IN4393-16 - Computer Vision
Final Project

April 14, 2019

Group 33
P.M. van der Burg - 4292286
K. Wendel - 4331362

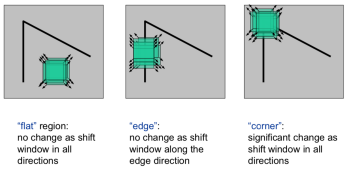
Contents

1 Interest Points and Correspondences

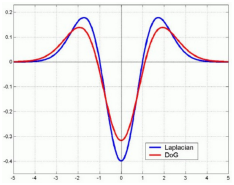
To be able to match two or more images, one has to find correspondences between these images. These correspondences can be found by finding interest points on two or more images, after which one tries to match these specific between those images.
For our implementation, two methods are used to find interest points: Harris corner detector and the *'VLfeat'*-toolbox.

1.1 Harris corner detector

The Harris corner detector localizes points with a corner shape by shifting a small window window over the image and measuring its response to intensity change. When this response is large in both directions, the point gets detected as a corner, as is illustrated in Figure 1a.



(a) Basic working principle of the Harris corner detector.



(b) Laplacian vs Difference of Gaussians

Figure 1: Harris corner detection and the Laplacian vs Difference of Gaussians

1.1.1 Difference of Gaussian

To be able to find corners, we first use blob detection to narrow our search field, which is often done by convolving the image with a scale normalized 2D Laplacian filter (2nd derivative of the Gaussian). We can also quite accurately approximate this second derivative by taking the Difference of Gaussians (DoG) (See Figure 1b). The responses are calculated over multiple image scales with multiple σ values for the Gaussian function, which is a technique used to make the response invariant to scaling. For every detected point location, the highest response is kept with the corresponding σ. This is all done in our function `DoG_final.m` which is in the attached function zip archive.

1.1.2 Our own Harris implementation

The blob points' coordinates from Section 1.1.1, along with the corresponding σ, are then passed to the `harris_final.m` function. Here, the image is first convolved with a derivative of the Gaussian kernel, after which all unique elements of the second moment matrix M are calculated for every pixel, which are I_x^2, $I_x I_y$ and I_y^2. Then, these M matrices are filtered with a Gaussian filter window per pixel. The formula for this filtered M is given in Equation 2, in which $w(x, y)$ is the filter window function.

$$M = \sum w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{1}$$

This filtered output is analysed on the specific locations of the interest points found with the *DoG* method. If both eigenvalues of M are large, a corner is detected. To prevent having to calculate all eigenvalues (which is computationally expensive), we use the following formula:

$$R = det(M) - k(trace(M))^2 \tag{2}$$

Where k is an empirical constant.
If the 'cornerness' R is above some threshold, it is acknowledged as a corner point.
These corner's coordinates, along with their corresponding *sigma* values, are then used to build descriptors to describe characteristics of an interest point, which is done in Section 1.2.

1.2 VL_Feat toolbox

Our implementation uses the VL_Feat toolbox to build descriptors in two ways, which are described in 1.2.1 and 1.2.2.

1.2.1 Custom frames

The descriptors are built using the function `vl_sift` with the Harris Corners as *custom frames* (SIFT stands for Scale Invariant Feature Transform). Then, a 16x16 window is placed around each keypoint and subdivided into a 4x4 subwindows, as can be seen in Figure 2. Then, the gradient magnitude and orientation for each of the 16 pixels in 4x4 subwindow are calculated and binned into 8 bins (45 degrees each), in which each pixel contributes an amount proportional to its magnituge. This results in 16 histograms (8 bins each) for each descriptor, resulting in 128 values (see Figure 2), which are then placed in a vector.

Using our own custom frames, no keypoint orientation is taken into account, which is a potential drawback of this method. A sample result can be seen in Figure 3.

