Reconstructing a 3D structure from unknown viewpoints using uncalibrated cameras

B. Stoeller, T. Kostelijk, F. Huizinga
{bramstoeller,mailtjerk,folkerthuizinga,}@gmail.com

January 29, 2010

Abstract

In this report we consider the problem of computing the 3D structure of an unknown arbitrarily-shaped scene from two photographs taken at unknown viewpoints using uncalibrated cameras.

1 Introduction

In this report we consider the problem of computing the 3D structure of an unknown arbitrarily-shaped scene from two photographs taken at unknown viewpoints using uncalibrated cameras. Our initial idea was to create a 3D object recorder which runs on a cell phone with a camera and an Internet connection, like a Google Android G1, an iPhone or any other smart phone. The user takes several pictures of an object which are sent to a server. This server computes a 3D model and pushes the model back to the cell phone. This allows one to share 3D models with your friends. This paper focusses on a prototype where several images are combined to form a 3D model of the real object as accurate as possible.
Reconstructing the structure of a 3D scene from two (or more) photographs is a fundamental problem in computer vision. It is also referred to as *multiple view geometry* [1]. If the internal and external camera parameters are known it is very well possible to achieve such a model from a set of photographs. However we do not have any information about the position of the camera[1] nor do we have information about the internal camera parameters due to the auto focus function of type of devices. This significantly increases the complexity of the problem.
Sections 2 to 11 describe a way to obtain information about the 3D space from only the information embedded in a set of photographs without knowing anything about the real world. Next we describe two methods to create an actual 3D model and we discuss the differences between these two approaches.

[1]One might suggest to use the sensors of a G1 or iPhone as an a priori estimation the of orientation of the camera, but we chose to use a method which is device independent and leave the sensor fusion as a future improvement.

1.1 Notations

We use italic characters for scalars like x. Two dimensional vectors are denoted by bold minuscules (lower case) like $\mathbf{x}$ while three dimensional vectors are denoted by bold majuscules (upper case) like $\mathbf{X}$. We always use homogeneous coordinates unless stated otherwise. I.e. a 2D point is written as a 3×1 vector: $\mathbf{x} = (x\ y\ 1)^\top$. A line/ray between two points $\mathbf{X}_1$ and $\mathbf{X}_2$ is represented by $\langle \mathbf{X}_1, \mathbf{X}_2 \rangle$. A matrix is denoted by a typewriter character like $\mathtt{F}$. We always use round parentheses for vectors and matrices except when we describe the contents of a composed matrix like $\mathtt{P} = [\mathtt{I}|\mathbf{0}]$. I.e. the matrix $\mathtt{P}$ consists of an identity matrix $\mathtt{I}$ (e.g. 3×3) followed by a zero vector $\vec{0}$ (3×1) resulting in a 3×4 matrix. Finally the *skew-symmetric matrix* [1] (equation A4.5, pp. 581) of a vector $\mathbf{e}$ is denoted by $[\mathbf{e}]_\times$.

2 Feature Extraction

To determine the 3D structure of a scene the unknown viewpoints of the photographs need to be computed. The viewpoints are calculated using stereo correspondence. Stereo correspondence is the process where features from both photographs are extracted and compared see figure 1. The locations of the matching features will be used as a source of determining the viewpoint of the cameras. This section describes the feature extraction process.

