



Radboud Universiteit Nijmegen



Master of Science Thesis

Artificial Intelligence

Machine Learning-based Indoor Localization for Micro Aerial Vehicles

Volker Strobel

s4491491

Internal Supervisor

dr. Louis Vuurpijl

Donders Institute for Brain, Cognition and Behaviour
Radboud University

External Supervisor

dr. Guido De Croon

Micro Aerial Vehicle Laboratory
Delft University of Technology

August 24, 2016

Volker Strobel

Machine Learning-based Indoor Localization for Micro Aerial Vehicles

Supervisors: dr. Louis Vuurpijl and dr. Guido De Croon

August 24, 2016

Radboud University Nijmegen

Artificial Intelligence

Comeniuslaan 4

6525 HP Nijmegen

Abstract

Micro Aerial Vehicles (MAVs) are versatile platforms: their applications range from surveillance to search and rescue operations. However, MAVs have limited processing power due to their small size and cannot fall back on standard localization techniques in the indoor environment. To address this issue, an efficient on-board localization technique using machine learning was developed in the scope of this thesis.

The vision-based approach estimates x, y -coordinates within a known and modifiable indoor environment. Its computational power is scalable to different platforms, trading off speed and accuracy. Histograms of textons—small characteristic image patches—are used as features in a k -Nearest Neighbors (k -NN) algorithm. Several possible x, y -coordinates that are outputted by this regression technique are forwarded to a particle filter to neatly aggregate the estimates and solve positional ambiguities. To predict the performance of the algorithm in different environments an evaluation technique is developed. It compares actual texton histogram similarities to ideal histogram similarities based on the distance between the underlying x, y -positions. The technique assigns a loss value to a given set of images, enabling comparisons between environments and the identification of critical positions within an environment. To compare maps before modifying an environment, a software tool was created that generates synthetic images to simulate those taken during an actual flight.

We conducted flight tests to evaluate the performance of the approach. A comparison of the localization technique with the ground truth showed promising results: the algorithm has a localization accuracy of approximately 0.6 m on a $5\text{ m} \times 5\text{ m}$ area at a runtime of 32 ms on board of an MAV. In a triggered landing experiment, the MAV correctly landed in or close to specified areas. The map evaluation technique was applied to various high-resolution images to identify suitable maps.

The presented approach is based on three pillars: (i) a shift of processing power to a pre-flight phase to pre-compute computationally complex steps, (ii) lightweight and adaptable algorithms to ensure real-time performance and portability to different platforms, (iii) modifiable environments that can be tailored to the presented algorithm. These pillars build a foundation for efficient localization in various GPS-denied environments.

Preface

This thesis concludes my student time in the Netherlands. After my first year in Nijmegen at the Radboud University, I headed to Delft to start my graduation project.

Without my supervisors—Guido de Croon, Louis Vuurpijl, and Roland Meertens—this thesis would not be the same. I want to take the opportunity to thank them for their advice, support, and ideas.

Thank you to all my colleagues in the MAVLab for their explanations, interesting discussions, and the nice time. I would like to thank Erik van der Horst for his help with software and hardware.

Thanks to my family and friends for making life fun. Thank you, Franzi, for supporting me in all life situations.

Contents

1	Introduction	1
1.1	Problem Statement and Research Questions	3
1.2	Contributions	3
1.3	Thesis Outline	4
2	Related Work	5
2.1	Vision-based Localization Methods	5
2.1.1	Optical Flow	5
2.1.2	Fiducial Markers	6
2.1.3	Homography Determination & Keypoint Matching	6
2.1.4	Convolutional Neural Networks	8
2.2	Texton-based Methods	8
3	Methods	10
3.1	Hardware and Software	10
3.2	Preliminary Dataset Generation	12
3.3	Machine Learning-based Approach and Filtering	14
3.3.1	Texton Dictionary Generation	15
3.3.2	Histogram Extraction	16
3.3.3	k -Nearest Neighbors (k -NN) algorithm	17
3.3.4	Filtering	17
3.4	Map Evaluation	21
3.4.1	Evaluation Scheme	21
3.4.2	Synthetic Data Generation	23
4	Analysis	27
4.1	Determining the Number of Image Patches	27
4.2	Determining k in the k -NN algorithm	28
4.3	Histogram Distance and Measurement Error	29
4.4	Flight Tests	30
4.4.1	Training Set based on Motion Tracking System	30
4.4.2	Baseline: Homography-based Approach	30
4.4.3	Training Set based on Homography-finding Method	31
4.4.4	Triggered Landing	32

4.5	Speed versus Accuracy Trade-Off	32
4.6	Comparing Maps	35
5	Discussion	37
5.1	General Discussion	38
6	Conclusion	41
	Bibliography	42

In the world of automation, micro aerial vehicles (MAVs) provide unprecedented perspectives for domestic and industrial applications. They can serve as mobile surveillance cameras, flexible transport platforms, or even as waiters in restaurants. However, indoor employment of these vehicles is still hindered by the lack of real-time position estimates. The focus of this thesis is, thus, the development of efficient indoor localization for MAVs combining computer vision and machine learning techniques.

While unmanned aerial vehicles (UAVs) for outdoor usage can rely on the global positioning system (GPS), this is usually not available in confined spaces and would not provide sufficiently accurate estimates in cluttered environments. If enough computational and physical power is available, a common approach to estimate a UAV's position is via active laser rangefinders [25, 5]. Although this approach is used in some simultaneous localization and mapping (SLAM) frameworks, it is usually not feasible for MAVs because they can carry only small payloads. A viable alternative are passive computer vision techniques. Relying on visual information scales down the physical payload since cameras are often significantly lighter than laser rangefinders [7, 4, 2]. However, this reduced physical payload must be traded off against the higher computational payload for the on-board CPU: vision-based position estimation is usually a time-consuming and memory-intensive procedure. One way to overcome this problem is to process the data on a powerful external processor by establishing a wireless connection between the MAV and a ground station. Such off-board localization techniques often lack the versatility, though, due to factors—such as the bandwidth, delay, or noise of the wireless connection—interfering with the system's reliability.

The developed framework uses a computationally efficient machine learning approach to estimate x, y -positions, which circumvents the requirement to store a map in the MAV's "mind." To assign x, y -coordinates to images in a training set, keypoints in the current image and a map image are detected in a pre-flight phase. This is then followed by finding a homography—a perspective transformation—between them to locate the current image in the map. As an alternative images can be aligned with high-precision position estimates from a motion tracking system.

In the next step, the complexity of these images is reduced by determining their histogram of textons—small characteristic image patches [46]. New images can then also be encoded as texton histograms and matched to images with known x, y -positions using the k -Nearest Neighbors (k -NN) algorithm. The k -NN estimates are passed to a particle filter to neatly aggregate the estimates and resolve positional ambiguities. The computational effort of the approach can be adjusted by modifying the amount of extracted patches and used

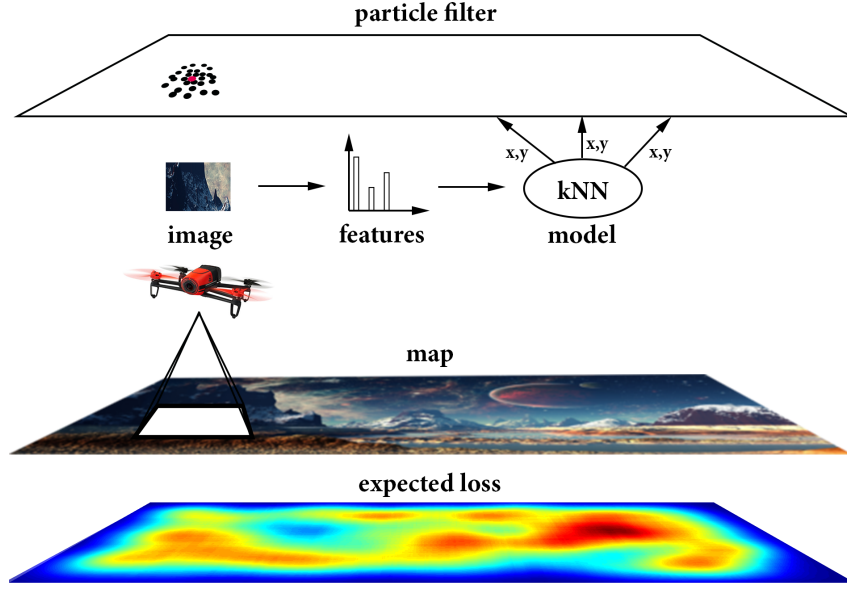


Fig. 1.1: The figure illustrates the developed system from a high-level perspective. A feature vector—the texton histogram—is extracted from the current camera image of the MAV. The feature vector is forwarded to a machine learning model that uses a k -Nearest Neighbors algorithm to output x, y -position estimates. These estimates are passed to a particle filter, which filters position estimates over time and outputs a final position estimate (red point). The expected loss shows regions in the map where a lower localization accuracy is expected. The average expected loss can be used as “fitness value” of a given map.

particles, resulting in a trade-off between accuracy and execution frequency. Figure 1.1 summarizes the algorithm.

In the presented approach, computational power is shifted to an offline training phase to achieve high-speed during live operation. In contrast to visual SLAM frameworks, this project considers scenarios in which the environment is known beforehand or can even be actively modified. The environment is non-dynamic and planar, therefore, the MAV will make use of texture on the bottom or ceiling of the environment. This opens the door for improving the accuracy of the algorithm by changing the map. On the basis of desired characteristics of a given map, an evaluation technique was developed that determines the suitability of an environment for the presented approach. This technique allows for spotting distant regions with similar image features, which could lead to deteriorated performance. The evaluation can be performed using a given map image or recorded images during flight. In the former case, synthetic images are generated from the map image that simulate images taken during flight.

1.1 Problem Statement and Research Questions

The goal of this thesis is to develop a fast localization technique for MAVs. Therefore, we formulated the following problem statement:

Problem statement: *How can x, y -coordinates be estimated in real-time and on-board of an MAV?*

It is assumed that the UAV flies at an approximately constant height, such that the estimation of height is not necessary. Since it is intended to further reduce the size of MAVs, lightweight and scalable position estimation algorithms are needed. The problem was addressed by combining computer vision and machine learning techniques for achieving real-time position estimates. We focus on the following research questions (RQs):

- **RQ 1:** *“Can 2D positions be estimated in real-time using a machine learning approach on a limited processor in a modifiable indoor environment?”*

Real-time position estimates can pave the way for autonomous flight of MAVs in various indoor environments; pursuing an “on-board design” to make the MAV independent of an external ground station is an important step for security and versatility.

- **RQ 2:** *“How can we predict and evaluate the suitability of a given map for the developed localization approach?”*

Computer vision techniques are commonly limited to environments with sufficient and informative texture. If an environment can be evaluated before actually flying in it, the performance of the approach can be predicted and possible dangers prevented.

1.2 Contributions

The first contribution of this thesis is a machine learning-based indoor localization system that runs in real-time on board of an MAV, paving the way to an autonomous system. In contrast to existing *active* approaches, the developed *passive* approach only uses a monocular downward-looking camera. Since computer vision-based localization approaches yield noisy estimates, a variant of a particle filter was developed that aggregates estimates over time to produce more accurate predictions. It handles the estimates of the k -NN algorithm in an integrative way and resolves position ambiguities. The method is a global localization system and does not suffer from error accumulation over time.

The second contribution is a map evaluation technique that predicts the suitability of a given environment for the presented algorithm. To this end, a synthetic data generation tool was developed that creates random variations of an image. The tool simulates different viewing angles, motion blur, and lighting settings; the generated synthetic images are labeled with x, y -coordinates based on the 3D position of the simulated camera model.

The developed software is made publicly available. It encompasses (i) the localization algorithm as part of the Paparazzi autopilot system [8], which consists of the texton-based approach in combination with a particle filter¹ (ii) software for augmenting an image with synthetic views², (iii) a script for labeling images with x, y -positions based on homographies³, and (iv) a script for evaluating a map based on histograms and corresponding x, y -positions⁴.

1.3 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 surveys existing indoor localization approaches related to this thesis. In Chapter 3, the developed texton-based approach is presented and its components, the k -NN algorithm and the particle filter, are introduced. Details about the synthetic data generation tool and map evaluation technique are also given. Chapter 4 describes the setup and results of the on-ground and in-flight experiments. We discuss the results and indicate future research directions in Chapter 5. Finally, we draw our conclusions in Chapter 6.

¹<https://github.com/paparazzi/paparazzi>

²<https://github.com/Pold87/draug>

³<https://github.com/Pold87/relocalizer>

⁴https://github.com/Pold87/map_evaluation

This chapter discusses advantages and disadvantages of different approaches for indoor localization. While a wide range of methods for indoor localization exists, from laser range scanners over depth cameras to radio-frequency identification tag (RFID) based localization, we only discuss methods that use the same technical and conceptual setup—localization with a monocular camera.

Generally, two types of robot localization techniques are distinguished: local techniques and global techniques [22]. Local techniques need an initial reference point and estimate coordinates based on the change in position over time. Once they have lost track, the position can typically not be recovered again. The approaches also suffer from “drift” since errors are accumulating over time. Global techniques are more powerful and do not need an initial reference point. They can recover when temporarily losing track and address the *kidnapped robot problem*, in which a robot is carried to an arbitrary location [21].

Target systems and test environments are often too different to draw comparisons: factors, such as the size of the environment, the speed of the robot or camera, or the processor play crucial roles for the evaluation. Therefore, comparing the accuracy and run-time of different localization methods is difficult.

2.1 Vision-based Localization Methods

2.1.1 Optical Flow

Optical flow algorithms are biologically inspired methods for navigation—taking inspiration from insects and birds [42]. They estimate the apparent motion between successive images, for example, by comparing the positions of their keypoints [9]. Optical flow methods belong to the class of *local* localization techniques and can only estimate the position relative to an initial reference point. The approaches suffer from accumulating errors over time and typically do not provide a means for correcting these errors.

