Saarland University
Department of Computer Science
Master's Program in Computer Science

Master Thesis

A Learning-Based Approach for Efficient Visual SLAM

Submitted by

Jonas Scheer[1]

Supervisors

Dr. Mario Fritz[2]
Dr. Oliver Grau[3]

Reviewers

Dr. Mario Fritz[2]
Prof. Dr. Christian Theobalt[2]

December 31, 2015

[1]Saarland University, Saarbrücken, Germany
[2]Max-Planck Institute for Informatics, Saarbrücken, Germany
[3]Intel Visual Computing Institute, Saarbrücken, Germany

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Declaration of Consent

I agree to make my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, December 31, 2015 _____

Jonas Scheer

Acknowledgment

Before I begin, I want to thank my supervisors, Dr. Mario Fritz and Dr. Oliver Grau for having this great opportunity. By getting advice from Oliver with his experience in both research and industry as well as Mario with his expertise of working at the Max-Planck-Institute, I had the chance to profit from excellent and constant guidance as well as critical comments.
I would also like to thank my friends and former working colleagues Farshad Einabadi, Daniel Pohl, Carlos Fernandez De Tejada and Stanislav Eppinger for useful discussions, suggestions and especially a nice and enjoying working atmosphere.

Abstract

Visual SLAM (vSLAM) and structure from motion (SfM) are well-established techniques. However, facilitating them on mobile devices brings the problem of power consumption and battery drain. For this reason, making the processing steps of SfM/SLAM less computationally expensive, is still of big interest.

The first step of each SfM or vSLAM pipeline is feature extraction. To improve the following processing steps for SfM, Hartmann et al.[1] trained a classifier to filter out features, which are unlikely to have a match in another image. Nevertheless, this classifier is not optimized for vSLAM, but 2D feature matching.

The purpose of this work is to investigate the limits of the classifier created by Hartmann et al. We create a *feature score* that achieves the same matching performance, but is easier to compute. Furthermore, we train an own classifier, better suited for vSLAM. A random forest is trained to keep features that are part of a valid 3D reconstruction and have a long track length. Thus, our classifier is able to hand over fewer features to the subsequent SLAM/SfM pipeline but does not degrade tracking accuracy. Additionally, our classifier decreases the number of outliers within a 3D reconstruction. We compare our classifier with the one created by Hartmann et al. as well as a feature selection by increasing the DoG threshold for sift features. In particular, we demonstrate that our method is able to generate a 3D reconstruction with tracked camera poses, by approximately using 50% fewer features than the baseline methods.

Contents

Part I

Introduction & Basic Concepts

Chapter 1

Motivation

Most modern mobile devices are equipped with cameras but currently, these are primarily used for taking photographs or capturing videos. The increasing processing power available in mobile devices enables them for more sophisticated tasks, such as augmented reality, camera tracking or structure-from-motion. Such applications use camera images to get information about their environment as well as the pose of the camera within this environment. Although, mobile devices benefit from a rise in computational power, the mentioned use cases bring them quickly to the limit. Additionally, battery capacity limits the broad application of more sophisticated techniques on mobile devices.

1.1 Applications

1.1.1 Augmented Reality

Augmented reality (AR) applications sense their environment and fuse virtual content into a physical world. AR applications for smartphones or tablets, for example, overlay the camera image with useful information to support the users in specific tasks. In figure 1.1, a building overlays the camera image of a smartphone and by moving it around, the user can view the building from different perspectives. To make this work, the AR-Application detects a predefined pattern within the camera image. In the given scenario of figure 1.1, this is a black-white square pattern, printed on a white piece of paper. After finding out the position of certain pixels of the pattern within the camera image, we can relate them to real world 3D-coordinates because we know the dimensions of the pattern. This gives us 3D-2D correspondences with all 3D-coordinates lying in a single plane. Therefore, we can estimate a homography and finally determine the camera pose relative to the physical pattern. Given the determined pose, we now can adjust a virtual camera with respect to the new pose. For example, we can change the camera transformation matrix in openGL or directX and render a virtual object afterwards. By doing so, we render the object with respect to the same camera pose as the physical camera, which gives us the effect shown

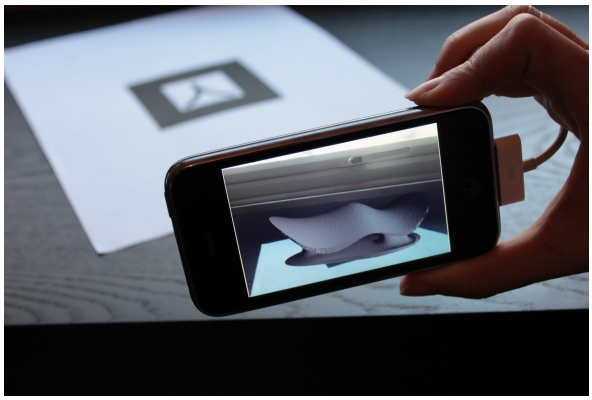

Figure 1.1: Augmented Reality with predefined marker using ARToolKit [2]

in figure 1.1.

While such AR-Applications are already flawlessly working for mobile devices, a more challenging task is to omit the use of a fiducial marker, like the black-white pattern. For that purpose, the smartphone has to orient itself within a previously unknown environment. To find out the camera pose, image features are extracted out of the camera image and tracked in subsequent frames. Due to the fact that extracted features will not necessarily lie in a 2D-plane, the estimation of a homography is insufficient and needs to be replaced with various more difficult steps. Systems that do not know the physical 3D world in advanced, but have to be tracked in such an environment are called *SLAM* systems. SLAM stands for *Simultaneous Localization and Mapping* thus, a map of the environment is created while, at the same time, the position and orientation of the system are estimated. An example of a SLAM system will be described in more detail in section 2.2.

While there are not many good to go, mobile SLAM systems, the hardware is already available whose full potential can be exploited by SLAM. With *Google Tango* a platform is offered which is explicitly designed for SLAM. Tango tablets or smartphones have built-in depth sensors like the *Microsoft Kinect* which are used to sense the surrounding environment and to track the device's position in 3D space. In 2015, Microsoft announced the *HoloLens*, a smartglasses device similar to *Google Glass*. While the processing power of Google Glass is not sufficient for AR, one of the main features of HoloLens is to perform SLAM like tasks.
An example of a professional markerless camera tracking system is *ncam* [3]. ncam is designed for the broadcast and filmmaking industry and consists of a multi-sensor bar mounted on a camera and an associated tracking server. By fusing different sensor data and evaluating them on the server, real-time camera tracking is possible. This enables a director to see instantly virtual characters

within the scene he is currently capturing. Due to ncam needs a dedicated tracking server as well as additional hardware its main priority is not to be used in a mobile scenario or as a mainstream device.

1.1.2 Structure from Motion

Besides augmented reality, another more sophisticated task for systems using cameras is *structure from motion* (SfM). By moving a camera in space, we want to sense the 3D environment to get a detailed 3D reconstruction. This is done similar to SLAM by tracking the camera pose, finding feature correspondences in subsequent frames and relate them to the 3D world. Whereas SLAM concentrates more on camera tracking as accurate as possible, SfM focuses on obtaining a 3D reconstruction as detailed as possible. Furthermore, SfM is in general performed as an offline batch process, whereas for SLAM, tracking the camera in real-time is one of the main objectives. Thus, in SfM having as many features as possible leads to a better 3D reconstruction. On the other hand, for camera tracking in SLAM, we want as little features as possible to speed up computation time. Another difference between SfM and SLAM is the fact that in a SLAM scenario images arrive sequentially one after another. SfM does not assume any ordering for the captured images. A ready to use SfM open-source software is Bundler [4] which was also used in this thesis.

1.2 Challenges for Mobile Devices

Mobile devices have a major bottleneck in contrast to stationary system. While CPU power is not the primary concern, battery life will bring essential limitations. Using batteries with a higher capacity is not a solution because this will increase the dimensions of our devices and will add weight. For example by adding batteries to make HoloLens last longer for augmented reality, will indeed make it unsuitable for most users. Thus, to use a camera of a mobile device for more computationally intensive applications, we have to improve the algorithms we are using to reduce CPU usage and consequently battery drain.

The first step of every SLAM or SfM system is to extract features out of captured images. For this purpose, there are several feature detectors available. Some of the most popular ones are [5] [6] [7] and [8]. [5] is currently the state of the art feature extractor and an example of extracted sift features is shown in figure 1.2. On the left side, we see an image where all possible sift features (n=15986) have been extracted. On the right side, a much smaller feature set is obtained.
In general we find a lot of image features, but many of them will give redundant information for SLAM or SfM. Consequently, by pruning redundant features the remaining SLAM/SfM pipeline has fewer data to process. Additionally, a

Figure 1.2: Left: all extracted image features (n=15986). Right: pre-selected set of features.

reduction of images could be taken into account, but for SLAM as a real-time application, each image is needed to achieve the desired results. For this reason, a reduction of features is a good starting point, if you want to improve SLAM or SfM. Fewer data has to be processed, which leads to a lower CPU usage and in particular for mobile devices, less battery drain.
Most SLAM and SfM systems use an outlier detection to discover features that resulted in a 3D point but do not fit into the 3D model. While this step makes the 3D model more robust, not all outliers are detected. Thus, the resulting is inaccurate and also can perturb camera tracking. Besides gaining a computational speed-up with fewer features, also a better camera pose estimation can be achieved. A feature reduction method should remove as many outliers as possible while retaining features that contribute to camera tracking.

1.3 Contribution of the Thesis

The main purpose of this thesis is to make SLAM/SfM less computational expensive by using fewer, but more significant images features. This is achieved with the help of learning based methods. To improve SfM, Hartmann et al. created a classifier [1], that selects features that are likely to have a match in another image. They demonstrated a speed-up for feature matching in SfM, due to the significant reduction of features.
In this thesis, the limitations of [1] are investigated. We propose a *feature score*, which can be used to classify features. This score is simple to compute and gives comparable performance as the method of Hartmann et al.
Additionally, we propose the assumption that a further reduction of features can be achieved by using only features that belong to a valid 3D model and have a long track length. We compare properties of long track features with the features that are selected by the classifier of Hartmann et al.
Finally, we train a random forest that selects features, which follow the long track criterion. We investigate, how many features are sufficient to create a 3D reconstruction and to get the camera poses for all used images. The feature reductions also leads to a decrease of outliers, while more significant features are

preserved. Furthermore, we explore the track length of the features preserved by the different classifiers.
As a baseline for our classifier we use the method of Hartmann et al. as well as a feature reduction, achieved by increasing the DoG threshold for sift features.
All contributions of the thesis are listed below:

- Feature Score competitive with [1].
- Criterion for SLAM: preserve long-track features
- Learn random forest
- Evaluation of classifier by investigating:
  - Feature reduction
  - Decrease of outliers
  - Track length of features
  - Measure the accuracy:
    - Reprojection error for 3D reconstruction
    - Rotation/translation error for registered cameras within 3D reconstruction.
  - Baseline methods:
    - High DoG threshold for sift
    - Classifier of Hartmann et al. [1]

1.4 Outline

The following document is divided into 3 parts. Part I gives a short introducing and investigates related literature of SLAM and feature selection. Especially the baseline classifier [1] is explained in more detail with its limitations. Additionally, the basic concepts, for the remaining thesis are presented. Part II starts with the *feature score*, looks at long-track features in more detail and explains the training our classifier. The last two chapters of Part II deal with the evaluation of the constructed classifier and present the tools used during this work 6. To finish the thesis, we give a short outlook for future work and summarize the results.

Chapter 2

Related Work

To make SLAM more computational efficient regarding CPU workload, the next two chapters summarize recent techniques used for camera tracking and 3D reconstruction. In particular, the main building blocks of a SLAM system are explained, such that the benefit of feature reduction is emphasized. Afterwards, three methods for selecting more significant image features are analyzed. Most notably, the work of Hartmann et al. [1] focuses on SfM and shows a huge speed-up for the feature matching procedure. These advantages are investigated in more detail as well as its limitations that justify our approach to learn long track features.

