Abstract. Accurate patient registration and referencing is a key element in navigated surgery. Unfortunately all existing methods are either invasive or very time consuming. In this semester thesis an extension to the work performed by the BIWI/CV Lab on "Markerless Endoscopic Registration and Referencing" is proposed, by introducing a new method combining tracking and matching methods to get nearly real-time 3D reconstruction. The 3D reconstruction can then be used for matching the pre-operative data to the intra-operative scene. In order to cope with the near real-time requirements for referencing, we use a novel, efficient, object-oriented 3D point management method during 3D model reconstruction.
1 Introduction

Over the past decade computer aided, navigated surgery has evolved from early laboratory experiments to an indispensable tool for many interventions in spine surgery [1]. Accurate patient registration with the pre-operative imaging modalities or referencing of the target anatomical structure is an essential component of these procedures. Many of today’s approaches are invasive and use either mechanical fixation devices, fiducial markers, ionizing radiation or require tedious manual registration with a pointing device. While the potential of ultrasound imaging for non-invasive registration and referencing is currently explored [2], its usability is constrained by the presence of air-filled cavities in the body. This thesis is a followup work of [3]. There, they used a tracked monocular endoscope to register and reference vertebrae during spine surgery. By tracking natural landmarks over multiple views, a 3D reconstruction of the surgical scene can be computed using photogrammetric methods. Their proposed algorithm consecutively refines the 3D model with each additional view. This reconstruction provides quantitative metric data about the target anatomy which can be used for 3D-3D registration of the anatomy to the pre-operative data and for further referencing. Their aim is to create an intra-operative support environment, which relies as much as possible on the tools and instrumentation used anyway during surgery and does not increase the invasiveness of the intervention just for navigation purposes. Their presented method is general and can be used for any rigid anatomical structure with sufficiently textured surfaces. In this thesis, an extension to the previously used reconstruction algorithm is presented. The aim is to allow a nearly real-time reconstruction of the scene allowing the system to be used neatly during surgery.
2 Related Work

Various approaches have been proposed to provide the surgeon with more information during endoscopic interventions. Most of them are tailored to specific procedures and are of limited general applicability. In particular the following application fields can be identified: (a) registration and referencing, (b) navigation aids, and (c) augmented reality.

A fiducial marker based system to register freehand 2D endoscopic images to pre-operative 3D CT models of the brain has been presented in [4], and in [5] Thoranaghatte proposed a markerless referencing method for the spine. The use of markers, however, often forces the surgeon to increase the invasiveness of the intervention just to be able to perform referencing during the surgery. In [6], a hybrid tracking system is proposed using a magnetic tracking sensor and image registration between real and virtual endoscopic images to compute the pose and position of the camera in the CT coordinate frame.

A non-tracked calibrated endoscope for 3D reconstruction and motion estimation from endo-nasal images is used in [7] for registering the CT to the endoscopic video. Another navigation aid using photogrammetry during endoscopic surgery has been studied in [8]. They use the structural information to prevent the endoscope image from flipping upside-down while rotating the camera. In [9] the pose of a specially marked tool inside the surgical scene has been determined from monocular laparoscopic images and used to create 3D renderings from different views for the surgeon.

Augmented reality systems are not necessarily aiming at quantitative measurements but rather want to improve the visual perception for the surgeon by extending the image with additional information. In [10–14] externally tracked cameras are used to augment the surgeon’s view by fusing pre-operative data with the actual endoscopic view. In contrast, [15] uses a stereo endoscope instead of a tracker for 3D-3D registration of the surgical scene with the pre-operative model data. An example for 2D-3D registration is presented in [16], where navigation is simplified by displaying the corresponding pre-operative CT or MRI slice next to the intra-operative endoscopic image.
3 System Overview

The entire hardware setup is depicted in Fig. 1. For our experiments a 10 mm radial distortion corrected endoscope (Richard Wolf GmbH) with an oblique viewing angle of 25° was used. To avoid interlacing artifacts, we relied on a progressive frame color CCD camera with a resolution of 800 × 600 pixels and 15 fps. As the depth-of-field of the endoscope/camera combination is in the range of 3 – 8 cm the focal length of the camera can be kept constant during the entire procedure, allowing to avoid the recalibration of the system during surgery.

A marker is attached to the endoscope that is being followed by the active optical tracker (EasyTrack500, Atracsys LLC). The EasyTrack provides accurate position (less than 0.2 mm error) and orientation information in a working volume of roughly 50 × 50 × 1500 cm$^3$. An external hardware triggering logic ensures the synchronized acquisition of the tracking data and the camera images during dynamic freehand manipulation.