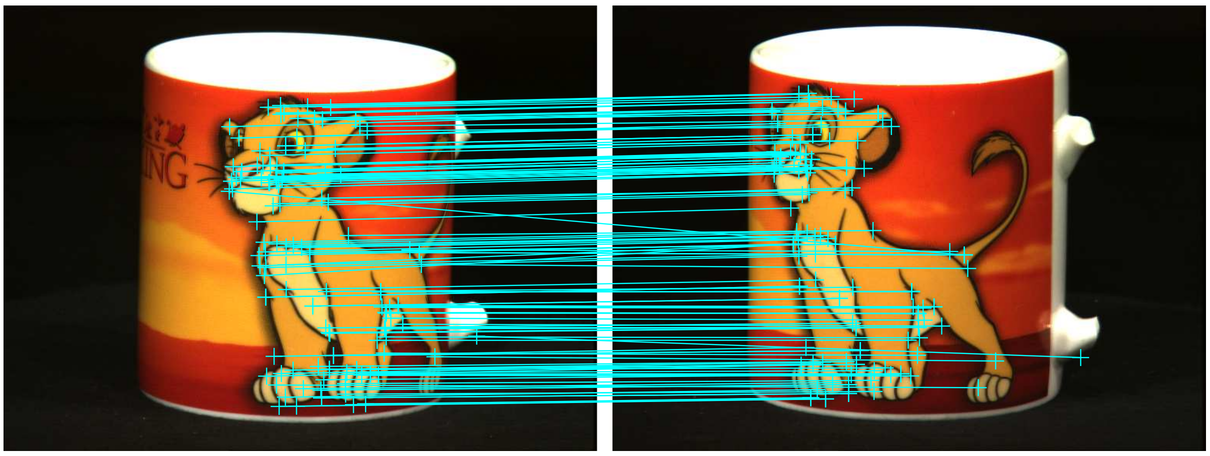

Figure 1: SIFT

2.1 SIFT

Because the accent of this paper is not the feature matching and lots of research is done in this field we use a standard feature extractor. We use the feature extractor SIFT, scale-invariant feature transform. This method, invented by David G. Lowe [2], is very popular and widely used. SIFT extracts features that are invariant to scaling, rotation, changes in illumination and even changes in viewpoint. These invariants are useful for finding point correspondences within two different views. We use a Matlab implementation of [3] In standard form, SIFT returns thousands of features. 100 Features will be more than enough to feed our algorithms. We experimented with several thresholds values to keep the retrieved features small. An example of the SIFT algorithm performed on a Lion King cup with threshold can be seen in figure 1 [4].

3 Feature Matching

Given the SIFT features within two images, the task is now to find point correspondences between the two images $\{\mathbf{x}_i \leftrightarrow \mathbf{x}'_i\}$. These correspondences are found by comparing all features between the two images in a brute force manner.

If we compare the location of the features in the two photographs, we observe a location difference, furthermore called shift. This shift is caused by the difference in viewpoints of the photographs.

3.1 Removing outliers

Because the features are compared without taking their location within the images into account, some correspondences of points are found that do not describe the same point in the original scene. These unwanted correspondences are outliers and need to be removed from the dataset. A common method to remove outliers is RANSAC, however because of time constraints we decided to apply a much simpler method of removing outliers with fairly good results.

3.2 Mean shift outlier removal

We use the fact that outliers have the property that they performed a shift that is either to large or too small compared to the mean shift of all the features in the first and second image. We iteratively remove the most extreme outlier and recompute the mean shift. An advantage is that the previous outliers does not influence this mean shift.

