDepth did not have prior expectations about a flat ground, the disparity map would also have flipped vertically. Input image: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

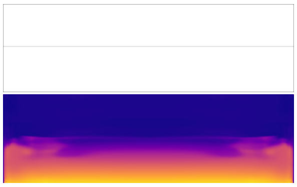
to measure, although the former also requires knowledge about the true size of various types of obstacles.

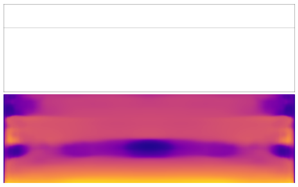
If MonoDepth indeed assumes the presence of a flat ground, it should be possible to make it 'see' a ground surface even if it isn't actually there. This is attempted in Figure 2.12. First, MonoDepth is presented with a completely blank input image to see if it will guess the presence of ground and sky. This is not entirely the case. However, when a thin horizon line is added to the image, MonoDepth is suddenly able to detect a floor and sky. Since the input image contains no features other than a single line, these can only come from MonoDepth's prior expectations.

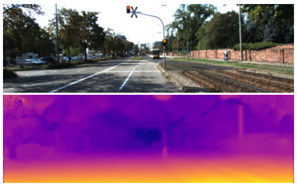
This prior expectation can also be demonstrated on real photographs, as shown in Figure 2.13. In this figure, a vertically flipped image from the KITTI dataset is presented to MonoDepth. If MonoDepth would not assign a high prior probability to the presence and location of the ground and sky, the resulting depth map would also be a flipped version of the original depth map. This is, however, not the case. Instead, the resulting depth map still assumes that the lower half of the image is closer than the upper half, confirming the presence of this bias in the estimation.

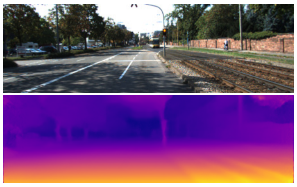
It appears that MonoDepth indeed assumes the presence of a flat ground in the lower


(a) Lower horizon line.


(b) The original horizon line, aligned with the horizon in the KITTI images.


(c) Higher horizon line.


(d) Lower horizon line.


(e) Original horizon line.

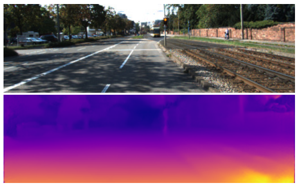
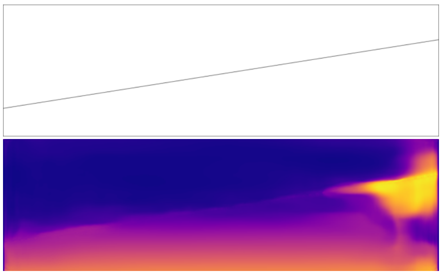

(f) Higher horizon line.

Figure 2.14: *a-c*: The vertical position of the horizon line does not change the ground plane. While it has some influence on the depth map, the ground plane still seems to end at the same height in the image. *d-f*: In real images, the extent of the ground plane matches the position of the horizon. This suggests that MonoDepth has a mechanism to detect the horizon that is not triggered in images *a-c*; it is not yet understood how this works. Input images d-f: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

half of the image, but does it assume a fixed depth map for the ground or does it detect the horizon in the input image? The latter would allow the network to correct changes in the pitch of the camera. To test this, the network is first presented with artificial images based on Figure 2.12, but with the horizon line shifted to different positions. The result is shown in Figure 2.14 a-c. In these artificial images it seems that MonoDepth does not use the position of the line for the ground plane, although it does have some influence on the depth map. This is also tested on real images by cropping different regions from one of the KITTI images, see Figure 2.14 d-f. The result is different; now the ground plane in the depth maps ends at the horizon in the input image. Apparently MonoDepth does estimate the position of the horizon. At the time of writing it is not yet clear how this works. Note that MonoDepth's correction to the pitch is not perfect: the depth towards obstacles appears to have changed, especially in Figure 2.14 d.

Does MonoDepth also detect roll angles? This is tested by rotating the input images, the result is shown in Figure 2.15. It appears that MonoDepth does not detect the roll angle of the camera: the ground surface still appears flat. Also note the tree trunks that appear vertical in the depth map even though they are clearly tilted in the input image. This seems another example of MonoDepth assuming the presence of certain features rather than actually observing them.

The second hypothesis is that MonoDepth detects the sky using color segmentation. While this sounds plausible, Figure 2.10 already hints that this is not entirely true: in the depth map the sky appears closer than the trees that occlude it. Figure 2.13 also shows that there is a prior expectation of the position of the sky in the upper half of the image; the sky color in the bottom half of the image does not result in an infinite depth (although its


(a) When the artificial horizon line is tilted, the ground surface remains flat in the depth map. There are some artefacts where the line is above the assumed horizon.

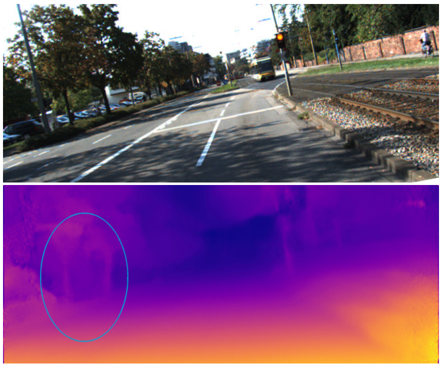
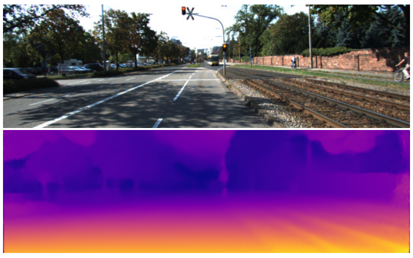
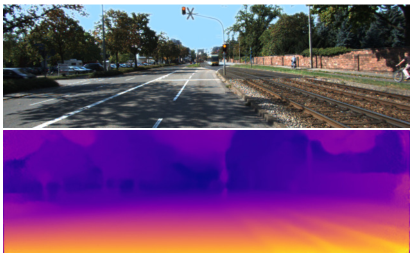
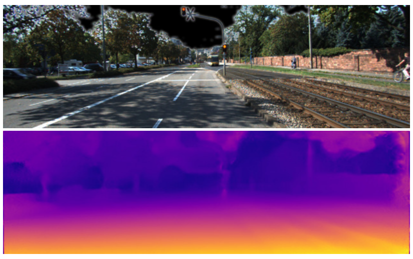

(b) Even though the input image is tilted, the road surface appears more-or-less flat in the disparity map. Also notice the trees that are vertical in the disparity map but clearly tilted in the input image.

Figure 2.15: Both in artificial and real images MonoDepth does not appear to detect roll motions and will still assume a flat road surface and vertically oriented obstacles. Input image b: KITTI stereo vision dataset (CC BY-NC-SA 3.0)


(a) Original image.


(b) Blue sky. No difference in the depth map.


(c) Black sky. The sky appears further away, there are some artifacts around the traffic light.

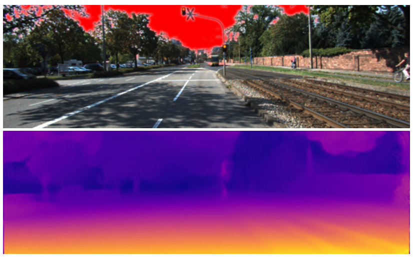

(d) Red sky. The sky appears further away.

Figure 2.16: Sensitivity to sky color. There is no perceivable difference between the depth maps for white and blue sky, colors that naturally occur in the training dataset. There is some difference when the sky is made black or red, but the effect remains relatively small. Input image: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

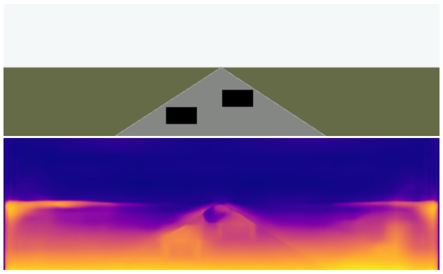


Figure 2.17: MonoDepth's depth perception can easily be triggered using simple visual features. Notice how the right rectangle appears to be further away than the left, even though they are the same size.

observed depth is further than that of the trees).

In Figure 2.16 the sky is replaced with different colors. The figure shows that unnatural colors have some effect on the depth estimate, but do not cause large disturbances such as objects appearing at close distance. These results show that MonoDepth does not detect the sky using (only) color segmentation. What mechanism it uses instead remains to be found in further research.

The final hypothesis on the workings of MonoDepth concerns the depth estimation of obstacles. Two options appear likely: the obstacle's size is used, or its vertical position in the image.

Figure 2.17 provides a first clue. The image shows that MonoDepth's perception of obstacles is easy to trigger: black rectangles are already shown as obstacles in the depth map. More importantly, however, is how it determined the distance towards the rectangles: the rectangles are the same size in the image but at different vertical positions. They are placed at different depths, which suggests that the vertical position had a strong influence on this estimate.

Further evidence towards this conclusion is presented in Figure 2.18. In this figure, three scenarios are presented to the network: the real-life scenario in which one car is smaller and higher in the image, one where the car is only smaller, and one where it is only placed at a higher position. In all cases, the vertical position of the car appears to control the depth estimate.

A final clue can be found back in Figure 2.14 d where the camera pitch was changed. When the camera is pitched up, the obstacles move downwards in the image. In the resulting depth map, the obstacles appear closer.

There are a few possible reasons why the vertical position is used as the main cue for depth. First of all, it might be easier for a CNN to measure the vertical position than the scale of obstacles because the convolution operation is translation-invariant but not scale invariant. Secondly, since the camera is fixed at a nearly constant height and attitude, distance estimates based on the vertical image position may be more accurate than estimates based on the scale as the latter depend on the real-world dimensions of the obstacle which can contain large variations.

What do these results mean for the use of MonoDepth on a UAV? The strong assumption of a flat ground in front of the camera is not compatible with the large pitch and

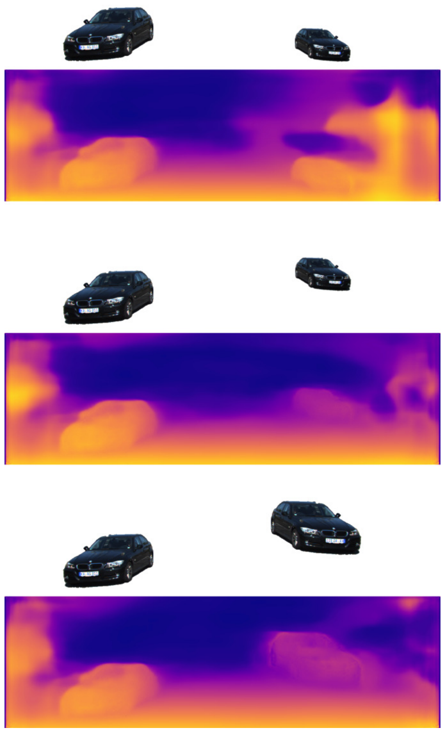


Figure 2.18: MonoDepth's depth estimation depends on the vertical position of objects in the image, not their apparent size. *Top*: the cars are assigned the same disparity, even though the car on the right has a smaller apparent size. *Middle*: the right car is smaller and positioned higher in the image (as would be the case in real images), it is correctly estimated to be further away. *Bottom*: the car on the right has the same apparent size as the one on the left. Still, it is indicated to be further away as its vertical position in the image is higher. Cars: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

roll angles expected on a UAV. The same holds for the use of vertical image position to estimate the depth towards obstacles: this assumes a constant height above the ground and a constant pitch angle. In conclusion, MonoDepth trained on the KITTI dataset can not be directly transferred to a UAV. Training on a suitable dataset for UAVs should reveal whether these problems come from the use of the KITTI dataset for training or whether they are more fundamental.

*The ideas in this section were developed together with Guido de Croon. In 2019 we published a conference paper on this topic at the International Conference on Computer Vision:*[15]

> Tom van Dijk and Guido de Croon. How Do Neural Networks See Depth in Single Images? In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. http://openaccess.thecvf.com/content_ICCV_2019/html/van_Dijk_How_Do_Neural_Networks_See_Depth_in_Single_Images_ICCV_2019_paper.html

### 2.3.2. Flight tests

Flight tests were performed to get more practical experience with visual obstacle avoidance. The system uses a combination of stereo vision and visual odometry for obstacle avoidance, even in GPS-less environments. These flight tests were performed as part of the *Percevite* project (www.percevite.org).

#### Stop-before-obstacle with OptiTrack

The first task towards obstacle avoidance was the implementation of visual obstacle detection on the Parrot SLAMDunk. This work started with a review on stereo vision and optical flow (subsection 2.2.1). The review showed that SGBM [2] still belongs to the best-performing algorithms. An implementation of SGBM was already present on the SLAMDunk and performed better than the OpenCV implementation due to its use of the GPU.

The depth map obtained from SGBM is then used as follows: a Region of Interest (ROI) is cropped from the center of the image where the view is not occluded by rotors or other parts of the drone. Of this region, the 5th percentile of depth values is calculated. The use of the 5th percentile instead of the minimum value adds some robustness to noise, at the cost of missing tiny obstacles (although this has not been a problem in practice). The 5th-percentile distance is sent to the autopilot as the distance to the nearest obstacle in front of the UAV. Additionally, the number of valid pixels (pixels for which a disparity can be found without ambiguity) is sent to the autopilot; if this value is too low the UAV is not allowed to move forward.

The movement logic is implemented as a Paparazzi[16] autopilot module (but general enough to be ported to other autopilots). The movement of the UAV is controlled using a waypoint; this waypoint can only be moved within the region that is observed to be free of obstacles with a sufficient safety margin. Since the waypoint is always in a safe region, this should prevent collisions when used in a static environment, provided that the drone

[15]Chapter 3
[16]http://wiki.paparazziuav.org/wiki/Main_Page

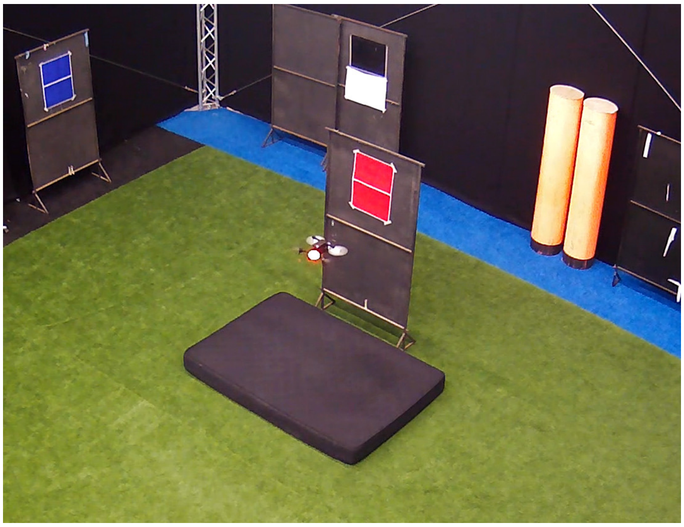


Figure 2.19: Parrot Bebop 2 with SLAMDunk stopping in front of an obstacle using SGBM and OptiTrack. (March 2018.)

can maintain its position without drift. At this stage, the OptiTrack system inside the TU Delft *Cyberzoo* was used for position feedback, so there was no drift present.

The full system was successfully demonstrated in March 2018 in the *Cyberzoo* (Figure 2.19). The code developed for the SLAMDunk/ROS is published at https://github.com/tomvand/percevite_slamdunk, the Paparazzi module is published at https://github.com/tomvand/paparazzi/tree/percevite.

### Implementation of 'embedded Visual Odometry'

While the system of March 2018 worked, its dependency on OptiTrack was a strong limitation. Work on VO started with a review of existing methods (Table 2.2.1). Out of the reviewed methods, the following were evaluated on the SLAMDunk: ORB-SLAM2[17] [31], OKVIS[18] [33] and SVO2[19] [30]. However, all of these packages had performance issues either in terms of run-time (ORB-SLAM2, OKVIS) or drift (SVO2). The optical flow algorithm of the Paparazzi autopilot was also evaluated but was found to produce poor velocity estimates or cause segmentation faults, leading to a crash of the autopilot.

Since none of the readily available packages was suitable for use on the SLAMDunk, there was no other choice but to implement a lightweight alternative. Encouraged by the results of [26], the *embedded Visual Odometer (eVO)* algorithm by Sanfourche *et al.* [29] was selected. The algorithm is a relatively straightforward implementation of VO using the Perspective-3-Point (P3P) algorithm. P3P is used to compute the UAV's pose relative to a single keyframe of 3D points; these 3D points are obtained using the depth map of SGBM. The run-time of eVO is reduced by using a fixed position for the points in the keyframe;

[17]https://github.com/raulmur/ORB_SLAM2
[18]https://github.com/ethz-asl/okvis_ros
[19]http://rpg.ifi.uzh.ch/svo2.html

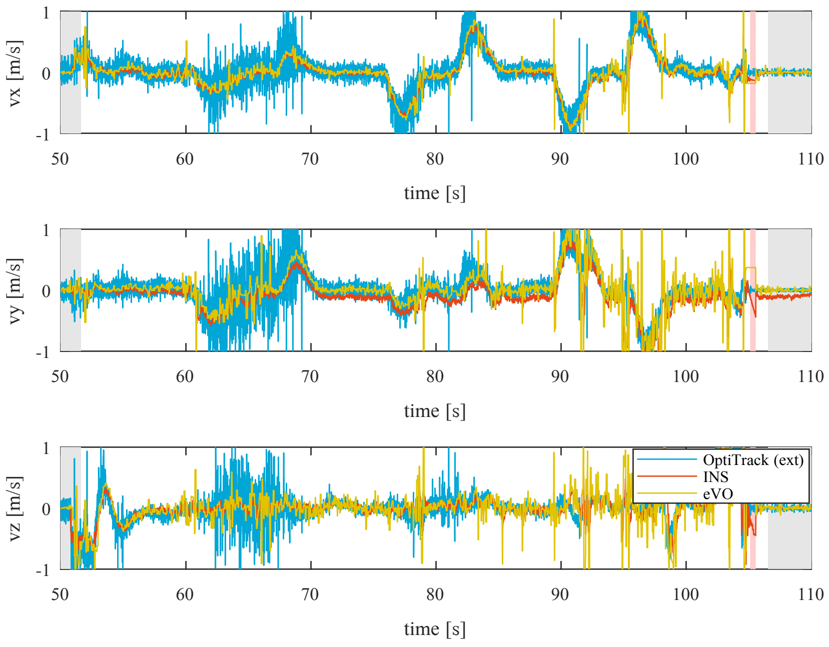


Figure 2.20: Body-frame velocities estimated using embedded Visual Odometer (eVO) compared to the ground-truth from OptiTrack. The figure shows both the raw estimates (eVO) and the velocity estimates after fusion with the accelerometer readings (INS). In general the estimate of eVO corresponds well to the OptiTrack measurements and appears to be less noisy (this may depend on the OptiTrack calibration and lighting conditions in the *Cyberzoo*). The INS estimates have a slight bias (especially in the y axis) that comes from the accelerometer. Possibly the SLAMDunk is not placed exactly above the Center-of-Gravity (CoG) of the drone.

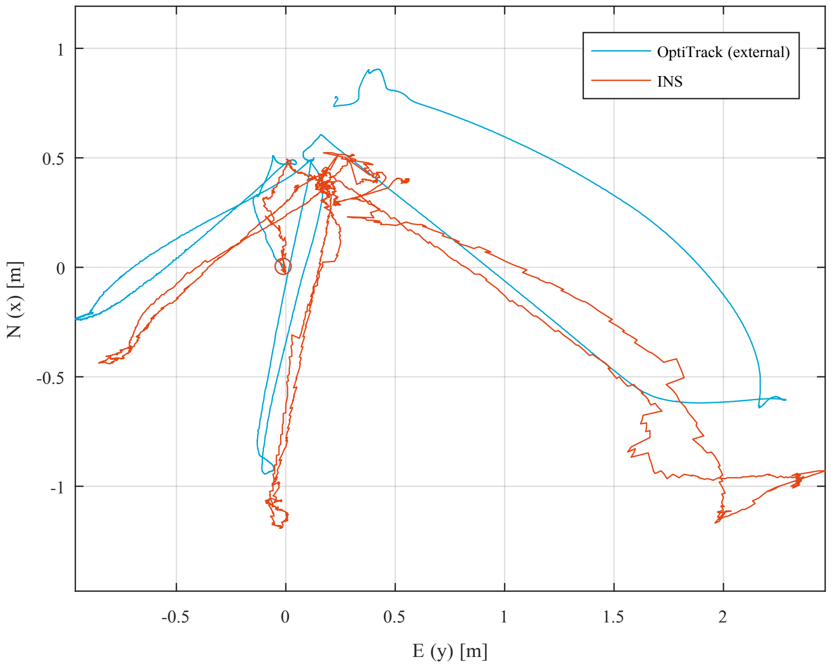


Figure 2.21: UAV trajectory estimated using eVO as input to the INS, compared to the ground truth captured using the OptiTrack system. The starting position of the UAV is indicated by the circle at $(0, 0)$. The final position error was 50 cm after a flight of 60 s.

these are not further refined when new measurements arrive. Secondly, the gyroscope is used to predict the next positions of the keypoints, thereby lowering the search region for the optical flow algorithm. My implementation of eVO is published at `https://github.com/tomvand/openevo` (and `https://github.com/tomvand/openevo-ros` for the ROS wrapper). This implementation does not reach the run-times reported in [29], but still runs at a rate of $\sim 5-10$ Hz on the SLAMDunk.

The velocity estimates from eVO are sent to Paparazzi, where the horizontal EKF combines the velocities with accelerometer measurements to get a final estimate of the UAV's velocity and position. The world position and angular rates – also estimated by eVO – are currently unused. The eVO algorithm was tested inside the *Cyberzoo* and found to be surprisingly robust: it was able to follow the drone's trajectory even when no textured panels were placed around the edge of the *Cyberzoo*. Test flight results are shown in Figure 2.20 and 2.21. The drone could successfully navigate between waypoints and stop in front of obstacles (Figure 2.22).

### Outdoor test flights

Outdoor test flights allowed eVO and SGBM to be tested in real outdoor environments with natural obstacles and lighting and under windy conditions. The velocity estimate

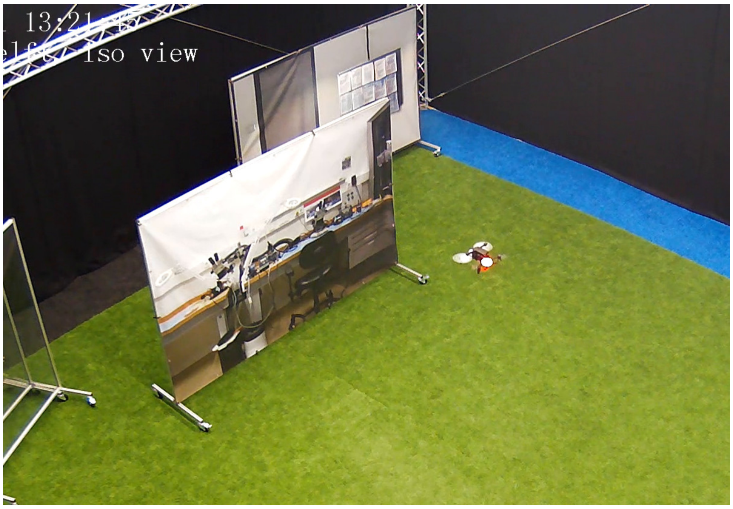


Figure 2.22: Stop-before-obstacle using eVO. (July 2018.)

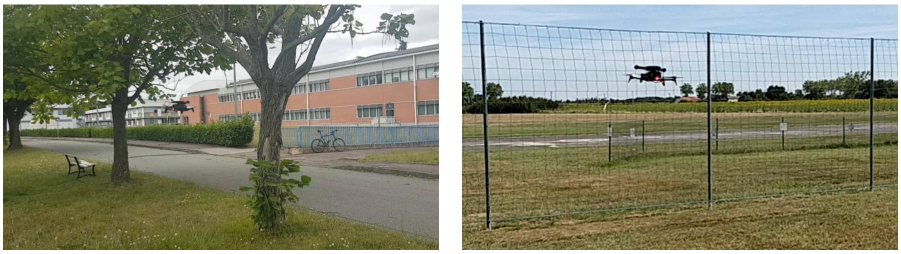


Figure 2.23: Outdoor stop-before-obstacle tests at ENAC. (August 2018.)

of eVO was found accurate enough for the UAV to maintain its position under windy conditions. SGBM was able to reliably detect obstacles, although it produced a few false positive detections when facing the sun. Tests of the full obstacle avoidance system were performed by sending the drone towards obstacles. In most of the cases the drone stopped successfully in front of the obstacle (Figure 2.23).

A second goal of the outdoor flights was to add GPS to the position estimate of the drone. The use of GPS allows the UAV to follow pre-programmed trajectories and is more in line with the target applications. Fusion of GPS with eVO velocity estimates was successfully demonstrated during short test flights. Longer GPS trajectories were however not tested because of battery limitations.

During the outdoor tests the ground-truth position of the drone could not be recorded, most test flights are instead recorded as videos of the UAV combined with logs of its internal estimates. The outdoor tests were very successful; during the week the UAV only crashed once, possibly as the result of human error instead of an algorithm failure.

*The collision avoidance system described here was used in the International Micro Air Vehicle (IMAV) competition 2018, where the ENAC/TU DELFT Team Paparazzi reached first place in the outdoor competition.* http://www.imavs.org/imav-2018-awards/

## References

[1] B. M. Albaker and N. A. Rahim, *A Survey of Collision Avoidance Approaches for Unmanned Aerial Vehicles,* in *Technical Postgraduates (TECHPOS), 2009 International Conference for* (2009).

[2] H. Hirschmüller, *Stereo Processing by Semiglobal Matching and Mutual Information* ¨, IEEE Transactions on Pattern Analysis and Machine Intelligence **30**, 328 (2008).

[3] R. Garg, V. B. Kumar, G. Carneiro, and I. Reid, *Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue,* in *European Conference on Computer Vision*, edited by B. Leibe, J. Matas, N. Sebe, and M. Welling (Springer International Publishing, Cham, 2016) pp. 740–756.

[4] H. Pham, S. A. Smolka, S. D. Stoller, D. Phan, and J. Yang, *A survey on unmanned aerial vehicle collision avoidance systems,* arXiv preprint (2015), arXiv:1508.07723 .

[5] D. Scharstein and R. Szeliski, *A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms,* International Journal of Computer Vision **47**, 7 (2002).

[6] M. Menze and A. Geiger, *Object scene flow for autonomous vehicles,* Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition **07-12-June**, 3061 (2015), arXiv:1512.02134 .

[7] B. Tippetts, D. J. Lee, K. Lillywhite, and J. Archibald, *Review of stereo vision algorithms and their suitability for resource-limited systems,* Journal of Real-Time Image Processing **11**, 5 (2016).

[8] A. Geiger, M. Roser, and R. Urtasun, *Efficient Large-Scale Stereo Matching,* in *Computer Vision – ACCV 2010* (2011) pp. 25–38.

[9] P. Pinggera, D. Pfeiffer, U. Franke, and R. Mester, *Know Your Limits: Accuracy of Long Range Stereoscopic Object Measurements in Practice,* in *European Conference on Computer Vision* (Springer, 2014) pp. 96–111.

[10] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, *A database and evaluation methodology for optical flow,* International Journal of Computer Vision **92**, 1 (2011), data CC BY-NC 2.0, arXiv:1412.0767 .

[11] A. Ranjan and M. J. Black, *Optical Flow Estimation using a Spatial Pyramid Network,* in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2017) arXiv:1611.00850 .

[12] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, *High Accuracy Optical Flow Estimation Based on a Theory for Warping,* Computer Vision - ECCV 2004 **3024**, 25 (2004), arXiv:0703101v1 [cs] .

[13] A. Geiger, P. Lenz, and R. Urtasun, *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,* Computer Vision and Pattern Recognition , 3354 (2012), data CC BY-NC-SA 3.0.

[14] J. J. Gibson, *The perception of the visual world* (Houghton Mifflin, Oxford, England, 1950).

