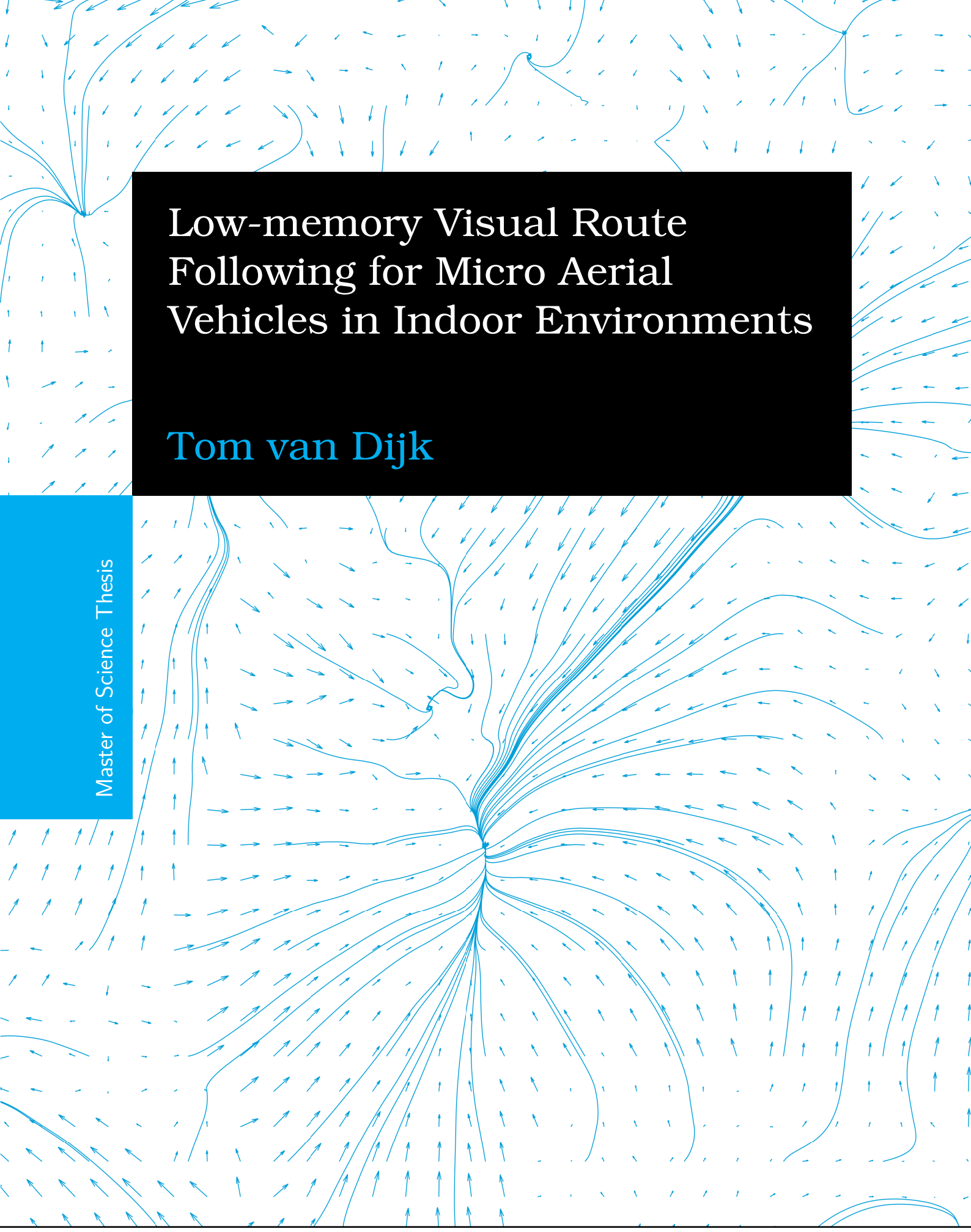
Low-memory Visual Route Following for Micro Aerial Vehicles in Indoor Environments

Tom van Dijk

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Low-memory Visual Route Following for Micro Aerial Vehicles in Indoor Environments

Master of Science Thesis

For the double degree of Master of Science in Systems and Control and Mechanical Engineering at Delft University of Technology

Tom van Dijk

October 9, 2017

Kimberly McGuire	Daily supervisor, AE
Guido de Croon	Supervisor, AE
Pascual Campoy	Supervisor, SC
Pieter Jonker	Supervisor, ME-BMD

Faculty of Mechanical, Maritime and Materials Engineering (3mE)

.

Delft University of Technology

Summary

Micro Aerial Vehicles (MAV) are ideal for indoor missions such as search and rescue because of their small size. Indoors, however, the presence of walls, ceilings and other obstacles means that the availability of an RF link with a ground station for remote control is not guaranteed; therefore, the drone should be able to operate autonomously. GPS is not available either for the same reason, so the drone needs to rely on another form of navigation, in this thesis: visual navigation.

The goal of this thesis is to find a visual route following method that can be used on MAVs. Unlike larger Unmanned Aerial Vehicles (UAV), lightweight (<50 grams) MAVs carry only a simple microcontroller that is incapable of running the Simultaneous Localization and Mapping (SLAM) algorithms that are typically used for visual navigation. The small amount of memory available on the microcontroller is a major limitation, therefore this work attempts to find an alternative method of route following that consumes as little memory as possible. To accomplish this goal, a biologically inspired approach towards navigation is followed in this thesis: using a combination of visual homing and odometry, the drone can travel long distances with a tiny map.

Visual homing allows the drone to return to the location where a reference image — a snapshot — was taken. While there is ample literature on visual homing, its memory consumption has received hardly any attention. To minimize the memory consumption of navigation, this work therefore identifies three visual homing methods that might still work with tiny snapshots: search-based homing, Matched Filter Descent in Image Distances (MFDID) and Fourier-based homing. The performance of these methods is evaluated and compared while the snapshot is reduced to a size of 64 bytes or less. Fourier-based homing performs best under these conditions, allowing the drone to return to a reference position using a snapshot compressed to only 8 to 12 bytes.

In order to follow longer routes, the drone can use visual homing to move from waypoint to waypoint along the route: sequential visual homing. However, visual homing only works over a short distance so this would require a large number of waypoints. Instead, odometry is first used to bring the drone close to the next waypoint before homing is attempted. This allows the distance between waypoints to be increased, resulting in a sparser and therefore more memory-efficient map.

The proposed route following method is tested in simulation and on a real drone. Visual homing towards a single point is shown to work on both platforms. Next, route following using sequential visual homing with and without odometry is demonstrated on a Parrot AR.Drone 2.0 in a short corridor section. With odometry, the waypoints can be spaced at a larger distance, leading to a more memory-efficient map. Using the same autopilot and settings, a simulated drone performs a similar experiment along a 63 meter route in a realistic indoor environment. In both experiments, the drone was able to follow the route with a map that consumes as little as 17.5 bytes per meter.

Table of Contents

Preface

I've always found robot navigation a fascinating problem. It is a very basic, almost essential first step towards completely autonomous behavior of robots with clear, practical applications. Navigation seems so simple to humans that we often don't even think about it, yet it remains a real challenge for machines.

This project seemed like an excellent opportunity to get to work with Simultaneous Localization and Mapping (SLAM), a very interesting non-linear filtering problem that is often used as the cornerstone for robot mapping, navigation and exploration. However, it quickly became clear that this is not going to work on smaller platforms. Instead, during this project I had the chance to look at alternative approaches to navigation, not just for robots but also navigation techniques used by insects such as bees or ants or the occasional rat or box jellyfish. Not a field I had expected to work with but very inspiring nonetheless; it seems there is more to copy from nature than smart mechanisms, it also provides smart control strategies.

In this thesis I present a solution to a problem that has received relatively little attention in literature: how can you *minimize* the memory consumption of visual navigation? I hope that the method presented here provides new opportunities to people working with tiny Micro Aerial Vehicles (MAV) or other mobile robots and that it may inspire further research into this fascinating field.

During this project I have received help from a number of people who I want to thank here, first of all Kimberly McGuire. You really put in a lot of time and effort which is hugely appreciated, you were always willing to help and have reviewed countless revisions of everything I have written during this research. Not only did this improve the quality of this work, it was also really motivating during the tougher parts of this project. Thank you for doing a fantastic job!

I would also like to thank Guido de Croon for his endless enthusiasm about the project and for the insightful discussions we've had; I always left with new ideas. Similarly, I would like to thank Pascual Campoy and Pieter Jonker for their support of this work.

Thanks to Titus Braber and the other people at the MAVLab for thinking along when I got stuck. I also want to thank Erik van der Horst for his support with all the practical aspects of working with drones and Paparazzi.

Finally, I want to thank my parents and my sister for their encouragement and support and for reminding me to step back and relax every once in a while.

Delft, University of Technology — Tom van Dijk
October 9, 2017

Thesis structure

This thesis consists of two parts: a paper and a collection of appendices.

The paper is the main part of this thesis. It contains a detailed account of my work and results on low-memory visual route following for Micro Aerial Vehicles. The paper presents the topic and research question of this thesis, an overview of related work, the methods used in this thesis and the results, discussion and conclusion. The paper is a self-contained document: it can be read on its own and it has its own appendix and list of references.

The appendices expand on the paper by presenting additional work that was performed for this thesis but that did not make it into the final paper, for reasons of brevity or because the work was abandoned in favor of different approaches. Since the appendices expand on parts of the paper, the paper should be read first.

Appendix A presents a detailed review of the image-based homing methods that were reviewed in Section 3 of the paper. This review would have taken up too much space in the paper, but is included here to ensure the reproducibilty of the experiments in this thesis.

Appendix B presents a method by which the radius of the catchment area could be estimated in-flight. As proposed in Section 4 of the paper, this information can be used to increase the distance between waypoints on the map, leading to an even lower memory consumption.

Appendix C explores the use of the omnidirectional camera for odometry as an alternative to the IMU-based odometry presented in the paper. Flight tests show that route following is also possible with visual odometry, although its performance is more difficult to predict as it depends on the shape and texture of the environment.

Appendix D presents a brief overview of the simulator used to demonstrate long-range visual route following. This simulator was developed as part of this thesis, as no suitable simulator existed that could be used to test vision-based control with the Paparazzi autopilot. The appendix gives a brief overview of its implementation and the environments that were built for this thesis.

Low-Memory Visual Route Following for Micro Aerial Vehicles in Indoor Environments

Tom van Dijk Kimberly McGuire Guido de Croon Pascual Campoy Pieter Jonker

***Abstract*— This paper presents a visual route following method that minimizes memory consumption to the point that even Micro Aerial Vehicles (MAV) equipped with only a simple microcontroller can traverse distances of a few hundred meters. Existing Simultaneous Localization and Mapping (SLAM) algorithms are too complex for use on a microcontroller. Instead, the route is modeled by a sequence of snapshots that can be followed back using a combination of visual homing and odometry. Three visual homing methods are evaluated to find and compare their memory efficiency. Of these methods, Fourier-based homing performed best: it still succeeds when snapshots are compressed to less than twenty bytes. Visual homing only works from a small region surrounding the snapshot, therefore odometry is used to travel longer distances between snapshots. The proposed route following technique is tested in simulation and on a Parrot AR.Drone 2.0. The drone can successfully follow long routes with a map that consumes only 17.5 bytes per meter.**

NOMENCLATURE

a_k, b_k	Coefficients of Fourier-transformed image [-]
a_x, a_y	Accelerometer measurements in body frame (forward, right) [$\mathrm{m/s^2}$]
β	Bearing relative to current heading [rad]
c_x, c_y	Lens center [px]
γ	Vertical image resolution [px/rad]
ϕ, θ, ψ	Roll, pitch, yaw [rad]
$\boldsymbol{h}$	Homing vector [-]
$I^C(\beta)$	Pixel intensity in current image [-]
$I^T(\beta)$	Pixel intensity in target image [-]
$\hat{I}(\beta\|\xi,\eta,\varsigma)$	Pixel intensity in predicted image [-]
m	Mass [kg]
μ	Linear drag coefficient [$\mathrm{N/(m/s)}$]
R	Environment radius [m]
r	Minimum catchment area radius [m]
r	Horizon sampling radius [px]
$\Delta r(\beta\|\phi,\theta)$	Horizon attitude correction [px]
ς	Relative orientation [rad]
ς_0	Relative orientation (coarse) [rad]
$\Delta\varsigma$	Relative orientation (correction) [rad]
u, v	Velocity in body frame (forward, right) [m/s]
x, y	Relative position [m]
ξ, η	Relative position, unknown scale [-]

I. INTRODUCTION

MICRO Aerial Vehicles (MAV) can be used for a variety of tasks of which search-and-rescue is an often-mentioned example. An MAV could, for instance, search for people inside the unstable remains of a building after an earthquake without putting the lives of firefighters or other personnel in immediate danger. In indoor scenarios such as these, availability of GPS or remote control is not guaranteed, so the drone would need to be able to navigate autonomously. This navigation typically relies on vision, as cameras provide a large amount of data for relatively little weight and do not require external infrastructure or modifications to the environment.

On larger UAVs, onboard visual navigation is typically performed using Simultaneous Localization and Mapping (SLAM) (e.g. [1], [2]). However, these algorithms are too complex to be performed on an MAV that only carries a microcontroller. Memory consumption tends to be another limiting factor, as visual SLAM consumes megabytes or gigabytes of memory when mapping long trajectories. Without SLAM, an alternative method of navigation has to be found.

In nature, honeybees and ants can successfully navigate over long distances without a detailed geometrical map of the environment. Instead, it is theorized that these insects remember images ('*snapshots*') seen during exploration and try to match these during navigation using a process called *visual homing*. Cartwright and Collet presented the snapshot model [3] to explain the homing behavior in bees and replicate it in simulation. Numerous other works have since been published on visual homing (Section II-B). Navigation using the snapshot model has been demonstrated successfully on mobile robots [4], [5] and quadrotors [6].

The goal of this work is to *minimize the memory consumption of visual route following on Unmanned Aerial Vehicles*. This makes it possible to perform long-range route following on microcontrollers and frees up resources on platforms with more computational power. Route following can be used to bring the drone back to its starting location, for instance to recharge or to re-establish communications with a ground station, or as part of a topological navigation strategy [7].

A route following technique is proposed that uses a combination of Fourier-based visual homing and odometry to traverse long distances up to $500\,\mathrm{m}$ while consuming less than ten kilobytes of memory. This technique is evaluated in simulation and in test flights on a Parrot AR.Drone 2.0. To achieve low memory consumption, the following contributions were made:

1) The memory efficiency of three image-based homing methods — search-based homing [8], Matched Filter Descent in Image Distances (MFDID) [9] and Fourier-based homing [5] — is evaluated and compared. The influence of the snapshot size on the performance of search-based homing and MFDID had not been evaluated before. Its influence on Fourier-based homing was investigated in [5] but not compared to other methods.
2) Fourier-based homing is implemented on a UAV. Earlier work has only used Fourier-based homing in simulation or on a wheeled mobile robot [5].

3) Fourier-based homing is combined with odometry to increase the distance between snapshots along the route. The use of odometry was originally proposed by Vardy [10] to increase the distance between snapshots but no further attempts were made to minimize memory consumption. It is the combination with Fourier-based homing proposed here that leads to the dramatic decrease in memory consumption. Unlike [10], the combination of homing and odometry is also evaluated on a vehicle in the real world.

The remainder of this paper is structured as follows: Section II gives an overview of related work on visual route following and visual homing techniques. Section III evaluates and compares the memory efficiency of three image-based homing methods. Section IV then shows how visual homing can be combined with odometry to efficiently traverse longer distances. In Section V, Fourier-based homing is implemented on a UAV. In Section VI, Fourier-based homing and the proposed visual route following method are tested in simulation and on a real UAV. The results are discussed in Section VII and the conclusions of the paper are presented in Section VIII.

II. Related work

A. Route following

Long-range navigation techniques that do not rely on SLAM tend to use one (or more) of the following behaviors: path following, visual compass following or sequential visual homing. *Path following* uses existing paths in the environment for navigation, for example corridors [11], [12]. Path following can be used over long distances while little to no data needs to be stored to recognize the path's end and is therefore highly memory efficient. The main limitation of path following, however, is that it requires the drone to follow clearly distinguishable paths in the environment; these paths may not be present or the drone could follow other trajectories, therefore path following is not generally applicable.

Visual compass following controls the drone's direction of travel to follow a recorded path. The horizontal offset of a reference image in the drone's forward field-of-view is used to determine the steering angle, correcting both lateral and course errors. Zhang and Kleeman prove that this scheme converges when 1) only features in the 180° forward field-of-view are used and 2) the reference images along the route are close together [13]. The latter requirement limits the memory efficiency of visual compass-based navigation.

Examples of visual compass following are [13], [14], where [13] is one of the very few examples where appearance-based navigation is demonstrated over long distances with trajectories up to 600 meter. However, with raw images stored every thirty centimeters it consumes too much memory for use on a microcontroller. In [15] Baddeley et al. use visual compass following to simulate ant behavior, but instead of storing reference images an Infomax neural network is trained to recognize 'familiar' images. The memory consumption is hard to judge as the capacity of the network is not known, however the large number of weights and the need to train the network during exploration make it unlikely to work on a microcontroller. In [16] De Cristóforis et al. use a combination of visual compass following and path following; the former allows navigation in arbitrary environments while the latter reduces the size of the map when paths are available.

Visual homing allows the drone to return to the location where a reference image was taken. *Sequential visual homing* allows long routes to be followed by homing from one waypoint to the next. Examples are [17]–[19]. Visual homing only works from a limited region surrounding the target image, the *catchment area*. If waypoints are spaced far apart, a different navigation method is required to bring the drone inside the next catchment area before homing is attempted. In [6] Denuelle and Srinivasan control the drone's position inside the catchment areas. The waypoints are spaced such that their catchment areas only slightly overlap, the drone is then guided through this overlap from one catchment area to the next. This effectively doubles the maximum distance between waypoints. In [10], Vardy uses odometry to move towards the next catchment area. This allows even longer distances to be traversed between waypoints.

Sequential visual homing is more generally applicable than path following and allows a larger distance between reference images than visual compass following. Therefore, this paper will use sequential visual homing for long-range navigation. As in [10], odometry will be used to traverse longer distances between waypoints. The next paragraphs will take a closer look at visual homing techniques.

B. Visual homing

Visual homing uses a reference image to guide the drone back to a known location. In 1983, Cartwright and Collett published a paper in which they model the homing behavior of bees. Their *snapshot model* assumes that bees compare their current visual input to an image taken at the target location — the snapshot — and use differences in feature bearings and sizes to estimate a homing vector that points towards the target. The snapshot model has inspired a large number of visual homing methods that can be broadly classified under two categories based on the way the snapshot is represented: feature-based homing and image-based homing.

Feature-based homing treats the snapshot as a collection of local features that each have their own bearings. Changes in these bearings are used to find a homing vector that points in the target direction. Examples are [3], [4], [19]. Feature-based homing requires the same features to be used in the current and target images. Practical applications therefore often rely on descriptors like SIFT [20] to match corresponding features between images. Given the generally large number of features, the size of the descriptor has a strong influence on the total size of the snapshot. Memory consumption can be dramatically reduced by choosing small descriptors such as D-BRIEF [21] which consumes only four bytes per feature. Memory consumption can be further reduced by sharing

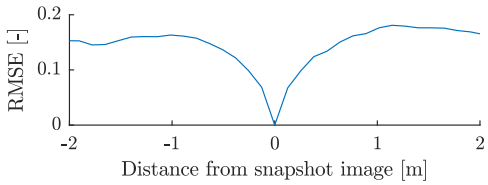


Fig. 1. Image Dissimilarity Function (IDF) along a single axis. The difference between the current image and the snapshot increases with the distance from the snapshot's location. Image-based homing tries to minimize the difference between the two images (e.g. through gradient descent) in order to move back towards the snapshot's location.

descriptors (and bearings) between multiple snapshots, as demonstrated in [22], [23].

Image-based homing uses the entire images to find a homing vector without extracting local features. Homing is then performed by minimizing the difference between the current and target images. While raw images consume significant amounts of memory, the snapshots used for image-based homing can be strongly compressed to reduce their memory consumption. An example is the work of Stürzl and Mallot [5] in which a mobile robot successfully homes to a target location using only the first five complex coefficients of a Fourier-transformed image.

A feature-based snapshot of similar size could only hold the bearings towards a handful of undescribed features. Furthermore, feature-based homing requires the detection and matching of features between the images, which are often computationally intensive steps. For these reasons, this work will use image-based homing to navigate towards nearby waypoints. The next paragraphs give a short overview of the image-based homing methods that will be compared in Section III.

C. Image-based homing

The key observation behind image-based homing is that the difference between the current and target images increases with the distance from the target [24], [25] (Fig. 1). The target is located at the (global) minimum of this *image difference function* (IDF); homing is performed by moving such that the difference between the images is minimized. Image-based homing can already be performed when only the difference to the target image is available, as demonstrated by Zeil et al.'s *RunDown* method [24].

In [8], Franz et al. present a method to predict new images for small, hypothetical movements (translation and rotation) by *warping* the current observation. Images are predicted for a large number of movements, the one resulting in best match with the target image is used as homing vector. This method will be called *search-based homing* for the remainder of this paper.

Instead of explicitly searching for the minimum of the IDF, it can also be found through gradient descent. Predicted images produced with Franz et al.'s warping method [8] can be used to estimate the spatial gradient of the IDF at the current position. Such an approach is taken in Möller and

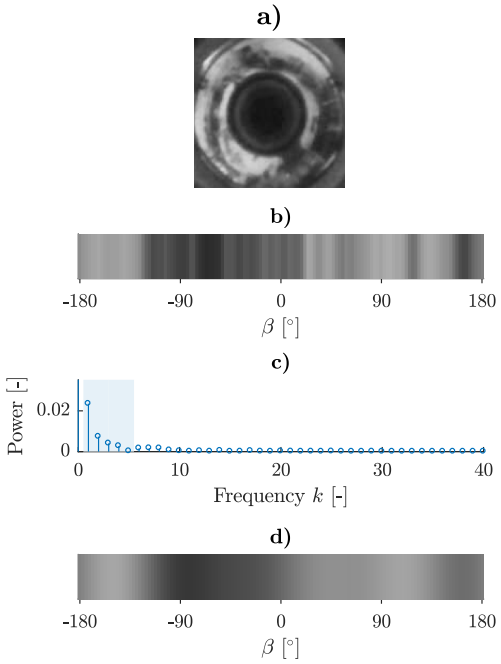


Fig. 2. Fourier-transformed horizon image. a) Original image. b) Extracted one-dimensional horizon image. c) Power spectral density of the horizon image. Note that most of the power is contained in the low-frequency components (highlighted), the high frequency components add relatively little information. Components in the highlighted region are stored for navigation while the others are discarded. The DC value is discarded as well as it does not provide directional information. d) Reconstructed horizon image using only the highlighted components.

Vardy's *Matched Filter Descent in Image in Image Distances (MFDID)* [9], [26]. Since it does not need to predict large amounts of images, MFDID is less computationally intensive than search-based homing. However, its use of gradient descent instead of search might cause it to get stuck in local minima of the image difference function that would be ignored by search-based homing.