4 Multiple View Geometry

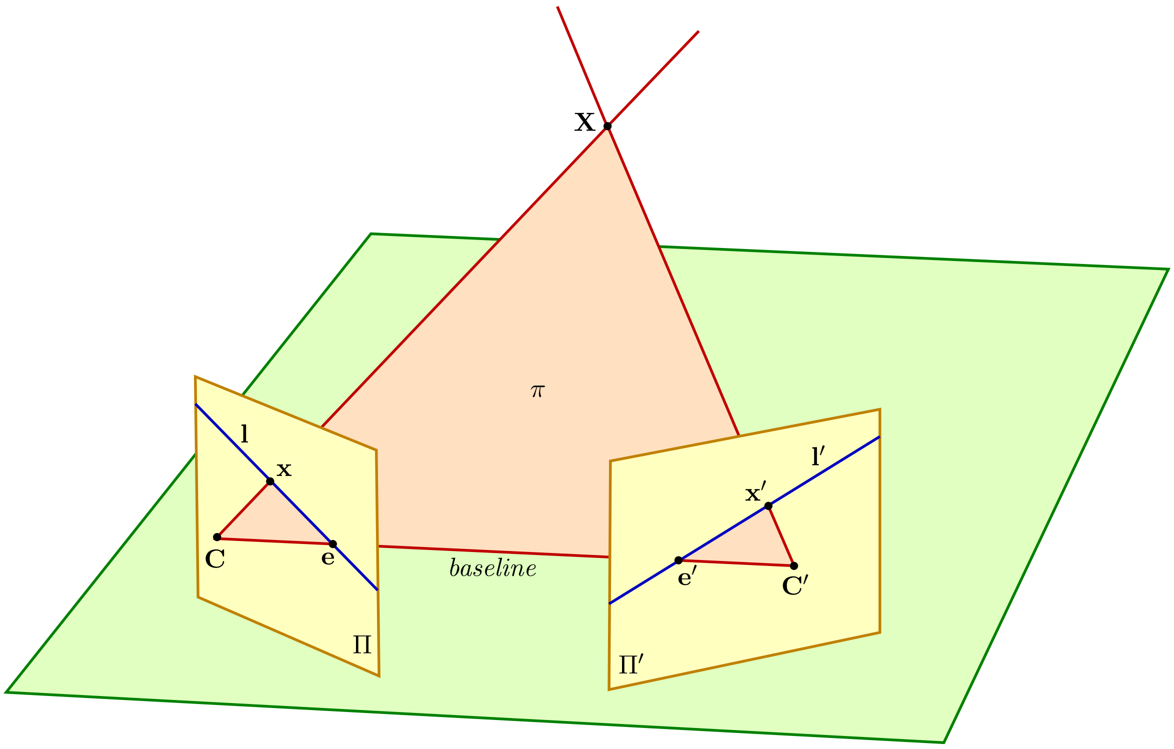

Figure 2: Multiple view geometry

Figure 2 shows a stereo graphic scene setup. Two cameras are indicated by their *camera centers* $\mathbf{C}$ and $\mathbf{C}'$ and their *image planes* or *retinas* Π and Π' respectively. The camera centers, 3D point $\mathbf{X}$, and its images $\mathbf{x}$ and $\mathbf{x}'$ lie in a common plane π. The line $\langle \mathbf{C}, \mathbf{C}' \rangle$ is called the *baseline* and intersects the image planes Π and Π' at *epipoles* $\mathbf{e}$ and $\mathbf{e}'$ respectively. These are the locations on the retina where the other camera center can be seen. An *image point* $\mathbf{x}$ back projects to a *ray* $\langle \mathbf{C}, \mathbf{X} \rangle$ in the 3D space. This ray is represented as the *epipolar line* $\mathbf{l}'$ on retina Π'. So even if we don't know the exact location of $\mathbf{X}$, given the position of $\mathbf{x}$ we can calculate the epipolar line or *scan line* $\mathbf{l}'$ where point $\mathbf{x}'$ must appear. And if we do know the a corresponding $\mathbf{x}$ and $\mathbf{x}'$ we can calculate the intersection between the ray from $\mathbf{C}$ through $\mathbf{x}$ and the ray from $\mathbf{C}'$ through $\mathbf{x}'$ which should be the real 3D point $\mathbf{X}$.

5 Epipolar geometry

Suppose a scene is viewed from different sides in two photographs. As can be seen in figure 2, 3D point $\mathbf{X}$ is now viewed as $\mathbf{x}$ in the first and $\mathbf{x}'$ in the second image. The camera centers $\mathbf{C}$ and $\mathbf{C}'$ of the images Π and Π' are connected with a baseline. Every view has a corresponding epipole $\mathbf{e}$ which is defined as the point of intersection of the baseline with the corresponding image plane. The epipolar plane contains this baseline, and the epipolar line $\mathbf{l}$ is the intersection of this epipolar plane π with the image plane Π. A point $\mathbf{x}$ in the left image Π will correspond to a point $\mathbf{x}'$ on the epipolar line $\mathbf{l}'$ in the right image Π' and vice versa. The position on the epipolar line $\mathbf{l}'$ depends on how far away the underlying 3D point $\mathbf{X}$ is positioned, i.e. the distance between $\mathbf{C}'$ and $'\mathbf{X}$.

6 Fundamental Matrix $\mathtt{F}$

The fundamental matrix or bifocal tensor, describes the epipolar geometry between two images [5]. See figure 2. It is the key concept in stereo vision where the internal parameters of the cameras are unknown.
The fundamental matrix $\mathtt{F}$, therefore plays an important role in the construction of the 3D structure. It defines the mapping between a 2D point $\mathbf{x}$ in the first image to the corresponding epipolar line as $\mathbf{l}' = \mathtt{F}\mathbf{x}$. Because the rays defined by points $\mathbf{x}$ and $\mathbf{x}'$ are coplanar we can state a very useful property of the fundamental matrix, $\mathbf{x}'^{\top}\mathtt{F}\mathbf{x} = 0$. This property is exploit to compute $\mathtt{F}$.