The entire setup is portable and can be installed in a sterile environment within minutes. The intrinsic camera and the extrinsic camera-marker calibration can be performed pre-operatively in one step by the surgeon without requiring assistance from a technician [3]. An example installation is depicted in Fig. 2.
4 3D Reconstruction Algorithm

The proposed 3D Reconstruction Algorithm extends and improves the previous algorithm used by C. Wengert in [3]. Briefly, the reconstruction there worked as follows. For two consecutive images, features are computed using the Fast-Hessian Corner detector. Then, using the SURF Feature Descriptor presented in the same publication [17], corresponding features are matched by an extensive search. After filtering wrong correspondences out by a RANSAC filter, the 3D points are reconstructed using the intrinsic and extrinsic camera parameters. In this existing algorithm, most computational time is spent for three activities: feature detection, feature description and matching. The aim of this semester thesis was to avoid performance pitfalls as much as possible by combining the previously raw matching technique with a variation of feature tracking techniques and improving the database and data structure.

The presented algorithm is divided into three parts. The first two parts manages the initialization, i.e. handle the first two pictures. The third part then represents the actual algorithm with most of the improvements. The single modules will be presented in the next chapter.

Part I For incoming image 1 (Fig. 3):

1. **Detect** image features using the Fast-Hessian Corner Detector.

2. **Describe** all found image features using the SURF Feature Descriptor algorithm.
Part II Initial Matching (Fig. 4). For incoming image 2:

1. **Baseline Test**: The new view should be distinct enough from the first one.

2. **Detect** image features using the Fast-Hessian Corner Detector.

3. **Describe** all found image features using the SURF Feature Descriptor algorithm.

4. Match actual features with the ones from last image using **Epipolar Matching**.

5. Filter out matching pairs, injuring the epipolar constraints using the **RANSAC Filter**.

6. **Reconstruction** of 3D points from legal correspondence pairs.

7. **Reject** 3D points lying not in given bounding box, passing not the backprojection filter or the empty cell test.

Part III Combined Tracking and Matching (Fig. 5). For incoming images $n \geq 3$:

1. **Baseline Test**: The new view should be enough distinct from the old one.

2. **Detect** image features using the Fast-Hessian Corner Detector.

3. **Fast Matching**: For each detected feature in the actual view, find nearest/best reprojected 3D point (and therefore giving a correspondence to features in at least two previous views).
4. For all unmatched features in the actual view (for which no correspondence has been found in 3.), do **Epipolar Matching** with the unmatched features in the previous view.

5. Filter out matching pairs, injuring the epipolar constraints using the **RANSAC Filter**.

6. **Reconstruct** 3D points for matching pairs found by "Epipolar Matching".

7. For 3D points having at least 2 corresponding features, **Bundle Adjustment** is performed, optimizing the 3D position with respect to all views.

8. **Reject** 3D points lying not in given bounding box, passing not the backprojection filter or the empty cell test.

**Remarks:**

As mentioned before, previously the major part of computing time was spent in the computation of the detection and description of the features as well as in the matching of features in two consecutive images (by combining all descriptors of one image against all descriptors of the other). Here, detection is obviously performed in the same manner as before, since all features first have to be found. But after the two initialization stages, the improvements of this algorithm comes into play. By doing first the "FastMatching", a certain
Fig. 5. Starting from the third image, the improved algorithm takes over.

amount of features can already be matched very cheaply (see "Results" for details). Since the complexity of a complete matching lies in $O(mn)$ (where $m,n$ are the number of features in each view), the reduction of $m$ and $n$ by "FastMatching" can decrease the matching time for a naive matching already significantly. In addition, descriptors need not being computed for all features found. Only those descriptors, being compared in "Epipolar Matching", are computed, decreasing the overall number of descriptor computation as well. Moreover, by the use of "Epipolar Matching", not all remaining features have to be compared, only those fulfilling the epipolar constraints, raising again a gain in time, but of course depending on the distribution of the features.

There are two reasons, why just the last two images are considered for matching, and not the last $n$ images are taken into account. The first is a memory consideration. Since for all features in these $n$ images a descriptor computation might be needed, the images and the surrounding datastructure must reside in memory, augmenting the memory requirements (next to the expensive storage of the other essential data like keypoints and 3D points). But this can be afforded. The second and more important reason is again the performance. The cost for a similar algorithm, taking also the last $n$ views into account, would raise the cost to a level not dealable today with the desired real-time requirements. Nevertheless, feature tracking is also performed in the proposed algorithm, since in "EpipolarMatching" pairwise matching may create long lists of correspondences and "FastMatching" has the ability to recover correspondences in even very old views.
5 Details of the 3D Reconstruction Algorithm

In this section, the newly introduced parts of the improved system will be explained in detail. For the other, known techniques, a brief description will be given with references to more profound literature.

