# Structure from Motion

Bram Kooiman 11415665
Zvonimir Dujmic 11029935

May 2018

## 1 Introduction

Structure from motion is an algorithm that uses 2D images to estimate 3D structure. We will use this algorithm to construct a 3D model of a house from a collection of regular 2D photos.

In the first chapter we discuss the Fundamental Matrix and how to obtain it using the Eight-Point Algorithm. The following chapters discuss implementations of Point-View Matrix construction and Structure From Motion. The subsequent chapter compares various optimisations of the algorithm. The final chapter discusses untested ideas on how the process could be optimised further.

## 2 Fundamental Matrix

In this chapter we estimate the fundamental matrix between two consecutive frames with the eight-point algorithm. We first introduce the fundamental matrix and the epipolar constraint by its math, then we compare unoptimised and optimised versions of a Matlab implementation of the eight-point algorithm.

### 2.1 Math behind the Fundamental Matrix and Epipolar Lines

We first discuss what the fundamental matrix is and why the epipolar constraint must hold. There are many points that may be responsible for a pixel occurence on a picture. These points lie on a line which is projected to a single point in the picture. If we take another picture from a different angle we can observe the points that may have been responsible for the pixel occurence in picture one as a line. This is the epipolar line. The object that was in fact responsible for the pixel occurence in picture one will also show a pixel occurence along this line in picture two. The epipolar lines, the object and the camera centers would geometrically all lie in the same plane. This plane is called the epipolar plane.

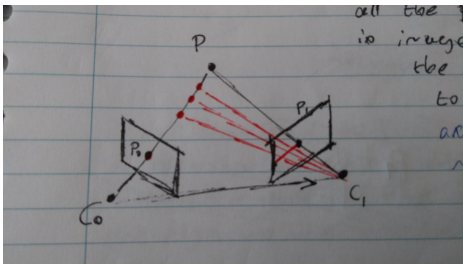


Figure 1: Epipolar line

We call the vector from $C_0$ to point $p_0$ the vector $\vec{p_0}$, the vector from $C_1$ to point $p_1$ the vector $\vec{p_1}$ and the translation vector between the two camera centers $\vec{t}$. If we want to express $\vec{p_1}$ in the coordinate system of $\vec{p_0}$, we need to multiply it with the rotation matrix between camera point 0 and 1 $R$. We use homogenous coordinates to represent the 2D points with a 3D vector. The fact that these vectors are coplanar can be mathematically expressed as the constraint that the dot-product of any of the vectors with the cross-product of the other two should be 0.

$$\vec{p_0} \cdot (\vec{t} \times R\vec{p_1}) = 0$$

We can also write the dot product as a vector multiplication of which the first vector is transposed and the cross-product as a matrix-multiplication of a skew-symmetric matrix of $\vec{t}$ which we write as $[t]_\times$:

$$\vec{p_0}^T([t]_\times R\vec{p_1}) = 0$$

The fundamental matrix $F$ is in fact $[t]_\times R$, so we write the epipolar constraint as:

$$\vec{p_0}^T F\vec{p_1} = 0$$

The above constraint should hold for every point correspondence. We use the data we have on point correspondences to estimate the elements of $F$. We gather the data in the matrix $A$ as described in the assignment. Observe that $\vec{f} = \vec{0}$ is a trivial solution, so we set the constraint $||\vec{f}|| = 1$. The fact that we have this constraint means that $A$ is now rank deficient and we can describe it with eight variables only. Hence we need only eight point correspondences. This is where the eight-point algorithm lends its name to. Observe that if we want to see

$$A\vec{f} = 0\vec{f}$$

it must mean that $\vec{f}$ is the eigenvector of $A$ corresponding to the eigenvalue 0. For data in the wild it is likely that the data is not so perfect that $A$ would have an eigenvalue 0, so instead we pick the eigenvector that has the lowest eigenvalue. Although the eight-point algorithm specifies to use eight point correspondences

in its very name, more can be included if we choose a least-squares minimisation approach. A least-squares generalisation of eigenvalue decomposition is singular value decomposition.

$$A = UDV^T$$

The least-'eigenvector' is then found on the last column of $V$. We need the fundamental matrix $F$ to be singular, in fact of rank 2, in order for it to be the fundamental matrix. Remember that the matrix $F$ is supposed to be equal to $[t]_\times R$ where $[t]_\times$ is a skew-symmetric matrix. This matrix is of rank 2, hence $F$ will also be of rank two. The $F$ we find through SVD may not in fact be of rank two, so we must force it. We do so by taking the SVD of $F$, set the smallest singular value in $D$ to 0 and reconstruct to find $F'$.