2.1 Advancements in visual SLAM

In the previous century, SLAM techniques were mainly used, to track a robot in an unknown environment and to give him the opportunity to navigate through unexplored terrain. For this purpose, robots were equipped with external sensors as in [9]. An ultrasonic sensor, to obtain information about the physical world, was attached to a robot. For a certain point in time t the robot can use the sensor information s_t and compare it to previously obtained data s_{t-1} to find out its new position. After cameras had started becoming smaller and cheaper, the ultrasonic sensor (or other sensors) was replaced by a conventional camera and SLAM was performed using camera images. At the beginning of visual SLAM, so called Filtering methods were used to hit real-time constraints. With the help of a *Kalman Filter* for example, the information of several previously taken images were fused to get the position of a new image. Another approach for estimating the camera pose (position and rotation) is *bundle adjustment*. Due to the fact that bundle adjustment is much more time consuming, filtering methods were the dominant technique, when performing SLAM. After computational power has grown and tasks could be separated on multi-core platforms, bundle adjustment started catching public interest. In [10] Strasdat et al. showed, that bundle adjustment outperforms filtering, so bundle adjust-

ment was used in most work that followed. While PTAM [11] only uses bundle adjustment for certain frames (Keyframes), newer systems like ORB-SLAM [6] completely rely on bundle adjustment to get the camera pose of each frame.
More recent approaches as [12] even use RGB-IR sensors to perform camera tracking. However, most RGBD sensors have a much more limited range in contrast to cameras that acquire images using a wide baseline. Furthermore, RGB-IR performs poorly in outdoor scenes under direct sunlight. To not limit ourselves to bright, short range scenes only, we decided to stick with conventional cameras instead of RGB-IR cameras.
In [13], a stereo camera system was used. By considering feature extraction, these systems will also benefit from having a lower amount of features, due to the remaining SLAM task are similar to the monocular setup.
To sum up, using a single camera for SLAM, will not constrain our research to a limited amount of scenes. Furthermore, with bundle adjustment, a main building block of most modern SLAM systems is taken into account. The processing steps of a SLAM pipeline for monocular cameras using bundle adjustment is presented in the next section in more detail.

2.2 SLAM in a nutshell - PTAM

To understand the different SLAM processing steps and figure out the bottlenecks, PTAM [11] is presented. With PTAM, Klein et al. created a real-time SLAM system by separating the task of camera tracking and map building into two separate threads. Figure 2.1 shows an example of PTAM with an estimated ground plane and the features PTAM is currently tracking (colored dots). By moving the camera in space, new features are added and the region in which the camera can be tracked grows. To meet real-time requirements, PTAM is only suited for small workspaces. In the remaining of this section the main building blocks of PTAM are explained.

2.2.1 Initialization

In camera tracking applications with markers, as described in chapter 1 the camera pose is determined with respect to a predefined pattern. One of the main challenges of markerless camera tracking is to find some reference points in the physical 3D environment, without using any predefined pattern. Thus in SLAM, the user is not limited to a particular scene. PTAM solves this problem by finding stereo correspondences in two images and using the five-point stereo algorithm of [14] employing RANSAC to get an essential matrix. Using triangulation, a base map can be determined which is finally refined through bundle adjustment. To apply all these steps, the user takes one image of the scene, moves the camera approximately 10cm to the right and takes another picture. This procedure assures a sufficiently large baseline such that stereo correspondences are likely to be found. Thus, instead of using a predefined marker, image features are automatically extracted, matched and mapped to 3D real world co-

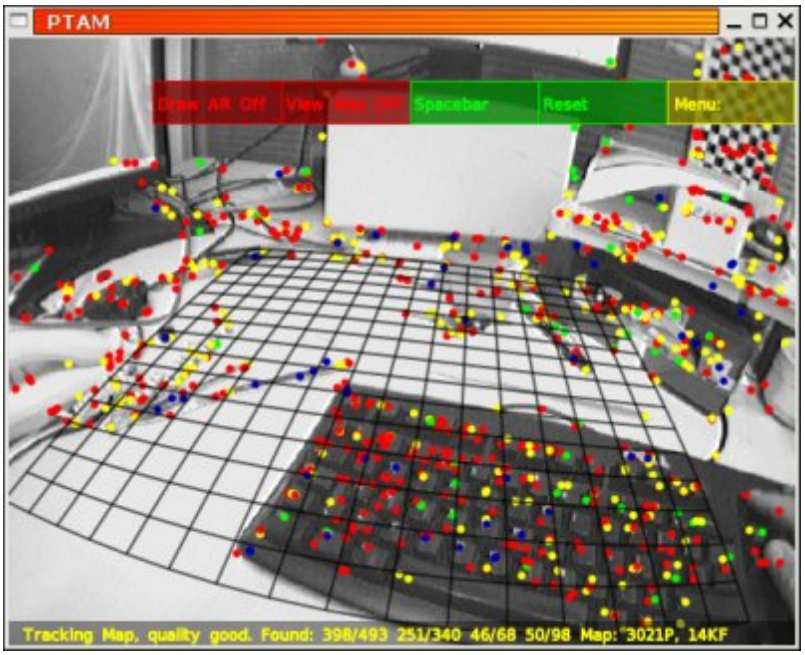

Figure 2.1: PTAM [11] during tracking image features.

ordinates. These 3D points are stored in a map and in the upcoming SLAM steps, they are used to determine the current position of the camera.
In this first step, a large amount of extracted features and corresponding 3D points leads to a lot of redundant information for camera tracking. This slows down the subsequent tracking and map building steps, especially in a mobile scenario. One of our most important contributions is to reduce the set of features to a minimum without decreasing the tracking accuracy. Furthermore, our approach removes many outliers that are not discovered by the outlier detection build into most SLAM systems. Such outliers perturb the 3D model and make tracking inaccurate. For this reason our approach, is already useful for initializing the system.

2.2.2 Keyframes and Map Building

PTAM uses the concept of *Keyframes* to track the camera path and connect 3D reference points of different frames. Instead of storing image features and the resulting 3D points of each single frame, this is only done for individual images. For an only slightly moving camera, mostly redundant information will be created which will result in a huge waste of storage. Thus, PTAM stores image features and 3D points together with so-called Keyframes, if the following criteria are met: The time, since the last keyframe was added to the system must exceed 20 frames. Furthermore, the camera has to move a minimum distance.

This gives a minimal baseline and ensures a certain quality for new feature triangulations. By adding a new keyframe, PTAM adds new 3D points to the map and the tracked scene grows. Consequently, a keyframe consists of 3D points already available in the map, which are used for tracking the current frame and new 3D points which will be generated via triangulation. Finally, after a new Keyframe was added to the system, PTAM performs bundle adjustment to optimize the poses of all Keyframes as well as the positions of the 3D world points. Obviously, this approach is limited to a certain amount of 3D points. Otherwise, the bundle adjustment step breaks the real-time constraint. By reducing the number of used features, a larger workspace can be explored by PTAM. Additionally, the advantages of the initialization procedure also hold for this processing step. Adding fewer features while adding a keyframe makes subsequent tracking faster and fewer outliers perturb the 3D model.

2.2.3 Camera tracking

By applying the steps described in the previous sections, an already built up map is given. For a newly acquired image, the camera pose is determined using the map and its contained 3D points. PTAM uses a prior pose estimate by assuming a camera movement in the current tracking step, similar to the previous tracking step. For the last tracked camera position C_{t-1} we can compute the velocity v_{t-1} of the camera movement from C_{t-2} to C_{t-1}. Using the same velocity as well as the directional vector $d = C_{t-1} - C_{t-2}$, we get the prior pose C_t for the current tracking step. In a next step, the stored 3D map points are projected into the camera image according to the estimated prior pose. By doing so, the pinhole camera model can be utilized and the 3D-2D projection can be described as in [15]:

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} * \big[R|T\big] * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.1}$$

The leftmost matrix on the right side of the equation is called intrinsic matrix I. The matrix $[R|T]$ is a 4×4 matrix consisting of a rotation and translation. Consequently, R is a 3×3 rotation matrix and T a 3×1 translation vector. $[R|T]$ is in homogeneous coordinates. $[R|T]$ is related to the camera movement mentioned above. I includes the focal length of a camera (a_x and a_y) in x and y-direction as well as the center point shift (x_0 and y_0) in both directions. Thus, $[R|T]$ moves a 3D point or the camera in space and I projects a 3D point $(X, Y, Z, 1)^t$ to 2D image $(u, v, 1)^t$.
After a 3D point was projected into a view, image features are extracted. Finally, for each 3D point a feature match is searched around its 2D projection. With the help of the projection, the feature search can be restricted to a certain patch only. This results in 2D-3D correspondences that can be used together with the prior pose to refine the camera position with respect to the map. To compute the prior pose update, the reprojection error is minimized. The get the

reprojection error of a single pixel, the difference of a projected 3D world point into the 2D view and its extracted image feature is computed. By summing up all these differences or using an appropriate loss function, the final pose is obtained by solving an risk minimization problem.

2.2.4 Limitations of PTAM

The projection of all 3D points in the current camera frame and the computation of the camera pose update, described in section 2.2.3, limit PTAM to small workspaces only. Thus, after a certain amount of 3D points are captured, PTAM is not able to operate in real time anymore. Reducing the number of features reduces the following matching computations and bundle adjustment needs to take fewer data into account. Additionally, our method removes outliers while reducing features. Thus, the resulting 3D map will be more accurate.
Another drawback of PTAM is the inability to detect loops. Loop closure is only feasible in simple scenarios. While ORB-SLAM can handle loop closure by using the bag of words approach presented in [16], improving the detection of loops is not considered in this work. Therefore, using less but more robust features, especially features with long tracks could lead to better visual words and improve the bag of words method. This is indeed not in the scope of this work.

2.3 Significant Feature Selection

features that In this thesis, an speed-up in estimating the camera position and orientation is achieved by selecting fewer features. The processing steps of SLAM, mentioned in the previous section become less computationally expensive. Additionally, the overall robustness and tracking accuracy increases by having a less perturbed 3D model, due to fewer outliers. In the following, three methods are presented to select more significant features and their limitations, when it comes to SLAM or SfM.

2.3.1 Utilize Symmetry Properties

Hauagge et al. presented an approach [17] to extract features that follow certain symmetry properties. They discovered that images of buildings appear differently depending on when they were captured and at which scale the images are taken. For example, having three images of a building, taken at different scales. $Image_1$ is a historical image while $Image_2$ was take in entirely different lighting conditions than $Image_3$. For such images, SIFT will create different feature descriptors for the same feature such that obvious feature matches will not be found.
Hauagge et al. try to detect features with bilateral and radial symmetries. A feature will be assigned a high score, if flipped across a symmetry axis, there

is another feature with similar values. They use two different similarity measurements and their results are shown in figure 2.2. If there is a feature pair,

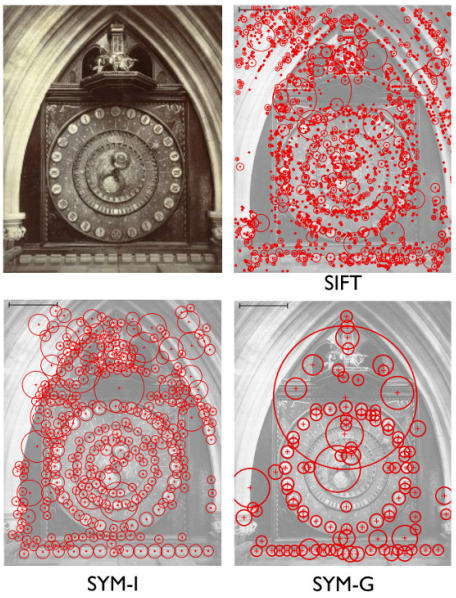

Figure 2.2: Symmetry features based on intensity (SYM-I) or gradient (SYM-G) values as well as SIFT features. (figure taken from [17])

perpendicular to the axis of symmetry with similar intensity values, both features are assigned high scores. In figure 2.2, the axis of symmetry is right in the middle of the image and features perpendicular to it can be found at the bottom of the images SYM-I and SYM-G. For the radial symmetries, a rotating axis through a point p is chosen, which results in circularly aligned features in figure 2.2.
Although this approach seems to detect a perfect set of features, it can not be directly applied to SLAM. The presented method was designed for SfM using scenes of buildings. Images in the given scenario do follow the assumption to have a vertical axis of symmetry in the middle of the image. However, this assumption is not true for most SLAM scenarios. Furthermore, SLAM is not necessarily applied to building scenes, thus to find the axis of symmetries is not a trivial problem. Without making to many assumptions about the scene, finding a proper setup of symmetry axis will break the real time assumptions of SLAM. Our feature reduction method is designed for arbitrary scenes without any predefined scene structure.

