3D reconstruction of a model castle and a teddy bear

Georg Loibl, *ME-student* and Martijn van Wezel, *CE-student*
Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
P.O. Box 5031, 2600 GA Delft, The Netherlands
(g.loibl, m.j.a.vanwezel)@student.tudelft.nl

***Abstract*—In 3D reconstruction of a Model Castle and a Teddy Bear the reconstruction is made from several images from different perspectives around the object. To create the 3D model, feature matching is used to identify corresponding features in sequential images. The feature matching method is based on a Harris corner detector [1] and SIFT [2]. After the features have been detected, the 8-point RANSAC algorithm filters out outliers within the previously computed feature matches by computing the optimal model using only inliers. To keep track of the features that are visible from one frame to the other, the point-view matrix is created. After estimating the 3D coordinates of the object using the Tomasi-Kanade factorization and eliminating the affine ambiguity of the transformation, the point clouds of each image are merged. For the final visualization colors are added to the 3D point cloud using surface rendering (Figure 6) and color dots (Figure 7).**

***Keywords*—Harris, SIFT, RANSAC, SfM, SVD**

I. Introduction

This paper is done as a coursework for the course IN4393 Computer Vision. The goal is the 3D reconstruction of a model castle as well as a teddy bear using a sequence of images made from different perspectives. The total project work is divided into five sections: Feature extraction and matching, RANSAC, Chaining, stitching and the 3D visualization.

II. Feature extraction and matching

The first step in the 3D reconstruction pipeline is the extraction of feature points in the images and the matching between these points. This is needed to understand which points in one image correspond to which points in the following image. In our code the user may choose at the beginning of the main script $FINAL\_Project\_CV$ whether to extract features with a self made Harris detector (plus SIFT to compute the descriptors) or to load the provided feature coordinates and descriptors. The first approach is described in the following two subsections.

Harris corner detection

The Harris corner detection is used to find feature points in an image. However, before using the detector, the scale of the local feature points has to be found. This is realized in the function $DoG.m$ which uses the Difference of Gaussian approximation of the Laplacian [3]. The resulting matrix with the size $n \times 3$ contains n found locations described by the corresponding column and row of the image and its scaling value. Within the function several parameters, such as the number of scaling levels or octaves, can be adjusted. For the purpose of this 3D reconstruction the number of scaling levels is set to 7 and the threshold to test for flatness is set to 0.7.
Now that the interest points from the Laplacian approximation are known, the Harris corner detection is the next step. First, the structure tensor M is computed using the image derivatives I_x and I_y.

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

The structure tensor M in equation 1 is then filtered with a Gaussian kernel. Weather a point is a corner or not depends on its eigenvalues λ_1 and λ_2. When $\lambda_1 >> \lambda_2$ or $\lambda_2 >> \lambda_1$ it is an edge. When λ_1 and λ_2 are both large the feature point is likely to be a corner. When both eigenvalues are small it is probably a flat region.
In order to increase the computational efficiency, the algorithm avoids the direct calculation of the eigenvalues and instead bypasses it by computing the determinant and the trace of the structure tensor. As a reminder, the determinant of M is $det(M) = \lambda_1 * \lambda_2$ and the trace of M is $trace(M) = \lambda_1 + \lambda_2$. Using these the measure of corner response R is calculated as follows:

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

The parameter k is an empirical constant that is set to 0.04. The analyzed point is a corner if the value of R is larger than a specified threshold otherwise the point is either an edge or a part of a flat region and thus, it can be discarded. The value of the threshold is the maximum of all values in the resulting R matrix times 0.005. The standard value of 0.3 leads to less feature points and was therefore significantly decreased.
The results of the self implemented Harris feature detector for the teddy bear and the model castle are shown in Figure 1. The red circles are the detected feature points in the image with their radii corresponding to the respective scaling value.