[15] A. Saxena, S. H. Chung, and A. Y. Ng, *Learning Depth from Single Monocular Images,* Advances in Neural Information Processing Systems **18**, 1161 (2006).

[16] A. Saxena, Min Sun, and A. Ng, *Make3D: Learning 3D Scene Structure from a Single Still Image,* IEEE Transactions on Pattern Analysis and Machine Intelligence **31**, 824 (2009).

[17] K. van Hecke, G. C. de Croon, D. Hennes, T. P. Setterfield, A. Saenz-Otero, and D. Izzo, *Self-supervised learning as an enabling technology for future space exploration robots: Iss experiments on monocular distance learning,* Acta Astronautica **140**, 1 (2017).

[18] D. Eigen, C. Puhrsch, and R. Fergus, *Depth Map Prediction from a Single Image using a Multi-Scale Deep Network,* in *Advances in Neural Information Processing Systems 27* (Curran Associates, Inc., 2014) pp. 2366–2374.

[19] D. Eigen and R. Fergus, *Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture,* in *Proceedings of the IEEE International Conference on Computer Vision* (2015) pp. 2650–2658.

[20] N. Smolyanskiy, A. Kamenev, and S. Birchfield, *On the Importance of Stereo for Accurate Depth Estimation: An Efficient Semi-Supervised Deep Neural Network Approach,* arXiv preprint arXiv:1803.09719 (2018), arXiv:1803.09719 .

[21] C. Godard, O. Mac Aodha, and G. J. Brostow, *Unsupervised Monocular Depth Estimation with Left-Right Consistency,* The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017).

[22] C. Goerzen, Z. Kong, and B. Mettler, *Journal of Intelligent and Robotic Systems: Theory and Applications*, Vol. 57 (2010) pp. 65–100.

[23] J. Minguez, F. Lamiraux, and J.-P. Laumond, *Motion Planning and Obstacle Avoidance,* Springer Handbook of Robotics , 1177 (2016).

[24] L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, *Stereo vision-based obstacle avoidance for micro air vehicles using disparity space,* in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 9836 (IEEE, 2014) pp. 3242–3249.

[25] K. Schmid, P. Lutz, T. Tomić, E. Mair, and H. Hirschmüller, *Autonomous Vision-based Micro Air Vehicle for Indoor and Outdoor Navigation,* Journal of Field Robotics **31**, 537 (2014).

[26] J. Marzat, S. Bertrand, and A. Eudes, *Reactive MPC for Autonomous MAV Navigation in Indoor Cluttered Environments : Flight Experiments,* (2017), 10.1016/j.ifacol.2017.08.1910.

[27] J. Engel, V. Koltun, and D. Cremers, *Direct Sparse Odometry,* IEEE Transactions on Pattern Analysis and Machine Intelligence (2017), 10.1109/TPAMI.2017.2658577.

[28] O. J. Woodman, *University of Cambridge*, Tech. Rep. (2007) arXiv:ISSN 1476-2986 .

[29] M. Sanfourche, V. Vittori, and G. Le Besnerais, *eVO: a realtime embedded stereo odometry for MAV applications,* in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on* (IEEE, 2013) pp. 2107–2114.

[30] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, *SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems,* IEEE Transactions on Robotics **33**, 249 (2017).

[31] R. Mur-Artal and J. Tardós, *ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras,* IEEE Transactions on Robotics **33**, 1255 (2017).

[32] V. Usenko, J. Engel, J. Stückler, and D. Cremers, *Direct Visual-Inertial Odometry with Stereo Cameras,* in *Robotics and Automation (ICRA), 2016 IEEE International Conference on* (IEEE, 2016) pp. 1885–1892.

[33] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, *Keyframe-based visual–inertial odometry using nonlinear optimization,* The International Journal of Robotics Research **34**, 314 (2015).

[34] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, *ORB-SLAM: A Versatile and Accurate Monocular SLAM System,* IEEE Transactions on Robotics **31**, 1147 (2015), arXiv:1502.00956 .

[35] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, *Robust Visual Inertial Odometry Using a Direct EKF-Based Approach,* in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (IEEE, 2015) pp. 298–304.

[36] C. Forster, M. Pizzoli, and D. Scaramuzza, *SVO: Fast semi-direct monocular visual odometry,* Proceedings - IEEE International Conference on Robotics and Automation , 15 (2014).

[37] J. Engel, T. Schops, and D. Cremers, *LSD-SLAM: Large-Scale Direct monocular SLAM,* Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **8690 LNCS**, 834 (2014).

[38] J. Engel, J. Stückler, and D. Cremers, *Large-Scale Direct SLAM with Stereo Cameras,* in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (IEEE, 2015) pp. 1935–1942.

[39] C. Kerl, J. Sturm, and D. Cremers, *Robust odometry estimation for RGB-D cameras,* in *2013 IEEE International Conference on Robotics and Automation* (IEEE, 2013) pp. 3748–3754.

[40] M. Li and A. I. Mourikis, *High-precision, consistent EKF-based visual-inertial odometry,* The International Journal of Robotics Research **32**, 690 (2013).

[41] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, *DTAM: Dense tracking and mapping in real-time,* in *2011 International Conference on Computer Vision* (IEEE, 2011) pp. 2320–2327.

[42] G. Klein and D. Murray, *Parallel tracking and mapping for small AR workspaces,* 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR (2007), 10.1109/ISMAR.2007.4538852, arXiv:arXiv:1407.5736v1 .

[43] C. Nous, R. Meertens, C. de Wagter, and G. de Croon, *Performance Evaluation in Obstacle Avoidance,* in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016) pp. 3614–3619.

[44] C. Sprunk, G. Parent, L. Spinello, G. D. Tipaldi, W. Burgard, and M. Jalobeanu, *An Experimental Protocol for Benchmarking Robotic Indoor Navigation,* in *Experimental Robotics* (Springer International Publishing, 2015) pp. 487–504.

[45] M. Poggi, F. Aleotti, F. Tosi, S. Mattoccia, and C. V. Jul, *Towards real-time unsupervised monocular depth estimation on CPU,* arXiv preprint arXiv:1806.11430 (2018), arXiv:arXiv:1806.11430v3 .

[46] S. Vijayanarasimhan, S. Ricco, C. Schmid, R. Sukthankar, and K. Fragkiadaki, *SfM-Net: Learning of Structure and Motion from Video,* CoRR **abs/1704.0** (2017), arXiv:1704.07804v1 .

[47] H. Jiang, E. Learned-Miller, G. Larsson, M. Maire, and G. Shakhnarovich, *Self-Supervised Relative Depth Learning for Urban Scene Understanding,* in *Proceedings of the European Conference on Computer Vision (ECCV)* (2018) pp. 19–35, arXiv:1712.04850 .

[48] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, *Unsupervised Learning of Depth and Ego-Motion from Video,* in *CVPR* (2017) p. 7.

[49] Y. Zhong, Y. Dai, and H. Li, *Self-Supervised Learning for Stereo Matching with Self-Improving Ability,* (2017), arXiv:1709.00930 .

[50] F. Ma and S. Karaman, *Sparse-to-Dense: Depth Prediction from Sparse Depth Samples and a Single Image,* arXiv preprint arXiv:1709.07492 (2017), arXiv:1709.07492 .

[51] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, *Deeper depth prediction with fully convolutional residual networks,* Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016 , 239 (2016), arXiv:1606.00373 .

[52] F. Ma, G. V. Cavalheiro, and S. Karaman, *Self-supervised Sparse-to-Dense: Self-supervised Depth Completion from LiDAR and Monocular Camera,* arXiv preprint arXiv:1807.00275 (2018), arXiv:1807.00275 .

[53] C. S. Weerasekera, T. Dharmasiri, R. Garg, T. Drummond, and I. Reid, *Just-in-Time Reconstruction: Inpainting Sparse Maps using Single View Depth Predictors as Priors,* arXiv preprint arXiv:1805.04239 (2018), arXiv:1805.04239 .

[54] M. Jaritz, R. de Charette, E. Wirbel, X. Perrotton, and F. Nashashibi, *Sparse and Dense Data with CNNs: Depth Completion and Semantic Segmentation,* arXiv preprint arXiv:1808.00769 (2018), arXiv:1808.00769 .

[55] X. Cheng, P. Wang, and R. Yang, *Depth Estimation via Affinity Learned with Convolutional Spatial Propagation Network,* arXiv preprint arXiv:1808.00150 (2018), arXiv:1808.00150 .

[56] A. Pilzer, D. Xu, M. M. Puscas, E. Ricci, and N. Sebe, *Unsupervised Adversarial Depth Estimation using Cycled Generative Networks,* arXiv preprint arXiv:1807.10915 (2018), arXiv:1807.10915 .

[57] R. Chen, F. Mahmood, A. Yuille, and N. J. Durr, *Rethinking Monocular Depth Estimation with Adversarial Training,* arXiv preprint arXiv:1808.07528 (2018), arXiv:arXiv:1808.07528v1 .

[58] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, *Indoor Segmentation and Support Inference from RGBD Images,* in *Computer Vision – ECCV 2012*, edited by A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 746–760.

[59] A. Saxena, S. H. Chung, and A. Y. Ng, *3-D Depth Reconstruction from a Single Still Image,* International Journal of Computer Visional of computer vision **76**, 53 (2007).

[60] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The Cityscapes Dataset for Semantic Urban Scene Understanding,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 3213–3223, arXiv:1604.01685 .

[61] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, *Virtual Worlds as Proxy for Multi-Object Tracking Analysis,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 4340–4349, arXiv:arXiv:1605.06457v1 .

[62] C. Zheng, T.-j. Cham, and J. Cai, *T²Net: Synthetic-to-Realistic Translation for Solving Single-Image Depth Estimation Tasks,* arXiv preprint arXiv:1808.01454 (2018), arXiv:arXiv:1808.01454v1 .

[63] M. D. Zeiler and R. Fergus, *Visualizing and Understanding Convolutional Networks,* in *Computer Vision – ECCV 2014*, edited by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars (Springer International Publishing, Cham, 2014) pp. 818–833.

[64] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks,* arXiv preprint arXiv:1312.6199 (2013), arXiv:1312.6199 .

[65] C. Olah, A. Mordvintsev, and L. Schubert, *Feature Visualization,* Distill (2017), 10.23915/distill.00007.

# 3
# Monocular depth perception

Deep neural networks have lead to a breakthrough in depth estimation from single images. Recent work shows that the quality of these estimations is rapidly increasing. It is clear that neural networks can see depth in single images. However, to the best of our knowledge, no work currently exists that analyzes what these networks have learned.

In this chapter we take four previously published networks and investigate what depth cues they exploit. We find that all networks ignore the apparent size of known obstacles in favor of their vertical position in the image. The use of the vertical position requires the camera pose to be known; however, we find that these networks only partially recognize changes in camera pitch and roll angles. Small changes in camera pitch are shown to disturb the estimated distance towards obstacles. The use of the vertical image position allows the networks to estimate depth towards arbitrary obstacles – even those not appearing in the training set – but may depend on features that are not universally present.

The contents of this chapter have been published as:

**van Dijk, T.**, de Croon, G. (2019). *How do neural networks see depth in single images?* Proceedings of the IEEE/CVF International Conference on Computer Vision.

## 3.1. Introduction

Stereo vision allows absolute depth to be estimated using multiple cameras. When only a single camera can be used, optical flow can provide a measure of depth; or if images can be combined over longer time spans then SLAM or Structure-from-Motion can be used to estimate the geometry of the scene. These methods tend to treat depth estimation as a purely geometrical problem, ignoring the *content* of the images.

When only a *single* image is available, it is not possible to use epipolar geometry. Instead, algorithms have to rely on *pictorial cues*: cues that indicate depth within a single image, such as texture gradients or the apparent size of known objects. Shape-from-X methods (e.g. [1, 2], [3], [4]) use some of these cues to infer shape, but often make strong assumptions that make them difficult to use in unstructured environments such as those seen in autonomous driving. Other cues such as the apparent size of objects may require knowledge about the environment that is difficult to program by hand. As a result, pictorial cues have seen relatively little use in these scenarios until recently.

With the arrival of stronger hardware and better machine-learning techniques – most notably Convolutional Neural Networks (CNN) – it is now possible to *learn* pictorial cues rather than program them by hand. One of the earliest examples of monocular depth estimation using machine learning was published in 2006 by Saxena *et al.* [5]. In 2014, Eigen *et al.* [6] were the first to use a CNN for monocular depth estimation. Where [6] still required a true depth map for training, in 2016 Garg *et al.* proposed a new scheme that allows the network to learn directly from stereo pairs instead [7]; this work was further improved upon by Godard *et al.* in [8]. In parallel, methods have been developed that use monocular image sequences to learn single-frame depth estimation in an unsupervised manner, of which the works by Zhou *et al.* [9] and Wang *et al.* [10] are examples. Recent work focuses primarily on the accuracy of monocular depth estimation, where evaluations on publicly available datasets such as KITTI [11] and NYUv2 [12] show that neural networks *can* indeed generate accurate depth maps from single images. However, to the best of our knowledge, no work exists that investigates *how* they do this.

Why is it important to know what these neural networks have learned? Firstly, it is difficult to guarantee correct behavior without knowing what the network does. Evaluation on a test set shows that it works correctly in those cases, but it does not guarantee correct behavior in other scenarios. Secondly, knowing what the network has learned provides insight into training. Additional guidelines for the training set and data augmentation may be derived from the learned behavior. Thirdly, it provides insight into transfer to other setups. With an understanding of the network, it is for instance easier to predict what the impact of a change in camera height will be and whether this will work out-of-the-box, require data augmentation or even a new training set.

In this chapter, we take four previously published neural networks (*MonoDepth* by Godard *et al.* [8], SfMLearner by Zhou *et al.* [9], Semodepth by Kuznietsov *et al.* [13] and LKVOLearner by Wang *et al.* [10]) and investigate their *high-level behavior*, where we focus on the distance estimation towards cars and other obstacles in an autonomous driving scenario. Section 3.2 gives an overview of related literature. In section 3.3 we show that the all of the networks rely on the vertical image position of obstacles but not on their apparent size. Using the vertical position requires knowledge of the camera pose; in section 3.4 we investigate whether the camera pose is assumed constant or observed from the images. For *MonoDepth* we investigate in section 3.5 how it recognizes obstacles and finds their ground contact point. We discuss the impact of our results in section 3.6.

## 3.2. Related work

Existing work on monocular depth estimation has extensively shown that neural networks can estimate depth from single images, but an analysis of how this estimation works is still missing. Feature visualization and attribution could be used to analyze this behavior. One of the earlier examples of feature visualization in deep networks can be found in [14]. The methods have been improved upon in e.g. [15, 16] and an extensive treatment of visualization techniques can be found in [17]. In essence, the features used by a neural network can be visualized by optimizing the input images with respect to a loss function based on the excitation of a single neuron, a feature map or an entire layer of the network. The concurrent work of Hu *et al*. [18] in which the authors perform an attribution analysis to find the pixels that contribute most to the resulting depth map is most closely related to our work. However, these methods only provide insight into the low-level workings of CNNs. A collection of features that the neural network is sensitive to is not a full explanation of its behavior. A link back to depth cues and behavior in more human terms is still missing, which makes it difficult to reason about these networks.

In this chapter, we take a different approach that is perhaps more closely related to the study of (monocular) depth perception in humans. We treat the neural network as a black box, only measuring the response (in this case depth maps) to certain inputs. Rather than optimizing the inputs with regards to a loss function, we modify or disturb the images and look for a correlation in the resulting depth maps.

Literature on human depth perception provides insight into the pictorial cues that could be used to estimate distance. The following cues from [19] and more recent reviews [20, 21] can typically be found in single images:

- Position in the image. Objects that are further away tend to be closer to the horizon. When resting on the ground, the objects also appear higher in the image.
- Occlusion. Objects that are closer occlude those that lie behind them. Occlusion provides information on depth order, but not distance.
- Texture density. Textured surfaces that are further away appear more fine-grained in the image.
- Linear perspective. Straight, parallel lines in the physical world appear to converge in the image.
- Apparent size of objects. Objects that are further away appear smaller.
- Shading and illumination. Surfaces appear brighter when their normal points towards a light source. Light is often assumed to come from above. Shading typically provides information on depth changes within a surface, rather than relative to other parts of the image.
- Focus blur. Objects that lie in front or behind the focal plane appear blurred.

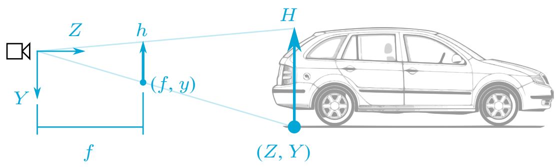


Figure 3.1: True object size $H$ and position $Y$, $Z$ in the camera frame and vertical image position $y$ and apparent size $h$ in image coordinates. Image coordinates are measured from the center of the image.

- Aerial perspective. Very far away objects (kilometers) have less contrast and take on a blueish tint.

Of these cues, we expect that only the *position in the image* and *apparent size of objects* are applicable to the KITTI dataset; other cues are unlikely to appear because of low image resolution (texture density, focus blur), limited depth range (aerial perspective) or they are less relevant for distance estimation towards obstacles (occlusion, linear perspective and shading and illumination).

Both cues have been experimentally observed in humans, also under conflicting conditions. Especially the vertical position in the visual field has some important nuances. For instance, Epstein shows that perceived distances do not solely depend on the vertical position in the visual field, but also on the background [22]. Another important contextual feature is the horizon line, which gains further importance when little ground (or ceiling) texture is present [23]. Using prismatic glasses that manipulated the human subjects' vision, Ooi *et al*. showed that humans in real-world experiments use the angular declination relative to the 'eye level' [24] rather than the visual horizon, where the eye level is the *expected* height of the horizon in the visual field. The apparent size of objects also influences their estimated distance. Sousa *et al*. performed an experiment where subjects needed to judge distances to differently-sized cubes [25]. The apparent size of the cubes influenced the estimated distance even though the true size of the cubes was not known and the height in the visual field and other cues were present. No work was found that investigates whether these observations also apply to neural networks for depth estimation.

## 3.3. Position vs. apparent size

As stated in section 3.2, the vertical image position and apparent size of objects are the most likely cues to be used by the networks. Figure 3.1 shows how these cues can be used to estimate the distance towards obstacles. The camera's focal length $f$ is assumed known and constant and is implicitly learned by the neural network. We furthermore assume that the camera's pitch angle relative to the horizon is small; pitch angles can therefore be approximated by a shift in vertical image coordinates $y$, where the horizon level $y_h$ is used as a measure for the camera's pitch. All coordinates are measured relative to the center of the image.

Given the obstacle's real-world size $H$ and apparent size $h$ in the image, the distance can be estimated using:

$$Z = \frac{f}{h} H \tag{3.1}$$

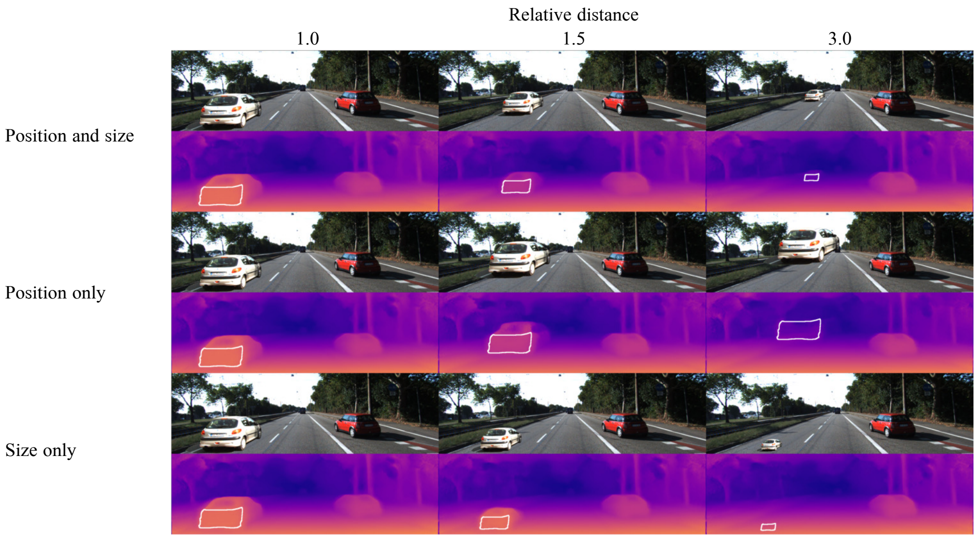


Figure 3.2: Example test images and resulting disparity maps from *MonoDepth*. The white car on the left is inserted into the image at a relative distance of 1.0 (left column), 1.5 (middle column) and 3.0 (right column), where a distance of 1.0 corresponds to the size and position at which the car was cropped from its original image. In the top row, both the position and apparent size of the car vary with distance, in the middle row only the position changes and the size is kept constant, and in the bottom row the size is varied while the position is constant. The region where the estimated distance is measured is indicated by a white outline in the disparity maps.

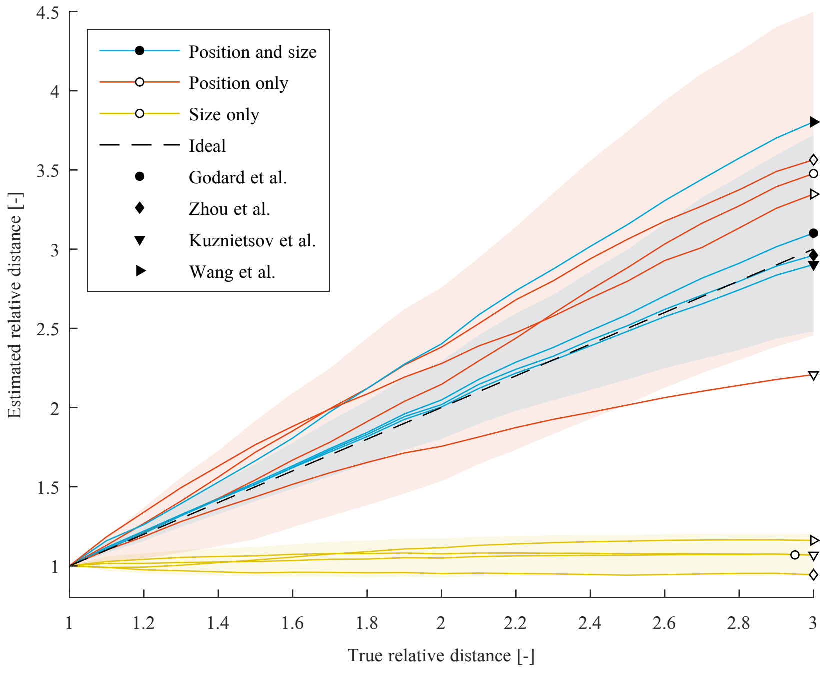


Figure 3.3: Influence of vertical image position and apparent size cues on depth estimates. Shaded regions indicate $\pm 1$ SD $(N = 1862)$ for the network by Godard *et al*. When both depth cues are present, all networks successfully estimate the distance towards the objects, except Wang *et al*.'s which overestimates the distance. When only the vertical position is available, the distance is either over- or underestimated and the standard deviation of the measurement increases (only shown for *MonoDepth*). When only the apparent size is available, none of the networks are able to estimate distance.

This requires the obstacle's true size $H$ to be known. The objects encountered most often in the KITTI dataset come from a limited number of classes (e.g. cars, trucks, pedestrians), where all objects within a class have roughly the same size. It is therefore possible that the networks have learned to recognize these objects and use their apparent size to estimate their distance.

Alternatively, the networks could use the vertical image position $y$ of the object's ground contact point to estimate depth. Given the height $Y$ of the camera above the ground, the distance can be estimated through:

$$Z = \frac{f}{y - y_h} Y \tag{3.2}$$

This method does not require any knowledge about the true size $H$ of the object, but instead assumes the presence of a flat ground and known camera pose $(Y,\, y_h)$. These assumptions also approximately hold in the KITTI dataset.

### 3.3.1. Evaluation method

To find which of these cues are used by the networks, three sets of test images are generated: one in which the apparent size of objects is varied but the vertical position of the ground contact point in the image is kept constant, one in which the vertical position is varied but the size remains constant, and a control set in which both the apparent size and position are varied with distance – as would be expected in real-world images.

The test images are generated as follows: the objects (mostly cars) are cropped from the images of KITTI's scene flow dataset. Each object is labeled with its location relative to the camera (e.g. one lane to the left, facing towards the camera) and with the position in the image it was cropped from. Secondly, each image in the test set was labeled with positions where an obstacle could be inserted (e.g. the lane to the left of the camera is still empty). Combining this information with the object labels ensures that the test images remain plausible.

The true distance to the inserted objects is not known; instead the network's ability to measure relative distances will be evaluated. Distances are expressed in relation to the original size and position of the object, which is assigned a relative distance $Z'/Z = 1.0$. The relative distance is increased in steps of 0.1 up to 3.0 and controls the scaling $s$ and position $x'$, $y'$ of the object as follows:

$$s = \frac{Z}{Z'}, \tag{3.3}$$

and

$$x' = x\frac{Z}{Z'}, \quad y' = y_h + (y - y_h)\frac{Z}{Z'} \tag{3.4}$$

with $x'$, $y'$ the new coordinates of the ground contact point of the object and with $y_h$ the height of the horizon in the image which is assumed constant throughout the dataset.

The estimated depth towards the car is evaluated by averaging the depth map over a flat region on the front or rear of the car (Figure 3.2). A flat region is used rather than the entire object to prevent the estimated length of the vehicle from influencing the depth estimate; the length is very likely dependent on the apparent size of the object, while the distance might not be.

### 3.3.2. Results

The results of this experiment are shown in Figure 3.3. When both the position and scale are varied, all depth estimates except Wang *et al.*'s behave as expected: the estimated depth stays close to the true depth of the object which shows that the networks still work correctly on these artificial images. When only the vertical position is varied, the networks can still roughly estimate the distance towards the objects, although this distance is slightly overestimated (Godard *et al.*, Zhou *et al.*, Wang *et al.*) or underestimated (Kuznietsov *et al.*). Additionally, the standard deviation of the distance estimate has increased compared to the control set. The most surprising result is found when only the apparent size of the object is changed but the ground contact point is kept constant: none of the networks observe any change in distance under these circumstances.

These results suggest that the neural networks rely primarily on the vertical position of objects rather than their apparent size, although some change in behavior is observed when the size information is removed. The fact that all four networks show similar behavior also suggests that this is a general property that does not strongly depend on the network architecture or training regime (semi-supervised, unsupervised from stereo, unsupervised from video).

## 3.4. Camera pose: constant or estimated?

The use of vertical position as a depth cue implies that the networks have some knowledge of the camera's pose. This pose could be inferred from the images (for instance, by finding the horizon or vanishing points), or assumed to be constant. The latter assumption should work reasonably well on the KITTI dataset, where the camera is rigidly fixed to the car and the only deviations come from pitch and heave motions of the car and from slopes in the terrain. It would, however, also mean that the trained networks cannot be directly transferred to different camera setups. It is therefore important to investigate whether the networks assume a fixed camera pose or estimate this on-the-fly.

If the networks can measure the camera pitch, then changes in pitch should also be observed in the estimated depth map. The unmodified KITTI test images already have some variation in the horizon level; in an initial experiment we look for a correlation between the true horizon level in the images (determined from the Velodyne data) and the estimated horizon level in the depth estimates from *MonoDepth*. The horizon levels were measured by cropping a central region of the disparity map (the road surface) and using RANSAC to fit a line to the disparity-y pairs. Extrapolating this line to a disparity of zero (i.e. infinite distance) gives the elevation of the horizon. For each image, this procedure was repeated five times to average out fitting errors from the RANSAC procedure.

Figure 3.4 shows the relation between the true and estimated horizon levels. While it was expected that *MonoDepth* would either fully track the horizon level or not at all, a regression coefficient of 0.60 was found which indicates that it does something between these extremes.