2.3.2 Good Features to Track for Visual SLAM

Zhang et al. [18] proposed a method to select *good features* for SLAM, which are of high utility for camera localization. They make use of observability theory

by assigning a high score to a feature if it can be used to determine the new position of a camera. Let us consider a camera movement m from c_1 to c_2 and camera images I_1 and I_2 with corresponding features $f_1 \in F_1$ and $f_2 \in F_2$. For given scene points $p \in P$, we can project them to I_1 and I_2. If we can find features for the projected points, these features can be *observed* under the given movement m. It is important that a single 3D point results in two features f_1 and f_2. Such features will be assigned a high observability score. The whole system dynamics are described by Zhang et al. in a linear system of equations. A single equation is given by the projection of one 3D world point under a given movement to image coordinates. By using observability theory, this linear system is transformed and a *Total Observability Matrix* (TOM) is built. A system is fully observable if TOM has full rank. Finally to determine, if a TOM has full rank, a singular value decomposition is performed.
Although it is a good idea to select only features with a high observability score, a much simpler and less computationally expensive method can be implemented in every SLAM system. During the initialization procedure as discussed in section 2.2.1 or while adding new scene points as in 2.2.2, we can add a counter to each 3D point. While tracking the camera, we can increment the counter if there was no feature found for a given 3D point. If the counter exceeds a certain threshold, the 3D point can be removed from our map thus, it becomes unobservable. This overcomes the singular value decomposition computations used by Zhang et al. Especially for high-resolution images, we get many features, so a simpler approach is favoured.
Additionally, most SLAM or SfM systems use outlier detection, usually after performing bundle adjustment. This guarantees that only observable 3D points will stay in the resulting 3D model. The kind of outliers which are not discovered by the outlier detection, but still perturb the final 3D model also follow the rules of system dynamics. Thus, they result in a high observability and can not be removed with the presented technique.
Zhang et al. need several frames to state if a feature has a high score or not. Additionally, a score can be updated if a feature has a high observability in later frames. For this reason, all 3D points are kept, not only the ones with a high observability score. Thus, In a SLAM pipeline as described in section 2.2, all 3D points have to be projected to a newly acquired frame. This means the following matching procedure does not benefit from a reduced set of features. Bundle adjustments is the first step that profits of the proposed method. However, bundle adjustment is one of the last steps in a SLAM pipeline thus, there are no significant improvements. Our classifier, on the other hand, improves the whole SLAM pipeline after features have been extracted.

2.3.3 Predicting Matchability

The work of Hartmann et al. [1] builds the basis for this thesis. A classifier was trained, with the main objective to speed up feature matching. For this purpose, they labeled features as *positive* if they can find a match within the training sequences, *negative* if not. With the help of this generated training

data, they learn a random forest that only picks features that are likely to have a match in another image. In the remaining part of the thesis, these features are referred to as *Matchable Features* or simply *Matchable*. Hartmann et al. compare their method with a random selection of features as well as an increased DoG threshold for sift features. The resulting reduced feature sets are shown in figure 2.3.

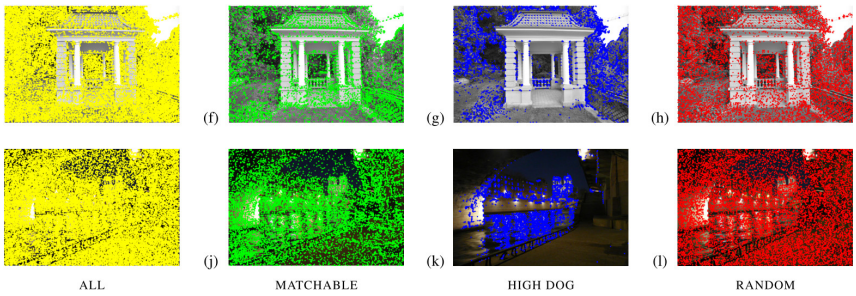

Figure 2.3: Feature selection results of [1]

The images show two scenes with the different features highlighted. For *All* features, the sift algorithm extracts plenty of features. For *Matchable* features there are much less features and the sets of features for the high DoG and the random selection are chosen as big as the Matchable set. While the Random selection clearly has no structure, the high DoG picks features in high-contrast regions as on vegetation for the upper image scene. Matchable features, on the other hand, are more concentrated around the building. The vegetation gives a highly textured area in which it is even difficult for humans to find interesting points. However, onto the building we have symmetric, clearly distinguishable elements that are expected to give robust features.
Figure 2.4 shows the results accomplished by Hartmann et al. for the different methods. They evaluate their method with matching ground truth data. This means, for a given feature in the test dataset, we know if it is a matchable feature or not. The detailed generation of the matching ground truth is explained in section 2.5, because it involves some deficits when it comes to SfM or SLAM. The ROC curve of the matchable features (green) is always better than the ROC curves of the random selection (red) and the high DoG (blue). This shows that a feature reduction using the classifier picks features that are likely to have a match in another image, while the other two methods do not. The three methods are tested on three different datasets. The PARK dataset is also shown in the first row of figure 2.3 and contains mainly vegetation. The URBAN dataset was captured in an Urban environment and the NOTRE DAME dataset is an unordered collection of Notre Dame images.

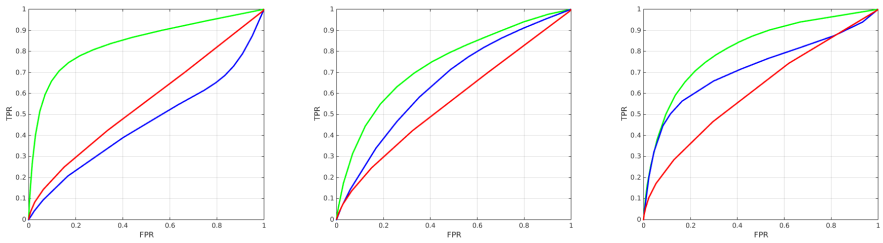

Figure 2.4: ROC curves of Park, Notre Dame and Urban dataset. Green: matchable features. Red: random selection. Blue: high DoG. (images taken from [1])

2.4 Benefits of Feature Classification

The previous section presented the feature selection method of Hartmann et al. In this section, the benefits of their work for SfM is explained in more detail. Investigating the matching procedure, shows that there is a profit of fewer features, due to fewer comparisons. To get all the feature matches in one image pair, an exact implementation needs $\mathcal{O}(n^2)$ comparisons, without making any assumption on the captured scene. Nevertheless, even inexact solutions like finding an approximated nearest neighbor (ANN) [19] or using locality-sensitive hashing techniques [20] only need superlinear complexity. While this speeds up feature matching, it is still not sufficient for SLAM or SfM applications. In figure 2.5, Hartmann et al. show how feature matching benefits by using their classifier. The left column uses all features available in an image, the right col-

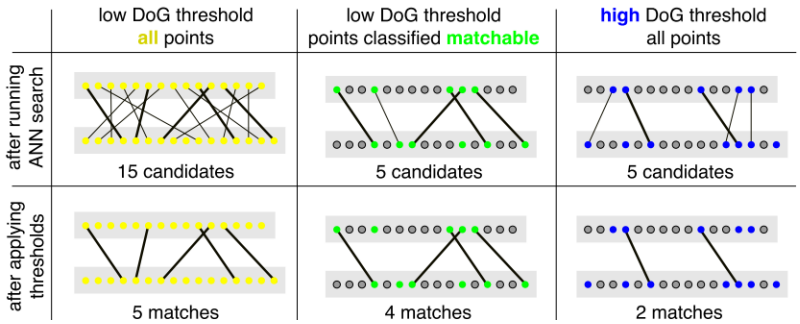

Figure 2.5: Feature reduction and matching using different pruning techniques. Hartmann et al. compare their approach [1] to feature selection using high DoG.

umn only features, for which the detector has found high DoG responses and the column in the middle only uses features selected by the proposed classifier. The first row shows the feature matches using ANN, but not all of them are valid sift matches (valid sift matches are defined in more detail in section 3.1). For this reason, Hartmann et al. try to predict which features will end up with a valid matching partner, what can be seen in the middle column of figure 2.5.

Nearly all candidates which are likely to have a suitable match are picked by the classifier, while all others are pruned. However, using the high DoG approach more features are discarded which have a valid matching partner, but also features are preserved which do not. Using all features, as shown in the left column is even worse, due to the matching procedure checks many features that do not have a match at all, thus is wasting a lot of computation time.
Hartmann et al. even examined the gained speed up for matching less features explicitly. They performed tests on three datasets and considered the time needed to detect features, build KD-trees which are used by ANN feature matching and also included the time to classify features when using their method. Figure 2.6 shows the results of Hartmann et al. All three datasets

URBAN	ALL	MATCHABLE	HIGH DOG
# image pairs		499,500	
DoG+SIFT	1,530 s	1,530 s	990 s
build KD-trees	140 s	35 s	35 s
query KD-trees	65,240 s	18,230 s	18,230 s
classification	—	80 s	—
total	66,910 s	19,875 s	19,255 s

PARK	ALL	MATCHABLE	HIGH DOG
# image pairs		7,260	
DoG+SIFT	780 s	780 s	345 s
build KD-trees	215 s	35 s	35 s
query KD-trees	11,895 s	1,960 s	1,960 s
classification	—	70 s	—
total	12,890 s	2,845 s	2,340 s

NOTREDAME	ALL	MATCHABLE	HIGH DOG
# image pairs		255,255	
DoG+SIFT	2,820 s	2,820 s	1,740 s
build KD-trees	600 s	205 s	205 s
query KD-trees	180,755 s	74,680 s	74,680 s
classification	—	205 s	—
total	184,175 s	77,910 s	76,625 s

Figure 2.6: Feature matching speed up, achieved by Hartmann et al. [1].

show a huge saving of computation time when using fewer features. While using high DoG is slightly faster than the used classifier, it is less accurate. The improvement of the classifier against using all features is 28.8% for the URBAN dataset, 22.0% PARK dataset and eve 42.3% for the NOTRE DAME dataset. Thus, giving the performance of the classifier as shown by the ROC curves in figure 2.4, a huge amount of valid feature matches are preserved while the speed-up can be up to 42.3%.

2.5 Deficits of Matchable Features

After we have shown the benefits of feature reduction, especially with the analysis done by Hartmann et al. we want to discuss some limitations of their work. The classifier of Hartmann et al. is well suited to pick features that most likely have a sift feature match in another image. However, such features are not necessarily of high importance for SLAM or SfM. As shown in figure 2.7, features can have a matching partner, but do not belong the same 3D point of an underlying scene. While there are consistent features in both images like the

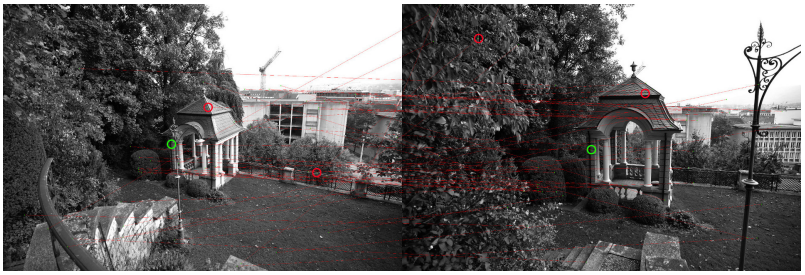

Figure 2.7: Geometrically correct (green) and wrong (red) sift matches.

feature on the house, marked with a green circle, the sift algorithm also gives wrong feature matches. The two feature pairs marked by red circles are the best correspondences according to the matching criterium described in [5], but geometrically they do not belong to the same 3Dfeatures that world point. A more detailed characterisation of *wrong* and *valid* feature matches is given in chapter 3.
Hartmann et al. did not consider these observations when creating their training and test data. Consequently, their random forest is not able to select features that belong to a valid 3D reconstruction and discard feature which do not. Additionally, their evaluation method using ROC curves gives no insights about the usability of a feature in SfM or SLAM, but only for matching.

We make use of the idea of Hartmann et al. to select features that are more suitable for SfM or SLAM by using learning based methods. By looking at the limitations of their classifier, we are able to present a classifier which is more appropriate for the SLAM or SfM scenario. Our classifier takes only features into account which do belong to a valid reconstruction. To achieve an even higher feature reduction, we go one step further and pick only the features that have a long track length. These features are not only matchable on the 2D image level but also show greater benefits for SfM and SLAM. By focusing on long track features, the number of features will be reduced significantly without making camera pose estimation less accurate.

Chapter 3

Preliminaries for Feature Selection

This chapter provides the basic concepts, used in this thesis. At first, a closer look at image features is taken in particular, the insights of the SIFT algorithm are presented. The construction of the feature descriptor is explained as well as the matching procedure. Additionally, some definitions for feature matching are introduced. With their help, we can create *feature scores* in chapter 4 and use them as a reference for the feature classification, proposed by Hartmann et al. To apply learning based methods to select more significant features, a random forest is used. For this reason, the creation and classification with the help of a random forest is discussed in this chapter.

3.1 SIFT

Lowe created SIFT [5] with the objective to detect interesting points within an image, which are robust against scale changes, translations and rotations. Features are extracted with a difference-of-Gaussian approach, so interesting points at different scales are detected. To make a feature, represented by its feature descriptor, invariant against scale, translation and rotation changes, its local neighborhood is examined. The neighborhood of a feature needs to be compensated for scale, location and rotation. For scale compensation, the descriptor is created on the same scale as the corresponding feature was detected. Due to the neighborhood is always relative to a given feature the location of a feature's neighborhood is automatically compensated. For a rotated neighborhood the dominant orientation is extracted. After rotating the neighbourhood according to this direction, the feature is robust against rotational changes.
For a given image feature and its compensated neighborhood a feature descriptor can now be computed. In a first step, the gradient for each pixel in a 16×16 neighborhood is calculated. These obtained directional informations are summed up in a 4×4 directional histogram, using gaussian weights to control

the influence of more distant pixels. This results in 16 histograms for 8 directions which gives us a feature vector of size 128. To obtain a unit vector, the feature vector is normalized. Figure 3.1 illustrates the feature descriptor computation in a 8×8 neighborhood and the accumulation to a 4×4 histogram. The image gradients in the 8×8 patch result in a 4×4 histogram. To summarize, the sift feature descriptor is a 128 dimensional vector and contains orientational information of its neighbouring pixels.