5.1 Datastructure

For the representation of the data (views, features, 3D points and relations between them) an object-oriented datastructure has been chosen. This is described in the following:

<table>
<thead>
<tr>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cal : CameraCalibration*</td>
</tr>
<tr>
<td>+ image : tlImage*</td>
</tr>
<tr>
<td>+ features : std::vector&lt;Keypoint*&gt;</td>
</tr>
<tr>
<td>+ hash : SHash&lt;Keypoint*, 2&gt;*</td>
</tr>
<tr>
<td>+ detdesc : DetDesc *</td>
</tr>
</tbody>
</table>

Fig. 6. The View object.

The View object is responsible for managing the data of one view i.e. of one image (Fig. 6). Therefore it holds the calibration of the current camera (intrinsic and extrinsic parameters) and of course the image itself. Then also an array holding all found features in the image, next to a Spatial Hash Table, holding pointers to the features (and allowing fast location-dependent lookup used by ”FastMatching”). Moreover it has an object holding the current feature detector/descriptor object, responsible for the detection and description of features in the current image and which could be replaced by others than the used Fast-Hessian/SURF combination.

As already seen in the View object, features are described by an Keypoint object (Fig. 7). A Keypoint object holds all information concerning a feature. These are: The location of the feature in the image (which is distorted, due to lens effects), its undistorted\(^1\) and its normalized\(^2\) location (which are computed and cached during the algorithm) and then of course the feature descriptor array. These are the stored information concerning the feature. The relations between features and 3D points are stored as well in each object. Each Keypoint

\(^1\) The acquisition of images usually brings in a slight distortion of the images, due to lens effects. These can be undone by an undistortion step.

\(^2\) The undistorted coordinate corresponds to the ideal projection of a 3D point into image space \((x_u = K[R[t]|x])\). The normalized coordinates here correspond to the projection of the 3D point, but without the transformation into the images space, i.e. \(x_n = [R[t]|x]\)
The Keypoint object.

<table>
<thead>
<tr>
<th>Keypoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ loc : matVector</td>
</tr>
<tr>
<td>+ uloc : matVector</td>
</tr>
<tr>
<td>+ nloc : matVector</td>
</tr>
<tr>
<td>+ featdesc : std::vector&lt;double&gt;</td>
</tr>
<tr>
<td>+ regdesc : std::vector&lt;double&gt;</td>
</tr>
<tr>
<td>+ lmatch : Keypoint*</td>
</tr>
<tr>
<td>+ rmatch : Keypoint*</td>
</tr>
<tr>
<td>+ p3d : Point3d*</td>
</tr>
<tr>
<td>+ parent : View*</td>
</tr>
</tbody>
</table>

Fig. 7. The Keypoint object.

contains a pointer to a matched Keypoint in a previous image and a pointer to a Keypoint in a later image (if any exist). That way, a doubled linked list between corresponding features is maintained. If a 3D point has been reconstructed from features, there is also a pointer to this 3D point stored in the Keypoint object. Finally, there is also a pointer to the View object, in which the actual feature resides.

The Point3d object holds the information for a reconstructed 3D point (Fig. 8). It contains the position of the 3D point and an array of pointers to the features, from which it has been reconstructed. Additionally, it maintains a flag describing the state of the point (valid or invalid due to the filtering).

There exist two containers for the View objects and the Point3d objects, which are called ViewDB and PointDB, simply managing the growing number of objects. These are embedded in the Control Object (here called ReconstructionManager), implementing the control structure of the whole algorithm.

5.2 FastMatching

As mentioned briefly in the algorithm overview, the FastMatching part consists of two steps. First, all reconstructed 3D points are projected into the current
The projection of the 3D points into the current view is very simple. Let $X$ be the current 3D point, $R$ the rotation matrix and $t$ the translation vector of the external calibration and $K$ the internal calibration matrix. Then, the undistorted image coordinate $x$ is given by

$$x = K[R|t]X$$

Later, for the RANSAC filter, the feature points are needed in undistorted form as well, so this can be computed at no additional cost beforehand, such that additional distortion of the projected 3D points is not needed.

A first possibility for the second step would be a brute force search. For all features, the nearest projected 3D point is looked up by linear search among them. Here it would be advantageous to reduce the search space to a proximity of the feature, not the whole workspace. This can be achieved by using spatial hash datastructures instead, improving the performance significantly (Fig. 10).