A second experiment was performed to rule out any potential issues with the Velodyne data and with the small ($\pm 10$ px) range of true horizon levels in the first experiment. In this second experiment, a smaller region is cropped at different heights in the image (Figure 3.5). For each image, seven crops are made with offsets between -30 and 30 pixels from the image center, which approximates a change in camera pitch of $\pm$2-3 degrees. Instead of

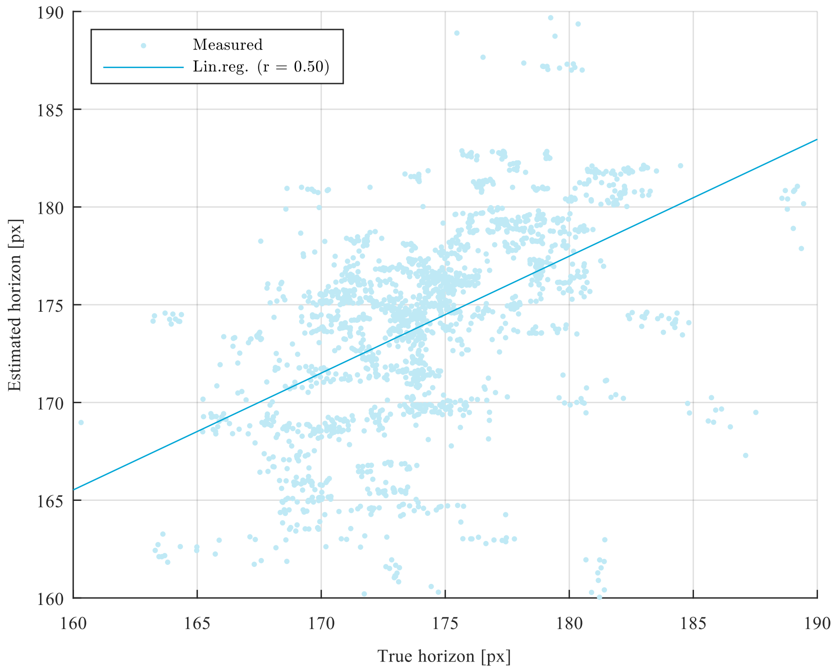


Figure 3.4: True and estimated horizon levels in unmodified KITTI images. Results for *MonoDepth* (Godard *et al.*). A medium-to-large correlation is found (Pearson's $r = 0.50$, $N = 1892$) but the slope is only 0.60, indicating that the true shift in the horizon is not fully reflected in the estimated depth map.

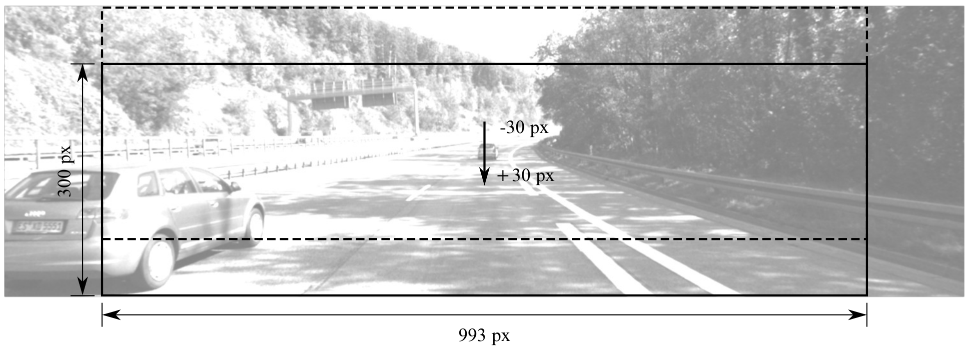


Figure 3.5: Larger camera pitch angles are emulated by cropping the image at different heights.

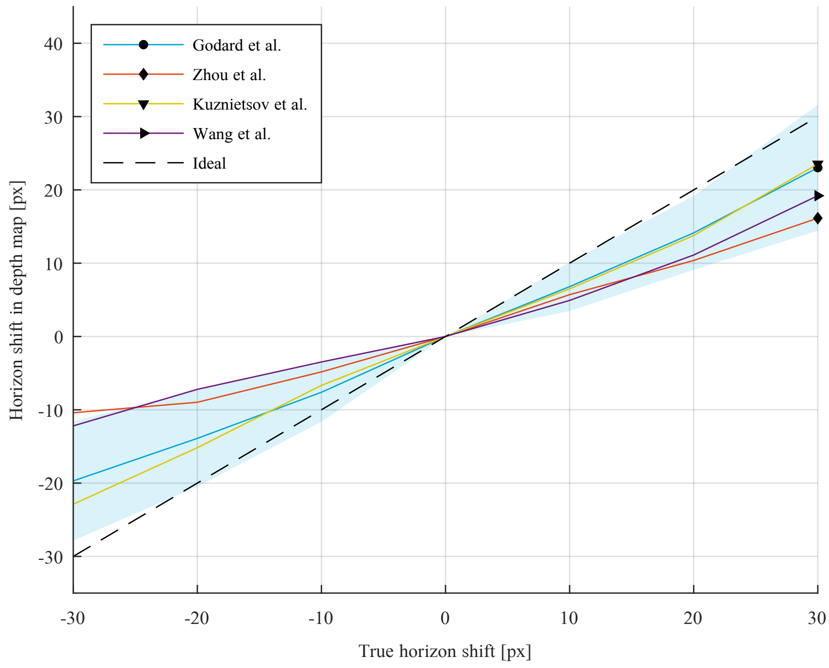


Figure 3.6: True and estimated shifts in horizon levels after cropping the images at different heights. Shaded regions indicate $\pm 1$ SD for the network by Godard *et al.* ($N = 194$, six outliers $> 3$ SD removed).

using the Velodyne data to estimate the true horizon level, the horizon level from the depth estimate of the centrally cropped image is used as a reference value. In other words, this experiment evaluates how well a *shift* in the horizon level is reflected in the depth estimate, rather than its absolute position.

The results for all four networks are shown in Figure 3.6. A similar result as in the previous experiment is found: all networks are able to detect changes in camera pitch, but the change in the horizon level is underestimated by all networks. Since the networks use the vertical position of obstacles to estimate depth, we expect this underestimation to affect the estimated distances. To test this hypothesis, we use the same pitch crop dataset and evaluate whether a change in camera pitch causes a change in obstacle disparities. The results are shown in Figure 3.7. The estimated disparities are indeed affected by camera pitch. This result also suggests that the networks look at the vertical image position of objects rather than their distance to the horizon, since the latter does not change when the images are cropped.

### 3.4.1. Camera roll

Similarly to the pitch angle, the roll angle of the camera influences the depth estimate towards obstacles. If the camera has a nonzero roll angle, the distance towards obstacles

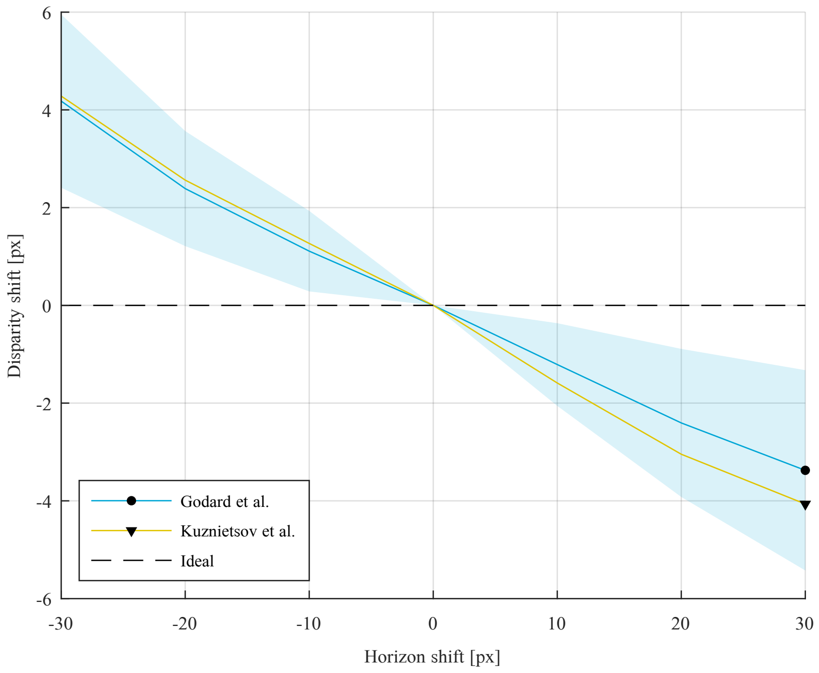


Figure 3.7: Changes in camera pitch disturb the estimated distance towards obstacles. Shaded regions indicate $\pm 1$ SD for the network by Godard *et al*.

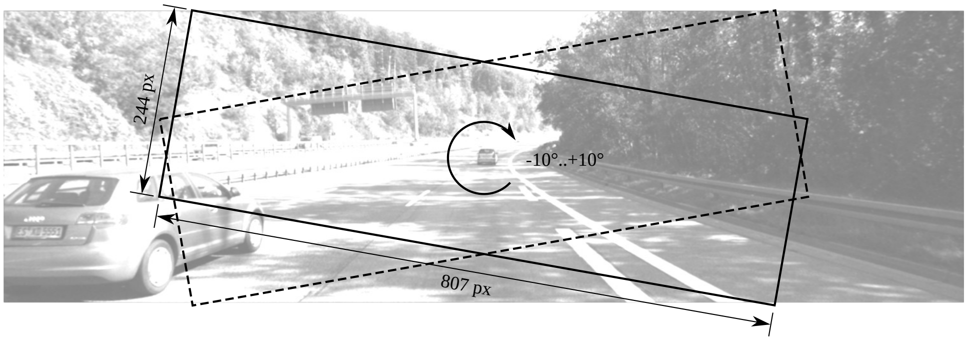


Figure 3.8: Camera roll angles are emulated by cropping smaller, tilted regions from the original KITTI images.

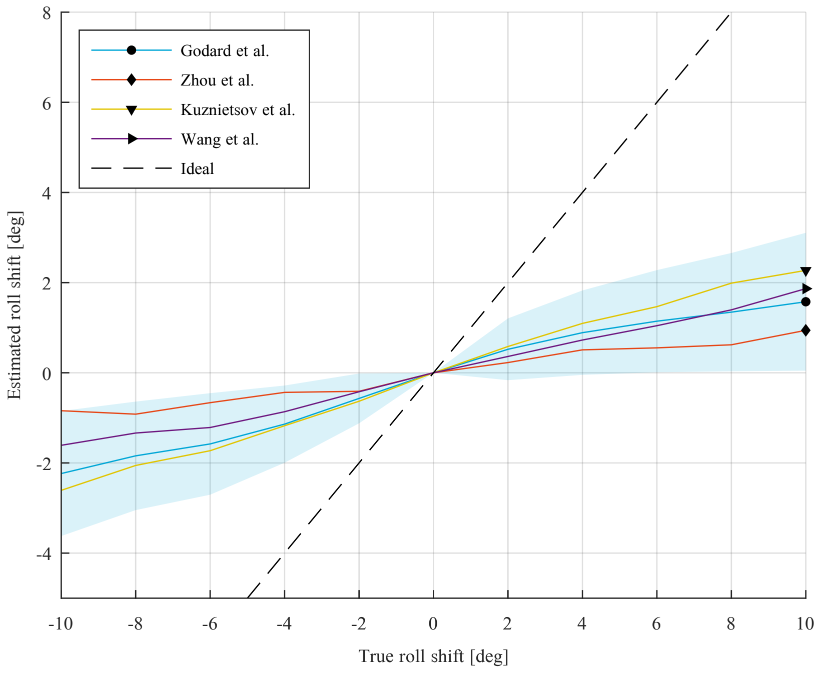


Figure 3.9: True and estimated roll shifts in the cropped images. For all networks, the change in road surface angle is smaller than the true angle at which the images are cropped. Shaded regions indicate $\pm 1$ SD for the network by Godard *et al*. ($N = 189$, eleven outliers $> 3$ SD removed).

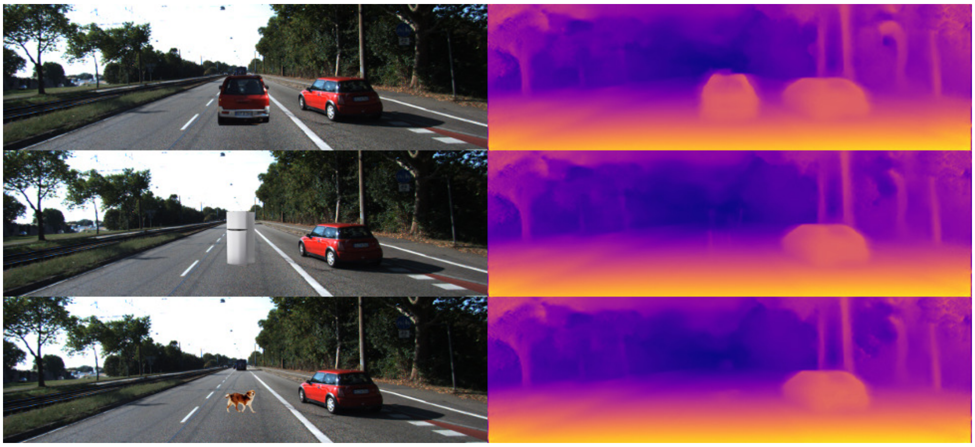


Figure 3.10: Objects that are not found in the training set (fridge, dog) are not reliably detected when pasted into the image.

does not only depend on their vertical position but also on their horizontal position in the image. A similar experiment was performed as for the pitch angle: a smaller region of the images was cropped at varying angles (Figure 3.8). To measure the roll angle, a Hough line detector was applied to a thin slice of the depth map to find the angle of the road surface. As in the previous experiment, we look for a correlation between the camera angle and the *change* in the estimated angle of the road surface. The result is shown in Figure 3.9 and is similar to that for pitch angles: all networks are able to detect a roll angle for the camera, but the angle is underestimated.

## 3.5. Obstacle recognition

Section 3.3 has shown that all four networks use the vertical position of objects in the image to estimate their distance. The only knowledge that is required for this estimate is the location of the object's ground contact point. Since no other knowledge about the obstacle is required (e.g. its real-world size), this suggests that the networks can estimate the distance towards arbitrary obstacles. Figure 3.10, however, shows that this is not always the case. The car is recognized as an obstacle, but the other objects are not recognized and appear in the depth map as a flat road surface.

To correctly estimate the depth of an obstacle, the neural networks should be able to: 1) find the ground contact point of the obstacle, as this is used to estimate its distance, and 2) find the outline of the obstacle in order to fill the corresponding region in the depth map. In this section, we attempt to identify the features that the *MonoDepth* network by Godard *et al*. uses to perform these tasks. The results of Figure 3.10 suggest that the network relies on features that are applicable to cars but not to the other objects inserted into the test images.

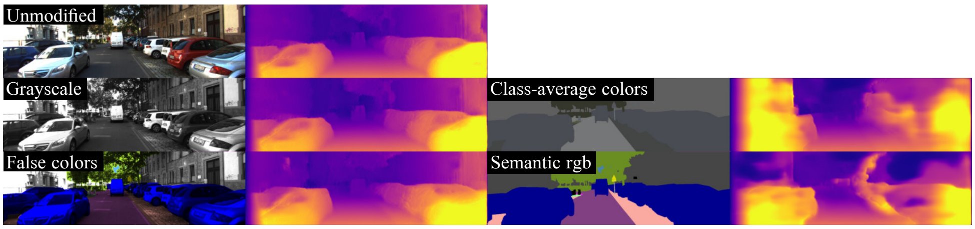


Figure 3.11: Example images and depth maps for unmodified, grayscale, false color, class average color, and semantic rgb images.

| Test set | Abs Rel | Sq Rel | RMSE | RMSE log | D1-all | $\delta < 1.25$ | $\delta < 1.25^2$ | $\delta < 1.25^3$ |
|---|---|---|---|---|---|---|---|---|
| Unmodified images | **0.124** | 1.388 | **6.125** | **0.217** | **30.272** | **0.841** | **0.936** | **0.975** |
| Grayscale | 0.130 | 1.457 | 6.350 | 0.227 | 31.975 | 0.831 | 0.930 | 0.972 |
| False colors | 0.128 | **1.257** | 6.355 | 0.237 | 34.865 | 0.816 | 0.920 | 0.966 |
| Semantic rgb | 0.192 | 2.784 | 8.531 | 0.349 | 46.317 | 0.714 | 0.850 | 0.918 |
| Class-average colors | 0.244 | 4.159 | 9.392 | 0.367 | 50.003 | 0.691 | 0.835 | 0.910 |

Table 3.1: *MonoDepth*'s performance on images with disturbed colors or texture. The unmodified image results were copied from [8]; the table lists results without post-processing. Error values for images that keep the value channel intact (*grayscale* and *false colors*) are close to the unmodified values. Images where the value information is removed and the objects are replaced with flat colors (*semantic rgb*, *class-average colors*) perform significantly worse.

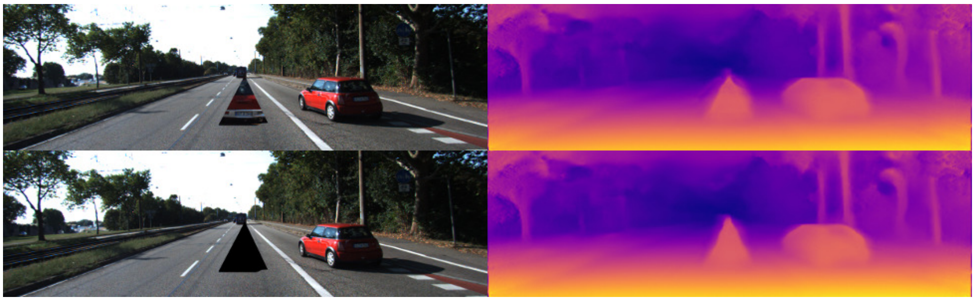


Figure 3.12: Objects do not need to have a familiar shape nor texture to be detected. The distance towards these non-existent obstacles appears to be determined by the position of their lower extent.

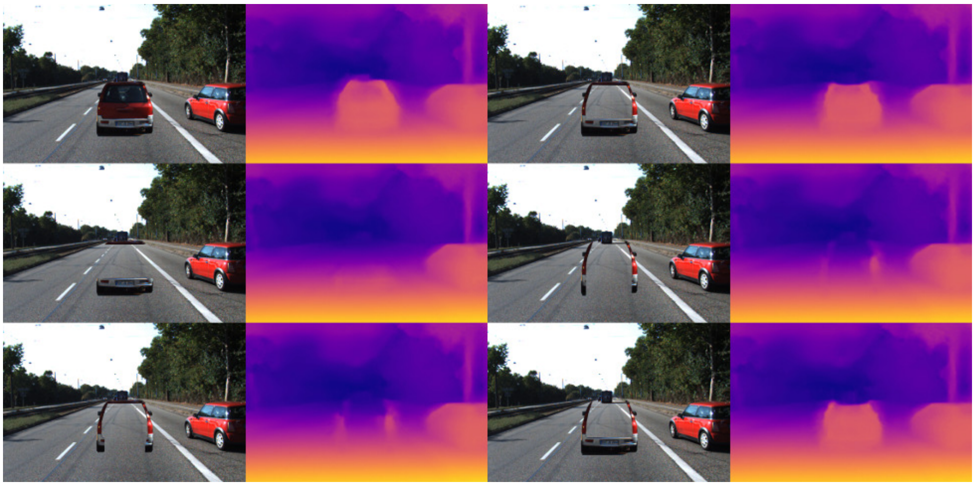


Figure 3.13: Influence of car parts and edges on the depth map. Removing the center of the car (top right) has no significant influence on the detection. The car's bottom and side edges (bottom right) seem most influential for the detected shape, which is almost identical to the full car image (top left).

### 3.5.1. Color and Texture

The objects inserted in Figure 3.10 differ from cars in terms of color, texture and shape. In a first experiment, we investigate how color and texture affect *MonoDepth*'s performance by evaluating it on modified versions of the KITTI images. To investigate the influence of color, two new test sets are created: one in which the images are converted to grayscale to remove all color information, and one in which the hue and saturation channels are replaced by those from KITTI's semantic_rgb dataset to further disturb the colors. Two other datasets are used to test the influence of texture: a set in which all objects are replaced by a flat color that is the average of that object class – removing all texture but keeping the color intact – and the semantic_rgb set itself where objects are replaced with unrealistic flat colors. Examples of the modified images and resulting depth maps are shown in Figure 3.11, the performance measures are listed in Figure 3.1.

As long as the value information in the images remains unchanged (the *unmodified*, *grayscale* and *false color* images), only a slight degradation in performance is observed. This suggests that the exact color of obstacles does not strongly affect the depth estimate. However, when the texture is removed (*class-average colors* and *semantic rgb*) the performance drops considerably. The network also performs better on the semantic_rgb dataset with false colors than on the realistically colored images. This further suggests that the exact color of objects does not matter and that features such as the contrast between adjacent regions or bright and dark regions within objects are more important.

### 3.5.2. Shape and contrast

Since color does not explain why the objects in Figure 3.10 are not detected, we next look at shape and contrast. A first qualitative experiment shows that objects do not need a familiar shape nor texture to be recognized (Figure 3.12). Furthermore, the distance towards these

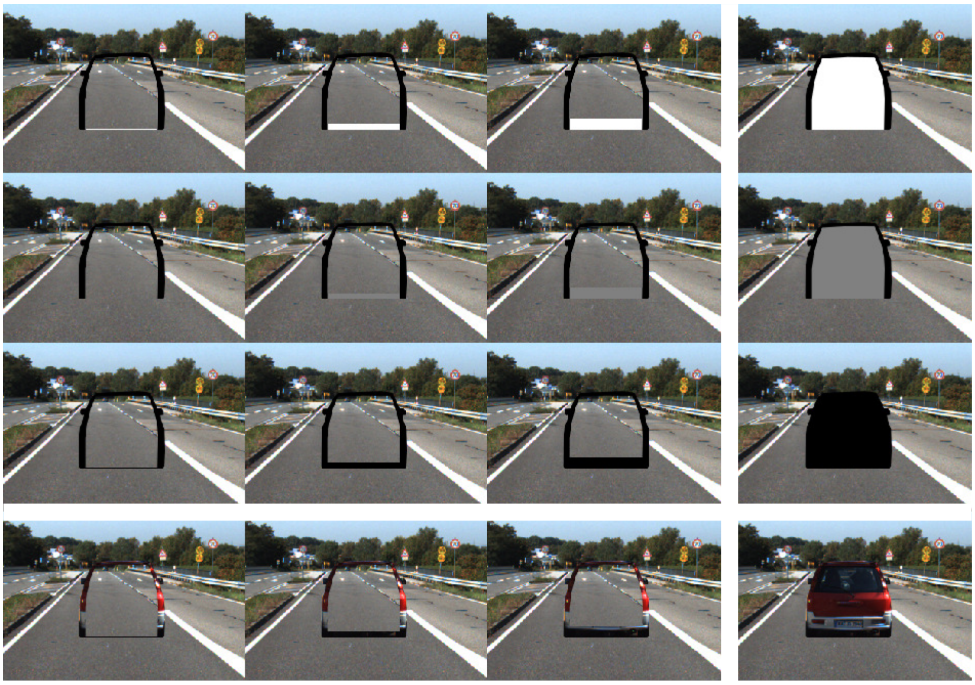


Figure 3.14: To measure the influence of the bottom edge, we vary its brightness and thickness. The experiment is repeated over 60 background images.

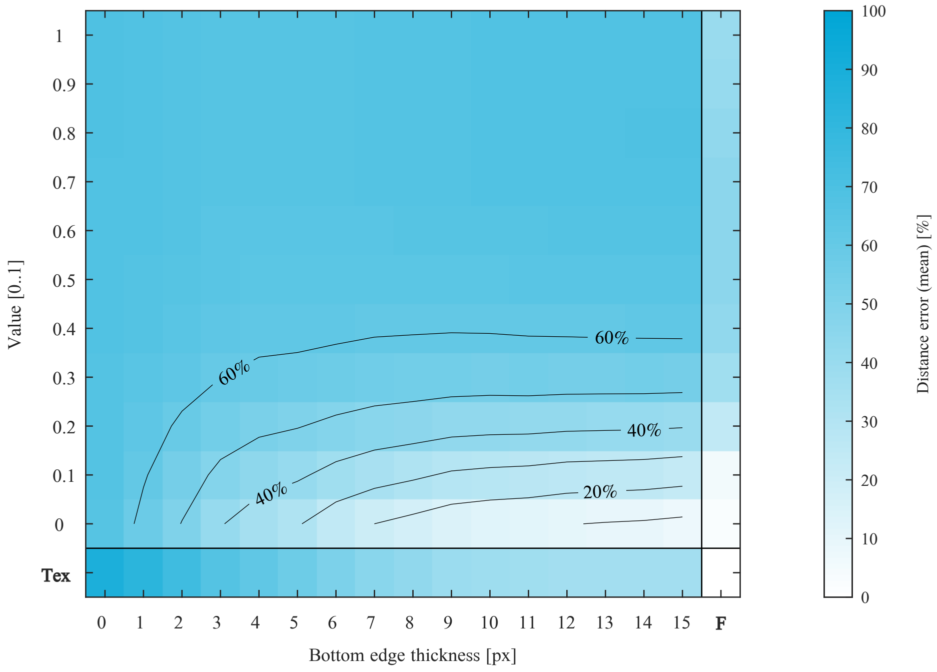


Figure 3.15: Mean distance error as a function of the bottom edge color and thickness. For comparison, we include results for realistically textured shapes (**Tex**) and completely filled shapes (**F**). Distance errors are measured relative to the estimated distance of the realistic (**F**, **Tex**) shape.

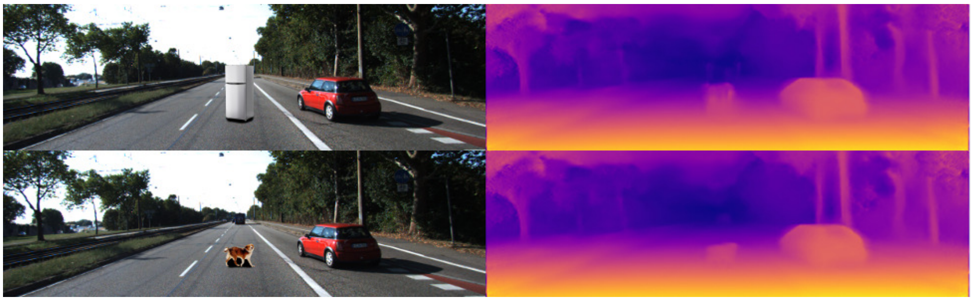


Figure 3.16: Adding a shadow at the bottom of the objects of Figure 3.10 causes them to be detected. The fridge, however, is only detected up to the next horizontal edge between the doors.

unfamiliar objects seems to follow from their lower extent, further supporting our claim that the networks use the ground contact point as the primary depth cue.

In a second experiment, we find the features that the network is the most sensitive to by systematically removing parts of a car until it is no longer detected. The car is still detected when the interior of the shape is removed, suggesting that the network is primarily sensitive to the outline of the object and ‘fills in’ the rest. The car is no longer detected when the side- or bottom edges are removed. However, when only the bottom edge is removed, the sides of the car are still detected as two thin objects.

We suspect that the dark region at the bottom of the shape is the main feature by which the network detects obstacles. The bottom edge formed by the shadow underneath the car is highly contrasting with the rest of the scene and an examination of the KITTI images shows that this shadow is almost universally present and could form a reliable feature for the detection of cars. We examine the influence of the bottom edge in a quantitative experiment where both the brightness and thickness of the lower edge are varied (Figure 3.14, 3.15). Additionally, the results are compared to completely filled shapes (**F**) and shapes with a realistic texture (**Tex**). We measure the error in the obstacle’s depth relative to the estimated depth of the realistic shape (**F**, **Tex**). The results are averaged over 60 background images in which the shape does not overlap other cars.