In [5], Stürzl and Mallot transform search-based homing to the frequency domain: *Fourier-based homing*. Under the observation that most information in images tends to be found in the lower frequencies (Fig. 2), only the first K low-frequency components are used while the others are discarded. Successful homing is demonstrated on a mobile robot with $K = 5$. With appropriate rounding, this snapshot would consume only ten bytes in total. Furthermore, the warping procedure of Franz et al. is replaced by its first-order Taylor approximation. Since the resulting difference function is quadratic in the hypothetical movements, its minimum can then be found directly and a computationally intensive search is avoided. (A summary of the mathematics behind Fourier-based homing can be found in the appendix of this paper.)

These image-based homing methods are reviewed in more

detail in Appendix A of the thesis.

III. Memory efficiency of image-based homing

The previous section has presented three image-based homing methods: search-based homing, MFDID and Fourier-based homing. For memory-efficient navigation, the snapshot used by these methods should be small, but apart from Fourier-based homing the homing performance at very small snapshot sizes has not received much attention in literature. Therefore, this section will evaluate the homing performance at very small snapshot sizes to determine the memory efficiency of these methods. Using these results, the methods can be compared and the most efficient approach can be identified.

For long-range trajectories, the total size of the map depends on two factors: the size of the snapshot images and the distance between them. The size of the snapshot is used as independent variable in this experiment, it is therefore known beforehand. The distance between the images follows from the *catchment area*, the region surrounding the snapshot from which homing will succeed. A larger catchment area means that homing can be performed over longer distances; the distance between waypoints can be increased and the map becomes sparser and more memory-efficient. The size of the catchment area will therefore be used as a performance measure by which the selected methods can be compared.

The catchment area is defined as the region surrounding the target image from which homing will succeed. It therefore has to be found through trial-and-error. A simple simulation based on a grid of panoramic images is used to generate a large number of homing trajectories; the trajectories that end close to the target position define the catchment area. For a single target, the catchment area is measured using the following procedure:

1) For each image in the grid, a homing vector towards the snapshot is calculated (Fig. 3a).
2) Starting at each grid position, a homing run is simulated (Fig. 3b). The trajectories are calculated with MATLAB's `stream2` function, which interpolates the homing vectors surrounding the current position and advances the trajectory in steps of 0.1 grid cell until the homing vector becomes zero or a maximum number of steps is reached. The final error between the end of the trajectory and the target position is stored in the starting cell.
3) All starting cells with a final position error below a threshold (here: 1 grid cell, $12.7\,\mathrm{cm}$) form the catchment area (Fig. 3c). The total size of the catchment area is stored for this target position.

This procedure is repeated for 112 target positions spread evenly throughout the test environment.

Panoramic images that are used as snapshots and current observations are obtained from a dataset published by Gaffin and Brayfield in 2016 [25]. The dataset contains $100 \times 100\,\mathrm{px}$ grayscale images taken at $12.7\,\mathrm{cm}$ intervals in a $7.3 \times 6.9\,\mathrm{m}$ room and part of the adjacent corridor. Pixels around the horizon are sampled using nearest-neighbor interpolation to create the one-dimensional horizon images used by the homing methods, where the resolution of the horizon image is equal to the snapshot size in bytes. For Fourier-based homing, the snapshot size follows from the number of coefficients, where each complex coefficient consumes two bytes of memory.

To ensure a fair comparison between the homing methods, their tuning parameters were optimized for each individual image size. For MFDID, these are the cutoff frequency of the low-pass filter and the use of a Hessian correction, for Fourier-based homing this was the number of iterations N_{it}. A horizon image of $256\,\mathrm{px}$ was used as input for Fourier-based homing's DFT.

The resulting catchment areas are averaged over all target positions to produce the graph shown in Fig. 4a. At small image sizes, Fourier-based homing has a significantly larger catchment area than search-based homing or MFDID. The statistical significance of this result was evaluated using a one-tailed paired-samples sign test. At snapshot sizes of 20 bytes or less, the median difference between MFDID and Fourier-based homing is significant with $p \leq 2.0 \cdot 10^{-7}$, $N = 112$.

The catchment areas of search-based homing and MFDID increase with larger image sizes. When the image size is increased, search-based homing performs better than MFDID. A possible reason is that MFDID's gradient descent gets stuck in local minima, while search-based homing explicitly searches through all possible movements and can therefore look beyond these incorrect minima.

The catchment area of Fourier-based homing, however, decreases at large snapshot sizes. A maximum of approximately $4\,\mathrm{m}^2$ is achieved at a snapshot size of 10–12 bytes, but the area reduces when the size of the snapshot is increased further. This counter-intuitive result was also observed by the original authors who noted linearization errors and errors in relative orientation estimation as likely causes [5]. The catchment area also drops at snapshot sizes below 8 bytes. At these sizes the final position error increases and more starting cells fall outside the selected threshold of one grid cell.

The total size of the map depends on both the size of the snapshots and the distance between them. Therefore, both Fourier-based homing (moderately sized catchment areas and small snapshots) and search-based homing (large catchment areas with large snapshots) could lead to a low total memory consumption. Under the assumption that the distance between waypoints grows linearly with the size of the catchment area, a measure of 'efficiency' can be obtained by dividing the size of the catchment area by the snapshot size in bytes (Fig. 4b), where a larger catchment area per byte indicates a higher memory efficiency. Using this efficiency measure, it follows that Fourier-based homing should lead to a lower memory consumption than search-based homing when long trajectories are mapped. Note that for Fourier-based homing, the largest catchment area is obtained at a snapshot size of 10-12 bytes (Fig. 4a). However, for a

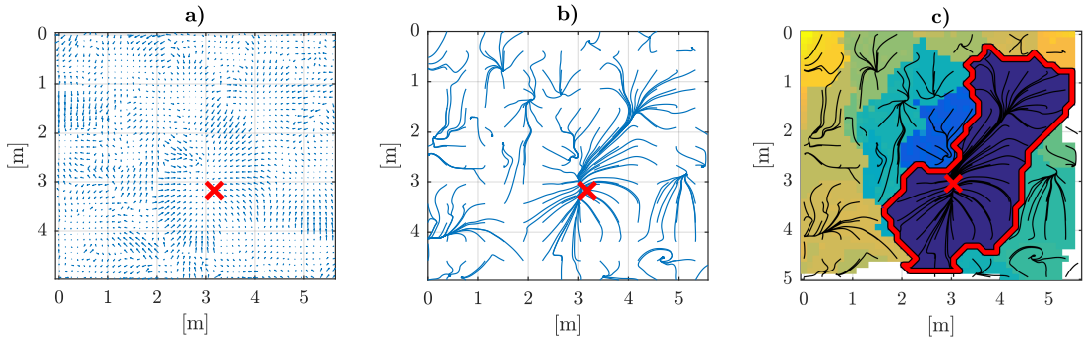


Fig. 3. Evaluation of the catchment area surrounding the target indicated by the red cross. a) For each image in the grid a homing vector towards the snapshot is calculated. b) Using these homing vectors, homing trajectories starting at each grid cell are generated. c) For each trajectory, the final position error relative to the snapshot location is determined. Starting positions where this error falls below the given threshold (here 1 grid cell, 12.7 cm) belong to the catchment area. The boundary of the catchment area is shown in red.

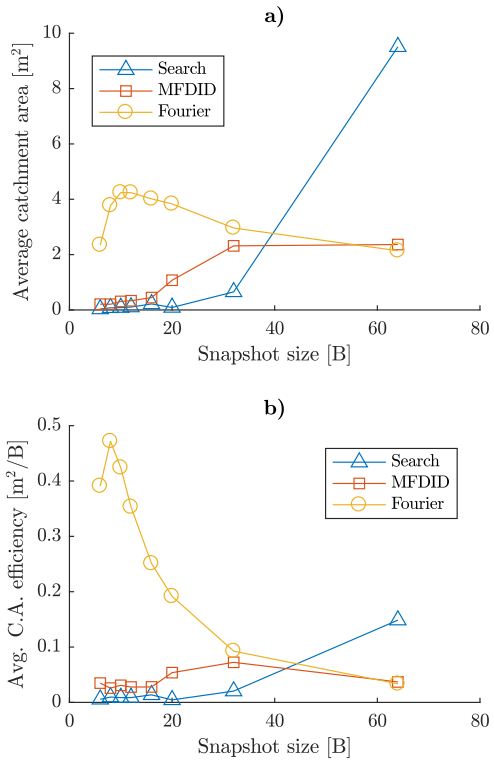


Fig. 4. Memory efficiency of search-based homing, Matched Filter Descent in Image Distances (MFDID) and Fourier-based homing. a) Average size of the catchment area as a function of the snapshot size in bytes. At snapshot sizes $\leq 32\,\mathrm{B}$, Fourier-based homing has the largest catchment area, while search-based homing performs best at snapshot sizes of 64 bytes. b) Average catchment area size per byte, used as a measure of memory efficiency. This plot is found by dividing a) by the snapshot size. Fourier-based homing is the most efficient at a snapshot size of 8 bytes.

maximal efficiency of $0.48\,\mathrm{B/m}$ a snapshot of eight bytes should be used (Fig. 4b).

Instead of the area, the minimum radius of the catchment area can also be used as a measure of performance (see Section IV). The results are the same, Fourier-based homing performs better than MFDID, with a maximum catchment area at twelve byte snapshots and an optimal efficiency at eight bytes.

At small snapshot sizes, Fourier-based homing performs better than search-based homing and MFDID. Therefore, this method will be used in the remainder of this paper.

IV. Distance between waypoints

With the memory consumption of the snapshot minimized, the next step is to maximize the distance between successive waypoints. When only visual homing is used to move between waypoints, each waypoint has to lie inside the catchment area of the next. This means that waypoints need to lie closely together at distances in the order of one meter or less, leading to a dense and memory-inefficient map. Instead, other navigation techniques should be used bring the drone towards the next waypoint before homing is attempted. In this paper, IMU-based odometry is used to traverse longer distances between waypoints because it does not make any assumptions about the environment and is therefore generally applicable.

The maximum distance between waypoints depends on the size of the catchment area of the next waypoint and the uncertainty in the drone's position at the start of the homing maneuver. After dead reckoning, the probability that the drone is inside the next catchment area should be sufficiently large, especially since this procedure should be repeated many times without failure.

Consider the following example: the drone should be able to follow a $500\,\mathrm{m}$ trajectory with waypoints every meter with an 80% reliability. The success rate with which the drone arrives inside the catchment area of the next waypoint should then be higher than $0.80^{1/500} \approx 0.9996$, because a single

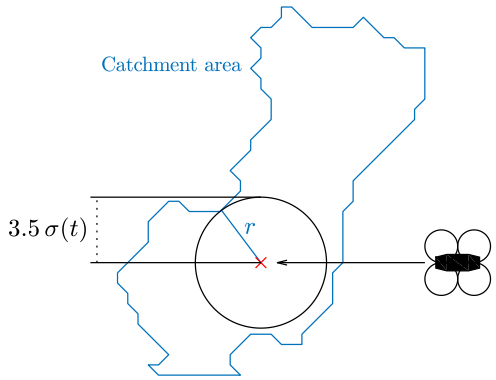


Fig. 5. The maximum distance between waypoints depends on the size of the catchment area r and the odometric uncertainty. The odometric uncertainty $\sigma(t)$ increases with time, but should remain small enough so that the UAV arrives inside the next catchment area with a sufficiently high success rate.

failure could already cause the drone to get lost. Assuming that the odometric error is normally distributed, this success rate is achieved when the minimal catchment area radius is larger than $N_\sigma = 3.5$ times its standard deviation (Fig. 5).

Odometry estimates the drone's position by integrating noisy velocity measurements. The standard deviation of integrated white noise grows with the square root of time, while an integrated bias grows linearly with time. Constant-speed drifts in position in the order of $20\,\mathrm{cm/s}$ or larger are not unheard of on drones. These could, for instance, be caused by a slight offset in the attitude estimation. Because these errors tend to be significantly larger than those caused by white measurement noise, the standard deviation of the position error will be assumed to grow linearly with time:

$$\sigma(t) = \sigma_{drift} t \tag{1}$$

Apart from the odometric uncertainty, the size of the catchment area is required to determine the maximum distance between waypoints. A constant minimum radius r could be assumed for the size of the catchment area. In this case, the time between waypoints would also be constant:

$$t_{max} = \frac{r}{2N_\sigma \sigma_{drift}} \tag{2}$$

where factor N_σ follows from the required success rate as shown in the example above. The maximum time is divided by two because the error is integrated during recording as well as during traversal on the way back. As an example, with an unknown, constant drift sampled from a zero-mean normal distribution with standard deviation $\sigma_{drift} = 20\,\mathrm{cm/s}$, a catchment area radius of $r = 1.5\,\mathrm{m}$ and $N_\sigma = 3.5$, the maximum time between the recording of waypoints is $1.1\,\mathrm{s}$.

In practice, however, the size of the catchment area varies per waypoint. When catchment areas are larger than expected, the odometric error can be allowed to grow further and the distance between waypoints can be increased, leading to a sparser and more memory-efficient map. The converse is also true, when the catchment areas become smaller, more waypoints should be created to prevent failures, however this can already be prevented by choosing a conservative value for the time between snapshots.

Appendix B of the thesis presents the first steps towards a method that can be used to estimate the radius of the catchment area in-flight. The experiments in Section VI, however, still use a constant time between waypoints for simplicity. The constant time between waypoints is increased until route following fails.

V. Practical implementation on a UAV

The experiment of Section III was performed using a dataset of panoramic images and did not include any quadrotor dynamics. For a proper evaluation, the proposed route following method has to be implemented on a real UAV. This section highlights some practical aspects of this implementation that have not been covered in the previous sections. The solutions presented here are applied on both the real and simulated drones in the experiments of Section VI.

A Parrot AR.Drone 2.0 will serve as testing platform. The drone is controlled using the Paparazzi autopilot[1] which provides attitude estimation and control, video handling and logging.

Before route following with visual homing and odometry can be performed on a UAV, the following problems need to be solved: 1) the UAV will pitch and roll, these movements need to be corrected when sampling the horizon from a panoramic image; 2) the velocity of the UAV needs to be estimated for use in odometry; and 3) a control loop is required to follow the homing vector.

A. Panoramic vision and image derotation

To capture panoramic images, the bottom camera of the AR.Drone 2.0 is fitted with a panoramic lens[2] (Fig. 7). The camera has a resolution of $320 \times 240\,\mathrm{px}$, however as shown in Fig. 8 only a small part of this image around the horizon is sampled. Camera frames are captured at $25\,\mathrm{fps}$.

The horizon is sampled between a radius r_{bottom} and r_{top} around the center of the lens c_x, c_y:

$$\begin{aligned} I(\beta, r) = I(&c_x + (r + \Delta r(\beta|\phi,\theta)) \cos\beta, \\ &c_y + (r + \Delta r(\beta|\phi,\theta)) \sin\beta) \end{aligned} \tag{3}$$

with $r_{bottom} \leq r \leq r_{top}$ and with β the bearing relative to the drone's forward axis. The minimal and maximal sampling radius were tuned by hand such that a large vertical portion of the image was sampled to provide robustness against altitude deviations, but not too large that parts of the floor and ceiling are sampled in an average office room. Fig. 8 gives an example of the raw image from the camera, the sampling region and the extracted horizon image.

[1] https://wiki.paparazziuav.org/wiki/Main_Page
[2] Kogeto Dot 360°

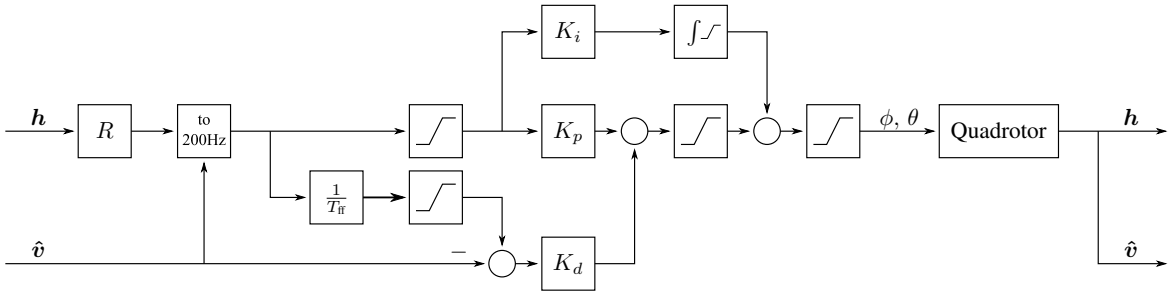


Fig. 6. Position control loop. The position of the quadrotor is controlled using the homing vector $\boldsymbol{h}$ (replaced by the odometry vector when traveling between waypoints) and the estimated velocity $\hat{\boldsymbol{v}}$. The homing vector $\boldsymbol{h}$ is scaled by the environment radius R (set by the operator) to produce a position setpoint in meters. In the 'to 200 Hz' block, the velocity estimate is used to update the position setpoint between camera frames. Time constant T_{ff} is used to generate a velocity reference. Note that both $\boldsymbol{h}$ and $\hat{\boldsymbol{v}}$ are already expressed in the drone's body-fixed coordinate frame and therefore do not need to be rotated by the heading of the drone.

Fig. 7. Bottom side of the AR.Drone 2.0 with panoramic lens.

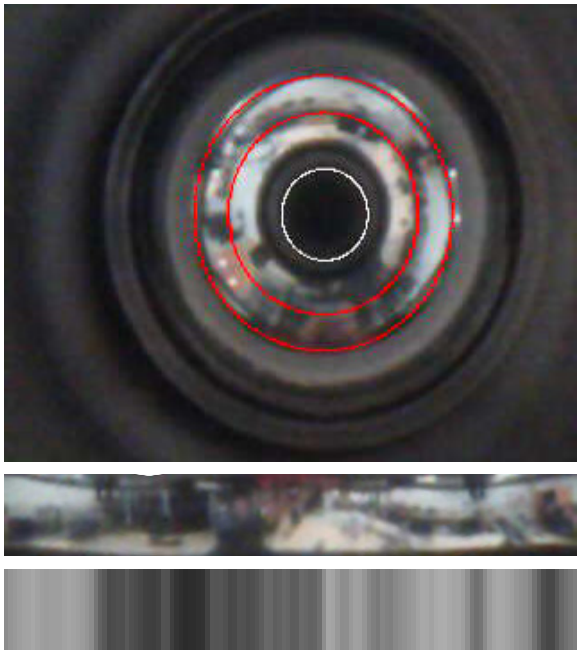


Fig. 8. Horizon images as captured by the AR.Drone 2.0. *Top:* raw image captured by the bottom camera. The white circle indicates the center of the lens. The horizon is sampled between the two red circles. *Middle:* unwrapped view of the horizon. *Bottom:* extracted one-dimensional horizon image.

Pitch and roll movements θ, ϕ are corrected by adjusting the sampling radius:

$$\Delta r(\beta|\phi,\theta) \approx \gamma(\phi \sin\beta + \theta\cos\beta) \tag{4}$$

This model assumes small bank angles and a constant vertical resolution γ in px/rad around the horizon. To obtain a one-dimensional horizon image, each pixel at bearing β was averaged over ten samples between r_{bottom} and r_{top}.

B. Velocity estimation and odometry

Compared to mobile robots, odometry is difficult to perform on a UAV as it has no physical contact to the ground and can therefore drift in all directions. An Inertial Measurement Unit (IMU) can be used to measure accelerations and angular velocities of the UAV, but integration of these measurements causes drift. This drift is particularly pronounced in the horizontal plane [27], because an attitude error of a few degrees can already cause a large change in horizontal velocity. Unknown biases in the gyroscopes and accelerometers can quickly cause such an offset in the attitude estimation. These biases tend to drift over time and are therefore difficult to eliminate through calibration.

The problem of estimating velocity is typically solved using a bottom camera. This camera measures the optical flow of the floor and is combined with a height sensor to provide an estimate of the drone's velocity. This, however, assumes that the floor has sufficient texture, which can not always be guaranteed in indoor environments. Similarly, a top-facing camera could be used to measure optical flow of the ceiling, although the same limitations apply here.

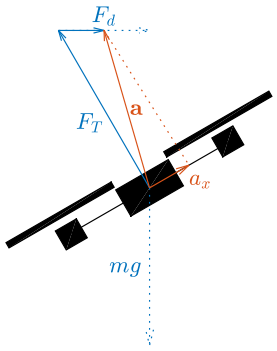


Fig. 9. Forces acting on the quadrotor during sideways acceleration. Gravity g is not measured by the accelerometer and thrust F_T only acts along the body-fixed z-axis. Therefore, the only acceleration that is measured on the body-fixed x (and y) axis comes from the drag F_d. A drag model can be used to estimate the velocity from this drag measurement.

Alternatively, the panoramic camera can be used to estimate the velocity of the drone.[3] The main difficulty of this method is that the distance towards tracked features needs to be known. Monocular vision-only solutions can therefore typically only estimate velocity up to an unknown scale. Visual-inertial odometry combines these measurements with readings from the IMU to provide an absolute scale. Visual-inertial odometry tends to be computationally complex as it has to maintain an estimate of the distance towards the tracked features.

Yet another method is suggested in [28], [29]. Here, the accelerometer is used to measure the drag force acting on the quadrotor. During flight, three forces act on the quadrotor: thrust F_T, drag F_d and gravity mg (Fig. 9). Accelerometers do not measure gravity (they measure *proper acceleration*, i.e. acceleration relative to an inertial frame in free fall), therefore only the thrust and drag are measured by this sensor. Since the thrust is assumed to act along the body's z-axis, the only force that remains to be measured on the body x- and y-axes is the drag (a_x in Fig. 9). Using a drag model, the airspeed of the drone (and ground velocity, assuming wind-still conditions) can then be found directly from the drag measurements a_x, a_y. In practice, at low speeds the drag acting on the quadrotor tends to grow linearly with velocity. Paradoxically, the accelerometer can therefore be used to measure the drone's velocity without integration (Fig. 10):

$$\hat{u} = \frac{a_x}{(\mu/m)}, \quad \hat{v} = \frac{a_y}{(\mu/m)} \tag{5}$$

A drag term (μ/m) of $0.75\,\mathrm{s}^{-1}$ was found by minimizing the error between the measured and ground-truth velocities obtained during a test flight. Because of its simplicity and because it does not make assumptions about the environment, drag-based velocity estimation will be used to measure the velocity of the drone.

[3]This option is further explored in Appendix C of the thesis.

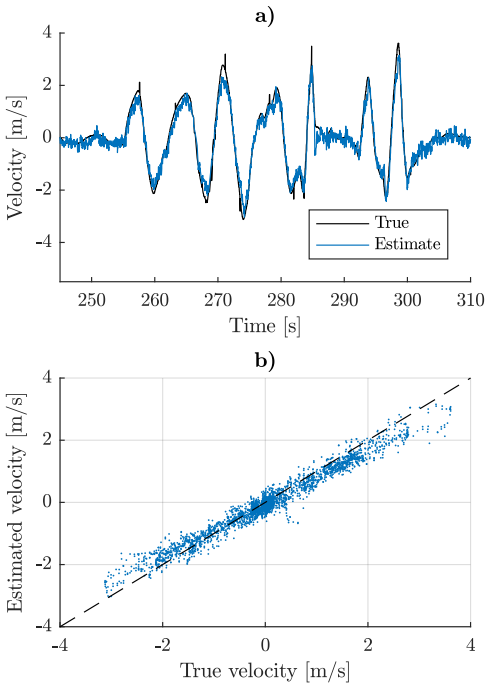


Fig. 10. Velocity estimation using the IMU and a linear drag force model. a) Comparison between the estimated velocity and the true velocity obtained through motion capture. b) Scatterplot of true and estimated velocities.