Chao et al. [9] compare advantages and disadvantages of different optical flow algorithms for the use with UAV navigation. Most approaches are computationally rather complex [36]. To render on-board odometry feasible for small MAVs, McGuire et al. [36] introduce a lightweight optical flow variant. The algorithm uses compressed representations of images in the form of edge histograms to calculate the flow.

2.1.2 Fiducial Markers

Fiducial markers (Figure 2.1), which are often employed in augmented reality applications [30, 24], have been used for UAV localization and landing [20, 39]. The markers encode information in the spatial arrangement of black-and-white or colored image patches. Their corners can be used for estimating the camera pose at a high frequency. The positions of the markers in an image are usually determined with *local thresholding*. Local thresholding is a simple method for separating objects—salient image regions—from a background. Its output is a binary image with two states: foreground (markers) and background. Marker positions are then often further refined by removing improbable shapes, yielding an adjusted version of possible marker positions [23].

An advantage of fiducial markers is their widespread use, leading to technically mature open-source libraries, including ArUco [23] and ARToolKit [30]. Given adequate lighting conditions, markers can be used in a wide variety of environments [28]. This makes them suitable for indoor localization. A drawback of the approach is that motion blur, which frequently occurs during flight, can hinder the detection of markers [3]. Furthermore, partial occlusion of the markers through objects or shadows break the detection; each marker needs to be fully in the camera view [28]. Another disadvantage is that markers might be considered as visually unpleasant and may not fit into a product or environment design [10]. They offer little flexibility because one has to rely on predefined marker dictionaries. Additionally, marker-based approaches always require the modification of the environment. Like most vision-based approaches, the detection of markers is prone to changes in lighting conditions and may not work in low-contrast settings [28].

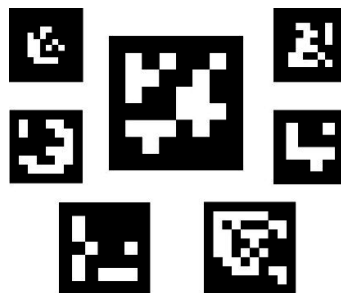


Fig. 2.1: Examples of fiducial markers of the ArUco library.

2.1.3 Homography Determination & Keypoint Matching

A standard approach for estimating camera pose is detecting and describing keypoints of the current view and a reference image [43], using algorithms such as Scale-invariant feature transform (SIFT) [35], followed by finding a homography—a perspective transformation—between both keypoint sets (Figure 2.2). A keypoint is a salient image location described by a feature vector. Depending on the algorithm, it is invariant to different viewing angles and scaling.

The SIFT algorithm transforms an image into a set of image features. It works in four subsequent stages using gray-scale images as input:

1. *Maxima detection*: The image is convolved with the *Difference of Gaussian* blob detector. By changing the variance of the Gaussian distribution, the maxima—potential keypoints—across different scales and spaces can be detected.
2. *Refinement of keypoints*: The potential keypoints are refined by removing maxima with small contrast and non-discriminative edges.
3. *Orientation assignment*: A histogram of the gradient orientations around the keypoint is created. The most frequent value indicates the keypoint orientation.
4. *Keypoint description*: The local image gradients are transformed into a feature vector by describing pixels around a radius of a keypoint.

To locate the current view in the reference image, keypoints from one set are matched with their nearest neighbor in the other set using the Euclidean distance between their feature vectors. Based on the matched keypoint descriptions, a homography is calculated between the coordinates of both keypoint sets. This allows for locating the current view in the reference image. The calculation of the homography matrix (H) needs four matches between both keypoint sets. Usually many more points are available, leading to an overdetermined equation. The solution to H is then computed by minimizing the errors between all the projected keypoints in a least-square sense.

While this *homography-based* approach is employed in frameworks for visual Simultaneous Localization and Mapping (SLAM), the pipeline of feature detection, description, matching, and pose estimation is computationally complex [31]. Therefore, ground stations for off-board processing or larger processors are usually needed for flight control. The approach

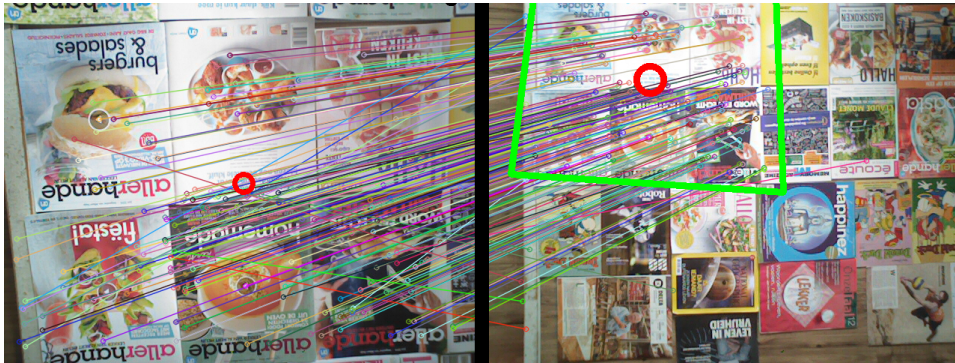


Fig. 2.2: Perspective transformation between keypoints of the current image (left) and the reference or map image (right).

has been employed for global localization for UAVs: Blösch et al. [7] evaluate it on a $3.5\text{ m} \times 2\text{ m}$ area and achieve a root mean square (RMS) positional error below 10 cm in x, y, z -direction. Calculations are executed on a powerful ground station, which is connected to the UAV with a USB cable. Subsequent research has brought the algorithm on

board of UAVs [1], achieving a frequency of 10 Hz with a 1.6 GHz on-board processor with 1 GB RAM. However, the required processing power is still too complex for small MAVs.

2.1.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specialized machine learning method for image processing [34]. The supervised method has outperformed other approaches in many computer vision challenges [18]. CNNs consist of multiple neuron layers that represent increasing levels of abstraction [34]. While their training is usually time-consuming, predictions with CNNs often take only few milliseconds, shifting computational effort from the test phase to the training phase. CNNs have been used as a robust alternative for keypoint detection and description if images were perturbed [18] but needed more computation time than SIFT.

In recent work, Kendall et al. present a framework for regressing camera positions based on CNNs [31]. The method has an accuracy of approximately 50 cm in indoor environments with a size between $2 \times 0.5 \times 1 \text{ m}^3$ and $4 \times 3 \times 1.5 \text{ m}^3$. It is rather robust to different lighting settings, motion blur, and varying camera intrinsics. The approach predicts positions on a modern desktop computer in short time.¹

2.2 Texton-based Methods

Textons are small characteristic image patches; their frequency in an image can be used as image feature vector. Varma and Zisserman [46] originally introduced textons for classifying different textures, showing that they outperform computationally more complex algorithms. For the classification, the approach compares texton histograms between a training set and the test sample. The class of the closest training sample is assigned to the test sample. A texton histogram is obtained by extracting small patches from an image and comparing them to all textons in a “texton dictionary.” The frequency of the most similar texton is then incremented in the histogram.

Texton histograms are flexible image features and their extraction requires little processing time, which makes them suitable for MAV on-board algorithms. The approach allows for adjusting the computational effort by modifying the amount of extracted image patches, resulting in a trade-off between accuracy and execution frequency [15]. A disadvantage is that it discards all information about the spatial arrangement of image patches, so that different images can have the same histogram.

De Croon et al. [14] use textons as image features to distinguish between three height classes of the MAV during flight. Using a nearest neighbor classifier, their approach achieves a height classification accuracy of approximately 78 % on a hold-out test set. This enables

¹However, in our implementation—employing the scientific computing framework Torch [11]—the approach was still computationally too involved for achieving real-time prediction on an Odroid XU-4 single board computer.

a flapping-wing MAV to roughly hold its height during an experiment. In another work, De Croon et al. [16] introduce the *appearance variation cue*, which is based on textons, for estimating the proximity to objects [16]. Using this method, the MAV achieves a high accuracy for collision detection and can avoid obstacles in a $5\text{ m} \times 5\text{ m}$ office space.

In the scope of this thesis, an efficient *global* localization was developed that draws upon the lightweight character of texton-based approaches. The homography-based approach is used in a pre-flight phase to assign x, y -coordinates to images.

Methods

This section describes the ideas behind the developed approach, the hardware, and software implementations. The approach is based on three “pillars”: (i) a shift of processing power to a pre-flight phase to pre-compute computationally complex steps, (ii) lightweight and adaptable algorithms to ensure real-time performance and portability to different platforms, (iii) modifiable environments to get the most out of the approach. The pseudo code in Algorithm 1 shows a high-level overview of the parts of the framework. Details are given in the following sections.

Algorithm 1 High-level texton framework

```

1:  $t \leftarrow 0$ 
2:  $\mathcal{X}_0 \leftarrow \text{INIT\_PARTICLES}$ 
3: while true do
4:    $t \leftarrow t + 1$ 
5:    $I_t \leftarrow \text{RECEIVE\_IMG\_FROM\_CAMERA}$ 
6:    $\mathcal{H}_t \leftarrow \text{GET\_TEXTON\_HISTOGRAM}(I_t)$ 
7:    $\mathbf{z}_t \leftarrow k\text{-NN}(\mathcal{H}_t)$ 
8:    $\mathcal{X}_t \leftarrow \text{PARTICLE\_FILTER}(\mathcal{X}_{t-1}, \mathbf{z}_t)$ 
9:    $x_t, y_t \leftarrow \text{MAXIMUM\_A\_POSTERIORI\_ESTIMATE}(\mathcal{X}_t)$ 
10: end
  
```

3.1 Hardware and Software

In our first approach, the commercially available Parrot AR.Drone 2.0 was equipped with an Odroid XU-4 single board computer, a Logitech 525 HD webcam, and a WiFi module. Figure 3.1 shows the setup. Instead of employing the AR.Drone 2.0 processor, the camera images were processed on the more powerful Odroid processor and the resulting x, y -estimates were sent over a USB data link to the MAV flight controller. The Odroid processor has a full operating system (Ubuntu 15.04) and can run arbitrary Linux software. However, the additional weight from the modifications of the system resulted in unstable flight performance. Therefore, we abandoned the idea of adding an external Odroid processor and modified the system to execute the localization algorithm directly on-board of the MAV. To this end, the software had to be ported from the high-level language Python to the low-level language C using the Paparazzi autopilot software [8]. This step removed the need for the additional payload and made the flight performance stable. Also, it circumvented the effort of buying and attaching an external processor, which can be another point of failure. Another advantage is that the framework can be easily ported to

any UAV supported by the Paparazzi software. The major disadvantage is that the on-board processors of many MAVs have a lower performance than the Odroid processor.

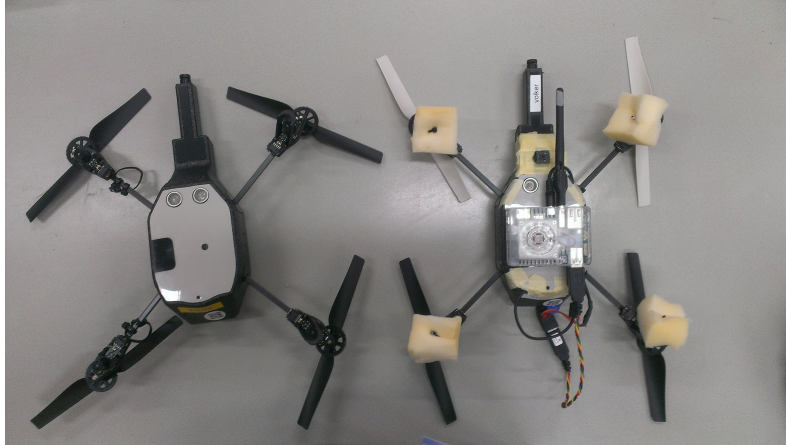


Fig. 3.1: Comparison of an unmodified Parrot AR.Drone 2.0 (left) and a modified version (right). The modified one was equipped with an Odroid XU-4 single board computer, a Logitech C525 HD camera, a WiFi module, and a USB connection between the Odroid board and the AR.Drone.2.0 flight controller. The additional weight led to unstable flight performance, therefore, we resorted to unmodified MAVs.

We decided to conduct all our tests with a quadcopter. Quadcopters allow for navigating in arbitrary directions without changing their yaw angle, show stable flight behavior, and often have high-resolution cameras. We used the *Parrot Bebop Drone* as a prototype. It is equipped with a lithium-ion polymer battery that lasts for approximately 11 minutes of flying time. The UAV's dimensions are $28 \times 32 \times 3.6$ cm and it weighs 400 g. It has two cameras: a front camera and a downward-looking bottom camera. The developed approach makes use of the bottom camera only. This camera has a resolution of 640×480 pixels with a frequency of 30 frames per second. The UAV's processor is a Parrot P7 dual-core CPU Cortex A9 with a tact rate of 800 Mhz. It is equipped with 8 GB of flash memory and runs a Linux operating system. The full specifications of the UAV can be found on its official website [40].

The original Bebop software development kit was replaced with the open-source autopilot software Paparazzi [8]. Paparazzi is used and advanced at the Micro Aerial Vehicle Laboratory at the TU Delft. The software provides a link between a ground station computer and the UAV to send commands and receive telemetry data. Furthermore, it provides functions for creating flight plans, plotting and logging telemetry data, and uploading firmware to the UAV. Its modular approach allows for combining functions regarding stabilization, localization, and control of UAVs, which are executed on board of the MAV. Paparazzi supports a wide range of commercially available aircrafts and associated hardware. Figure 3.2 shows the ground control station of Paparazzi.