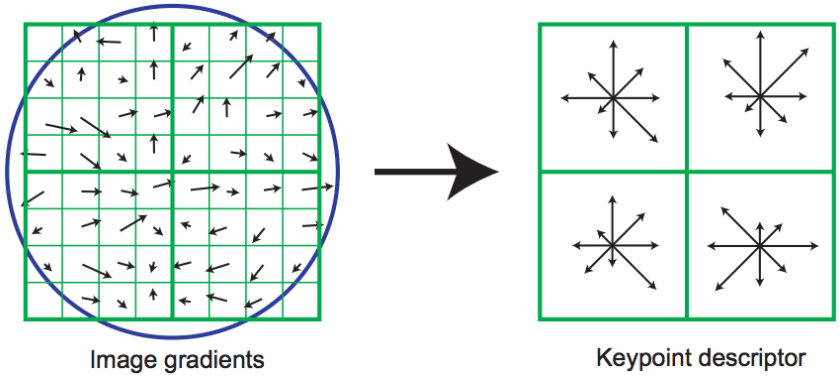

Figure 3.1: Feature histogram for a 8×8 neighborhood. (figure taken from [5])

3.2 Feature Matching

After extracting image features, SLAM or SfM applications want to find them again in another image. This is done by comparing the feature descriptors in both images. Feature matching tries to minimizing the Euclidean distance to find the most similar feature descriptors in two images. This gives us the *Nearest Neighbour* and the *Second Nearest Neighbour* of a feature:

Definition 3.1 (**Nearest Neighbour**).
For two images I_1 and I_2, the Nearest Neighbour for feature $feat_1 \in I_1$ with descriptor d_1 is the feature $feat_2 \in I_2$ with descriptor d_2, for which holds:

$$NN_{I_2}(feat_1) = \min_{feat_2 \in I_2} \{\|d_1 - d_2\|\}$$

$NN_{I_2}(feat_1)$ will also be abbreviated with $NN(feat_1)$ or simply NN.

Definition 3.2 (**Second Nearest Neighbour**).
Given a feature $feat_1 \in I_1$ and its Nearest Neighbour $NN \in I_2$. For the Second

Nearest Neighbor $NN2 \in I_2$ *of* $feat_1$ *holds:*

$$NN2(feat_1) = \min_{feat \in \{I_2 \setminus NN\}} \{\|d_1 - d_2\|\}$$

$NN2_{I_2}(feat_1)$ *will also be abbreviated with* $NN2(feat_1)$ *or simply* $NN2$.

We use NN and its value determined by the euclidian distance of its descriptor interchangeably: $NN = \|NN\| = \|d_{NN}\|$. NN_2 is always greater or equal to NN, otherwise NN_2 would be the best match. The second nearest neighbor is used to discard non-unique matches. For example in an image of a checkerboard, you can match a feature multiple times within the other image. Without having an additional procedure which is, for example, aware of the structure of the features, matching can not be done unambiguously. Hence, for real world scenes, features might be similar but not completely the same. The nearest neighbor is used to detect the unambiguously of a feature. By computing the ratio r of two features as in equation 3.1, you can prune features that are very similar to some others.

$$r = \frac{NN}{NN_2} = \frac{\|d_{NN}\|}{\|d_{NN_2}\|} \tag{3.1}$$

Thus, you take the euclidian norm of the descriptors of both NN and $NN2$ and compute the ratio. If the best match and the second nearest neighbor have nearly equal values, the ratio comes close to 1. If they differ, the ratio becomes smaller. To reject features that might be unambiguous, only features with a ratio lower than a certain threshold are preserved. For this reason a valid match is defined as follows (as suggested in [5]):

Definition 3.3 (**Valid Match).**
Given the nearest neighbor NN *and the second nearest neighbor of a feature* f*, than* f *is a valid feature, if it passes the ratio test:*

$$r = \frac{NN}{NN_2} < t$$

For some threshold t.

Hartmann et al. used a threshold of $r = 0.8$ and we follow their example. Nevertheless, our approach is more robust against changes, due to the features we use fro training, as presented in section 5.2. Our features will belong to a valid 3D reconstruction. Thus, a lower ratio threshold will give more features that are pruned again, if they can not be assigned to 3D world points. Only a threshold which is too high can lead to different results. By doing so, features that belong to a 3D model can be lost.

3.3 Random Forest

Similar to Hartmann et al, a random forest [21] is used for feature classification. A random forest consists of several decision trees and is trained in an offline

process. During training of a random forest, bootstrap aggregating is exploited to avoid overfitting and reduce variance. For our classifier we use the same configuration as Hartmann et al. and limit the depth of the random forest to 25. We use the library of Stefan Walk to produce a random forest and perform classification, which is explained in more detail in section 6.4.
In the following sections, training and classifications with the help of a random forest is explained in more detail.

3.3.1 Decision Trees

To understand how a random forest works, learning with a single decision tree must be explained first. A decision tree is a rooted tree, for which in each non-leaf node, a decision according to a certain features attribute is made. For the tree in figure 3.2 the different attributes are the weather conditions: outlook, humidity and windy. Depending on the attributes of a test feature, a decision can be made. In section 3.1 the sift descriptor is explained which is used as

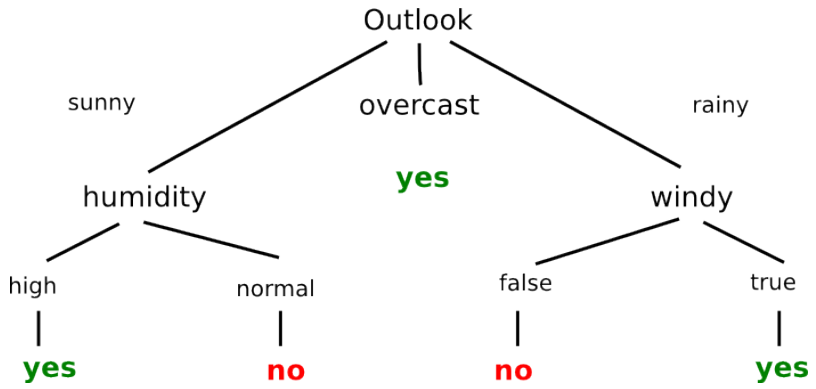

Figure 3.2: Decision tree with weather conditions as feature attributes.

image feature. This leads to a 128 dimensional vector that is used to traverse the decision tree which gives a categorization for it. To create a random forest from a set of training data, a certain feature attribute is considered for all features. According the different values of these attributes, a split criterion is created. The mostly used split metrics are the variance reduction as in [22], the entropy [23] as a measure of information gain of the *Gini impurity* as described in section 3.3.2. After a split criterion was created for a node, the remaining attributes can be used for creating additional splitting nodes. The most valuable attribute is chosen as root node and the remaining nodes are chosen with decreasing importance for larger depth. To find out the importance of a node, the entropy of a attribute can be computed. This procedure results in a decision tree that can be used to categorize a feature of the test dataset.
The main problem of decision trees is that they can grow very deep and tend to overfit. This problem is solved by using several trees trained on different parts

of the same training set. Each tree can vote for a category and aggregating all votes gives the final classification of a feature.

3.3.2 Splitting Criterion - Gini Impurity

The gini impurity is a measure of misclassification and is used by the *CART* algorithm [22]. The gini impurity is measured as follows:

$$I(f) = \sum_{i=0}^{m} f_i * (1 - f_i) \tag{3.2}$$

For a given set of items $i \in \{1, 2, .., m\}$ (feature attributes in our case). f_i is the fraction of items having the value i. Thus, the gini impurity for a set of feature attributes is computed by summing the probability f_i of having such a attribute value times the probability $1 - f_i$ of a mistake in categorizing this attribute. By rewriting equation 3.2, the misclassification property can be see more easily:

$$\sum_{i=0}^{m} f_i * (1 - f_i) = \sum_{i=0}^{m} (f_i - f_i^2) = \sum_{i=0}^{m} f_i - \sum_{i=0}^{m} f_i^2 = 1 - \sum_{i=0}^{m} f_i^2 \tag{3.3}$$

which leads to:

$$1 - \sum_{i=0}^{m} f_i^2 = \sum_{i \neq k} f_i * f_k \tag{3.4}$$

3.3.3 From Decision Trees to Random Forest

Instead of a single decision tree, multiple trees are created based on several random decisions. While creating a single tree, a random sample of features with replacement is chosen out of the training dataset. According these features a attribute is chosen which is of high importance, e.g has a high entropy as mentioned in the previous section. However, not all attributes are taken into account, but only a randomly chosen subset of attributes. Based on the randomly selected features as well as the randomly chosen attributes a split criterion is computed (e.g by gini impurity) and a tree node is created. By using this procedure, several decision trees are generated. To sum up, the different steps for creating a tree are listed blow:

1. Sample N features at random with replacement
2. to create a node:
   (a) Sample M feature attributes
   (b) take the attribute that provides the best split (according entropy)
   (c) compute gini impurity for chosen attribute
   (d) use gini impurity as split criterion

Part II

Feature Classification for Visual SLAM

Chapter 4

Classification with Feature Scores

In this chapter we derive two metrics (*feature scores*), which are used to rank features. The presented scores measure the matching quality with respect to the next frame in an image sequence (Next-Frame Score) or the uniqueness of the feature within a single frame (Intra-Frame Score).
We check the performance of these scores and compare them with the features, selected by the method of Hartmann et al. Furthermore, the track length is investigated for the Intra-Frame Score as well as the classifier of Hartmann et al. While non of the methods is well suited for preserving long track features, this is explored in more detail in the next chapter.

4.1 Next-Frame Score

With the *Next-Frame Score* S_{next}, we rate features that can be found in the next frame of an image sequence. By doing so, we exploit the definitions 3.1 and 3.2 of the nearest neighbour and the second nearest neighbour of a feature. After computing the ratio as in equation 3.1, a metric to a measure a feature can be created.

Definition 4.1 (**Next-Frame Score).**
Given an image sequence. For an image feature with nearest neighbour (NN), second nearest neighbour (NN2) with respect to the next frame within the sequence, the Next-Frame Score is defined as follows:

$$S_{nxt} = 1 - \frac{NN}{NN2} = 1 - r$$

With r as in equation 3.1.

Thus, we exploit the ratio r as proposed in [5] to detect good matches, with

the additional restriction of NN and $NN2$ computed from the next frame of an image sequence. In contrast to section 3.1, a small ratio also gives bad feature ratings.

4.2 Intra-Frame Score

The *Intra-Frame Score* only uses the information available in the same frame as the feature was found. For comparison reasons it would be advantageous to apply the Next-Frame Score, replacing the next frame in an image sequence by the same frame. However, due to the definition of the Next-Frame Score this always results in a score value of 1. Obviously, the best match of a feature by matching an image to itself is the same feature again. For equation 3.1 the nominator has a value of zero: $\|d_{NN}\| = 0$. Hence, the Intra-Frame Score would always be 1. For this reason, we exploit to use the second nearest neighbour to compute the Intra-Frame Score:

Definition 4.2 (**Intra-Frame Score).**
For an image feature with second nearest neighbour (NN2) computed from the same image, the Intra-Frame Score is defined as follows:

$$S_{int} = NN2 - t$$

For some threshold value t.

By choosing the threshold appropriately, a negative Intra-Frame Score indicates unambiguous features.