A downside of drag-based velocity estimation is that any bias of the accelerometer is directly visible in the velocity estimate. The accelerometer bias tends to slowly drift over time and is therefore difficult to estimate and correct. Temperature calibration of the accelerometer is required to bring the bias drift down to usable levels. Calibration was performed during four test flights, where the drone hovered in one position for ten to fifteen minutes while the temperature and accelerometer measurements were logged. These were used to generate a look-up table that provides a bias correction for a given temperature.

To construct odometry vectors between waypoints, the velocity is sampled and integrated at 200 Hz. The estimated heading, based on gyroscope and magnetometer, is used to determine the direction of movement.

C. Closed-loop position control

A PID controller is used to track the position setpoints provided by the homing vector or the odometry. The controller is shown in Fig. 6. The controller generates attitude setpoints ϕ, θ for the low-level stabilization controller of Paparazzi. An integrative action is required to correct offsets in the attitude estimate, which can have errors in the order of two degrees. The control actions are saturated to limit the bank angle of the drone to a maximum of ten degrees; this prevents the frame of the quadrotor from blocking parts of

the horizon.

The gains K_p, K_d and K_i use the default values of Paparazzi's AR.Drone 2.0 configuration. Environment radius R is tuned by hand based on the size of the environment. With a correct setting of R, the length of $R\boldsymbol{h}$ should be roughly equal to the distance to the snapshot.

The homing vector is updated at $25\,\mathrm{Hz}$ as it is limited by the camera framerate. The $200\,\mathrm{Hz}$ velocity measurements are used to update the position setpoint between camera frames. When a new homing vector $\boldsymbol{h}$ arrives, it overwrites the existing setpoint.

Low-level attitude control is performed using Incremental Nonlinear Dynamics Inversion (INDI) [30] as provided by Paparazzi. The tuning parameters were estimated using Adaptive INDI during a test flight.

Vertical guidance of the UAV is performed using the onboard sonar or using the altitude provided by an external motion capture system.

VI. Experimental results

The previous sections have presented a navigation method for quadrotors that drastically reduces the amount of memory required for route following. The working principles and efficiency of visual homing were demonstrated in a simple simulation without quadrotor dynamics. In this section, the presented method will be evaluated on a simulated and a real quadrotor.

This section will demonstrate that:

1) visual homing as presented in Section III works on a real quadrotor.
2) routes can be traversed using sequential visual homing, optionally combined with odometry.
3) the use of odometry increases the maximum allowable distance between waypoints.
4) a memory consumption of less than twenty bytes per meter is achieved.

A. Experimental setup

Flight tests will be performed in simulation and on a Parrot AR.Drone 2.0. This section gives a brief overview of the equipment used for the experiment; the experiments themselves and their results are described in Sections VI-B to VI-D.

1) Simulation: The quadrotor simulations are performed in Gazebo[4,5]. The drone is modeled as a rigid body with four forces and torques acting upon it. Drag acting on the drone grows linearly with velocity. The simulated drone is equipped with an omnidirectional camera at its bottom side with a resolution of $240 \times 240\,\mathrm{px}$. The image is disturbed using a per-pixel Gaussian noise signal with a standard deviation of 0.007 (with pixel values between [0..1]).

The simulated drone is controlled using the same autopilot as the real drone. This includes the low-level attitude control loop. Measurements from the simulated drone are disturbed by noise that should roughly match that of the real platform.

During the outbound flight, the true position is used to guide the drone through the environment. On the way back, the true position is not available to the controller but only logged for later evaluation.

2) Experimental: The real-world experiments are performed on the AR.Drone 2.0 described in the previous section. The experiments are performed in the TU Delft's *Cyberzoo*, a $10 \times 10 \times 7\,\mathrm{m}$ testing ground for UAVs. The area is surrounded by black curtains to prevent disturbances from outside the test area. These curtains do not provide useful features for navigation, so additional texture was provided by shower curtains hanging around the edge of the test area (Fig. 11).

The position of the drone is recorded using an Optitrack[6] motion capture system. This position is logged for later analysis and used to control the drone's position when not flying autonomously. This position measurement is also used to guide the drone along its outbound trajectory during recording.

B. Visual homing

In this first experiment, the drone homes towards a single target image in the test environment. The goal of this experiment is to show that visual homing works on a UAV despite the additional dynamics and disturbances, and to show that the catchment area is sufficiently large for navigation.

In simulation, homing is evaluated in Gazebo's *Café* environment[7] (Fig. 12). In the real experiment, homing is performed inside the cyberzoo (Fig. 11). One of the curtains was opened and shower curtains were hung on the other sides to provide additional features to the otherwise featureless walls.

The following procedure was used for the experiment: first, the drone takes off and hovers at a central position where it records its target image. Then, the drone moves to its starting position (selected from a 5×5 grid surrounding the target, see Fig. 13) from which homing is attempted. The drone follows the homing vector until 1) the vector (ξ, η) becomes smaller than 0.01, indicating arrival at the target, or 2) 20 seconds have passed. The resulting trajectories are shown in Fig. 13.

In both the simulation and the real test flight, the drone can successfully find its way back towards the target location in most runs. In simulation, 14 out of 15 runs were successful, the other run did not arrive at the target in time but appeared to follow a correct trajectory.

On the real drone, 13 out of 15 runs were successful. During the other two runs, the drone moved in an incorrect direction, indicating that it was outside the catchment area of this snapshot. The closest failed run started at a distance of $1.79\,\mathrm{m}$ from the target position, which is assumed to be the minimum radius of the catchment area of this snapshot.

[4] `http://gazebosim.org/`

[5] More information on the simulator can be found in Appendix D of the thesis.

[6] `http://optitrack.com/`

[7] `http://models.gazebosim.org/cafe/` by Nate Koenig.

Fig. 11. TU Delft's *Cyberzoo*, a $10 \times 10 \times 7\,\mathrm{m}$ testing area for UAVs. To provide additional texture for visual homing, the left curtain is partially opened and shower curtains are hanged on the back- and right walls.

Fig. 12. Gazebo's *Café* environment.

TABLE I
MAXIMAL MEMORY-EFFICIENCY IN THE CORRIDOR ENVIRONMENT.

Strategy	Wpt. distance [m]	Map size [B/m]
Snapshots only	0.3	56.7
Snapshots + IMU odo.	1.2	17.5

When the two failed runs are discarded, the average final position error in the Cyberzoo is $35\,\mathrm{cm}$ (SD = $16\,\mathrm{cm}$, $N = 22$). In Gazebo the average final position error is $22\,\mathrm{cm}$ (SD = $11\,\mathrm{cm}$, $N = 23$).

C. Route following

In the second experiment, the drone follows routes that are too long to perform in a single visual homing maneuver. Routes are followed by sequentially homing towards waypoints along the route, with and without odometry. The goal of this test is to show that long-range navigation is possible using sequential visual homing and that the use of odometry increases the maximum distance between waypoints, thereby improving the memory efficiency of the map.

Route following is evaluated on the real drone in a 'corridor' setup inside the Cyberzoo (Fig. 14). Compared to Section VI-B, the nearby walls drastically reduce the size of the catchment areas; it is not possible to fly from one end of the corridor to the other in a single visual homing maneuver.

The drone uses its ground truth position to fly along the corridor while recording waypoints and odometry vectors. The drone then follows the route back to its starting location using sequential visual homing. Waypoints were recorded at 0.3, 1.0, 2.0 and 4.0 second intervals whilst flying at a speed of $0.3\,\mathrm{m/s}$ (corresponding to 10, 30, 60 and 120 centimeter distances between waypoints). The largest intervals between snapshots that still resulted in at least one successful return flight are compared, this ensures that the maps are as sparse as possible. On the way back, the drone follows the odometry vector until it is twenty centimeters from its endpoint (if applicable), followed by a visual homing maneuver that is performed for four seconds (this time was required to arrive and stabilize at the waypoint).

The resulting flight trajectories are shown in Fig. 15. The maximum distances for which successful runs were observed are listed in Table I. When only snapshots are used for navigation, the maximum distance achieved between waypoints was $30\,\mathrm{cm}$. With odometry a maximum distance of $1.2\,\mathrm{m}$ was achieved.

The memory consumption of the map is calculated by dividing the size of the snapshot plus the odometry vector (if applicable) by the distance between the waypoints. In this test, the snapshots were 17 bytes in size, 16 bytes for the complex coefficients (this value was found in an earlier version of the simulation in Section III) and an additional byte for the compass heading used to improve orientation estimation. Snapshot-only navigation consumed 56.7 bytes per meter. When odometry is used to move between waypoints before homing is attempted, the memory consumption of the map is reduced to 17.5 bytes. The use of odometry is therefore an effective way of lowering the memory consumption of navigation, in this case memory consumption was reduced by 69% compared to sequential visual homing without odometry.

The trajectories in Fig. 15 also show that visual homing successfully brings the UAV close to its waypoint, but that the IMU-based dead reckoning has large errors in both direction and distance. At these distances between waypoints, visual homing is just able to correct these errors. If the

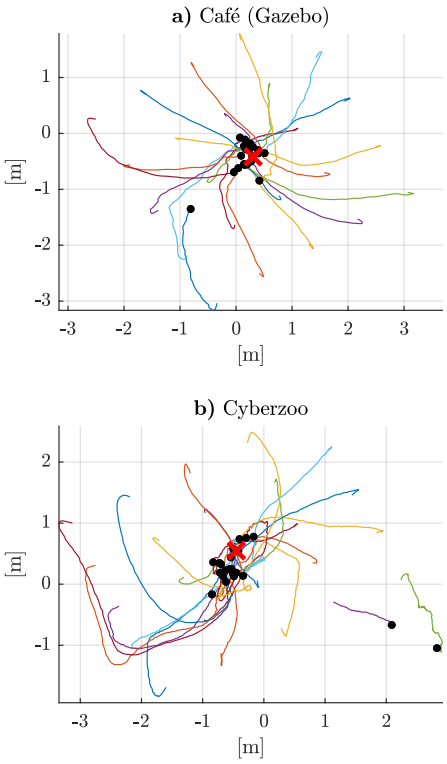


Fig. 13. Visual homing results. The drone homes towards a snapshot taken at the red cross. The lines show the quadrotor trajectories, the black dots the endpoints. a) Homing trajectories in Gazebo's Café environment (Fig. 12). b) Homing trajectories in the cyberzoo (Fig. 11). The two bottom-right trajectories do not guide the drone towards the snapshot; these starting points therefore lie outside the catchment area.

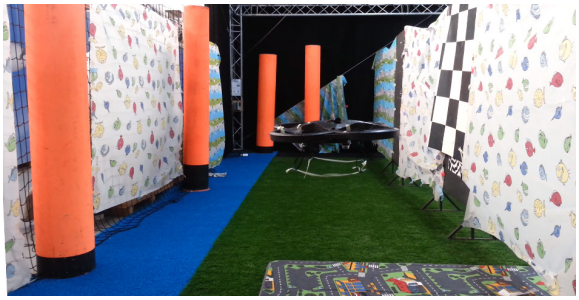


Fig. 14. 'Corridor' environment inside the Cyberzoo. The corridor is a rectangular area of approximately $10 \times 3\,\mathrm{m}$. Shower curtains have been placed along the sides of the corridor to provide sufficient texture for navigation.

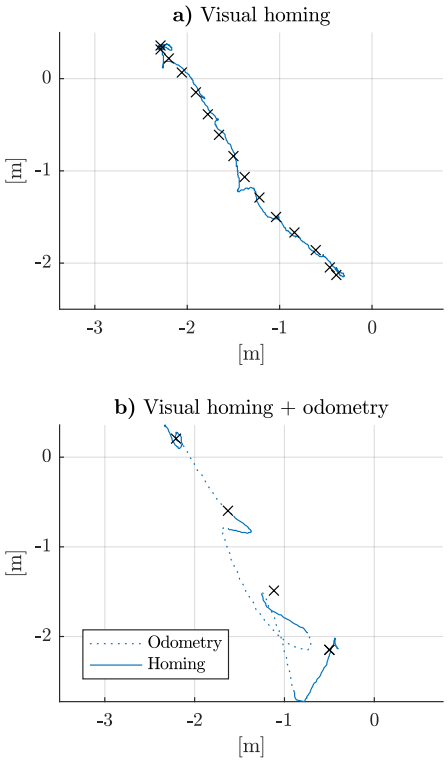


Fig. 15. Route-following results without (top) and with (bottom) odometry between snapshots. Dotted lines indicate dead-reckoning using odometry, while solid lines indicate visual homing maneuvers. The waypoints along the route are indicated by black crosses. The drone travels from the top left to the bottom right. The use of odometry allows for a larger distance between waypoints.

odometric error could be reduced further, an even larger distance between waypoints might be achieved.

D. Long-range route following

The previous experiment has demonstrated the use of sequential visual homing and odometry to travel longer distances. The environment, however, was limited in size and had different visual features than a generic indoor environment. To show that route following also works over long distances in realistic environments, a similar experiment is repeated in a model of (part of) the Aerospace Engineering faculty of the TU Delft (Fig. 16).

In this environment, the drone follows a trajectory of $63\,\mathrm{m}$. The same settings are used as in the previous experiment: four seconds between snapshots whilst flying at a speed of $0.3\,\mathrm{m/s}$. The resulting trajectory is shown in Fig. 17.

The drone successfully followed the recorded route. Since the recording settings were unchanged, this map also consumes $17.5\,\mathrm{B/m}$, showing that this is sufficient for navigation over long distances. On one occasion, visual homing failed to bring the drone back to its intended waypoint, but the homing maneuver at the next waypoint corrected this error before the drone got lost.

Fig. 16. *Aerospace Engineering* indoor environment in Gazebo. This environment is modeled after the area surrounding TU Delft's MAVLab.

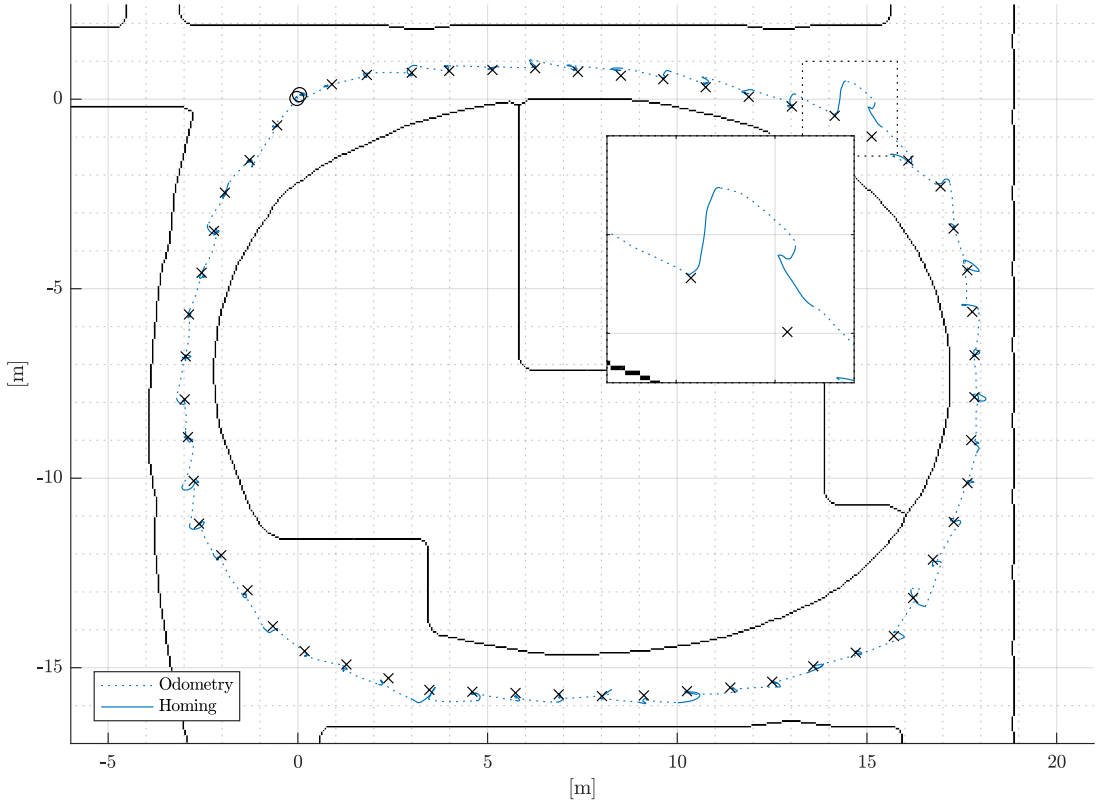


Fig. 17. Simulation results of long-range route following in the Aerospace Engineering environment (Fig. 16). The trajectory has a total length of 63 m. The drone travels counterclockwise, the black circles in the top-left corner indicate the start and end of the trajectory. Odometry and visual homing are used with waypoints at 1.20 m intervals. *Inset:* visual homing failed on one occasion, but this was corrected at the next waypoint.

VII. Discussion

The experiments of Section VI have shown that sequential visual homing, combined with odometry, is a viable way of route following in indoor environments. The short 'corridor' experiment has shown that this concept works on a real drone and the long-range experiment in the 'Aerospace Engineering' environment shows that it also works over longer distances and in more difficult environments. However, that does not mean that this route following method is without limitations. Section VII-A will discuss the limitations of Fourier-based homing. Section VII-B discusses the inaccuracies of IMU-based odometry and suggests ways to reduce its drift.

In the experiments, a memory consumption of 17.5 bytes per meter was achieved. Odometric uncertainty limited the maximum distance between waypoints. Section VII-C suggests that path following can be used to lower the memory consumption even further.

Memory consumption has been the focus of this work, as this allows visual route following to be performed on a microcontroller-equipped MAV. Computational complexity, however, has not been investigated. Section VII-D will argue that Fourier-based homing should not have problems running on a microcontroller, and that sequential visual homing with odometry is a valid solution for visual route following on MAVs.

A. Limitations of Fourier-based homing

Fourier-based homing has been shown to work in realistic environments with very small snapshots. However, it is not without limitations. Like other visual homing methods, Fourier-based homing assumes that there is sufficient texture around the UAV. Homing will not succeed in a featureless, gray room as the image difference function does not change with position and there is therefore no minimum to converge to. However, lack of texture would also cause these problems in other visual navigation methods. The use of panoramic images should at least provide some advantage to homing methods that look at the floor or ceiling, as more features are expected around the horizon.

Repetitive environments also adversely affect the shape of the image difference function, as it creates many closely-spaced local minima next to each other. In this case, odometry would not only help the drone traverse longer distances between waypoints, but it also helps the drone converge to the correct local minimum by placing it roughly besides it.

Another assumption of Fourier-based homing is that the environment is static, *including illumination*. In real environments, changes in natural light could cause problems near windows. Auto-exposure, used to prevent under- and overexposure of parts of the image, can also cause pixel intensities to change. On the other hand, a lack of auto-exposure could cause saturation of pixel intensities. Light flicker (from TL or LED lighting) has also been observed to affect homing performance in test flights.

Fourier-based homing discards the mean brightness of the image and is therefore expected to cope reasonably well with changes in overall illumination brightness. Changes in camera gain or illumination direction, however, are not canceled by this operation and could have a large effect on the homing performance. Visual navigation using keypoint detectors and descriptors might be more reliable under these circumstances as descriptors like SIFT and D-BRIEF tend to be robust against illumination changes, but feature-based homing consumes more memory. Instead, part of the problem might be solved by preprocessing the images before they are transformed to the frequency domain. In [8], [25] histogram equalization is used to control the brightness and contrast of the image, but its influence on homing performance is not investigated. Histogram equalization might even cause pixel intensities of parts of the environment to unintentionally change, thereby actually hindering homing performance. Papers [5], [9], [31], on the other hand, do not use histogram equalization. Zhang and Kleeman use patch normalization in [13] to support visual route following in mixed indoor/outdoor environments under varying lighting conditions. While this works with raw images, it is not known how this operation would affect the Fourier-transform of the image. By equalizing small patches in the image, large bright and dark regions in the image are replaced by smaller, highly contrasted regions that all have an equal average brightness. Additionally, noise is amplified in uniform regions. Neither of these effects is expected to have a positive effect on the homing performance of Fourier-based homing, as this method primarily relies on the *low-frequency* content of the image.

B. Odometry and measurement bias

Fourier-based homing is combined with odometry to increase the distance between waypoints, thereby increasing the sparsity of the map. The odometry method used in this work is extremely simplistic: estimation of the velocity only requires scaling and low-pass filtering of the accelerometer measurements. However, this method is not without problems, as demonstrated in the 'corridor' test in the Cyberzoo where the largest deviations from the planned path occur whilst flying with odometric guidance. Firstly, this method measures the the airspeed of the drone, not its speed relative to the ground. This means that deviations caused by wind or other airflows can not be measured or corrected. The main cause of error, however, is the bias of the accelerometer. It is not constant but drifts slowly over time and is therefore difficult to correct.

Part of the bias drift comes from the temperature of the sensor. Calibration of the temperature response is essential to bring the bias drift to usable levels. The sensor on the AR.Drone 2.0 was calibrated by hovering in one position using the Optitrack system until the battery was empty after about fifteen minutes. The temperature and accelerometer readings were logged and used to construct a lookup-table of bias values. After this correction, the odometric drift stayed within $20\,\mathrm{cm/s}$ during hover. While this is quite decent for IMU-only odometry, this bias is still significant in corridors that are only two to three meters wide; within a few seconds

the drone would drift so much that it collides with the wall.

This problem could be reduced if the bias can be estimated and corrected *during* the flight. This requires the velocity of the drone to be known. One way to achieve this is to hover at intermediate snapshots. Since the drone is stationary, its velocity is zero, so any deviations in the velocity estimate can then be assumed to come from sensor bias.

Alternatively, the bias might be estimated once the drone detects arrival at a waypoint. The odometry vectors recorded during the outbound and inbound flight should sum to zero (assuming that the drone's heading did not change). The remaining odometry vector could then be assumed to be a result of bias, and divided by time to find the average velocity error during the outbound- and inbound flight. This does, however, assume that the bias during route following is the same as during recording, which might not be true if a large amount of time has passed.

If the bias is difficult to correct, its influence might at least be avoided by *flying faster*. The odometric uncertainty caused by bias grows linearly with time, therefore less time spent between waypoints means a smaller error in the position estimate. The speed of the drone is ultimately bounded by the sensors and control loop: large overshoots must be avoided, but the IMU-based speed estimate is noisy so the differential gain K_d can not be set too high. Additionally, the shutter time of the camera might place an upper bound on the speed of the drone: the images should not become (too) blurry.

Regardless of the underlying method, any reduction in odometric error improves the memory efficiency of the map, as the distance between waypoints can be increased.