$$UDV^T = F$$
$$F' = UD'V^T$$

## 2.2 Implementation of the Eight-Point Algorithm

We find point correspondences by detecting and matching SIFT-features across the two frames. The eight-point algorithm is named after the fact that it can estimate the fundamental matrix based on eight point-correspondences, but in this work we use twenty. An amount high enough to offer the benefits of more available information, but not so high that a random pick would include many outliers.

A beneficial step to take is normalisation of the point correspondences before the eight-point algorithm so that their mean is 0 and their standard deviation is $\sqrt{2}$. It is important to denormalise the matrix F after the algorithm.

As mentioned before, including an outlier (a 'wrong correspondence') in the eight-point algorithm will make for a bad estimate of the fundamental matrix. In order to prevent such inclusion we perform RANSAC. Twenty points at random are selected to determine an estimate of F. Given the current estimate of F we count how many of the other point correspondences agree with the estimate. If the number of agreeing point correspondences is higher than the previous best estimate of F, the current estimate becomes the new best estimate. A point correspondence is counted as an inlier if the Sampson distance between the two points as determined with the current estimate of $F$ is below a threshold $t = .25$:

$$d = \frac{(p_0 F p_1)^2}{(Fp_0)_1^2 + (Fp_0)_2^2 + (F^T p_1)_1^2 + (F^T p_1)_2^2}$$

We compare the performance of the unoptimised eight-point algorithm, normalised eight-point algorithm and normalised eight-point algorithm with RANSAC. We make a visual assessment of the epipolar lines and a quantative assessment of the the epipolar constraint. For all constraints it should hold that.

$$\vec{p_0}^T F \vec{p_1} = 0$$

By taking the sum of this product over all point correspondences we can evaluate the quality of the estimate of $F$ by seeing how close it is to zero.

Correspondences are found by functions from the package vl_feat where we set the threshold of vl_ubcmatch to 4. We use twenty point correspondences for one estimate of $F$. The threshold below which RANSAC counts an inlier is .25 and the maximum amount of iterations is set to 300.

| | run 1 | run 2 | run 3 |
|---|---|---|---|
| unoptimised | 19.17 | 23.47 | -23.68 |
| norm | 0.79 | -1.91 | -1.93 |
| norm+RANSAC | 1.20 | -0.85 | -0.95 |

Table 1: Deviation from the epipolar constraint

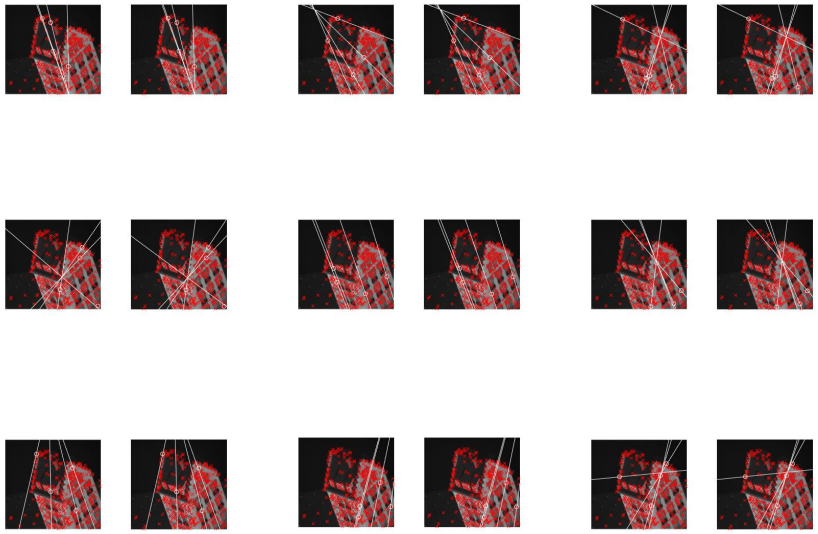

Figure 2: Epipolar lines for 3 separate runs for unoptimised (left), normalised (center) and normalised with RANSAC(right) on two consecutive image pairs. The most important thing to note is that the estimate of the epipole is changing for each run. Normalised with RANSAC seems to give the most reliable result

From table 1 we see that normalisation grants a strong improvement. From figure 2 we see that additionally performing RANSAC makes for more reliable results.