4.3 Comparison of Feature Scores

The performance of the SfM classifier of Hartmann et al. is now compared with the two, previously proposed metrics: Next-Frame Score and Intra-Frame Score. For the Next-Frame Score a threshold of $50,000$ was used. Figure 4.1 shows a ROC curve for the classifier of Hartmann et al, classification with Next-Frame Score and classification with Intra-Frame Score. The same ground-truth data is used as by Hartmann et al. To get this, the features are separated into matchable (positive) and non-matchable (negative) features. A feature is labeled positive, if there is a feature match according to definition 3.3 within the 13 preceding and 13 subsequent images. Having this matching ground truth, the classification results of S_{nxt}, S_{int} and the method of Hartmann et al. are computed. True-Positive TP, False-Positive FP, True-Negative TN and False-Negative FN classifications are summed up and the True-Positive rate (TPR) as well as the False-Positive (FPR) rate are computed as follows:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{4.1}$$

$$TPR = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{4.2}$$

By thresholding FP, False-Positives and True-Positives are plotted against each other. While the Next-Frame Score only performs slightly better than a ran-

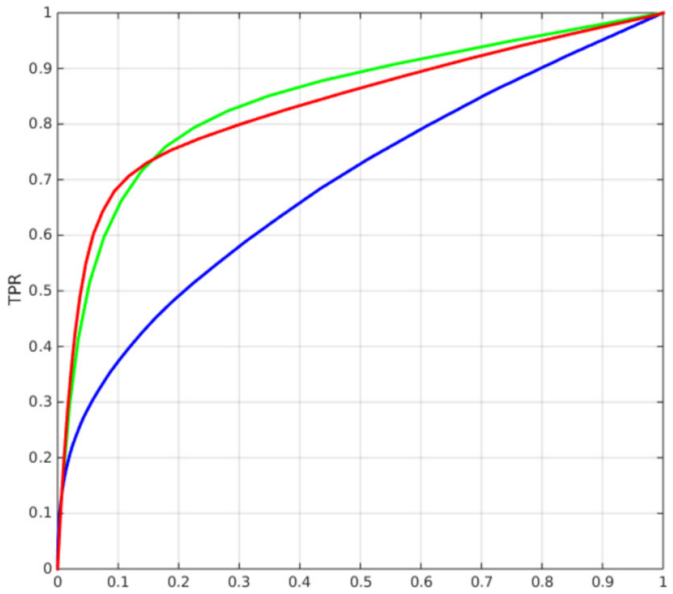

Figure 4.1: ROC curves of [1] (green), Next-Frame Score (blue) and Intra-Frame Score (red). x-axis: False-Positive rate. y-axis: True-Positive rate

dom selection of features, the Intra-Frame Score can be used as a meaningful classifier. Figure 4.2 shows, how the Intra-Frame score selects features. On the

Figure 4.2: Classification of features. Left: All image features. Middle: Classifier of [1]. Right: NN2 score.

left side, there are all sift features found in the image. The image in the middle shows the features picked by the SfM classifier and the right image the features selected with the Intra-Frame Score. As you can see, the SfM classifier as well as Intra-Frame Score selected features appearing on structural elements. Both pick many features on the handrail, but only very little on the grass.
The Intra-Frame metric gives similar results as the classifier of Hartmann et al.

Nevertheless, computing the Intra-Frame score is very time-consuming, due to the $\mathcal{O}(n^2)$ matching complexity. However, as mentioned in section 2.4, using an ANN approach decreases computation time significantly. In table 4.1 the time to create a hash table as well as the time needed to query the hash table and query the SfM classifier is shown:
For feature sets of approximately 18000 or smaller, the classifier is even slower

num features	build hash-table	query hash-table	total hashing	query classifier
2,059	0.015008	0.008311	0.023	0.032
3,316	0.023629	0.007853	0.031	0.051
9,415	0.068592	0.042828	0.111	0.144
17,812	0.125736	0.132377	0.258	0.273
21,226	0.150134	0.257036	0.407	0.316
42,852	0.301039	0.823134	1.124	0.652

Table 4.1: Computation time of ANN and random forest of [1].

than the ANN approach. To prevent the feature set of becoming too large, the DoG threshold for sift can be adjusted, such that approximately 18000 features are extracted. Afterwards, the Intra-Frame Score can be used as a classifier that has the same speed and performance as the classifier of Hartmann et al.
Using the Intra-Frame Score to improve matching for SLAM gives slightly different results. For SLAM, feature matching is done by using a prior pose estimate and then project 3D points into the new frame as discussed in section 2.2.3. Thus, the previously $\mathcal{O}(n^2)$ matching complexity comes down to a linear search of features in a restricted area.
This shows that SLAM does not benefit as much as SfM from the presented feature classifier. Instead, we can simply compute the Intra-Frame score, using the features in a given grid cell and get the same performance as the classifier of Hartmann et al.

Intra-Frame Track-Length

Regarding matchability, the proposed Intra-Frame Score makes the method of Hartmann et al. less important. However, the Intra-Frame Score is not able to pick features, that have a long track length. Figure 4.3 shows a comparison of the classifier of Hartmann et al. and the Intra-Frame Score as well as using no selection method at all. The Track length is considered, so a high amount of features that can be viewed in multiple frames is desired. The Track Length was computed on the PARK dataset from [1]. When using all features, most of them have a small track length. 71% have a track length of just 2, whereas the SfM classifier (red) only picks 60% and the Intra-Frame Score (orange) 66.8%. Consequently, more features are selected which can be viewed in more than two frames by the SfM classifier and the Intra-Frame Score. Nevertheless, the SfM classifier picks more features with a high track length which can be observed,

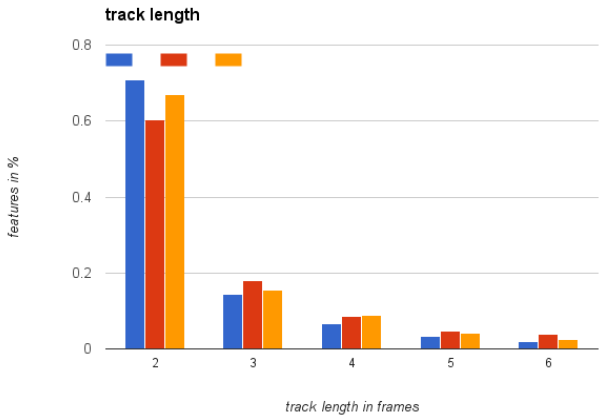

Figure 4.3: Track Length for All features (blue), [1] (red) and Intra-Frame Score (orange).

especially with the help of the most right bars. The number of features (track length 6) selected by the SfM classifier (3.76%) is higher than the amount of using all features (2.06%) or the Intra-Frame Score (2.53%).

To sum up, the Intra-Frame Score gives the same performance as the classifier of Hartmann et al. However the Intra-Frame Score does not prefer features that have a long track length. The main purpose of this work is to reduce the set of features as much as possible by retaining long track features. This is achieved to a certain degree by the classifier of Hartmann et al, but the Intra-Frame Score does not give the desired results. In the following chapter, a classifier that is better suited for this purpose is created.

Chapter 5

Long-Track Features

Using the observations of chapter 4, we finally create the desired feature classifier. We start with our training objective in section 5.1 by using features that have a long track length. Finally, the training of the classifier and the generation of the training data are are explained.

5.1 Track Length Criterion

As shown in chapter 4 the classifier of Hartmann et al. was only optimized to improve the matching procedure. This is not sufficient for SLAM, thus, we are going a step further and optimize our classifier to select only features, that contribute to camera tracking. We distinguish the following sets of features:

Definition 5.1.

1. *All: All available sift features.*
2. *Matchable: Matchable features according to definition 3.3*
3. *Geo: features that belong to a valid geometric reconstruction*
4. *$Track_{\geq n}$: features with a track length lager equal n*
5. *$Track_{max}$: maximal track length.*

The *Matchable* features are the ones, for which a sift match can be found. As shown in figure 2.7, having a sift match does not imply a geometrically valid 3D point. These are contained in *Geo*. Besides a valid underlying 3D model, *Geo* assumes that all images of a sequence can be registered with respect to the reconstruction. Creating a 3D model out of a set of images, for each image a position and orientation has to be computed. While *Geo* already uses a valid 3D model of a scene, the set of features needed for camera tracking can be

restricted even further. $Track_{\geq n}$ contains all features that belong to a valid reconstruction as well as have a track length larger or equal to n. Thus, they can be found in at least n consecutive frames of an image sequence and belong to a valid geometrically 3D reconstruction.
This leads to the following inclusions:

$$All \supseteq Matchable \supseteq Geo = Track_1 \supseteq Track_{n>1} \tag{5.1}$$

$Track_{max}$ are the features having a maximal track length, but will still give a valid reconstruction. For example, having 3 images I_1, I_2, I_3 of the same scene, but only I_1 and I_2 as well as I_2 and I_3 share the same scene view. This gives tracks of at most $t = 2$, due to no feature can be found in all 3 images. Thus, $Track_{max} = Track_{\geq 2}$. Figures 5.1 and 5.2 show the difference of *Geo* and $Track_{\geq n}$. In figure 5.1 the points are so dense, that we can not distinguish

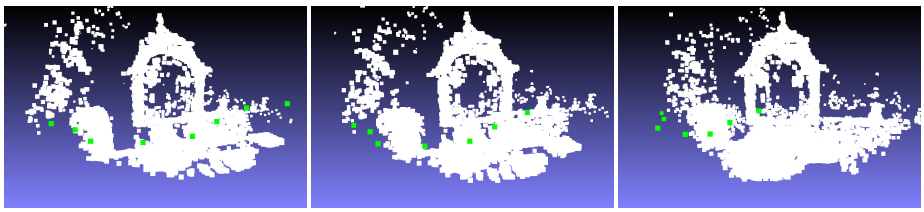

Figure 5.1: *Geo* features. All features that belong to a valid 3D reconstruction.

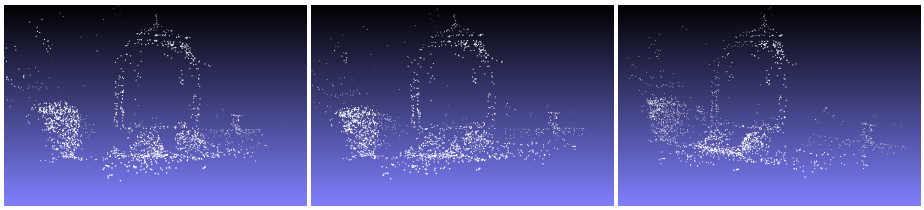

Figure 5.2: $Track_{\geq n}$ features. features that belong to a 3D reconstruction and have a track length greater than 3.

them and mainly see huge white areas. In figure 5.2, on the other hand, you have to look really close to recognize single points.
Depending on the scene, $Track_{max}$ differs. Especially image sequences captured at high frame rate result in a high track length. On the other side, sequences captured with fast camera movements give shorter track length. Thus, we have to find a good trade of for $Track_{max}$:

$$Track_{\geq 1} \supseteq Track_{\geq i} \supseteq Track_{max} \supseteq Track_{\geq k} \tag{5.2}$$

To improve camera tracking, we use $Track_{max}$ to train our classifier. The set $Matchable$ is a superset of the feature set Hartmann et al. used to train their SfM classifier:

$$Matchable \supseteq SfMClassifier \supseteq Geo$$

The features obtained by increasing the sift threshold do not fit into equation 5.1. By changing the parameters, you can always find a set of the same size as the sets listed in equation 5.1. As already shown in [1], Matchable features outperform features chosen with high sift thresholds. However, this has not to be true for camera tracking. Camera tracking still works with only a few features, while this is not sufficient to obtain a dense reconstruction for SfM. The results of features using a high sift thresholds compared to a classifier optimized for long track features can be found in section 7.5.
In [24] khan et al. also perform feature reduction. However, they concentrate on image-based localization. Their main objective was to improve bag of words for large scale image datasets. Similarly as Hartmann et al, they selected features with long 2D track length, thus features for which definition 3.3 holds several times. Nevertheless, they only work on the sift-matching level and do not take the geometry of the underlying scene into account.
Finally, to train our SLAM classifier we select features that have a maximal track length, but still can be used to obtain a 3D reconstruction in which each image can be registered.

Training Objective 5.1 (Track Length).
The features, used for training $feat_{training}$ are selected as follows:

$$Geo \supset feat_{training} \supseteq Track_{max}$$

Hint: no strict inclusion for Geo.

5.2 Training of the SLAM Classifier

We take the PARK dataset of [1] for training. The original dataset contains 121 images with a resolution of 3024×2016. The images are captured with a DSLR camera and most of the images are take with a wide baseline. Thus, the distance of two consecutive images is approximately more than one meter. Due to, some parts of the image sequence are challenging to create a single valid reconstruction, we decided to select three subsets for which we can find $Track_{max}$ with $Track_{max} \subset Track_{\geq 1}$. Figure 5.3 shows three consecutive images from PARK. While the first two images can easily be matched and give a valid reconstruction, the third image suffers from large illumination changes. Registering it with second image still works, but finding geometrically valid matches to the first image does not work. Thus we can not obtain a valid 3D reconstruction that gives camera poses for all images and uses tracks lager than two. Consequently, we could only train a classifier with $Geo \supseteq feat_{training}$ and being less restrictive as the desired training objective 5.1. By picking 3 subsets we get a strict inclusion for our training features and can optimize for obtaining a long track length.

Figure 5.3: Three consecutive images from PARK dataset [1]. Features with a track length of 3 can not be found.

For training, the features need to be classified as positive or negative. We define two disjoint sets of positive and negative sift features according to definition 5.2.