SIFT points

The scale-invariant feature transform (SIFT) is a feature detection algorithm used to find the interest points in an image. However, as the feature points have already been computed with the self implemented Harris detector, the function $vl\_sift.m$ is used to retrieve the descriptors of each feature point. For this purpose, the function takes the

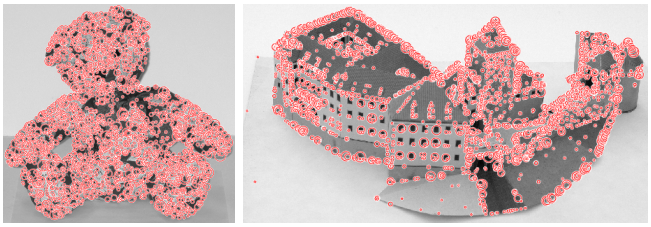

Fig. 1: Extracted feature points with the self implemented Harris corner detector

image as well as the previously detected feature points as input and returns the corresponding descriptors (in addition to the coordinates of the feature points). The detected feature points are given to the function within a frames matrix containing the location (row/column), the scaling value as well as the orientation. At this moment there is no orientation of the feature points yet. Thus, it is possible to simply set all orientations to zero. Doing so, the SIFT implementation automatically calculates the corresponding orientations.

Feature matching

After detecting valid feature points and their corresponding descriptors, the next step is to find the matches between two frames using the function $vl\_ubcmatch.m$. The function uses the previously calculated descriptors of the two frames as well as a threshold parameter as input. As mentioned in the function description, a descriptor is matched to another descriptor only if the distance of these two descriptors multiplied by the specified threshold is not greater than the distance of the first descriptor to all other descriptors.

For the presented example, the resulting point cloud had the highest quality with a threshold set to 1.15.

Figure 1 shows all the computed matches between the first and the second frame of the teddy bear sequence. It is obvious, that the model still contains many outliers.

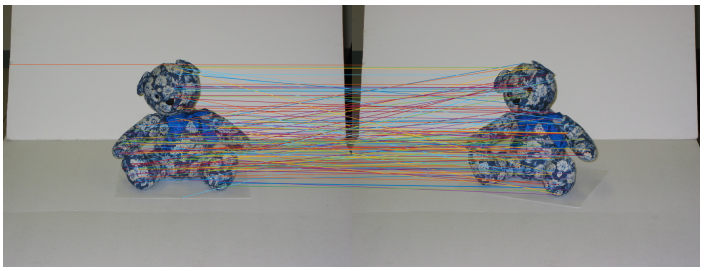

Fig. 2: Matched feature points between first and second frame of the teddy bear sequence

III. RANSAC

In contrast to the Least Squares method, RANSAC is an iterative method that avoids the impact of outliers by finding the optimal model using only inliers. To find the best model for the previously computed matches, the normalized 8-point RANSAC algorithm is used.

The computed matches are now used as inputs to the core function of the implemented RANSAC algorithm $estimateFundamentalMatrix.m$. In order to estimate the fundamental matrix, a linear system of 8 equations has to be solved. To do so, eight point pairs that are randomly selected are needed. To improve the conditioning of the problem and hence the stability of the whole algorithm, a normalization of the point correspondences is performed. After that, the first estimation of the fundamental matrix is computed and de-normalized. Next, inliers are identified by computing the perpendicular errors between the points and the epipolar lines in each image using the Sampson error. An inlier is found if the computed Sampson error is smaller than a pre-defined threshold, which was set to 35 in the presented examples. If the model has more inliers than the best model found so far, it is saved as the new best fit.

The number of total iterations is dependant on several parameters, such as the probability t that the algorithm fails (0.001), the number of found inliers, the number of total pairs as well as the number of data points P required to fit the model (8 points):

$$iterations = \frac{log(t)}{log(1 - \left(\frac{\#inliers}{\#total\ pairs}\right)^P)} \tag{3}$$

The formula is recalculated in each iteration and thus the number of total iterations is dynamically adapted. However, it is useful to set a minimum number of iterations to avoid that the algorithm terminates too early. Moreover, during the optimization process of the RANSAC parameters it turned out to be useful to introduce a maximum number of iterations to increase computational efficiency while the quality of the final point cloud seemed to be insignificantly effected. Figure 3 shows the first 60 inliers (blue) and the first 60 outliers (red) of the first and second frame.

IV. Chaining

As a single point is unlikely to remain visible in all the consecutive frames, the feature points that are visible from one frame to another need to be tracked. To do so, the point-view matrix (PV) is created, which saves the coordinates of the corresponding points between two consecutive frames. In detail, the code loops through each frame of the video sequence and tries to find points in the next frame that have already been visible in the current one by using the function $intersect.m$. After that, the point view matrix is extended by the points in the next frame that have not been visible in the current one yet. To determine these points, the function $setdiff.m$ can be used. At the end of the loop it compares the last frame with the very first one to close the circle.