6.1 Solving $\mathtt{F}$

If we use seven known point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ we can rewrite $\mathbf{x}'^{\top}_i\mathtt{F}\mathbf{x}_i = 0$ as an 7×9 system of non homogeneous linear equations.

$$\mathtt{A} = \begin{pmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ x_2x'_2 & x_2y'_2 & x_2 & y_2x'_2 & y_2y'_2 & y_2 & x'_2 & y'_2 & 1 \\ x_3x'_3 & x_3y'_3 & x_3 & y_3x'_3 & y_3y'_3 & y_3 & x'_3 & y'_3 & 1 \\ x_4x'_4 & x_4y'_4 & x_4 & y_4x'_4 & y_4y'_4 & y_4 & x'_4 & y'_4 & 1 \\ x_5x'_5 & x_5y'_5 & x_5 & y_5x'_5 & y_5y'_5 & y_5 & x'_5 & y'_5 & 1 \\ x_6x'_6 & x_6y'_6 & x_6 & y_6x'_6 & y_6y'_6 & y_6 & x'_6 & y'_6 & 1 \\ x_7x'_7 & x_7y'_7 & x_7 & y_7x'_7 & y_7y'_7 & y_7 & x'_7 & y'_7 & 1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{1}$$

It is easy to see that the parameters of this equation constructing $\mathtt{F}$ can be solved by performing a singular value decomposition (SVD). We used the fact that $[\mathtt{USV}^{\top}] = \mathtt{svd(A)}$, where the last column in $\mathtt{V}$ is equal to the entries of $\mathtt{F}$. Now we have solved the correspondence between the points in one image and the epipolar line in the other image.

7 Epipolar parametrization

This section explains how to calculate the epipoles, let

$$\mathtt{F} = \begin{pmatrix} a & b & \alpha a + \beta b \\ c & d & \alpha c + \beta d \\ \alpha' a + \beta' c & \alpha' b + \beta' d & \alpha' \alpha a + \alpha' \beta b + \beta' \alpha c + \beta' \beta d \end{pmatrix} \tag{2}$$

then the two epipoles are defined as $(\alpha, \beta, -1)^\top$ and $(\alpha', \beta', -1)^\top$ They can be calculated by isolation or, better, by computing the kernel of $\mathtt{F}$. See equation 2 based on [1] (equation 11.8, pp. 296) for more details.

8 Projection Matrices $\mathtt{P}, \mathtt{P}'$

A projection matrix $\mathtt{P}$ maps every 3D point $\mathbf{X}$ in the real world to some 2D point $\mathbf{x}$ on the retina Π of a camera. The 3D space is defined in the origin of the first camera. This means that camera center $\mathbf{C}$ is located at $(x, y, z)^\top = (0, 0, 0)^\top$ and it is looking in the direction $(0, 0, 1)^\top$. Therefore the projection matrix $\mathtt{P}$ of $\mathbf{C}$ is defined by a 3×4 identity matrix $\mathtt{P} = [\mathtt{I}|0]$:

$$\mathtt{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3}$$

The transformation $\mathbf{x} = \mathtt{P}\mathbf{X}$ from a 3D point $\mathbf{X}$ to a 2D point $\mathbf{x}$ in the first camera is given by:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \overset{\textit{homogeneous}}{\equiv} \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{4}$$

Which is quite intuitive as the larger the distance (z) between the camera and a point $\mathbf{X}$, the closer $\mathbf{x}$ is to the center of the retina. And therefore objects farther away appear smaller than objects close to the camera. The other camera is shifted and rotated over three axis. The projection matrices corresponding to a fundamental matrix $\mathtt{F}$ may be chosen as $\mathtt{P} = [\mathtt{I}|0]$ and as suggested by Luong and Viéville [6] $\mathtt{P}'$ an be defined as $\mathtt{P} = [[\mathbf{e}']_\times F|\mathbf{e}']$. Where $[\mathbf{e}']_\times$ is defined as the *skew-symmetric matrix* of $\mathbf{e}'$:

$$[\mathbf{e}]_\times = \begin{pmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{pmatrix} \tag{5}$$