C. Further reductions in memory consumption

Odometry was used to travel between waypoints because it does not depend on the environment and is therefore always applicable. However, other methods of navigation could also be used to travel towards the next waypoint and might actually be preferable in terms of reliability and/or memory consumption. Corridor following, for instance, does not need to store any data and can be performed over arbitrary distances as long as the end of the corridor can reliably be detected. It does, however, require the drone to be inside a corridor which makes it less generic than odometry. To truly minimize the memory consumption of navigation, ad-hoc solutions like these should be used to minimize the number of waypoints that need to be stored and to maximize the distance between them.

D. Fourier-based homing on microcontrollers

Minimization of memory consumption is essential for the use of visual route following on microcontroller-equipped MAVs. This work has proposed an approach to visual route following that should be simple enough to run on such a platform. In terms of memory consumption this is clearly the case, but the computational complexity of the algorithm was not evaluated. Fourier-based homing has few computationally intensive steps, though. The Fourier transform of the horizon can be performed efficiently on microcontrollers; the FFT is commonly used in Digital Signal Processing and efficient implementations are therefore widely available. Fourier-based homing also requires a matrix inversion, but since this matrix is only 3×3 no problems are expected here either. Stürzl and Mallot also published an efficient relative orientation estimation that avoids the need to inefficiently search through all possible orientations. The complexity of the other operations grows linearly with the number of coefficients K, which is very low in practice. Other problems might be caused by the lack of a Floating Point Unit (FPU) on some microcontrollers. The Fourier-transformed images, however, are already stored in a fixed-point format to reduce their size, so rounding errors in the stored images have already been shown to have little influence on the homing performance. The influence of rounding errors in intermediate steps was not investigated and is left for future work.

Since both memory consumption and computational complexity have been reduced to the point that they can be run on a microcontroller, this work has succeeded in its goal of bringing visual route following to Micro Aerial Vehicles.

VIII. Conclusion

This paper has presented a route following method that focuses on low memory consumption. This allows navigation to be performed as a lightweight background process, or to be performed on simple microcontrollers, allowing even the tiniest drones to navigate over long distances in indoor environments.

Three image-based homing methods were compared, of which Fourier-based homing by Stürzl and Mallot [5] was shown to be the most memory efficient. Fourier-based homing was implemented and tested in simulation and on a real quadrotor. In both cases it was able to successfully guide the drone back to a target location with snapshots compressed to 16 bytes. The results of Section III suggest that the size of the snapshot could be reduced further to only 8–12 bytes.

Sequential homing between waypoints is a viable method of long-range navigation, but it requires each waypoint to lie inside the catchment area of the next. Odometry can be used to travel towards the next waypoint before homing is attempted. This concept was tested on an AR.Drone 2.0 in a short corridor environment. A memory consumption of 17.5 bytes per meter was achieved, with $1.20\,\mathrm{m}$ between snapshots. The same concept was successfully demonstrated over longer distances in simulation.

Appendix
Fourier-based homing

In Section III, Fourier-based homing is found to be the most memory-efficient image-based homing method evaluated in this work. Its ability to work with heavily compressed images is essential in bringing down the memory consumption of long-range visual navigation. Therefore, this section will briefly summarize the work of Stürzl and Mallot [5].

Search-based homing, from which Fourier-based homing originates, finds a homing vector $\boldsymbol{h} = (\xi, \eta)$ and rotation ς that minimizes the difference between the rotated target

image $I^T(\beta|-\varsigma)$ and the predicted image $\hat{I}(\beta|\xi,\eta)$ (Fig. 18), where $I(\beta)$ is the intensity of the pixel at bearing β in image I and where $\xi = x/R$ and $\eta = y/R$ with R assumed known:

$$E(\xi,\eta,\varsigma) = \frac{1}{2}\sum_{\beta}\left(\hat{I}(\beta|\xi,\eta) - I^T(\beta|-\varsigma)\right)^2 \quad (6)$$

$$(\boldsymbol{h},\varsigma) = \underset{(\xi,\eta),\varsigma}{\arg\min}\, E(\xi,\eta,\varsigma) \quad (7)$$

Unlike search-based homing, Fourier-based homing uses the Fourier-transform of the horizon images. The images are approximated using only the first K low-frequency components:

$$I(\beta) \approx \frac{1}{2}a_0 + \sum_{k=1}^{K} a_k \cos k\beta + b_k \sin k\beta \quad (8)$$

where $K \leq w/2$ with w the width of the original horizon image in pixels. Through Parseval's theorem, the error between the Fourier-transformed predicted and target images can be found directly from the coefficients a_k, b_k:

$$\mathcal{E}(\xi,\eta,\varsigma) = \frac{1}{2}\sum_{k=1}^{K}(\hat{a}_k(\xi,\eta) - a_k^T(-\varsigma))^2 + (\hat{b}_k(\xi,\eta) - b_k^T(-\varsigma))^2 \quad (9)$$

The DC value $\hat{a}_0$ (and a_0^T) is excluded from this error as it does not depend on ξ, η or ς and therefore does not provide any directional information.

The predicted image $\hat{I}(\beta|\xi,\eta)$, defined by its coefficients $\hat{a}_k(\xi,\eta)$, $\hat{b}_k(\xi,\eta)$, is replaced by its first-order Taylor approximation:

$$\hat{a}_k(\xi,\eta) = a_k^C + a_{k,x}^C\xi + a_{k,y}^C\eta \quad (10)$$

$$\hat{b}_k(\xi,\eta) = b_k^C + b_{k,x}^C\xi + b_{k,y}^C\eta \quad (11)$$

where $a_{k,x}^C$, $a_{k,y}^C$, $b_{k,x}^C$, $b_{k,y}^C$ are linear combinations of the coefficients of the current image:

$$\begin{aligned} a_{k,x}^C &= \frac{1}{2}\left(-(k-1)a_{k-1}^C + (k+1)a_{k+1}^C\right) \\ a_{k,y}^C &= \frac{1}{2}\left((k-1)b_{k-1}^C + (k+1)b_{k+1}^C\right) \\ b_{k,x}^C &= \frac{1}{2}\left(-(k-1)b_{k-1}^C + (k+1)b_{k+1}^C\right) \\ b_{k,y}^C &= \frac{1}{2}\left(-(k-1)a_{k-1}^C - (k+1)a_{k+1}^C\right) \end{aligned} \quad (12)$$

Similarly, the relative orientation $\varsigma = \varsigma_0 + \Delta\varsigma$ of the target image is replaced by its first order Taylor approximation:

$$\begin{aligned} a_k^T(\varsigma) &\approx a_k^T(\varsigma_0) + b_k^T(\varsigma_0)k\Delta\varsigma \\ b_k^T(\varsigma) &\approx b_k^T(\varsigma_0) - a_k^T(\varsigma_0)k\Delta\varsigma \end{aligned} \quad (13)$$

with $a_k(\varsigma) = a_k\cos k\varsigma + b_k \sin k\varsigma$ and $b_k(\varsigma) = b_k\cos k\varsigma - a_k \sin k\varsigma$. The coarse orientation ς_0 is estimated beforehand using the phase information of the low-frequency components. The relative orientation is refined by estimating a correction $\Delta\varsigma$ during relative pose estimation below.

The linear approximations of the coefficients of $\hat{I}(\xi,\eta)$ (11) and the rotated target image (13) can then be inserted into (9) to produce an error function that is quadratic in ξ, η, $\Delta\varsigma$. The minimum of the error function can be found by setting the gradient of (9), $\nabla_{\xi,\eta,\Delta\varsigma}\mathcal{E}(\xi,\eta,\varsigma_0+\Delta\varsigma)$, which is affine in $\xi,\eta,\Delta\varsigma$ to zero through a 3×3 matrix inversion.

$$(\boldsymbol{h},\varsigma) = \underset{((\xi,\eta),\varsigma_0+\Delta\varsigma)}{\arg\min}\, \frac{1}{2}\sum_{k=1}^{K}(\hat{a}_k(\xi,\eta) - a_k^T(-\varsigma))^2 + (\hat{b}_k(\xi,\eta) - b_k^T(-\varsigma))^2 \quad (14)$$

The homing vector can be found directly from (14) or estimated over multiple iterations where the warped version of the current image is used as the starting point for the next step.

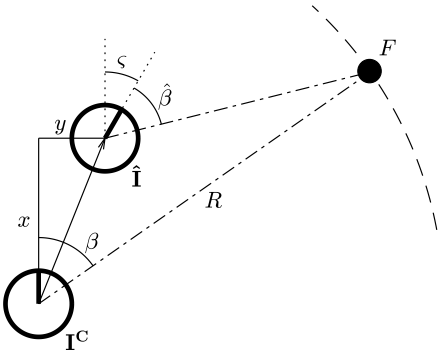


Fig. 18. Image warping [8]. After a small movement $(x,\ y)$ and rotation ς, the new bearing $\hat{\beta}$ towards feature F can be predicted, assuming that distance R is known. During image warping, the intensities of all pixels in the predicted image $\hat{I}$ at bearings $\hat{\beta}$ are sampled from the current image I^C at bearings β, where $\beta = \hat{\beta} + \varsigma - \sin^{-1}\left(\frac{x}{R}\sin(\hat{\beta}+\varsigma) - \frac{y}{R}\cos(\hat{\beta}+\varsigma)\right)$ [5]. Distance R is known and constant under the equal distance assumption [8].

REFERENCES

[1] 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*, vol. 31, no. 4, pp. 537–570, 2014.

[2] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard MAVs in unknown environments," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 1872–1878, 2015.

[3] B. A. Cartwright and T. S. Collett, "Landmark Learning in Bees Experiments and Models," *Journal of Comparative Physiology*, vol. 151, pp. 521–543, 1983.

[4] D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, and R. Wehner, "A mobile robot employing insect strategies for navigation," *Robotics and Autonomous Systems*, vol. 30, no. 1, pp. 39–64, 2000.

[5] W. Stürzl and H. A. Mallot, "Efficient visual homing based on Fourier transformed panoramic images," *Robotics and Autonomous Systems*, vol. 54, no. 4, pp. 300–313, 2006.

[6] 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, may 2016, pp. 3455–3462.

[7] M. O. Franz and H. A. Mallot, "Biomimetic robot navigation," *Robotics and Autonomous Systems*, vol. 30, no. 1, pp. 133–153, 2000.

[8] 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*, vol. 79, no. 3, pp. 191–202, 1998.

[9] R. Möller and A. Vardy, "Local visual homing by matched-filter descent in image distances," *Biological Cybernetics*, vol. 95, no. 5, pp. 413–430, oct 2006.
[10] A. Vardy, "Long-range visual homing," *2006 IEEE International Conference on Robotics and Biomimetics, ROBIO 2006*, pp. 220–226, 2006.
[11] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, "MAV navigation through indoor corridors using optical flow," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3361–3368, 2010.
[12] C. Bills, J. Chen, and A. Saxena, "Autonomous MAV flight in indoor environments using single image perspective cues," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5776–5783, 2011.
[13] 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*, vol. 28, no. 3, pp. 331–356, 2009.
[14] Y. Matsumoto, M. Inaba, and H. Inoue, "Visual navigation using view-sequenced route representation," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 1996, pp. 83–88.
[15] B. Baddeley, P. Graham, P. Husbands, and A. Philippides, "A model of ant route navigation driven by scene familiarity," *PLoS Computational Biology*, vol. 8, no. 1, 2012.
[16] P. De Cristóforis, M. Nitsche, T. Krajník, T. Pire, and M. Mejail, "Hybrid vision-based navigation for mobile robots in mixed indoor/outdoor environments," *Pattern Recognition Letters*, vol. 53, pp. 118–128, 2015.
[17] F. Labrosse, "Short and long-range visual navigation using warped panoramic images," *Robotics and Autonomous Systems*, vol. 55, no. 9, pp. 675–684, 2007.
[18] E. Mair, M. Augustine, B. Jäger, A. Stelzer, C. Brand, D. Burschka, and M. Suppa, "A biologically inspired navigation concept based on the Landmark-Tree map for efficient long-distance robot navigation," *Advanced Robotics*, vol. 28, no. 5, pp. 289–302, 2014.
[19] J. Courbon, Y. Mezouar, N. Guénard, and P. Martinet, "Vision-based navigation of unmanned aerial vehicles," *Control Engineering Practice*, vol. 18, no. 7, pp. 789–799, 2010.
[20] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, nov 2004.
[21] T. Trzcinski and V. Lepetit, "Efficient Discriminative Projections for Compact Binary Descriptors," in *European Conference on Computer Vision*. Springer, 2012, pp. 228–242.
[22] M. Augustine, F. Ortmeier, E. Mair, D. Burschka, A. Stelzer, and M. Suppa, "Landmark-Tree map: A biologically inspired topological map for long-distance robot navigation," in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, dec 2012, pp. 128–135.
[23] A. Stelzer, E. Mair, and M. Suppa, "Trail-Map: A scalable landmark data structure for biologically inspired range-free navigation," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. IEEE, dec 2014, pp. 2138–2145.
[24] J. Zeil, M. I. Hofmann, and J. S. Chahl, "Catchment areas of panoramic snapshots in outdoor scenes," *Journal of the Optical Society of America A*, vol. 20, no. 3, pp. 450–469, 2003.
[25] D. D. Gaffin and B. P. Brayfield, "Autonomous Visual Navigation of an Indoor Environment Using a Parsimonious, Insect Inspired Familiarity Algorithm," *Plos One*, vol. 11, no. 4, p. e0153706, 2016.
[26] R. Möller, A. Vardy, S. Kreft, and S. Ruwisch, "Newton-based matched-filter descent in image distances," *Biological Cybernetics (Submitted)*, 2006.
[27] O. J. Woodman, "An Introduction to Inertial Navigation," *University of Cambridge*, no. 696, pp. 1–37, 2007.
[28] P. J. Bristeau, F. Callou, D. Vissière, and N. Petit, *The Navigation and Control technology inside the AR.Drone micro UAV*. IFAC, 2011, vol. 18, no. PART 1.
[29] R. C. Leishman, J. C. MacDonald, R. W. Beard, and T. W. McLain, "Quadrotors and accelerometers: State estimation with an improved dynamic model," *IEEE Control Systems*, vol. 34, no. 1, pp. 28–41, 2014.
[30] E. J. J. Smeur, Q. Chu, and G. C. H. E. de Croon, "Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 3, pp. 450–461, 2016.
[31] A. Vardy and R. Möller, "Biologically plausible visual homing methods based on optical flow techniques," *Connection Science*, vol. 17, no. 1-2, pp. 47–89, 2005.

Appendices

Appendix A

Review of image-based homing methods

Nomenclature

a_k, b_k	Coefficients of Fourier-transformed image [-]
$\mathcal{A}_k$	Amplitude of the k-th frequency component [-]
A, $\boldsymbol{b}$, c	Coefficients of $\mathcal{E}(\xi, \eta, \Delta\varsigma\|\varsigma_0)$ [-]
β	Bearing relative to current heading [rad]
$E(\xi, \eta, \varsigma)$	Error between predicted image $\hat{I}(\beta\|\xi, \eta, \varsigma)$ and target image $I^T(\beta)$ [-]
$\mathcal{E}(\xi, \eta, \Delta\varsigma\|\varsigma_0)$	Quadratic approximation of $E(\xi, \eta, \varsigma)$
ϕ_k	Phase of the k-th frequency component [rad]
$\boldsymbol{h}$	Homing vector [-]
$I^C(\beta)$	Pixel intensity in current image [-]
$I^T(\beta)$	Pixel intensity in target image [-]
$\hat{I}(\beta\|\xi, \eta, \varsigma)$	Pixel intensity in predicted image [-]
$\nabla_\beta I^C(\beta)$	Gradient of current image [rad^{-1}]
k	Frequency [-]
K	Number of frequencies in snapshot [-]
R	Environment radius [m]
ς	Relative orientation [rad]
ς_0	Relative orientation (coarse) [rad]
ς_k	Relative orientation at frequency k [rad]
$\Delta\varsigma$	Relative orientation (correction) [rad]
w	Image width [px]
w_k	Weight of orientation estimate at frequency k [-]
x, y	Relative position [m]
ξ, η	Relative position, unknown scale [-]

In Section 3 of the paper, the performance of three image-based homing methods was compared: search-based homing [1], Matched Filter Descent in Image Distances (MFDID) [2] and Fourier-based homing [3]. These methods were only briefly mentioned in Section 2, but the underlying equations of search-based homing and MFDID were not presented. For the

completeness of this thesis and to ensure its reproducibility, this appendix summarizes the mathematics behind image warping, search-based homing, MFDID and Fourier-based homing.

The following sections summarize existing homing methods for the completeness of this thesis. The methods presented here are not my own work.

A-1 Image warping (Franz et al. 1998) [1, 3]

While image-based homing can be performed with exploratory movements, these are difficult to perform accurately on a UAV and could use a considerable amount of energy. Instead, nearby images are predicted through a procedure called *image warping*. These predicted images can then be used to search for the target location. The idea of image warping was originally published by Franz et al. as part of their homing method [1]. This section, however, will summarize image warping under a Cartesian movement (x, y) as described in [3] as this formulation is easier to integrate into the other two homing methods.

Images are modeled as a function $I(\beta)$, where β is the bearing of a pixel with intensity $I(\beta)$. To minimize memory consumption, only one-dimensional horizon images will be considered here.

The goal of image warping is to predict a new image $\hat{I}(\beta)$ that would be expected after a hypothetical movement (x, y) and rotation ς from the current position (Figure A-1). As the name suggests, this predicted image is found by warping the image $I^C(\beta)$ observed at the current location. For each bearing $\hat{\beta}$ in the predicted image, a corresponding bearing β in the current image $I^C(\beta)$ can be found:

$$\beta = \hat{\beta} + \varsigma - \sin^{-1}\left(\frac{x}{R}\sin(\hat{\beta}+\varsigma) - \frac{y}{R}\cos(\hat{\beta}+\varsigma)\right) \tag{A-1}$$

However, this requires the distance R to all features to be known. Franz et al. assumed this distance to be known and equal for all features, the *equal distance assumption* [1], this assumption is also made by the other methods under review in this section [2, 3]. Under the equal distance assumption, (A-1) can be expressed in the unscaled coordinates $\xi = x/R$, $\eta = y/R$. For small movements $\xi^2 + \eta^2 \ll 1$, (A-1) can then be approximated as follows:

$$\beta \approx \hat{\beta} + \varsigma - \xi\sin(\hat{\beta}+\varsigma) + \eta\cos(\hat{\beta}+\varsigma) \tag{A-2}$$

And therefore:

$$\hat{I}(\beta|\xi,\eta,\varsigma) = I^C(\beta + \varsigma - \xi\sin(\beta+\varsigma) + \eta\cos(\beta+\varsigma)) \tag{A-3}$$

This predicted image is used in all of the following methods.

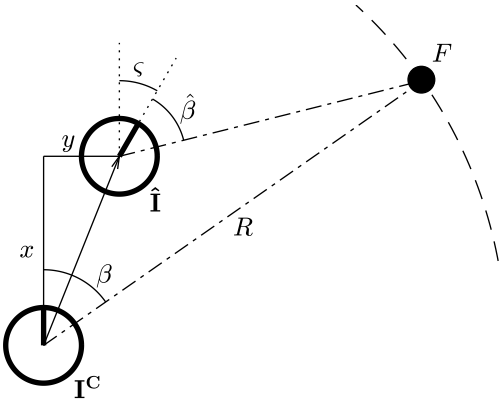


Figure A-1: Image warping under the equal distance assumption. When the distance R to a feature F at bearing β is known, it is possible to predict its new bearing $\hat{\beta}$ after a small movement (x, y) and rotation ς. During image warping, the intensities of all pixels in the predicted image $\hat{I}$ at bearings $\hat{\beta}$ are sampled from the current image I^C at bearings β.

A-2 Search-based homing (Franz et al. 1998) [1]

Search-based homing was introduced by Franz et al. in [1] as an alternative to existing feature-based homing methods that required an equal distribution of landmarks. To estimate the homing vector, the authors used the warping procedure described above to produce a large number of predicted images for all hypothetical displacements ξ, η and rotations ς under consideration. These predicted images are then compared to the target image $I^T(\beta)$ to obtain an error $E(\xi, \eta, \varsigma)$:

$$E(\xi, \eta, \varsigma) = \frac{1}{2} \sum_{\beta} \left(\hat{I}(\beta | \xi, \eta, \varsigma) - I^T(\beta) \right)^2 \tag{A-4}$$

The hypothetical movement that results in the smallest error is then selected as the homing vector $\boldsymbol{h}$:

$$(\boldsymbol{h}, \varsigma) = \underset{((\xi,\eta),\varsigma)}{\arg\min}\, E(\xi, \eta, \varsigma) \tag{A-5}$$

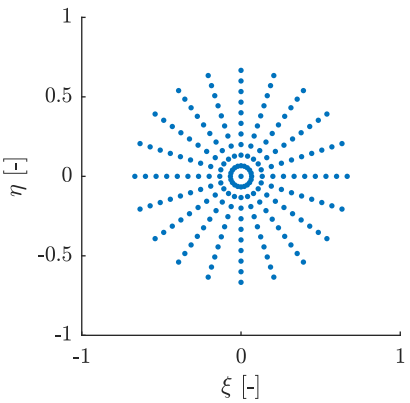


Figure A-2: Sampling points for search-based homing used in this thesis.

In this thesis, ξ, η are radially sampled at twenty directions from the current position and at ten distances up to $\frac{2}{3}$ together with the point (0, 0) (Figure A-2). While search-based homing typically produces good results, it has the disadvantage that a large number of images need to be predicted which makes it computationally intensive compared to the two other methods summarized next.

A-3 Matched Filter Descent in Image Distances (Möller and Vardy, 2006) [2]

Instead of directly searching for the best match with the target image, Möller and Vardy's Matched Filter Descent in Image Distances (MFDID) uses the (negative) spatial gradient of the error $E(\xi, \eta)$ as the homing vector. As long as the UAV follows this vector, the error will decrease until it is minimized when the drone is at the target location (provided that it does not get stuck in a local minimum of $E(\xi, \eta)$).

Unlike search-based homing, MFDID does not include the orientation ς in its relative pose estimation. The orientation therefore has to be found and corrected beforehand. The relative orientation is found by searching for a rotated version of the current image that has the lowest difference to the target image:

$$\varsigma = \arg\min_{\varsigma} \frac{1}{2} \sum_{\beta} \left(\hat{I}(\beta|\varsigma) - I^T(\beta) \right)^2 \tag{A-6}$$

Because the images are aligned beforehand, the relative orientation ς is not included in the pose estimation below.