Figure 3.15 shows that the bottom edge needs to be both thick and dark for a successful detection, where a completely black edge with a thickness of $\geq 13$ px leads to an average distance error of less than 10% relative to a realistic image. A white edge does not result in a successful detection despite having a similar contrast to the road surface. Furthermore, a completely black edge results in a smaller distance error than when realistic textures are used. This suggests that the network is primarily looking for a dark color rather than contrast or a recognizable texture. Finally, the results show that completely filled shapes result in a better distance estimate. We suspect that completely filling the shape removes edges from the environment that could otherwise be mistaken for the outline of the obstacle.

As a final test, we add a black shadow to the undetected objects of Figure 3.10. The objects are now successfully detected (Figure 3.16).

## 3.6. Conclusions and future work

In this chapter we have analyzed four neural networks for monocular depth estimation and found that all of them use the vertical position of objects in the image to estimate their depth, rather than their apparent size. This estimate depends on the pose of the camera, but changes to this pose are not fully accounted for, leading to an under- or overestimation of the distance towards obstacles when the camera pose changes. This limitation has a large impact on the deployment of these systems, but has so far received hardly any attention in literature. We further show that *MonoDepth* can detect objects that do not appear in the training set, but that this detection is not always reliable and depends on factors such as the presence of a shadow under the object.

While our work shows *how* these neural networks perceive depth, it does not show where this behavior comes from. Likely causes are the lack of variation in the training set, which could be corrected by data augmentation, or properties inherent to convolutional neural networks (e.g. their invariance to translation but not to scale). Future work should investigate which of these is true and whether the networks can learn to use different depth cues when the vertical image position is no longer reliable.

**Acknowledgements** We would like to thank the authors of [8–10, 13] for making their code and models publicly available. This work was performed as part of the Percevite project, funded by the SESAR Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 763702.

## References

[1] Ruo Zhang, Ping-Sing Tsai, J. Cryer, and M. Shah, *Shape-from-shading: a survey,* IEEE Transactions on Pattern Analysis and Machine Intelligence **21**, 690 (1999).

[2] J. T. Barron and J. Malik, *Shape, Illumination, and Reflectance from Shading,* IEEE Transactions on Pattern Analysis and Machine Intelligence **37**, 1670 (2015).

[3] A. Lobay and D. A. Forsyth, *Shape from Texture without Boundaries,* International Journal of Computer Vision **67**, 71 (2006).

[4] P. Favaro and S. Soatto, *A geometric approach to shape from defocus,* IEEE Transactions on Pattern Analysis and Machine Intelligence **27**, 406 (2005).

[5] A. Saxena, S. H. Chung, and A. Y. Ng, *Learning Depth from Single Monocular Images,* Advances in Neural Information Processing Systems **18**, 1161 (2006).

[6] D. Eigen, C. Puhrsch, and R. Fergus, *Depth Map Prediction from a Single Image using a Multi-Scale Deep Network,* in *Advances in Neural Information Processing Systems 27* (Curran Associates, Inc., 2014) pp. 2366–2374.

[7] R. Garg, V. B. Kumar, G. Carneiro, and I. Reid, *Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue,* in *European Conference on Computer Vision*, edited by B. Leibe, J. Matas, N. Sebe, and M. Welling (Springer International Publishing, Cham, 2016) pp. 740–756.

[8] C. Godard, O. Mac Aodha, and G. J. Brostow, *Unsupervised Monocular Depth Estimation with Left-Right Consistency,* The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017).

[9] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, *Unsupervised Learning of Depth and Ego-Motion from Video,* in *CVPR* (2017) p. 7.

[10] C. Wang, J. Miguel Buenaposada, R. Zhu, and S. Lucey, *Learning Depth From Monocular Videos Using Direct Methods,* in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[11] M. Menze and A. Geiger, *Object scene flow for autonomous vehicles,* Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition **07-12-June**, 3061 (2015), arXiv:1512.02134 .

[12] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, *Indoor Segmentation and Support Inference from RGBD Images,* in *Computer Vision – ECCV 2012*, edited by A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 746–760.

[13] Y. Kuznietsov, J. Stuckler, and B. Leibe, *Semi-Supervised Deep Learning for Monocular Depth Map Prediction,* in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2017) pp. 2215–2223, arXiv:1702.02706 .

[14] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, *Visualizing higher-layer features of a deep network,* (2009).

[15] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks,* arXiv preprint arXiv:1312.6199 (2013), arXiv:1312.6199 .

[16] M. D. Zeiler and R. Fergus, *Visualizing and Understanding Convolutional Networks,* in *Computer Vision – ECCV 2014*, edited by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars (Springer International Publishing, Cham, 2014) pp. 818–833.

[17] C. Olah, A. Mordvintsev, and L. Schubert, *Feature Visualization,* Distill (2017), 10.23915/distill.00007.

[18] J. Hu, Y. Zhang, and T. Okatani, *Visualization of Convolutional Neural Networks for Monocular Depth Estimation,* (2019), arXiv:1904.03380 .

[19] J. J. Gibson, *The perception of the visual world* (Houghton Mifflin, Oxford, England, 1950).

[20] J. E. Cutting and P. M. Vishton, *Perceiving Layout and Knowing Distances: The Integration, Relative Potency, and Contextual Use of Different Information about Depth,* in *Perception of Space and Motion* (Elsevier, 1995) pp. 69–117.

[21] E. Brenner and J. B. Smeets, *Depth Perception,* in *Stevens' Handbook of Experimental Psychology and Cognitive Neuroscience*, edited by J. Wixted (John Wiley & Sons, New York, 2018) 4th ed., Chap. Depth Perc, pp. 385–414.

[22] W. Epstein, *Perceived Depth as a Function of Relative Height under Three Background Conditions,* Journal of Experimental Psychology **72**, 335 (1966).

[23] J. S. Gardner, J. L. Austerweil, and S. E. Palmer, *Vertical position as a cue to pictorial depth: Height in the picture plane versus distance to the horizon,* Attention, Perception, & Psychophysics **72**, 445 (2010).

[24] T. L. Ooi, B. Wu, and Z. J. He, *Distance determined by the angular declination below the horizon,* Nature **414**, 197 (2001).

[25] R. Sousa, J. B. Smeets, and E. Brenner, *Does size matter?* Perception **41**, 1532 (2012).

# 4 Visual route following

Navigation is an essential capability for autonomous robots. Especially visual navigation has been a major research topic in robotics, since cameras are lightweight, power-efficient sensors that provide rich information on the environment. However, the main challenge of visual navigation is that it requires substantial computational power and memory for visual processing and storage of the results. As of yet, this has precluded its use on small, extremely resource-constrained robots such as lightweight drones. Inspired by the parsimony of natural intelligence, we propose an insect-inspired approach toward visual navigation that is specifically aimed at extremely resource-restricted robots. It is a route-following approach, in which a robot's outbound trajectory is stored as a collection of highly compressed panoramic images together with their spatial relationships as measured with odometry. During the inbound journey, the robot uses a combination of odometry and visual homing to return to the stored locations, with visual homing preventing the buildup of odometric drift. A main novelty of the proposed strategy is that the number of stored compressed images is minimized by spacing them apart as far as the accuracy of odometry allows. To demonstrate the suitability for small systems, we implement the strategy on a 56-gram nano drone. The drone can successfully follow routes up to 100m with a trajectory representation that consumes less than 20 bytes per meter. The presented method forms a substantial step toward the autonomous visual navigation of tiny robots, facilitating their more widespread application.

An edited version of this chapter has been published as:

**van Dijk, T.**, De Wagter, C., de Croon, G.C.H.E. (2024). *Visual route following for tiny autonomous robots.* Science Robotics.

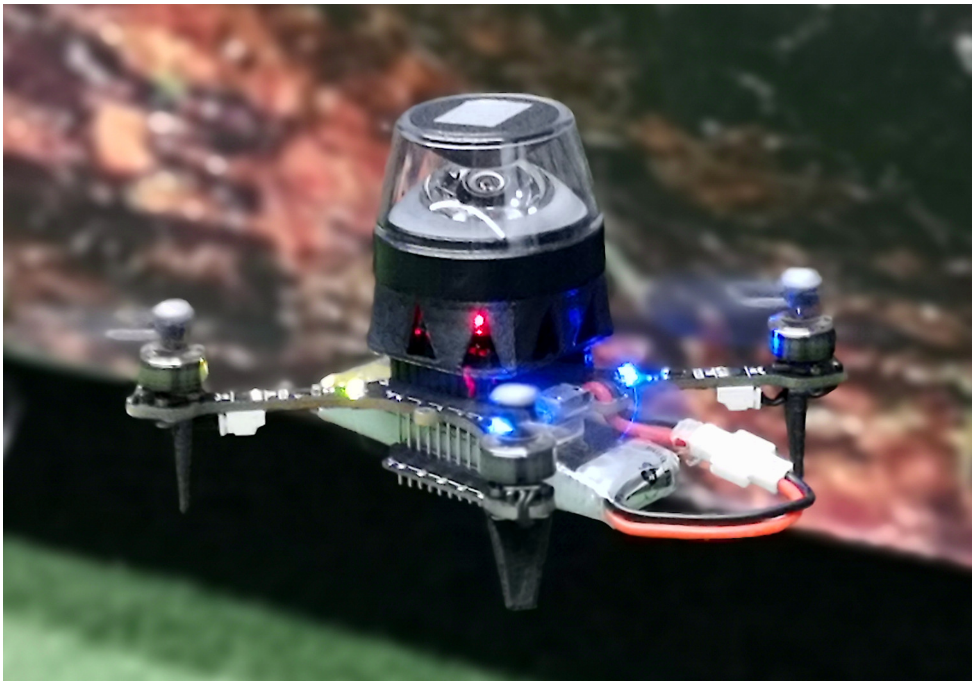


Figure 4.1: 56-gram nano-drone that performs autonomous visual route-following with its panoramic camera. We propose an insect-inspired approach to visual route-following that allows tiny, resource-restricted robots to follow long routes using only a microcontroller and exceptionally little memory.

## 4.1. Introduction

In order to do useful tasks, mobile autonomous robots need to navigate through their surrounding environment. Unlike their fixed counterparts, mobile robots will need to travel toward locations that are relevant to their mission or to return to their base after their mission has been completed. Currently, most autonomous robots rely on external infrastructure for localization and navigation, such as the Global Positioning System (GPS) outdoors [1] or ultra-wideband localization systems indoors [2]. However, for many applications this dependence on external infrastructure is undesirable. For one, the external infrastructure might not always be available, such as GPS in dense urban environments, extreme environments such as in caves, or when it is jammed. Secondly, it might be too impractical or time-consuming to set up additional infrastructure, especially in new or unknown environments such as in search-and-rescue operations. Even in fully controlled environments such as greenhouses or warehouses, costs might be another prohibitive factor. Hence, for many applications, it is vital that robots can navigate using only their own sensors, without reliance on external infrastructure.

While options are abundant for larger robots, this is unfortunately not the case for tinier systems such as the 56-gram nano drone considered in this chapter (Fig. 4.1) or even smaller systems [3]. First of all, the sensors might be too large, heavy or power-hungry for use on small platforms. This is for instance the case with Light Detection and Ranging

(LiDaR) sensors [4], which otherwise provide a popular and high-precision solution for larger robots. Vision-based navigation could be a solution here because cameras are passive sensors that can be both very lightweight and power efficient (e.g. [5, 6]). However, here we run into the second issue: excessive computational demands of the underlying vision algorithms. Mainstream approaches toward visual navigation tend to rely on Simultaneous Localization and Mapping (SLAM) [7], a class of algorithms that typically construct and maintain detailed, metrically accurate maps of the environment, while taking measurement uncertainties explicitly into account. The calculations that deal with these uncertainties and correct the map globally make SLAM computationally complex and memory intensive, requiring hundreds of megabytes to even several gigabytes to map medium-sized spaces in the order of tens of square meters [8]. Achieving navigation with the maps built by visual SLAM additionally requires algorithms for path planning and trajectory tracking. The resulting computational and memory demands require powerful embedded processing units that far exceed smaller robots' payload capacity or power budget.

For this reason, there has been an increasing amount of work that focuses on more efficient navigation solutions. For instance, the class of topological SLAM methods saves on computation and memory by forgoing the construction of a global metric map [9, 10]. A major concern for these methods is the visual distinctiveness of different places in the environment, since place recognition is necessary to globally correct the map. This typically still requires quite complex visual processing. Reducing the computational complexity of visual place recognition is hence an active research area [11, 12]. Other approaches to achieve more efficient navigation leverage the design of smaller, more powerful processors. For example, in [13] judicious software-hardware co-design led to a 2.4mW-consuming custom-designed chip that was able to perform visual-inertial odometry (VIO). VIO algorithms form a subset of SLAM in which there is no recognition of already visited places (known as “loop closure”), and hence they cannot eliminate odometric drift. Moreover, there is still some progress in scaling down existing processors despite the slowing down of processor miniaturization due to impending physical limits [14]. Examples of light-weight computing units include Google Coral's TPU [15], Intel's NCS [16] and the JeVois smartcam, cf. [17]. Until now, these processors have not yet brought vision-based navigation to very small robots, like lightweight nano-copters. Furthermore, even if processing power will increase in the future, it is questionable if one has the luxury to spend all that processing power on navigation. Real-world applications involve many other tasks, such as perception for obstacle avoidance or for recognizing mission-relevant objects. Hence, a parsimonious solution to navigation is and will remain highly relevant for small, autonomous robots.

Luckily, nature is a great source of inspiration for parsimonious solutions to navigation. Insects such as ants and bees can navigate over remarkable distances despite their tiny brains. For example, the desert ant Cataglyphis [18] can forage over long distances and then walk straight back to its nest, with journey lengths of up to 1 km! To bring a comparable algorithm to our robots, we must first understand how it works.

Biologists have studied insect navigation for more than a century, and have revealed its two core ingredients [19]. The first ingredient is path integration, which has a counterpart in robotics called odometry, i.e. the integration of traveled distance and direction to estimate one's position. For instance, ants determine the distance they have traveled by counting the number of steps they have taken, and by integrating ventral optical flow (i.e. how fast the ground is seen moving past them) [20]. The direction is measured with respect to the sun, and the corresponding polarization of the sky [18]. Using these measurements, ants can maintain an estimate of their position relative to their nest [19]. In recent work, the mechanisms behind path integration have been traced back all the way to the neural level, where ring attractors in the central complex play an important role [21–23].

While path integration can provide an estimate of position, it has one major downside: it is susceptible to drift because it integrates its measurement errors. To solve this, nature uses a second mechanism: view memory. Here, the environment itself forms an additional cue for localization or navigation. There is less consensus on how this view memory is used by insects [24–26]. The most relevant model for our work is the snapshot model, proposed by Cartwright and Collet [24] to describe the homing behavior of bees. In this model, the authors posit that bees remember the presence and location of landmarks in their visual field, as seen at their goal location. Then, in order to return, they try to maneuver such that the landmarks in their field of view move back to their remembered positions.

Of course, the highly efficient nature of insect navigation has not gone unnoticed by roboticists [27, 28]. An early study focused on visual homing with the help of artificial landmarks and showed that visual homing can enable robots to return to a known spot in the environment [29]. However, visual homing only works within a small region surrounding the target, called the catchment area. To navigate longer distances, a straightforward approach would be to home toward nearby targets in sequence. As long as the robot is inside the catchment area of the next snapshot, this will indeed succeed. However, catchment areas tend to be limited in size. As a result, snapshots need to be spaced close together, which means that a large number of snapshots needs to be stored to remember longer routes. Depending on the representation of the snapshot, which can range from full-resolution images [30] to more compressed representations [31], this can still require more memory than is available on tiny robotic platforms, like the 56-gram drone used in this study.

To reduce memory requirements, further proposals have been made in two directions. First of all, a reduction of the memory consumption of the snapshots. Many papers already reduce the images to a single row of pixels, where the lateral flow of features is sufficient for visual homing. However, Stürzl et al. take this one step further [31], by transforming this line into the frequency domain and remembering only the lowest-frequency components, thereby drastically reducing the size of the snapshot even further. The second direction is to increase the spacing between snapshots. In [32] an improvement is proposed in which the homing vector is used as a position estimate relative to the snapshot. This allows the drone to navigate some distance toward the next catchment area, provided that the vector is accurate enough. As a result, the overlap between snapshots is reduced, though not eliminated. A simulation study in [33] proposes to combine odometry with visual homing. New snapshots are taken when the odometry and visual homing estimates of the direction toward the snapshot start to diverge. However, since this happens at the edge of the catchment area, this method still results in considerable overlap between the catchment areas of subsequent snapshots. In this chapter, we propose an approach to substantially increase the distance between snapshots and combine it with a memory-efficient homing algorithm.

### 4.1.1. Efficient visual route following

To bring visual navigation to tiny robots, we present a highly memory-efficient strategy for visual route following. We propose to traverse longer distances by better exploiting the combination of visual homing and odometry (Fig. 4.2). In this framework, we assume that the robot first performs an outbound flight towards a mission goal, followed by an inbound flight back to the starting location. The outbound flight can in principle be performed under any control law, including manual control. Since our focus in this chapter lies on route following during the inbound flight, we assume that the outbound flight was performed without collision and that the environment is static such that the route remains free of obstacles.

During the inbound flight, most of the distance is covered using odometry, but without any correction, the odometric drift would eventually become too large. To correct this drift, we let the robot use visual homing to periodically return to known locations in the environment (the snapshot locations). During homing it compares its omnidirectional image only with the current active snapshot. Since the robot is now at a known position, it can reset its pose estimate and thereby eliminate any incurred odometric drift. After this reset, the robot can start a new leg of odometry and visual homing, where the initial error only consists of the homing inaccuracies at the last snapshot.

The proposed strategy is extremely memory efficient as we propose to space the snapshots as far apart as possible. Specifically, we forward that the distance between snapshots is ultimately limited by the accuracy of the odometry, in that the strategy will succeed as long as the drone can reliably end up inside the next catchment area. Given a reasonable odometry accuracy, this distance can be far greater than when overlapping catchment areas for consecutive snapshots are required.

In the remainder of this chapter, we implement and evaluate this strategy step-by-step. First, we evaluate and compare different visual homing algorithms by their memory efficiency, where we take both the size of the snapshot and of the resulting catchment area into account. After selecting a suitable algorithm for efficient visual homing, we continue with the key experiment of this chapter: we implement our strategy on a tiny 56-gram nano drone and let it follow trajectories of up to 100 meters. We show that periodically homing toward a snapshot indeed eliminates the buildup of odometric error over time. Finally, we demonstrate the same strategy on more complex trajectories and environments.

By implementing our strategy on a 56-gram nano drone equipped with only an STM32F4 microcontroller featuring a mere 192 kB of memory, we prove that the proposed algorithm is especially suitable for tiny robots. With our contribution, we bring autonomous visual navigation to a class of robots where this was previously unavailable.

## 4.2. Results

### 4.2.1. Selected homing algorithms for comparison

As a first step toward memory-efficient visual route following, we compare homing algorithms in terms of their memory efficiency. To aid the search for an efficient algorithm, first, we broadly categorize visual homing algorithms along two axes. Firstly, we make a distinction between ‘steering methods’ or ’visual compass’-based methods, and ‘vector methods’. The steering methods are characterized by their calculation of a steering angle

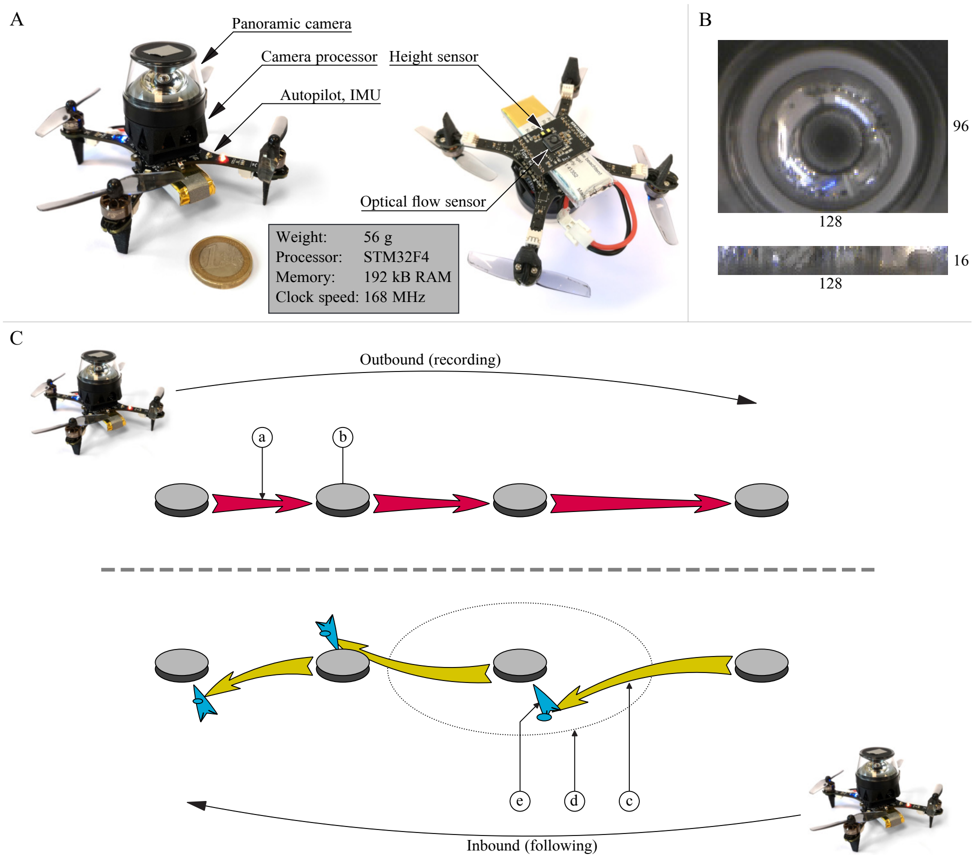


Figure 4.2: Proposed navigation strategy. (A) The experimental platform considered in this chapter, a 56-gram Crazyflie Brushless nano drone. (B) Raw and unwrapped omnidirectional camera image. (C) The route following strategy. During the outbound trajectory, which could be performed under an arbitrary control law (a), the robot periodically takes snapshots (b) of its surroundings. To follow the same route in reverse, the robot first uses odometry (c) to move toward the location of the next snapshot. For success, it is vital that the drone ends up inside the catchment area (d) of this snapshot. Hence, the distance between snapshots has to be proportional to the expected odometry drift and catchment area size. After the odometry movement has been completed, the robot uses visual homing (e) to converge to the snapshot location and thereby cancel the incurred odometric drift. These steps are repeated until the robot is back at its intended location.

or rate for position control. Typically, these methods depend only on features in the forward field of view of the robot. Steering methods are often used in visual-teach-and-repeat navigation tasks and have been successfully demonstrated over long distances, such as in [34–37]. Additionally, convergence has been proven for straight-line segments [38] and more complex paths [36, 39].

On the other hand, vector-based methods typically do not produce a steering angle, but a vector toward the snapshot. In contrast to steering-based methods, vector methods tend to not have an obvious ‘forward’ direction and typically rely on a panoramic field of view. While vector methods are often used to home toward a single point, they have also been used in visual route following such as in [30, 40], though research in this direction has been far less extensive than for steering methods.

While steering-based methods might be an obvious choice, in this chapter we choose to focus on vector-based approaches. Our reasoning is as follows: first of all, steering-based methods require their reference images to be spaced close together. Following the proof in [38], the distance between images must be smaller than the distance to the dominant feature in the environment; where a distance of 35cm is used in said article. We expect to achieve a far greater spacing using odometry. Secondly, we aim to bring the robot as close to the snapshot as possible, ideally on top of it. As a result, we have no guarantee that the snapshot will be in front of the robot after traveling with odometry, it may just as likely lie behind it or to its side. Finally, we think that a vector is a more natural way to express movement for holonomic systems like our drone.

The second axis by which we categorize homing algorithms is the way in which snapshots are represented. We consider two broad categories of snapshot representations for use in visual homing: landmark-based and holistic representations. Landmark-based methods represent the snapshot as a collection of point landmarks that each have their own bearing. Consequently, in landmark-based visual homing, points in the environment are tracked from the current to the target image. From the point correspondences between these two images, a homing vector directed toward the target can be derived, for instance using Visual Servoing [41–43]. In order to describe and track landmarks, keypoint detectors and descriptors from computer vision are used. Examples include the computationally expensive Scale Invariant Feature Transform (SIFT) [44] features and the much more efficient Binary Robust Invariant Scalable Keypoints (BRISK) [45]. Additionally, the bearing toward each landmark needs to be stored. All in all, this still leads to a sizable memory consumption, especially when a large number of snapshots needs to be remembered. In order to reduce memory consumption, it is possible to share descriptors between multiple snapshots as demonstrated by Stelzer et al. [46]. However, the size of a snapshot remains in the order of hundreds of bytes or more.

On the other end of the spectrum are the holistic methods. Unlike landmark-based algorithms, these operate on the image as a whole. Instead of matching the bearings of landmarks, the entire current and target images are matched, for instance with the Sum of Square Differences (SSD). This leads to an Image Difference Function (IDF) (see Fig. 4.3), which should be zero when the current and target view coincide and - more importantly - increase smoothly with distance near the target location. By finding the direction in which the IDF decreases, homing can be performed. One option to detect and follow the gradient down the IDF is to make physical movements, as demonstrated by Zeil et al.

[47]. Finding the gradient in this way may be time-consuming. Therefore, Franz et al. [30] propose an alternative method in which small movements are simulated by warping the image. Using this method, the authors perform a brute-force search over multiple potential movements and select the best match. Hence we term the method 'search' in the following. As a computationally more efficient alternative, Moller et al. [48] suggest only predicting two perpendicular movements and using these to estimate the gradient of the IDF. By following the gradient, the robot will end up in the (local) minimum of the IDF. In later work, they include the second-order gradient as well. They term their approach 'MFDID'. Storing entire images is not ideal in terms of memory efficiency; in fact, it is worse than most landmark-based approaches. A first improvement is that the snapshot images can be vertically averaged, as just the lateral flow should already be sufficient to find the homing vector. On top of that, Stürzl et al. [31] show that these one-dimensional snapshots can be significantly compressed while maintaining homing performance. This compression is performed by first transforming the snapshots to the frequency domain, and then only keeping the lowest frequency components, where most of the power is found in natural images. The authors show that homing is still possible using only the lowest 5 components. With appropriate rounding, such a snapshot could be stored in as little as 10 bytes per snapshot. Figure 4.3A shows a raw panoramic image and beneath it the reconstruction by using the highly compressed Fourier representation. It can be observed that this method, which we term 'Fourier', captures the coarse vertical structures in the environment. Because these images can ultimately be compressed further than the bearing-descriptor pairs of landmark-based homing, we will focus on holistic algorithms for the remainder of this chapter.