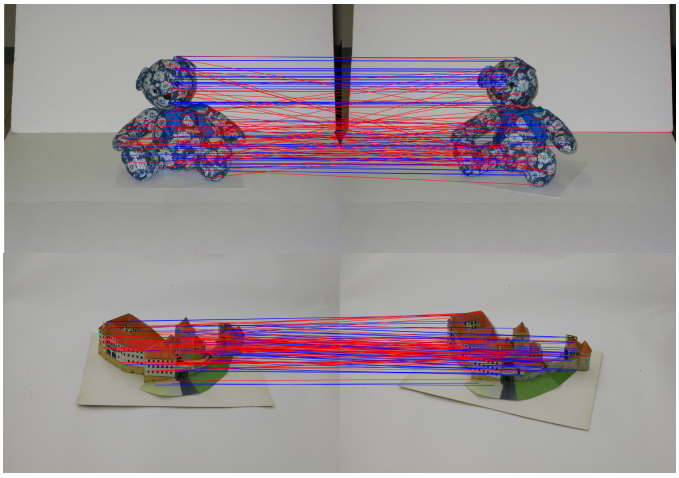

Fig. 3: Result after running RANSAC: inliers are blue, filtered outliers are red (only first 120 points are shown)

V. Stitching

Before the point clouds of each image can be stitched to each other, the point cloud of each single image has to be computed. As first step, the measurement matrix X is created by taking blocks of the point-view matrix that are composed of three consecutive images. The indices saved in the point-view matrix are then used to obtain the coordinates which are saved in the measurement matrix.
Secondly, we estimate the 3D coordinates of each block using Tomasi- Kanade factorization [4]. In detail, the measurement matrix is first centered by subtracting the centroid of the image points. In order to enforce the rank criteria, the measurement matrix is decomposed in its singular values and then, only the corresponding top three values of the rows/columns of the resulting matrices are taken.

$$[U, W, V] = svd(X)$$
$$U_{new} = U(:, 1:3)$$
$$W_{new} = W(1:3, 1:3)$$
$$V_{new} = V(:, 1:3)$$

The motion matrix M and the shape matrix S are then computed by using the following relation:

$$M = U \cdot W^{0.5}$$
$$S = W^{0.5} \cdot V^T$$

At this point, we have only an affine transformation. To eliminate the affine ambiguity, further constraints must be introduced in order to impose that image axes are perpendicular and their scale is equal to 1. As can be seen in the lecture slides, first the equation has to be solved for L. We do this by making use of the Matlab function $lsqnonlin.m$. The final matrix C is then recovered using the Cholesky decomposition. Finally the motion matrix M as well as the shape matrix S have to be updated:

$$M = MC$$
$$S = C^{-1}S$$

Note that in case the input matrix L for the Cholesky decomposition is not positive definite, the second output value is non-zero and the calculated cloud will not be used in the further procedure.
As a last step the calculated point clouds per image have to be stitched together to form one final merged point cloud. To do so, the first frame is taken as the reference to which all other point clouds are transformed. Then, the optimal transformation between the shared points of the merged point cloud and the point cloud of the single image is calculated using the Matlab function $procrustes.m$. This transformation is then used to transform the new points to fit to the merged point cloud. The result is the merged point cloud of all the images, as can be seen in Figure 5 with the feature points of the self implemented Harris Detector.

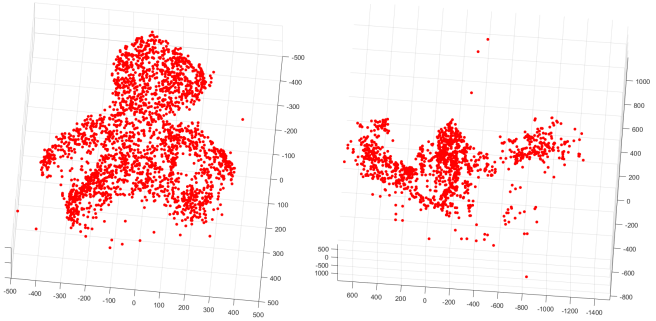

Fig. 4: Merged point clouds after stitching (own feature points used)

When reviewing the point cloud there are some holes and gaps. A reason for this is that all the pictures are approximately taken from the same height and therefore some details might be occluded. This effect is clearly visible in the teddy bear example, see Figure 5. On the one hand, the bottom of the object is not visible because it is covered by the floor, on the other hand the chin of the bear is only partially observed. Thus, no feature points can be found in these parts of the object and consequently the information is missing in the final point cloud. Another reason is that the rotation of the camera around the object might be too great and therefore many feature points can not be tracked in at least three consecutive frames.