The presented approach is implemented as a module in Paparazzi's computer vision framework. Since low-level routines, like accessing camera information or attitude control for different platforms, are already implemented in Paparazzi, the module can be readily

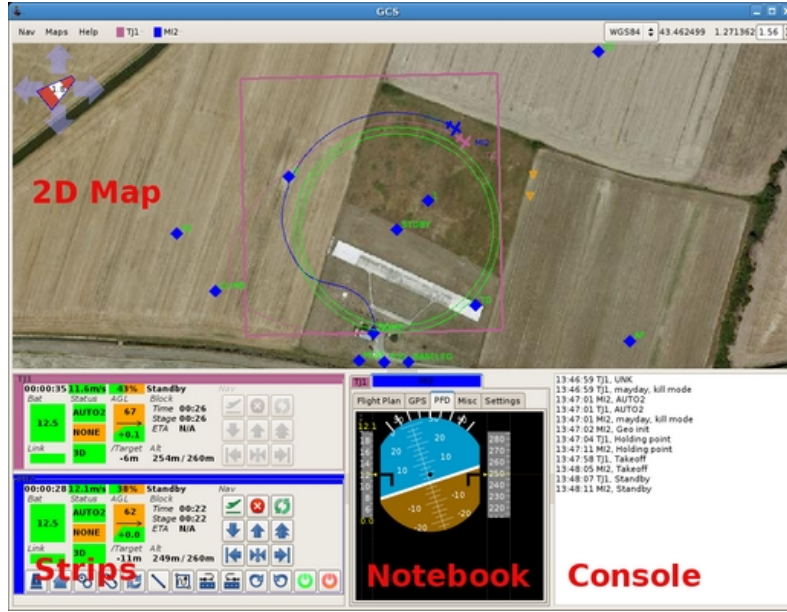


Fig. 3.2: The ground control station of the Paparazzi software. It displays information about the status of the UAV and provides functions for controlling the vehicle (from PaparazziUAV wiki [48]).

used across different platforms. Modules are written in the C programming language and are cross-compiled on the host PC to make them suitable for the UAV's processor. Afterwards, they are uploaded to the microprocessor of the UAV to run them *on board*. A downlink connection—from the UAV to the ground station—permits monitoring the state of the aircraft and includes information about speed, altitude, position, or battery status.

3.2 Preliminary Dataset Generation

The first pillar of the presented method is to shift computational effort to a pre-flight phase. Since the MAV will be used in a fixed environment, the results of these pre-calculations can be employed during the actual flight phase. Supervised machine learning methods need a training set to find a mapping from features to target values. In this first step, the goal is to label images with the physical x, y -position of the UAV at the time of taking the image. Therefore, a method for obtaining the physical position of the UAV is needed and GPS information is not available in the indoor environment. In the presented approach, the image is later converted to a texton histogram as described in the next section (Section 3.3).

One possible way to create the data set is to align the images with high-precision position estimates from a motion tracking system. The used camera forwards 640×480 pixel images in Y'UV422 color space—a three-channel color space that encodes gray-scale information in the channel Y and color information in the channels U and V. The x, y -position is broadcast to the UAV via the ground station, which is connected to the motion tracking system. The data set is created by saving the image with the corresponding position from

the motion tracking system on the MAV's hard disk. The approach yields high-quality training sets since motion tracking systems can track rigid bodies at a high frequency with an error tolerance of a few millimeters. Major disadvantages of the approach are that motion tracking systems are usually expensive and time-consuming to move to different environments. The workflow is illustrated in Figure 3.3.

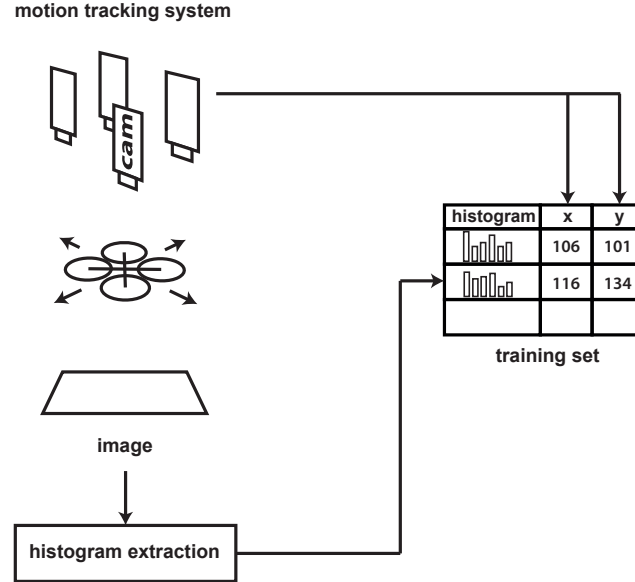


Fig. 3.3: Training dataset generation if the motion tracking system is used. The texton histograms of the camera images during flight are extracted and aligned with the highly accurate position estimates of the motion tracking system. The result is a high-quality training set of texton histograms and corresponding x, y -positions.

As an alternative, we sought a low-budget and more flexible solution. Of the presented approaches in Chapter 2, the homography-based approach (Section 2.1.3) promises the highest flexibility with a good accuracy but also requires the most processing time. Since fast processing time is not relevant during the pre-flight phase, the approach is well-suited for the problem. The required image dataset can be obtained by using images gathered during manual flight or by recording images with a hand-held camera. To get a hyperspatial image of the scene for creating a map, the images from the dataset have to be stitched together. The stitched image has a higher resolution than the single images and contains a greater range of detail (Figure 3.4). With certain software packages the images can be “orthorectified” by estimating the most probable viewing angle based on the set of all images. However, since a downward-looking camera is attached to the UAV, most images will already be roughly aligned with the z -axis, given slow flight [7]. We used the freeware software Microsoft Image Composite Editor (ICE) [37] for the stitching process. However, this closed-source software does not publish details about its used techniques. As an open-source alternative, the panorama photo stitching software *hugin* [13] is available. In our tests, Microsoft ICE yielded results of a better quality.

Keypoints of the current image and the stitched map image are detected and described using the SIFT algorithm. The keypoint sets are further refined using Lowe’s ratio test [35].

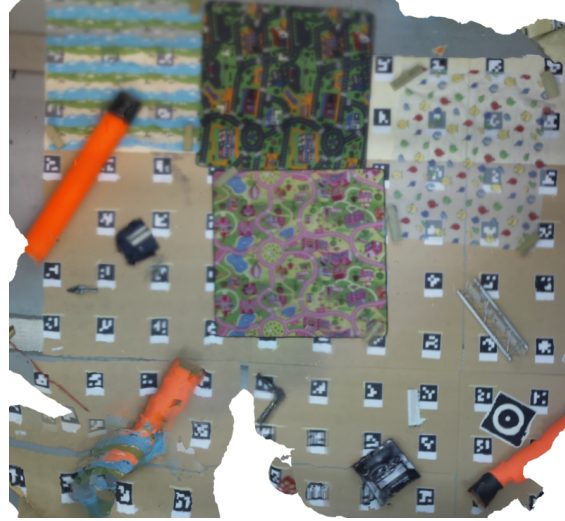


Fig. 3.4: This figure shows the created orthomap of a texture-rich floor. It is stitched together using 100 single images and represents a real world area of approximately 8×8 meters. Image distortions, non-mapped areas, and slightly skewed seams at several points are visible.

This is followed by a matching process, that identifies corresponding keypoints between both images. The matching uses a 'brute-force' matching scheme and every keypoint is compared to every other keypoint. These matches allow for finding a homography between both images. For determining the x, y -position of the current image, its center is projected onto the reference image using the homography matrix. The pixel position of the center in the reference image can be used to determine the real world position by transforming the pixel coordinates to real-world coordinates, based on the scale factors C_x and C_y , with $C_x = \frac{\text{width}(W)}{\text{width}(I)}$ and $C_y = \frac{\text{height}(W)}{\text{height}(I)}$, where W is the real-world dimension and I the digital pixel image. Performing this step for all recorded images yields a preliminary dataset of images—that is later converted to a dataset of texton histograms—labeled with x, y coordinates. An illustration of the approach can be seen in Figure 3.5.

The stitching process can be time-consuming and error-prone. It can be impeded by distortions and perspective transformations of the recorded images. To circumvent the need for stitching together multiple images, an image with a high-resolution camera from a top view point can be taken that captures the entire area in some environments. Yet another method could start with an existing image and modify the environment accordingly—for example by painting the floor or printing posters—to correspond to the image. The homography-based process introduces noise into the dataset, since it only has a limited accuracy (Section 2.1.3) that depends on the quality of the keypoint matches.

3.3 Machine Learning-based Approach and Filtering

In this section, the core of the developed algorithm is described: the implementation of the texton framework, consisting of the texton dictionary generation, the extraction of

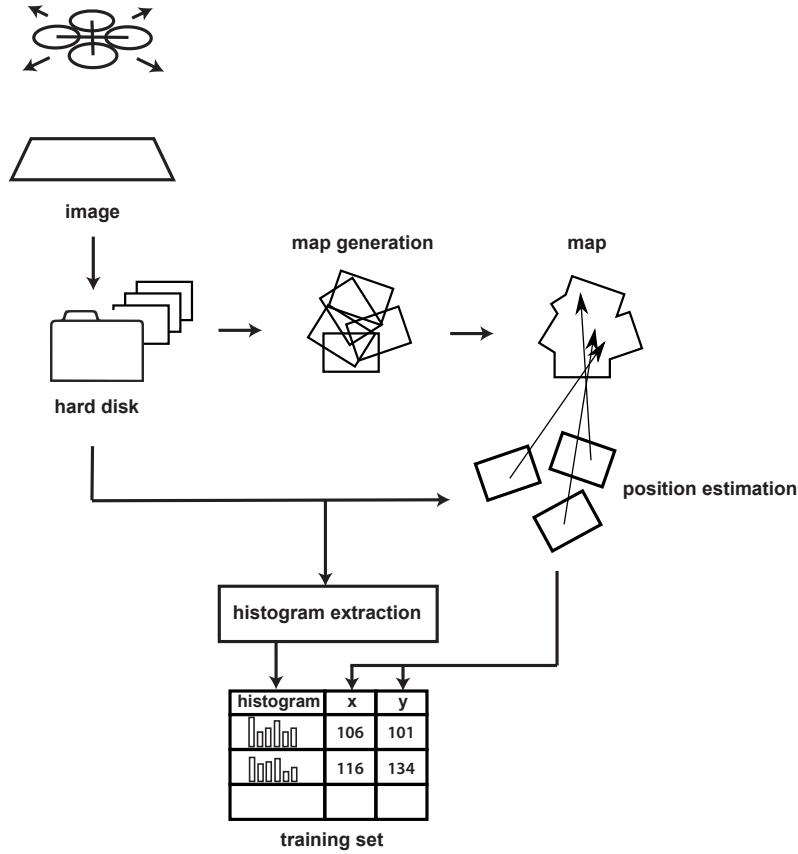


Fig. 3.5: The figure illustrates the training set generation when applying the homography-based approach. Images from an initial flight are stitched together to create an orthomap. The same images are used to detect and describe their keypoints using SIFT, followed by finding a homography between the keypoints of the flight images and the orthomap to obtain x, y -coordinates per image. The training set is created by extracting texton histograms from the images.

the histograms, the k -Nearest Neighbors (k -NN) algorithm, and the particle filter. The dictionary of textons constitutes the basis for determining the texton histograms. These histograms are used as features in the k -NN algorithm. The algorithm outputs k possible x, y -coordinates for a given image, which are forwarded to the particle filter to yield a final position estimate.

3.3.1 Texton Dictionary Generation

For learning a suitable dictionary for an environment, image patches were clustered. The resulting cluster centers—the prototypes of the clustering result—are the textons [47]. The clustering was performed using a competitive learning scheme with a “winner-take-all strategy,” a simple variant of a Kohonen network [32]. In the beginning, the dictionary is initialized with $n = 20$ random image patches from the first image, which form the first guess for cluster centers. Then, a new image patch x is extracted and compared to each texton d_j in the tentative dictionary using the Euclidean distance. The most similar texton d_r is the “winner.” This texton is then adapted to be more similar to the current

patch by calculating the difference in pixel values between the current image patch and the texton and updating the texton with a learning rate of $\alpha = 0.02$:

$$d_r := d_r + \alpha(x - d_r)$$

The first 100 images of each dataset were used to generate the dictionary. From each image, 1 000 randomly selected image patches of size $w \times h = 6 \times 6$ pixels were extracted, yielding $N = 100\,000$ image patches in total that were clustered. An example of a learned dictionary of grayscale textons can be found in Figure 3.6. For our approach, we also used the color channels U and V to obtain color textons.

Different maps and environmental settings require different texton dictionaries. If one would use the same dictionary for each map, it might happen that the histogram has only a few non-zero elements, and thus, cannot represent the variance in the map. While we set the number of textons to $n = 20$ for all maps, this parameter is also map-dependent and should ideally be adapted to the given map.

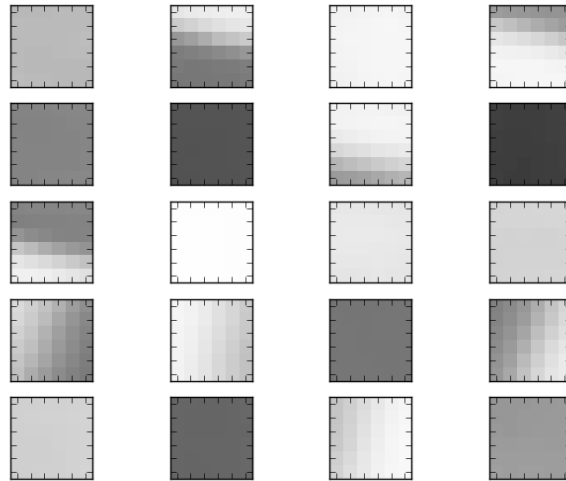


Fig. 3.6: The figure shows a dictionary consisting of 20 grayscale textons ($w \times h = 6 \times 6$ pixels).