### 4.2.2. Comparison of holistic visual homing algorithms

The selection of a specific visual homing algorithm means exploring a trade-off between the catchment area size, memory consumption of snapshots and computational demands. We evaluate the catchment areas from the above-mentioned holistic vector-based methods [30, 31, 48] with the help of the publicly available panoramic image dataset by Gaffin and Brayfield [49]. The dataset contains $100 \times 100$px grayscale images taken at 12.7cm intervals in a $7.3 \times 6.9$m room and part of the adjacent corridor. To evaluate the catchment area, we generate snapshots throughout the environment and for each snapshot we calculate the homing vectors at all panoramic image locations in the room (Fig. 4.3B). Then, from each starting position, homing trajectories are generated by integrating the bilinearly-interpolated homing vectors. These trajectories let us determine the final position of the robot after homing, and thereby find the set of starting locations from which homing would be successful, i.e., the catchment area. Since the catchment area can have a highly irregular shape (see the dark blue area in Fig. 4.3B), we take the number of successful starting cells as a measure of the size of the catchment area. The top plot in Figure 4.3C shows the relationship between the snapshot size in bytes (B) and the average catchment area in m2. The Fourier method leads to the largest catchment areas for snapshot sizes below 32 bytes, while the Search method gives the largest catchment areas above. The lower plot in Figure 4.3C shows the average ratio of the catchment area with respect to the snapshot size. It shows that the Fourier method is highly efficient for small snapshots. Because this algorithm is both memory-efficient and computationally cheap, it will be used in our fur-

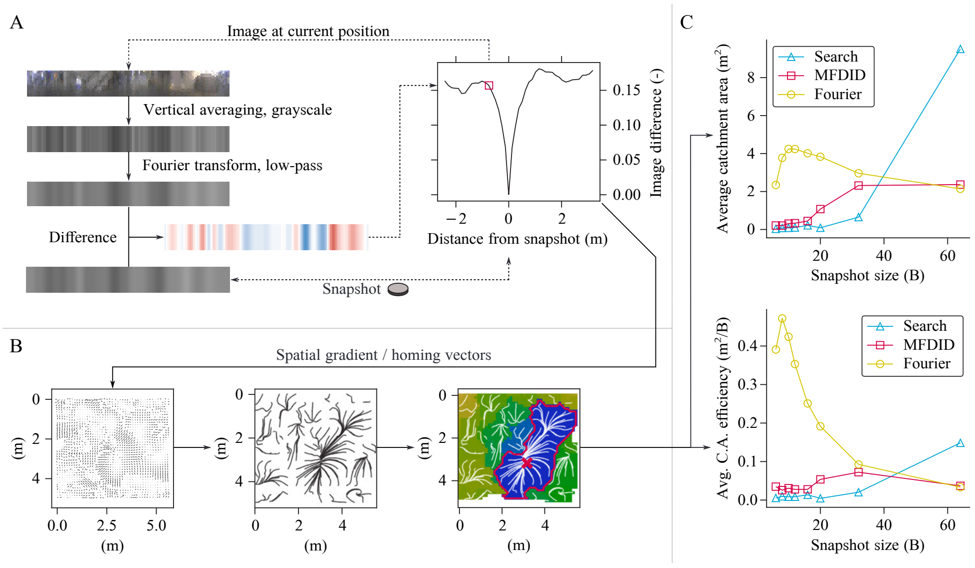


Figure 4.3: Comparison of visual homing algorithms. (A) Illustration of the Image Difference Function (IDF) with the Fourier method [31]. Images are compressed using vertical averaging and a low-passed Fourier transform. The difference between the compressed image at the current and target location increases smoothly with distance, as shown in the plot. Finding the minimum in this IDF brings the robot back to the target position. (B) To evaluate the size of the catchment area belonging to a snapshot, we generate all homing vectors (left), predict the robot trajectories (center), and collect all starting points for which the endpoint error is small (right, error indicated by color). (C) Homing algorithms are compared by the size of their catchment area, given a snapshot size in bytes. At small snapshot sizes, Fourier-based homing performs best, while at larger sizes Search-based homing is more effective. In terms of efficiency in $m^2$ of catchment area per byte, Fourier-based homing performs best.

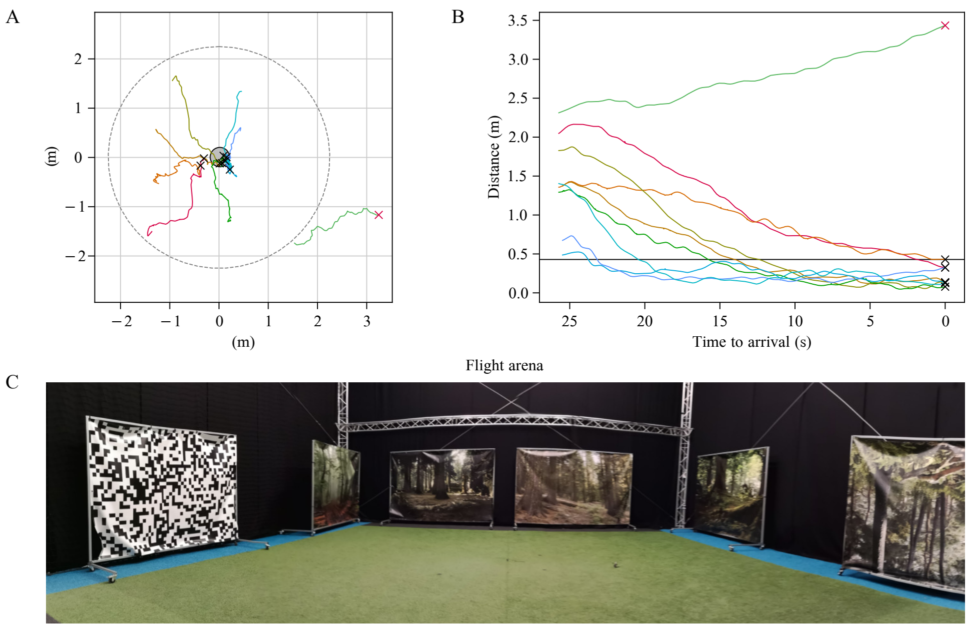


Figure 4.4: Visual homing toward a single point. The drone is commanded to home over longer distances toward a snapshot at the center of the flight arena (grey circle). Visible are the homing trajectories (A, colored lines, crosses for the end positions) and the decreasing distance toward the snapshot over time (B, colored lines, black line indicates the worst homing performance attained in this experiment for trajectories converging to the snapshot). For illustration purposes, a possible catchment area is drawn with a dashed line. One of the starting positions lies outside of the catchment area and leads the drone to diverge (red cross as end position). Homing is successful for distances well over one meter, while the odometric drift between snapshots during route following is expected to be significantly smaller.

ther experiments. Please note that the choice for this representation entails a dependence on contrasts along the horizon line, and mainly large vertical features such as walls, doors, windows, trees, etc. Furthermore, the idea of snapshot matching relies on these contrasts being static.

### 4.2.3. Visual homing and odometry

Before combining visual homing and odometry into a single navigation strategy, we examine these elements in isolation. We first validate the homing performance of the selected visual snapshot representation on our robotic platform, a 56-gram, Bitcraze Crazyflie Brushless (Fig.4.1, 4.2A). This tiny 12.5cm drone carries a 10-gram panoramic camera assembly (included in the 56-gram take-off weight). The assembly includes an STM32F4 chip for processing the omnidirectional images onboard in real time. Furthermore, the drone is equipped with a ‘flow deck’ with a downward-looking camera and a tiny laser ranger to measure optical flow and height, respectively. Combining these measurements results in velocity estimates, which are used for odometry.

In the homing experiment, the drone is first directed to the center of our testing environment, a $10 \times 10 \times 7$m flight arena termed the ‘Cyberzoo’ (see Fig. 4.4C for an impression).

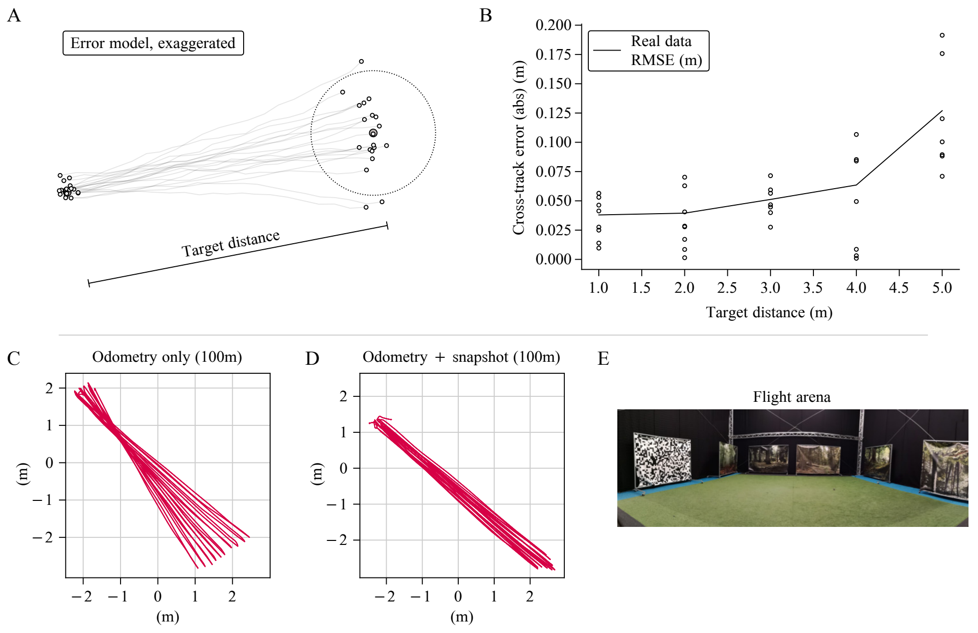


Figure 4.5: Combining visual homing and odometry. Each leg of the route-following strategy consists of a homing operation toward a snapshot, followed by an inbound maneuver toward the next snapshot using odometry. (A) The error model with exaggerated noise parameters, showing both the initial pose error after homing and the increased spread due to odometry. (B) Experimental data for real-world odometry experiments in which the traveled distance is varied. (C, D) Proof-of-concept demonstration. The drone is commanded to fly a 5m trajectory back and forth for a total distance of 100m. In (C), the drone only uses odometry, while in (D) it periodically homes to a snapshot. Repeatedly homing toward snapshots is shown to prevent drift over longer trajectories. (E). Overview photo of the test environment.

Subsequently, we commanded the drone to go to a small number of locations away from the target, to a maximum distance of approximately two meters. Then, the drone performed visual homing with the Fourier method. Figure 4.4 shows an overhead view of the drone's trajectories (A), the distance to the target location over time during homing (B), and an overview image of the flight arena (C). We observe that eight out of nine runs bring the drone close to the target location, i.e. within $\sim$0.5m, at which point the run is ended. The failed run starts at one of the outer positions. By definition, the failed homing attempt means that the starting position is outside of the (unknown) catchment area of the snapshot in the center. The experiment also shows that the drone does not always fly straight to the target location, an indication that the homing vector can point in a different direction than the target vector. As long as the homing vector is within $\pm 90^\circ$, however, the distance will decrease and the drone will eventually arrive at the target location.

As explained earlier, our strategy relies on spacing the snapshots as far apart as odometry allows. Specifically, the drone should end up just inside the next catchment area. The drone's positioning accuracy depends on two main factors: the accuracy of its starting pose after homing, and the drift incurred while moving toward the new position. This is demonstrated by means of a simulation of multiple trajectories in Fig 4.5A, in which the standard deviations of the initial position and heading, yaw rate and velocity errors are exaggerated to demonstrate their effect. The model shows that at short distances the position error is primarily caused by errors in the initial pose, as it is almost constant. For longer distances the error begins to grow because of integrated odometry errors. At the end of a route leg traveled by means of odometry, the cross-track error is larger than the along-track error. This can be seen in Figure 4.5A, where there is a larger spread orthogonal to than along the route, and in Figure 5C, in which this is also the case for the real-world odometry experiments. This effect is caused by the heading error, consisting of an initial offset and subsequent drift. Figure 4.5B shows the absolute cross-track errors of the drone experiment. Overall the accuracy is quite good, with a cross-track RMSE of 13cm after five meters of travel. The plots in Figure 4.5C, 4.5D show a similar experiment but over longer distances. The drone traverses a line of approximately five meters back and forth ten times. In C, the drone only uses odometry for this procedure. In D, the drone has recorded a snapshot in the top left corner and uses this to re-align itself on each arrival there. The results show that the odometry does indeed drift and that the drift becomes significant for longer distances. They also show that our periodic re-alignment scheme, while introducing some error due to homing inaccuracies, prevents the buildup of odometric drift over time and as a result keeps the error bounded when traveling longer distances.

### 4.2.4. Route following with minimal memory

With the core principles proven, we now demonstrate the complete strategy on more complex trajectories and environments. We created different types of trajectories, of which the outbound portion is traversed using odometry (without any global position feedback). After the outbound journey is completed, the drone starts its inbound journey with the help of the proposed insect-inspired navigation strategy. We qualitatively compare the route following accuracy with respect to the outbound trajectory. A motion capture system is used to record the absolute position of the drone during its outbound and inbound flight. These measurements are never communicated to the drone, they are only used for evaluation after the experiment. The trajectories consist of multiple traversals of an S- (Fig. 4.6A) or U-shape (Fig. 4.6B). The trajectories were repeated to maximize the travel distance within the limited testing area. The resulting path lengths (of the outbound route) were 40m for the S-shaped trajectory and 56m for the U-shaped one. Ultimately, the length of the experiment was limited by the battery capacity of the drone.

Above we have mentioned that snapshots are spaced as far as odometry allows. Of course, given the varying unknown shapes and sizes of catchment areas and the variable nature of drift, choosing a spacing between snapshots has implications for the trade-off between navigation robustness and memory expenditure (see Section A.3). In our experiments, we use a fixed one or two-meter spacing between snapshots during the experiments. Firstly, these are conservative values where the position error is primarily dominated by homing inaccuracies while the odometric drift between snapshots remains small. Secondly, this gives us a larger number of visual homing attempts, which gives a better indication of its use and robustness during route following.

Figure 4.6 shows the resulting trajectories for the proposed method (A, B). It successfully and reliably follows the outbound trajectory back to the start. The route following memory for the U-trajectory consists of 31 16-byte snapshots, and two to three 2-byte

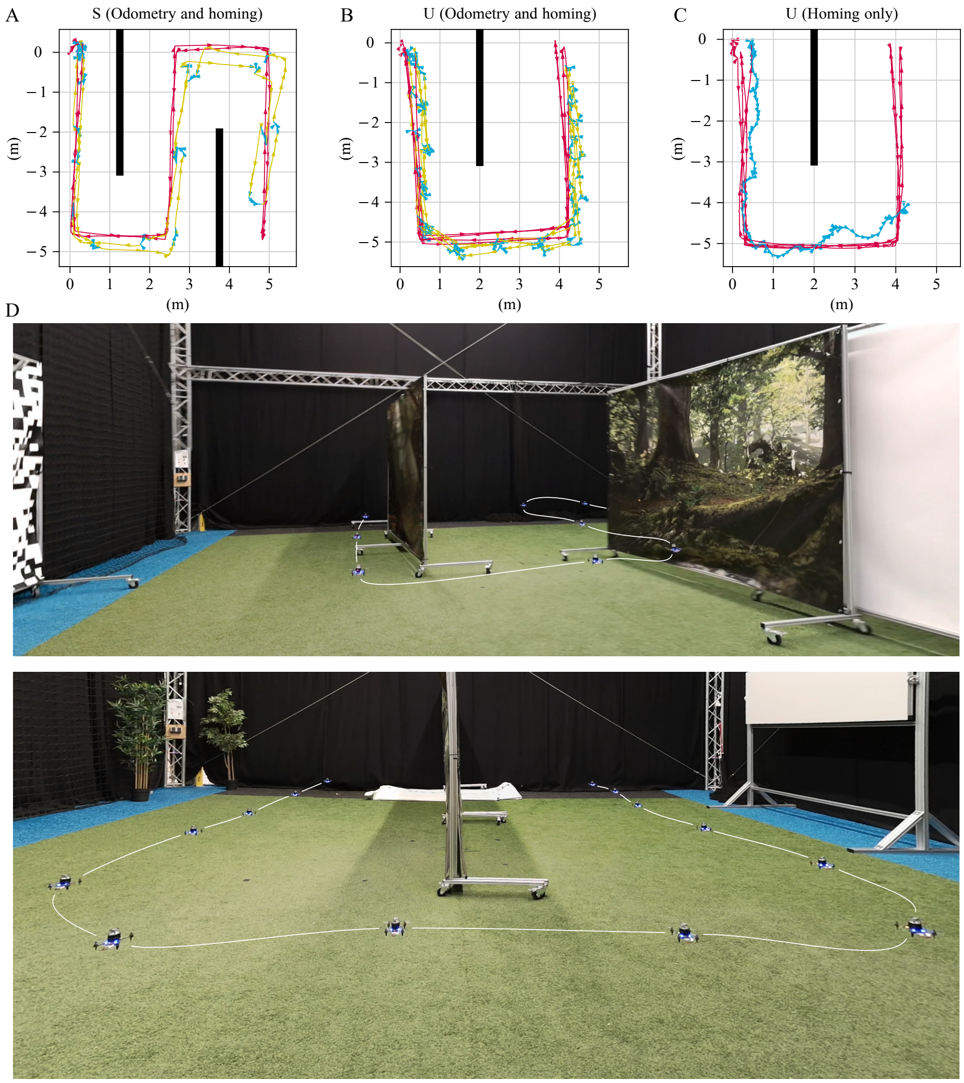


Figure 4.6: Validation of the complete route following strategy. The drone can successfully follow long routes in complex environments where it is impossible to see the entire route from a single point-of-view. The outbound route is shown in red, route following using odometry in yellow and homing in blue (A, B, C). The drone traverses the route multiple times during the outbound- and inbound segments to maximize the length of the trajectory (all in the order of 50m). For comparison, a route-following attempt in which the drone only uses sequential visual homing as a strategy is shown (C). The drone can successfully follow the route but stops early because the increased travel time caused the battery to run out before even completing one stretch of the trajectory. Time-lapse photos of the experiment are shown in D.

odometry vectors between snapshots, leading to a total memory size of 0.65kB for a distance of 56m. We also compare our method to sequential visual homing on one of the trajectories (Fig. 4.6C). The experiment shows that homing between successive snapshots is indeed a viable method of navigation. However, the snapshots had to be spaced at a distance of 25cm; earlier attempts with 1m spacing consistently failed. Because the catchment area scales proportionally to the size of the environment (i.e. distance to the dominant features, the walls in this case), they were significantly smaller than in Figure 4.4. Besides the increased memory consumption, the homing procedure is also relatively slow (to prevent overshoot or large pitch/roll angles), and as a result the drone traveled a significantly shorter distance before the battery ran out. In comparison, the new strategy has a significantly higher average speed, while the tracking error is in the same order of magnitude.

We also performed a number of experiments to evaluate the robustness of the proposed approach. One important characteristic of the chosen snapshot representation (with vertical averaging and Fourier compression) is that it depends on prominent vertical features (contrasts along the horizon line). Such features are commonly present in both indoor and outdoor environments. To illustrate this, we made a set of snapshots in various places in the building of the Faculty of Aerospace Engineering at TU Delft (see Section A.1). The low resolution of the snapshots may initially seem purely disadvantageous for accurate homing, but actually also brings some robustness against small, dynamic objects. This is illustrated with an experiment in which we monitor the resulting home vector while we move objects around the robot (Section A.1). Sometimes in indoor environments there are corridors with purely uniform walls. In that case, the current approach will not be able to correct the drift in the direction of the corridor (lateral drift can be cancelled due to the different appearance of the floor and walls). Furthermore, in order to show that the drone is also able to follow routes in different indoor environments, we have performed additional experiments in three different places at the faculty of aerospace engineering: close to an airplane simulator SIMONA [50], in an office hallway, and in our lab-space (Fig. 4.7). Videos of these flights can be found as supplementary material. Finally, since the flight time of the real drone is limited, we performed simulation experiments to show that also longer distances can be covered with the proposed strategy. Specifically, in the AirSim simulator [51], a simulated AR drone is able to use the strategy to successfully track a 300m trajectory in a forest environment (Section A.2).

## 4.3. Discussion

We have proposed an insect-inspired navigation strategy for route following, which extensively depends on odometry to reduce memory usage. The strategy was demonstrated on the lightest robot to date to perform vision-based navigation, a 56-gram Crazyflie drone, leading to successful navigation for as long as the battery lasted.

The experiments show that even tiny robots can navigate autonomously. Of course, the strategy studied in this chapter is a route-following strategy. This sets it clearly apart from SLAM-based navigation approaches and has two important implications. Firstly, the robot will perform an inbound route that is identical to the outbound route. This is similar to other teach-and-repeat methods in robotics [37, 52]. By not going straight back to the starting location as insects are known to do (see, e.g., [18]), the robot follows a trajectory that is suboptimal in terms of path length. In contrast, robots performing metric SLAM

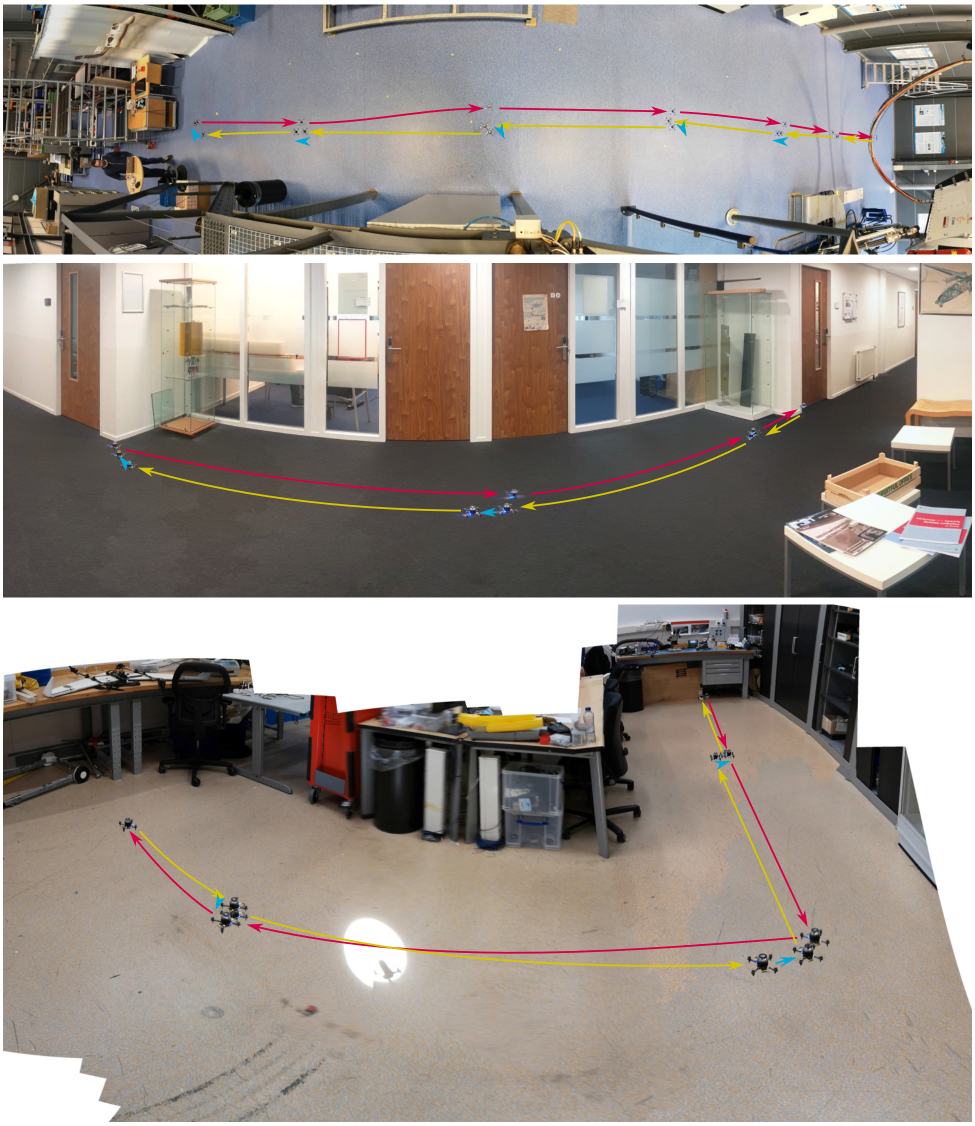


Figure 4.7: Flights in different indoor environments. In order to test the robustness of the proposed visual navigation method, we performed tests in various indoor environments. A time lapse image of each test is shown for three environments, from top to bottom: a large indoor test facility for airplane simulation (SIMONA), an indoor office hallway, and the Micro Air Vehicle lab space (the bright spot on the floor is due to direct, bright sunlight). The outbound trajectory is indicated by red arrows, odometry-based indoor trajectories by yellow arrows, and homing maneuvers by blue arrows.

can plan and execute optimal paths at the expense of considerable processing. How to perform straight returns to the initial location while coping with odometric drift without metric maps is an important topic of future study. Secondly, the current algorithm is unable to combine multiple paths to move between arbitrary locations in the environment. It will not be able to fly to another previously visited place that is relevant to its task, without first returning to the home location. Also here, insects are able to do this, and it is hypothesized that insects may store different places in a nest-centered reference frame [19]. Such a representation is reminiscent of topological mapping approaches, which may provide fertile ground for cross-fertilization with the method proposed in this chapter. For example, in [53] new nodes (places) are created in a topological map by estimating online whether the robot is about to leave the catchment area of the previous node. This leads to overlapping catchment areas, an idea very similar to that of [33] in the insect route-following literature. The proposed idea of further spacing snapshots (or nodes in the topological graph) apart is directly relevant to such a topological mapping approach.

In terms of biological plausibility, we do not believe that the proposed strategy is an accurate model for explaining insect navigation behavior. While some switching between path integration and visual homing occurs when insects move from unknown to known environments; behavioral experiments show that they mostly use these cues simultaneously [54, 55] in contrast to our strategy. However, our results do support the general idea that path integration and visual homing are best used in combination. Even more, they suggest that path integration has a dramatic impact on the efficiency and parsimony of navigation even when other (visual) cues are present, and encourages further research into the integration of these cues cf. [19, 55, 56].

Concerning the impact on robotics, the performed experiments are highly encouraging as they show that also tiny robots are able to perform vision-based autonomous navigation. Future work could focus on improved robustness by introducing obstacle avoidance capabilities. For instance, the omnidirectional image could be used to determine optical flow for collision avoidance [57, 58]. Moreover, the robot could estimate catchment area size online and be endowed with a search procedure when losing the route. Additional experiments in varying environments could identify which elements of the method need most improvement. Still, as already hinted above, the choice for route-following implies a cost in terms of tracking accuracy and flexibility in navigation targets. Even if we further approach the impressive navigation capabilities of insects, it may be that on these characteristics a traditional SLAM-based navigation approach remains superior. However, for autonomously navigating robots optimality in terms of mass and energy expenditure are also important. This is definitely the case for applications such as greenhouse monitoring by flying robots. The driving factors for that application are safety and navigation in narrow, cluttered environments. Tiny, light-weight flying robots are hence ideal for such an application, in which it is most important that the robots fly out, gather data, and come back to a fixed charging station. The data can then be uploaded to a server for mission-specific processing, such as evaluating crop growth or disease detection. We expect that the best navigation solution will eventually be task-dependent, and venture that even for much bigger robots in the order of kilograms, insect-inspired navigation may be the best option when efficiency is more important than high positioning accuracy along the trajectory. If orders of magnitude in computation and memory can be saved from the task of navigation, this computation can be used for onboard mission-relevant tasks such as the recognition of diseases or pests in a greenhouse application or the counting of products in a warehouse monitoring application. Hence, the current work will not only benefit tiny robots such as the 56-gram drone used in this chapter or even the insect-sized Harvard Robobee [59], but much larger robots as well.

## 4.4. Materials and methods

### 4.4.1. Hardware

The experiments in this chapter are performed on a prototype of the Crazyflie Brushless drone provided by Bitcraze. The Flowdeck V2 (PMW3901 optical flow sensor, VL53L1x laser ranger) is used for velocity and altitude control and odometry. For navigation, the drone is equipped with a TCM8230MD camera with a Kogeto Dot 360 panoramic lens. Processing is performed using two STM32F4 microcontrollers, one on the autopilot and one on the camera assembly. Visual processing is performed on the camera micro-controller; state estimation and control are performed on the autopilot. Logging is performed off-board using the radio link.

The default Crazyflie firmware is used as autopilot. State estimation is performed with the default Extended Kalman Filter. A custom on-board app communicates with the camera over a UART link and sends position setpoints and measurement updates to the autopilot's controller and estimator.

### 4.4.2. Image processing

Visual navigation begins with the pre-processing of the camera frames. Raw images are captured at a 128x96 pixel resolution (Fig. 4.2B). A custom auto-exposure routine adjusts the shutter time to keep the horizon's mean luma at a fixed value (80 out of 255) while ignoring the rest of the image (e.g. the lens fixture).

The image is then reprojected to cylindrical coordinates at a 128x16 pixel resolution. We use a look-up table and nearest-neighbor sampling for computational efficiency. The cylindrical images are aligned with the drone's North estimate by offsetting the sampling angle. Derotation of pitch and roll angles was implemented but not used, as the angles during the experiment remained sufficiently small. The images are then converted to grayscale and vertically averaged to produce a one-dimensional periodic signal. We use the Fast Fourier Transform from the ARM CMSIS DSP Library to transform this signal to the frequency domain. For memory efficiency, the DC- and higher-frequency components are dropped. The remaining complex coefficients are quantized to pairs of 8-bit signed integers, with a fixed per-frequency scaling to cover most of the 8-bit range.