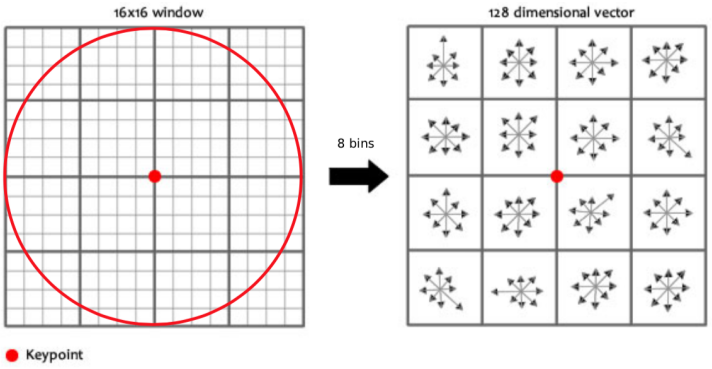


Figure 2: The general idea of a SIFT descriptor

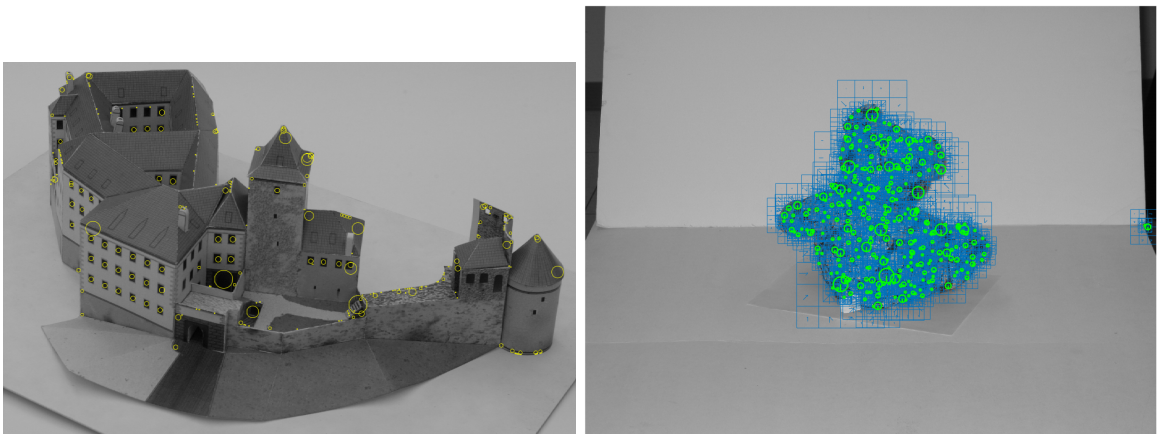


Figure 3: Example image with the results of our Harris corner detector. On the left only the detected corners are plotted. The size of the circles is proportional to the scale at which the point are detected. On the right the SIFT descriptors are also plotted.

1.2.2 VL native point detector

With this method, the `vl_sift` function is used natively, which activates the function's own interest point detector, and can be seen in `load_features_final.m`. This method is similar to our own implementation, as it also uses Difference of Gaussians with a fluctuating smoothing parameter over multiple image scales

to find interest points. Next, the interest points are analyzed in the same way as in the previous section. The biggest difference with the Harris implementation is that this method calculates an orientation for every keypoint, which makes this method significantly more resistant to the rotation of an image. This difference is visualized in Figure 4. Furthermore, this method allows easy adjustment of parameters to optimize the results.