Instead of searching through all possible movements, MFDID estimates the spatial gradient of the error at the current position $(\xi, \eta) = (0, 0)$ and uses this as the homing vector $\boldsymbol{h}$:

$$\boldsymbol{h} = -\boldsymbol{\nabla}_{\xi,\eta} E(0,0) \tag{A-7}$$

$$= -\sum_{\beta} \left(I^C(\beta) - I^T(\beta) \right) \cdot \boldsymbol{\nabla}_{\xi,\eta} \hat{I}(\beta|0,0) \tag{A-8}$$

To find the spatial gradient $\boldsymbol{\nabla}_{\xi,\eta} \hat{I}(\beta|0,0)$, Möller and Vardy use the following procedure: Consider the pixel intensity $\hat{I}(\beta + \Delta\beta|\xi, \eta)$ sampled from the predicted image. For small $\Delta\beta$ and $\xi^2 + \eta^2$, the intensity of this pixel can be found from the first-order Taylor approximation:

$$\hat{I}(\beta + \Delta\beta|\xi,\eta) \approx I^C(\beta) + \nabla_\beta^\top I^C(\beta)\Delta\beta + \boldsymbol{\nabla}_{\xi,\eta}^\top \hat{I}(\beta|0,0) \begin{bmatrix} \xi & \eta \end{bmatrix}^\top \tag{A-9}$$

When $\Delta\beta$ is chosen such that the same feature is in view in both images, the pixel intensities should be equal:

$$I^C(\beta) + \nabla_\beta^\top I^C(\beta)\Delta\beta + \boldsymbol{\nabla}_{\xi,\eta}^\top \hat{I}(\beta|0,0) \begin{bmatrix} \xi & \eta \end{bmatrix}^\top \approx I^C(\beta) \tag{A-10}$$

and therefore:

$$\boldsymbol{\nabla}_{\xi,\eta}^\top \hat{I}(\beta|0,0) \begin{bmatrix} \xi & \eta \end{bmatrix}^\top \approx -\nabla_\beta^\top I^C(\beta)\Delta\beta \tag{A-11}$$

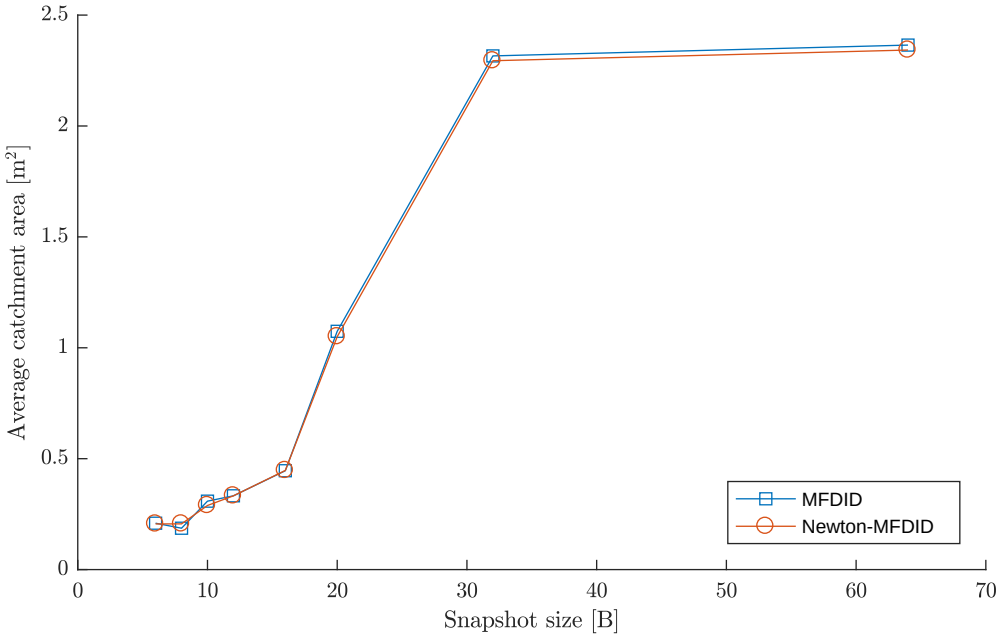


Figure A-3: Average catchment area vs. snapshot size for MFDID [2] and Newton-MFDID [4].

Factors $\Delta\beta/\xi$ and $\Delta\beta/\eta$ are then obtained from an optic flow equation for two dimensional images. In this review, this equation is replaced by warping equation (A-2) (with $\Delta\beta = \hat{\beta} - \beta$), producing the same result:

$$\boldsymbol{\nabla}_{\xi,\eta}\hat{I}(\beta|0,0) = \nabla_{\beta}^{\top} I^{C}(\beta) \begin{bmatrix} -\sin\beta & \cos\beta \end{bmatrix}^{\top} \tag{A-12}$$

This spatial gradient can then be inserted into (A-8) to find the homing vector $\boldsymbol{h}$:

$$\boldsymbol{h} = \sum_{\beta} \begin{bmatrix} \sin\beta & -\cos\beta \end{bmatrix}^{\top} \nabla I^{C}(\beta) \left(I^{C}(\beta) - I^{T}(\beta) \right) \tag{A-13}$$

The gradient image $\nabla I^{C}(\beta)$ can be obtained by filtering the current image using a simple kernel like $[-1\,0\,1]/2$.

As an additional preprocessing step, Möller and Vardy apply a low-pass filter to the images. Low-pass filtering of the images is observed to improve the homing performance as long as the cut-off frequency is not set too low. In this thesis, the low-pass filtering is applied to a $1024\,px$ horizon image before it is subsampled to the low resolutions used in the comparison of Section 3 of the paper. The cut-off frequency of the low-pass filter was optimized for each snapshot size.

In [4], Möller et al. noted that in anisotropic environments, the spatial gradient of the error does not always point straight towards the target location. Inspired by Newton's method for optimization, the authors correct for this effect using the Hessian of the error surface evaluated at the target position. The homing performance of MFDID was evaluated with and without this Hessian correction where the best results were kept for the comparison of Section 3. As shown in Figure A-3, there is little to no difference between the performance of MFDID and Newton-MFDID when applied to this dataset.

A-4 Fourier-based homing (Stürzl and Mallot, 2006) [3]

In [3], Stürzl and Mallot transform search-based homing to the frequency domain. An image $I(\beta)$ is approximated by its first K frequency components:

$$I(\beta) \approx \frac{1}{2}a_0 + \sum_{k=1}^{K} a_k \cos k\beta + b_k \sin k\beta \tag{A-14}$$

where $K \leq w/2$ with w the width of the original horizon image in pixels.

Through Parseval's theorem, the error between the predicted and target images (A-4) can be found directly from the coefficients a_k, b_k of the two images:

$$\begin{aligned}\mathcal{E}(\xi,\eta,\varsigma) = \frac{1}{2}\sum_{k=1}^{K} &(\hat{a}_k(\xi,\eta) - a_k^T(-\varsigma))^2 \\ &+(\hat{b}_k(\xi,\eta) - b_k^T(-\varsigma))^2\end{aligned} \tag{A-15}$$

where coefficients $a_k^T(\varsigma)$ and $b_k^T(\varsigma)$ form a rotated version of the target image $I^T(\beta)$:

$$\begin{aligned}a_k^T(\varsigma) &= a_k^T \cos k\varsigma + b_k^T \sin k\varsigma \\ b_k^T(\varsigma) &= b_k^T \cos k\varsigma - a_k^T \sin k\varsigma\end{aligned} \tag{A-16}$$

Note that the DC values a_0 are not included in error $\mathcal{E}(\xi,\eta,\varsigma)$ as they do not depend on bearing β and therefore do not provide any directional information.

Estimation of the homing vector is performed in two steps: first, as in MFDID the current and target images are rotationally aligned. This alignment is performed using the phase of the frequency components of the image, resulting in a coarse estimate of the relative orientation ς_0. Then, the pose of the rotated target image relative to the current image is estimated. Unlike MFDID, however, this pose estimation also includes a correction $\Delta\varsigma$ of the orientation estimate.

Coarse orientation estimation Fourier-based homing relies on the phase information of the frequency components to estimate the orientation of the target image relative to the current image. Each frequency component has an amplitude $\mathcal{A}_k$ and phase ϕ_k:

$$\mathcal{A}_k = \sqrt{a_k^2 + b_k^2}, \quad \phi_k = \text{atan2}(b_k, a_k) \tag{A-17}$$

A first guess of the orientation ς_0 is made using the phase of the first component:

$$\bar{\varsigma_1} = \varsigma_1 = \phi^C - \phi^T \tag{A-18}$$

with weight

$$w_1 = \mathcal{A}_1^C \mathcal{A}_1^T \tag{A-19}$$

This initial estimate is then refined by repeating the following steps for $k = 2 \dots K$:

1. $n_k = \text{round}\left(\frac{\phi^T - \phi^C + k\bar{\varsigma}_{k-1}}{2\pi}\right)$

2. $\varsigma_k = \frac{\phi^C - \phi^T + 2\pi n_k}{k}$
3. $w_k = \mathcal{A}_k^C \mathcal{A}_k^T k^2$
4. $\bar{\varsigma}_k = \frac{\sum_{l=1}^{k} w_l \varsigma_l}{\sum_{l=1}^{k} w_l}$

The final orientation estimate ς_0 is found by taking the weighted average of the orientation estimates ς_k:

$$\varsigma_0 = \frac{\sum w_k \varsigma_k}{\sum w_k} \tag{A-20}$$

This procedure works as long as the amplitude of the first frequency component is large enough, because it cannot recover from an initial guess that is more than 180° off. When w_1 is small, Stürzl and Mallot also start a second estimation procedure from angle $\bar{\varsigma}_1 = \varsigma_1 + \pi$.

However, because the coefficients of the image are rounded to 8-bit fixed point numbers, cases where w_1 and w_2 were exactly zero were occasionally encountered during early test flights performed for this thesis, leading to divide-by-zero errors in the fourth step of the algorithm. To solve this problem and to increase the accuracy of the orientation estimate, the heading of the drone is stored in an additional byte in the snapshot and used to provide an initial guess of the relative orientation that is averaged with ς_1. The weight of the drone's heading estimate was set to $w_\psi = 5000$. With a nonzero weight for the drone's heading, the weight of the initial guess w_1 can never become zero.

Relative pose estimation Once the images have been aligned, Fourier-based homing aims to find ξ, η, $\Delta\varsigma$ that minimize the error $\mathcal{E}(\xi, \eta, \varsigma)$. However, unlike search-based homing it does not search through all possible movements. Instead, the error $\mathcal{E}(\xi, \eta, \varsigma)$ is approximated as a quadratic function of which the minimum can be found directly. To form this quadratic approximation, the predicted image $\hat{I}(\beta|\xi, \eta)$ is replaced by its first-order Taylor approximation.

$$\hat{I}(\beta|\xi, \eta) \approx I^C(\beta) + \boldsymbol{\nabla}_{\xi,\eta}^\top \hat{I}(\beta|0, 0) \cdot \begin{bmatrix} \xi & \eta \end{bmatrix}^\top \tag{A-21}$$

The spatial gradient $\boldsymbol{\nabla}_{\xi,\eta} \hat{I}(\beta|0, 0)$ is found using (A-3) and the chain rule:

$$\boldsymbol{\nabla}_{\xi,\eta}^\top \hat{I}(\beta|0, 0) = \boldsymbol{\nabla}_{\xi,\eta}^\top \Big(I^C(\beta - \xi \sin\beta + \eta \cos\beta) \Big) \tag{A-22}$$

$$= \nabla_\beta I^C(\beta) \begin{bmatrix} -\sin\beta & \cos\beta \end{bmatrix} \tag{A-23}$$

where the gradient image $\nabla_\beta I^C(\beta)$ is found through:

$$\nabla_\beta I^C(\beta) = \sum_{k=1}^{K} -k a_k^C \sin k\beta + k b_k^C \cos k\beta \tag{A-24}$$

The spatial gradient (A-23) is inserted into (A-21) to produce:

$$\hat{I}(\beta|\xi, \eta) \approx I^C(\beta) + \nabla_\beta I^C(\beta) \cdot (-\xi \sin\beta + \eta \cos\beta) \tag{A-25}$$

Which, when written out, becomes:

$$\hat{I}(\beta|\xi,\eta) = \frac{1}{2}a_0^C + \sum_{k=1}^{K}\Big(a_k^C\cos k\beta + b_k^C\sin k\beta + \\ (-ka_k^C\sin k\beta + kb_k^C\cos k\beta)(-\xi\sin\beta + \eta\cos\beta)\Big) \quad \text{(A-26)}$$

Using the trigonometric product identities, the final product can be rewritten as:

$$\begin{aligned} -ka_k^C\sin k\beta\cdot -\xi\sin\beta &= -ka_k^C\frac{\cos\left((k-1)\beta\right)-\cos\left((k+1)\beta\right)}{2}\xi \\ -ka_k^C\sin k\beta\cdot \eta\cos\beta &= -ka_k^C\frac{\sin\left((k+1)\beta\right)+\sin\left((k-1)\beta\right)}{2}\eta \\ kb_k^C\cos k\beta\cdot -\xi\sin\beta &= -kb_k^C\frac{\sin\left((k+1)\beta\right)-\sin\left((k-1)\beta\right)}{2}\xi \\ kb_k^C\cos k\beta\cdot \eta\cos\beta &= kb_k^C\frac{\cos\left((k+1)\beta\right)+\cos\left((k-1)\beta\right)}{2}\eta \end{aligned} \quad \text{(A-27)}$$

After shifting the indices such that all sines and cosines are expressed in $k\beta$ and using the fact that a_k, b_k were assumed zero for $k > K$, (A-27) can be rewritten in the form:

$$\hat{I}(\beta|\xi,\eta) \approx \frac{1}{2}a_0^C + \sum_{k=1}^{K}\hat{a}_k(\xi,\eta)\cos k\beta + \hat{b}_k(\xi,\eta)\cos k\beta \quad \text{(A-28)}$$

with

$$\begin{aligned} \hat{a}_k(\xi,\eta) &= a_k^C + a_{k,x}^C\xi + a_{k,y}^C\eta \\ \hat{b}_k(\xi,\eta) &= b_k^C + b_{k,x}^C\xi + b_{k,y}^C\eta \end{aligned} \quad \text{(A-29)}$$

and where

$$\begin{aligned} a_{k,x}^C &= \frac{1}{2}\Big(-(k-1)a_{k-1}^C + (k+1)a_{k+1}^C\Big) \\ a_{k,y}^C &= \frac{1}{2}\Big((k-1)b_{k-1}^C + (k+1)b_{k+1}^C\Big) \\ b_{k,x}^C &= \frac{1}{2}\Big(-(k-1)b_{k-1}^C + (k+1)b_{k+1}^C\Big) \\ b_{k,y}^C &= \frac{1}{2}\Big(-(k-1)a_{k-1}^C + (k+1)a_{k+1}^C\Big) \end{aligned} \quad \text{(A-30)}$$

The important point here is that the coefficients $\hat{a}_k(\xi,\eta)$, $\hat{b}_k(\xi,\eta)$ of the predicted image $\hat{I}(\beta|\xi,\eta)$ are linear in ξ, η.

In a similar way, the relative orientation $\varsigma = \varsigma_0 + \Delta\varsigma$ of the target image (A-16) is replaced by its first-order Taylor approximation:

$$\begin{aligned} a_k^T(\varsigma) &\approx a_k^T(\varsigma_0) + b_k^T(\varsigma_0)k\Delta\varsigma \\ b_k^T(\varsigma) &\approx b_k^T(\varsigma_0) - a_k^T(\varsigma_0)k\Delta\varsigma \end{aligned} \quad \text{(A-31)}$$

The linear approximations of the coefficients of $\hat{I}(\beta|\xi,\eta)$ (A-28) and the rotated target image (A-31) can then be inserted into (A-15) to produce an error function that is quadratic in ξ, η, $\Delta\varsigma$:

$$\mathcal{E}(\xi,\eta,\Delta\varsigma|\varsigma_0) = \frac{1}{2}\sum_{k=1}^{K}\Big(a_k^C + a_{k,x}^C\xi + a_{k,y}^C\eta - a_k^T(-\varsigma_0) + b_k^T(-\varsigma_0)k\Delta\varsigma\Big)^2 + \Big(b_k^C + b_{k,x}^C\xi + b_{k,y}^C\eta - b_k^T(-\varsigma_0) - a_k^T(-\varsigma_0)k\Delta\varsigma\Big)^2 \quad \text{(A-32)}$$

The idea behind the use of the first-order Taylor approximation is that the error (A-32) is now quadratic in (ξ, η, $\Delta\varsigma$), which means that its minimum can be found directly without searching. Stürzl and Mallot correctly note that the gradient of the error, which should be set to zero, is a linear function of the homing vector ξ, η and $\Delta\varsigma$, but the exact form of this equation is not presented in their paper. It can be found as follows: the error is first rewritten as:

$$\mathcal{E}(\xi,\eta,\Delta\varsigma|\varsigma_0) = \frac{1}{2}\sum_{k=1}^{K}\left(\begin{bmatrix} a_{k,x}^C & a_{k,y}^C & b_k^T(-\varsigma_0)k\end{bmatrix}\begin{bmatrix}\xi\\ \eta\\ \Delta\varsigma\end{bmatrix} + a_k^C - a_k^T(-\varsigma_0)\right)^2 + \left(\begin{bmatrix} b_{k,x}^C & b_{k,y}^C & -a_k^T(-\varsigma_0)k\end{bmatrix}\begin{bmatrix}\xi\\ \eta\\ \Delta\varsigma\end{bmatrix} + b_k^C - b_k^T(-\varsigma_0)\right)^2 \quad \text{(A-33)}$$

Then, (A-33) is rewritten as the following quadratic surface:

$$\mathcal{E}(\xi,\eta,\Delta\varsigma|\varsigma_0) = \frac{1}{2}\boldsymbol{h}^\top \mathrm{A}\boldsymbol{h} + \boldsymbol{b}^\top\boldsymbol{h} + c \quad \text{(A-34)}$$

where

$$\boldsymbol{h} = \begin{bmatrix}\xi & \eta & \Delta\varsigma\end{bmatrix}^\top \quad \text{(A-35)}$$

$$\mathrm{A} = \sum_{k=1}^{K}\left(\begin{bmatrix} a_{k,x}^C\\ a_{k,y}^C\\ b_k^T(-\varsigma_0)k\end{bmatrix}\begin{bmatrix} a_{k,x}^C & a_{k,y}^C & b_k^T(-\varsigma_0)k\end{bmatrix} + \begin{bmatrix} b_{k,x}^C\\ b_{k,y}^C\\ -a_k^T(-\varsigma_0)k\end{bmatrix}\begin{bmatrix} b_{k,x}^C & b_{k,y}^C & -a_k^T(-\varsigma_0)k\end{bmatrix}\right) \quad \text{(A-36)}$$

$$\boldsymbol{b} = \sum_{k=1}^{K}\left((a_k^C - a_k^T(-\varsigma_0))\begin{bmatrix} a_{k,x}^C\\ a_{k,y}^C\\ b_k^T(-\varsigma_0)k\end{bmatrix} + (b_k^C - b_k^T(-\varsigma_0))\begin{bmatrix} b_{k,x}^C\\ b_{k,y}^C\\ -a_k^T(-\varsigma_0)k\end{bmatrix}\right) \quad \text{(A-37)}$$

$$c = \frac{1}{2}\sum_{k=1}^{K}\Big((a_k^C - a_k^T(-\varsigma_0))^2 + (b_k^C - b_k^T(-\varsigma_0))^2\Big) \quad \text{(A-38)}$$

The minimum of this error can be found by setting the gradient of the error to zero:

$$\mathrm{A}\boldsymbol{h} + \boldsymbol{b} = \boldsymbol{0} \quad \text{(A-39)}$$

The solution of which can be found through a 3×3 matrix inversion of A:

$$\boldsymbol{h} = -\mathrm{A}^{-1}\boldsymbol{b} \quad \text{(A-40)}$$

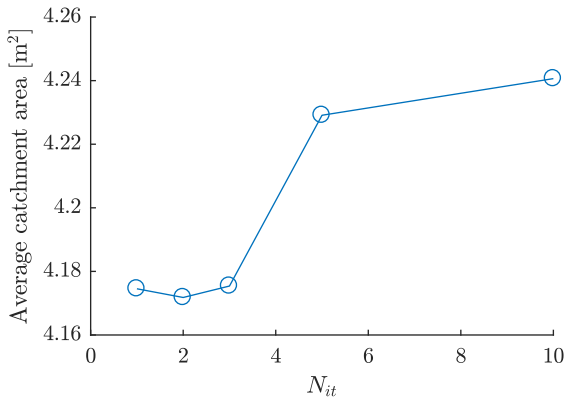


Figure A-4: Influence of the number of iterations N_{it} on the performance of Fourier-based homing. The difference is only a few square centimeters on average.

Multiple iterations The homing vector can be found directly from (A-40) or estimated over multiple iterations where the warped version of the current image is used as the starting point for the next step. The number of iterations N_{it} was optimized for the comparison. However, as shown in Figure A-4 it has very little influence on the homing performance, therefore a value of $N_{it} = 1$ was used in the thesis to avoid wasting unnecessary CPU cycles.

Coarse-to-fine homing Stürzl and Mallot observed that the catchment area is larger when a small number of frequencies is used for homing. On the other hand, the final homing error is observed to increase as well. To achieve both a large catchment area and a small final error, the authors propose a coarse-to-fine approach where homing is first performed using a small number of frequencies to bring the drone close to the target position and then followed with a larger number of frequencies to reduce the remaining error.

The authors also proposed an alternative coarse-to-fine approach to the iterations of Fourier-based homing by increasing K with each step. This latter approach would have the advantage that the drone does not have to physically converge to an intermediate location before the number of components K can be increased, but unfortunately this was not further investigated in [3].

Coarse-to-fine homing was not implemented in this thesis but could increase the size of the catchment area, thereby leading to a sparser and more memory-efficient map.

Appendix B

Online estimation of the catchment area radius

In Section 4 of the paper, it was argued that the maximum distance between waypoints depends on the odometric uncertainty and the size of the catchment area. If the maximum distance is increased, the map becomes sparser and its memory consumption decreases. While the odometric uncertainty is relatively straightforward to estimate when using IMU-based odometry, there are no known methods to estimate the size of the catchment area. The paper uses a simple time interval between the creation of snapshots, but the major downside of this method is that it does not adapt to the size of the catchment areas; as a result it will produce more waypoints than necessary. Therefore, this appendix makes the first steps towards the estimation of the radius of the catchment area, as this would lead to a further reduction in memory consumption. A method is presented that can reliably detect the edge of the catchment area using a change in the direction of the homing vector. However, because of time constraints this method has not been implemented on the UAV.

Estimation of the size of the catchment area could be approached in two ways: it could be estimated from a single image, or from a signal that changes when crossing the boundary of the catchment area. In the first case, it might be possible to use machine learning to estimate the (minimal) radius of the catchment area directly from the raw snapshot image or perhaps even its Fourier transform. This option, however, is left open for further research as it diverges too far from the original research direction of this thesis.

Instead, this appendix explores the second option: signals that are already available for homing are examined to see if they respond to the crossing of the edge of the catchment area. The catchment area results directly from the visual homing process, therefore the only signals that *could* contain information about the catchment area are the visual input and the homing vector. (In other words, the IMU and other sensors won't be of much help here.) Section B-1 gives a qualitative overview of the behavior of these signals around the edge of the catchment area. Section B-2 then constructs and evaluates a 'detector' that measures the radius of the catchment area.

