Machine Learning-based Indoor Localization for Micro Aerial Vehicles

Volker Strobel

August 18, 2016

Abstract

Widespread applications, ranging from surveillance to search and rescue operations, make Micro Air Vehicles (MAVs) versatile platforms. 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 creates synthetic images that could be taken during an actual flight.

We conducted several flight tests to evaluate the performance of the approach. A comparison of the localization technique with the ground truth showed promising results. In a triggered landing setting, the MAV correctly landed in 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 can build a foundation for efficient localization in various GPS-denied environments.

Contents

Chapter 1

Introduction

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 system is usually not available in confined spaces and would not provide sufficiently accurate estimates in cluttered environments. If sufficient computational and physical power is available, a typical approach to estimate a UAV's position is by using 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]. Additionally, many commercially available drones are already equipped with cameras. In contrast to other existing approaches, it does not rely on additional information, such as data from the inertial measurement unit (IMU). The only required tool is a camera, which minimizes possible points of failure. Cameras are not only lightweight but also robust to external influences like magnetic fields. This minimizes points of failure. However, this reduced physical payload is not without cost:

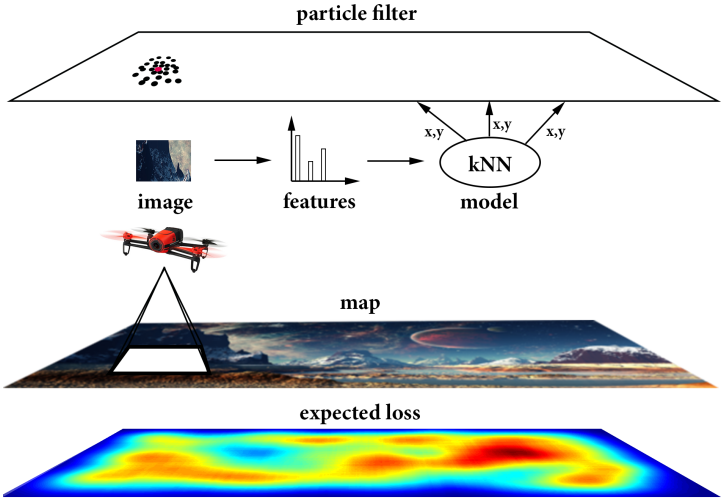

Figure 1.1: The figure illustrates the developed system from a high-level perspective. A feature vector—the texton histogram—is extracted from the current camera image of the UAV. The feature vector is forwarded to a machine learning model that uses a k-Nearest Neighbors algorithm to output x, y-position 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.

it 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-intense 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 to function in changing environments, 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 the position, which circumvents the requirement to store a map in the UAV'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 [43]. New images can then also be encoded by 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 solve positional ambiguities. The computational effort of the approach can be adjusted by modifying the amount of extracted patches and used particles, resulting in a trade-off between accuracy and execution frequency. Figure 1.1 summarizes the algorithm.

In the presented approach, computational power will be 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 be even actively modified. The environment is non-dynamic and planar, therefore, the UAV 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 will be 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 fur-

ther 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 filter (ii) software for augmenting an image with synthetic views, (iii) a script for evaluating a map based on histograms and corresponding x, y-positions.

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. The results are then discussed in Chapter 5. Finally, we draw our conclusions and indicate future research directions in Chapter 6.

Chapter 2

Related Work

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.

One distinguishes two types of robot localization: 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 lost track, the position can typically not be recovered. 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 Fiducial Markers

Fiducial markers (Figure 2.1), which are often employed in augmented reality applications [30, 23], have been used for UAV localization and landing [20, 38]. 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 [24].

An advantage of fiducial markers is their widespread use, leading to technically mature open-source libraries, including ArUco [24] 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 downside is that markers might be considered as visually unpleasant and may not fit into a product or environmental design [10]. They offer little flexibility, since 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].

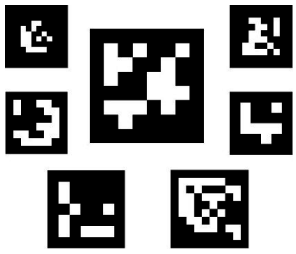

Figure 2.1: Examples of fiducial markers of the ArUco library.

2.1.2 Homography Determination & Keypoint Matching

A standard approach for estimating camera pose is detecting and describing keypoints of the current view and a reference image [41], 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 varying 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 a 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 allow 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 of 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 com-

plex [31]. Therefore, ground stations for off-board processing or larger processors are usually needed for flight control. In this thesis, the homography-based approach is used in a pre-flight phase to assign x, y-coordinates to images. The approach has been employed for global localization for UAVs:

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

Blösch et al. [7] evaluate it on a $3.5\,m \times 2\,m$ 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.

2.1.3 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, which represent increasing levels of abstraction [34]. While their training is usually time-consuming, predictions with CNNs often takes 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 achieves an accuracy of ap-

proximately 50 cm in indoor environments with a spatial extent between $2 \times 0.5 \times 1\,m^3$ and $4 \times 3 \times 1.5\,m^3$. The approach 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. However, 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.

2.1.4 Optical Flow

Optical flow algorithms are biologically inspired methods for navigation—taking inspiration from insects and birds [40]. 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 histogram to calculate the flow.

2.2 Texton-based Methods

Textons are small characteristic image patches; their frequency in an image can be used as image feature vector. Varma et al. [43] originally introduced textons for classifying different textures, showing that they outperform computationally more complex algorithms, like Gabor filters [43]. For the classification, the approach compares texton histograms between a training set and the test sample and the class of the closest training sample is assigned to the test sample. A texton histogram is obtained by extracting 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. A disadvantage is that it discards all information about the spatial arrangement of texton, 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 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 $5m \times 5m$ office space.