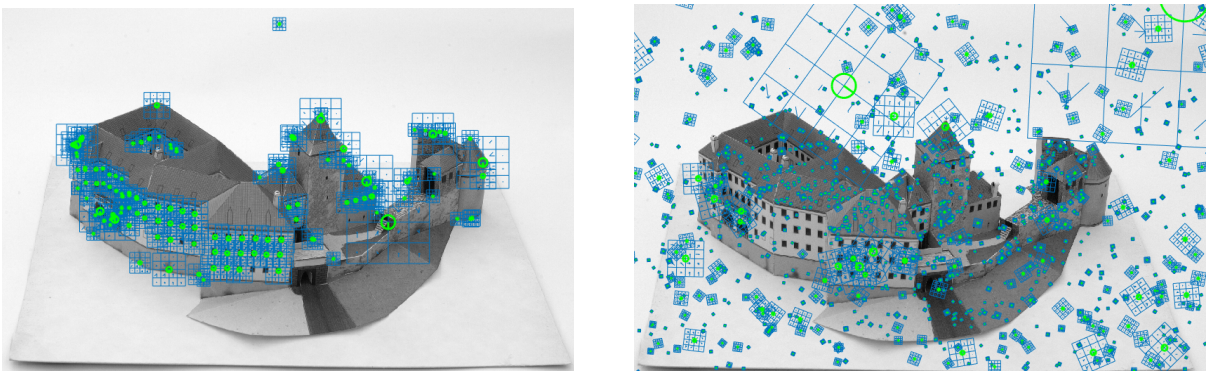


Figure 4: Example image with SIFT descriptors based on our own Harris interest points on the left and a small sample from the native `vl_sift` function on the right. The size of the blocks is proportional to the scale at which the point is detected and the orientation represents the keypoint orientation

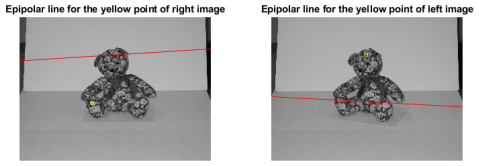
2 Normalized 8-point RANSAC

The previous steps gave us interest points and correspondences/matches in each of the consecutive images. In our case, these images are either of the teddybear or castle series. Both series of images are created by keeping the camera at a fixed position while rotating the object slightly for each picture. Note that this is the same as keeping the object of interest at a fixed position and rotating and translating the camera around the object.

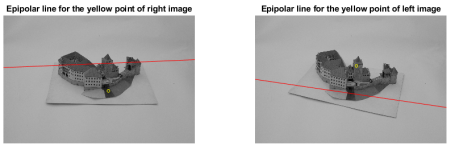
It is important to note here that the camera is moving for each picture, thus its center of projection changes. This makes homography estimation unsuitable, as it assume that the center of projection is the same for each picture (no camera translation) and will find the homography transformation between the point correspondences. The 8-point RANSAC algorithm that uses epipolar geometry is more suitable, as there we assume that the cameras did move and find the transformation between the two images.

In epipolar geometry, we use the epipolar constraint that a image point in the first view must occur in the second view on the line carved out by a plane connecting the world point and the optical centers. The finding of correspondences in images is now reduced to a one dimensional problem where we search along the epipolar line. This constraint is formulated as $x'_T F x = 0$, where x and x' are points in the first and second image respectively, and F is the fundamental matrix. This F will be estimated using RANSAC, where 8 correspondences are randomly selected and used to calculate the transformation matrix F for those points. Then the points are transformed and their perpendicular error between the point and the epipolar line is computed, which is also called the Sampson distance. Points that have a Sampson distance within a threshold will be considered inliers. The normalized 8-point algorithm will be used as it is considered more stable and gives better results [1].

The RANSAC algorithm is repeated until a reasonable amount of inliers is found with a high probability. For the bear image-set, the inlier percentage is high (around 70-80%) with a Sampson distance threshold of 1. The castle set is somewhat harder as inlier percentages around 50% were founded with a threshold of 10. The epipolar intermediate results are shown in Figure 5 and Figure 6, and show that the estimated fundamental matrix is as expected. A subset of the inliers that were founded are shown in Figure 7. This process of finding good matches is repeated in the final 3D reconstruction for every pair of consecutive images.

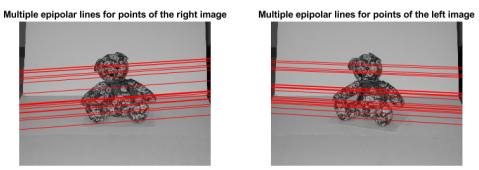


(a) An epipolar line and their corresponding point in the other image for the first two images of the bear set.