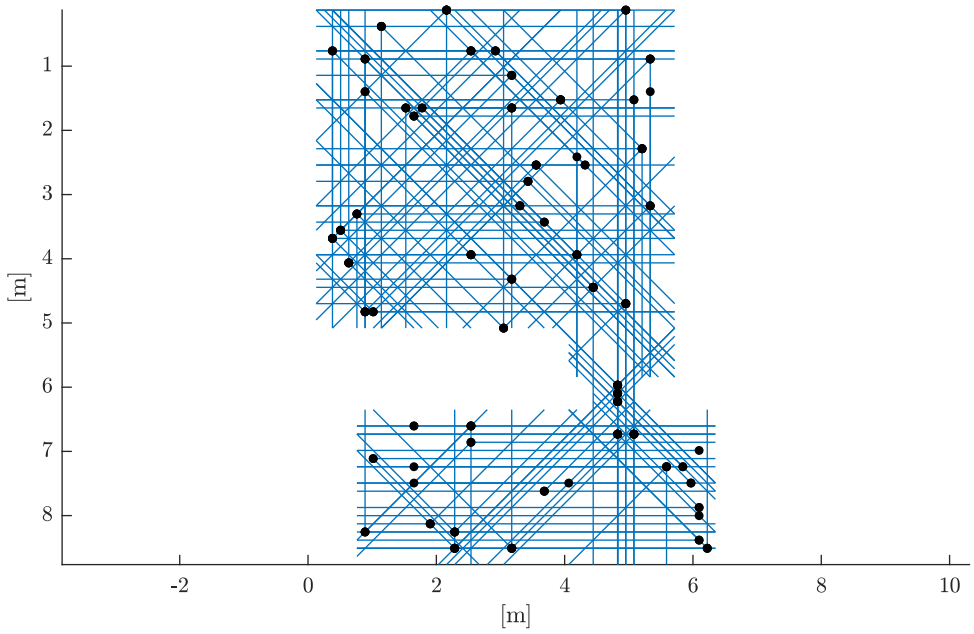


Figure B-1: Trajectories along which the detection of the catchment area's edge will be evaluated. The snapshots are shown as black dots, the trajectories themselves as blue lines. The trajectories and snapshots are spread evenly throughout the environment.

B-1 What happens around the edge of the catchment area?

This section examines the behavior of image dissimilarities and the homing vector when crossing the edge of the catchment area. This investigation is performed in MATLAB using the following procedure: first, a grid of Fourier-transformed images is constructed by extracting and transforming the horizon of the images in Gaffin and Brayfield's dataset [5]. Then, straight-line 'trajectories' are extracted from this dataset, starting from a snapshot taken at a random position (Figure B-1). When these trajectories contain at least as many cells inside the catchment area of the target position as outside that catchment area, they are stored for further analysis. This resulted in a collection of 203 trajectories that all cross the edge of a random catchment area.

For each of these trajectories, various signals can be examined. In this section, the following signals are logged and investigated:

- Image dissimilarity:
  - Between the current and target images.
  - Between the warped and target images. The 'warped' image is the result from the hypothetical movement that provides the best match with the target image (i.e. the 'warped' image is obtained by warping the current image using the homing vector).
- Homing vector:

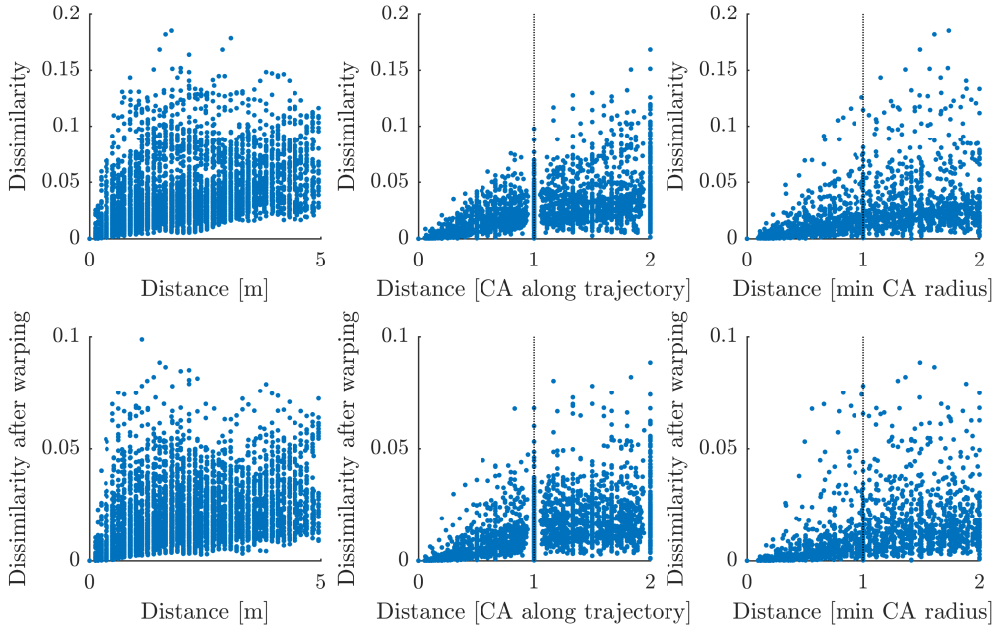


Figure B-2: Behavior of image dissimilarity when leaving the catchment area. The left column plots the dissimilarities against the true distance from the snapshot. The middle column is plotted against a normalized distance, where 1 corresponds to the edge of the catchment area along the direction of travel. In the right column, the distance of 1 corresponds to the minimal radius of the catchment area. The edge of the catchment area is indicated by a black dotted line. The top row shows the dissimilarity of the current and target images. The bottom row shows the dissimilarity between the warped and target images. None of the signals shows a drastic change in behavior when crossing the edge of the catchment area.

- Length
- Direction (compared to true homing direction)
- Relative orientation

The image dissimilarities and homing vectors are based on Fourier-transformed images with $K = 8$ and $N_{it} = 1$.

The behavior around the edge of the catchment area is evaluated by plotting these signals against a 'normalized distance'. This is the distance from the target image divided by 1) the radius of the catchment area along the direction of travel, or 2) the minimum radius of the catchment area. In these cases, a distance of 1 indicates the edge of the catchment area. In order to detect this edge, the signal should change drastically when crossing a normalized distance of 1.

Image dissimilarity The behavior of image dissimilarity is shown in Figure B-2. The dissimilarity is shown to increase with distance, but there is no sharp transition around the edge of the catchment area. It is therefore not possible to use a threshold on the image dissimilarity to determine whether the current image was taken inside or outside the catchment area.

Similarly, the dissimilarity between the target and the warped image is evaluated. The warped image should provide a good match with the target image as long as such a match can be found. It could therefore have shown a stronger transition around the edge of the catchment area, but Figure B-2 shows that it does not. Like the dissimilarity between the current and target images, there is no clear opportunity to threshold this signal to find the edge of the catchment area.

Homing vector The same procedure is used to examine the behavior of the homing vector, the results are shown in Figure B-3. Similarly to the image dissimilarity, the length of the homing vector might be used to detect when the drone is too far from the target snapshot. However, as seen in Figure B-3, the length of the homing vector already stops increasing before the edge of the catchment area. There is no clear opportunity to threshold the length of the homing vector to separate samples inside and outside the catchment area.

Fourier-based homing uses a visual compass to rotationally align the target image to the current image. This produces an estimate of the relative orientation ς. The alignment of the two images has been observed to fail when the distance between them is too large. Figure B-3, however, shows that errors in the orientation estimate do not seem to occur sharply at the edge of the catchment area.

Finally, the direction of the homing vector is investigated. Inside the catchment area, the homing vector points more-or-less towards the target location. Outside of the catchment area, however, the vector appears to point in a random direction (see also Figure B-4). In fact, around the edge the vector tends to point *away* from the catchment area, because if it pointed towards it, it would likely have been part of the catchment area in the first place. Unlike the signals examined before, this effect is clearly visible in the results of Figure B-3 (see the outlined results). A very sharp transition in the distribution of the direction errors is observed when crossing the edge of the catchment area. Figure B-5 shows the distribution of the angular error inside and outside the catchment area.

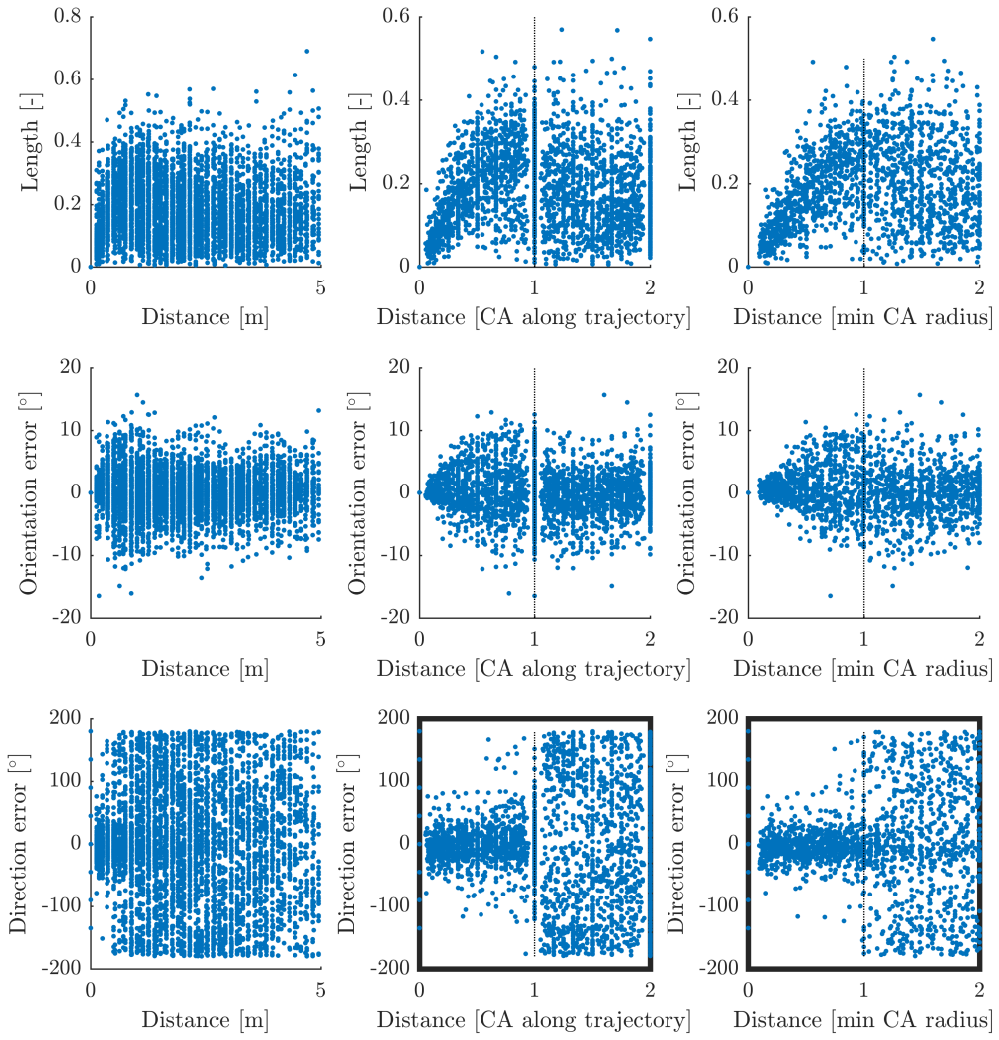


Figure B-3: Behavior of the homing vector around the edge of the catchment area. The left column is shown against the true distance from the snapshot, the distance in the middle column is normalized with the radius of the catchment area along the direction of travel, the right column with the minimum radius of the catchment area. The edge of the catchment area is located at a normalized distance of 1 and indicated with a black dotted line. The top row shows the length of the homing vector. The second row shows the error in the relative orientation estimate ς. The bottom row shows the angular error between the estimated homing vector and the true vector pointing back to the snapshot. None of the signals seem straightforward to threshold, except for the angular error of the homing vector (bottom row, indicated by black borders). The angular error shows a large difference when crossing the edge of the catchment area, and might therefore be used to detect it.

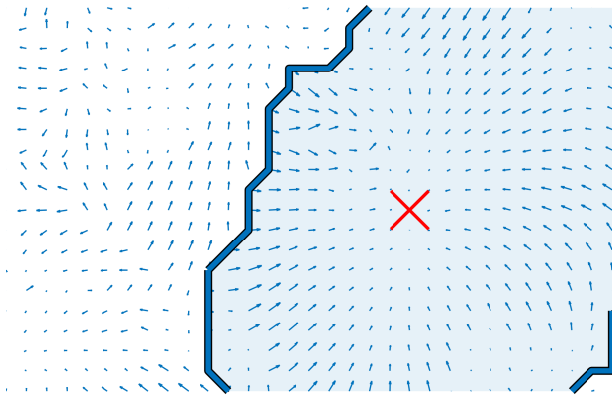


Figure B-4: Example of homing vectors inside and outside the catchment area. The catchment area is highlighted in blue. Inside the catchment area, the vectors point roughly towards the snapshot's location indicated by the red cross. Outside the catchment area, the vectors tend to point away from the catchment area. A sharp change in the direction of the homing vector is observed at the edge of the catchment area.

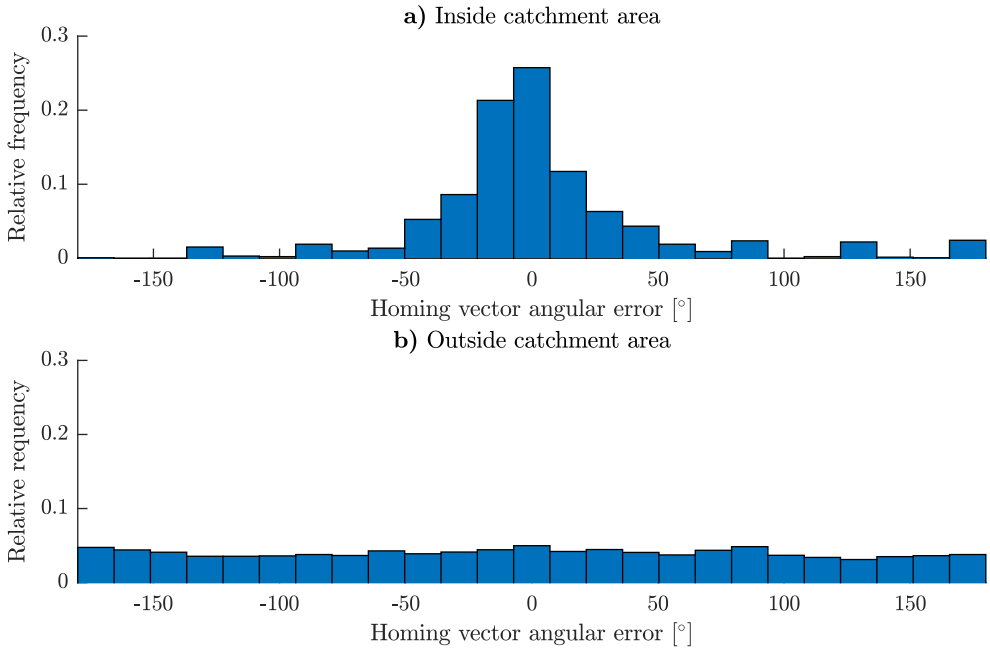


Figure B-5: Distribution of the angular error of the homing vector inside and outside the catchment area. a) Inside the catchment area, the homing vector point roughly towards the snapshot: the distribution is centered around an error of $0°$. b) Outside the catchment area, the angular error appears to be distributed uniformly.

B-2 Detecting the edge of the catchment area

The error in the direction of the homing vector shows a clear change when crossing the edge of the catchment area. If this change can be reliably detected, it could be used to measure the radius of the catchment area. In this section, signals such as the angular error and the angular rate of the homing vector and others are thresholded to determine when the UAV crosses the edge of the catchment area (Figure B-6). The UAV leaves the catchment area when the signal exceeds a preset threshold t. The accuracy with which these methods detect the edge of the catchment area is evaluated and compared.

The crossing of the edge of the catchment area is detected by thresholding one of the following signals:

- Homing vector
  - Angular error
  - Angular rate
  - Length and rate of length change
  - Relative orientation and rate of relative orientation
- Dissimilarity
  - Between current and target images, and rate of change of dissimilarity between current and target images
  - Between warped and target images, and rate of change of dissimilarity between warped and target images
- Distance

Given the results of Section B-1, the best results are expected for the homing vector's angular error and possibly angular rate.

In order to compare these detection methods, a performance measure is required. The RMS of the percentual error will be used as a performance measure:

$$E(t) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{r_{i,est}(t)}{r_{i,true}} - 1\right)^2} \tag{B-1}$$

Lower values of $E(t)$ indicate better performance, $E(t)$ is zero when all estimated radii $r_{i,est}(t)$ are exactly equal to the true catchment area radii $r_{i,true}$.

The estimated radii $r_{i,est}(t)$ are evaluated using the same trajectories as used in Section B-1. The first position along trajectory i where threshold t is exceeded is used as the estimated radius $r_{i,est}(t)$ (Figure B-6). If during one of the trajectories the threshold t is not exceeded at all, the result is invalid and $E(t)$ is undefined for that value of t.

The estimated catchment area radii $r_{i,est}(t)$ and therefore the error $E(t)$ depend on the choice of the threshold t. Threshold t will be optimized such that the error $E(t^*)$ is minimal. The same threshold is used for all trajectories.

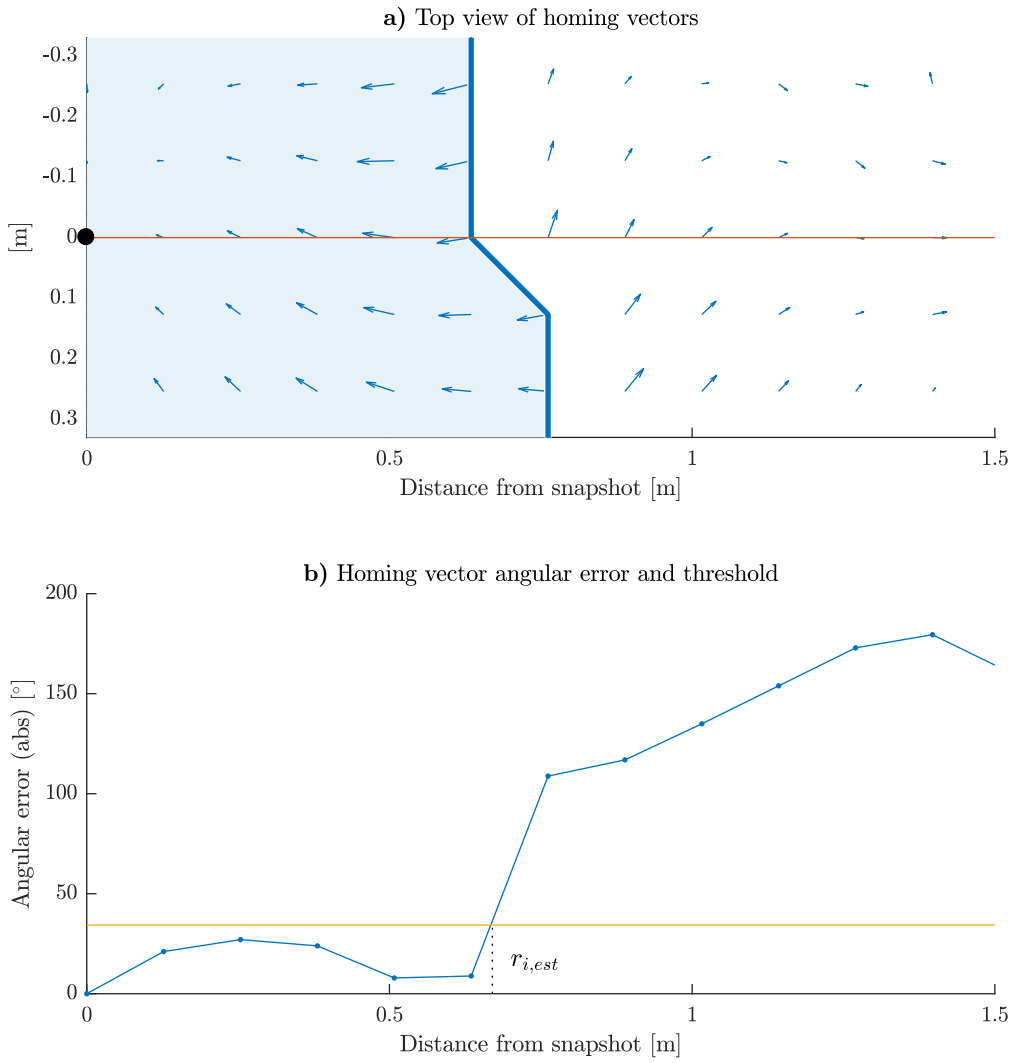


Figure B-6: Estimation of the radius of the catchment area along the direction of travel. A signal, in this case the angular error of the homing vector, is thresholded to find the radius $r_{i,est}$ of the catchment area. a) Top view of the trajectory, catchment area and homing vectors. The trajectory is shown as a red line starting at the snapshot shown as a black dot. The catchment area is highlighted in blue. The arrows depict the homing vectors. b) Angular error of the homing vector (blue) and the threshold t^* (yellow). When the angular error first exceeds the threshold, the edge of the catchment area is detected at radius $r_{i,est}$. The horizontal axes of both plots are aligned.

Table B-1: Comparison of catchment area edge detection methods.

Signal	$E(t^*)$	t^*	
Angular rate (abs)	0.38	113.92	°/m
Angular error (abs)	0.58	34.35	°
Dissimilarity after warping	0.66	0.00	-
Distance	0.72	0.39	m
Dissimilarity rate	0.75	0.01	m^{-1}
Dissimilarity	0.78	0.01	-
Relative orientation rate (abs)	0.79	10.79	°/m
Relative orientation (abs)	0.86	1.49	°
Homing vector length	0.87	0.12	-
Dissimilarity rate	0.89	0.02	m^{-1}
Homing vector length rate	0.95	0.39	m^{-1}

Table B-1 lists the resulting RMS errors $E(t^*)$ and optimal choice of thresholds t^* for all methods. Detection methods based on the direction of the homing vector have the smallest error. The angular rate of the homing vector appears to be a better predictor of the edge of the catchment area than the error between the estimated and true homing vector directions.

The distribution of the catchment area radii estimated by the three best detectors of Table B-1 is shown in Figure B-7. Both the angular rate and angular error of the homing vector show a large peak around 1, indicating that in roughly 50% of the cases the estimated radius $r_{i,est}$ is equal to the true radius $r_{i,true}$. The other 50% of the distribution appears to lie at estimated radii below 1. In these cases, the estimated catchment area radius is smaller than its true value. Waypoints will then be created closer together than necessary, but this will only reduce the memory efficiency of the map, whereas an overestimation of the radius would space waypoints too far apart and could cause the drone to get lost.

The method presented in this appendix is a first step towards the estimation of the size of the catchment area. This estimate can be used to make the route following method presented in the paper even more memory-efficient by increasing the maximal distance between waypoints.