### 4.4.3. Homing implementation

For the comparison of homing methods, we implemented the algorithms by Franz et al. [30] ('Search'), Möller et al. [48] ('MFDID') and Stürzl et al. [31] ('Fourier') as described in the respective papers. For each choice of snapshot size, the parameters were re-tuned using a grid search to maximize the size of the catchment area. For the Search algorithm, we used a bearing-distance search grid, as the exact grid was not described in the article. For MFDID, we included the use of the Newton-based correction [60] as part of the parameter search, but found little difference in this dataset which mainly consists of a square, open room.

The Fourier-based homing algorithm by Stürzl et al. [31] was also implemented on the experimental hardware. For an efficient implementation, we wish to highlight an important property of this algorithm. To find the homing vector, Stürzl et al. derive and minimize an 'approximate IDF' $\mathcal{E}_2$ in terms of hypothetical movement $\mathbf{h} \in \mathbb{R}^3$ in the form of a quadratic surface:

$$\mathcal{E}_2(\mathbf{h}) = \frac{1}{2}\mathbf{h}^\top \mathrm{A}\mathbf{h} + \mathbf{b}^\top \mathbf{h} + c$$

Here, A, $\mathbf{b}$ and $c$ are fully defined by the complex coefficients of the Fourier-transformed, derotated images. As a result, the homing vector can be found using only the Fast Fourier Transform and a $3 \times 3$ matrix inversion, which makes this algorithm highly efficient in terms of runtime. For the full definition, we refer to [31].

While the images are already coarsely aligned with respect to the north estimate, we did include the coarse rotation alignment step of the algorithm. However, we have replaced the phase-based algorithm with a brute-force search over all possible rotations, as we found that this provided more robust results in practice. The phase-based algorithm appeared to lack robustness when symmetries were present in the environment or when lower frequencies were absent in the panoramic images, though we did not fully investigate this further.

For the sequencing of snapshots, it is important to detect arrival after homing. This was initially determined by observing the difference between the currently relevant snapshot and the current observation. If this difference did not reach a new minimum during the last 10 frames ($\sim$1 second), the drone was considered to have arrived. While this worked, it resulted in long hover times near the snapshots. In the final experiments, this detection was replaced by a simple timeout. This significantly reduced hovering times and thereby allowed longer travel distances, at the cost of a slightly higher homing position error.

### 4.4.4. Route memory

The 'route memory', containing the snapshots and odometry vectors, is implemented in memory using two stacks (Fig.4.8). The odometric trajectory is recorded as a sequence of translation vectors. While recording, a new vector is pushed onto the stack every 0.3m. These vectors are stored as a pair of 8-bit signed integers, with a resolution of 10cm. Each new vector is calculated by comparing the current position estimate to the sum of all previous vectors, this prevents the buildup of rounding errors. To reduce memory consumption, the recorded trajectory is simplified when a new snapshot is taken. At that time, the complete trajectory since the last snapshot is decimated using the Ramer-Douglas-Peucker algorithm [61, 62] ($\epsilon = 0.1$m). This strongly reduces the number of odometry vectors, while keeping the resulting deviation within strict bounds. Additional vectors may be remembered to keep the lengths within 8-bit integer bounds. After simplifying the trajectory, the snapshot is pushed onto its own stack together with the size of the odometry stack at that time, so that its position in the odometry frame can be retrieved during route following.

During route following, after each odometry segment the drone should be close to the true position of the snapshot on top of the stack (Fig. 4.8). The drone will then start a homing maneuver towards this snapshot, removing the need for along-route localization. Once

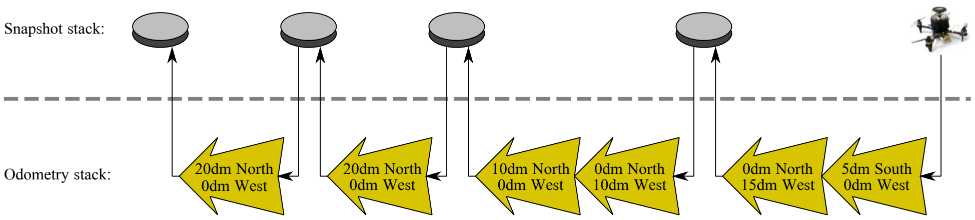


Figure 4.8: Route representation in memory. The trajectory is represented using two stacks: one holding snapshots and one holding odometry vectors. When a new snapshot is pushed, it is stored together with the number of odometry vectors that were present at that time. This allows the snapshot position to be found by adding all odometry vectors before it and aids in the sequencing of odometry- and homing maneuvers. A single odometry vector consists of two `int8` numbers at decimeter resolution (two bytes). A snapshot consists of 8 complex coefficients (2x `int8`, so 16 bytes in total) plus a `uint16` odometry count (2 bytes).

the homing maneuver has completed, the drone re-aligns its position and heading estimate to those of the snapshot as recorded by the odometry vectors and the north alignment of the snapshot. In practice, this re-alignment is implemented as an absolute position and heading measurement with a small covariance, for easier integration with the Kalman filter.

### 4.4.5. Experimental setup

During the flight experiments in Figures 4.4-4.6, the true position of the drone is measured using an OptiTrack motion capture system. This position data is not communicated to the drone at any time; it is only recorded for analysis after the flight. The timestamps are aligned by maximizing the cross-correlation between the north or east positions in both log files. The outbound trajectories during the flight experiments do not use the true position either; these rely entirely on the onboard position estimate.

The experimental flights shown in Figures 4.4-4.6 are performed over artificial grass. Judging by the performance of odometry-only navigation, this provides sufficient texture for the Crazyflie's optical flow sensor. The drone uses its downward-facing laser range sensor to maintain a constant height during the experiment. On the sides of the flight area, canvas panels with mostly natural scenes provide additional texture for navigation. For the U- and S-shaped trajectories, these panels were also placed in the center of the flight area to block the line of sight between the extreme points.

The experimental flights shown in Figure 4.7 are performed at different locations in the Faculty of Aerospace Engineering: In a large open space of the SIMONA airplane simulator, a narrow corridor, and in the Micro Air Vehicle lab. Each experiment starts with a manually designed, preprogrammed outbound flight that is executed based on odometry. The outbound flight is followed by an autonomous inbound flight. Figure 4.7 contains stitched time lapse images, in which we show the drone when making the snapshot during the outbound flight (red arrows), and when the drone thinks to arrive at the snapshot location with odometry during the inbound flight (yellow arrows), and after homing to the snapshot (blue arrows). The videos can also be found as supplementary material.

### 4.4.6. Simulation setup

Figure 4.5A shows the results of elementary simulation experiments to illustrate the effects of odometry drift. The simulation includes both an initial position and heading offset due to imperfect homing and odometry drift along the route leg. Hence, we initialize the pose as:

$$x,\ y,\ \psi \leftarrow x_0 + \mathcal{N}(\sigma_{x_0}),\ y_0 + \mathcal{N}(\sigma_{y_0}),\ \psi_0 + \mathcal{N}(\sigma_{\psi_0})\ ,$$

where $(x_0,\ y_0,\ \psi_0)$ is the actual snapshot position and $\mathcal{N}(\sigma)$ is a normally-distributed random variable with zero mean and standard deviation $\sigma$. Then, for each timestep, the state is updated as follows:

$$\begin{aligned}
\psi &\leftarrow \psi + \mathcal{N}(\sigma_\psi) \\
x &\leftarrow x + \Delta x \cos\psi + \mathcal{N}(\sigma_x) \\
y &\leftarrow y + \Delta x \sin\psi + \mathcal{N}(\sigma_y)
\end{aligned}$$

For the simulation in Figure 4.5A, the following values were used: $\Delta x = 0.25$m, $\sigma_{x_0} = \sigma_{y_0} = 0.10$m, $\psi_0 = 5^\circ$, $\sigma_x = \sigma_y = 0.025$m, and $\sigma_\psi = 2^\circ$. Please note that these values are large compared to the real homing and drift errors, so that the figure clearly shows the effects they have on the position error at the end of the route leg.

**Acknowledgements** We would like to thank Tobias Antonsson of Bitcraze AB for letting us use their Crazyflie Brushless prototype drone, which made the experiments possible due to the increased payload capabilities. We would like to thank Kimberly McGuire, Pascual Campoy and Pieter Jonker for their guidance and inputs during the preparatory study, which was performed in the context of TvD's MSc thesis.

## References

[1] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of unmanned aerial vehicles*, Vol. 1 (Springer, 2015).

[2] 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.

[3] S. B. Fuller, *Four wings: An insect-sized aerial robot with steering ability and payload capacity for autonomy,* IEEE Robotics and Automation Letters **4**, 570 (2019).

[4] T. Raj, F. Hanim Hashim, A. Baseri Huddin, M. F. Ibrahim, and A. Hussain, *A survey on lidar scanning mechanisms,* Electronics **9**, 741 (2020).

[5] S. Balasubramanian, Y. M. Chukewad, J. M. James, G. L. Barrows, and S. B. Fuller, *An insect-sized robot that uses a custom-built onboard camera and a neural network to classify and respond to visual input,* in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)* (IEEE, 2018) pp. 1297–1302.

[6] S. Viollet, S. Godiot, R. Leitel, W. Buss, P. Breugnon, M. Menouni, R. Juston, F. Expert, F. Colonnier, G. L'Eplattenier, *et al.*, *Hardware architecture and cutting-edge assembly process of a tiny curved compound eye,* Sensors **14**, 21702 (2014).

[7] H. Durrant-Whyte and T. Bailey, *Simultaneous localization and mapping: part i,* IEEE robotics & automation magazine **13**, 99 (2006).

[8] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Luján, S. Furber, A. J. Davison, *et al.*, *Slambench2: Multi-objective head-to-head benchmarking for visual slam,* in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018) pp. 3637–3644.

[9] J. Boal, A. Sánchez-Miralles, and A. Arranz, *Topological simultaneous localization and mapping: a survey,* Robotica **32**, 803 (2014).

[10] E. Garcia-Fidalgo and A. Ortiz, *Vision-based topological mapping and localization methods: A survey,* Robotics and Autonomous Systems **64**, 1 (2015).

[11] S. Hussaini, M. Milford, and T. Fischer, *Spiking neural networks for visual place recognition via weighted neuronal assignments,* IEEE Robotics and Automation Letters **7**, 4094 (2022).

[12] B. Ferrarini, M. Milford, K. D. McDonald-Maier, and S. Ehsan, *Highly-efficient binary neural networks for visual place recognition,* in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2022) pp. 5493–5500.

[13] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, *Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones,* IEEE Journal of Solid-State Circuits **54**, 1106 (2019).

[14] J. Shalf, *The future of computing beyond moore's law,* Philosophical Transactions of the Royal Society A **378**, 20190061 (2020).

[15] Y. Sun and A. M. Kist, *Deep learning on edge tpus,* arXiv preprint arXiv:2108.13732 (2021).

[16] N. Tiwari and K. Mondal, *Ncs based ultra low power optimized machine learning techniques for image classification,* in *2019 IEEE Region 10 Symposium (TENSYMP)* (IEEE, 2019) pp. 750–753.

[17] S. Li, E. van der Horst, P. Duernay, C. De Wagter, and G. C. de Croon, *Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone,* Journal of Field Robotics **37**, 667 (2020).

[18] R. Wehner, M. V. Srinivasan, *et al.*, *Path integration in insects,* The neurobiology of spatial behaviour , 9 (2003).

[19] B. Webb, *The internal maps of insects,* Journal of Experimental Biology **222**, jeb188094 (2019).

[20] B. Ronacher and R. Wehner, *Desert ants cataglyphis fortis use self-induced optic flow to measure distances travelled,* Journal of Comparative Physiology A **177**, 21 (1995).

[21] S. S. Kim, H. Rouault, S. Druckmann, and V. Jayaraman, *Ring attractor dynamics in the drosophila central brain,* Science **356**, 849 (2017).

[22] T. Stone, B. Webb, A. Adden, N. B. Weddig, A. Honkanen, R. Templin, W. Wcislo, L. Scimeca, E. Warrant, and S. Heinze, *An anatomically constrained model for path integration in the bee brain,* Current Biology **27**, 3069 (2017).

[23] J. Green, A. Adachi, K. K. Shah, J. D. Hirokawa, P. S. Magani, and G. Maimon, *A neural circuit architecture for angular integration in drosophila,* Nature **546**, 101 (2017).

[24] B. Cartwright and T. Collett, *How honey bees use landmarks to guide their return to a food source,* Nature **295**, 560 (1982).

[25] B. Baddeley, P. Graham, P. Husbands, and A. Philippides, *A model of ant route navigation driven by scene familiarity,* PLoS computational biology **8**, e1002336 (2012).

[26] F. Le Möel and A. Wystrach, *Opponent processes in visual memories: A model of attraction and repulsion in navigating insects' mushroom bodies,* PLoS computational biology **16**, e1007631 (2020).

[27] A. Denuelle and M. V. Srinivasan, *Bio-inspired visual guidance: From insect homing to uas navigation,* in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (IEEE, 2015) pp. 326–332.

[28] J. Dupeyroux, J. R. Serres, and S. Viollet, *Antbot: A six-legged walking robot able to home like desert ants in outdoor environments,* Science Robotics **4**, eaau0307 (2019).

[29] D. Lambrinos, H. Kobayashi, R. Pfeifer, M. Maris, T. Labhart, and R. Wehner, *An autonomous agent navigating with a polarized light compass,* Adaptive behavior **6**, 131 (1997).

[30] M. O. Franz, B. Schölkopf, H. A. Mallot, and H. H. Bülthoff, *Where did i take that snapshot? scene-based homing by image matching,* Biological Cybernetics **79**, 191 (1998).

[31] W. Stürzl and H. A. Mallot, *Efficient visual homing based on fourier transformed panoramic images,* Robotics and Autonomous Systems **54**, 300 (2006).

[32] A. Denuelle and M. V. Srinivasan, *A sparse snapshot-based navigation strategy for uas guidance in natural environments,* in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2016) pp. 3455–3462.

[33] A. Vardy, *Long-range visual homing,* in *2006 IEEE International Conference on Robotics and Biomimetics* (IEEE, 2006) pp. 220–226.

[34] S. Raj, P. R. Giordano, and F. Chaumette, *Appearance-based indoor navigation by ibvs using mutual information,* in *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)* (IEEE, 2016) pp. 1–6.

[35] J. Courbon, G. Blanc, Y. Mezouar, and P. Martinet, *Navigation of a non-holonomic mobile robot with a memory of omnidirectional images,* in *ICRA 2007 Workshop on" Planning, perception and navigation for Intelligent Vehicles* (2007).

[36] T. Krajník, J. Faigl, V. Vonásek, K. Košnar, M. Kulich, and L. Přeučil, *Simple yet stable bearing-only navigation,* Journal of Field Robotics **27**, 511 (2010).

[37] D. Dall'Osto, T. Fischer, and M. Milford, *Fast and robust bio-inspired teach and repeat navigation,* in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2021) pp. 500–507.

[38] A. M. Zhang and L. Kleeman, *Robust appearance based visual route following for navigation in large-scale outdoor environments,* The International Journal of Robotics Research **28**, 331 (2009).

[39] T. Krajník, F. Majer, L. Halodová, and T. Vintr, *Navigation without localisation: reliable teach and repeat based on the convergence theorem,* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018) pp. 1657–1664.

[40] F. Labrosse, *Short and long-range visual navigation using warped panoramic images,* Robotics and Autonomous Systems **55**, 675 (2007).

[41] F. Chaumette and S. Hutchinson, *Visual servo control. i. basic approaches,* IEEE Robotics & Automation Magazine **13**, 82 (2006).

[42] M. Liu, C. Pradalier, Q. Chen, and R. Siegwart, *A bearing-only 2d/3d-homing method under a visual servoing framework,* in *2010 IEEE International Conference on Robotics and Automation* (IEEE, 2010) pp. 4062–4067.

[43] F. Chaumette, S. Hutchinson, and P. Corke, *Visual servoing,* Springer handbook of robotics , 841 (2016).

[44] D. G. Lowe, *Object recognition from local scale-invariant features,* in *Proceedings of the seventh IEEE international conference on computer vision*, Vol. 2 (Ieee, 1999) pp. 1150–1157.

[45] S. Leutenegger, M. Chli, and R. Y. Siegwart, *Brisk: Binary robust invariant scalable keypoints,* in *2011 International conference on computer vision* (Ieee, 2011) pp. 2548–2555.

[46] A. Stelzer, M. Vayugundla, E. Mair, M. Suppa, and W. Burgard, *Towards efficient and scalable visual homing,* The International Journal of Robotics Research **37**, 225 (2018).

[47] J. Zeil, N. Boeddeker, and W. Stürzl, *Visual homing in insects and robots,* Flying insects and robots , 87 (2010).

[48] R. Möller and A. Vardy, *Local visual homing by matched-filter descent in image distances,* Biological cybernetics **95**, 413 (2006).

[49] D. D. Gaffin and B. P. Brayfield, *Autonomous visual navigation of an indoor environment using a parsimonious, insect inspired familiarity algorithm,* Plos one **11**, e0153706 (2016).

[50] S. Edvani, E. va, S. Bettendorf, S. Edvani, E. va, and S. Bettendorf, *Simona-a reconfigurable and versatile research facility,* in *Modeling and Simulation Technologies Conference* (1997) p. 3809.

[51] S. Shah, D. Dey, C. Lovett, and A. Kapoor, *Airsim: High-fidelity visual and physical simulation for autonomous vehicles,* in *Field and Service Robotics: Results of the 11th International Conference* (Springer, 2018) pp. 621–635.

[52] P. Furgale and T. D. Barfoot, *Visual teach and repeat for long-range rover autonomy,* Journal of field robotics **27**, 534 (2010).

[53] M. A. Khan and F. Labrosse, *Visual topological mapping using an appearance-based location selection method,* in *Annual Conference Towards Autonomous Robotic Systems* (Springer, 2020) pp. 90–102.

[54] C. Buehlmann, M. Mangan, and P. Graham, *Multimodal interactions in insect navigation,* Animal cognition **23**, 1129 (2020).

[55] X. Sun, S. Yue, and M. Mangan, *A decentralised neural model explaining optimal integration of navigational strategies in insects,* Elife **9**, e54026 (2020).

[56] X. Sun, S. Yue, and M. Mangan, *How the insect central complex could coordinate multimodal navigation,* Elife **10**, e73077 (2021).

[57] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, *Mav navigation through indoor corridors using optical flow,* in *2010 IEEE International Conference on Robotics and Automation* (IEEE, 2010) pp. 3361–3368.

[58] J. R. Serres and F. Ruffier, *Optic flow-based collision-free strategies: From insects to robots,* Arthropod structure & development **46**, 703 (2017).

[59] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood, *Controlled flight of a biologically inspired, insect-scale robot,* Science **340**, 603 (2013).

[60] R. Möller, A. Vardy, S. Kreft, and S. Ruwisch, *Visual homing in environments with anisotropic landmark distribution,* Autonomous Robots **23**, 231 (2007).

[61] U. Ramer, *An iterative procedure for the polygonal approximation of plane curves,* Computer graphics and image processing **1**, 244 (1972).

[62] D. H. Douglas and T. K. Peucker, *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,* Cartographica: the international journal for geographic information and geovisualization **10**, 112 (1973).

# 5

# Conclusion

In Chapter 1, we set out to find a visual navigation strategy that is suitable for tiny drones. Now, at the end of this dissertation, we can reflect back upon the previous chapters to see if they indeed made this possible. I will start by discussing the individual research questions, and end this chapter with a conclusion about visual navigation on tiny drones and an outlook to further research.

## 5.1. Perception

In Chapter 1, the problem of visual navigation was split into three separate problems: perception, avoidance and route following. Using perception, the drone should be able to detect the presence of obstacles, and estimate their location so a route can be followed around it. While bigger drones can use systems like LIDAR, the options for tiny drones are limited. As discussed in Chapter 2, visual perception provides a great trade-off of information versus weight, but the processing of this information on tiny systems remains a challenge. Perceiving obstacles in such a way that the algorithm can be implemented on a tiny drone is the core of the first research question:

**RQ 1**

How can tiny drones perceive obstacles?

In Chapter 2, visual perception was split into further categories: stereoscopic depth estimation, monocular depth estimation, and depth estimation using optical flow. Optical flow was the first option to be deemed infeasible. While it has the advantage that only a single camera is required, it has the fundamental flaw that its estimates become inaccurate near the focus-of-expansion, which by definition is the exact direction in which the drone is flying!

Stereo vision does not have this problem. Secondly, it is the simplest of the three classes of depth estimation, as it is mostly a geometrical problem. The main challenge in stereo vision is to find correspondences between the left- and right images. While this is still not a simple, light-weight process, many algorithms are available that do not rely on deep learning (e.g. [1]) and the concept has also already been demonstrated on microcontrollers [2, 3]. The disadvantage of stereo vision is probably obvious: it requires two cameras. Additionally, the baseline between the cameras needs to be sufficiently large to see obstacles further away. While this is still possible on the multiple-centimeters-scale drones in this dissertation, it will eventually limit the usability on insect-size drones such as [4].

Finally, (single-frame) monocular vision has the advantage that it also works in the direction of movement, but only requires a single camera (which means that the baseline is not a concern anymore). The downside of monocular vision is that it requires significant amounts of processing, as unlike the previous methods its estimate relies primarily on the *content* of the images rather than a ‘simpler’ geometrical problem. Or does it?

Because of the potential advantages of monocular depth estimation, Chapter 3 took a closer look at these algorithms. While many implementations exist, there had so far not been an explanation of how they actually function. Using black-box experimental methods, partly inspired by human psychology, it was found that the evaluated networks did not learn to recognize familiar objects or reason about their size. Instead, the single cue of vertical position in the image together with contrasting outlines of the obstacle, was more-or-less enough to estimate its shape and distance. While this does not directly result in a lighter algorithm that is suitable for microcontrollers, it at least strongly suggests that such an algorithm is possible. If I had the time to write one more paper during my PhD studies, I would have loved to tackle this problem. But for now, the answer to Research Question 1 will be that **light-weight monocular vision can provide obstacle perception for tiny drones, but a practical demonstration remains future work.**

While the article provides a good starting point for light-weight monocular depth estimation, it also raised some new questions. If the neural networks use the vertical position in the image to determine the distance to obstacles, how then does it respond to changes in camera pose? This question has been answered in detail by Benjamin Keltjens [5], one of the MSc students that I supervised. Keltjens found that the evaluated neural networks can learn to cope with changes in camera pose if they are trained on a representative dataset. (How exactly they do that remains open for investigation). Additionally, said paper focused on self-supervised learning for depth estimation in indoor environments, where a lack of texture leads to poor gradients in reconstruction-based loss functions. By extending the loss function with an additional supervisory signal based on depth interpolation between reliable edges, the performance in indoor environments can be strongly improved irrespective of the underlying neural network.

As mentioned above, stereo vision is still a viable option for small drones with spans in the order of a few centimeters. Stereo vision is expected to provide better results at close range because of its geometrical underpinnings, though its performance will likely drop below that of monocular depth estimation at longer ranges. Two MSc students have looked at the problem of fusing these estimates to obtain an optimal result. Alexios Lyrakis developed a method to estimate the certainty of the depth estimates from stereo vision [6]. Using this information, a better weighing between the two estimates can be made. Additionally, the certainty can be taken into account when planning new waypoints to avoid obstacles [6]. Dani Tóth's work focused on the fusion of monocular- and stereoscopic depth estimates through a neural network, with the goal to achieve a higher accuracy than either method in isolation. At the time of writing, this work is still in progress.

Besides the works above, my own investigation into monocular depth estimation has also been well picked up by the scientific community. The problems raised by the paper are recognized and steps are taken to improve upon this. For example, [7–9] have proposed improved methods for data augmentation. And in [10] the authors explicitly try to weaken the dependence on the vertical position cue. In contrast, [11] try to *increase* the dependence on this cue, recognizing that it provides a good estimate if the conditions are right. Other works try to keep the vertical position cue, but address the dependence on camera pose by supplementing the missing camera pose information. In [8, 12, 13], the camera pose is provided as an extra input to the network. Alternatively, [14] estimates the ground plane in the input image, while [15] finds the horizon and vanishing points, with the advantage that neither requires an external measurement of the camera pose.

## 5.2. Avoidance

Once obstacles are detected, the drone needs to take some action to avoid them. At the same time, the drone should continue to make progress towards its target location. This lead to the second research question:

**RQ 2**

How can tiny drones avoid obstacles while moving towards a specific point?

Chapter 2 gave an overview of avoidance strategies and the maps that they could use. Ideally though, to minimize memory consumption, avoidance without any map would be the most efficient strategy. Such a solution already exists in the form of bug algorithms. My former colleague and MSc-supervisor Kimberly McGuire has already written an extensive review on this topic [16] and demonstrated its use in a real-world experiment on tiny drones [17]. The limitation of the existing works, however, is that they mainly focus on planar, two-dimensional movement. The few three-dimensional methods that exist (e.g. [18, 19]) assume omnidirectional vision, which may be difficult to implement within the weight constraints of tiny drones.

The latter half of Chapter 2 discusses initial flight tests with a Parrot Bebop 2 and SLAMDunk. After these initial flight tests, a very basic bug-inspired avoidance algorithm was implemented on this drone. It was closely related to TangentBug [20], but extended to three dimensions by looking for solutions on three different planes (climbing, level, descending). The algorithm was demonstrated at the International Micro Air Vehicle conference and competition in 2018, but did not result in an article.

The astute reader will have noticed that there is no further chapter on this topic. The reason is that the area of three-dimensional bug algorithms was further investigated by MSc students that I have supervised: Alexios Lyrakis, Ralph Schmidt and Ruben Meester, while I was working on route following. Lyrakis' work primarily focused on the uncertainty estimation for stereoscopic depth estimates, but includes the selection of possible escape points under uncertainty when a potential collision is detected [6]. Schmidt worked on an extension of the WedgeBug algorithm [22] to three dimensions, demonstrating its use in simulation. Finally, Ruben Meester developed an extensive state-machine-based algorithm that performed autonomous flights on a drone with a forward-facing stereo camera [23]. The algorithm 'FrustrumBug' was evaluated in simulation in diverse realistic environments (city, forest) and achieved a success rate of more than 90% (Fig. 5.1a). Additionally, the algorithm was implemented on a real-world stereo camera powered by a Jevois smart machine vision processor[1], where it was able to run in real-time. Preliminary flight tests with this system were successfully performed in the TU Delft Cyberzoo (Fig. 5.1b). This work has been accepted and presented at the International Micro Air Vehicle conference and competition, 2023 [21]. Based on the work of these thesis students and especially the results by Meester, the answer to the second research question is therefore that **bug algorithms are effective in three-dimensional, limited-field-of-view applications and provide a lightweight goal-oriented avoidance strategy that is suitable for tiny drones**.