Spatial hash datastructures work as follow: A given closed area is divided into equiareal rectangular cells (the dimensionality of the space can be arbitrary, but the memory requirements increases exponentially with the space dimension). The objects, which will later be looked up according to their position, are then placed (or linked) into the cell enclosing its position. If several objects are placed into
the same cell, they are arranged for example in a linear list or array. Then, for an arbitrary position, it can easily be computed, which cell contains that position and which objects are in proximity of the actual position, reducing search costs dramatically. Of course, cell sizes have to be chosen cautiously. Too large cells implies large number of objects in same cells augmenting linear search costs. Choosing too small cells let most cells be empty and increasing the number of cells with the power of the space dimension.

**Fig. 10.** Example of a two-dimensional spatial hash datastructure.

Now, using this spatial hash datastructure, the second possibility lies in doing again linear search to find the nearest neighbor, but this time in the reduced search space (depicted in Fig. 11). The cell sizes have been chosen to be the maximal distance allowed between feature and projected point. This then implies to search the actual cell and all its neighbors for the nearest object. The curse with the nearest neighbor is that the nearest projected point doesn’t need to be the right one. Depending on the geometry and the attitude of the image plane, 3D points of other features may lie nearest.

**Fig. 11.** The spatial hash datastructure allows much faster nearest neighbor search.
The third possibility tackles with this problem. The idea is, to get the right 3D point by using descriptors (as depicted in Fig. 12). For each 3D point, at least two corresponding features exist in the former views. Therefore for one of these features its descriptor can be computed and linked to the 3D point. Now, not the nearest neighbor is taken in the (limited) search space, but the 3D point having the most similar descriptor. The performance of these comparisons can further be improved by comparing first the background colors of both features (by the sign of the laplacian). If those already doesn’t match, no further comparison is made. Only if this first stage has been passed the more costly descriptor comparison is performed. Using this framework, the mentioned problem can be solved, giving better results in such situations. But on the other hand, this third possibility is much more expensive. For all detected features in a view, descriptors need to be computed, giving away a large part of the possible performance increase.

![Fig. 12](image). The spatial hash datastructure allows to quickly resolve all neighbors. As a descriptor for the projected 3D point, the descriptor of its the last corresponding feature is used. Then the right 3D point can be chosen by descriptor comparison.

The three possibilities will be discussed further and compared in the section "Results".

5.3 EpipolarMatching

The key idea of EpipolarMatching is, as suggested by the name, to use additional knowledge than just the feature positions and their descriptions. Since the whole camera calibration is approximately known for each view, this information should be used.

Epipolar geometry relates different views of a 3D point to each other (Fig. 13). Consider a feature in the first view. Assuming the distance of the corresponding 3D point from the camera is unknown, therefore it could lie anywhere on the
prolongation line of the camera center to the feature in the image plane. For a second, different view this must hold obviously as well and the feature must therefore lie on the projection of this line into its own image plane, which is known as the epipolar line. This relation can of course also be formulated mathematically using the fundamental matrix $F$: for two points $\mathbf{x}$ in the first image plane and $\mathbf{y}$ in the second, the following quadratic form must hold:

$$\mathbf{x}^T F \mathbf{y} = 0$$

By fixing $\mathbf{x}$ by a given feature, the quadratic form turns into a line equation on which the corresponding feature has to lie. Since the calibration parameters are measured by the tracking device and are only an approximation, a certain threshold has to be taken into account. But nevertheless, the search region is significantly reduced by demanding the matching feature to be in the proximity of the epipolar line. The EpipolarMatching then comes in two variants.

In the first, all unmatched features are taken into account: For each unmatched feature in the first view, the epipolar line constraint is computed. Then all features in the second view are submitted to the proximity-test and only the features passing the test have their descriptor computed and compared with the one of the actual feature (Fig. 14).

In the second, a spatial hash table is used again (2D) for considering just the features in the proximity of the epipolar line. The image is again divided into rectangular cells. Similar to the Bresenham Algorithm known from computer graphics [18], only the features in the cells touched by the epipolar line and those in a small band around it are used for the descriptor comparison (Fig. 15).

In both cases, again the same test on the background is performed, reducing further the number of actual full descriptor comparisons.

The two methods and the brute force approach are compared in the "Results" section.
Fig. 14. Only features with small distances to the epipolar line are considered as candidates for correspondences.

Fig. 15. Just the features in adjacent hash cells are considered as candidates for correspondences.