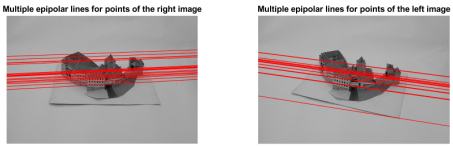


(b) An epipolar line and their corresponding point in the other image for the first two images of the castle set.

Figure 5: Visualization of the one epipolar line and the corresponding point. On the left side are the bear images and on the right side the castle images.

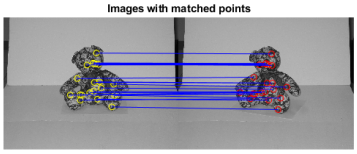


(a) Visualisation of multiple epipolar lines.

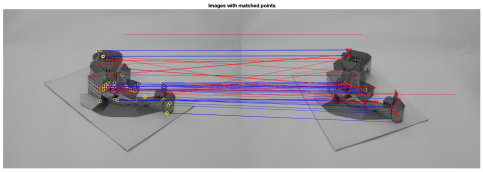


(b) Visualisation of multiple epipolar lines.

Figure 6: Visualization of the multiple epipolar lines. On the left side are the bear images and on the right side the castle images.



(a) 20 Randomly selected matches of the 2974 inliers



(b) 20 Randomly selected matches of the 811 inliers.

Figure 7: Visualization of a subset of matches that where found by the using the 8-point normalized RANSAC algorithm.

3 Chaining

After the matches are found, we want to find points that remain visible in consecutive frames to be able to keep track of them. The way we implemented this is by constructing a *Point View (PV) matrix*. This matrix will contain all matched points over all consecutive images. An example of what a PV matrix looks like can be seen in Figure 8. Every row represents one of the views and the columns represent point matches between images. The elements of the matrix are black if the point was visible in the image.

Figure 8: Graphical example of what a Point View matrix can look like.

For our implementation, we use the matches found by the RANSAC 8-point algorithm in Section 2.

3.1 Step-by-step explanation

1. First, all correspondences between the first and second image are put in the first two rows of the PV matrix. These values are the indices of the matched points for an image.
2. Then, we look at the matches between the second and third image. We add the index values to the third row of the PV matrix which are already present in the second row. These values are points which remain visible over the first three frames.
3. Next, all points which match between frame two and three, but not between frame one and two, are appended to the second and third row of the PV matrix after the existing data.
4. Repeat steps 2 and 3 for all remaining frames.
5. Finally, the points of the last frame are matched with the first frame.

The resulting PV matrices for the teddybear and castle images can be seen in Figure 9 and Figure 10 respectively.

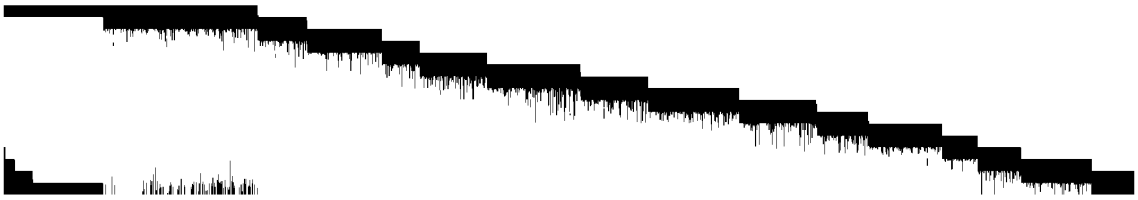


Figure 9: Resulting 16x10933 PV matrix from the teddybear images.

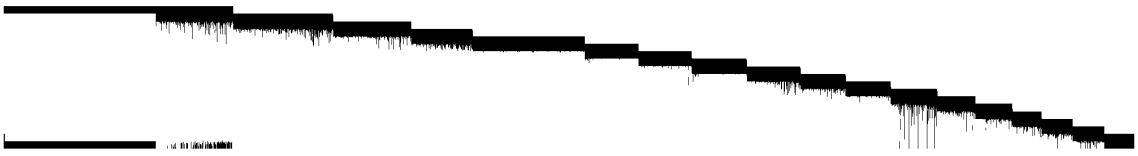


Figure 10: Resulting 19x22424 PV matrix from the castle images.