Definition 5.2 (Posivite & Negative Features)**.**
The set of All features can be partitioned into positive and negative features. The set of positive labeled features is given by:

$$feat_{pos} = Track_{\geq n}$$

The set of negative labeled features is given by:

$$feat_{neg} = All \setminus feat_{pos}$$

We want to point out that the definition of positive and negative features is only valid with the context of this thesis. There is no overall definition of a positive or negative feature.
The amount of features and images used by the different subsequences are shown in table 5.1. The last column shows the track length which was used for a certain

	sequence	images	features	track length
	sequ 1	17	36,506	2
	sequ 2	29	25,083	3
	sequ 3	9	9,162	2
sum		**55**	**70,751**	

Table 5.1: Used images & features for training.

subsequence. In particular, for subsequence 2 a higher track length was used. Due to the wide baseline of the images, features that are visible from very different viewpoints are preserved.
The SLAM classifier is trained with $70,751$ positive as well as negative labeled features. Due to, $Track_{\geq n}$ is in general much smaller than All, the set of positive features is much smaller than the set of negative features.

$$feat_{neg} \gg feat_{pos} \tag{5.3}$$

For this reason we use a random selection of negative features to prevent an imbalanced classifier. Additionally, we can distinguish two different subsets of negative features as defined as followed:

Definition 5.3 (Non-Matchable & Short-Track Features)**.**
The set of negative labeled features $feat_{neg}$ can be split into two disjoint sets $feat_{nomatch}$ and $feat_{short}$:

$$feat_{neg} \supseteq feat_{nomatch} \supset Geo$$

and:

$$feat_{short} = feat_{neg} \setminus feat_{nomatch}$$

This gives negative features $feat_{nomatch}$ which can not be matched at all and features that have valid sift matches but do not fulfill the minimum track length criterion of the training objective $feat_{short}$. For these two sets have different cardinalities, similar to $feat_{pos}$ and $feat_{neg}$ in equation 5.3. The size of $feat_{short}$ is much smaller than $feat_{nomatch}$:

$$feat_{nomatch} \gg feat_{short} \tag{5.4}$$

To not only construct $feat_{neg}$ out of features taken from $feat_{nomatch}$, we randomly pick the same amount features for both sets.
Hartmann et al. train a random forest for their SfM classifier. To prevent adding unnecessary complexity we follow their idea and also train a random forest. We use a depth of 25 and 25 trees. The tools used to obtain the training features with their labels as well as the resulting random forest are presented in chapter 6.

Chapter 6

Used Software and Tools

This chapter gives a short overview of the tools used within the scope of this thesis. These were used to create the training data as discussed in the previous chapter as well as evaluation.

6.1 VL Feat

VLFeat [25] is used to extract sift features from images. VLFeat is an open source library which implements popular computer vision algorithms, especially for feature extraction and feature matching. For example there are implementations for HOG or MSER features [26][27] or KD-Trees [28] to speed up feature matching. The main advantage of using VLFeat is the fact that, Hartmann et all provide a random forest implementation included into VLFeat. Thus, the feature classification process is encapsulated into the feature extraction step, but also can be switched off. VLFeat builds the first building block of the processing pipeline for both, creating test data as well as training data.
For creating test data we enable classification and give the path of a random forest. Finally, VLFeat produces text files, containing the positive classified features. For each single images, a file is created, which contains all features within the image. These files can be handed over to Theia, presented in section 6.3, to build a 3D reconstruction with registered cameras.
To get training data, we switch off classification and use all feature which can be found by VLFeat. Again, we get text files, containing the image features. These are handed over to Bundler as well as Theia for further processing.

6.2 Bundler

Bundler [4] is a structure from motion software, which creates 3D reconstructions out of an unordered collection of images from the same scene. Hartmann et al. use Bundler to check the 3D reconstruction by using their classifier as well as a high sift threshold.

By taking feature files from VLFeat, as discussed in section 6.1, Bundler compares each image against all other images and stores valid feature matches. We adjusted Bundler, such that it produces two text files with image descriptors. The *matching text file* contains all descriptors that have a matching partner in another image, the *non matching text file* contains all descriptors for which no match could be found. By doing so, we obtain the feature set $feat_{nomatch}$ as described in definition 5.3.

6.3 Theia SfM

Theia [29] is a structure from motion Library which is used in this thesis to create 3D reconstructions with registered cameras. Furthermore, Theia is used to compare the reconstructions of different methods (high DoG, Hartmann et al. and ours) against each other. Thus, Theia provides statistics to evaluate how many features can be reduces for each method and how accurate the resulting camera poses are. Similar as in the work of Hartmann et al. a reconstruction is creating, using all features. This reconstruction is used as ground truth and compared with a significantly reduced feature set, created by the high DoG method, the classifier of Hartmann et al. as well as our method.
Theia is the second building block of our processing pipeline. It reads the feature files created by VLFeat, matches all the features of different feature files and uses the obtained matches to create a 3D reconstruction. To read in text files, containing image features, we had to slightly adjust Theia.
Besides creating reconstructions and compare their qualities, Theia is also used to create the feature sets $feat_{short}$, as in definition 5.3 and $feat_{pos}$, as in definition 5.2. We adjusted Theia, such that the feature descriptors of both sets are stored in a single text file, respectively.
Consequently, we use a mix of Bundler and Theia to produce our training data. The negative labeled training features $feat_{neg}$ are crated with Bundler and Theia: $feat_{neg} = feat_{nomatch} \cup feat_{short}$. Theia returns the set of all positive labeled features $feat_{pos}$.

6.4 Random Forest Library

To perform feature classification as well as creating a classifier, the random forest classifier template library (rf-library) by Stefan Walk [30] is used. To simplify feature classification after feature extraction, Hartmann et al. implemented the random forest code into VLFeat, as discussed in section 6.1. To train a classifier, the rf-library needs two text files. One includes all positive labeled feature descriptors, the other one contains all negative labeled feature descriptors. In our project, these files are created with Bundler and Theia as described in sections 6.3 and 6.2. Finally, the rf-library creates random forest as binary file. To classify features, this can be used by VLFeat.

6.5 Processing Pipeline Overview

The processing steps of creating a 3D reconstruction is shown in figure 6.1. Starting with input images, VLFeat extracts features and produces feature files. Depending on whether the classification was enabled or not, the file content differs. Afterwards, the files are handed over to Theia which creates a 3D reconstruction. A reconstruction is considered valid, if there is a camera pose for each input image. The quality of the reconstruction is used to check the quality of a given classification method.

Training of a random forest is illustrated in figure 6.2. The feature files are

Figure 6.1: 3D reconstruction with classification.

create as in the previous case, but without classification enabled. The final feature files are passed to Bundler and Theia which create the positive and negative labeled features. The corresponding feature descriptors are used by the random forest library to generate the final classifier.

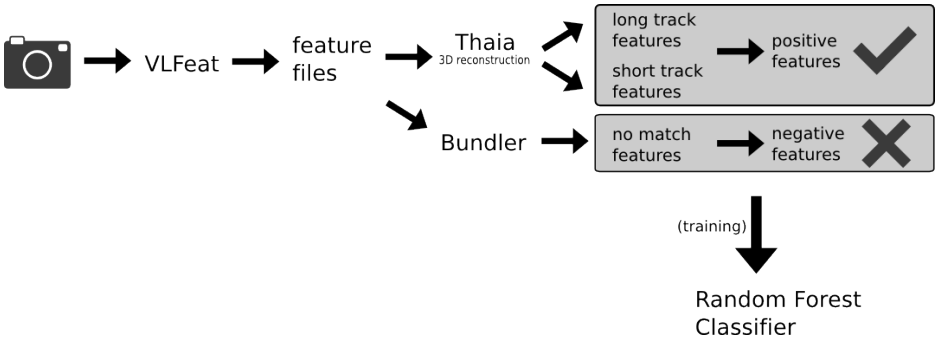

Figure 6.2: Training

Chapter 7

Evaluation and Results

The improvements for SLAM and SfM, achieved by our classifier are presented in this chapter. The datasets and baseline methods which are used to to evaluate our method are illustrated as well as the contributions of our approach. We investigate the track length of the selected features and show that we can filter our redundant image features that do not contribute to estimating the camera pose. This is demonstrated by reducing the set of features selected by our method as well as the baseline methods to a minimum and examining the resulting camera accuracy. Additionally, we measure the saving of computation time when using a lower amount of features as selected by our classifier.

7.1 Datasets

We are using two example scenes of the multi-view dataset denseMVS [31] to evaluate our classifier. The first one, shown in figure 7.1, pictures the front of a church. The second one, presented in figure 7.2, shows a fountain from multiple viewpoints. We chose these two datasets, due to they have many overlapping

Figure 7.1: Church scene of the test dataset [31].

images. This mimics the SLAM scenario, in which a scene is discovered step-by-step. In a SfM scenario on the other hand, it is common to have an unordered image collection of a scene. The second reason why we chose this dataset is the fact that it contains high-resolution images. Thus, we do not limit ourselves to

Figure 7.2: Fountain scene of the test dataset [31].

use outdated hardware using low resolution cameras.
The dataset characteristics are as follows:

1. Canon D60 digital camera
2. images: 25
3. resolution: 3072×2048
4. avg Features/Frame: $93,641$ (sift features)

7.2 Baseline Methods

We compare our classifier against the one created by Hartmann et al. Furthermore, we compare against a feature selection created by increasing the sift threshold. To check the quality of a reconstruction we create a reference reconstruction, which uses no feature selection at all. This gives us 3 test reconstructions and one ground truth reconstruction.
We check, how much we can reduce the amount of features given a certain method. Furthermore we check the quality of a reconstruction by picking approximately the same amount of features for each method. However, this was not always possible, due to the final amount of 3D points produced by Theia is not predictable.
The following abbreviation is used for the different methods.

- **no cl**: no classifier was used
- **har-cl**: Feature selection with the classifier of Hartmann et al.
- **hDoG**: Feature selection with high threshold for the DoG response.
- **track-cl**: Our classifier that preserves long track features.

In section 4.3, we compared the Intra-Frame Score with the random forest of Hartmann et al. and showed similar quality with respect to matchability. Anyhow, we showed that the random forest preserves features with a long track length better. For this reason, we do not consider the Intra-Frame Score in the following tests, because there is already one counter example.

7.3 Track length Comparisons

After computing a 3D reconstruction, Theia provides statistics about the amount of features, with a certain track length. These are used to create track length histograms for both test scenes. The track length is shown at the x-axis and the amount of features having a certain track length at the y-axis. We do not count, if a feature can be matched in several other images. Instead we sum the views a 3D world point can be reprojected to.
Figure 7.4 shows that our classifier picks less features with a track length of 2, but starts having more features for longer tracks. For a track length of 3, both classifier perform nearly the same. Starting with a track length of 4 our classifier demonstrates its benefits and a significant difference can be observed. Figure

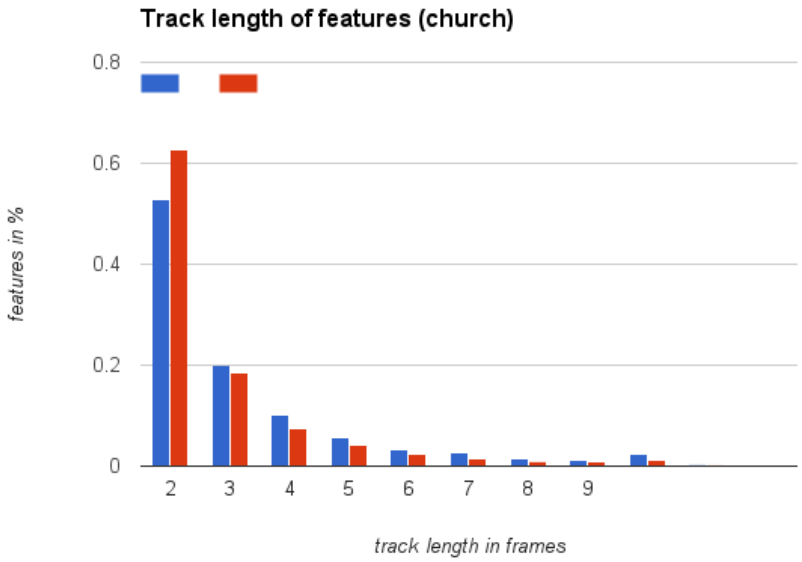

Figure 7.3: Track Length for our classifier (blue), as well as [1] (red).