5.4 Other steps in the reconstruction process

The methods discussed in this subsection face with the other steps in the reconstruction process, apart from the correspondence finding, and have just been adapted for the current implementation.

Baseline Test Before any image is processed further through the reconstruction pipeline, it is first checked if the new position of the camera is valid. For this, two tests are performed. The first checks, if the baseline between the two views is large enough. If this test fails, the angle between the viewing direction is considered. If this is larger than some threshold, the view is accepted nonetheless. These tests are necessary to avoid large errors during the reconstruction (See [19] for more information).

Fast Hessian Corner Detection For each incoming image in the 3D reconstruction algorithm, the first step is the extraction of characteristic features. Lots of algorithms exist for this purpose, but since the focus in this work was on performance, the Fast-Hessian Corner Detector has been chosen. For Hessian Detectors, the determinant of the Hessian matrix is used as criteria for the existence of corners. It is used both for finding the location of a feature and also its scale (by convolution with Gaussians of different widths).
The modification made in Fast-Hessian Detectors lies in the discretization of the filter. The filter is cropped through the discretization, such that the resulting filter is different from the original one anyway. If the discretized filter is approximated by a piecewise constant filter (Fig. 16), a much more efficient implementation is then possible by the use of integral images. For more details, see [17].

![Fig. 16. Cropped filter (left) and the approximated filter (right) used by FastHessian (from [17]).](image)

**SURF Feature Descriptor** In order to be able to compare features based on characteristics of their surroundings, lots of descriptor algorithms have been proposed. Again, the aim lies on high performance, but of course the quality is crucial, too. The good performance of the SIFT Descriptor compared to other descriptors [20] is remarkable. Therefore, the similar, but boosted SURF descriptor has been used. Like SIFT, it mixes localized information and the distribution of image gradients of the surrounding area to construct a robust descriptor very efficiently. For more information, see [17].

**RANSAC Filter** The RANSAC (Random Sample Consensus) algorithm is a general method for fitting models in presence of many data outliers (similar to the Cross-Validation Method described e.g. in [21]). The idea is very simple. For a given data set, a randomly chosen subset is taken and the model estimated through this subset. Then the whole dataset is taken to validate the model. This procedure is repeated, until an acceptable model has been found.

For the reconstruction process, RANSAC is used to validate the correspondences found. For this purpose, the fundamental matrix is fitted to the data as described above. Then by the epipolar constraint, the correspondences are checked to be valid, i.e. it is tested if the corresponding features lies on the epipolar line ($x^T F y = 0$ for features $x$ and $y$).

**3D Reconstruction** The heart of the whole reconstruction algorithm is of course the generation of the 3D points from corresponding features. If there were no sampling/discretization limitations, two corresponding feature locations could be found perfectly and the camera parameters would also be known at absolute precision, then the corresponding point in space could be computed easily by taking the intersection of the rays from the two camera centers through
the features. Due to the mentioned restrictions, these rays will almost never cross in practice. Therefore, the positions have to be estimated in an optimal fashion. This is done by minimizing a suitable cost function. But minimizing errors is inappropriate in the 3D projective space, since concepts like distance and perpendicularity are not valid. The key idea therefore is to estimate a 3D position \( \mathbf{X} \), such that the reprojection error (the 2-norm of the distance of the original from the reprojected image points) gets minimized (see Fig.17). This is done by combining the equations \( \mathbf{x} = \mathbf{P} \mathbf{X} \) and \( \mathbf{x}' = \mathbf{P}' \mathbf{X} \) into a single equation \( \mathbf{A} \mathbf{X} = \mathbf{0} \). By having the cross-product between \( \mathbf{x} \) and \( \mathbf{P} \mathbf{X} \) equal zero, i.e. \( \mathbf{x} \times (\mathbf{P} \mathbf{X}) = \mathbf{0} \) and \( \mathbf{x}' \times (\mathbf{P}' \mathbf{X}) = \mathbf{0} \), this gives three equations linear in \( \mathbf{X} \) each, where one is linearly dependent. Therefore the matrix \( \mathbf{A} \) can be defined as follows:

\[
\mathbf{A} = \begin{pmatrix}
    x\mathbf{p}_3^T - \mathbf{p}_1^T \\
    y\mathbf{p}_3^T - \mathbf{p}_2^T \\
    x'\mathbf{p}'_3^T - \mathbf{p}'_1^T \\
    y'\mathbf{p}'_3^T - \mathbf{p}'_2^T
\end{pmatrix}
\]

Then, the least squares solution of the equation \( \mathbf{A} \mathbf{X} = \mathbf{0} \) can be computed elegantly by using the singular value decomposition of \( \mathbf{A} \). For more details, see [19].