4 Stitching

In the chaining step the point-matrix was constructed that contains all of the observed interest points and their occurrences in the different frames. The next step is to use this information to estimate the corresponding 3D coordinates and stitch all of the points together.

4.1 Measurement matrix

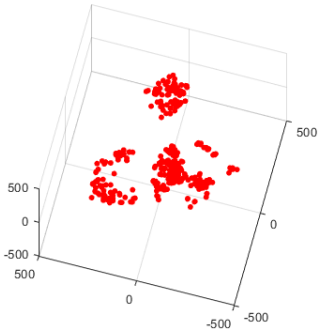
The point-view matrix consists of row and columns that indicate the image/frame and the corresponding interest point respectively. So when a interest point was observed in a specific frame, that specific row and column will contain a value. Note that this matrix is sparse as not all of the points are visible in all views. To deal with missing data and the sparsity, a dense block in the point view matrix is taken of three images where the interest points were observed in all images. Each separate coordinate (x and y) of these interest points will an row in the measurement matrix. The final size of the measurement matrix is thus $2F \times P$ with P points and F frames.

4.2 Estimate 3D coordinates with Tomasi-Kanade Factorization

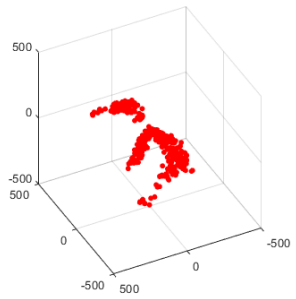
Next, the structure and camera motion can be recovered from the measurement matrix using Tomasi-Kanade factorization. This factorization uses the fact that the rank of the measurement matrix is at most 3 [2]. The following steps are taken in the factorization of the measurement matrix D:

1. Center the image points by subtracting the centroid of the points in the frame
2. Do singular value decomposition of D: $D = UWV'$
3. Enforce the rank constrain by only keep the top 3 rows or columns of the SVD decompositions. So $W_3 =$ the three highest values, $U_3 =$ the first three columns of U, and $V_3' =$ first three rows of V
4. Now the motion M and structure S can be computed by: $M = U\sqrt{W}$ and $S = \sqrt{W}V'$

These solutions M and S still contain ambiguity as they are not an unique solutions. How this can be eliminated is explained in section 5. Some examples of estimated structure are shown in Figure 11.



(a) Estimated structure from the first bear image. The reconstruction shows the front of the bear was estimated correctly.



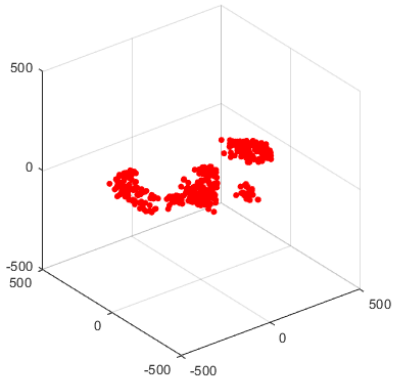
(b) Estimated structure from the sixth bear image which is a side view. The reconstruction shows that one side of the bear was estimated correctly.

Figure 11: The estimated structure using Tomasi-Kanade factorization

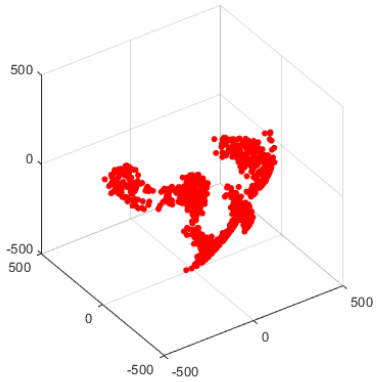
4.3 Iterative stitching

The previous step yielded multiple estimations of the 3D locations of the points as a series of point clouds. Next, these clouds are going to be stitched together to get the final 3D reconstruction in the following way. Firstly, the reconstruction is initialized with all of the points of the first point cloud. Then