3.3.2 Histogram Extraction

The images from the preliminary dataset (Section 3.2) are converted to the final training set that consists of texton histograms and x, y -values. It is the purpose of the conversion to obtain a more representative and dense description of an image, which should facilitate and speed-up recognition during the prediction step [26]. To extract histograms in the *full sampling* setting, a small window—or kernel—is convolved across the width and height of an image and patches are extracted from all positions. Each patch is compared with all textons in the dictionary and is labeled with the nearest match based on Euclidean distance comparing the pixels values in the channels Y, U, and V. The frequency of each

label is reported in the corresponding “bin” of the texton histogram. The histogram is normalized by dividing the number of cases in each bin by the total number of extracted patches to yield the relative frequency of each texton.

The convolution is a time-consuming step, since all possible combinations of width and height are considered: $(640 - w + 1) \cdot (480 - h + 1) = 301\,625$ samples are extracted. To speed up the time requirements of the histogram extraction step, the kernel can be applied only to randomly sampled image position instead [15]. This sampling step speeds up the creation of the histograms and permits a trade-off between speed and accuracy. The random sampling step introduces random effects into the approach. Therefore, to generate the training dataset, no random sampling was used to obtain high-quality feature vectors.

3.3.3 k -Nearest Neighbors (k -NN) algorithm

The k -Nearest Neighbors (k -NN) algorithm is the “machine learning-core” of the developed approach. Taking a texton histogram as input, the algorithm measures the Euclidean distance of this histogram to all histograms in the training dataset and outputs the k most similar training histograms and the corresponding x, y -positions.

While the k -NN algorithm is one of the simplest machine learning algorithms, it offers several advantages [33]. It is non-parametric, allowing for the modeling of arbitrary distributions. Its capability to output multiple predictions enables neat integration with the developed particle filter. Its simplicity combines with transparency: it allows for spotting the possible sources of error such as wrongly labeled training examples. k -NN regression often outperforms more sophisticated algorithms [12]. A frequent point of criticism is its increasing computational complexity with an increasing size of the training dataset. While the used training datasets consisted of fewer than 1000 images, resulting in short prediction times, time complexity can be reduced by storing and searching the training examples in an efficient manner, for example, with tree structures [6].

3.3.4 Filtering

Computer vision-based estimations are often noisy or ambiguous. Texton histograms obtained during flight will not perfectly match the ones in the training dataset: blur, lighting settings, viewing angles, and, other variables change the shape of the histograms.

To filter out outliers and smooth estimates, a popular filter choice is the Kalman filter. However, the Kalman filter is not able to represent multimodal probability distributions [17]. This makes it unsuitable for the presented *global* localization approach. The “naive” k -NN regression calculates the mean of the k outputs and forwards this value to the Kalman Filter. However, if the output values are distant, averaging them yields a value in the center between them, which is not likely to be the correct position (Figure 3.7). This approach

can lead to biased predictions, especially, if the model outputs belong to distant locations due to similar texton distributions at these positions.

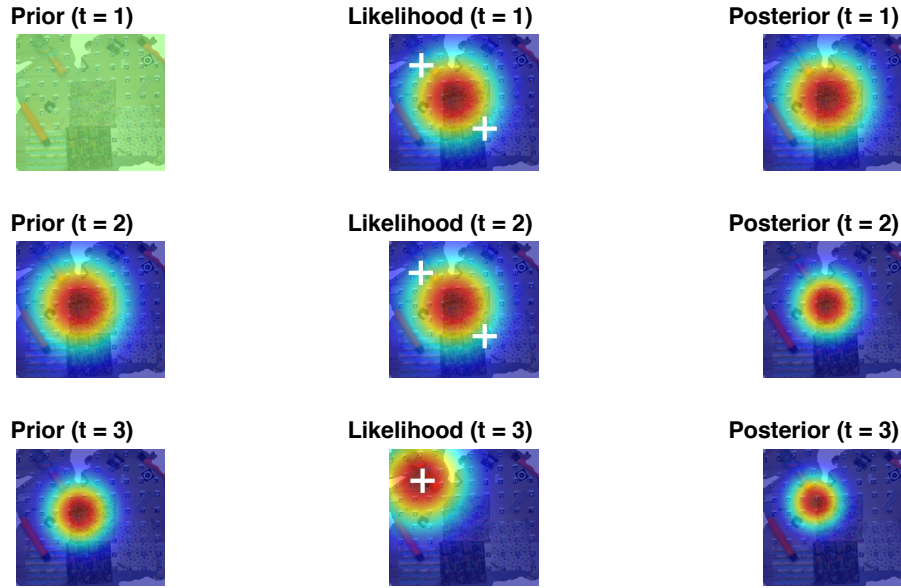


Fig. 3.7: The figure illustrates three time steps of a Kalman filter. The colors represent the probability of an x, y -position (red: high probability; blue: low probability). In timestep $t = 1$, the filter is initialized with an uniform prior and each position has equal probability. To incorporate measurement error, the likelihood (measurement model) is calculated using a Gaussian distribution that is centered around the mean of the $k = 2$ predictions (white crosses) from the k -NN algorithm. The posterior results from the multiplication of the prior with the likelihood and indicates the position estimates after one timestep. In the next timestep, the previous posterior becomes the new prior. The filter receives distant measurements in time steps $t = 1$ and $t = 2$ that are averaged to receive a position in the middle. In time step $t = 3$, the ambiguity is resolved but the filter only slowly adapts to the new position.

We decided to use a more sophisticated method to capture *multimodal distributions*. Given an adequate measurement model, a general Bayesian filter can simultaneously maintain multiple possible locations and resolve the ambiguity as soon as one location can be favored (Figure 3.8). In this case, the predictions of the k neighbors can be directly fed into the filter without averaging them first. The filter is able to smooth the estimations, handle uncertainty, and simultaneously keep track of several position estimates. However, a general Bayesian filter is computationally intractable. Therefore, a variant based on random sampling was used: the particle filter. While its computational complexity is still high compared to a Kalman filter, one can modify the amount of particles to trade off speed and accuracy and adapt the computational payload to the used processor.

The weighted particles are a discrete approximation of the probability density function (*pdf*) of the state vector (x, y -position of the MAV). Estimating the filtered position of the MAV can be described as $p(X_t | Z_t)$, where X_t is the state vector at time t and $Z_t = \mathbf{z}_1, \dots, \mathbf{z}_t$ are all outputs of the k -NN algorithm up to time t , with each $\mathbf{z}_i$ representing the k x, y -outputs of the algorithm at time i .

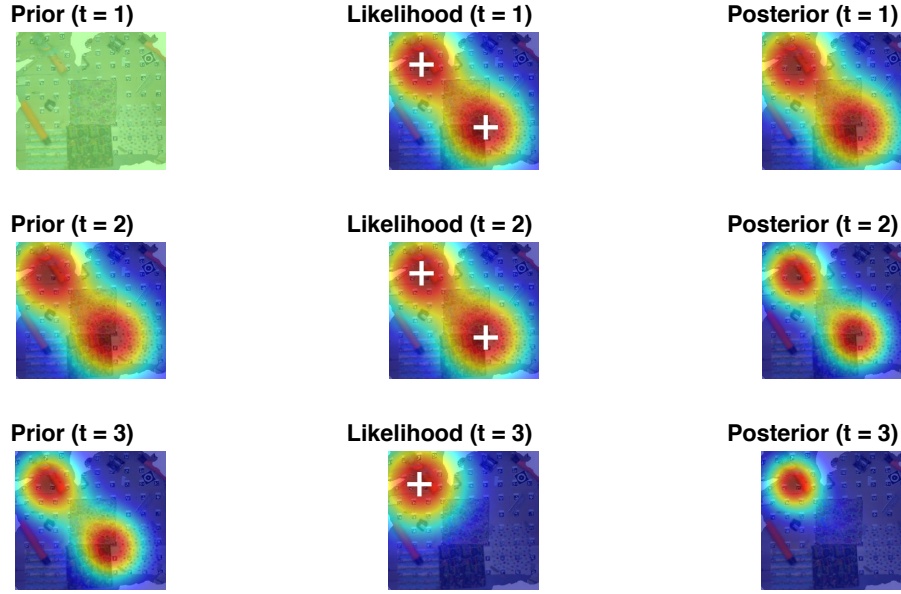


Fig. 3.8: Three time steps of a general Bayesian filter. The colors represent the probability of an x, y -position (red: high probability; blue: low probability). In contrast to the Kalman filter, the likelihood (measurement model) is calculated using a *mixture* of Gaussian distributions centered around the outputs the k -NN algorithm (white crosses). The filter can immediately resolve the ambiguity in time step 3 and the posterior gets updated accordingly.

The used particle filter is initialized with $M = 50$ particles at random x, y -positions. To incorporate the measurement noise for each of the k estimates from the k -NN algorithm, we developed a two-dimensional Gaussian Mixture Model (GMM) as measurement model. The GMM is parameterized by the variances $\Sigma^{[j]}, j \in \{1, \dots, k\}$ that are dependent on the rank j of the prediction of the k -NN algorithm (for example, $j = 2$ is the second nearest neighbor). The variance matrix $\Sigma^{[j]}$ specifies the variances of the deviations in x -direction and y -direction and the correlation ρ between the deviations. The values for $\Sigma^{[j]}$ were determined by calculating the variance-covariance matrix for the difference between the ground truth T from the motion tracking system and the predictions P_j of the k -NN algorithm: $\Sigma^{[j]} := \text{Var}(T - P_j)$.

In contrast to the measurement model, the used *motion model* is simple. It is solely based on Gaussian process noise and does not consider velocity estimates, headings, or control inputs. Its mean and variance are dependent on the expected velocity of the MAV. We used the forward difference $T_t - T_{t-1}$ to estimate the average movement and its variance-covariance matrix Σ_{process} between timesteps t and $t - 1$. While the employed motion model is simple, the developed software provides functionality for including an odometry-based motion model based on optical flow.

The algorithm of the developed particle filter is presented in the pseudo code in Algorithm 2. In the pseudo code, $\mathcal{X}$ is the list of particles, f the two-dimensional Gaussian probability density function, $z_t^{[i]}$ the i th neighbor from the k NN prediction, $x_t^{[m]}$ the m th particle at time t , and $w_t^{[m]}$ its corresponding weight.

Algorithm 2 Particle filter update

```
1: procedure PARTICLE_FILTER( $\mathcal{X}_{t-1}, z_t$ )
2:   ▷ Initialize particle list
3:    $\mathcal{X}_{\text{temp}} := \emptyset$ 
4:   for  $m = 1$  to  $M$  do
5:     ▷ Add random process noise (motion model)
6:      $x_t^{[m]} \leftarrow x_{t-1}^{[m]} + \mathcal{N}(0, \Sigma_{\text{process}})$ 
7:     ▷ Iterate over predictions from  $k$ -NN (measurement model)
8:      $w \leftarrow 0$ 
9:     for  $i = 1$  to  $k$  do
10:      ▷ Gaussian Mixture Model
11:       $w \leftarrow w + f(z_t^{[i]}, x_t^{[m]}, \Sigma_{\text{measurement}}^{[i]})$ 
12:      $\mathcal{X}_{\text{temp}} := \mathcal{X}_{\text{temp}} \cup (x_t^{[m]}, w)$ 
13:   ▷ Importance resampling
14:    $\mathcal{X}_t \leftarrow \text{RESAMPLING\_WHEEL}(\mathcal{X}_{\text{temp}})$ 
15:   return  $\mathcal{X}_t$ 
```

The “resampling wheel” [45] (Algorithm 3) performs the importance resampling step. Its underlying idea is that the particles are arranged in a “wheel,” with each particle occupying a slice that corresponds to its weight. The particles are then resampled with a probability proportional to the area of the slices. This step ensures that particles with a low weight are removed and replaced with well-performing ones. Otherwise, the algorithm might “collapse” when all but one particle have a low weight.

Algorithm 3 Resampling wheel

```
1: procedure RESAMPLING_WHEEL( $\mathcal{X}_{\text{temp}}$ )
2:   ▷ Initialize particle list
3:    $\mathcal{X}_t \leftarrow \emptyset$ 
4:   ▷ Sample random index from the number of particles
5:   sample  $i \sim M \cdot \mathcal{U}(0, 1)$ 
6:    $\beta \leftarrow 0$ 
7:   for  $m = 1$  to  $M$  do
8:      $\beta \leftarrow \beta + \mathcal{U}(0, 1) \cdot 2 \cdot \max(w_t)$ 
9:     while  $\beta > w_t^{[i]}$  do
10:       $\beta \leftarrow \beta - w_t^{[i]}$ 
11:       $i \leftarrow (i + 1) \bmod M$ 
12:      $\mathcal{X}_t \leftarrow \mathcal{X}_t \cup \mathcal{X}_{\text{temp}}^{[i]}$ 
13:   return  $\mathcal{X}_t$ 
```

With the GMM, the information of all k neighbors can be used, yielding a possibly multimodal distribution. While a multimodal distribution allows for keeping track of several possible positions, certain subsystems—for example a control loop—often need *one* point estimate. Using a weighted average of the particles would again introduce the problem that it could fall into a low density region (an unlikely position). Instead, we used a maximum a posteriori (MAP) estimate, as described by Driessen and Boers [19]. This

approach is a discrete approximation of the true MAP estimate [19]. It uses the following formula to obtain the MAP estimate X_t^{MAP} —the “final” x, y -position:

$$X_t^{\text{MAP}} = \arg \max_{x_t^{[i]} \in \{i=1, \dots, M\}} \sum_{j=1}^M f(x_t^{[i]}; x_{t-1}^{[j]}, \Sigma_{\text{process}}) w_{t-1}^{[j]}$$

Therefore, the final position estimate is equal to the position of one of the particles.