In the scope of this thesis, an efficient *global* localization was developed that draws upon the lightweight character of texton-based approaches and combines their flexibility with the advantages of homography-based approaches.

Chapter 3

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 about the parts of the framework and the pillars will be given in separate sections.

Algorithm 1 High-level texton framework

1: $t \leftarrow 0$
2: $\mathcal{X}_0 \leftarrow$ INIT_PARTICLES
3: **while** true **do**
4: $\quad t \leftarrow t + 1$
5: $\quad I_t \leftarrow$ RECEIVE_IMG_FROM_CAMERA
6: $\quad \mathcal{H}_t \leftarrow$ GET_TEXTON_HISTOGRAM(I_t)
7: $\quad \mathbf{z}_t \leftarrow k$-NN$(\mathcal{H}_t)$
8: $\quad \mathcal{X}_t \leftarrow$ PARTICLE_FILTER$(\mathcal{X}_{t-1}, \mathbf{z}_t)$
9: $\quad x_t, y_t \leftarrow$ MAXIMUM_A_POSTERIORI_ESTIMATE$(\mathcal{X}_t)$
10: **end**

3.1 Hardware and Software

In our first approach, the commercially available Parrot AR.Drone2.0 was equipped with an Odroid XU-4 single board computer, a Logitech 525 HD webcam, and WiFi module. Figure 3.1 shows the setup. Instead of employing the AR.Drone2.0 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 through the modifications of the system resulted in unstable flight performance. Therefore, we modified the system to execute the localization algorithm directly on-board of the MAV and not on the additional Odroid processor. To this end, the software had to be ported from the high-level language Python to the low-level language C without relying on many existing software libraries. However, this step removed the need for the additional payload and made the flight performance very 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.

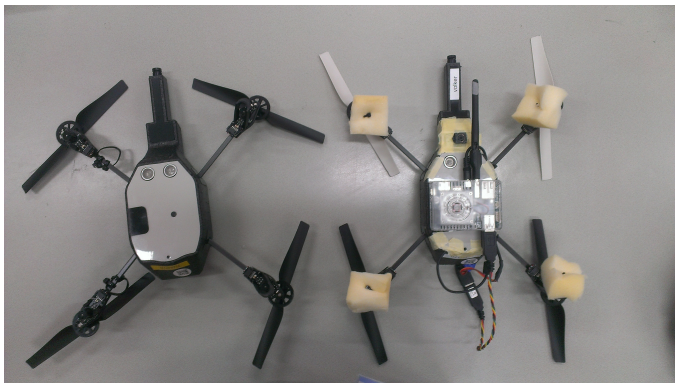

Figure 3.1: Comparison of an unmodified Parrot AR.Drone.2.0 (left) and a modified version (right). The modified one was equipped with an Odroid XU-4 single board computer, a Logitech C525 HD camera, a WiFi module, and a USB connection between the Odroid board and the AR.Drone.2.0 flight controller.

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 flying time. The UAV's dimensions are $28 \times 32 \times 3.6\,\text{cm}$ and it weighs $400\,\text{g}$. 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 [39].

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 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, for example information about speed, altitude, position, or battery status.

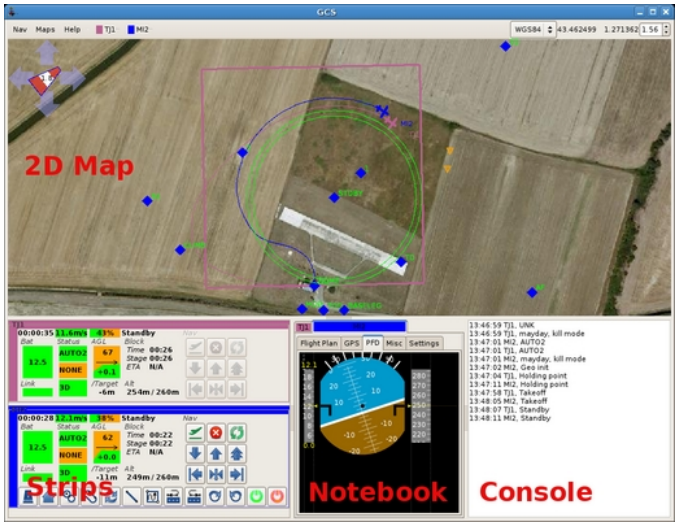

Figure 3.2: The ground control station of the Paparazzi software. It displays information about the status of the UAV and provides functions for controlling the vehicle (from PaparazziUAV wiki [45]).

3.2 Preliminary Dataset Generation

A main idea 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 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 within an error of 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.

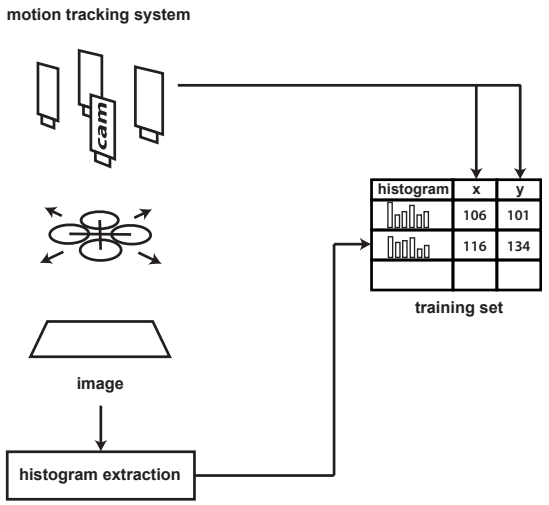

Figure 3.3: Training dataset generation if the motion tracking system is used. The texton histograms of the camera images during flight are extracted and aligned with the highly accurate position estimates of the motion tracking system. The result is a high-quality training set of texton histograms and 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 2.1.2) 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 3.5). 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 better quality results.