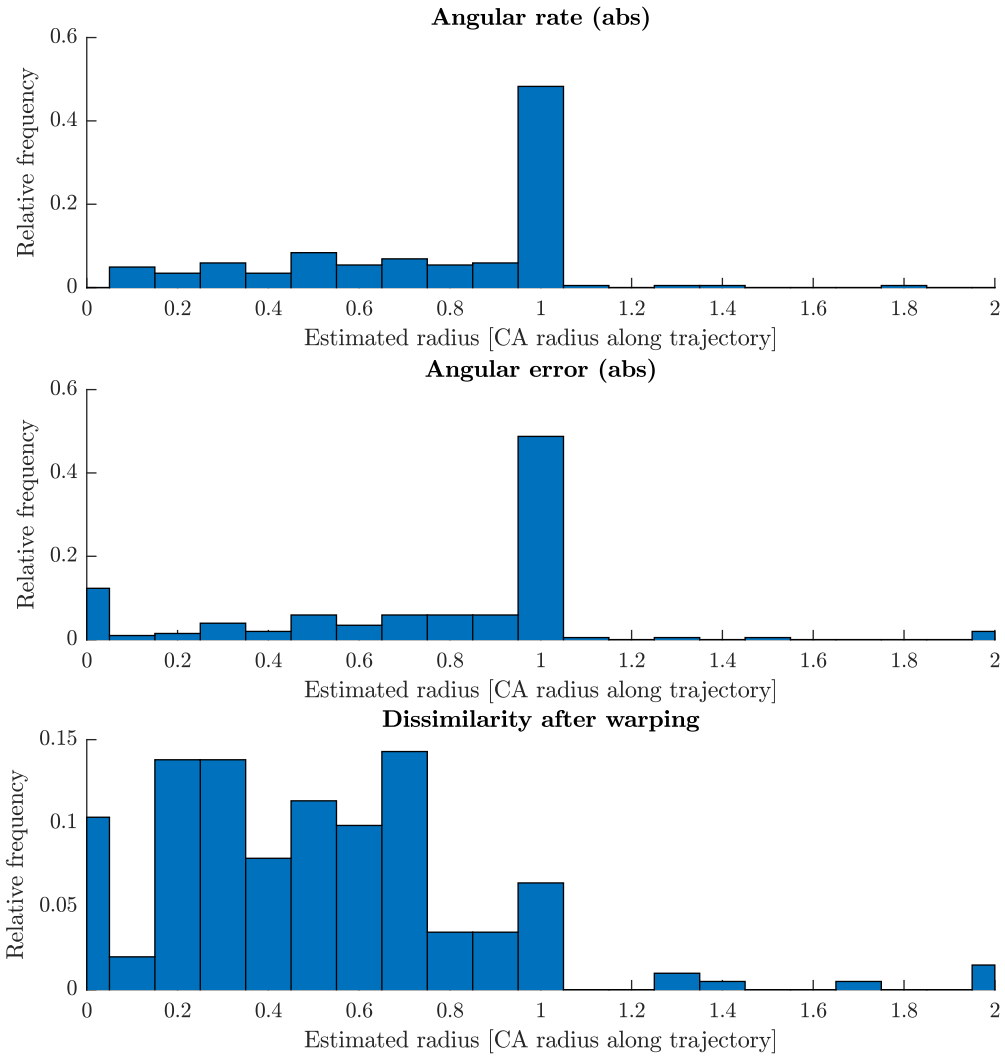


Figure B-7: Catchment area radius estimations by the three best-scoring methods. The estimated radius is normalized by the true radius along the direction of travel. The angular rate of the homing vector and its angular error show similar behavior: in roughly 50% of cases the size of the catchment area is measured correctly; in the other cases the estimate is smaller than the true radius, which would place waypoints too close together, resulting in a less efficient map but not in failure of the visual homing maneuvers. Dissimilarity after warping performs significantly worse than the first two methods.

Appendix C

Unscaled visual odometry

In the paper, a choice was made to use IMU-based odometry with a drag force model of the drone because of its simplicity and independence of the environment. As an alternative, it might be possible to use the omnidirectional camera to estimate the movement of the drone. A disadvantage of the use of an omnidirectional camera is that the distance to the environment is not known, therefore the movement can only be estimated up to an unknown scale. However, when the drone follows the same route back, the environment is located at the same distance and the movements should still sum to zero. Odometry can be performed by integrating the homing vector between successive frames.

Section C-1 briefly reviews properties of the homing vector that are relevant for unscaled visual odometry. The homing vector can then be integrated from frame to frame, but Section C-2 will show that drift can be reduced if movements are only integrated once the drone is far-away enough from the previous reference image. Section C-3 presents results of flight tests that show that the drone can find its way back to the starting point and that unscaled visual odometry can successfully replace IMU-based odometry in the route following experiments of Section 6 of the paper with a distance of 1.20 m between snapshots. Finally, Section C-4 will briefly discuss the differences between unscaled visual odometry and IMU-based odometry with the drag model.

C-1 Visual odometry using the homing vector

Appendix B showed that there is little to no correlation between the length of the homing vector and the true metric distance to an arbitrary snapshot (Figure B-3 on page 33). However, the reason that the length of the homing vector correlates so poorly to metric distance from an *arbitrary* snapshot is that the environment is located at wildly differing distances, which causes the scale of the homing vector to differ between snapshots.

When the drone retraces its route back to the starting position, the environment will be located at the same distance as during recording. For a *single* snapshot, as opposed to *all* snapshots examined in Appendix B, the length of the homing vector is actually linearly

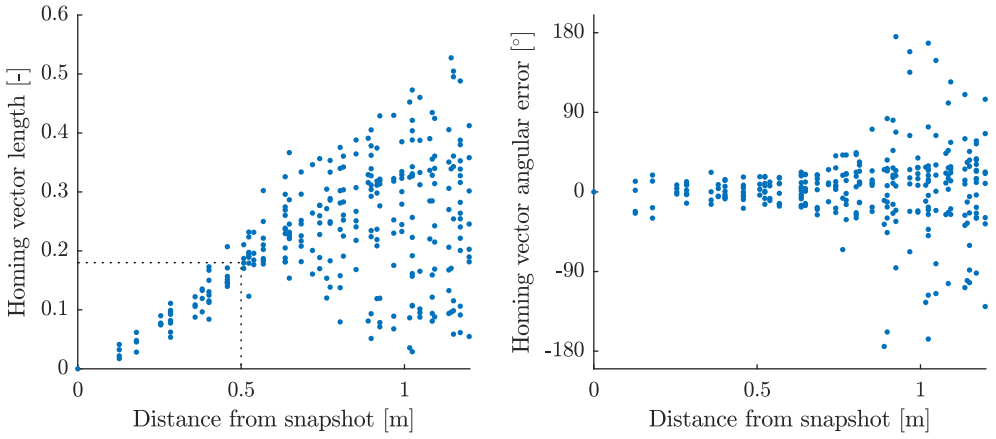


Figure C-1: Behavior of the homing vector around a single snapshot. The length of the homing vector is linearly correlated with the distance from the snapshot, but only in a small (approx. $< 0.5\,\mathrm{m}$) region surrounding the snapshot where $|\boldsymbol{h}| \leq 0.18$ (dotted area). The direction of the homing vector is accurate up to approximately $0.6\,\mathrm{m}$.

correlated to the true distance from the snapshot, as shown in Figure C-1. This means that the length of the homing vector *can* be used to measure distances along the route. Distances between successive frames can be integrated to estimate the position of the drone along the route. While this estimate is not guaranteed to be metrically accurate (the size of the environment and therefore scale of the homing vector can change along the route), the distance estimate is at least repeatable when following the same trajectory. For navigation using odometry, repeatability is more important than metrical accuracy; the drone should end up at the same real-world location, it does not matter if it estimates to have moved 1 meter or 100 meter.

While the length of the homing vector can be integrated to estimate the distance from a snapshot in a straight line, this becomes more complicated when trajectories with corners or curves are mapped. If the scale of the homing vector changes after the drone has changed direction, the displacements can not be summed together because the direction of the homing vector will then be incorrect. This problem can be solved by only travelling in straight lines between waypoints. If a change in direction is observed, a waypoint can be created at that location. Even without unscaled visual odometry this is desirable behavior, as it prevents the drone from cutting corners when following the route. Any errors caused by a change in the environment size are then eliminated by homing towards the waypoints along the route.

C-2 Use of intermediate snapshots to reduce drift

The simplest way of estimating the position of the drone is to sum the homing vectors obtained between successive camera frames. However, as the homing vector always contains a small error, the position estimate will quickly drift from the true position of the drone.

When the drone receives a new image I_k, this image is compared to the previous image I_{k-1}

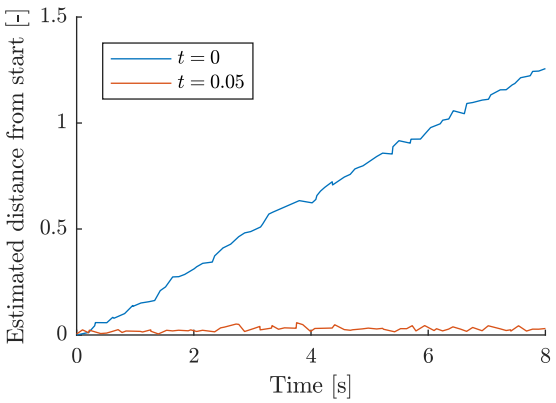


Figure C-2: Position estimate of the drone while stationary. If the homing vectors are integrated according to (C-2) ($t = 0$), the position estimate drifts away from the true position (which is 0 as the drone is stationary). When the position is estimated relative to a reference snapshot as in (C-4) with threshold $t = 0.05$, the drone's estimated position does not drift.

to estimate the homing vector between them:

$$\boldsymbol{h}_k = \boldsymbol{h}(I_k, I_{k-1}) + \boldsymbol{\epsilon}_k \tag{C-1}$$

with $\boldsymbol{\epsilon}_k$ the measurement error of the homing vector. These homing vectors are summed to estimate the position of the drone $\hat{\boldsymbol{x}}_k$:

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{h}_k \tag{C-2}$$

At each timestep, a new error $\boldsymbol{\epsilon}_k$ is added to the position estimate $\hat{\boldsymbol{x}}_k$, which causes drift.

However, when the drone is stationary, it is not necessary to estimate the drone's position relative to the last image. Instead, an earlier image I_r could be used as a *reference snapshot*:

$$\boldsymbol{h}_{k|r} = \boldsymbol{h}(I_k, I_r) + \boldsymbol{\epsilon}_k \tag{C-3}$$

where $r \leq k - 1$. The current position $\hat{\boldsymbol{x}}_{k,r}$ is then estimated through:

$$\hat{\boldsymbol{x}}_{k|r} = \hat{\boldsymbol{x}}_{r|r} + \boldsymbol{h}_{k|r} \tag{C-4}$$

The difference here is that only a single homing vector is added to the estimated position of the reference snapshot $\hat{\boldsymbol{x}}_{r|r}$ instead of all homing vectors between times r and k as in (C-2). This means that also only a *single* error $\boldsymbol{\epsilon}_k$ is added to the reference position $\hat{\boldsymbol{x}}_{r|r}$. The drift is therefore significantly smaller; in fact, as long as the previous position estimate $\hat{\boldsymbol{x}}_{r|r}$ is not updated, the estimated position of the drone $\hat{\boldsymbol{x}}_{k|r}$ will not drift at all. It only contain a single measurement error $\boldsymbol{\epsilon}_k$ at each timestep, this error is not accumulated between successive images (Figure C-2).

This method works as long as the drone stays close to the last reference position $\hat{\boldsymbol{x}}_{r|r}$, because otherwise the length of the homing vector $\boldsymbol{h}_{k|r}$ no longer correlates to the true distance from the reference snapshot. Therefore, the reference snapshot I_r and position $\hat{\boldsymbol{x}}_{r|r}$ are replaced by the current image and position estimate I_k, $\hat{\boldsymbol{x}}_{k|r}$ when the length of the homing vector $\boldsymbol{h}_{k|r}$ exceeds a threshold t. According to Figure C-1, this threshold should lie somewhere between

0 and 0.18. The threshold should be picked as small as possible (to ensure that the homing vector stays linearly correlated to the true distance from the snapshot), but large enough that the reference snapshot is not unnecessarily updated during hover. A threshold of $t = 0.05$ produced good results during test flights and will be used for the remainder of this appendix.

C-3 Results

Repeatability in the Cyberzoo To test whether unscaled visual odometry works can be used to retrace the path of the drone and to evaluate its accuracy, test flights were performed on the AR.Drone 2.0 in the Cyberzoo. The estimated position can not be compared to the true position as it does not have a defined scale, but its repeatability can be evaluated as follows: the drone uses odometry to record its outbound flights (straight lines of varying length) and then returns to the starting position by steering the estimated position back to zero. The remaining error gives an indication of the accuracy of the odometry.

The resulting trajectories are shown in Figure C-3. The drone is able to return to within approximately 1 to 2 m of the starting position. In most cases, the drone arrived short of the target position. Figure C-4 shows the relation between the starting distance and the final position error of the drone. The odometric error grows with distance.

Figure C-5 compares the estimated distance to the true distance from the starting point. Samples were collected during the inbound and outbound parts of all runs. Despite the dependence on the size of the environment, the estimated distance appears to be linearly correlated to the true distance from the starting point.

Route following Unscaled visual odometry was also evaluated in the 'Corridor' environment in the Cyberzoo. The result is shown in Figure C-6. The drone is able to follow the route back with a distance of 1.20 m between waypoints, thereby achieving the same performance as IMU-based odometry in the paper. Errors in route following in this run appear to come mostly from errors in visual homing rather than from the unscaled visual odometry.

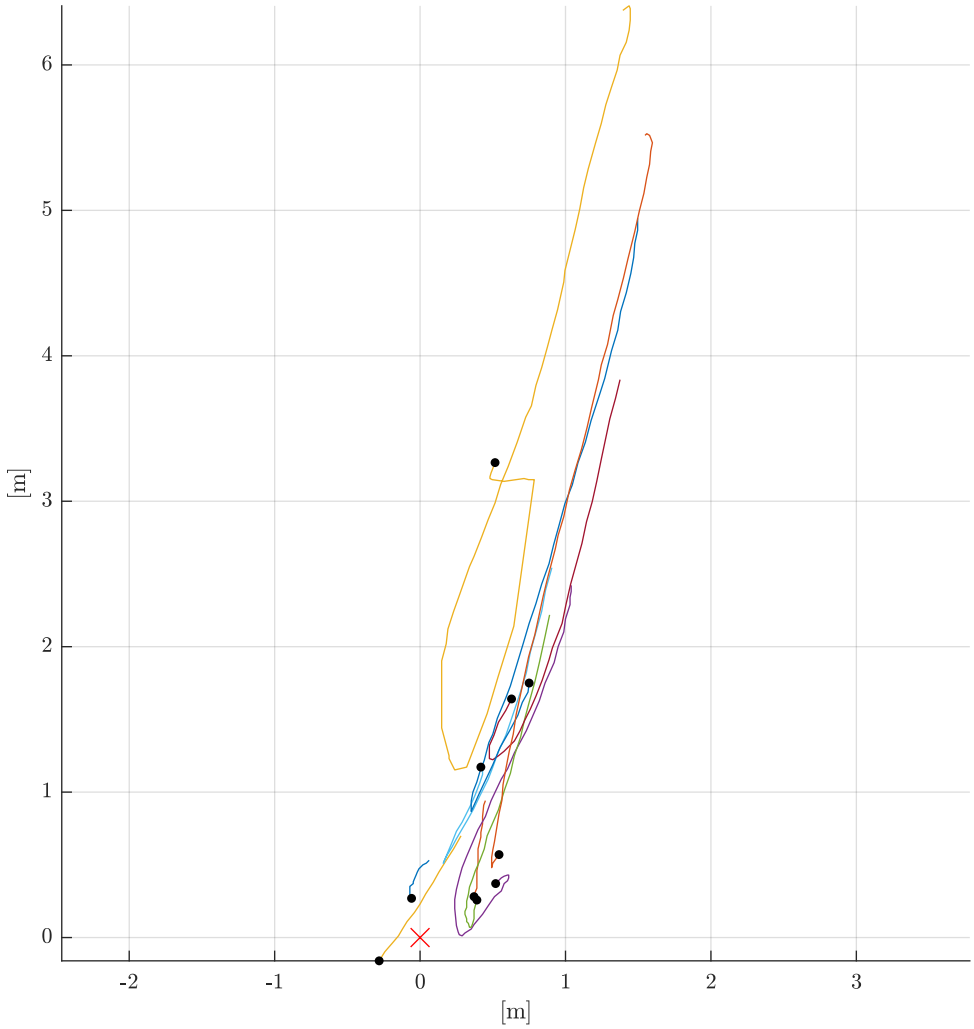


Figure C-3: Trajectories when retracing outbound flights using unscaled visual odometry. The endpoints are marked by black circles.

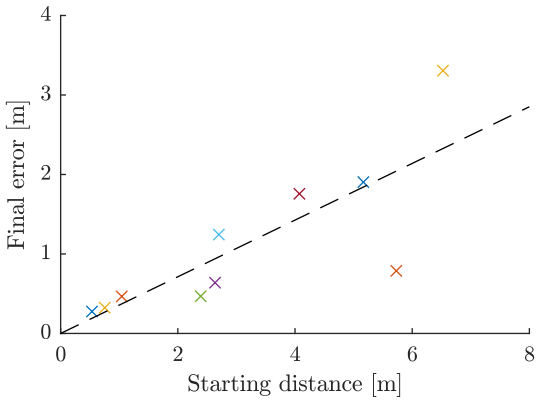


Figure C-4: Final position error as function of the starting distance. The two appear to be linearly correlated with an average final error of 36% of the starting distance. Taking into account that part of this error was also incurred during the recording of the outbound flight, the odometric drift lies in the order of 18% of the traveled distance.

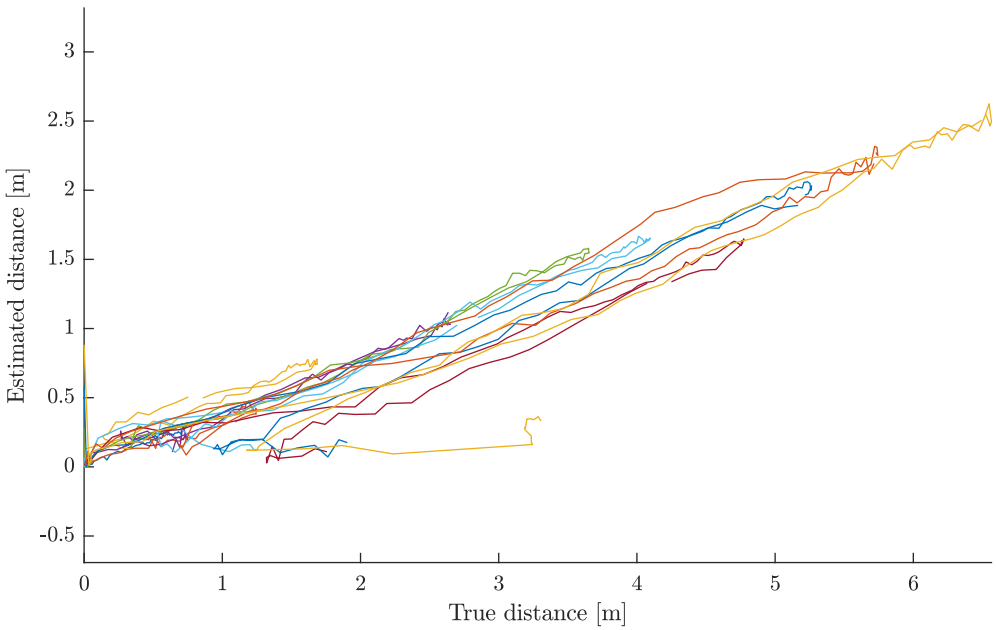


Figure C-5: Estimated distance from the start position. Despite the unpredictable nature of the scale of the environment and therefore length of the homing vectors between images, the unscaled odometric distance appears to be linearly correlated with the true distance in this experiment.

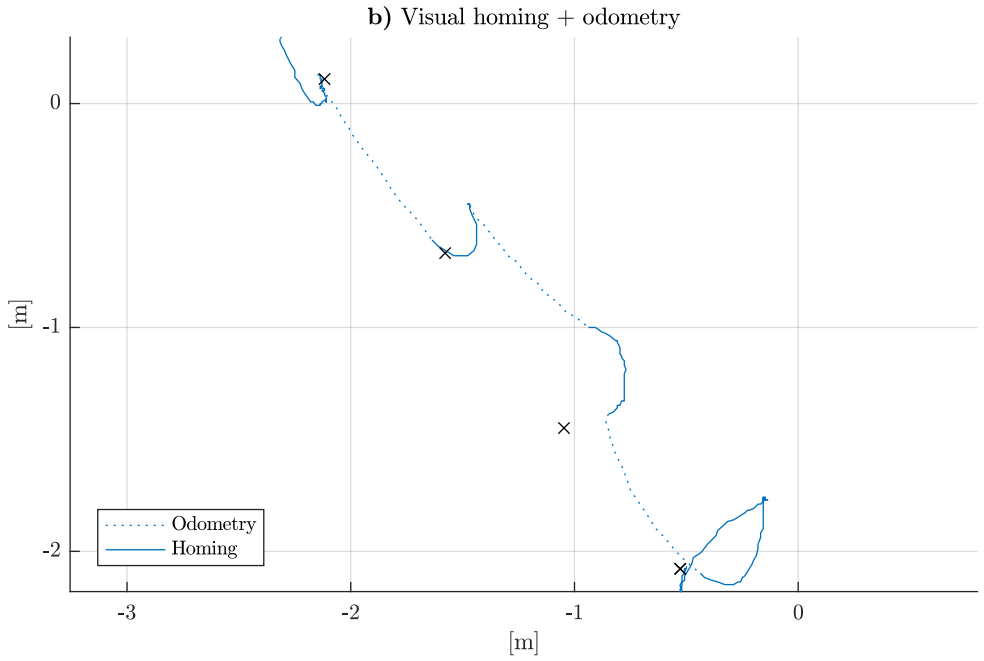


Figure C-6: Route following results in the 'corridor' environment in the Cyberzoo (see Section 6 of the paper). The drone travels from the top left to the bottom right. It can successfully follow a route with 1.20 m distance between waypoints using unscaled visual odometry; in fact, in this trajectory most errors appear to come from visual homing.

C-4 Comparison with drag-based odometry

Unscaled visual odometry and drag-based odometry have some fundamentally different properties. The most important difference is that the behavior of visual odometry strongly depends on the shape and texture of the environment. While IMU-based odometry behaves the same in any environment, the same cannot be said for unscaled visual odometry. The fact that the length of the homing vector and its error scale with the size of the environment makes it difficult to model the uncertainty of the position estimate. This is the main reason why unscaled visual odometry was not used in the paper.

Interestingly, the size of the catchment area also scales with the size of the environment. Unlike IMU-based odometry, unscaled visual odometry therefore automatically 'adapts' to the size of the catchment area. How this compares to IMU-based odometry with estimation of the catchment area radius was not investigated for this thesis.

Finally, unscaled visual odometry differs from IMU-based odometry in terms of drift. Using the intermediate snapshots as proposed in Section C-2, its error is expected to grow with distance rather than time, which might be advantageous when flying slowly.

Unscaled visual odometry has been shown to work as a replacement for IMU-based odometry in route following. Further research would be required to compare the two and decide which approach leads to the most memory-efficient map.