![Fig. 17. The projected rays do not intersect in general. An approximated solution is chosen instead.](image)

**Bundle Adjustment** During the reconstruction process lots of images passes the pipeline. Initially, the position of a reconstructed 3D point is computed by taking the information of just two views. Due to the constant matching of features in consecutive images, a reconstructed 3D point may correspond to several
features in different images later. Bundle Adjustment deals with this fact and does the optimization of the 3D points: It minimizes the reprojection errors between the observed (features) and the predicted points (reprojection). The minimization is carried out with the aid of the Levenberg-Marquardt (LM) algorithm. Due to the large number of unknowns, general purpose implementations of LM (like MINPACK’s ‘lmder’) rise too high computational costs. Therefore the package ’Sparse Bundle Adjustment (sba)’ [22] has been used, which takes the lack of interaction between the variables into account resulting in a sparse Jacobian and therefore offers great performance for this purpose.

**Backprojection Filter** It is sometimes possible, that wrong correspondences are set up resulting in outliers. The described RANSAC algorithm tries to avoid this, but may fail rarely. Especially when several features are linked to the same 3D point, an outlier can disturb the optimization process severely. The Backprojection Filter, together with the Empty Cell Test, is the last instance, validating the out-coming spatial points. It comes in two flavors: A simple (and therefore fast) and a thorough variant.

The simple test takes all features in the actual view and backprojects the corresponding 3D points (if any) into the image plane. If then the error exceeds a certain threshold, the 3D point gets invalidated.

In the thorough variant, each 3D point is projected into all views, where it has a corresponding feature. If the error in any of this view exceeds a threshold, the 3D point is again invalidated.

*Remark*: The first variant is an "augmenting" algorithm, validating the points for each new view. By this augmenting procedure, the validity of the 3D points is already satisfied, even if a 3D point is not checked in the actual view (since only the 3D points are checked, which have been modified). Therefore the second variant does too much, it redundantly checks points again and again, which are not changed at all...

**Empty Cell Test** Next to the Backprojection Filter, another filtering step is performed. Based on the assumption, that most reconstructed 3D points have several neighbors in their proximity (since they are samples on a surface), the Empty Cell Test checks the number of neighbors for each 3D point. If this number is lower than a threshold, the point is assumed to be an outlier and marked invalid. Because this procedure is performed for all points for each new view, it is also possible that a point, beforehand classified as an outlier, becomes valid again. For this test, again the spatial hash datastructure is used (3D version this time). The first, brute force variant checks in all cells of the hash domain if the number of points is smaller than the threshold. Depending on the resolution of the hash, a huge amount of tests has to be performed. The second variant is more elegant. Starting from the reconstructed 3D points, for each one the number of points in its current and its neighboring cell is checked, being much more efficient than the first variant.
5.5 Memory Considerations

As one can imagine, the memory requirements of the algorithm are very demanding. Depending on the configuration of the algorithm, the requirements differ although. The images themselves would quickly fill the whole memory, if all of them would be stored. Therefore only the last two images are kept in memory. The size of all key- and 3D-points grow as well, as long new images are coming in. Here the amount of memory depends on the configuration chosen. In a configuration, where the third FastMatching variant (with descriptors) is taken, the feature descriptors is needed for every single keypoint (since the image should be released quickly, the descriptor computation must be performed for all potential features beforehand). Next to the mentioned performance loss, the storage of an extra array of 64 double’s for each point is quite demanding. For the other variant of FastMatch, no descriptors are needed. In that case, the descriptors are just used in the EpipolarMatching step (only comparing current and last view), such that the descriptors must be kept for these two views and can be released afterwards, reducing the memory requirements further. Nevertheless, the algorithm is acquiring memory as long it proceeds images, because new features are entering the database permanently, and all features are kept there, leaving room open for further improvements in the memory management.
6 Results

6.1 Test Case: Book

In this first test case, a book (see Fig. 18) with nice detectable features on the front side should be reconstructed. The aim was not to reconstruct the three-dimensional shape of the book, but to check the reconstruction accuracy of the algorithm. The attention lay on the planar front side: all reconstructed points there were assumed to lie in a plane, which is also easily visually checkable.

Fig. 18. Book with front side, which is planar and rich in features.
The results for this case are quite good (Fig. 19). All points are lying in close proximity of the plane. Since the reconstruction error goes quadratic in the $z$-direction of the camera, this result is satisfactory, since the accuracy is well enough for the registration of a pre-operative model.