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]. 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 on 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 $C_x = \frac{width(W)}{width(I)}$ and $C_y = \frac{height(W)}{height(I)}$, 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 3.4.

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 starts with an existing image and modifies 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 2.1.2) that depends on the quality of the keypoint matches.

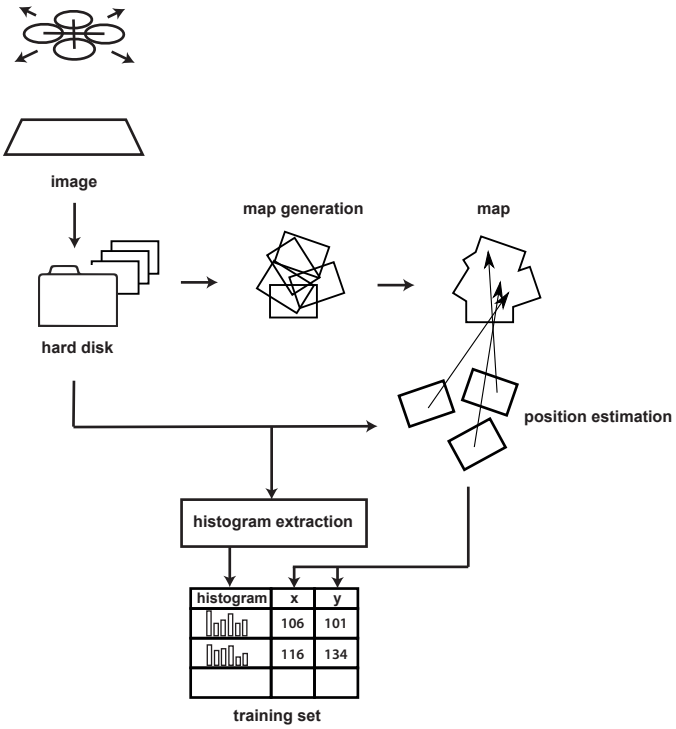

Figure 3.4: The figure illustrates the training set generation when applying the homography-based approach. Images from an initial flight are stitched together to create an orthomap. The same images are used to detect and describe their keypoints using SIFT, followed by finding a homography between the 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.

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 the histograms, the k-Nearest Neighbors (k-NN) algorithm, and the particle filter. The dictionary of textons constitutes the base for determining the texton histograms. These histograms are used as features in the k-NN algorithm. The algorithm outputs k possible x, y-

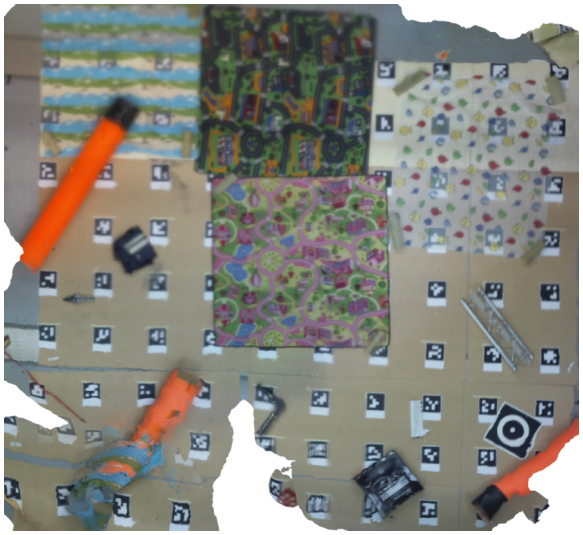

Figure 3.5: This figure shows the created orthomap of a texture-rich floor. It is stitched together using 100 single images and represents a real world area of approximately 8×8 meters. Image distortions, non-mapped areas, and slightly skewed seams at several points are visible.

coordinates for a given image, which are forwarded to the particle filter to yield a final position estimation.

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 [44]. 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 $\alpha = 0.02$:

$$d_r := d_r + \alpha(x - d_r) \tag{3.1}$$

The first 100 images of each dataset were used to generate the dictionary. From each image, 1000 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.

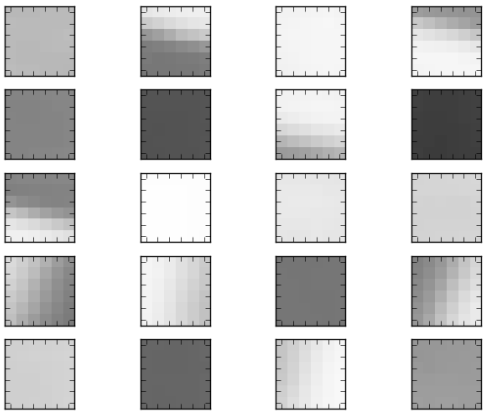

Figure 3.6: The figures shows a dictionary consisting of 20 grayscale textons ($w \times h = 6 \times 6$ pixels).

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. One can see that the random sampling step introduces random effects into the approach. Therefore, for generating 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 algorithm. 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 rather 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.