An epipolar line is visible where an epipolar plane intersects with the camera image. All epipolar planes pivot around the translation vector from $C_0$ to $C_1$. If the epipole - the point where all epipolar lines intersect - is visible in the image it means that the location where the camera must have been to take the one image is visible in the other. From the rightmost column in 2 we estimate that we're either zooming in or zooming out between these two images.

# 3 Pointview Matrix

In order to reconstruct structure from motion we need the point-view matrix. On its columns we find points and on its rows we find frames in which the points occur. Entries of this matrix are the x- and y-coordinates where the point occurs in the frame. If a point does not occur in the frame (or the frame does not posses the point) the coordinates are set to the flag 'NaN' (Not-A-Number). Here we see a point-view matrix expressed in variables and an example of how one could look like.

$$\begin{bmatrix} x_{11} & x_{12} & ... & x_{1p} \\ y_{11} & y_{12} & ... & y_{1p} \\ x_{21} & x_{22} & ... & x_{2p} \\ y_{21} & y_{22} & ... & y_{2p} \\ : & : & : & : \\ x_{f1} & x_{f2} & ... & x_{fp} \\ y_{f1} & y_{f2} & ... & y_{fp} \end{bmatrix} \begin{bmatrix} 1 & 2 & NaN \\ 3 & 4 & NaN \\ 5 & 6 & 7 \\ 8 & 9 & 10 \\ NaN & 11 & 12 \\ NaN & 13 & 14 \end{bmatrix}$$

As we shall see in the next section, we cannot include NaN-entries in the 3D point-cloud construction. In this section we discuss constructing the point-view matrix.

For the first image pair we insert two rows (in reality this would be four rows; one for each x-coordinates and one for each y-coordinates, but we refer to it as 'one' per frame). For the correspondences between frames 2 and 3 we add a row holding the coordinates of the points in frame 3. The column the coordinates of a point should be placed in are determined by the column of the corresponding point in the row above. If a point is present in the previous frame, but not in the current, we put a NaN in the column of the point with no correspondence. If a point is present in the new frame, but not in the previous, we add a new column.

# 4 Structure From Motion

From a dense point-view matrix the structure $S$ and motion $M$ are as follows:

$$M = U * D^{\frac{1}{2}}$$

$$S = D^{\frac{1}{2}} * V^T$$

Where $U$, $D$ and $V$ are obtained by singular value decomposition.
As mentioned before, the matrix is likely to be sparse. We need to parse through the matrix, grabbing dense submatrices, constructing their 3D point-clouds and merge them into a final model. We walk through an example of parsing. Imagine a point-view matrix:

$$\begin{bmatrix} a & b & c & d & NaN \\ f & g & h & i & NaN \\ k & l & m & n & o \\ p & q & r & s & t \\ NaN & v & w & x & y \end{bmatrix}$$

Every element is in truth two elements; the x- and y-coordinate of the point. The first dense submatrix it would find is:

$$\begin{bmatrix} a & b & c & d \\ f & g & h & i \\ k & l & m & n \\ p & q & r & s \end{bmatrix}$$

From which we can construct a pointcloud of 4 points. We trim the pointview matrix by removing an amount of rows dictated by the NaN-entries that constrained the submatrix' width. Likewise, we remove an amount of columns dictated by the NaN-entries that constrained the height of the submatrix. We're left with:

$$\begin{bmatrix} l & m & n & o \\ q & r & s & t \\ v & w & x & y \end{bmatrix}$$

From which we can again construct a point-cloud of 4 points. In order to merge the pointclouds of both submatrices we need an overlap of at least 3 points (to constrain all degrees of freedom). This condition is met in this case, because the points of the 2nd, 3rd and 4th column are present in both clouds. This parsing continues untill the point-view matrix is completely trimmed away.