The estimation of *uncertainty* is a core part of the developed approach, due to its importance for safety and accuracy. Therefore, uncertainty was modeled using the spread of the particles—as expressed by their variance in x -direction and y -direction. Initially, we planned to include the distance between the current histogram obtained from the camera image and each of the k neighbors from the training set as confidence value. One could thus reduce the measurement noise if a high similarity between the current histogram and a training histogram is achieved. While we found no correlation between these variables (Section 4.3), we still provide the functionality for incorporating the distance in the developed software. We also tried to use the amount of detected keypoints (K) as a confidence value for the quality of the sample in the training set if the homography-based approach is used for labeling. Again, no linear relationship between K and the error in x -direction (X) and the error in y -direction (Y) could be found.

3.4 Map Evaluation

3.4.1 Evaluation Scheme

The performance of the developed method depends on the environment: a texture-rich environment without repeating patterns will be better suited than a texture-poor environment. Ideally, one would like to know if the algorithm will work in a given environment. Therefore, we propose an evaluation scheme that can compare different environments and areas within an environment. This scheme assigns a global fitness value or global loss value to a “map”—expressed as dataset $\mathcal{D}$ consisting of N texton histograms h_i and corresponding x, y -coordinates $\text{pos}_i = (x_i, y_i)$. The fitness value is intended to be proportional to the accuracy that can be expected when using this dataset as training set for the developed localization algorithm. The scheme allows for inspecting the dataset and detecting regions within the map that are responsible for the overall fitness value.

The idea behind the global loss function L is that histograms h_i and h_j in closeby areas should be similar and the similarity should decrease with increasing distance of the corresponding x, y -coordinates pos_i and pos_j . Therefore, the approach is based on the difference between *actual* and *ideal* texton histogram similarities in a dataset. The ideal texton similarity distribution is modeled as a two-dimensional Gaussian distribution around each x, y -position in the dataset (Figure 3.9). Using this idea, a histogram is compared to all others by comparing expected similarities to actual similarities. This results in a loss

value per sample of the dataset (local loss). Applying the algorithm to each sample in the dataset yields the global loss of a dataset. A visualization of the global loss is illustrated in Figure 3.10.



Fig. 3.9: *Left:* Actual similarity between histogram h_i (pos_i : white cross) and all other histograms; the heatmap shows low similarity in blue and high similarity in red. For the visualization, the actual similarities were smoothed with a Gaussian filter. *Middle:* Ideal histogram similarity distribution for the given position pos_i . Histograms h_j taken at closeby positions should have a high similarity to h_i . The farther away the position pos_j , the lower the similarity between h_i and h_j should be. *Right:* The difference between the actual and the ideal similarity shows regions that do not follow the ideal similarity distribution for histogram h_i (high loss: red; low loss: blue)

The method uses the cosine similarity (CS) to compare histograms:

$$CS(h_i, h_j) = \frac{h_i^T h_j}{\|h_i\| \|h_j\|}$$

The cosine similarity has the convenient property that its values are bounded between -1 and 1 . In the present case, since the elements of the histograms are non-negative, it is even bounded between 0 and 1 . Let the function f describe the non-normalized one-dimensional Gaussian probability density function:

$$f(x; \mu, \sigma) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Since we assume that the ideal similarity in x -position is independent of the y -position, the ideal two-dimensional similarity function $d_e(\text{pos}_i, \text{pos}_j; \Sigma)$ can be modeled as the product of the respective one-dimensional function f :

$$d_e(\text{pos}_i, \text{pos}_j; \Sigma) = f(x_i; x_j, \sigma_x) \cdot f(y_i; y_j, \sigma_y)$$

This function is also bounded between 0 and 1 , which makes the functions d_e and CS —ideal similarity and actual similarity—easily comparable. In summary, we propose the following global loss function (L) for evaluating a given dataset ($\mathcal{D}$):

$$L(\mathcal{D}) = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N CS(h_i, h_j) - f(x_i; x_j, \sigma_x) \cdot f(y_i; y_j, \sigma_y)$$

The simple difference—in contrast to least absolute deviations or least square errors—ensures that similarities that are *less* similar than the ideal similarity *reduce* the loss.

Therefore, a high variation in texture is always seen as “positive”. The variances σ_x and σ_y specify the dimension of the region, where similar histograms are desired. The lower their value, the more focused the ideal similarity will be, requiring a high texture variety for getting a low loss value. A high value might overestimate the suitability of a dataset. While the approach is relatively robust to the choice of the parameter values, we still need to find a heuristic for suitable values.

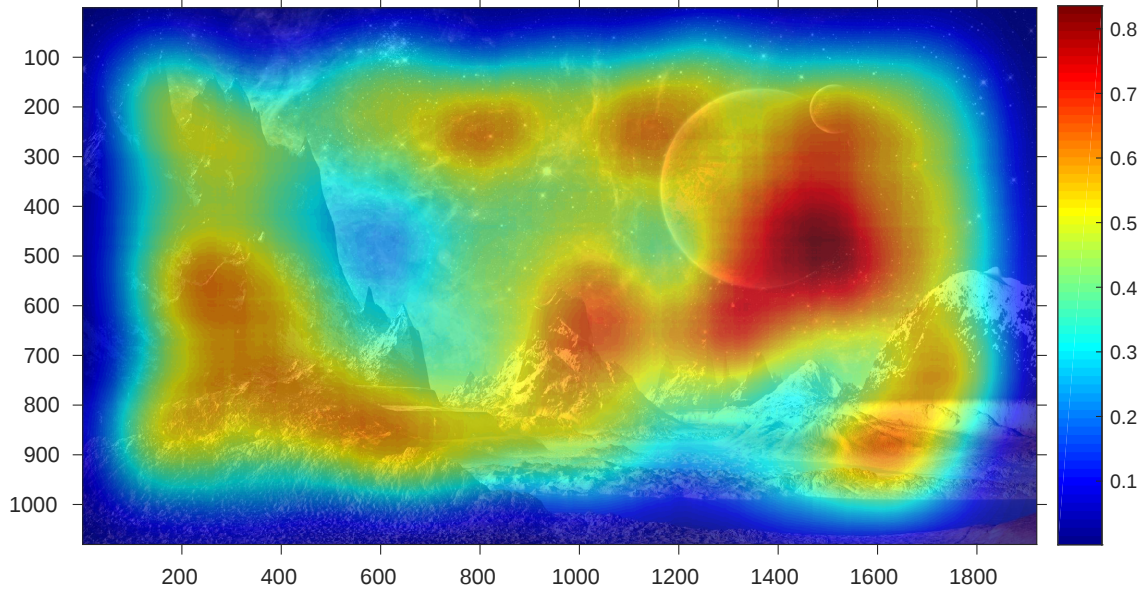


Fig. 3.10: The figure shows the loss of a map: the regions that did not follow the ideal similarity pattern are displayed in red. For the visualization, the loss values per sample in the dataset were smoothed with a Gaussian filter. This assigns a loss value to each x, y -position of the map. The synthetic data generation tool was used for generating the underlying dataset (Section 3.4.2).

3.4.2 Synthetic Data Generation

To compare environments before actually flying in them, a software tool was developed that creates synthetic images to simulate those taken during an actual flight. The tool generates the patches based on perspective transformations of an image. Examples of generated images are displayed in Figure 3.11.

The application allows for comparing and predicting the performance of different “maps” as specified by an image. The software is written in C++ and OpenCV 3.0.0. The algorithm simulates a simple camera model that moves above the image (Figure 3.12). It generates a specified amount of image patches using random values—sampled from uniform and normal probability distributions—for various parameters:

- rotational angles: roll α , pitch β , yaw γ
- translational shifts: dx , dy , dz
- brightness: addition of constant value b to all pixels

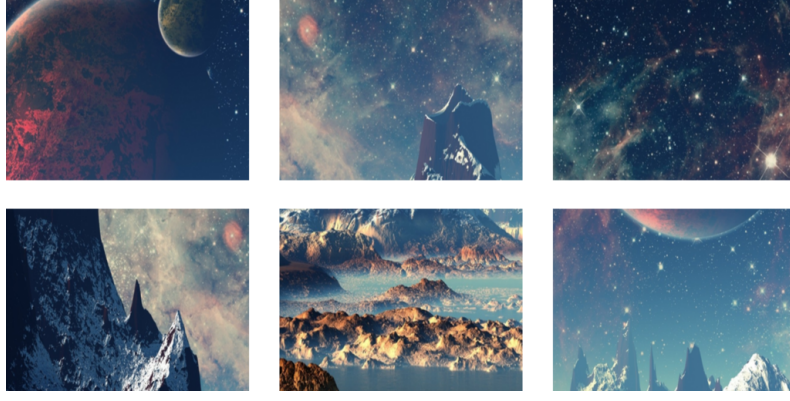


Fig. 3.11: Six image patches generated by means of the synthetic data generation tool.

- contrast: multiplication of pixel values with constant value c
- blur: application of a box filter with kernel size $kw \times kh$

By finding a homography M —a perspective transformation specified by rotational and translational parameters—one can obtain image patches and consequently texton histograms to create a training dataset. The tool labels the generated patches with the corresponding simulated x, y -position of the camera model, which represents the position of the UAV.



Fig. 3.12: Illustration of the camera model for the synthetic flight. The developed tool extracts image patches from an given image to simulate those taken with the bottom camera of the MAV during an actual flight.

The steps for specifying the homography are outlined in the following. The implementation is partly based on work by Jepson [29]. Hartley and Zisserman [27] describe multiple view geometry and image transformations in computer vision in detail. To simulate camera movements in the 3D world, a 2D to 3D projection of the image is performed first, using the matrix P_3 , with the width w and height h of the image:

$$P_3 = \begin{bmatrix} 1 & 0 & -\frac{w}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The result is a 3D space with the center of the image as point of origin. The camera rotations are specified by the rotation matrix $R = R_x \cdot R_y \cdot R_z$. By building rotation

matrices R_x , R_y , and R_z around the axes x , y , and z , the rotations with the corresponding angles α , β , and γ can be defined separately:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The 3D translational matrix T specifies the location of the camera in world coordinates:

$$T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, a rotation followed by translation can be specified by matrix H :

$$H = T \cdot R$$

However, this matrix H describes how the world is transformed relative to the camera coordinates, while the position of the camera is fixed. Instead, we would like to specify the camera movement relative to a fixed world. To this end, the inverse of H is needed:

$$H^{-1} = (T \cdot R)^{-1} = R^{-1} \cdot T^{-1}$$

The transposed rotation matrix is equal to its inverse: $R' = R^{-1}$. The inverse of T negates the translations:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To obtain a 2D image again, a projection from 3D space to 2D is applied using the matrix P_2 . The matrix needs the focal distance f (the distance between camera and image).

$$P_2 = \begin{bmatrix} f & 0 & \frac{w}{2} & 0 \\ 0 & f & \frac{h}{2} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The ratio between dz and f specifies the size of the patch. The final 3×3 perspective transformation matrix M becomes:

$$M = P_2 \cdot R^{-1} \cdot T^{-1} \cdot P_3$$

The pixel values of the image patch at position x, y are calculated by applying the perspective transformation M to the original image:

$$\begin{aligned} \text{patch}(x, y) &= \text{original}(x', y') \\ &= \text{original} \left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}} \right) \end{aligned}$$

For modifying brightness and contrast, each pixel value is transformed with

$$\text{patch}(x, y) := c \cdot \text{patch}(x, y) + b$$

The blurring is performed by convolving the image patch with a box filter:

$$\frac{1}{kw \cdot kh} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

The script provides a command line interface for selecting the original image and the amount of desired image patches. It creates a dataset of image patches and a comma-separated values (CSV) file that specifies the sampled values from the random distributions per patch.

In this chapter, the setup of the experiments is presented and the results are described. The flight tests were carried out in an indoor flight arena of the TU Delft: the “CyberZoo” (Figure 4.1).

To compare estimates from the developed framework to a ground truth, we employed the motion tracking system OptiTrack [38]. This system uses an array of cameras and reflective markers attached to the body of the MAV to track the MAVs at a high frequency within an error of a few millimeters. We used OptiTrack as MAV guidance system for autonomous flight, yielding accurate and stable control. The *Parrot BeboP Drone* was the prototype for all flight tests.

4.1 Determining the Number of Image Patches

The computational complexity of the developed framework can be modified by changing the number of extracted image samples in the *random sampling* step of the texton histograms creation. The goal is to use as few samples as possible, while still obtaining an adequate localization accuracy.

To determine a suitable number of extracted samples, we compared the influence of random sampling by measuring the cosine similarity between histograms based on *random sampling* and the histogram obtained by *full sampling* based on $N = 5\,000$ images that we obtained from different environments. The independent variable is the number of extracted patches per image.

Figure 4.2 displays the results: the mean cosine similarity of histograms as a function of the number of samples and the corresponding standard deviations.



Fig. 4.1: The indoor flight arena—the “CyberZoo”—at the TU Delft.

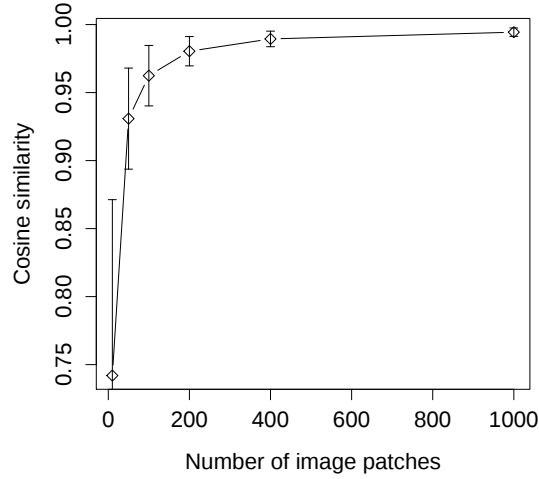


Fig. 4.2: Mean cosine similarity between histograms obtained by random sampling to the histograms obtained by full sampling (301 625 samples) as a function of the number of extracted samples. The error bars show the standard deviation of the cosine similarity. The squares indicate the positions at which the dependency was evaluated.