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 complex. 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 $p(X_t \mid Z_t)$, where X_t is the state vector at time t and $Z_t = \mathbf{z}_1, ..., \mathbf{z}_t$ are all outputs of the k-NN

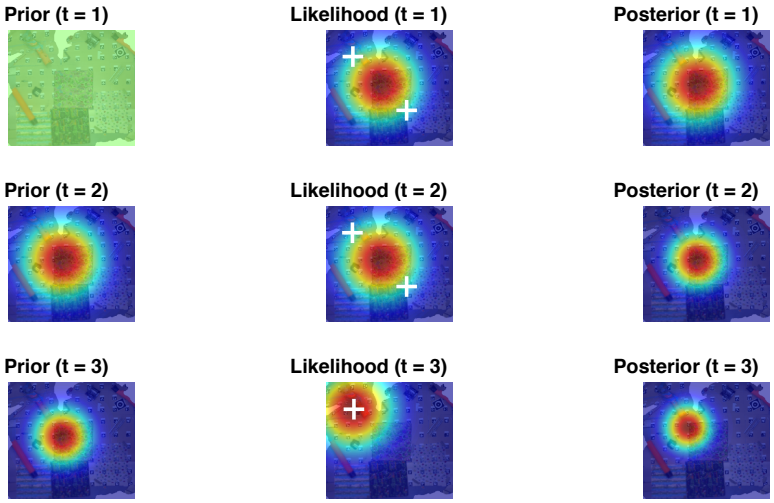

Figure 3.7: The figure illustrates the measurement model of a Kalman filter. The colors represent the probability of an x, y-position (red: high probability; blue: low probability). In timestep $t = 1$, the filter is initialized with an uniformed prior and each position has equal probability. To incorporate 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.

algorithm up to time t, with each $\mathbf{z}_i$ representing the k x, y-outputs of the algorithm at time i.

The used particle filter is initialized with $M = 100$ 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 parameters of the GMM are obtained by comparing the ground truth positions (random variable T) from the motion tracking system to the estimates from the k-NN algorithm (random variable P_j) in a recorded dataset. The GMM is parameterized by the variances $\Sigma^{[j]}, j \in \{1, \ldots, k\}$ that are dependent on the rank j of the

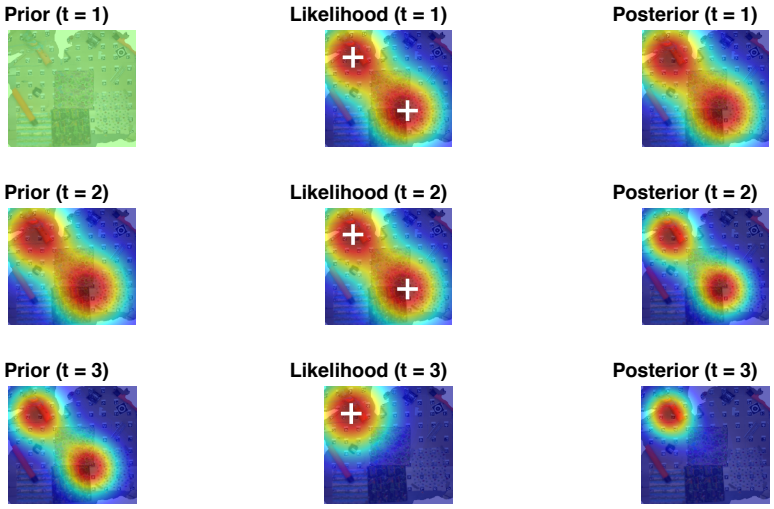

Figure 3.8: Three time steps of a Bayesian filter. The colors represent the probability of an x, y-position (red: high probability; blue: low probability). In contrast to the Kalman filter, the likelihood (measurement model) is calculated using a *mixture* of Gaussian distributions centered around the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

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 and the predictions P_j of the k-NN algorithm: $\Sigma^{[j]} := \mathrm{Var}(T - P_j)$.

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 ith neighbor from the kNN prediction, $x_t^{[m]}$ the mth particle at time t, and $w_t^{[m]}$ its corresponding weight.

Algorithm 2 Particle filter update

procedure PARTICLE_FILTER($\mathcal{X}_{t-1}, z_t$)
    ▷ Initialize particle list
    $\mathcal{X}_{\text{temp}} := \varnothing$
    for $m = 1$ to M **do**
        ▷ Add random process noise (motion model)
        $x_t^{[m]} \leftarrow x_t^{[m]} + \mathcal{N}(0, \Sigma_{\text{process}})$
        ▷ Iterate over predictions from k-NN (measurement model)
        $w \leftarrow 0$
        for $i = 1$ to k **do**
            ▷ Gaussian Mixture Model
            $w \leftarrow w + f(z_t^{[i]}; x_t^{[m]}, \Sigma_{\text{measurement}}^{[i]})$
        $\mathcal{X}_{\text{temp}} := \mathcal{X}_{\text{temp}} \cup (x_t^{[m]}, w)$
    ▷ Importance resampling
    $\mathcal{X}_t \leftarrow$ RESAMPLING_WHEEL($\mathcal{X}_{\text{temp}}$)
    return $\mathcal{X}_t$

The "resampling wheel" [42] (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.

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 et al. [19]. This approach is a discrete approximation of the true MAP [19]. It uses

Algorithm 3 Resampling wheel

```
1: procedure RESAMPLING_WHEEL(X_temp)
2:     ▷ Initialize particle list
3:     X_t ← ∅
4:     ▷ Sample random index from the number of particles
5:     sample i ~ M · U(0, 1)
6:     β ← 0
7:     for m = 1 to M do
8:         β ← β + U(0, 1) · 2 · max(w_t)
9:         while β > w_t^[i] do
10:            β ← β − w_t^[i]
11:            i ← (i + 1) mod M
12:        X_t ← X_t ∪ X_temp^[i]
13:    return X_t
```

the following formula to obtain the MAP estimate X_t^{MAP}—the "final" x, y-position:

$$X_t^{\text{MAP}} = \underset{x_t^{[i]} \in \{i=1,\ldots,M\}}{\arg\max} \sum_{j=1}^{M} f(x_t^{[i]}; x_{t-1}^{[j]}, \Sigma_{\text{process}}) w_{t-1}^{[j]} \tag{3.2}$$

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, 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 Pillar III: 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 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 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. 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.