## 4.1 Removing Affine Ambiguity

Because we don't know the intrinsic parameters of the camera, we are dealing with some ambiguity. Imagine that two identical photographs could be made if we take a picture of two objects that are an affine transformation of each other, if the intrinsic parameters of both cameras have the inverse relation.

$$D = MS = (MC^{-1})(CS)$$

We try to find the matrix $C$ that emulates our camera to be an orthographic camera; one that does not cause *any* affine deformation. The projection matrix of a camera that performs an affine deformation is as follows.

$$A = \begin{bmatrix} a & b & c & t_1 \\ d & e & d & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It maps a real-world homogeneous coordinate vector $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}$ to camera coordinates $\begin{bmatrix} x & y & 1 \end{bmatrix}$. We ignore the translation part because we center the data. An orthographic camera does not perform any deformation and it's intrinsic matrix looks like this:

$$O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Observe that $OO^T = I$. We are looking for a matrix C that transforms the affine matrix A to an orthographic one.

$$ACC^T A^T = I$$

We use this constraint on the motion matrices of the available frames and solve for C. We apply this transformation to the motion matrix $M$ and its inverse to the shape matrix $S$. We do this for every dense submatrix within the sparse point-view matrix before merging the points to the total pointcloud using procrustes.

To properly visualise the effect of affine disambiguation we use a method that will be introduced in the next section: t-densification.

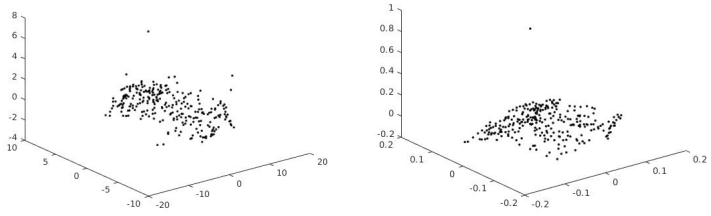

Figure 3: Effect of affine ambiguity removal. Tested on a sparse point-view matrix obtained with t-Densification t=8 and SIFT-threshold=4

# 5 Optimisations

Directly performing the steps as described above will result in a rather sparse pointview matrix and consequently a poor 3D model. In this chapter we discuss the effects of various optimisation methods that aim to improve the 3D reconstruction. The final chapter discusses ideas for optimisation that we have not tried out first-hand.

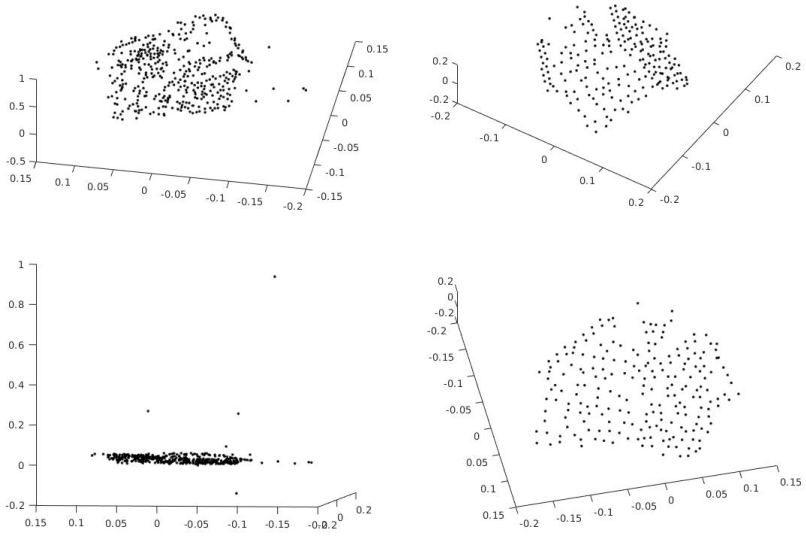

Figure 4: Unoptimised Structure From Motion(left) and SFM on pre-computed point-view matrix(right)

A dense, high-quality pre-computed point-view matrix is available to us. Visualising it will give us an indication of what quality to expect as an upper bound.

The sparse pvm has been constructed with SIFT-keypoint discriptors matches with threshold 4.

All algorithms discussed in this chapter include affine disambiguation in their process.

## 5.1 Geometric Outlier Removal

A failure mode we see the pointcloud of of figure 4 is that there is a point far outside the cloud that is not supposed to be there. We test whether this is due to the inclusion of background by removing outliers from the 2D frames. This is motivated by the observation that most keypoints are detected on the house and an outlier would be a point on the background. In each frame we determine the mean and standard deviation distance to the nearest neighbour and remove points whose distance to the nearest neighbour is more than two standard deviations above the mean. Performing geometric outlier removal takes quite some time, but has no notable benefit.