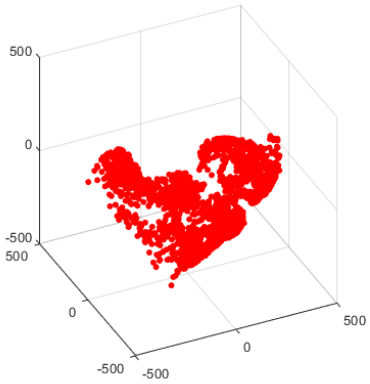
for each point cloud, the shared points are founded between the reconstruction and the point cloud. A procrustes analysis is then performed to find the optimal transformation between these shared points as they belong to the same 3D locations. The founded transformation is then used to transform all other points of the point cloud, and these points are added to the final reconstruction.
Intermediate results of stitching are given in Figure 12. It can be seen that the reconstruction begins to take the shape of the bear after stitching a few point clouds.



(a) The stitched point cloud after stitching two point clouds. Only the front of the bear is reconstructed.



(b) The stitched point cloud after stitching six point clouds. The reconstruction begins to take the form of the bear.



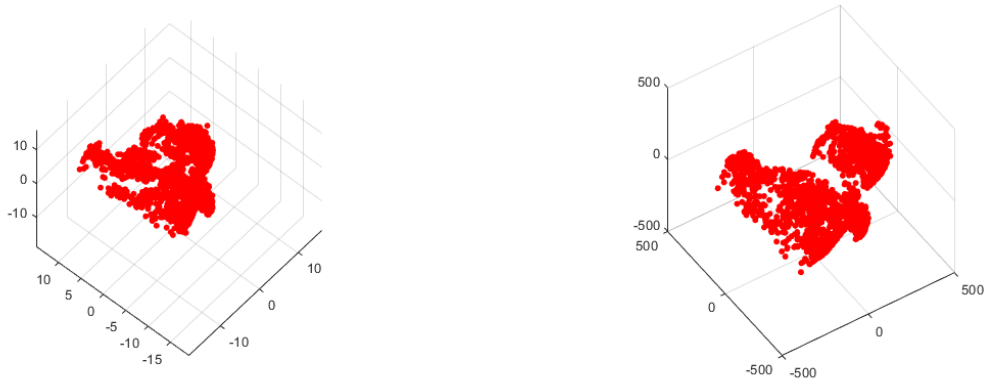
(c) The stitched point cloud after stitching all point clouds. The reconstruction clearly denotes a bear.

Figure 12: Intermediate results during stitching of the final 3D reconstruction.

5 Affine Ambiguity

One of the problems that occurs with structure from motion is ambiguity. When the recovered scene is transformed with a transformation Q and the inverse of this transformation is applied to the camera motion, the same image will be obtained. The recovered solution from structure from motion is thus not unique and the real solution is a linear transformation of it. This means that if there are no constrains on the recovered camera matrices, the reconstruction will be projective. The affine ambiguity in this reconstruction can be removed if additional constraints are added. This can be achieved when we assume an orthographic projection where the image axes are perpendicular and the scale is 1. These constraints are formulated as followed: $a_1 * a_2 = 0$ and $|a_1|^2 = |a_2|^2 = 1$ where a_1 and a_2 are the image axes. This gives us $3m$ equations in the form of $A_i L A_i^T = I$, where L can be recovered with a Least Square solution. The affine ambiguity can then be removed by doing a Cholesky decomposition $L = CC^T$, and update M and S with $M = MC$ and $S = C^{-1}S$.

To illustrate the effect of eliminating affine ambiguity two reconstruction are made. One uses the constraints given above while the other has no constraints, as shown in Figure 13. It can be seen that the reconstruction where the ambiguity is eliminated visual looks better visually and is scaled better.



(a) 3D Reconstruction of the bear with affine ambiguity.

(b) 3D Reconstruction of the bear without affine ambiguity.

Figure 13: Two reconstructions of the bear: the left picture has affine ambiguity and for the right one this ambiguity was eliminated. Notice the scale of the axes in the left picture.