4.2 Determining k in the k -NN algorithm

The number of nearest neighbors, k , influences the predictions of the k -NN algorithm: a low value can lead to unstable predictions, while a higher value reduces the variance in the predictions. However, a too high value can lead to a high bias resulting in inaccurate predictions. In this experiment, we analyzed the influence of the number of nearest neighbors, k , on the distance between ground truth and the predictions. The value of the measurement model was set to a constant value for all neighbors: $\forall j \in \{1, \dots, k\} : \Sigma^{[j]} = \mathbf{c}$.

This ensures that the effect of the number of neighbors is measured without influence of the underlying measurement model. The results are displayed in Figure 4.3.

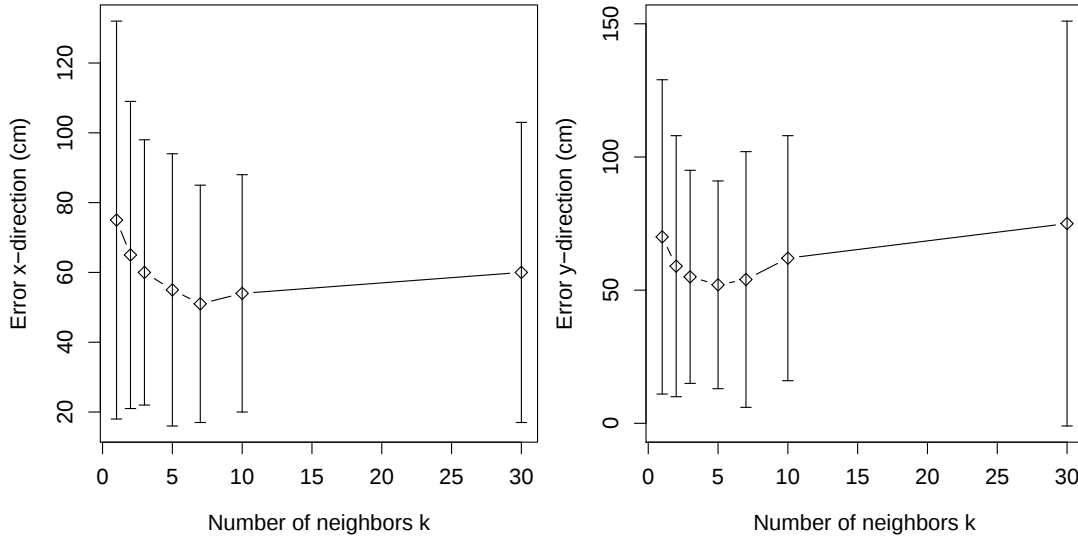


Fig. 4.3: The distance between ground truth and predictions in x -direction and y -direction as a number of the neighbors in the k -NN algorithm. The error bars show the standard deviation of the distances.

4.3 Histogram Distance and Measurement Error

As described in Section 3.3.4, we planned to include the distance to the neighbors in the k -NN algorithm as confidence value for the predictions. Figure 4.4 shows the dependence structure between the distance to the nearest neighbor and the error in x -direction and y -direction, measured as the distance in cm between the ground truth and the label of the nearest neighbor.

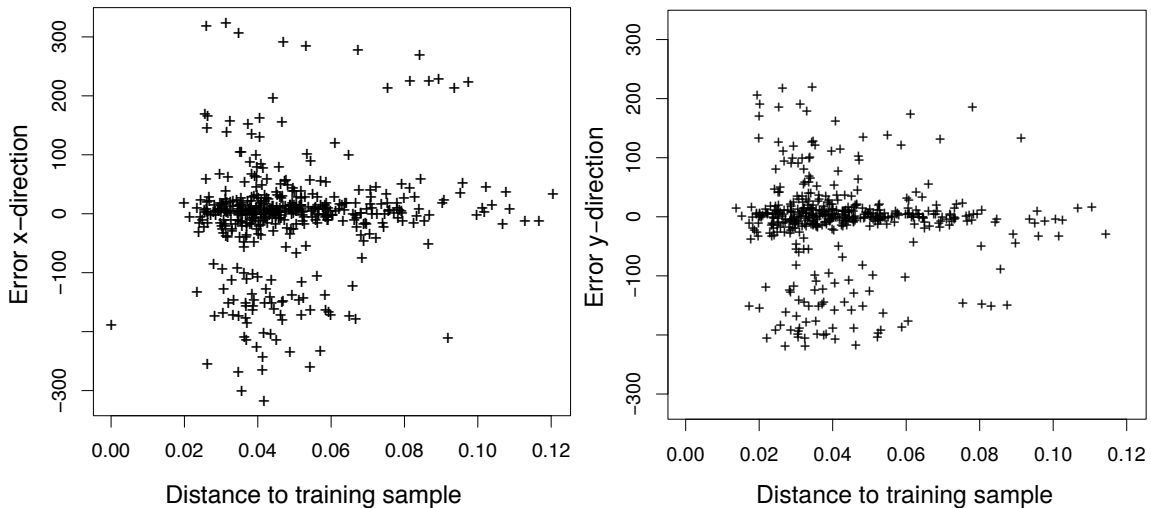


Fig. 4.4: Distance between ground truth and predictions from a $k=1$ -NN model in x -direction (Left) and y -direction (Right) as a function of the distance to the closest training sample.

4.4 Flight Tests

In these experiments, the MAV was guided along flight plans using the motion tracking system. If not otherwise stated, we used the following default values for the parameters in our framework.

- number of samples in the histogram extraction step: 400
- number of textons in the dictionary: 20
- number of particles of the particle filter: 50
- number of histograms / images in the training set: 800
- number of histograms / images in the test set: 415
- number of neighbors in the k -NN algorithm: 5

Map-dependent texton dictionaries were by used and created by conducting an initial flight over the respective maps.

4.4.1 Training Set based on Motion Tracking System

In this experiment, the position estimates were calculated on board of the MAV using the texton-based approach with the particle filter. The Euclidean distances between the estimates of the motion tracking system and the texton-based approach were measured in x -direction and y -direction.

The training dataset was composed of 800 texton histograms with corresponding x, y -coordinates that were obtained from the motion tracking system. The images were recorded in a 5×5 meter area at a height of approximately one meter in a time span of one hour before the experiment to keep environmental factors roughly the same.

The results can be found in Table 4.1. They are based on 415 images, which corresponds to a flight time of approximately 35 seconds.

Tab. 4.1: Differences between ground truth and the developed framework in x -direction and y -direction. The used training set was based on the position estimates of the motion tracking system (Section 3.2).

	x-position	y-position
Error in cm	61	59
STD in cm	39	39

4.4.2 Baseline: Homography-based Approach

To find a baseline for our approach and to provide a homography-based training set, we used the homography-based approach to estimate x, y -coordinates in the same environment and based on the same images as the texton-based framework. The required hyperspatial

image (Figure 4.5) of the environment was stitched together using 800 images and the software Microsoft ICE.



Fig. 4.5: The created map that was stitched together using 800 images. The “non-mapped” area in the center of the image is a result of the set flight path. An image distortion can be seen at the right-hand side, where the landing spot appears twice, while in reality, only one circle was visible.

We estimated the x, y -coordinates of the 415 test images using the homography-based approach (3.2) and compared the predictions to the ground truth. The predictions were not filtered. The results can be found in Table 4.2.

Tab. 4.2: Error statistics for the homography method.

	x-position	y-position
Error in cm	31	59
STD in cm	68	77

4.4.3 Training Set based on Homography-finding Method

In this experiment, the training dataset was created by estimating the x, y -positions of the 800 training images using the homography-finding method from the previous section and the same hyperspatial image. Apart from that, the settings are the same as in Experiment 4.4.1.

Tab. 4.3: Differences between ground truth and the developed framework in x -direction and y -direction. The used training set was based on the position estimates of the homography-finding method (Section 3.2).

	x-position	y-position
Error in cm	54	97
STD in cm	41	61

4.4.4 Triggered Landing

For the triggered landing experiment, the MAV was guided along random flight paths, which covered a 5×5 meter area; during navigation, the MAV was programmed to land as soon as its position estimates were in a “landing zone”: an x, y -position with a specified radius r . A safety criterion was introduced such that the landing is only performed if the standard deviations of the particles in x -direction and y -direction are below thresholds θ_x and θ_y . We set the parameters to $\theta_x = \theta_y = 60$ cm. The x, y -coordinate of the circle was specified in the flight plan; the radius was set to $r = 60$ cm. We performed six triggered landings; after each landing, the x, y -center of the zone was randomly set to another position in the map. For the texton framework, the same training set as in Experiment 4.4.1 was used.

Four out of six landings were correctly performed in the landing area. The distances of the two outliers were 14 cm and 18 cm, measured as distance to the circumference of the landing area.

4.5 Speed versus Accuracy Trade-Off

Adapting the frequency of the main loop of the developed approach to make it suitable for different platforms with varying processing power is one of its core parts.

Figures 4.6 and 4.7 show the speed versus accuracy trade-off as a function of the used particles and of the used samples in the histogram extraction step, respectively. As a reference, the frequency using *full sampling* in the histogram extraction step was 0.1 Hz. The above stated *default* values were used for the *ceteris paribus* assumption, when varying the parameters.

While the bottom camera of the Parrot Bebop Drone has a frequency of 30 Hz, the Paparazzi software currently only receives the images with a frequency of 12.5 Hz. Therefore, the maximum achievable frequency without further image processing is 12.5 Hz, which is the baseline for the conducted experiments.

Figure 4.8 illustrates the frequency as a function of the used histograms in the k -NN algorithm. We did not compare the frequency to the distance between ground truth and the predictions, since our training dataset did not contain more than 800 histograms.

After having received the image, the processing time of the presented algorithm using the *default* parameter values is 32 ms, which includes the histogram extraction (16 ms) as well as the k -NN predictions, the filtering and the output of the best x, y -coordinate (16 ms).

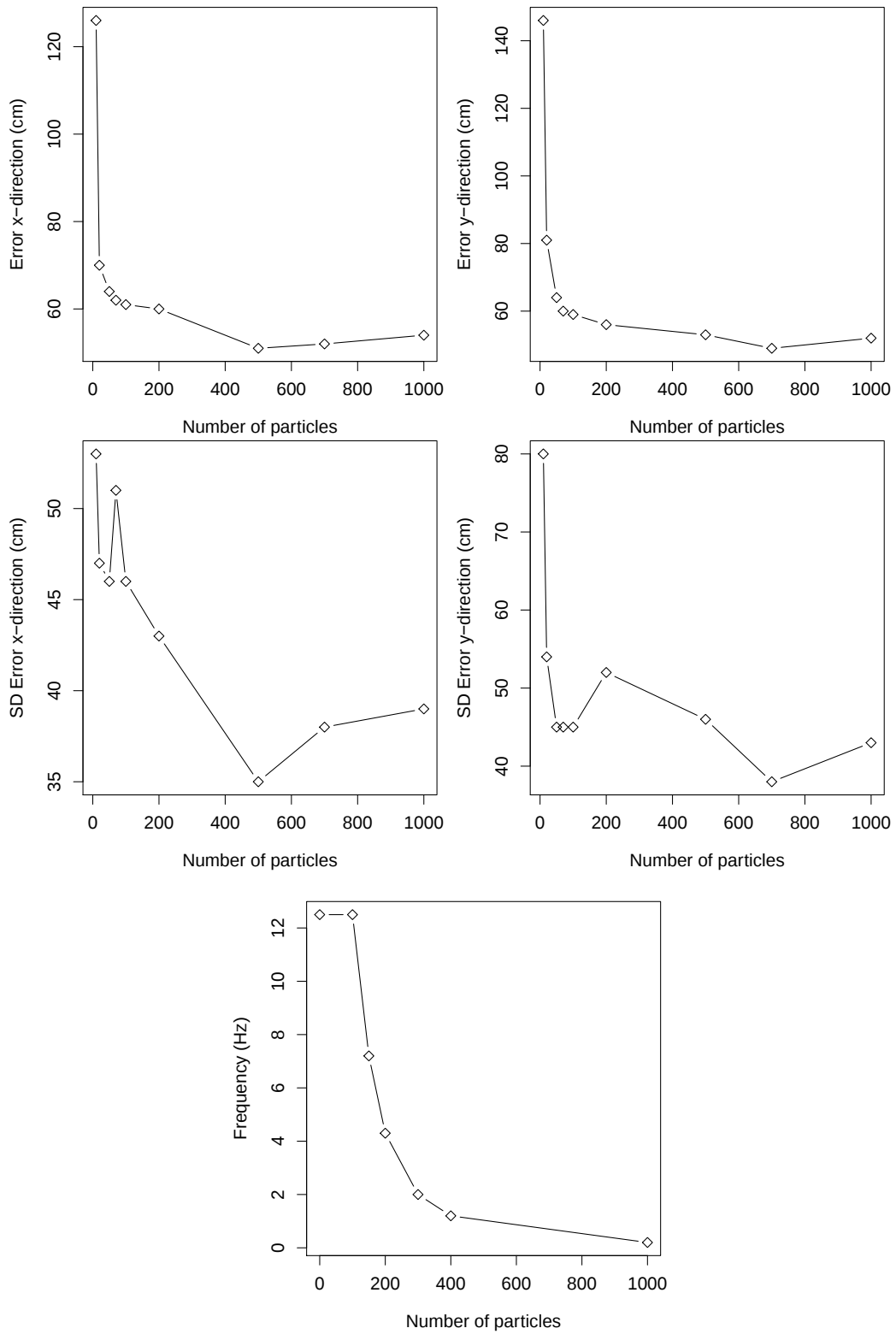


Fig. 4.6: Top row: Distance in cm between ground truth and the predictions in x -direction and y -direction as a function of the **number of used particles** in the particle filter. Middle row: Standard deviations of the distances. Bottom row: Frequency of the algorithm.