Appendix D

Paparazzi + Gazebo: a new simulator for vision-based UAV control

The use of simulation has proven extremely helpful during this thesis. There are many advantages to the use of simulation: firstly, simulation gives easy access to ground truth information on the state of the quadrotor, and this information can also be used by the autopilot which means that algorithms can be tested without noise or other disturbances. Secondly, simulations can be performed much quicker then flight tests. Small changes in the code can be tested in a matter of seconds. Thirdly, simulation can also be used to reduce the time required for real-world flight tests. Many of the experiments performed for this thesis were first tested in simulation, the actual flight test could then be finished in as little as thirty minutes. Finally, simulation allowed the route following algorithm to be evaluated over long trajectories, while such an experiment is difficult to perform in the real world; there are for instance no clear-cut solutions that allow the true position of the drone to be recorded over the entire trajectory.

Like most of the work at TU Delft's MAVLab, the code for this research is based on the Paparazzi autopilot[1]. While a simulator based on JSBSim[2] was included with Paparazzi, this simulation was limited to the dynamics of the UAV; vision was not part of the simulation. Paparazzi's simulator could be connected to FlightGear[3] for visualization for the user, but there was no way to feed this information back into the autopilot.

As no suitable simulator was available for the experiments in this thesis, I have extended Paparazzi's NPS (New Paparazzi Simulator) framework with the ability to use the Gazebo[4] simulator. The new simulator is able to send images captured by simulated cameras back to the autopilot and therefore allows closed-loop simulation of vision-based control schemes for UAVs.

[1] https://wiki.paparazziuav.org
[2] http://jsbsim.sourceforge.net/
[3] http://www.flightgear.org/
[4] http://gazebosim.org/

While not as relevant to my own work, the use of Gazebo is also a first step towards the simulation of interaction between the drone and the environment. For instance, range finders can be used to measure distances towards obstacles in the environment. Other interactions such as picking up and carrying objects could also be simulated. Whilst not currently supported, further developments of the Gazebo simulator for Paparazzi could also allow simulations to be performed with multiple drones, allowing swarming or cooperative carrying to be tested in simulation as well.

The Gazebo extension for NPS was merged with the 'master' branch of Paparazzi on June 13, 2017[5]. Since then, the simulator has already been used for other projects at the MAVLab.

Section D-1 of this appendix gives an overview of the implementation of the simulator. Section D-2 shows the environments that were used in the experiments for this thesis.

D-1 Implementation

The role of Gazebo and NPS in the overall control loop is shown in Figure D-1. The simulation starts with motor commands coming from the stabilization controller. The motor commands are transformed into forces and torques that are applied to a rigid-body model of the quadrotor. The parameters of the quadrotor model — mass, inertia, nominal motor thrusts and torques — were copied from the original JSBSim model. They were not validated for the experiments in this thesis but result in behavior that is comparable to the real AR.Drone 2.0. Aerodynamic effects on the quadrotor are mostly neglected as they were not that relevant for the experiments in this thesis. A simple linear model is used for drag; the results in Section 5-2 of the paper show that this is sufficiently accurate.

To provide feedback to the autopilot, sensor measurements are generated based on the true state of the quadrotor. The generation of sensor measurements is handled by the NPS framework. The measurements are corrupted with realistic disturbances, for instance the IMU measurements contain an unknown bias and white measurement noise and the GPS sensor (unused in this thesis) produces delayed output.

NPS uses an existing interface, ABI messages[6], to share these measurements with other parts of the autopilot. This decouples modules and other parts of the autopilot from a specific sensor implementation. As a result, these parts do not require any modifications to work with the simulator.

An effort was made to treat video handling in a similar way: by emulating the video_thread[7] module that is normally used on Linux-based autopilots, the video output of Gazebo can be used without modification by the subscribing modules.

The camera images are obtained from virtual cameras attached to the quadrotor. Gazebo supports standard pinhole cameras, but also cameras with more exotic types of lenses. For the experiments of this thesis, the drone was fitted with an omnidirectional camera on its bottom side. The horizon can be extracted from these images using the same procedure as described in Section 5-1. The cameras are continuously polled during simulation. When a

[5] https://github.com/paparazzi/paparazzi/pull/2069
[6] https://wiki.paparazziuav.org/wiki/ABI
[7] http://docs.paparazziuav.org/v5.10/module__video_thread.html

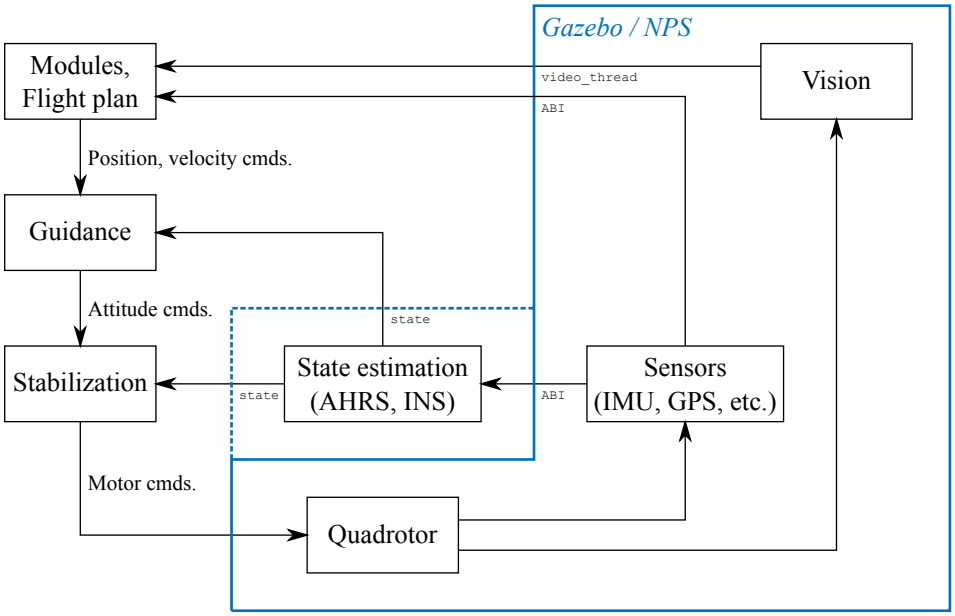


Figure D-1: Gazebo and NPS in the overall control loop. Simulation starts with the motor commands coming from the stabilization controller that are applied as forces and torques on a rigid-body model of the quadrotor. Realistic sensor measurements including noise and other disturbances are generated based on the true state of the simulated quadrotor. State estimation can performed using these measurements or replaced by the true state of the quadrotor; the latter is used in this thesis. Vision is handled through emulation of the `video_thread` module that would normally be used on Linux-based autopilots such as on the AR.Drone 2.0. By communicating through existing interfaces (ABI, video_thread and state), the modules, guidance, stabilization and optionally AHRS and INS can be used in Gazebo without modification.

new image is available, it is converted to the YUV422 format used by Paparazzi and passed to the rest of the autopilot through the video_thread mockup.

State estimation in the autopilot can be handled in two ways: the existing filters can be used with the simulated data, or the estimated state can be overwritten by the drone's true state obtained directly from Gazebo (see Figure D-1). In this thesis, the latter option is used as the state estimation is not yet validated in simulation and could contain unexpected bugs.

The other parts of the autopilot, most notably the modules and the guidance and stabilization controllers collect all their data through ABI messages, the state interface or the video_thread module. They can therefore be used in Gazebo *without modification*. This saves on development time and ensures that the drone behaves the same in simulation as in the real world.

D-2 Environments

In order to perform visual route following experiments, realistic indoor environments are required. This section briefly showcases the environments that were used and developed for the experiments of this thesis.

Café Small-scale experiments were performed in the 'Café' environment (Figure D-2). The Café model is created by Nate Koenig and publicly available through Gazebo's model database[8]. The model contains a large, open room (and a kitchen that is too small for quadrotor operation) with windows an furniture along the walls. The drone could successfully home towards snapshots in this environment, as shown in Section 6 of the paper.

Cyberzoo On occasion, flight tests that worked in the Café environment failed when they were performed in the real Cyberzoo. A likely cause for these failures was the lack of features along the walls of the cyberzoo. In order to verify this, a simple model of the Cyberzoo was made for Gazebo (Figure D-3). Simulation results from this environment could then be compared to those obtained in the Café environment. Tests that work in the Café environment but failed in the real and simulated Cyberzoo likely suffered from a lack of texture.

Aerospace Engineering To test long-range route following, a large indoor environment was required. Unfortunately, no suitable models were found online. The only indoor models in which trajectories of 100 meter or more could be evaluated often have no textures as they are designed for SLAM with range-finders.

Therefore, a new indoor environment had to be created for the experiments of this thesis. The new environment is based on the faculty of Aerospace Engineering at TU Delft, specifically the part surrounding the MAVLab. The environment model is created in SketchUp based on a floorplan and photographic material of the faculty (Figure D-4, D-5).

The environment consists mostly of corridors, but also contains some irregularly shaped junctions and a curved wall in the middle. Several props were made to add texture to the

[8] http://models.gazebosim.org/

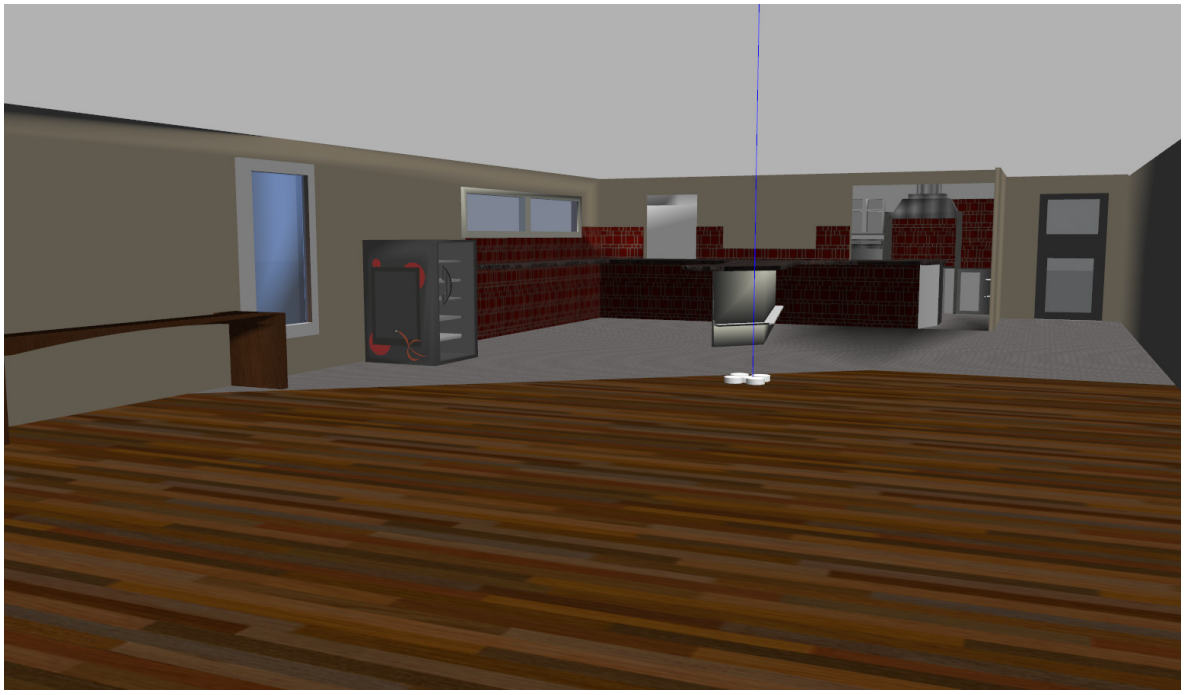


Figure D-2: 'Café' environment used in Gazebo. This environment is used for small-scale experiments such as homing towards a single snapshot.

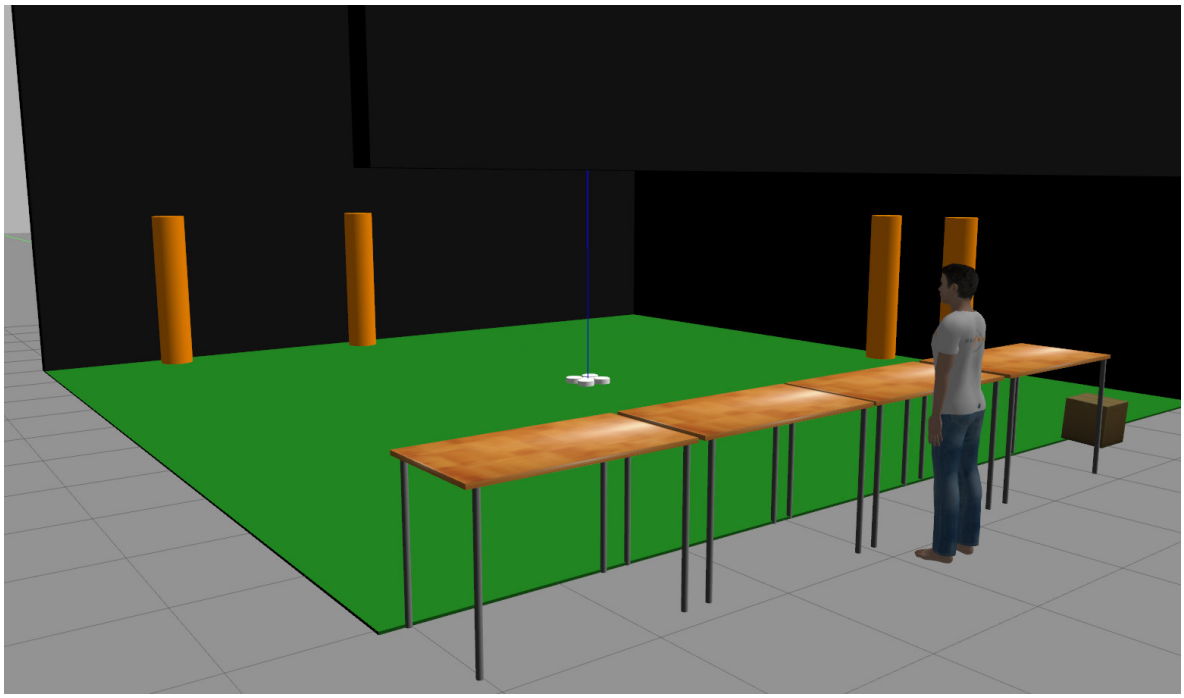


Figure D-3: 'Cyberzoo' environment in Gazebo. By comparing simulation results in this environment to those obtained in the 'Café' environment, a lack of features can be confirmed as a cause of failure of flight tests performed in the real Cyberzoo.

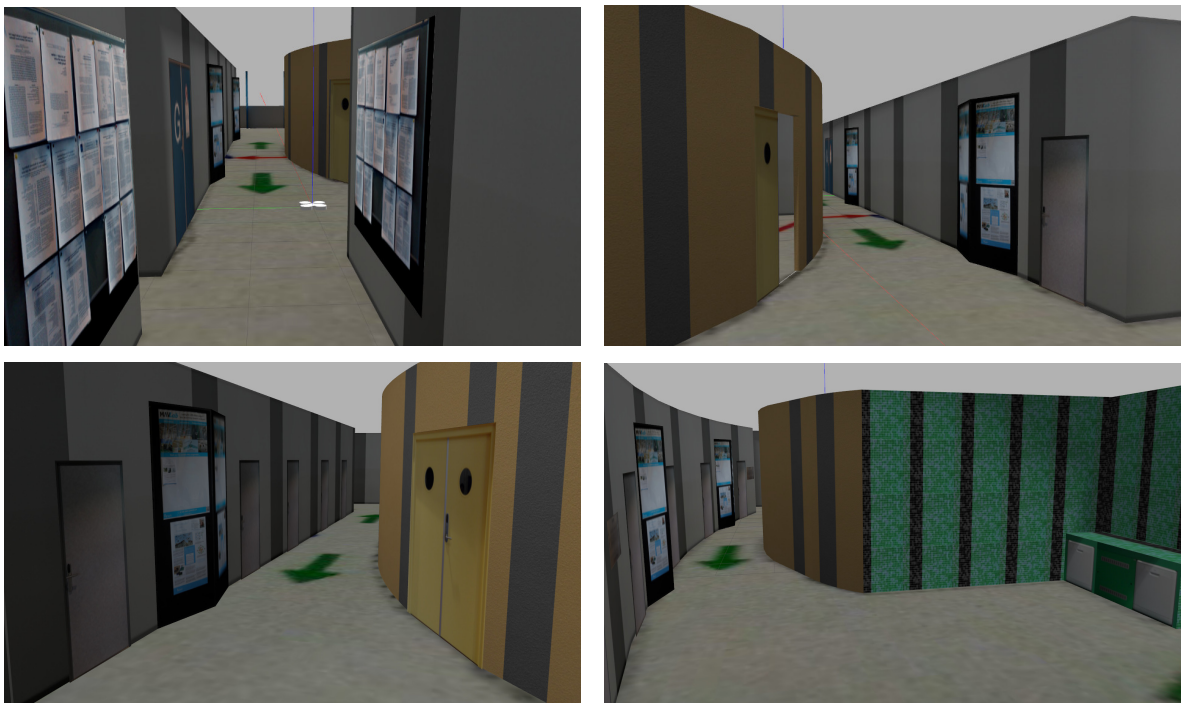


Figure D-4: 'Aerospace Engineering' environment used to demonstrate route following over longer distances. This environment is modeled after the part of the Aerospace Engineering faculty of the TU Delft surrounding the MAVLab. Black bars are added to the walls to compensate for the lack of illumination differences and lack of small-scale features in the environment.

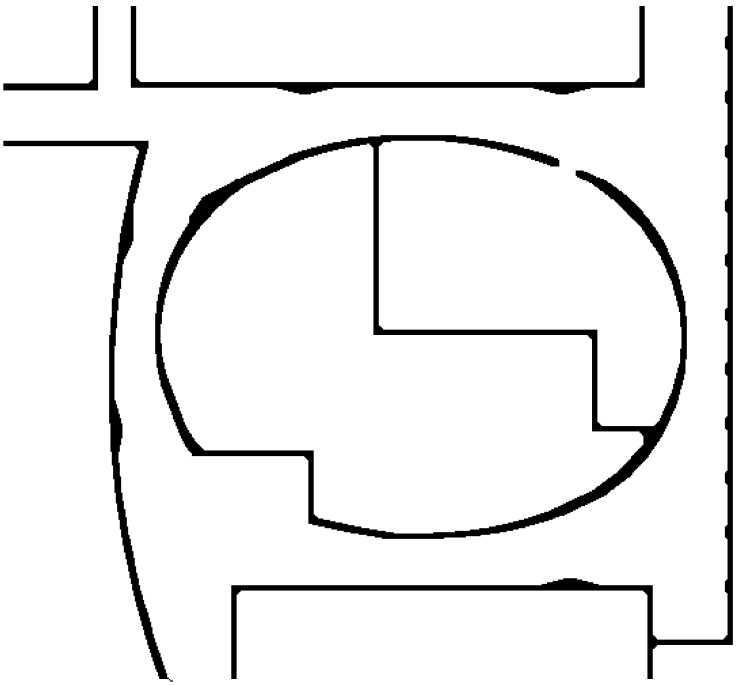


Figure D-5: Floorplan of the Aerospace Engineering environment.

environment: doors, posters, noticeboards. However, because the environment still lacked in contrast (for instance, there were no dark or bright spots on the walls because the whole scene is lit by a single, global light source), black bars were added to the walls to compensate for this missing texture.

In Section 6 of the paper, this environment is used to demonstrate that the drone can follow long routes in realistic environments.

Bibliography

[1] 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*, vol. 79, no. 3, pp. 191–202, 1998.

[2] R. Möller and A. Vardy, "Local visual homing by matched-filter descent in image distances," *Biological Cybernetics*, vol. 95, pp. 413–430, oct 2006.

[3] W. Stürzl and H. A. Mallot, "Efficient visual homing based on Fourier transformed panoramic images," *Robotics and Autonomous Systems*, vol. 54, no. 4, pp. 300–313, 2006.

[4] R. Möller, A. Vardy, S. Kreft, and S. Ruwisch, "Newton-based matched-filter descent in image distances," *Biological Cybernetics (Submitted)*, 2006.

[5] D. D. Gaffin and B. P. Brayfield, "Autonomous Visual Navigation of an Indoor Environment Using a Parsimonious, Insect Inspired Familiarity Algorithm," *Plos One*, vol. 11, no. 4, p. e0153706, 2016.

Glossary

Catchment area Region surrounding a *snapshot* from where *visual homing* succeeds.

Dead reckoning Navigation method where *odometry* is used to move between locations.

Drag-based velocity estimation Estimation of the drone's velocity that uses the IMU to measure the drag acting on the quadrotor and a drag model to convert this drag into a velocity.

Homing, Feature-based Class of *visual homing* techniques that uses the bearings towards local features in the current and *snapshot* images to find the *homing vector*. Contrast with *image-based homing*.

Homing, Fourier-based *Image-based homing* method presented in [3] that uses Fourier-transformed horizon images as *snapshots*.

Homing, Image-based Class of *visual homing* techniques that uses raw or compressed images as *snapshots*. Homing is performed by moving such that the difference between the current and snapshot images is minimized; the snapshot is located at the global minimum of the *Image Difference Function*. Contrast with *feature-based homing*.

Homing, Search-based *Image-based homing* method presented in [1] that uses *warping* to predict a large number of images for hypothetical movements. The movement that results in the best match with the *snapshot* is used as *homing vector*.

Homing, Visual Navigation method where the drone compares its current observation to a *snapshot* to find a *homing vector* that guides the drone to the location where the snapshot was taken.

Homing vector Vector that, when followed, guides the drone to the location of a *snapshot*. Note that the vector does not necessarily point straight towards the target location; it can also point to the side of it, which results in curved homing trajectories. An error of less than $\pm 90^\circ$ is sufficient for successful homing [1]. Depending on the homing method, the length of the homing vector might give an indication of the distance towards the snapshot.

IDF See *Image Difference Function*.

Image Difference Function Function of position with a value equal to the difference between an image taken at that position and a *snapshot*. Used in *image-based homing*.

MFDID See *Matched Filter Descent in Image Distances*.

Matched Filter Descent in Image Distances *Image-based homing* method presented in [2] that uses the spatial gradient of the *Image Difference Function* as *homing vector*.

Odometry Estimation of the position of the drone by integration of velocity measurements.

Odometry, IMU-based *Odometry* method that integrates *drag-based velocity estimates*. Can also refer to the integration of accelerations and angular velocities measured by the IMU, but this method is not used in this thesis.

Path following Navigation method where the drone follows a clearly distinguishable path in the environment such as a wall or corridor. Contrast with *route following*.

Route following Navigation method where the drone follows a route that has been recorded earlier. Unlike *path following*, the drone is not necessarily bound to a clearly distinguishable path in the environment.

Sequential visual homing *Route following* method that relies on *visual homing* to move the drone along a sequence of *waypoints*. Can be combined with *odometry* to increase the distance between waypoints.

Snapshot Reference image; *visual homing* can be used to return to the location where the snapshot was taken. See also *waypoint*.

Visual compass Estimation of the orientation of the drone relative to a *snapshot* or other reference image.

Visual compass following *Route following* technique where the drone uses a *visual compass* to adjust its heading to match the orientation of reference images taken along the route.

Warping Method presented in [1] that predicts images that would be obtained after small, hypothetical movements.

Waypoint Point along the recorded route that the UAV will home towards. Contains a *snapshot*. Is connected to other waypoints and can contain an *odometry* vector that points towards the next waypoint.