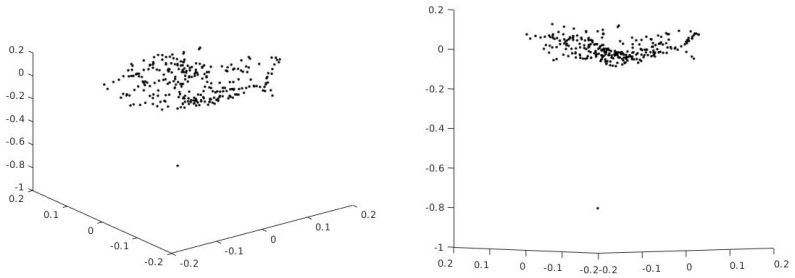

Figure 5: 3D model obtained by removing geometric outliers from 2D frames. t=4

## 5.2 Correspondence Outlier Removal

An outlier on the background of the image may still be a helpful point in determining a 3D model if it is reliably found in many frames. That is, the point must have correct correspondences throughout multiple frames. We filter out bad correspondences by estimating the fundamental matrix and keeping only the correspondences whose Sampson distance is below the threshold of .025. These are the points considered 'inliers' in fundamental matrix estimation. This also does not appear to help the 3D model, and the pvm is actually sparser for it.

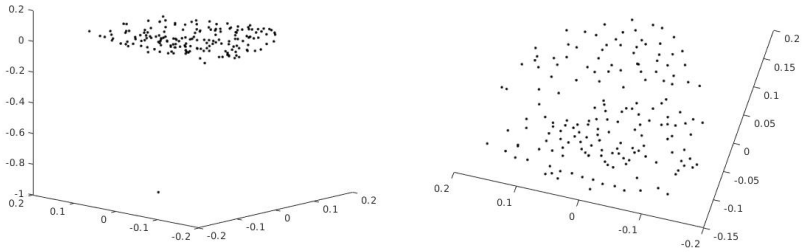

Figure 6: 3D model obtained by removing bad correspondences via sampson distance after fundamental matrix estimation

## 5.3 Outlier removal by SIFT-thresholding

An easy outlier removal method is to set the confidence threshold for SIFT keypoint matching higher. This effectively only labels two points corresponding if the algorithm is very sure of the correspondence. Instead of 4, the threshold is set to 14. Though most noisy points are moved, the one extreme outlier is still not been filtered out, leaving the rest of the 3D model almost in a plane. For the following algorithms we keep the threshold at 14.

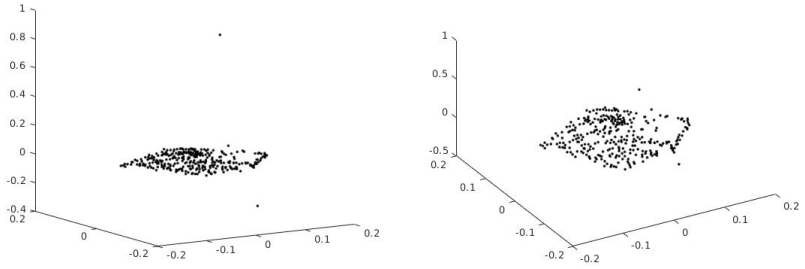

Figure 7: 3D model obtained by only keeping correspondences with high confidence

## 5.4 "Superdense" Densification

The point-view matrix can be densified by only keeping those points that are visible in all frames. This would only work for a set of frames that describe only part of an object like in this assignment (360-degree view cannot be done). Though the 3D model does not appear to get trapped in a 2D plane anymore, the resulting amount of points is too few to present a proper model. Ironically, densifying the point-view matrix sparsifies the 3D model.

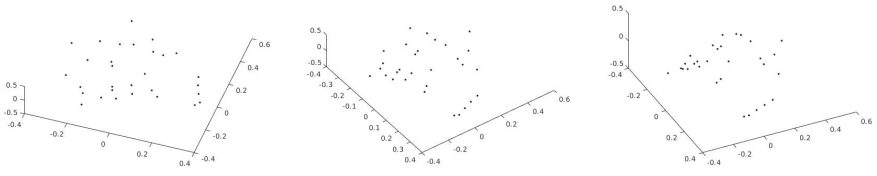

Figure 8: "Superdense"-densification

## 5.5 "t-Densification"

Points are removed if they are present in fewer than t-frames. If $t$ is set to a low value this would help remove spurious correspondences that hamper parsing of the sparse point-view matrix. If $t$ is set to a high value this corresponds to only keeping those points that are present in nearly all of the frames. A maximum value of $t$ would effectively be the "superdense" densification. A negative value for $t$ means an amount below the maximum value. Indeed, Figure 10 looks much like Figure 8. We feel the 3D model of Figure 9 is the best in this work.