### 6.2 Test Case: Vertebrae

![Vertebrae Model](image)
The second test case handles the aimed vertebrae reconstruction during the spine surgery. A plastic model of the vertebrae is coated with landmarks similar to the ones in the real surgery case (depicted in Fig. 20). Here the goal is not to check specially the accuracy, but to test the overall quality of the reconstruction of the object also with respect to the registration of the pre-operative model. Again, the result is satisfactory (Fig. 21). The shape of the object gets well

![Image](image.jpg)

**Fig. 21.** Above: The GUI shows detected features of the actually processed image and the corresponding 3D reconstruction of the vertebrae. Below: Closer view of the reconstruction.

represented by the point sample. Specially, there are almost no outliers, due to the final backprojection test and facilitating the pre-operative model registration.

### 6.3 FastMatching Comparison

In this subsection the performance and the quality of the different FastMatching variants should be compared. As seen in the FastMatching section, the different variants evolved as improvements of each other. Nevertheless, this doesn’t mean, that the last variant is the most appropriate one. The performance cannot be
measured directly indeed, since the number of 3D points increases constantly and may even differ between the different variants. Therefore the running time is measured and compared per 3D point. The results for an example sequence are depicted in Tab. 1). At first sight, the brute force is assumed to be much worse than the second one using the hash table. On the tests, there were indeed some differences, but in these orders of magnitude, they were insignificant for the overall algorithm. The third algorithm however had significant larger costs, as expected. As explained before, this is due to the additional descriptor computation, such that the differences in the FastMatching computation time then really starts to influence the overall computation time. For example in the last image of the example sequence the running times for FastMatching were 0.028s (Brute Force, 472 3D points), 0.0034s (FastMatching, 448 3D points) and 0.22s (Descriptor FastMatching, 441 3D points).

<table>
<thead>
<tr>
<th></th>
<th>Running Time (per 3D point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force</td>
<td>$4.9 \times 10^{-5}$</td>
</tr>
<tr>
<td>FastMatching</td>
<td>$1.2 \times 10^{-5}$</td>
</tr>
<tr>
<td>Descriptor FastMatching</td>
<td>$8.2 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

Table 1. Results of the different FastMatch variants w.r.t. running times for a single 3D point.

For the quality, the second variant is clearly not worse than the brute force search. Because the cells of the spatial hash have exactly the size of the allowed threshold, it makes absolutely no sense to search for other matches beyond the directly neighboring cells and the result of both algorithms is the same for any case. The geometrical problems mentioned before cannot be resolved by this two methods however. From a quality point of view, the last variant is certainly the best one, since only features are matched, that belong together. On the other hand, there is still the reprojection filter at the end of the algorithm, being able to remove bad reconstructed points based on mismatches. On the examined sequences, no obvious differences in the quality were observable, which was satisfying. Nevertheless, maybe a more thorough analysis could reveal some differences. Overall, the second variant, in combination with the reprojection filter, seems to be the best choice with respect to performance and quality.

### 6.4 EpipolarMatching Comparison

For the EpipolarMatching, again three different versions exist. With respect to performance, obviously the initial brute force approach is the worst. Not only the descriptors for all points have to be computed, but also all of them have to be compared, which is expensive. The restriction of the search space to a band around the epipolar line should therefore severely reduce the number of features for comparison. Again, the measurement of these differences is not quite easy.
Dependent on the distribution of the features in both images and the threshold chosen, these extensions have shown improvements from factor 2 to almost no improvement. In Tab. 2, the largest, smallest and average times of the three variants are depicted. The two variants for reduction take two similar approaches by utilising the epipolar constraint. As one can see, the approach with the spatial hash table is a bit faster than the other, since not all features have to be passed.

<table>
<thead>
<tr>
<th>Running Times (sec)</th>
<th>Largest</th>
<th>Smallest</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force</td>
<td>0.77</td>
<td>0.15</td>
<td>0.45</td>
</tr>
<tr>
<td>EpipolarMatching</td>
<td>0.48</td>
<td>0.15</td>
<td>0.33</td>
</tr>
<tr>
<td>EpipolarMatchingHash</td>
<td>0.39</td>
<td>0.15</td>
<td>0.27</td>
</tr>
</tbody>
</table>

Table 2. Results of the different EpipolarMatching variants w.r.t. running times.

With respect to quality, there should not be any difference between the brute force and the other two variants. But since the epipolar line depends also just on measurements, the correct feature may lie in some distance to the line. Therefore a trade-off in the size of the threshold and the size of the resulting search space has to be taken. On the examined sequences, no differences in the quality have been observed.