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

$$CS(h_i, h_j) = \frac{h_i^T h_j}{||h_i||\,||h_j||} \tag{3.3}$$

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:

$$f(x; \mu, \sigma) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

Figure 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)

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:

$$d_e(\text{pos}_i, \text{pos}_j; \Sigma) = f(x_i; x_j, \sigma_x) \cdot f(y_i; y_j, \sigma_y) \tag{3.5}$$

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 ($\mathcal{D}$):

$$L(\mathcal{D}) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} CS(h_i, h_j) - f(x_i; x_j, \sigma_x) \cdot f(y_i; y_j, \sigma_y) \tag{3.6}$$

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.

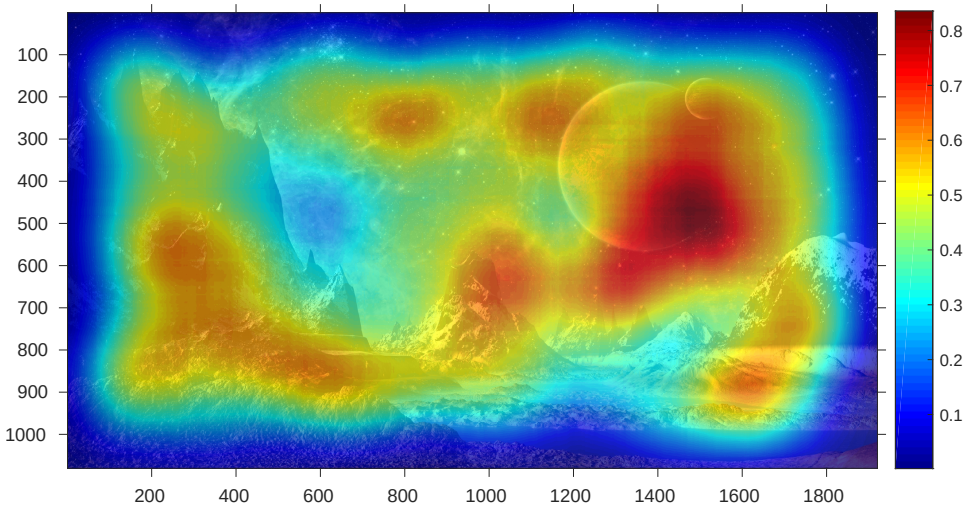

Figure 3.10: The figure shows the loss of a map: the regions that did not follow the ideal similarity pattern are displayed in red. For the visualization, the loss values per sample in the dataset were smoothed with a Gaussian filter. This assigns a loss value to each x, y-position of the map. The synthetic 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

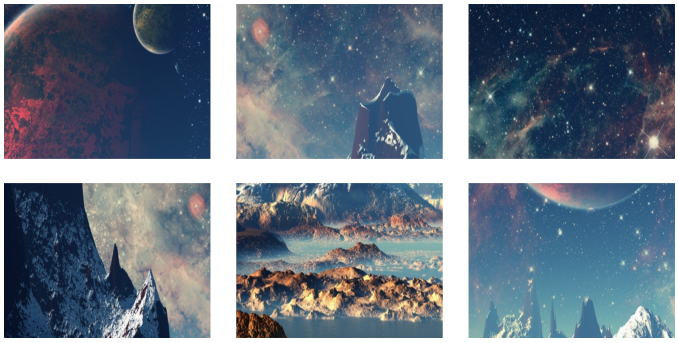

Figure 3.11: Six image patches generated by means of the synthetic data generation tool.

- brightness: addition of constant value b to all pixels
- 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.

Figure 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 et al. [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:

$$\tilde{T} = \begin{bmatrix} 1 & 0 & 0 & \mathrm{d}x \\ 0 & 1 & 0 & \mathrm{d}y \\ 0 & 0 & 1 & \mathrm{d}z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$H = T \cdot R \tag{3.7}$$

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:

$$H' = (T \cdot R)^{-1} = R^{-1} \cdot T^{-1} \tag{3.8}$$

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

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -\mathrm{d}x \\ 0 & 1 & 0 & -\mathrm{d}y \\ 0 & 0 & 1 & -\mathrm{d}z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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 $\mathrm{d}z$ 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:

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

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

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

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.

Chapter 4

Analysis

In this chapter, the setup of the conducted experiments is presented. We examine different parameter choices in Experiment 1 to 6 in on-ground experiments using recorded data. Afterward, the found parameters are used to show the validity during flight in Experiments 7 and 8. The experiments were conducted in TU Delft's indoor flight arena, the "CyberZoo." 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.

4.1 Analysis – Determining the Number of Image Patches

The developed framework modifies it computational complexity by changing the number of extracted image samples. To increase the speed of the algorithm, the goal is to use as few samples as possible. To determine a suitable number of extracted samples, in this experiment, the average cosine similarity between $D = 20$ datasets of histograms is compared. Each dataset consists of $N = 10300$ histograms. The independent variable is the number of extracted image patches M. The histograms were generated using the same images. Due to the random sampling of the extracted image patches, the histograms of each datasets will differ. This deviation will be measured using the cosine similarity. Therefore, each of the D datasets was

compared to all the other $D - 10$ datasets and the average cosine similarity was determined as well as the standard deviation of the cosine similarity was measured. Comparing the cosine similarity between the histograms has the advantage that the number of samples can be determined independent of a specific task.

Figure 4.1 displays the cosine similarity of histograms as a function of the number of samples and the corresponding standard deviations.

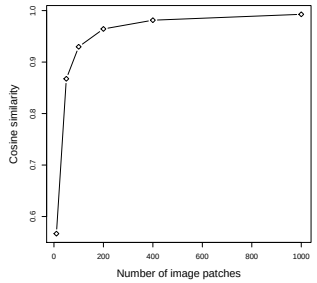

Figure 4.1: Cosine similarity of histograms as a function of the number of samples *Right*: Standard deviation of cosine similarity in relation to the number of samples. The squares indicate the positions at which the dependency was evaluated.