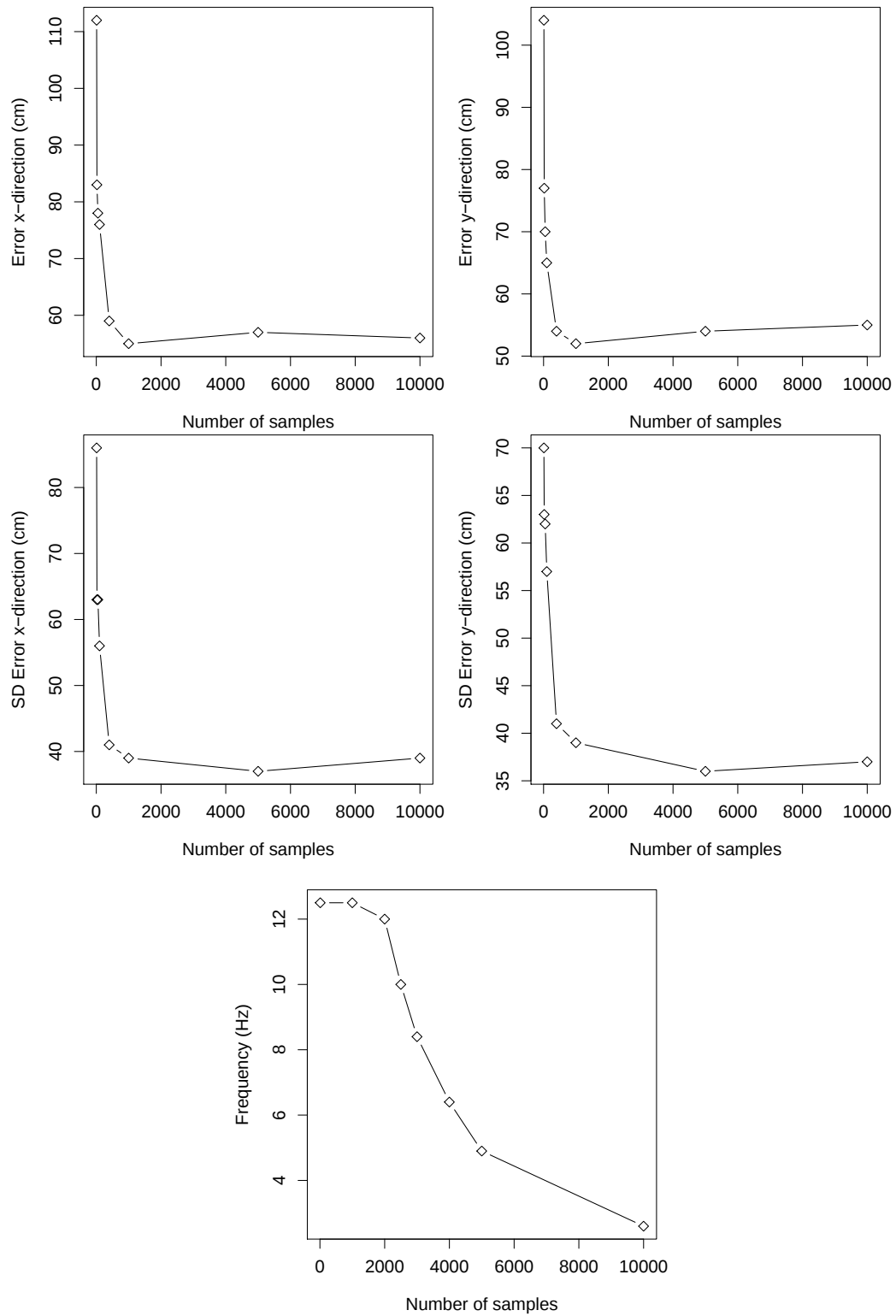


Fig. 4.7: Top row: Distance in cm between ground truth and the predictions in x -direction and y -direction as a function of the **number of used samples** in the histogram extraction step. Middle row: Standard deviations of the distances. Bottom row: Frequency of the algorithm.

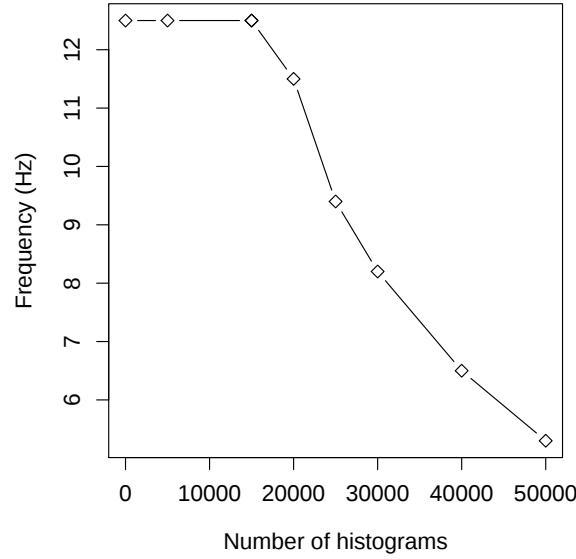


Fig. 4.8: Frequency of the main loop as a function of the number of histograms in the training set.

4.6 Comparing Maps

For the map comparison, 46 possible maps have been collected using the search term ‘wallpaper’ in Google’s image search. This search term was used, since it is (i) a general term, without any specific image categories, (ii) wallpapers are likely to have a high resolution, and (iii) wallpapers are often visually pleasant. Only images with a minimum resolution of 1920×1080 pixels were chosen. Images with a higher resolution were converted to 1920×1080 pixels. For each image, we generated 1000 random image patches using the synthetic data generation tool described in Section 3.4.2, followed by the histogram extraction (Section 3.3.2) using the same texton dictionary for all maps. This yielded a labeled dataset of histograms and corresponding positions. For each map, we determined the expected overall loss based on the method described in Section 3.4.2. The compared maps were then sorted according to their estimated loss.

In Table 4.4, the results of the map evaluation procedure for the $N = 46$ maps are shown.

Tab. 4.4: Results of the map evaluation procedure on synthetic data

Statistic	Value
mean	0.81
median	0.79
standard deviation	0.10
max	0.98
min	0.58

Figure 4.9 shows the map with the highest global loss value and the map with the lowest global loss value.

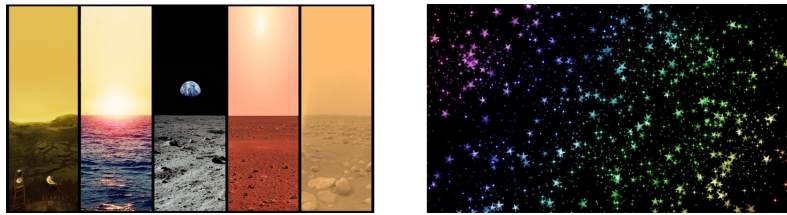


Fig. 4.9: *Left:* Image with the lowest loss value [41]; *Right:* Image with the highest loss value [44]

In this chapter, the results of the experiments are discussed with regards to accuracy and frequency. Afterward, in the General Discussion, we address the research questions and indicate directions for future research.

The comparison between random sampling and full sampling (Section 4.1) has shown that only a small part of the maximum amount of samples in the histogram extraction step is necessary. In fact, $\frac{1000}{301625} = 0.33\%$ of the maximum amount of samples suffice to achieve cosine similarities greater than 99 % between the texton histograms. This sets the stage for large speed-ups during live operation and allows for executing the main loop of the algorithm in real-time. In contrast, the full sampling technique—where a kernel is convolved over the entire image—would require 10 seconds to process one image.

The experiments addressing the “Speed versus accuracy trade-off” (Section 4.5) show that with an increasing accuracy of the approach, the frequency of the algorithm decreases. However, the errors reach a plateau after which no large improvements can be expected at the lower end of parameter ranges. By optimizing the parameters, one can obtain localization errors below 50 cm with the developed approach.

The flight tests showed the real-world suitability of the method. It yields slightly less accurate results than the unfiltered homography-finding method. While we did not test the frequency of the homography-based approach on board of an MAV, on a desktop computer, it took 200 ms per image. Therefore, the developed algorithm runs at a much higher frequency. The training set generation based on the homography method yielded higher errors in the flight test. Filtering the estimates of the homography-method first could improve the accuracy. The triggered landing (Experiment 4.4.4) showed good accuracy: while most landings were triggered inside the landing zone, two out of the six landings were outliers. However, their distance to the landing area were rather small, with an average distance of 16 cm.

The evaluation of different maps using the synthetic data identified high differences between the evaluated images. The range of losses from 0.58 to 0.98 underlines the varying suitability of different maps for the developed algorithm. The image with the minimum and the one with the maximum loss value are shown in Figure 4.9. The different patterns of the images are visible: while the image with the minimum value roughly fulfills the desired properties—closeby areas have similar color values, distant areas are dissimilar, the image with the maximum loss is mainly black resulting in similar histograms all over and leading to high loss values. The image with the lowest loss value still has a loss of 0.58 and, therefore, leaves room for improvement. Adapting the texton dictionary to each image to make them better suited for the respective image could lead to different results.

While we compared the settings of different parameters, there are no generally optimal parameters for the presented framework: setting the number of textons, the number of images patches, or the number of neighbors is dependent on the environment and the size of the training dataset. The parameters have to be adapted to the particular environment.

5.1 General Discussion

In this thesis, we addressed the problem of estimating x, y -coordinates in real-time and on-board of an MAV. We identified a computer vision-based approach to be most suitable due to the limited payload capacity of MAVs. We recapitulate the research questions (RQs):

- **RQ 1:** *“Can 2D positions be estimated in real-time using a machine learning approach on a limited processor in a modifiable indoor environment?”*
- **RQ 2:** *“How can we predict and evaluate the suitability of a given map for the developed localization approach?”*

Regarding RQ 1, the conducted experiments provide supportive evidence that a texton-based machine learning approach is able to accomplish real-time indoor localization. The developed algorithm runs with a high frequency—32 ms per iteration with the default settings—on a single board computer with limited CPU. Shifting processing power to an offline training step by creating a training set and relying on random sampling are the cornerstones of the developed algorithm. For finding a way to modify the environment in a certain direction, we introduced RQ 2. In the scope of this research question, we developed a framework for evaluating different maps based on an ideal similarity of histograms. To answer RQ 2, we still have to evaluate, whether the regions with a high loss value within an environment are actually the regions where the algorithm performs worse. We also still need to investigate whether the results of the synthetically generated images generalize to the real-world. The requirement here is that maps that follow the ideal similarity distribution in the synthetically generated images also follow this distribution after being recorded with a camera. Or stated differently, for maps with a low loss value, distant image positions should neither have similar histograms in the synthetic images nor in real-world images.

Despite the overall promising results of our localization algorithm, we noticed drawbacks during the flight tests and identified several directions for future research that are described in what follows.

The accuracy—that is the difference between the estimates of the motion tracking system and the texton-based approach— could be further improved by combining the presented global localization technique with a local technique. To this end, odometry estimates using optical flow or the inclusion of data from the inertial measurement unit (IMU)—which is a

sensor that measures the angular displacement and acceleration of the MAV— could be suitable.

The current implementation assumes constant height up to few centimeters and only small rotations of the MAV. While a quadroter can move in every direction without performing yaw movements, other MAVs or the use of the front camera for obstacle avoidance could require them. The inclusions of images of arbitrary yaw movements into the dataset would inflate its size to a great extent. This could lead to a deterioration of the accuracy and increase the time-complexity of the k -NN algorithm. Instead, a “derotation” of the camera image could be performed to align it with the underlying images of the dataset. We developed a prototype for this procedure.¹ In the prototype, the rotation is done on the basis of data from the IMU.

While the current map evaluation approach used existing images, it could also serve as a fitness function for an optimization approach—for example, an evolutionary algorithm—that modifies a given image. While the solution to a loss value of zero or near zero might be unique and independent of the original image, a higher loss value might change the initial image only to a certain extent, yielding an “improved version of the image”, which is better suited for the developed algorithm and still visually pleasant.

Initially, we planned to use the synthetic images (Section 3.4.2) directly as training data. This was not successful and not further investigated: the reality gap between the synthetic data and real-world data was too large. Figure 5.1 shows an example of two image patches, one synthetically generated, one taken with the camera of the MAV. While the patches can be easily identified as similar for human eyes, the texton maps, where different colors represent different textons, are dissimilar. Blur, lighting settings, and camera intrinsics modify low-level features of the image to a strong extent. The texton images in the figure show that corresponding regions are classified into different textons, resulting in different histograms. This makes the transfer from the synthetic data to the real world difficult. A possible improvement might be to find a mapping from histograms of synthetic images to histograms of real images, by mapping “synthetic textons” to “real-world textons.”

The presented synthetic data generation software generates image patches by drawing samples from parametric distributions only. This was motivated by the fact that an ideal map should be independent of previous estimates and based on single images—ideally requiring no filtering. In the future, the synthetic flights could be performed by simulating a continuous route above the image. This would allow testing the ability of the particle filter on synthetic flights.