[1]jevois.org

a)

b)

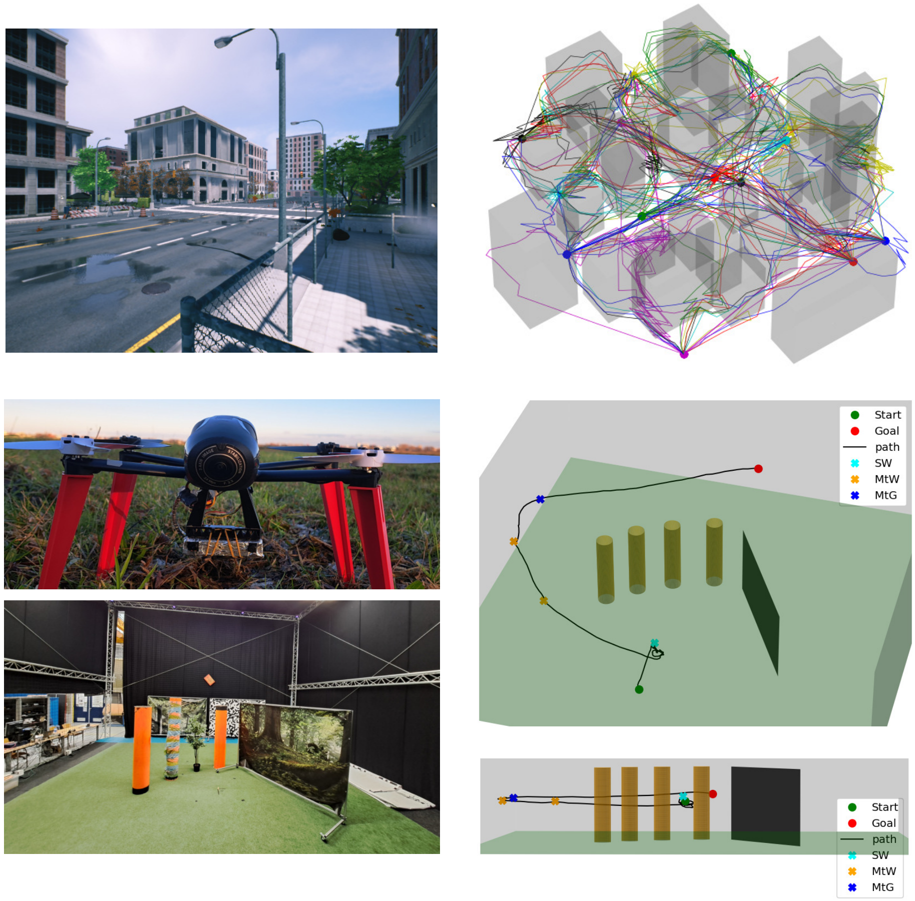


Figure 5.1: Flight results from Ruben Meester's FrustrumBug [21]. a) Flight trajectories in the simulated UrbanCity environment. Out of the 182 flights, 177 successfully reach their goal. In the remaining five flights, the drone either got stuck in repetitive behavior (4x) or collided with a building (1x). b) Real-world flight result in the TU Delft Cyberzoo. Image of the drone and environment, plus the real-world trajectory plotted in a schematic overview of the test environment. The drone can successfully find its way to the goal position despite the obstacles and the local minimum that they form. Images reproduced with permission.

## 5.3. Route following

If an environment is traversed for a longer time, it becomes more efficient to follow known, obstacle-free paths than to search for a new route every time the drone needs to move between locations. On larger-scale drones, building a map of the environment using Simultaneous Localization and Mapping is the most common solution to this problem, but this is too complex for tiny drones. Because of the limited processing and memory capabilities, an alternative strategy is required:

**RQ 3**

How can tiny drones retrace known paths?

In Chapter 4 an alternative strategy is proposed. Rather than building a geometrically accurate map of the environment, the drone will remember just enough to reset its odometric error from time to time. The chapter shows that the deviation from the path remains bounded, and that long routes can be remembered using only a tiny amount of memory. In fact, the traveled distance was bounded by the battery capacity rather than by memory limitations.

The proposed method is so far limited to planar movement. Applicability to three-dimensional routes still needs to be proven. However, the general idea of using visual homing to correct odometry drift is sufficiently demonstrated in Chapter 4. Secondly, other work has already shown that the image difference function and catchment areas are also applicable in three dimensions [24]. The main challenge then, would be to find a snapshot representation and visual homing algorithm that are as efficient as the Fourier-based algorithm by Stürzl and Mallot for planar visual homing [25]. And while it would still need some work to implement on a real drone, recent work by Differt and Stürzl proposes a three-dimensional visual homing algorithm with support for highly compressed snapshots [26]. Therefore, I can say with reasonable confidence that **tiny drones can retrace known paths by combining odometry with periodic homing maneuvers to counteract drift**.

Because there was little time between the publication of my article and the finalization of this thesis, it is not possible to comment on its impact on the field. However, I hope it will be seen as encouragement to look more towards behavior-based solutions for robotics than more-and-more complex modeling and planning algorithms.

## 5.4. Navigation

With all research questions answered, we end up at the main goal of this dissertation: bringing visual navigation to tiny drones.

**Main Research Question**

How do we bring visual navigation to tiny drones?

Because all research questions provided a positive answer, it is now possible to propose a strategy for visual navigation on tiny drones where this was previously not possible due to size, weight and power constraints. **Tiny drones can visually navigate by us-**

**ing lightweight monocular vision algorithms to perceive obstacles, three-dimensional bug algorithms to avoid them while moving to new locations, and odometry and visual homing to retrace known paths**.

## 5.5. Outlook

With the main research question answered, what work remains for future research? On a high level, there is still room to investigate the strategies proposed in this thesis. These can be tested individually, for instance by evaluating the simulation results in the real world, or extending the test flights to varied real-world environments. Or, the integration of all strategies proposed here could be investigated, since they have only been tested in isolation.

More into detail, the results in this thesis are primarily proof-of-concepts, and robust implementations still need to be developed. For monocular depth estimation, a likely strategy has been identified, but still needs to be implemented and tested on lightweight hardware. There are also a few key items that remain for visual route following. Firstly, the current strategy has been demonstrated for planar movement. While the steps towards a three-dimensional solution are clearly described above, they still need to be implemented and evaluated. Secondly, the current strategy relies on panoramic- or omnidirectional vision. A solution with a forward-facing camera would save more weight for the smallest of drones.

This dissertation covered the autonomous navigation between different workspaces. What was not discussed, is the movement within a workspace, i.e. small, precise movements over a short distance. This is an area where the proposed strategies are less useful, and where further developments in lightweight positioning techniques like Visual Servoing (e.g. [27]) could provide new opportunities.

Finally, at a really high level, this thesis has shown that a complex task like visual navigation can be performed without complex online modeling and/or state estimation. Instead, the problem is solved by the interplay between relatively simple behaviors and the environment. I think a stronger focus on behavior-based solutions like these could provide a huge boost to robots in general, not just tiny ones, without the cost of additional sensors or hardware. I hope that this work might set an example for future research into robot navigation.

## References

[1] H. Hirschmuller, *Stereo processing by semiglobal matching and mutual information,* IEEE Transactions on pattern analysis and machine intelligence **30**, 328 (2007).

[2] C. De Wagter, S. Tijmons, B. D. Remes, and G. C. de Croon, *Autonomous flight of a 20-gram flapping wing mav with a 4-gram onboard stereo vision system,* in *2014 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2014) pp. 4982–4987.

[3] S. Tijmons, G. C. De Croon, B. D. Remes, C. De Wagter, and M. Mulder, *Obstacle avoidance strategy using onboard stereo vision on a flapping wing mav,* IEEE Transactions on Robotics **33**, 858 (2017).

[4] S. B. Fuller, *Four wings: An insect-sized aerial robot with steering ability and payload capacity for autonomy,* IEEE Robotics and Automation Letters **4**, 570 (2019).

[5] B. Keltjens, T. van Dijk, and G. de Croon, *Self-supervised monocular depth estimation of untextured indoor rotated scenes,* arXiv preprint arXiv:2106.12958 (2021).

[6] A. Lyrakis, *Low-cost stereo-based obstacle avoidance for small uavs using uncertainty maps,* (2019).

[7] Q. Lian, B. Ye, R. Xu, W. Yao, and T. Zhang, *Exploring geometric consistency for monocular 3d object detection,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022) pp. 1685–1694.

[8] Y. Zhao, S. Kong, and C. Fowlkes, *Camera pose matters: Improving depth prediction by mitigating pose distribution bias,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021) pp. 15759–15768.

[9] J. L. G. Bello and M. Kim, *Self-supervised deep monocular depth estimation with ambiguity boosting,* IEEE transactions on pattern analysis and machine intelligence **44**, 9131 (2021).

[10] R. Peng, R. Wang, Y. Lai, L. Tang, and Y. Cai, *Excavating the potential capacity of self-supervised monocular depth estimation,* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021) pp. 15560–15569.

[11] D. Kim, W. Ka, P. Ahn, D. Joo, S. Chun, and J. Kim, *Global-local path networks for monocular depth estimation with vertical cutdepth,* arXiv preprint arXiv:2201.07436 (2022).

[12] Y. Lu, X. Ma, L. Yang, T. Zhang, Y. Liu, Q. Chu, J. Yan, and W. Ouyang, *Geometry uncertainty projection network for monocular 3d object detection,* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021) pp. 3111–3121.

[13] Y. Liu, L. Wang, and M. Liu, *Yolostereo3d: A step back to 2d for efficient stereo 3d detection,* in *2021 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2021) pp. 13018–13024.

[14] Y. Zhou, Q. Liu, H. Zhu, Y. Li, S. Chang, and M. Guo, *Mogde: Boosting mobile monocular 3d object detection with ground depth estimation,* Advances in Neural Information Processing Systems **35**, 2033 (2022).

[15] Y. Zhou, Y. He, H. Zhu, C. Wang, H. Li, and Q. Jiang, *Monoef: Extrinsic parameter free monocular 3d object detection,* IEEE Transactions on Pattern Analysis and Machine Intelligence **44**, 10114 (2021).

[16] K. N. McGuire, G. C. de Croon, and K. Tuyls, *A comparative study of bug algorithms for robot navigation,* Robotics and Autonomous Systems **121**, 103261 (2019).

[17] K. McGuire, C. De Wagter, K. Tuyls, H. Kappen, and G. C. de Croon, *Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment,* Science Robotics **4**, eaaw9710 (2019).

[18] I. Kamon, E. Rimon, and E. Rivlin, *Range-sensor based navigation in three dimensions,* in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, Vol. 1 (IEEE, 1999) pp. 163–169.

[19] I. Kamon, E. Rimon, and E. Rivlin, *Range-sensor-based navigation in three-dimensional polyhedral environments,* The International Journal of Robotics Research **20**, 6 (2001).

[20] I. Kamon, E. Rimon, and E. Rivlin, *Tangentbug: A range-sensor-based navigation algorithm,* The International Journal of Robotics Research **17**, 934 (1998).

[21] R. Meester, T. van Dijk, C. D. Wagter, and G. C. de Croon, *Frustumbug: a 3d mapless stereo-vision-based bug algorithm for micro air vehicles,* in *14th annual International Micro Air Vehicle Conference and Competition*, edited by D. Moormann (Aachen, Germany, 2023) pp. 72–84, paper no. IMAV2023-9.

[22] S. L. Laubach and J. W. Burdick, *An autonomous sensor-based path-planner for planetary microrovers,* in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, Vol. 1 (IEEE, 1999) pp. 347–354.

[23] R. Meester, *Frustumbug: a 3d mapless stereo-vision-based bug algorithm for micro air vehicles,* (2023).

[24] T. Murray and J. Zeil, *Quantifying navigational information: the catchment volumes of panoramic snapshots in outdoor scenes,* PloS one **12**, e0187226 (2017).

[25] W. Stürzl and H. A. Mallot, *Efficient visual homing based on fourier transformed panoramic images,* Robotics and Autonomous Systems **54**, 300 (2006).

[26] D. Differt and W. Stürzl, *A generalized multi-snapshot model for 3d homing and route following,* Adaptive Behavior **29**, 531 (2021).

[27] L. Mejias, S. Saripalli, P. Campoy, and G. S. Sukhatme, *Visual servoing of an autonomous helicopter in urban areas using feature tracking,* Journal of Field Robotics **23**, 185 (2006).

# A

# Supplementary materials for Chapter 4: Visual route following

In this chapter, we provide the following supplementary material. First, we present omnidirectional images taken in different indoor and outdoor environments (Section A.1). Subsequently, we explain the experimental setup and results for the simulation experiments (Section A.2). Finally, we introduce a straightforward theoretical model for the spacing of snapshots (Section A.3).

## A.1. Presence of texture in different environments

As stated in Chapter 4, an important characteristic of the chosen snapshot representation (with vertical averaging and Fourier compression) is that it depends on prominent vertical features. Such features are commonly present in both indoor and outdoor environments. To illustrate this, we made a set of snapshots in various places in the building of the Faculty of Aerospace Engineering at TU Delft.

The raw images cannot be stored in-flight on the Crazyflie drone itself. Hence, in order to be able to store the images, we made a setup in which the drone is connected to Raspberry Pi 3 board. This board is mounted on a ground robot, and the tiny Crazyflie quadrotor is mounted on a custom support, so that it is slightly elevated above the ground. It is important to note that the ground robot was not operational and was only used to support the Crazyflie and log the images.

Below, we show snapshots taken at different locations. From left to right we show: An external image showing the environment, the omnidirectional image, and the snapshot at

An edited version of this chapter has been published as supplementary material for:

**van Dijk, T.**, De Wagter, C., de Croon, G.C.H.E. (2024). *Visual route following for tiny autonomous robots.* Science Robotics.

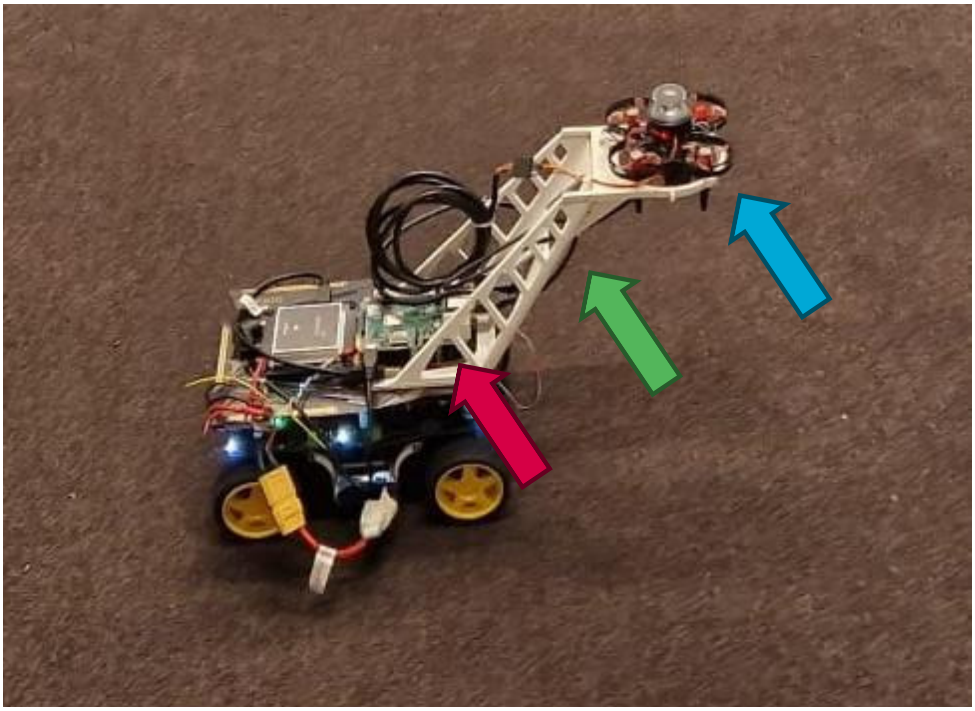


Figure A.1: Setup for storing the images. The Crazyflie drone (blue arrow) is connected to a Raspberry Pi 3 (red arrow) for storing omnidirectional images. The Raspberry Pi 3 is mounted on a ground robot. A custom support (green arrow) elevates the Crazyflie for capturing of the omnidirectional images while keeping the robot outside the field-of-view.

different stages of processing. The stages of processing are, from top to bottom: the cylindrical reprojection of the omnidirectional image, the vertically averaged pixel values, and the reconstruction from the selected Fourier components. Finally, although the approach has been designed for static environments, we also show how the snapshots change when objects change place. For instance, we show a snapshot without and with a person in view, or shift around objects like chairs, plants, or boxes.

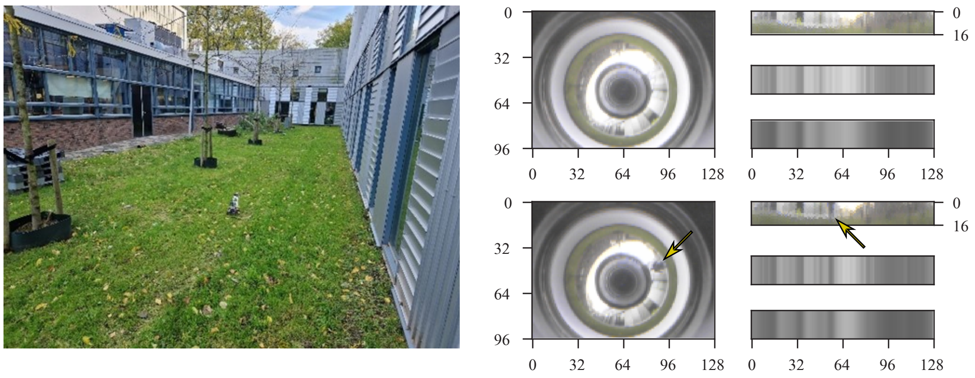
From the above dataset, we make the following observations. First and foremost, in general, there is ample vertical texture both indoors and outdoors, leading to clear contrasts in the snapshot representation. Even in the texture-poor indoor corridor, there are some features that remain, like the blue door and contrast between the floor and walls. Less texture will lead to poorer homing performance. In a completely uniform corridor, homing in the direction of the corridor will be particularly poor. Second, the main layout of the snapshot is determined by large structural elements (walls, windows, doors, trees, corridors, etc.). Dynamic objects do lead to small changes in the snapshot representation. Larger vertical objects (like persons) typically introduce additional vertical "stripes" in the snapshot representation (with broader stripes if they are closer to the robot).

Smaller or more far away objects seem to have little influence on the representation. In this manner, the coarse resolution of the snapshots may bring robustness to small changes in the environment. In order to gain more insight into this matter, we performed a preliminary investigation. Specifically, we first placed the robot at a given location and made a snapshot. Subsequently, we displaced the robot and made a video, during which the experimenters are moving around the scene, also occasionally moving objects around like chairs, large plants, etc. We then ran the vision algorithm on the video, extracting the Fourier components and determining a homing direction to the first snapshot. The effect of the dynamic environments on the homing vectors is shown in Movies S7-S13[1]. The videos are structured the same as the plots in Figure A.2, with the raw camera image on the left, the unwrapped image in the top center, the horizon line in the middle and the snapshot in the bottom center. On the right is the homing vector, in unscaled form (equivalent to an environment radius of 1), with the plot axes ranging between -0.3 and 0.3. The videos contain most of the same scenes as Figure A.2, with people walking in view and moving objects. In general, the homing vector remains quite stable, even though the homing algorithm was made for static scenes. A good example can be seen in Movie S10, where the vector keeps pointing in a similar direction while people are moving in the frame and moving furniture inside the office. However, highly-contrasting moving objects (such as the people in front of the yellow wall in Movie S11) can have a noticeable effect on the homing vector. If the vector still points within 90 degrees to the actual homing vector, this could still lead to successful homing though.

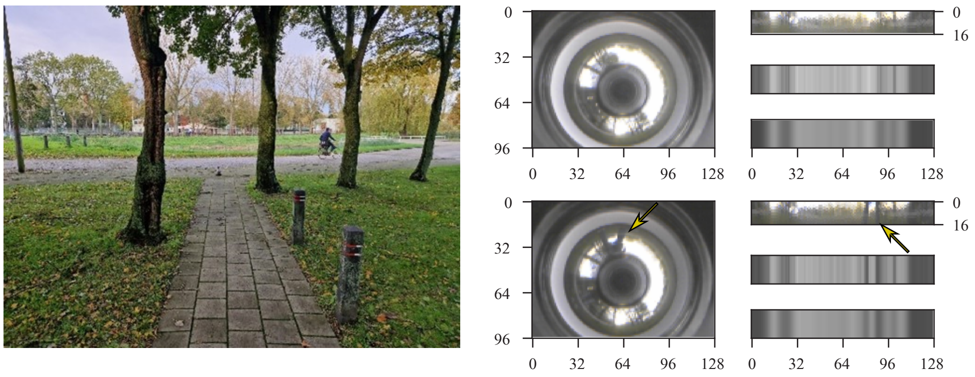
## A.2. Route-following experiments in simulation

While the experiments in Chapter 4 demonstrated the principles behind our navigation strategy, they were ultimately limited in length by the flight time of the drone. In order to evaluate the feasibility and robustness of the strategy over longer distances, we implement the same strategy in simulation. As a simulated environment we select a forest, allowing us

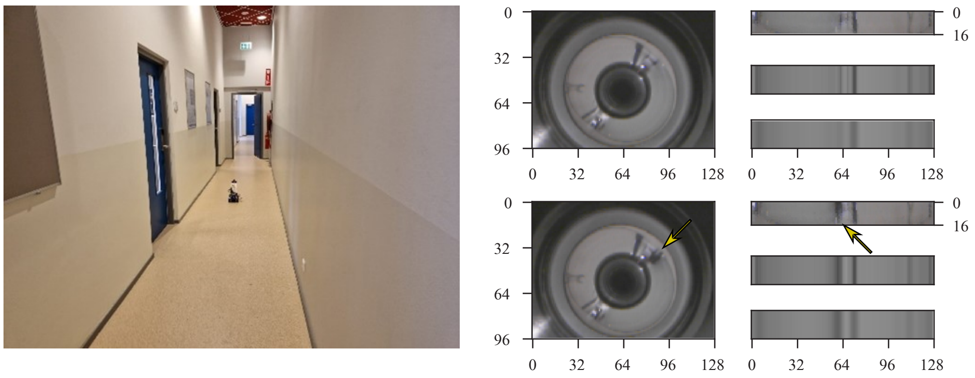
[1] https://www.youtube.com/playlist?list=PLEmDnfepWI6Qiaw74hw9VTlko_EdNQ1NB



(a) Outdoor grass area in-between buildings of the faculty of aerospace engineering. *Bottom right:* Same environment, but with a person (yellow arrow), visible around pixel 55.

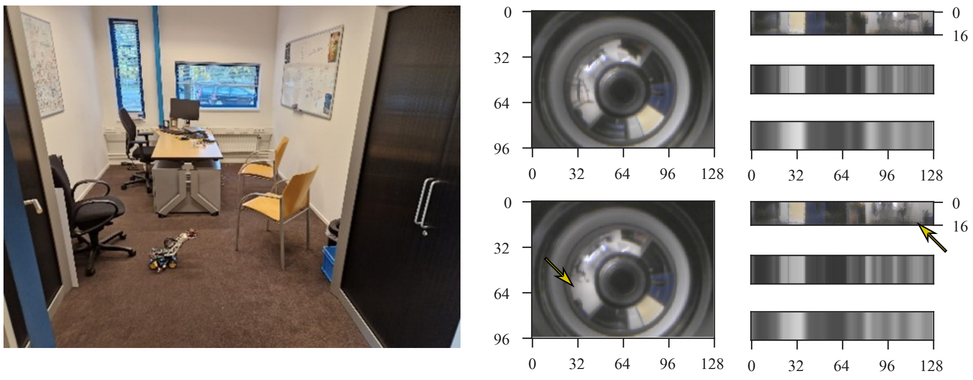


(b) Outdoor pedestrian area in front of TU Delft's SIMONA building. *Bottom right:* Same environment, now with a person standing close-by (yellow arrow). It has an effect around pixels 80-90.

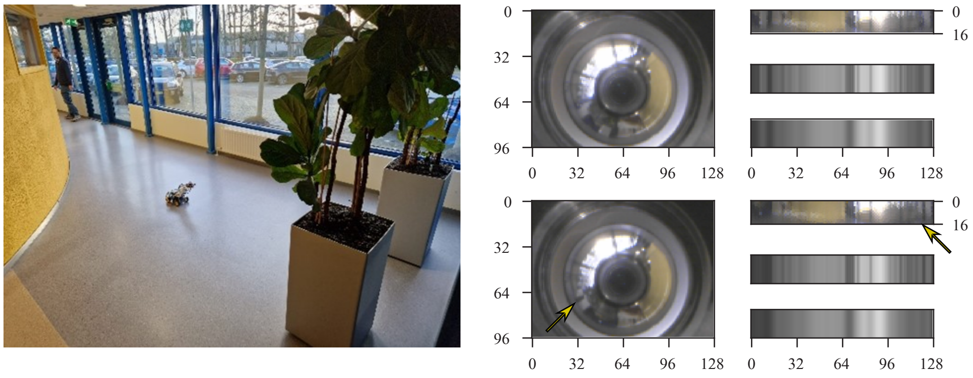


(c) Indoor environment with as little texture as we could find in our building; A corridor with uniformly colored walls. *Bottom right:* Same environment, but with a person.

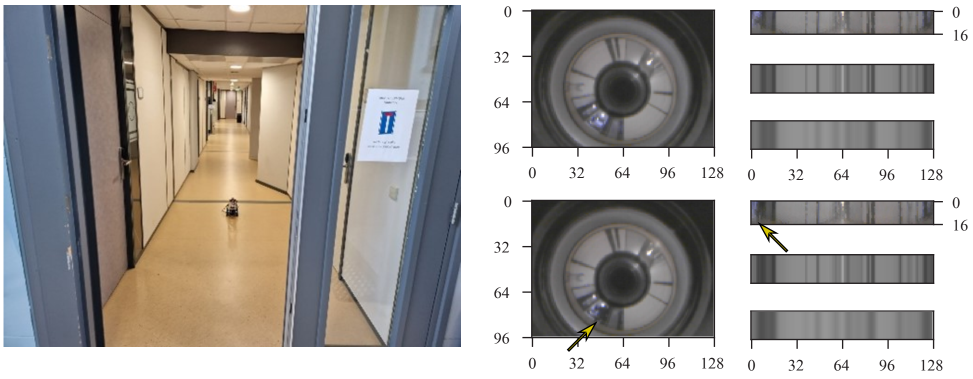
Figure A.2: Snapshots taken in a variety of environments. The left images show an overview of the scenes. In the center are two raw camera images with minor scene differences between them (yellow arrows), to give an impression of the sensitivity to changes. The right column shows the post-processing steps for each raw image. Top: unwrapped panoramic image; middle: vertically averaged image; bottom: Fourier-compressed snapshot. *Continued on page 113, 114.*



(d) Office with chairs close to the robot. *Bottom right:* Same office, with the chairs further away (see the absence of the chair at the yellow arrow).

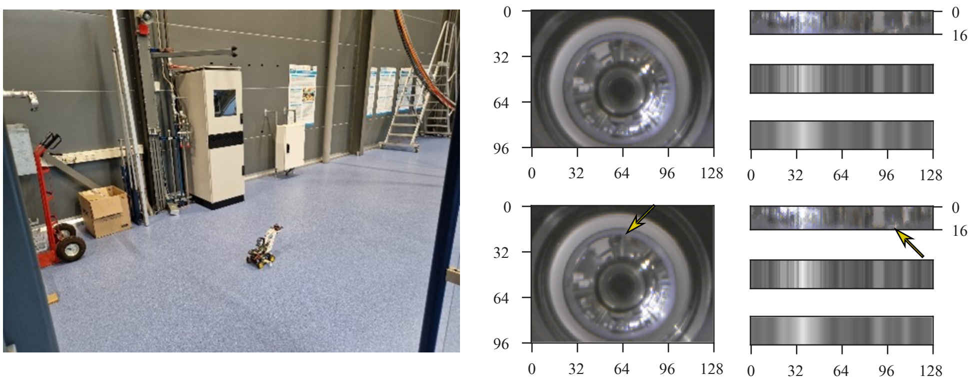


(e) Indoor area with two large plants. *Bottom right:* Same area with plants moved closer together.



(f) Indoor corridor with little texture on the walls. *Bottom right:* Same area, but now with a person (yellow arrow).

Figure A.2: *(Continued)*


(g) Indoor airplane simulation facility. *Bottom right:* Same area, but now with a cardboard box moved (yellow arrow).