4.2 Analysis – Setting the Baseline for k-NN and determining k

In a standard setting, the training error ϵ_t of a k=1-nearest neighbor algorithm is $\epsilon_t = 0$ because the nearest neighbor of the sample will be the sample itself, given that each feature vector is unique. However, in the presented framework, the random sampling in the histogram extraction step leads to varying histograms.

	x-position	y-position
Error in *cm*	31	75
STD in *cm*	73	369

Table 4.1: Error statistics for the homography method.

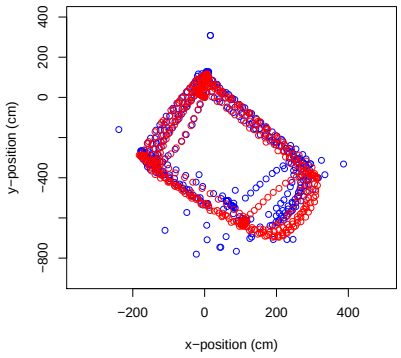

Figure 4.2: The estimates of the homography method compared to the ground truth of the motion tracking system. TODO: Legend

4.3 Experiment – Motion tracking system for Stabilization, Texton-based Approach for estimation

4.3.1 Training Set based on Motion Tracking System

In this experiment, a fixed route was set using the ground control station. While the stabilization and guidance were performed using the motion tracking system, the position estimates were performed on board of the MAV using the texton-based approach. The Euclidean distances between the estimates of the motion tracking system and the texton-based approach were measured separately for the x- and y-direction. The training dataset was composed of 500 images recorded at an height of approximately 1 m, recorded in a time span of one hour before the experiment. The corresponding x, y-coordinates were obtained from the motion tracking system.

	x-position	y-position
Error in cm	46	54
STD in cm	56	71

Table 4.2: Estimates of the texton-based approach

4.3.2 Training Set based on Homography-finding Method

In this experiment, the training dataset was created using the homography-finding method. Apart from that, the settings are the same as in Experiment 4.3.1.

Figure 4.3: The created map that was stitched together using 445 images. A non-mapped area in the middle of the map can be seen, which is a result of the set flight path. An image distortion can be seen at the right-hand side, where the landing spot sign appears twice, while in reality, only one circle was visible.

4.4 Experiment – Triggered Landing

In the triggered landing experiment, the UAV was programmed to land as soon as its position estimates are in the landing zone. The landing zone was defined as a circle with a radius of 60 cm. The x, y-coordinate of the center of the circle was specified in the flight plan. A safety criterion based on the variance of particles was introduced, such that the landing is only performed if the criterion holds. The criterion parameter was set to 60 cm in both $x-$ and y-direction. The experiment was conducted on a $5m \times 5m$ map and the ground truth was based on the position data from the motion tracking system. The training dataset was composed of 500 images recorded at an height of approximately 1 m, recorded in a time span of one hour before the experiment. Table 4.3 shows the results of the triggered landings. Four

out of six landings were correctly performed in the landing area. The mean distance of outliers was 16 cm.

	x-position	y-position
Error in *cm*	TODO	TODO
STD in *cm*	TODO	TODO

Table 4.3: Results of the triggered landings

4.5 Experiment – Determining the Frequency

The frequency of the proposed algorithm is determined by varying the number of samples in the texton-based approach, the number of textons, and the number of particles of the particle filter.

4.6 Experiment – Comparing different possible maps

For the map comparison, 50 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 likely to be visually pleasant since they are often used as desktop backgrounds. The criteria for the images were that their minimum resolution was 1920×1080. Images with a higher resolution were converted to 1920×1080. For each image, we generated 1000 random image patches using the simulation method described in Section 3.4.2, followed by the histogram extraction method. 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.

Figure 4.4 shows the map with the highest global loss value and the map with the lowest global loss value.

Statistic	Value
mean	0.57
median	0.55
standard deviation	0.14
max	0.99
min	0.24

Table 4.4: Results of the map evaluation procedure on synthetic data

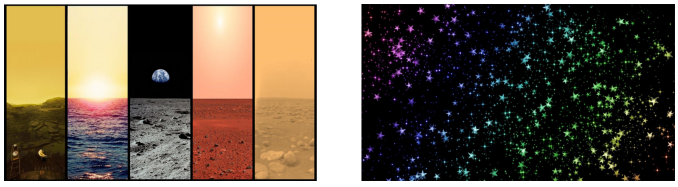

Figure 4.4: Image with the lowest loss value; *Right*: Image with the highest loss value

Chapter 5

Discussion

In this chapter, the results of the experiments are discussed with regards to accuracy, frequency, and future improvements. Afterward, in the General Discussion, the research questions are addressed and discussed.

The comparison between sub-sampling and full sampling (Experiment 4.1) has shown that only a small part of the maximum amount of samples is necessary. In fact, $\frac{400}{640 \times 480} = 0.13\,\%$ of the maximum amount of samples suffices to achieve cosine similarities between the texton histograms larger than 99 %. This set the stage for large speed-ups during live operation. Additionally, the technique allows for further speed-ups depending on the processing power of the platform at hand.

In the real-time position estimation experiment (Experiment 4.3.1), the initial mean error was rather large: 130 cm in x-direction and 90 cm in y-direction. A more in-depth analysis revealed that the estimates of the particle filter were lacking behind the position estimates of the motion tracking system. This was due to the simple motion model of the particle filter that was based on Gaussian noise only. By shifting the estimates of the particle filter by six frames, that is approx. 0.46 seconds at a frequency of 13 Hz, the error could be reduced to 46 cm in x-direction and 54 cm in y-direction. This lag can be addressed by to strategies: (i) slower speed during flight, (ii) a better or more flexible motion model.