The shift of the processing power could be further amplified by using a different regression technique. While the k -NN algorithm showed one of the best performances in our tests, larger training data sets are penalized due to a greater prediction time. However, the

¹The prototype uses an AR.Drone 2.0 and the software rospy:
<https://github.com/Pold87/derotator>

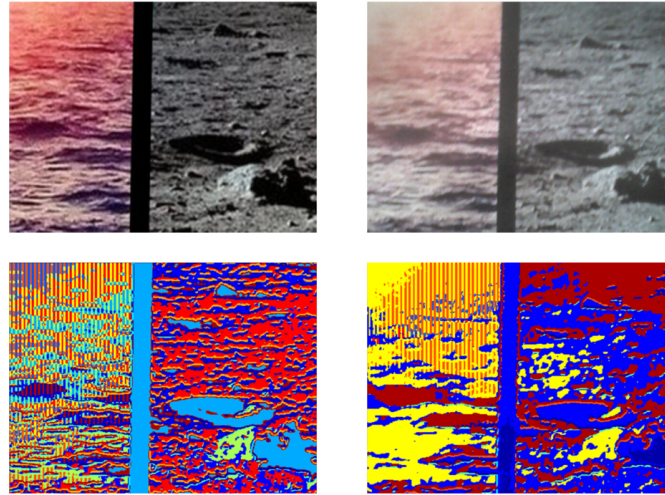


Fig. 5.1: Exemplifying the reality gap. *Top left:* image patch generated using the synthetic data generation tool. *Top right:* image patch taken with the MAV's camera after printing the patch. *Below left:* Texton image of the synthetic image. In the image, the colors indicate the closest texton (dictionary size: 20 textons) at the respective position. *Below right:* Texton image of the real image.

choice of a different regression technique is not straightforward: it should be able to output multiple predictions, since certain map regions might be ambiguous.

This thesis presented a novel approach for lightweight indoor localization of MAVs. We pursued an on-board design to foster real-world use. The conducted experiments underline the applicability of the system. Promising results were obtained for real-time position estimates and accurate landing in the indoor environment.

The approach is based on three pillars that we identified for indoor localization for MAVs. The first pillar shifts computational effort from the flight phase to a preprocessing step. This provides the advantages of sophisticated algorithms, without affecting performance during flight. The second pillar states that on-board algorithms should be able to trade off speed with accuracy. This capacitates their use on a wide range of models. Examples of these adaptable algorithms are the random sampling steps in the texton-based approach and the particle filter. The third pillar is a known—and possibly—modifiable environment. This knowledge and flexibility allows for predicting and improving the quality of the approach.

The developed algorithms set the stage for global localization in various GPS-denied environments, such as homes, offices, or factory buildings. While the used platform for this project was the Parrot Bebop Drone, the characteristics of the developed system generalize to smaller MAVs in a flexible and innovative way. We hope that our indoor localization approach will pave the way for various applications, including delivery, search and rescue, or surveillance to support human operators in everyday life.

Bibliography

- [1]Markus Achtelik, Michael Achtelik, Stephan Weiss, and Roland Siegwart. „Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments“. *IEEE International Conference on Robotics and automation (ICRA)*. IEEE. 2011, pp. 3056–3063 (cit. on p. 8).
- [2]Spencer Ahrens, Daniel Levine, Gregory Andrews, and Jonathan P How. „Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments“. *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2009, pp. 2643–2648 (cit. on p. 1).
- [3]Eman R. AlBasiouny, Amany Sarhan, and T. Medhat. „Mean-shift-FAST algorithm to handle motion-blur with tracking fiducial markers“. *Tenth International Conference on Computer Engineering & Systems (ICCES)*. IEEE. 2015, pp. 286–292 (cit. on p. 6).
- [4]Adrien Angeli, David Filliat, Stéphane Doncieux, and Jean-Arcady Meyer. „2D Simultaneous localization and mapping for micro air vehicles“. *European Micro Aerial Vehicles (EMAV)*. 2006 (cit. on p. 1).
- [5]Abraham Galton Bachrach. „Autonomous flight in unstructured and unknown indoor environments“. PhD thesis. Massachusetts Institute of Technology, 2009 (cit. on p. 1).
- [6]Nitin Bhatia et al. „Survey of nearest neighbor techniques“ (2010). arXiv: 1007.0085 [cs.CV] (cit. on p. 17).
- [7]Michael Blösch, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. „Vision based MAV navigation in unknown and unstructured environments“. *2010 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 21–28 (cit. on pp. 1, 7, 13).
- [8]Pascal Brisset, Antoine Drouin, Michel Gorraz, Pierre-Selim Huard, and Jeremy Tyler. „The Paparazzi solution“. *2nd US-European Competition and Workshop on Micro Air Vehicles 2006*. 2006 (cit. on pp. 4, 10, 11).
- [9]Haiyang Chao, Yu Gu, and Marcello Napolitano. „A survey of optical flow techniques for UAV navigation applications“. *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2013, pp. 710–716 (cit. on p. 5).
- [10]Hung-Kuo Chu, Chia-Sheng Chang, Ruen-Rone Lee, and Niloy J. Mitra. „Halftone QR codes“. *ACM Transactions on Graphics (TOG)* 32.6 (2013), p. 217 (cit. on p. 6).
- [11]Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. „Torch7: A Matlab-like environment for machine learning“. *BigLearn, NIPS Workshop*. EPFL-CONF-192376. 2011 (cit. on p. 8).
- [12]CPSC 340: Machine Learning and Data Mining. URL: <https://www.cs.ubc.ca/~schmidtm/Courses/340-F15/L7.pdf> (visited on Aug. 10, 2016) (cit. on p. 17).

- [13]Pablo d'Angelo et al. *Hugin - Panorama photo stitcher*. URL: <http://hugin.sourceforge.net/> (visited on Aug. 10, 2016) (cit. on p. 13).
- [14]G.C.H.E. De Croon, K.M.E. De Clercq, Ruijsink, Remes, and C. De Wagter. „Design, aerodynamics, and vision-based control of the DelFly“. *International Journal of Micro Air Vehicles* 1.2 (2009), pp. 71–97 (cit. on p. 8).
- [15]G.C.H.E. De Croon, C. De Wagter, B.D.W. Remes, and R. Ruijsink. „Sub-sampling: Real-time vision for Micro Air Vehicles“. *Robotics and Autonomous Systems* 60.2 (2012), pp. 167–181 (cit. on pp. 8, 17).
- [16]G.C.H.E. De Croon, E. De Weerd, C. De Wagter, B.D.W. Remes, and R. Ruijsink. „The appearance variation cue for obstacle avoidance“. *IEEE Transactions on Robotics* 28.2 (2012), pp. 529–534 (cit. on p. 9).
- [17]Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. „Monte carlo localization for mobile robots“. *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2. IEEE. 1999, pp. 1322–1328 (cit. on p. 17).
- [18]Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. „Discriminative unsupervised feature learning with convolutional neural networks“. *Advances in Neural Information Processing Systems*. 2014, pp. 766–774 (cit. on p. 8).
- [19]Hans Driessen and Yvo Boers. „MAP estimation in particle filter tracking“. *2008 IET Seminar on Target Tracking and Data Fusion: Algorithms and Applications*. IET. 2008, pp. 41–45 (cit. on pp. 20, 21).
- [20]Daniel Eberli, Davide Scaramuzza, Stephan Weiss, and Roland Siegwart. „Vision based position control for MAVs using one single circular landmark“. *Journal of Intelligent & Robotic Systems* 61.1-4 (2011), pp. 495–512 (cit. on p. 6).
- [21]Sean P. Engelson and Drew V. McDermott. „Error correction in mobile robot map learning“. *IEEE International Conference on Robotics and Automation, 1992 Proceedings*. IEEE. 1992, pp. 2555–2560 (cit. on p. 5).
- [22]Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. „Monte carlo localization: Efficient position estimation for mobile robots“. *AAAI/IAAI* 1999 (1999), pp. 343–349 (cit. on p. 5).
- [23]S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. „Automatic generation and detection of highly reliable fiducial markers under occlusion“. *Pattern Recognition* 47.6 (2014), pp. 2280–2292 (cit. on p. 6).
- [24]S. Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. „Automatic generation and detection of highly reliable fiducial markers under occlusion“. *Pattern Recognition* 47.6 (2014), pp. 2280–2292 (cit. on p. 6).
- [25]Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. „Towards a navigation system for autonomous indoor flying“. *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2009, pp. 2878–2883 (cit. on p. 1).
- [26]Isabelle Guyon and André Elisseeff. „An introduction to feature extraction“. *Feature extraction*. Springer, 2006, pp. 1–25 (cit. on p. 16).
- [27]Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003 (cit. on p. 24).

- [28]Eva Hornecker and Thomas Psik. „Using ARToolKit markers to build tangible prototypes and simulate other technologies“. *IFIP Conference on Human-Computer Interaction*. Springer. 2005, pp. 30–42 (cit. on p. 6).
- [29]Michael Jepson. *Rotation in 3D using OpenCV's warpPerspective*. URL: <http://jepsonsblog.blogspot.nl/2012/11/rotation-in-3d-using-opencvs.html> (visited on Dec. 10, 2015) (cit. on p. 24).
- [30]Hirokazu Kato and Mark Billinghurst. „Marker tracking and HMD calibration for a video-based augmented reality conferencing system“. *2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99), 1999 Proceedings*. IEEE. 1999, pp. 85–94 (cit. on p. 6).
- [31]Alex Kendall, Matthew Grimes, and Roberto Cipolla. „PoseNet: A convolutional network for real-time 6-DOF camera relocalization“. *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2938–2946 (cit. on pp. 7, 8).
- [32]Teuvo Kohonen. „The self-organizing map“. *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480 (cit. on p. 15).
- [33]Mirosław Kordos, Marcin Blachnik, and Dawid Strzempa. „Do we need whatever more than k-NN?“. *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2010, pp. 414–421 (cit. on p. 17).
- [34]Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. „Gradient-based learning applied to document recognition“. *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 8).
- [35]David G. Lowe. „Object recognition from local scale-invariant features“. *The Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. IEEE. 1999, pp. 1150–1157 (cit. on pp. 6, 13).
- [36]Kimberly McGuire, Guido de Croon, Christophe de Wagter, et al. „Local Histogram Matching for Efficient Optical Flow Computation Applied to Velocity Estimation on Pocket Drones“ (2016). arXiv: 1603.07644 [cs.R0] (cit. on p. 5).
- [37]*Microsoft Image Composite Editor*. URL: <http://research.microsoft.com/en-us/um/redmond/projects/ice/> (visited on Aug. 2, 2016) (cit. on p. 13).
- [38]*OptiTrack - Motion Capture Systems*. URL: <http://optitrack.com/> (visited on Aug. 3, 2016) (cit. on p. 27).
- [39]Parrot. *CES 2015 : Parrot Bebop Dance Choreography*. 2015. URL: https://www.youtube.com/watch?v=A_3UifFb45Y (visited on Oct. 15, 2015) (cit. on p. 6).
- [40]*Parrot Bebop Drone*. URL: <http://www.parrot.com/products/bebop-drone/> (visited on July 14, 2016) (cit. on p. 11).
- [41]Prov0lone. *The horizons of Venus, Earth, Moon, Mars, and Titan*. URL: <http://imgur.com/gallery/A1gIYXd> (visited on July 10, 2016) (cit. on p. 36).
- [42]Franck Ruffier, Stéphane Viollet, S. Amic, and N. Franceschini. „Bio-inspired optical flow circuits for the visual guidance of micro air vehicles“. *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*. Vol. 3. IEEE. 2003, pp. III–846 (cit. on p. 5).
- [43]Stephen Se, David Lowe, and Jim Little. „Global localization using distinctive visual features“. *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 1. IEEE. 2002, pp. 226–231 (cit. on p. 6).

- [44]*Sparkle Wallpaper*. URL: <http://e2ua.com/group/sparkle-wallpaper/> (visited on July 10, 2016) (cit. on p. 36).
- [45]S. Thrun. *Artificial intelligence for robotics*. URL: <https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373> (visited on Mar. 10, 2016) (cit. on p. 20).
- [46]Manik Varma and Andrew Zisserman. „A statistical approach to texture classification from single images“. *International Journal of Computer Vision* 62.1-2 (2005), pp. 61–81 (cit. on pp. 1, 8).
- [47]Manik Varma and Andrew Zisserman. „Texture classification: Are filter banks necessary?“ *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003 Proceedings*. Vol. 2. IEEE. 2003, pp. II–691 (cit. on p. 15).
- [48]*Wiki PaparazziUAV*. URL: https://wiki.paparazziuav.org/wiki/Main_Page (visited on July 14, 2016) (cit. on p. 12).

List of Figures

1.1	High-level perspective of the developed system.	2
2.1	Examples of fiducial markers.	6
2.2	Perspective transformation between keypoints	7
3.1	Modified Parrot AR.Drone 2.0	11
3.2	Ground control station of Paparazzi	12
3.3	Training dataset generation with the motion tracking system	13
3.4	Orthomap of a texture-rich floor	14
3.5	Training dataset generation with the homography-based approach	15
3.6	Texton dictionary	16
3.7	Time steps of a Kalman filter.	18
3.8	Time steps of a general Bayesian Filter.	19
3.9	Calculation of histogram losses.	22
3.10	Global loss of a map.	23
3.11	Example image patches of a synthetic flight	24
3.12	Camera model of the synthetic flight.	24
4.1	The CyberZoo.	28
4.2	Mean cosine similarity between histograms.	28
4.3	Error as a function of k	29
4.4	Histogram distance and measurement error.	29
4.5	Hyperspatial image of the map.	31
4.6	Error as a function of particles.	33
4.7	Error as a function of samples.	34
4.8	Frequency as a function of the number of histograms.	35
4.9	Image with the highest and the lowest loss value.	36
5.1	Exemplifying the reality gap.	40

List of Tables

4.1	Estimates of the texton-based approach	30
4.2	Error statistics for the homography method.	31
4.3	Estimates of the texton-based approach	31
4.4	Map evaluation procedure on synthetic data.	35

Declaration

I declare that I have developed and written the enclosed Master's thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master's thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Nijmegen, August 24, 2016

Volker Strobel