9 Triangulation

If the two rays from image points $\mathbf{x}$ and $\mathbf{x}'$ are back-projected in the scene they intersect at the 3D point $\mathbf{X}$ they describe. Since the correspondences $\mathbf{x}$ and $\mathbf{x}'$ come from measurements the rays are skew and will therefore not exactly intersect. To asses this problem one should use a method to estimate this intersection given the measurements $\mathbf{x}$, $\mathbf{x}'$ and the corresponding projection matrices $\mathtt{P}$ and $\mathtt{P}'$. Intuitively one would determine the intersection as the midpoint of the common perpendicular to the two rays in space. But this method is not suitable because concepts as distance and perpendicularity are not valid in the context of projective geometry. i.e. this method will give different results depending on the projection. Instead, a projective-invariant method triangulation is used. The key idea of triangulation is to estimate the 3D point $\hat{\mathbf{X}}$ which satisfies

$$\hat{\mathbf{x}} = \mathtt{P}\hat{\mathbf{X}}, \hat{\mathbf{x}}' = \mathtt{P}'\hat{\mathbf{X}} \tag{6}$$

Where $\hat{\mathbf{x}}$ is an adapted image point of $\mathbf{x}$. Because this method only minimizes distances in the 2D image, the projections will not have any influence which makes this method projective invariant.

To obtain the 3D point $\hat{\mathbf{X}}$. We can combine equation 6 into a form $\mathtt{A}\mathbf{X} = 0$.

$\mathbf{x} = \mathtt{P}\mathbf{X}$ implies that $\mathbf{x} \times (\mathtt{P}\mathbf{X}) = 0$, writing this out gives:

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \times \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix} \mathbf{X} \tag{7}$$

$$= \begin{pmatrix} x & y & 1 \end{pmatrix} \times \begin{pmatrix} \mathbf{p}_1^\top X \\ \mathbf{p}_2^\top X \\ \mathbf{p}_3^\top X \end{pmatrix} \tag{8}$$

$$\begin{aligned} y(\mathbf{p}_3^\top X) - (\mathbf{p}_2^\top X) &= 0 \\ (\mathbf{p}_1^\top X) - x(\mathbf{p}_3^\top X) &= 0 \\ x(\mathbf{p}_2^\top X) - y(\mathbf{p}_1^\top X) &= 0 \end{aligned} \tag{9}$$

Using the linear independent equations for both images we get:

$$\mathtt{A} = \begin{bmatrix} x\mathbf{p}_3^\top - \mathbf{p}_1^\top \\ y\mathbf{p}_3^\top - \mathbf{p}_2^\top \\ x'\mathbf{p}_3^\top - \mathbf{p}_1^\top \\ y'\mathbf{p}_3^\top - \mathbf{p}_2^\top \end{bmatrix} \tag{10}$$

Now the 3D point $\hat{\mathbf{X}}$ is estimated.

10 Perspective Correction

The point cloud obtained by triangulation described previously is suffering from perspective distortion, see figure 3. This is a direct result from the constructed projection matrices that do not take the vanishing points into account. This section describes the steps to remove this projective distortion by upgrading it to a metric reconstruction. Properties as parallelism, ratio between lines and ratio between areas are lost in the projective reconstruction. An affine projection would be more useful because it recovers these properties. The recovery of the affine

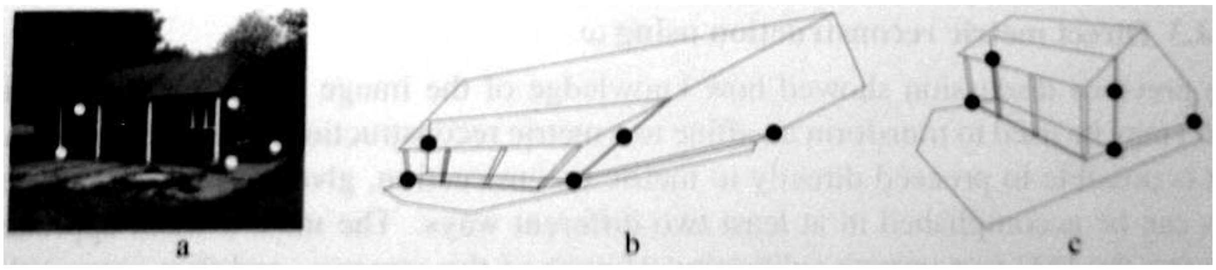

Figure 3: *borrowed from [1] (figure 10.6, pp. 276)*

properties involves finding the line l_∞ at infinity which describes the horizon. Parallel lines on the Euclidean plane intersect on l_∞ so one could for example use this property to identify the lines that intersect on l_∞ as parallel on the original plane. However a less intuitive but more general approach would be to transform the l_∞ to its canonical position in the projection $(0,0,0,1)^\top$.