### 6.5 Overall Performance

For the overall performance just one configuration of all the different variants has been considered. It is using the FastMatching algorithm without descriptors, the EpipolarMatching variant with the hash table and the incrementing Backprojection Filter. The result for the example sequence is depicted in Tab. 3.

<table>
<thead>
<tr>
<th>Running Time (sec) for image</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>% of total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Detection</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>18.54 %</td>
</tr>
<tr>
<td>FastMatching</td>
<td>0</td>
<td>0.0018</td>
<td>0.004</td>
<td>0.0034</td>
<td>0.006</td>
<td>0.005</td>
<td>0.9 %</td>
</tr>
<tr>
<td>EpipolarMatching</td>
<td>0.21</td>
<td>0.33</td>
<td>0.38</td>
<td>0.32</td>
<td>0.24</td>
<td>0.14</td>
<td>71.53 %</td>
</tr>
<tr>
<td>RANSAC Filter</td>
<td>0.004</td>
<td>0.008</td>
<td>0.007</td>
<td>0.01</td>
<td>0.01</td>
<td>0.006</td>
<td>1.99 %</td>
</tr>
<tr>
<td>3D Reconstruction</td>
<td>0.015</td>
<td>0.02</td>
<td>0.009</td>
<td>0.04</td>
<td>0.017</td>
<td>0.007</td>
<td>4.77 %</td>
</tr>
<tr>
<td>Bundle Adjustment</td>
<td>0.0006</td>
<td>0.001</td>
<td>0.0009</td>
<td>0.002</td>
<td>0.002</td>
<td>0.001</td>
<td>0.33 %</td>
</tr>
<tr>
<td>BackprojectionFilter</td>
<td>0.0004</td>
<td>0.0016</td>
<td>0.0007</td>
<td>0.001</td>
<td>0.001</td>
<td>0.0006</td>
<td>0.24 %</td>
</tr>
<tr>
<td>Empty Cell Test</td>
<td>0.0012</td>
<td>0.0047</td>
<td>0.0054</td>
<td>0.007</td>
<td>0.009</td>
<td>0.011</td>
<td>1.69 %</td>
</tr>
<tr>
<td>Total</td>
<td>0.32</td>
<td>0.47</td>
<td>0.51</td>
<td>0.49</td>
<td>0.37</td>
<td>0.25</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Performance of the individual parts during a short image sequence.
From the table one can easily see, that the EpipolarMatching step takes the longest, followed by the Feature Detection step. In both of them, the necessary detection and description is performed, clearly indispensable for the algorithm. As said before, the Fast-Hessian/SURF combination has been chosen, being fast and having good quality. This combination is of course not mandatory, leaving it open to exchange them with other algorithms. The overall performance is therefore mainly linked to the performance of the choice made there. For the purpose of this thesis, it has shown that this combination is the best suited, as has been found out in previous studies.
7 Conclusions and Future Work

7.1 Conclusions

By putting all these parts of the algorithm together and arriving at a good interplay of the individual building blocks, a serious increase in performance has been achieved. Nevertheless, the quality of the reconstruction is still maintained, enabling as next the registration step to fit the pre-operative model into the reconstruction. Also, the reconstruction of a 3D model happens now on the fly, i.e. real-time reconstruction of the surgical scene has got possible by the improved treatise of the incoming image stream. This is primarily possible, because no high framerate for the reconstruction is needed. Since the Baseline Test require the camera to perform a certain movement until it accepts a new image, this fits perfectly together. For the time, that the reconstruction takes, the camera can just be moved to a new valid position.

7.2 Future Work

The current implementation works already quite satisfactory. Nevertheless helpful improvements could be the following:

- **Hidden 3D points:** During the FastMatching algorithm, all 3D points are projected into the new view. But considering the shape of the real 3D model, also points invisible to the camera are projected into the view, which makes no sense. This problem could be tackled, as mentioned before, by using also feature descriptors for the tracking decision with the drawback of augmented computing time. Another idea would be to use the available points to build a approximative 3D surface and then to decide based on the surface normal in each point, if the point should be projected into the view or not.

- **Memory Management:** New Keypoint objects are created as long new images are proceeded. Therefore the required memory for processing and image sequence depends on the length of this sequence. An improvement of the algorithm could lie in the removal of old and/or seldom used features from the database to hold its size constant and therefore let the algorithm accept arbitrary long image sequences.

- **2D tracking** In addition to the tracking performed by the FastMatching algorithm, another tracking method could further improve the reconstruction quality. Beeing able to compute the trifocal tensor for three consecutive images, a correspondance found for the first two images makes a prediction for the corresponding feature in the third image possible.
References