The triggered landing (Experiment 4.4) showed a high 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. A comparison without the safety criterion, showed, that the criterion can have advantages.

The evaluation of different maps using the synthetic data showed high differences between the evaluated images. The range of losses from 0.24 to 0.99 underlines the varying suitability of different maps for the proposed algorithm. The image with the minimum and the one with the maximum loss value based on their color histogram are shown in Figure 4.4. The different patterns of the images are clearly visible: while the image with the minimum value 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 the place and leading to high loss values. This initial evidence can be taken to test the predictive power of the evaluation algorithm for texton histograms.

5.1 General Discussion

In this chapter, the results will be discussed regarding error statistics, execution frequency, robustness, and scalability. To begin with, we recapitulate the research questions:

- R1: Can accurate 2D positions be estimated in real-time, using a machine learning-based approach on a limited processor in a modifiable indoor environment?
- R2: Is accurate real-world localization regression or classification possible when the training data comprises synthetic data only?
- R3: Can we predict the goodness of a given map for the proposed localization approach?

Regarding R1, the conducted experiments provide supportive evidence that a texton-based machine learning approach is able to accomplish real-time indoor localization. The proposed algorithm runs with as frequency of 30 Hz on a single board computer with limited CPU. Shifting processing power to an offline training step and relying on random sampling are the cornerstones for running the algorithm on processors with limited CPU. Despite the small ratio between extracted image patches (s) and the maximum amount of different image matches ($s*$), the accuracy is hardly affected, and only slightly improves when incorporating more textons.

Regarding research question R2, the initial idea—to use the synthetically generated images directly as training data—was not successful and not further followed up. This might be also the reason that only few projects have used synthetic images for real-world phenomena. The reality gap between the synthetic data and real-world data was huger than expected. 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 too strong extent. 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'.

Referring to R3, we found some initial evidence that the proposed map evaluation generalizes to the real-world. In contrast to R2, the generalization from the synthetic data to real-world data is of a different nature in this case. 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 not have similar histograms using the synthetic images nor the real-world images.

Despite the overall promising results, we noticed drawbacks of the proposed approach during the flight tests and directions for future research.

The accuracy—that is the difference between the estimates of the motion tracking system and the texton-based approach— could be further improved by incorporating more features, for example histogram of oriented gradients. Additional improvements could be obtained by investigating further regression techniques, like Gaussian processes or Bayesian networks that can inherently handle space and time.

The developed method sets the stage for numerous future research directions and improvements. The current implementation assumes rather constant height up to few centimeters and no yaw rotations of the MAV. While a quadroter can move in every direction without performing yaw movements, using the algorithm on another vehicle could require arbitrary yaw movements. To limit the complexity of the dataset, a "derotation" of the incoming image could be performed to align it with the underlying images of the dataset. While the current approach normalizes each 5×5 image

patch to unit mean and zero variance—giving robustness to different lighting conditions—this procedure could be further extended, for example by using specific color models.

While the current map evaluation approach used existing fixed images, it could also serve as a fitness function for an optimization approach—for example, an evolutionary algorithm—which modifies a given image. This could allow to find a near optimal solution for a given regression technique and give insightful view in the underlying structure of certain regression techniques. 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 proposed algorithm.

The presented software *draug* generates image patches based on 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 only—ideally requiring no filtering. In the future, the possibility to set flight routes by setting way points above an image could be included. This would allow to test 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. In the current implementation using k-NN regression, larger training data sets are penalized due to a greater prediction time. However, the choice of a different regression technique is not as straightforward as it might seem. The technique should be able to output multiple predictions, since certain map regions might be ambiguous.

The presented approach is a vision-only approach. This makes it robust to external disruptions such as magnetic fields and reduces the amount of points of failure. Additionally, the approach can be used on different devices, such as handheld cameras. Still, future developments could incorporate data from the inertial measurement unit (IMU) in the particle filter's motion model.

Additionally, the time complexity of the algorithm can be further reduced with the aim of running the algorithm on fly-sized MAVs. Depending on the target platform, parallelization or threading could be used on multi-core systems to simultaneously compute texton histograms, make predictions and run the particle filter.

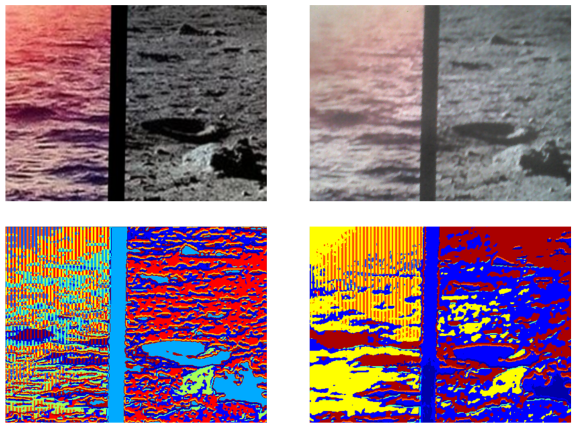

Figure 5.1: Exemplifying the reality gap. *Top left*: image patch generated using the synthetic data generation tool. *Top right*: image patch taken with the MAV's camera after printing the patch. *Below left*: texton image of the synthetic image. *Below right*: Texton image of the real image. The texton images shows that corresponding regions get classified into different textons, resulting in different histograms. This makes the transfer from the synthetic data to the real world difficult.

Chapter 6

Conclusion

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 allows their use on a wide range of models. Examples of these adaptable algorithms are 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 proposed 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 et al. "Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments". *Robotics and automation (ICRA), 2011 IEEE international conference on*. IEEE. 2011, pp. 3056–3063.