To make this more clear, consider a point $\mathbf{X}$ lying on this canonical position, such that it satisfies $(0,0,0,1)^\top \cdot \mathbf{X}^\top = 0$. It shows directly that this point should be of the form $\mathbf{X} = (x,y,z,0)^\top$ where x, y, z are arbitrary and the homogeneous factor should be 0. Transforming this homogeneous coordinate to real word coordinates involves division by this homogeneous factor of 0 which causes all points on this canonical line to lie at the (infinite) horizon.

To transform the line of infinity in the image $l\infty$ to its canonical representation we have to find

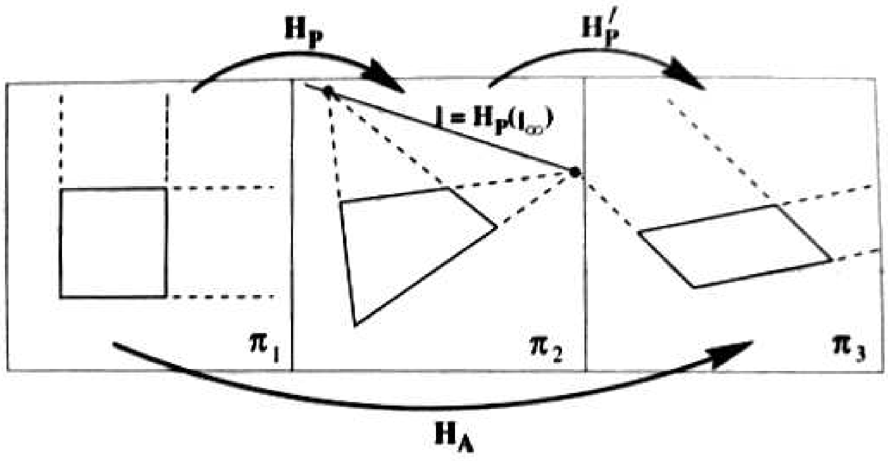

Figure 4: *borrowed from [1] (figure 2.12, pp. 47)*

a transformation $\mathtt{H}$ such that $\mathtt{H}^{-1} l_\infty = (0, 0, 0, 1)^\top$. $\mathtt{H}^{-1}$ can be calculated using a Householder matrix [1]. Next the transformation $\mathtt{H}$ is applied to all 3D points in the projective reconstruction. The affine reconstruction is now realized see figure 4.
This reconstruction is far more useful but real Euclidean measurements cannot be made. A second upgrade should be performed, the metric reconstruction. This again involves searching for a homography where a Cholesky factorization is used. Unfortunately, due to lack of time, we did not fully comprehend this.

11 3D Model Reconstruction

Now that the projection matrices are estimated for both cameras we can start the 3D reconstruction. We will provide the reader with a quick overview of two different methods suited for obtaining the 3D model.

11.1 Point Cloud and RANSAC

As explained in section 9, given the projection matrices, we can triangulate all point correspondences obtained by sift to construct a point cloud. This point cloud can then be used to create a polygonal structure using the RANSAC technique: random 3-point subsets of the data are selected to define planes and the number of 3D points which are less than a user-specified distance from each plane are counted. The plane with the greatest number of consistent points is stored, and the data points which were consistent with it removed from the structure [7]. The image of the camera most perpendicular to a polygon can be chosen as the texture for that particular polygon.

11.2 Space Carving

Space carving is another way of reconstructing the 3D model. Assuming a sufficient number of photographs from different views of an object are available and their projection matrices are known, a 3D structure can be carved out of a volume. Space carving starts with a volume $\mathcal{V}$ composed of cubic voxels v containing the *entire* target object. For instance, a cube-like volume consisting of voxels (in the order of 10^6 to 10^9). For each voxel at the surface $v \in Surf(\mathcal{V})$ one can calculate where it appears on the image plane Π_i of each camera $\mathbf{C}_i$ that can observe v using its projection matrix $\mathtt{P}_i$.
The next step is to check if the projection is consistent in the image of every camera that can observe the voxel. If so, the color of each image at the position (i.e. a small region) where voxel

v could be seen is more or less the same[2]. The voxels that are not consistent are being removed. This iterative process starts at the surface of $\mathcal{V}$ and works its way down to the surface of the object literally carving out the model. The remaining voxels form a volume $\mathcal{V}^*$ which is a model of the target object.
As the volumetric model has a large complexity in terms of voxel amount, it is inappropriate for real time rendering applications. A possible solution for reducing the model complexity would be isosurface extraction using a marching cubes algorithm [8]. The image of the camera most perpendicular to a polygon can be chosen as the texture for that particular polygon.