7.4 shows similar results. For the histogram bars for track length greater than 5 our classifier selects clearly more features. For tracks smaller than 5 har-cl picks slightly more features. By having a look at the image sequence (appendix) we can observe that the perspective changes from one image to another a much smaller than for the church scene. This leads to more features with a high track length and explains the even better performance of our classifier for this scene.
We excluded the statistics for the hDog method here. A high DoG response leads to features in high frequent areas as shown in figure 7.8. In general such areas are robust against viewpoint changes and therefore will produce features with long tracks. However the amount of areas that give a high DoG response

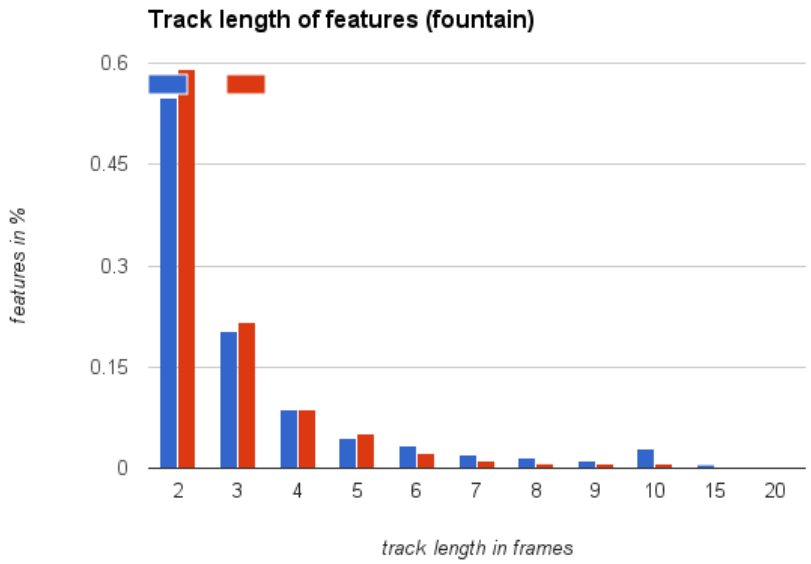

Figure 7.4: Track Length for our classifier (blue), as well as [1] (red).

are very limited for most scenes such that features can only be reduced to a certain amount. This is explained in more detail in section section 7.4.2, where a reconstruction using the hDoG method is not possible anymore, but a feature reduction using our approach leads to nearly 50% less features.

7.4 Feature Reduction

7.4.1 Feature Reduction per Frame

Table 7.1 shows how much we can reduce the amount of features, using our classifier. The first column shows the sum of all features, extracted in each image of the sequence. The second column shows the average amount of features per frame. Column three lists the amount of 3D points of the resulting 3D reconstruction and the last column shows the mean reprojection error in pixel. After creating a reconstruction, the obtained 3D points can be reprojected to the registered views. By comparing these reprojections with the original sift-feature positions, we can get a quality measure of the computed 3D points as well as the computed camera poses. In both tables, we clearly see the advantage of the SFM classifier. For the church scene, it selects less than the half of the features in average as the high DoG method. For the fountain scene the classifier nearly

Church scene				
	feature amount	avg features	3D points	mean repr. error
all features	2136475	85459	150893	0.445066
high DoG	67543	2701	4624	0.416873
Hartmann et. al	139986	5599	9129	0.524981
Our Classifier	31331	1253	2836	0.351269
Fountain scene				
	feature amount	avg features	3D points	mean repr. error
all features	2545581	101823	221311	0.735637
high DoG	50686	2027	4596	0.746021
Hartmann et. al	39903	1596	2454	0.842059
Our Classifier	20379	815	1691	0.574793

Table 7.1: features per frame and mean reprojection error.

performs 50% better than the approach of Hartmann et al. Additionally in both cases, the SfM classifier leads to a lower reprojection error, thus gives a better reconstruction quality with approximately half the features. It is also worth to mention, that for the church scene, the classifier of Hartmann et al. outperforms the high DoG method, while it is the other way around, for the fountain scene. The table also shows the amount of 3D points, after the reconstructions finishes. Again, around 50% of 3D points are sufficient for our classifier to achieve a reconstruction with a lower reprojection error.
In contrast to using all features, all three methods reduce the amount of data significantly. For the church dataset, our classifier only picks 0.015% of the available features. For the fountain dataset, a reduction to even 0.008% is achieved.
Curiously, when using all features, the reprojection error is lower. This is due to the high amount of outliers, which is disdussed in more detain in sectino 7.4.3.

7.4.2 Maximum Feature Reduction

To demonstrate the limits of the different feature reduction methods, thresholds are increased until Theia was not able to register all cameras anymore. Only for our classifier the threshold remains below this limit, so all 25 images are assigned a position and rotation within the 3D model. Table 7.2 shows that our classifier creates a 3D model with less 3D points than the classifier of Hartmann et al. as well as an increased DoG. While our method is still able to register all cameras, the baseline methods are not. Additionally, our approach selects less 2D feature points per frame, in average. Thus, less data is handed over to the subsequent SfM pipeline, but the 3D reconstruction is still better.

Church scene			
	3D points	avg. features	reg. cameras
high DoG	4043	2397	24
Hartmann et. al	8220	5188	24
Our Classifier	2836	1253	**25**
Fountain scene			
	3D points	avg. features	reg. cameras
high DoG	3154	1471	22
Hartmann et. al	1864	1334	24
Our Classifier	1691	815	**25**

Table 7.2: Limits of baseline methods. Our methods can register all cameras, while the high DoG method and [1] register fewer cameras.

7.4.3 Point Clouds

Figure 7.5 shows the different point clouds for the church scene 7.1, after running reconstruction with Theia. When using all features, a lot of redundant informa-

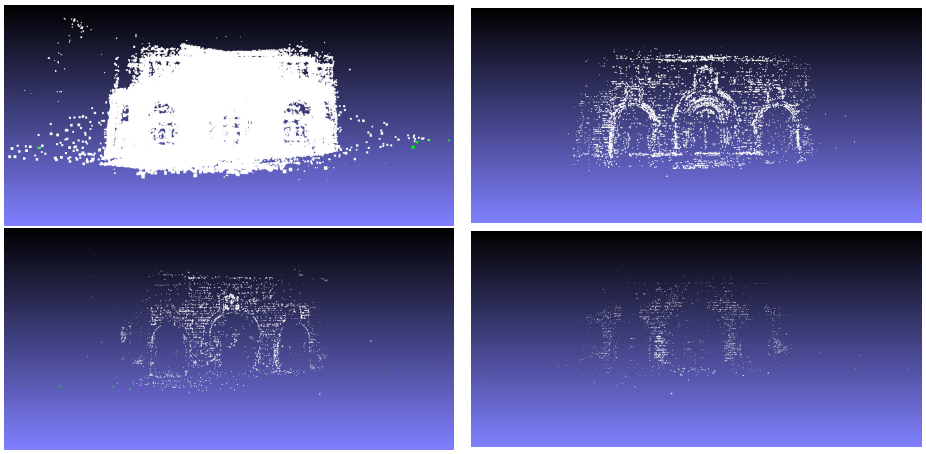

Figure 7.5: Surviving features for the different methods. Top left: all features. Top right: SfM classifier[1]. Bottom left: high DoG. Bottom right: our classifier.

tion remains in the final reconstruction, which is obviously not ideal for camera tracking. For the classifier of Hartmann et al, the structure of the church is well recognizable, but compared to the other two approaches the amount features is still very high. While the point cloud for the high DoG approach is even more spare, the one created our classifier is barley recognizable with the human eye. The exact amount of 3D points can be looked up in section 7.4.1. This section also shows a higher quality of the 3D model using our approach with respect to the mean reprojection error.

Even better observations can be made for the second dataset, pictured in figure 7.6. Using all features results in a dense point cloud having much redundant

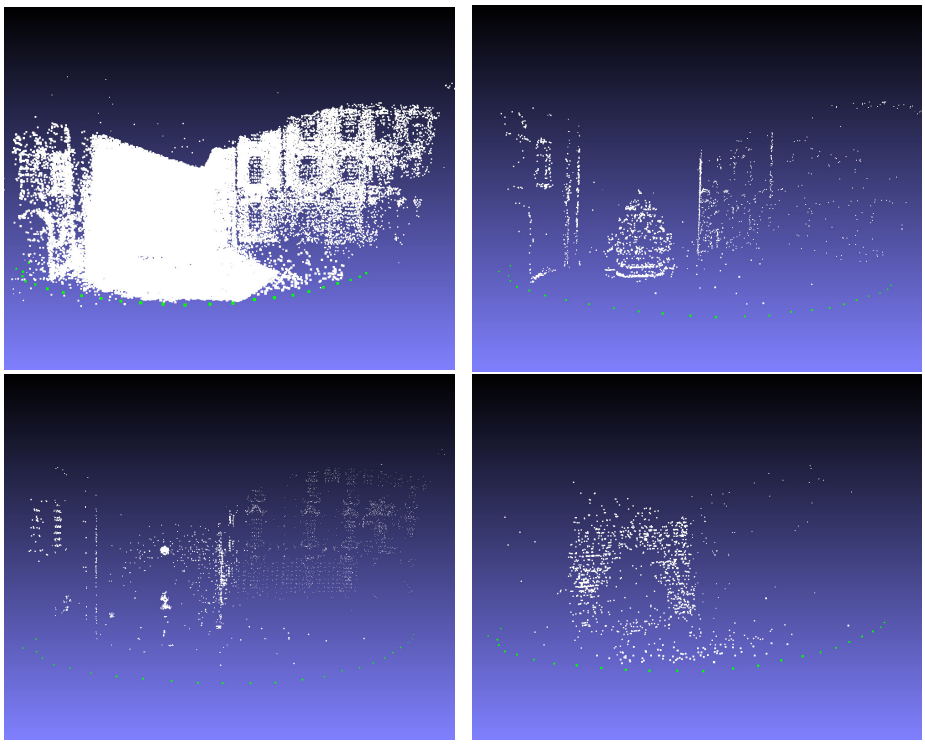

Figure 7.6: Surviving features for fountain scene. Top left: all features. Top right: SfM classifier[1]. Bottom left: high DoG. Bottom right: our classifier.

information. Although the classifier of Hartmann et al. and the approach using a high DoG have more features spread over the whole scene, these are not the features with the highest track length. Observing the whole fountain scene (appendix) shows the visibility of the fountain and the wall behind in every image. The surrounding buildings on the other hand are only exposed in the latest images of the sequence. For this reason, a feature reduction that prefers features around the fountain is able to register all camera poses. When reducing features such that most features belong to the buildings in the background, it is likely that such an image can not be fitted into the remaining scene. This demonstrates the tendency of our classifier to select features that can viewed in as many images as possible.

7.4.4 Outlier Reduction

Figure 7.7 shows the reconstructions for all 4 methods from a side view (church scene). What stands out, is the high amount of features, which seem to fly in the air, especially when using all features. Inspecting the whole image sequence (appendix), these features are clearly outliers and should not remain in a reconstruction. When using one of the three feature reduction methods, the amount of outliers is reduced. However we can not provide a quantitative measurement at this point, because you have to manually filter out and decide which points are outliers. To perform camera pose estimation as accurate as

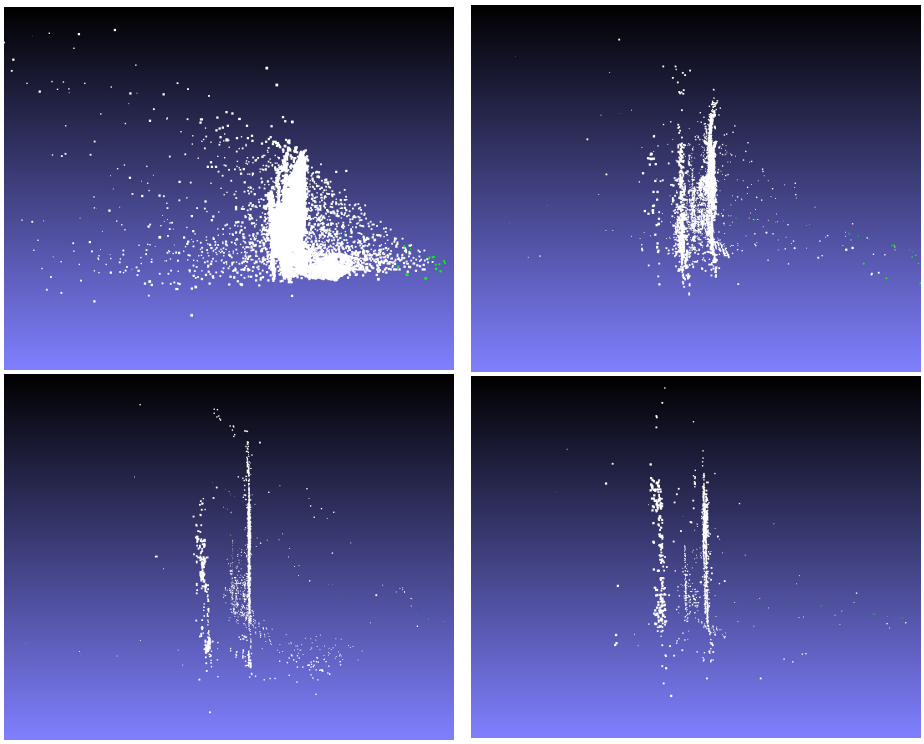

Figure 7.7: Side view of the reconstruction with surviving features. Top left: all features. Top right: SfM classifier[1]. Bottom left: high DoG. Bottom right: our classifier.

possible, a detailed 3D model is needed. The mentioned kind of outliers perturb the 3D model and consequently tracking accuracy decreases. The outliers can not be removed by the outlier detection used by Theia. For camera tracking applications such errors accumulate and will finally make tracking unfeasible. This emphasizes the need of feature reduction methods. While figure 7.7 shows fewer outlier for our method this is clearly an advantage for most SLAM and SfM applications.