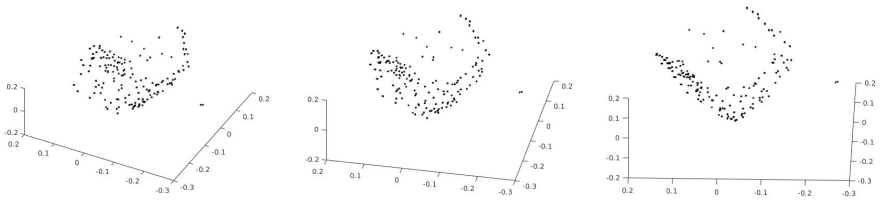

Figure 9: t-Densification for t=8

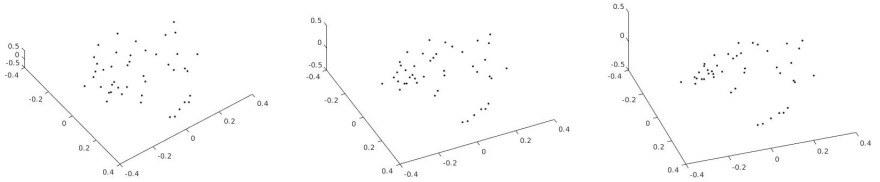

Figure 10: t-Densification for t=-8

## 5.6 Quantitative performance of densification methods

We test the above implementations of point-view matrix densification. Visualisations of matrices with a density ¿.9 will be omitted, because they are largely white patches.

| Method | pvm-density | n_points |
|---|---|---|
| sparse | 0.13 | 3315 |
| Geo outlier removal | .13 | 3315 |
| RANSAC outlier removal | .07 | 3093 |
| SIFT outlier removal | .11 | 2984 |
| t=3 | 0.24 | 1077 |
| t=8 | 0.43 | 427 |
| t=-8 | 0.97 | 44 |
| t=-3 | 0.996 | 33 |
| superdense | 1.0 | 30 |
| pre-computed | 1.0 | 215 |

Table 2: Performance of densification methods

Figure 11: sparse PVM

Figure 12: PVM after RANSAC outlier removal

Figure 13: PVM after geometric outlier removal

Figure 14: PVM for t-densification t=3

Figure 15: PVM for t-densification t=8

# 6 Ideas

These are ideas that might improve the performance, but have not been implemented and tested.

## 6.1 Exhaustive PVM Construction

In the current implementation, we construct the point-view matrix by correspondences between consecutive frames. If many points are shared between frame 1 and 3, but not in frame 2, this would not be captured. We construct the point-view matrix from correspondences between frame 1 and 2, frame 1 and 3 up to $n$ and then from correspondences between 2 and 3, 2 and 4, and so on. This would be a very exhaustive process. Likely to take much time, but also likely to improve performance.

### 6.2 Interpolated point merging

In the current implementation of pointcloud-merging the points in the cloud extracted from each dense submatrix are added to the full cloud, even the overlapping points. Points in the overlap of dense submatrices should have exactly the same relative 3D position, but due to noise we can expect there to be some small difference between them. We could consider adding a point that is the average between the location according to the current model and according to the new added submatrix.

### 6.3 n-Densification

like t-Densificiation, we remove rows that have many NaN-entries in them. We could do this iteratively, untill we're left with $n$ maximally dense points.

### 6.4 Iterative Purification Densification

After each trim of the pvm, we could consider 'purifying' the resulting pvm; removing those columns that contain only NaN values. This would require us to keep explicit track of the points we remove, so as not to confuse which points need to be merged. In particular we need to be smart about assigning a value to the variable 'idx'.

### 6.5 Dense-SIFT

In the current implementation we extract SIFT-features at salient points only. It might be worth experimenting with dense sampling. However, this would increase training time dramatically.

## 7 Appendix: Self-evaluation

Bram programmed epipolar line estimation and the eight-point algorithm. Zvonimir programmed chaining the for point-view matrix construction. Zvonimir programmed structure and motion from a dense block. Bram programmed parsing for dense submatrices within a sparse matrix and merging the pointclouds. Zvonimir helped with the stitching function. Bram wrote the report.