12 Implementation

For our implementation we used Matlab as the prototyping environment. We applied the SIFT library of [2] to some images from the dataset [4]. To obtain correct correspondences, the outlier removal function was applied (see section 3.1). From these correct point correspondences, we were able to construct a fundamental matrix `F` by applying the eightpoint algorithm which makes use of the function `linsolve`.
To ensure the correctness of `F`, we manually selected 8 point correspondences Later on a more suitable solution was found by performing the SVD. We also applied SVD on all point correspondences found by SIFT.
With both methods we got stuck with an `F` which contained very small values.

$$\mathtt{F} = \begin{pmatrix} 0.000000001716732 & -0.000000125985657 & 0.000031618609317 \\ 0.000000176118952 & -0.000000008553157 & 0.000392929989641 \\ -0.000041312297443 & -0.000405217748327 & -0.002289378115507 \end{pmatrix} \tag{11}$$

If we ignored this (intuitively bad) result and calculated the projection matrices they also contain very small values.
Projection matrices contain rotation translation and shearing which are hard to interpret if you look only at the values. To get a clear idea of how a certain projection matrix behaves we created an artificial cube with colored edges. Due to perspective transformation and knowledge of the model (its a cube) the projection to the real world frame gives us insight how the projections behaved.
If we applied this evaluation on our projections matrices, it resulted in a unwanted largely stretched cube. Because of this result, we searched for something to compare our code with. We used a library from the writers of the book [1] which was downloadable from [3]. This library contained several functions like the conversion from `F` to `P` and vice versa. It also contained example stereo images with their corresponding `F` matrices. We compared all parts of our code, including the results and got equal results.
The `F` matrices of the library where also very small. This was very surprising, we could conclude that we calculated it correctly and the matrices should be small.
Still the projection matrices came up with incorrect results. The only explanation we could think of is because we do not apply the perspective correction (see Section 10). Unfortunately we could not implement this, as the theory was hard to understand and we where running out of time.

13 Conclusion

Constructing a 3D model from photographs taken from various views using uncalibrated cameras is a daunting task. The steps require a lot of knowledge about linear algebra and multiple view geometry. We underestimated the time it took to understand this theory.
At the point of implementation we got stuck on the problem of the small `F` values, this took

[2]The Euclidean distance in RGB space is often used here

us a lot of time. We have an idea of the cause of this problem but do not know this for sure. We think the problem occurs because we do not apply perspective correction. Unfortunately we could not finish the prototype. However this report gives a good start to implement the system and could be of good use for further research.

References

[1] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[2] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[3] A. Vedaldi, “Sift for matlab,” a Matlab implementation of SIFT. [Online]. Available: http://www.vlfeat.org/∼vedaldi/code/sift.html

[4] J. M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders, “The amsterdam library of object images,” *International Journal of Computer Vision*, vol. 61, no. 1, pp. 103–112, 2005. [Online]. Available: http://www.science.uva.nl/research/publications/2005/GeusebroekIJCV2005

[5] Q. Luong, “Matrice fondamentale et calibration visuelle sur l’environment,” 1992.

[6] Q.-T. Luong and T. Viéville, “Canonical representations for the geometries of multiple projective views.” *Computer Vision and Image Understanding*, vol. 64, no. 2, pp. 193–229, 1996. [Online]. Available: http://dblp.uni-trier.de/db/journals/cviu/cviu64.html#LuongV96

[7] A. W. Fitzgibbon and A. Zisserman, “Automatic 3D model acquisition and generation of new images from video sequences,” in *Proceedings of European Signal Processing Conference (EUSIPCO ’98), Rhodes, Greece*, 1998, pp. 1261–1269. [Online]. Available: http://www.robots.ox.ac.uk/∼vgg

[8] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, 1987.