7.4.5 Feature Distribution

After reducing features to a minimum, the distribution of features is investigated. This is show by figure 7.8 for the different methods. The first row shows

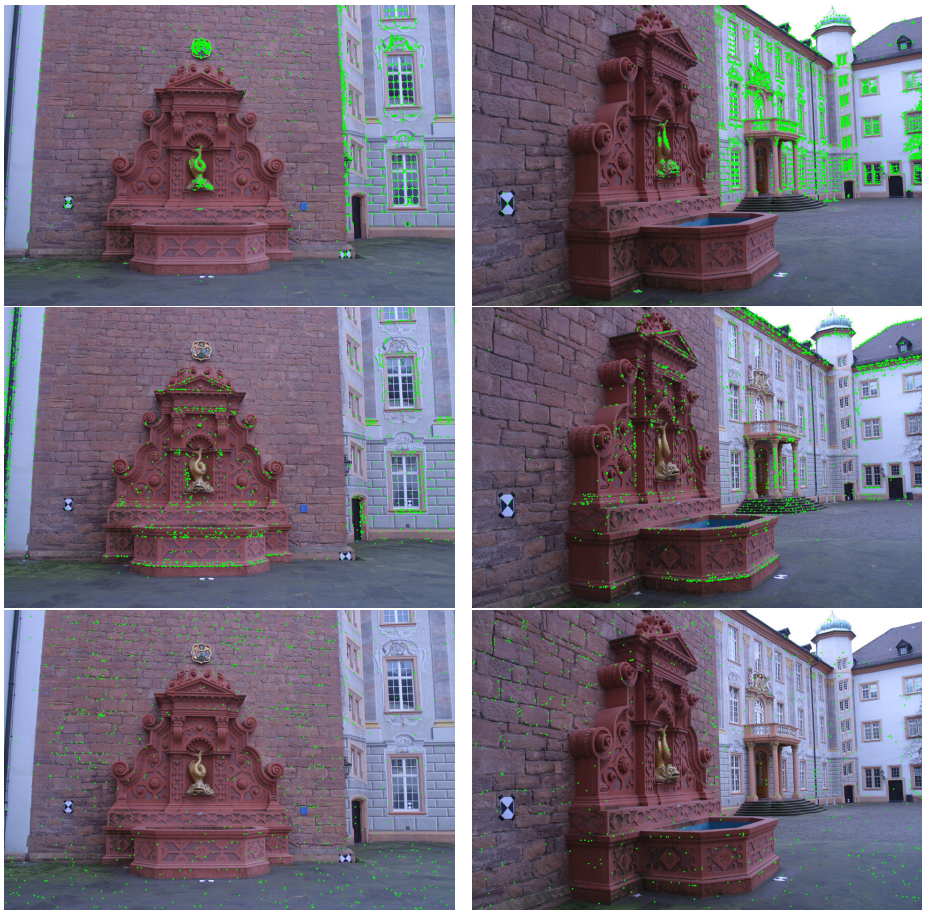

Figure 7.8: Feature distribution when reducing amount of feature. Top: high DoG with 1945 (left) and 5074 (right) features. Middle: har-cl with 1566 (left) and 1867 (right) features. Bottom: Our classifier with 824 (left) and 570 (right) features.

the result for the high DoG method which clearly selects features in areas where you can find many structural elements like the parts of the buildings. As already shown for the point clouds in section 7.4.3, features that belong to the buildings do not contribute best for camera tracking. The fountain and the wall behind it are visible in all images of the sequence (appendix), while the buildings are not. Thus, reducing features to a minimum should select features around the fountain. In particular, in the right image of the high DoG method, reducing

features will exclusively pick features that belong to the buildings. These features can not be registered with the rest of the scene and for this reason there no camera pose can be estimated.
The better distribution of features, such that they have a high track length leads to less features for our classifier. In the first view, our approach achieves a reduction of 57% compared to the high DoG method and 47% compared to the classifier of Hartmann et al, the second view results in a reduction of 88% and 69% respectively.

7.5 Camera Pose Accuracy

The camera accuracy is measured by comparing a reconstruction with reduced features to the reconstruction which uses all available features. Thus, using all features is considered as ground truth. The reduced reconstruction is aligned to the ground truth reconstruction with a similarity transformation. Afterwards, for all cameras pairs of both reconstructions, the error of its position and rotation is measured and an average value computed. The results are shown in table 7.3. Our classifier results in less 3D points, but still achieves a lower position as well

Fountain scene			
	3D points	mean position error	mean rotation error
high DoG	4624	0.005991	0.004881
Hartmann et. al	9129	0.006487	0.002644
Our Classifier	3586	0.005087	0.001826
	3D points	mean position error	mean rotation error
Fountain scene			
high DoG	4596	0.033531	0.009252
Hartmann et. al	2454	0.034065	0.013414
Our Classifier	1691	0.025814	0.002521

Table 7.3: Camera rotation & position errors. Upper table: Church scene. Lower table: Fountain scene.

as rotation error in both scenes. This shows that the features that are chosen by our method are more valuable for obtaining the pose of a camera.
The reconstructions are generated without including any information about the camera poses, thus the reconstructions have no fixed scale. This also means that the rotation and position errors are relative to some scaling factor. Nevertheless, this does not change the accuracy of our method with respect to the high DoG approach or the classifier of Hartmann et al.

7.6 Timings

To quantify the speed-up of our classifier, the average feature matching time, as well as the time needed to perform bundle adjustment was investigated. Theia was executed with two frames only. A full bundle adjustment of several frames would even take longer. Furthermore, a larger scene with more 3D world points influences bundle adjustment even more.
Systems like ORB-SLAM [6] use bundle adjustment in each tracking step to get the pose of the camera. However, they only use a small part of the scene. By sticking with only two frames, we limit the scene to a minimum but still can detect improvements by our method. The results are listed in table . Due to the

	features	feature matching	bundle adjustment
all features	85459	3.477	0.146
Hartmann et. al	5263	0.0695	0.0162
Our Classifier	2107	0.0239	0.011

Table 7.4: Average timings for church scene.

fewer features selected by our classifier we can gain a bundle adjustment speed-up of 67% compared to the classifier of Hartmann et al. Different than shown in table 7.1, the threshold for our random forest was chosen less restrictive, because only two frames are considered and a reconstruction with fewer features was not possible. For this reason 2107 features were used rather than 1253 which is the average for our classifier for the church scene. However, even with less restricting settings, our classifier still outperforms the classifier of Hartmann et al.
When using a SLAM setup as used in ORB-SLAM, where bundle adjustment is performed for every frame and real time bounds have to be considered, an decrease in computation time from $0.0162s$ to $0.011s$ (67%) gives a significant speed-up.

Part III

Conclusions and Outlook

Chapter 8

Future Work

This thesis shows the advantages of using a classifier that preserves long track image features to reduce the amount of data a SfM or SLAM pipeline has to process. While the amount of features can be lowered significantly, there are several concept to improve the presented method.
During the training phase, the features are labeled positive, if they can be viewed in a minimal amount of views and they belong to a valid 3D reconstruction. For future research one can go a step further and assign different weights to the different feature classes presented in section 5.1. By doing so, a feature that is not matchable at all penalizes the result more than features that belong to a valid reconstruction but do not fulfill the minimum track length criterion.
Another penalization strategy can involve different weights for specular areas. Such areas are sensitive for most feature detectors, but depend strongly on the viewing angle. Thus, they should no be considered by reconstruction as used in most SfM or SLAM approaches.
As illustrated in figure 7.7, the final reconstructions still include outliers which are not removed by the outlier detection. By manually labeling such features followed by learning a classifier we can try to get rid of these kind of outliers. However, this includes a lot of manually interactions.

In the used SfM framework for evaluation, feature extraction and matching takes a lot of time. Although the proposed classifier speeds-up these steps significantly a further improvemnt can be achieved by using the GPU implementation of sift [32]. In particular the slow matching process for sift features is the main reason why most modern SLAM systems prefer binary features. Using our classifier as well as GPU-Sift could lead to a accurate real-time SLAM system which makes use of the accuracy of sift. Alternatively, an approach as presented in [33] can be used to speed up feature matching.

Of course, all assumptions made in section 5.1 are not restricted to sift features only. By training a random forest for binary features like BRISK or FAST, the ideas can be tested for features that can be matched much faster.

In [34] and [35] SLAM concepts are presented which also work in partly dynamic environments. By separating features that belong to a dynamic part or a static part of the scene, a new classifier can be trained. Afterwards, it can be tested, if features in moving scene areas can be predicted in advance.

This shows the importance of this work. Although the long track classifier was already able to reduce the amount of features significantly, there are plenty of research topics available in this area.

Chapter 9

Conclusion

In this thesis a classifier was created which improves the processing steps of a SLAM or SfM pipeline by decreasing the amount of redundant data for camera pose estimation. Handing over a reduced set of image features speeds-up SfM as well as SLAM without degrading the tracking accuracy. Additionally, the amount of outliers is reduced, such that the resulting 3D model is less perturbed. We ware able to show a feature reduction of nearly 50% in average. Furthermore, a better distribution of image features was achieved, such that camera pose estimation works fewer data. The features that are selected by the created classifier have a long track length, thus can be seen in as many views as possible. The reduction of features also leads to a less computationally SLAM or SfM pipeline.

Part IV

Appendix

Evaluation Scenes

Figure 9.1: fountain scene [31].

Figure 9.2: church scene [31].

List of Figures

List of Tables

Bibliography

[1] W. Hartmann, M. Havlena, and K. Schindler, "Predicting matchability," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 9–16, June 2014.

[2] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99)*, (San Francisco, USA), Oct. 1999.

[3] N. Technologies, "ncam camera tracking."

[4] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3d," in *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, (New York, NY, USA), pp. 835–846, ACM, 2006.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," vol. 60, (Hingham, MA, USA), pp. 91–110, Kluwer Academic Publishers, Nov. 2004.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564–2571, Nov 2011.

[7] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2548–2555, Nov 2011.

[8] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ECCV'06, (Berlin, Heidelberg), pp. 430–443, Springer-Verlag, 2006.

[9] J. Leonard and H. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pp. 1442–1447 vol.3, Nov 1991.

[10] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Editors choice article: Visual slam: Why filter?," *Image Vision Comput.*, vol. 30, pp. 65–77, Feb. 2012.

[11] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, (Nara, Japan), November 2007.

[12] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," ACM Symposium on User Interface Software and Technology, October 2011.

[13] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial odometry using nonlinear optimization," *Int. J. Rob. Res.*, vol. 34, pp. 314–334, Mar. 2015.

[14] H. Stewénius, C. Engels, and D. Nistér, "Recent developments on direct relative orientation," 2006.

[15] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[16] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, pp. 1188–1197, October 2012.

[17] D. Hauagge and N. Snavely, "Image matching using local symmetry features," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 206–213, June 2012.

[18] G. Zhang and P. A. Vela, "Good features to track for visual slam," June 2015.

[19] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, pp. 891–923, Nov. 1998.

[20] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, (San Francisco, CA, USA), pp. 518–529, Morgan Kaufmann Publishers Inc., 1999.

[21] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[22] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984. new edition.

[23] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, pp. 3–55, Jan. 2001.

[24] N. khan, B. McCane, and S. Mills, "Feature set reduction for image matching in large scale environments," in *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, IVCNZ '12, (New York, NY, USA), pp. 67–72, ACM, 2012.

[25] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms." `http://www.vlfeat.org/`, 2008.

[26] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893 vol. 1, June 2005.

[27] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *Proceedings of the British Machine Vision Conference*, pp. 36.1–36.10, BMVA Press, 2002. doi:10.5244/C.16.36.

[28] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, June 2008.

[29] C. Sweeney, *Theia Multiview Geometry Library: Tutorial & Reference*. University of California Santa Barbara.

[30] W. Stefan, "random-forests." `https://github.com/stefan-w/random-forests`, 2012.

[31] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, and U. Thoennessen, "On benchmarking camera calibration and multi-view stereo for high resolution imagery," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, June 2008.

[32] C. Wu, "SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)." `http://cs.unc.edu/~ccwu/siftgpu`, 2007.

[33] F. Alhwarin, D. Ristić-Durrant, and A. Gräser, "Vf-sift: Very fast sift feature matching," in *Proceedings of the 32Nd DAGM Conference on Pattern Recognition*, (Berlin, Heidelberg), pp. 222–231, Springer-Verlag, 2010.

[34] W. Tan, H. Liu, Z. Dong, G. Zhang, and H. Bao, "Robust monocular slam in dynamic environments," in *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pp. 209–218, Oct 2013.

[35] P. Alcantarilla, J. Yebes, J. Almazan, and L. Bergasa, "On combining visual slam and dense scene flow to increase the robustness of localization and mapping in dynamic environments," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1290–1297, May 2012.