6 3D Model

6.1 3D Reconstructions

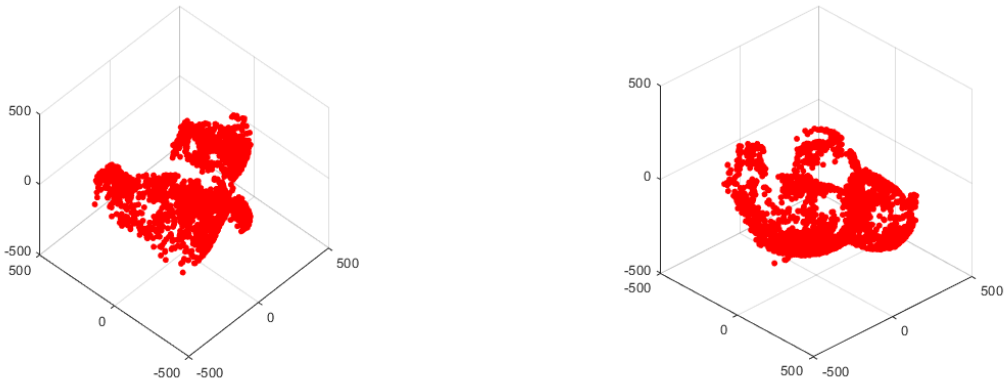
Below are the reconstructions given for the bear (Figure 14) and the castle (Figure 15). The bear was created using the provided features and a RANSAC threshold of 1. The castle was created with features extracted with `vl_sift` with 5 levels per octave, a peak threshold of 0.15, and a edge threshold of 15.

6.2 3D Reconstructions (Harris)

Below are the reconstructions given for the bear (Figure 16). The bear was created using our own Harris implementation and a RANSAC threshold of 1.

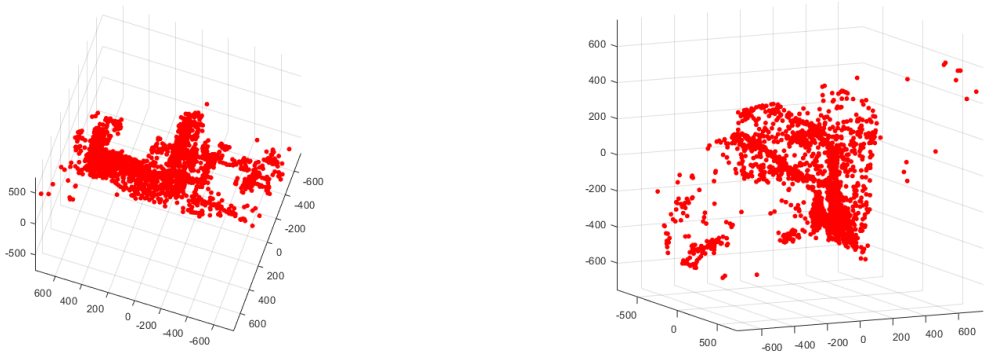
6.3 3D Surface rendering

The result of the 3D reconstruction can be improved by projecting every point onto the main view, which is the first image/the front view in our reconstructions, and use these projected points to find the color or interpolated color of related points. Results of the bear image are shown in Figure 18.



(a) (b)

Figure 14: The final reconstructions of the bear image set.