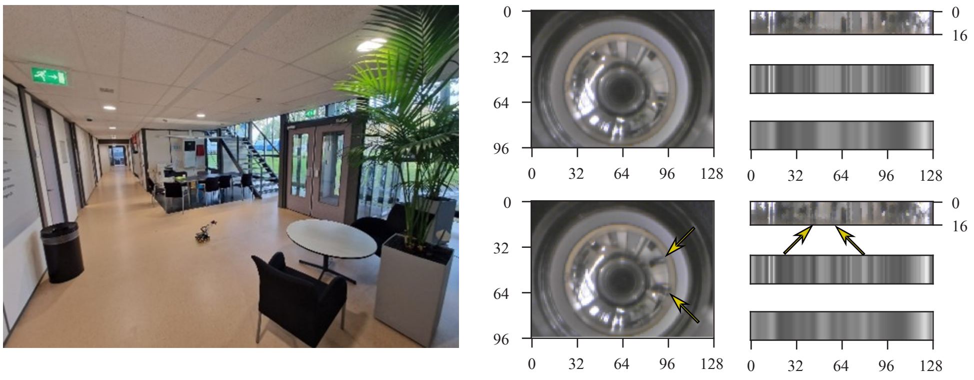

(h) Entrance area of the SIMONA building with tables and chairs. *Bottom right:* Same area with two persons (yellow arrows).

Figure A.2: *(Continued)*

to additionally test how the strategy fares in a natural outdoor environment without having to deal with wind or other disturbances.

To produce relevant results, the simulation should have a high visual fidelity. Furthermore, it should include measurement errors in odometry, as this is the reason a route-following strategy is required in the first place and has a strong influence on the robustness of the proposed strategy.

For this experiment, we use Microsoft's Airsim (https://microsoft.github.io/AirSim/) as a simulator. Specifically, we use the pre-compiled Forest binary from the 1.2.0 linux release. The Forest environment provides realistic imagery of a pinewood forest (Fig. A.3) and is large enough to evaluate long trajectories.

Airsim does not have a native omnidirectional camera. Therefore, we capture four images with a 90 degree field-of-view at a 90 degree interval and combine these into a cube-map (Fig. A.3), from which a cylindrical panorama is sampled.

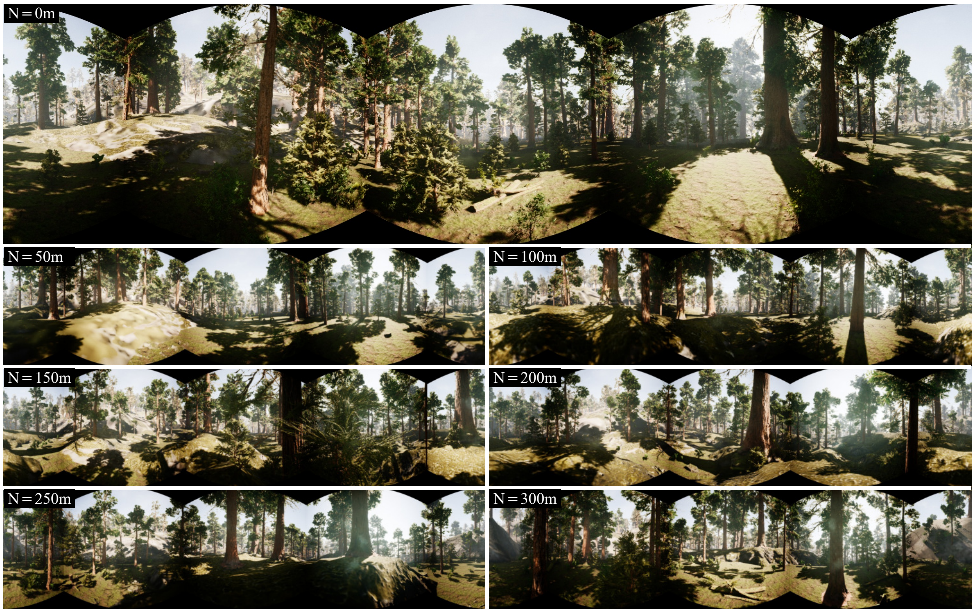


Figure A.3: Airsim Forest environment. Example panoramic photos of Airsim's Forest environment, taken along the 300m trajectory flown by the simulated drone (see Figure A.4). Labels indicate the North position, the East position is 0m for all images.

The need to rotate the drone/camera assembly to capture panoramic images prevents us from using Airsim's physics simulation. Instead, a simplified movement model is implemented using numpy. The movement model contains both the true movement of the drone and a simulation of its odometry error, and is updated at a frequency of 10Hz (simulation time) with a velocity capped at 1m/s.

Similar to our real-world experiments, this model assumes planar movement at a fixed distance above the ground. We assume that elevation changes in the direct vicinity of a snapshot are small enough not to disturb the homing strategy. In Airsim, the ground level is measured by aiming a depth camera toward the ground and adjusting the virtual drone's height accordingly, so that panoramic images are captured at the correct height.

We also note that airsim does not recognize collisions between the drone and the trees, but that collision avoidance is also outside the scope of our investigation. A collision with a tree during either the outbound- or inbound journey therefore does not result in a crash, but can cause visual disturbances by obstacles filling large parts of the field-of-view.

To improve the speed of the simulation, images are only captured on a 1x1m grid (except during the outbound flight, when arbitrary positions for snapshots are allowed). This allows the images to be cached, saving significant rendering time. To calculate homing vectors at arbitrary positions, a homing vector is calculated for each of the four surrounding grid positions and then bilinearly interpolated. This is the same procedure as used in Figure 4.3B on real-world imagery.

For accuracy, our navigation strategy is directly ported from the camera firmware to

Python with as few changes as possible. The simulation therefore uses the same recording, image processing and route following algorithms as the real drone.

To realistically represent the odometry errors and their effect on both the recording and route following process, these are also implemented in our drone model. The odometry error model assumes the presence of Gaussian white noise on the velocity measurements (independent and equally-distributed on both axes), plus Gaussian white noise on the yaw rate measurement. Note that both these measurements are integrated by the state estimator to obtain position and heading, leading to drift over time. The true pose is only used for evaluation and to control the drone during the outbound flight; the simulated navigation stack only has access to the estimated state, which is disturbed by the above errors.

We will now explain the mathematical model underlying the simulation, starting with some notation.

| | |
|---|---|
| $\mathrm{H}_{a,b}$ | is a homogeneous transformation matrix $\in \mathbb{R}^{3\times 3}$ representing the pose of frame $b$ expressed in frame $a$. |
| $\mathrm{T}_a(\Delta p)$ | is a homogeneous transformation matrix representing a translation $\Delta p$ expressed in frame $a$. |
| $\mathrm{R}_a(\Delta\psi)$ | is a homogeneous transformation matrix representing a rotation $\Delta\psi$ expressed in frame $a$. |
| $\mathbf{h}_a$ | is a homogengeous vector $\in \mathbb{R}^3$ expressed in frame $a$ |
| $\mathcal{N}(\sigma)$ | is a zero-mean normal-distributed random variable with standard deviation $\sigma$, or a vector of independently-distributed variables of such type. |

Frame indices are arranged such that $\mathbf{h}_a = \mathrm{H}_{a,b}\mathbf{h}_b$. Moreover, the following identity holds: $\mathrm{H}_{a,b} = \mathrm{H}_{b,a}^{-1}$.

The drone's state is represented by two transformation matrices: $\mathrm{H}_{\text{world,drone}}$ representing the true pose of the drone in the world, and $\mathrm{H}_{\text{odo,world}}$ representing the difference between the true and estimated pose (with "odo" as an abbreviation for "odometry"). For a movement with translation $\Delta p$, the simulation state is updated as follows:

$$\mathrm{H}_{\text{world,drone}} \leftarrow \mathrm{T}_{\text{world}}(\Delta p)\ \mathrm{H}_{\text{world,drone}}$$

$$\mathrm{H}_{\text{odo,world}} \leftarrow \left(\mathrm{T}_{\text{drone}}(\mathcal{N}(\sigma_t))\ \mathrm{R}_{\text{drone}}(\mathcal{N}(\sigma_r))\ \mathrm{H}_{\text{drone,odo}}\right)^{-1}\ \mathrm{H}_{\text{drone,world}}$$

where $\mathcal{N}(\sigma_t)$ is the Gaussian noise on the translational part of the motion and $\mathcal{N}(\sigma_r)$ on the rotational part (the heading). For a movement planned and executed in the drone's odometry frame, the true $\Delta p$ is found by:

$$\Delta p_{\text{world}} = \mathrm{H}_{\text{world,odo}}\ \Delta p_{\text{odo}}$$

For control purposes, the pose of the drone in its internal odometry frame can be found by:

$$\mathrm{H}_{\text{odo,drone}} = \mathrm{H}_{\text{odo,world}}\ \mathrm{H}_{\text{world,drone}}$$

We demonstrate our navigation strategy by sending the drone on a 300 meter straight-line trajectory through the simulated forest. The drone then has to return to its starting

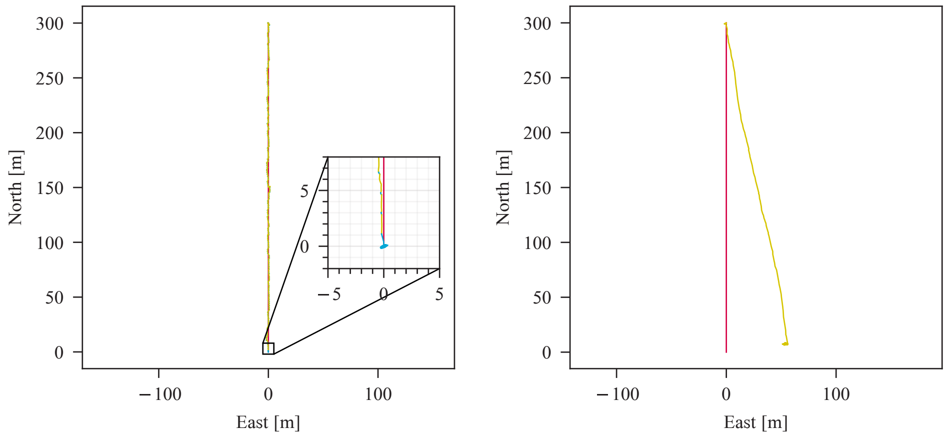


Figure A.4: Route following results in Airsim's Forest. *Left:* Route following result with the proposed navigation strategy. The outbound path is shown in red, odometry segments in yellow and homing segments in blue. The drone is able to record and follow of path of 300 meters with limited deviations. *Right:* The same experiment is repeated, but using only odometry. The drone has an endpoint deviation of more than 50 meters.

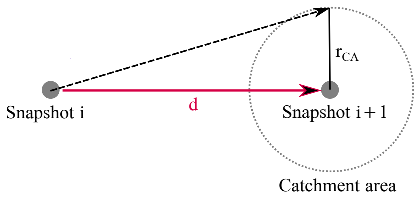


Figure A.5: Simplified theoretical model for spacing snapshots. The figure shows two subsequent snapshots, $i$ and $i + 1$, and the straight route leg between them with distance d. A circular catchment area (CA) with radius $r_{\text{CA}}$ is shown as a dotted circle. The dashed black arrow indicates that the robot deviates from the path due to odometry drift.

position, using the same algorithm and tuning as the real drone. The drone successfully finds its way back to the starting position (Fig. A.4). For comparison, we repeat the same experiment using only odometry and observe that there is an excessive deviation from the intended path (50m), highlighting the need for a route following strategy over longer distances.

## A.3. Theoretical model for spacing snapshots

In this section, we provide a theoretical model for the spacing of snapshots with the proposed strategy. We first present a highly simplified model, since it suffices for conveying the main intuitions on the memory gain obtained by the strategy and the scalability of the approach. Subsequently, we discuss the effect of making several model elements more realistic.

Simplified model Figure A.5 illustrates a simplified theoretical model for the spacing

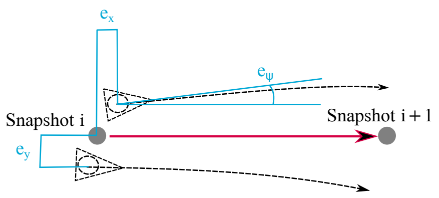


Figure A.6: Illustration of the initial error components due to imperfect homing. The figure shows two subsequent snapshots, $i$ and $i+1$, and the straight route leg between them with distance $d$. It also shows two examples of where the robot can be after homing, with its location (black, dashed circle) and heading (black, dashed triangle). The dashed arrow lines show how the position of the robot can evolve over time when performing the leg. The initial errors in position are shown in purple with the error along the leg direction $e_x$, orthogonal to the leg direction $e_y$. The initial heading error $e_\psi$ is shown for the top example.

of snaphots. In this model, the robot starts perfectly at snapshot $i$, facing in the correct direction towards snapshot $i+1$. The robot travels this leg of the route with the help of odometry. The odometry in this simplified model only introduces a cross-track drift, which is constant per covered meter. In this model we ignore heading drift and along-track drift:

$$\begin{aligned} \psi &\leftarrow 0 \\ x &\leftarrow x + \Delta x \\ y &\leftarrow y + C\,\Delta x \end{aligned}$$

The constant $C$ in this model captures what is maximally expected as cross-track drift per meter. Note that while traveling this leg, the robot keeps its heading constant and attempts to keep its y-position also constant. When traveling a distance $\Delta x = d$, the accumulated drift is $\Delta y = y_{i+1} - y_i = C\,d$. If we want to minimize the number of snapshots while retaining successful homing, the end point of the route leg has to end on the circumference of the catchment area. This implies that the optimal snapshot spacing $d^*$ according to this model is:

$$d^* = \frac{r_{\text{CA}}}{C}$$

Under this (heavily) simplified model, the required snapshot memory increases linearly with the distance.

### A.3.1. Impact of model refinements

In reality, there is not only a cross-track error. In fact, in real flights there are two types of errors: an initial error due to imperfect homing, and an additional drift during the leg due to odometry errors. Moreover, catchment areas vary in both shape and size, depending on the robot's surroundings.

Figure A.6 illustrates the initial errors that arise due to imperfect homing, i.e., an initial position and heading offset. From this initial position, the robot's estimate of where it is with respect to snapshot i + 1 will further deviate due to odometry drift. In the simulation experiments (both of section A.2 and in Figure 4.5A in Chapter 4, p.85), we assume a more realistic model with a Gaussian drift in both position and heading.

$$
\begin{aligned}
&\psi \leftarrow \psi + \mathcal{N}(\sigma_\psi) \\
&x \leftarrow x + \Delta x \cos\psi + \mathcal{N}(\sigma_x) \\
&y \leftarrow y + \Delta x \sin\psi + \mathcal{N}(\sigma_y)
\end{aligned}
$$

Including an initial offset and more realistic drift leads to both a cross-track and along-track error. However, both the simulations and real-world experiments show that the cross-track error is still by far the most significant (an asymmetry arising due to the heading error). A bigger difference with the simplified model arises in terms of the increase of cross-track error over time. Figure 4.5A shows that this relationship is nonlinear, with a nonlinearly increasing error for longer route legs. This property becomes really important when devising a strategy that allows the robot to determine the route leg length dynamically. However, in our setup we assume a fixed route leg length, which means that we can only regard the distribution of position errors at that given length. We can then take, e.g., the maximal offset at that location, and add an extra safety margin to take into account the along-track error and possible odometry drift outliers. From that point on, one can assume to deal with the simplified model.

Moreover, in the real world catchment areas have unknown, irregular shapes, depending on the robot's surroundings. For instance, a snapshot located in the center of a large open field will have a large catchment area (potentially combined with a lower homing accuracy). In a small, cluttered space, omnidirectional vision inputs may change very quickly, leading to a small catchment area (but potentially a higher homing accuracy). Figure A.7 shows three irregularly shaped catchment areas (thin blue, green, and orange dotted lines). Since we do not know up front what the shape and size of a catchment area is, we need to assume a shape and size for the catchment area. As there is no fundamental reason for asymmetry when looking at all possible catchment areas, we assume a circular shape for the catchment area. Moreover, because ending up inside of the catchment area is vital for the proposed strategy, we need to be conservative with our estimate of the catchment area size. That is, we cannot assume it to be too large, as ending up outside of the catchment area poses a substantial risk for getting lost.

One strategy to estimate the catchment area is to assume it to be minimal among a set of measured catchment areas. Figure A.7 shows what this looks like with a thick, grey dotted circle that represents the maximal circular catchment area that intersects with all irregularly shaped ones. Of course, one can be even more conservative by setting the catchment circle radius even smaller. Changing the circle radius means moving on the trade-off between route-following success and memory and time expenditure. Assuming a smaller radius means shorter legs and a lower probability of ending up outside the CA. However, it also means storing more snapshots in memory, and performing visual homing more often. Because visual homing takes time, this might finally even reduce route-following success when the route is long. This is illustrated in Figure 4.6C in Chapter 4, where snapshots are placed really close to one another.

For the snapshot spacing in the real-world experiments, we made a conservative choice for the distance between snapshots of $1 - 2$ meters. The odometry drift results in Figure 4.5C show a cross-track error of $< 8$cm, implying a very small required CA size. This while the CA that can be deduced from the experiments in Figure 4.4A possibly has a

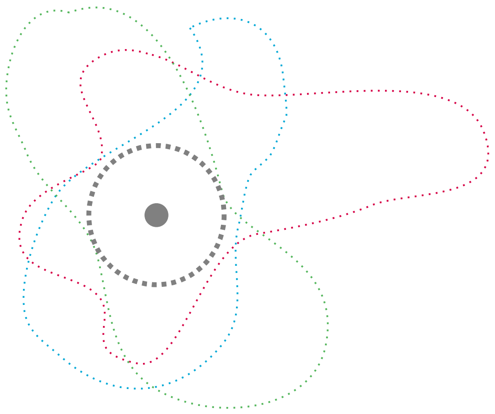


Figure A.7: Varying catchment areas modeled by a circular region of intersection. The snapshot location is represented by a full, grey circle. The blue, green, and orange dotted lines show possible, irregularly shaped catchment areas. The thick, grey, dotted line shows an assumed circular catchment area as the maximal circular intersection area of the real catchment areas.

radius of ∼2.25m. Future work could investigate interesting avenues to better exploit the relationship between odometry drift and visual homing. For instance, the robot could estimate CA size dynamically and have a procedure to deal with "getting lost" when outside a CA. Moreover, one could also improve the odometry error model. For instance, odometry gets less accurate when flying really fast, due to motion blur. Moreover, visual odometry like the one from the Crazyflie's flowdeck works less well when there is little visual texture, and the height measurements from the downward-pointing laser can be impaired by direct sunlight. Investigation into these matters and their use in a revised route-following strategy could allow to further reduce memory requirements and / or improve robustness.

## A.4. Position plots over time for flight experiments

Figure 4.6 in Chapter 4 shows a top view of the robot's positions during S- and U-shaped trajectories. In this section, we show the same information over time in Figure A.8. The plots over time show that the battery during the homing-only U-flight was depleted after 200 seconds, earlier than during the odometry-and-homing U-flight. This is purely due to differences between batteries. However, it is also clearly visible that the drone gets much less far along the trajectory during the 200s of homing-only flight than during the same period of the odometry-and-homing flight.

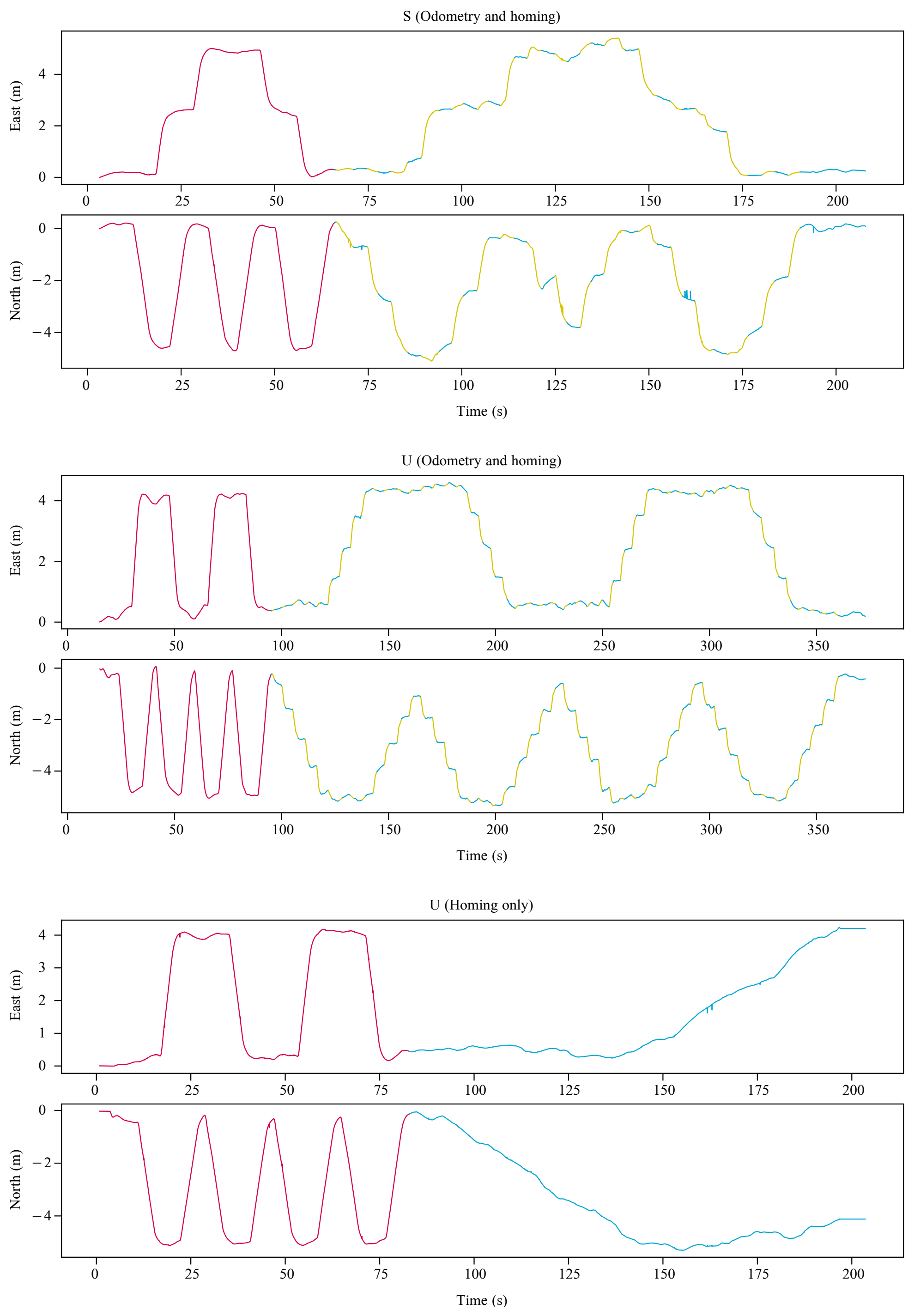


Figure A.8: Time sequence plots of Fig. 4.6. The plots show the east and north position of the drone over time in the same experiments. The same color coding as Fig. 4.6 is used: red is the outbound segment, yellow indicates route following by odometry and blue by homing.

# Acknowledgements

Whew, you've made it all the way to the end of my thesis! All that remains now is for me to thank the people that made this dissertation possible.

First of all, I want to thank Guido and Christophe. Guido, for your endless enthusiasm and for all the great discussions we've had. I always left with new ideas and inspiration and think I could have filled a lifetime with further research if we investigated everything that we discussed. Christophe, for your incredible technical insight – of which I am still jealous – and your ongoing willingness to help despite already having so many responsibilities. A lot of my work is in one way or another dependent on things you have made, so many, many thanks for that! If time were infinite, I think the three of us could have done many more exciting research projects together. It was a true pleasure to work with you both.

Next, I would like to thank all of my current and former colleagues: labmates, office-mates and other PhDs within our section. I would write down all of your names, but I've met so many great people during my time at the lab that it probably wouldn't fit here. I have great memories of our lunches together, the PhD drinks, barbecues and work outings, board game nights ("lekker doorbranden") and especially traveling abroad. To those people that have already graduated, I hope you are doing well. For those that still need to graduate: hang in there, I'm rooting for you!

Not everything went smoothly during my PhD. I would like to thank Guido and Christophe again for their support and consideration during this time, and the TU Delft for being so accommodating. I would especially like to thank the fellow PhD's who I could talk to during this period, for which I am still deeply thankful, it meant a lot to me. I would also like to thank some people outside the university, who I will leave unnamed, that have helped me recover and learn a lot about myself.

It is also customary to thank my family here, but I think they are already well aware of how I feel ;). Nevertheless, I'll say again that I am deeply grateful for my parents' and sister's continuous support and company throughout my PhD. They have helped me keep my sanity, as it were, and I have always loved spending time together. I look forward to our further adventures in the future. Thank you for always being there for me.

The same thanks go out to all of my friends, both those who I've known for years and the new friends I made during my time at the MAVLab. Thank you for all the good times we've had together, and I hope we'll have many more in the future!

By now, it is definitely time for me to leave. I have good memories of the MAVLab and PhD life, but after seven years I am looking forward to something new. I am excited to start a career outside of academia, and thanks to you all I can now happily round off this PhD phase of my life.

– Tom

# Curriculum Vitæ

**Tom van Dijk**

| | |
|---|---|
| 09-06-1992 | Born in Tilburg, the Netherlands. |

## Education

| | |
|---|---|
| 2003–2009 | VWO NT + NG<br>Theresialyceum, Tilburg |
| 2009–2012 | BSc Mechanical Engineering<br>Delft University of Technology<br>*Minor:* Electrical Engineering for Autonomous Exploration Vehicles |
| 2012–2017 | MSc Mechanical Engineering<br>Delft University of Technology |
| 2012–2017 | MSc Systems & Control<br>Delft University of Technology |
| 2017–2024 | PhD Aerospace Engineering<br>Delft University of Technology<br>*Thesis:* Visual Navigation for Tiny Drones<br>*Promotor:* Prof.dr. G. C. H. E. de Croon<br>*Copromotor:* Dr.ir. C. De Wagter |

## Work experience

| | |
|---|---|
| 2012–2013 | Drivetrain Engineer (sensors)<br>Formula Student Team Delft |
| 2013–2014 | Electronics Engineer (wiring)<br>Formula Student Team Delft |
| 2014–2015 | Vehicle Dynamics Engineer (control system)<br>Formula Student Team Delft |
| 2024– | Medior Embedded Developer<br>Fox Crypto B.V. |

# List of Publications

7. **van Dijk, T.**, De Wagter, C., de Croon, G.C.H.E. (2024). *Visual route following for tiny autonomous robots*. Science Robotics.

6. Meester, R.S., **van Dijk, T.**, De Wagter, C., de Croon, G.C.H.E. (2023). *Frustumbug: a 3D mapless stereo-vision-based bug algorithm for Micro Air Vehicles*. 14th annual International Micro Air Vehicle conference and competition.

5. Keltjens, B., **van Dijk, T.**, de Croon, G. (2021). *Self-supervised monocular depth estimation of untextured indoor rotated scenes*. arXiv:2106.12958.

4. Wijnker, D., **van Dijk, T.**, Snellen, M., de Croon, G., De Wagter, C. (2021). *Hear-and-avoid for unmanned air vehicles using convolutional neural networks*. International Journal of Micro Air Vehicles, 13.
   - *Conference version:* Wijnker, D., **van Dijk, T.**, Snellen, M., de Croon, G., De Wagter, C. (2019). *Hear-and-avoid for UAVs using convolutional neural networks*. 11th International Micro Air Vehicle competition and conference.

3. Olejnik, D. A., Duisterhof, B. P., Karásek, M., Scheper, K. Y., **van Dijk, T.**, de Croon, G. C. (2020). *A Tailless Flapping Wing MAV Performing Monocular Visual Servoing Tasks*. Unmanned Systems, 8(04), 287-294.
   - *Conference version:* Olejnik, D. A., Duisterhof, B. P., Karásek, M., Scheper, K. Y. W., **van Dijk, T.**, de Croon, G. C. H. E. (2019). *A Tailless Flapping Wing MAV Performing Monocular Visual Servoing Tasks*. 11th International Micro Air Vehicle competition and conference.

2. **van Dijk, T.** (2020). *Self-Supervised Learning for Visual Obstacle Avoidance*. TU Delft.

1. **van Dijk, T.**, de Croon, G. (2019). *How do neural networks see depth in single images?* Proceedings of the IEEE/CVF International Conference on Computer Vision.