[2] Spencer Ahrens et al. "Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments". *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE. 2009, pp. 2643–2648.

[3] Eman R AlBasiouny, Amany Sarhan, and T Medhat. "Mean-shift-FAST algorithm to handle motion-blur with tracking fiducial markers". *Computer Engineering & Systems (ICCES), 2015 Tenth International Conference on*. IEEE. 2015, pp. 286–292.

[4] Adrien Angeli et al. "2D simultaneous localization and mapping for micro air vehicles". *European Micro Aerial Vehicles (EMAV)*. 2006.

[5] Abraham Galton Bachrach. "Autonomous flight in unstructured and unknown indoor environments". PhD thesis. Massachusetts Institute of Technology, 2009.

[6] Nitin Bhatia et al. "Survey of nearest neighbor techniques". *arXiv preprint arXiv:1007.0085* (2010).

[7] Michael Blösch et al. "Vision based MAV navigation in unknown and unstructured environments". *2010 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 21–28.

[8] Pascal Brisset et al. "The Paparazzi solution". *2nd US-European Competition and Workshop on Micro Air Vehicles 2006*. 2006.

[9] Haiyang Chao, Yu Gu, and Marcello Napolitano. "A survey of optical flow techniques for UAV navigation applications". *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*. IEEE. 2013, pp. 710–716.

[10] Hung-Kuo Chu et al. “Halftone QR codes”. *ACM Transactions on Graphics (TOG)* 32.6 (2013), p. 217.

[11] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. “Torch7: A matlab-like environment for machine learning”. *BigLearn, NIPS Workshop*. EPFL-CONF-192376. 2011.

[12] *CPSC 340: Machine Learning and Data Mining*. URL: `https://www.cs.ubc.ca/~schmidtm/Courses/340-F15/L7.pdf` (visited on 08/10/2016).

[13] Pablo d’Angelo et al. *Hugin - Panorama photo stitcher*. URL: `http://hugin.sourceforge.net/` (visited on 08/10/2016).

[14] G.C.H.E. De Croon et al. “Design, aerodynamics, and vision-based control of the DelFly”. *International Journal of Micro Air Vehicles* 1.2 (2009), pp. 71–97.

[15] GCHE De Croon et al. “Sub-sampling: Real-time vision for Micro Air Vehicles”. *Robotics and Autonomous Systems* 60.2 (2012), pp. 167–181.

[16] G.C.H.E De Croon et al. “The appearance variation cue for obstacle avoidance”. *IEEE Transactions on Robotics* 28.2 (2012), pp. 529–534.

[17] Frank Dellaert et al. “Monte carlo localization for mobile robots”. *IEEE International Conference on Robotics and Automation, 1999.* Vol. 2. IEEE. 1999, pp. 1322–1328.

[18] Alexey Dosovitskiy et al. “Discriminative unsupervised feature learning with convolutional neural networks”. *Advances in Neural Information Processing Systems*. 2014, pp. 766–774.

[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.

[20] Daniel Eberli et al. “Vision based position control for MAVs using one single circular landmark”. *Journal of Intelligent & Robotic Systems* 61.1-4 (2011), pp. 495–512.

[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.

[22] Dieter Fox et al. “Monte carlo localization: Efficient position estimation for mobile robots”. *AAAI/IAAI* 1999 (1999), pp. 343–349.

[23] S. Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. *Pattern Recognition* 47.6 (2014), pp. 2280–2292.

[24] S. Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/j.patcog.2014.01.005. URL: http://www.sciencedirect.com/science/article/pii/S0031320314000235.

[25] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. “Towards a navigation system for autonomous indoor flying”. *IEEE International Conference on Robotics and Automation, 2009 (ICRA'09)*. IEEE. 2009, pp. 2878–2883.

[26] Isabelle Guyon and André Elisseeff. “An introduction to feature extraction”. *Feature extraction*. Springer, 2006, pp. 1–25.

[27] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[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.

[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 12/10/2015).

[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.

[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.

[32] Teuvo Kohonen. “The self-organizing map”. *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480.

[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.

[34] Yann LeCun et al. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[35] David G. Lowe. “Object recognition from local scale-invariant features”. *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[36] Kimberly McGuire et al. “Local Histogram Matching for Efficient Optical Flow Computation Applied to Velocity Estimation on Pocket Drones”. *arXiv preprint arXiv:1603.07644* (2016).

[37] *Microsoft Image Composite Editor*. URL: http://research.microsoft.com/en-us/um/redmond/projects/ice/ (visited on 08/02/2016).

[38] Parrot. *CES 2015 : Parrot Bebop Dance Choreography*. 2015. URL: https://www.youtube.com/watch?v=A_3UifFb45Y (visited on 10/15/2015).

[39] *Parrot Bebop Drone*. URL: http://www.parrot.com/products/bebop-drone/ (visited on 07/14/2016).

[40] Franck Ruffier et al. “Bio-inspired optical flow circuits for the visual guidance of micro air vehicles”. *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*. Vol. 3. IEEE. 2003, pp. III–846.

[41] Stephen Se, David Lowe, and Jim Little. “Global localization using distinctive visual features”. *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*. Vol. 1. IEEE. 2002, pp. 226–231.

[42] S. Thrun. *Artificial intelligence for robotics*. URL: https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373 (visited on 03/10/2016).

[43] 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.

[44] Manik Varma and Andrew Zisserman. “Texture classification: Are filter banks necessary?” *2003 IEEE Computer Society Sonference on Computer Vision and Pattern Recognition, 2003 Proceedings*. Vol. 2. IEEE. 2003, pp. II–691.

[45] *Wiki PaparazziUAV*. URL: https://wiki.paparazziuav.org/wiki/Main_Page (visited on 07/14/2016).