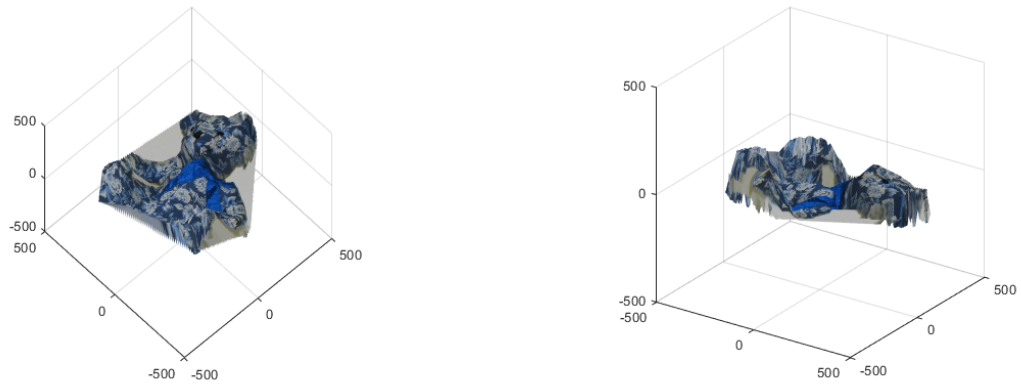


(a) (b)

Figure 15: The final reconstructions of the castle image set. Notice the larger tower in the middle and the smaller tower on the right.

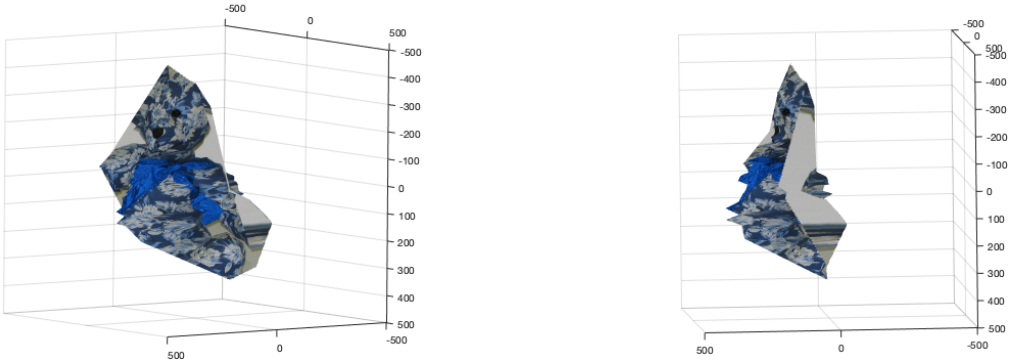
(a) (b)

Figure 16: The pointcloud reconstructions of the bear image set using our own Harris implementation.



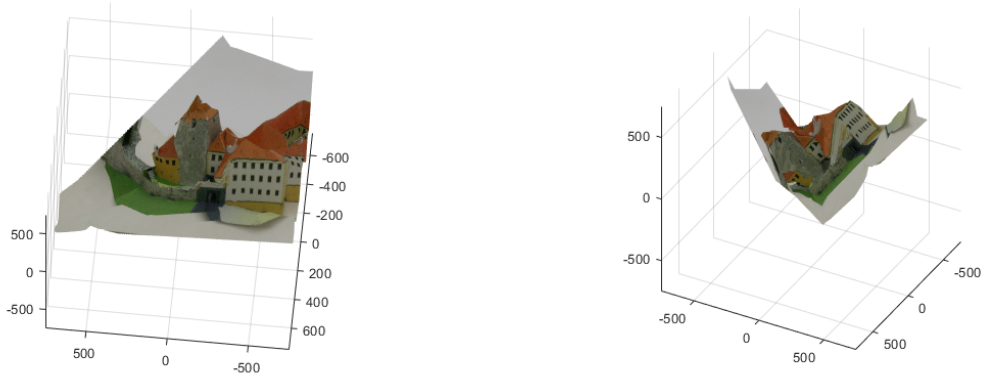
(a) (b)

Figure 17: The reconstruction of the bear image set including the surface rendering



(a) (b)

Figure 18: The reconstruction of the bear image set using the Harris implementation



(a) (b)

Figure 19: The reconstruction of the castle image set including the surface rendering

References

[1] Richard I Hartley. "In defence of the 8-point algorithm". In: *Proceedings of IEEE international conference on computer vision*. IEEE. 1995, pp. 1064–1070.

[2] Carlo Tomasi and Takeo Kanade. "Shape and motion from image streams under orthography: a factorization method". In: *International Journal of Computer Vision* 9.2 (1992), pp. 137–154.