VI. 3D Visualization

To visualize the 3D model two different methods are used. To generate the surface one can make use of the function $TriScatteredInterp.m$ or the more recommended function $scatterInterpolant.m$. However, the quality of the output of the latter function seems to be quite poor due to several outliers. This can be solved by using a linear interpolation method and no extrapolation method ('none') as additional input to the function. Figure 6 shows the results of the

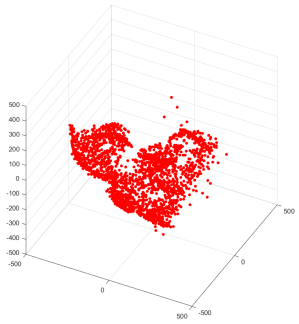

Fig. 5: Holes in the point cloud of the teddy bear due to occlusion effects (bottom and chin area are only partially covered)

visualization method for the teddy bear and the Model Castle.

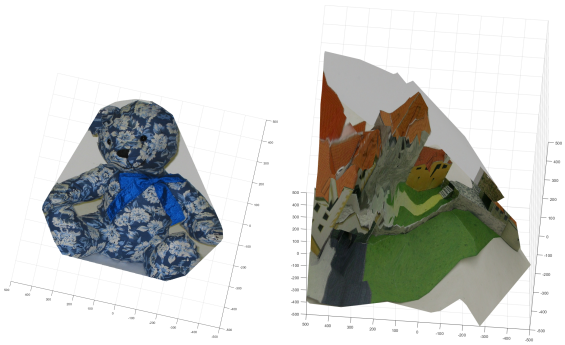

Fig. 6: Surface render of one image on the teddy-bear and point cloud.

For the second visualization method the corresponding RGB value is assigned to each of the 3D points in the point cloud. However, since three frames are used for a 3D point, the average of the three corresponding RGB values is used for the further process. Moreover, one has to pay attention that some frames might not be used to create a point cloud due to a non-positive definite input matrix for the Cholesky decomposition. Furthermore, point clouds might be discarded as the Procrustes analysis needs a certain number of shared points. Figure 7 shows the results of the second visualization method for two different perspectives again for the teddy bear and the model castle. Figure 8 compares the main view of the point cloud with the first frame of the corresponding video sequence.

VII. Conclusion

The 3D reconstruction was successfully implemented for the model castle and the teddy bear. The implementation works with our own Harris corner detector as well as with the provided feature points and descriptors. The code is structured in a way that the user can reconstruct other objects without much efforts by simply adding the new images to a folder and adjusting the directories. Moreover, a second visualization method (other than the one provided) that assigns the RGB value to each feature point was introduced. However, during the implementation the most crucial part was the extraction of feature points as lots of parameters have to be adjusted in order to create a large number of matches of high quality. Furthermore the complete computation (without any saved files, such as the descriptors, the point-view matrix or the matches) can take up to three hours depending on the hardware and chosen parameters.

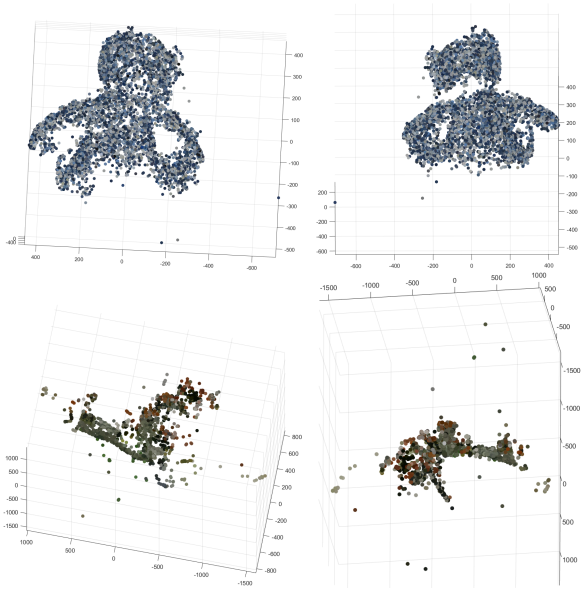

Fig. 7: Results of the second visualization method for different perspectives (front/back) of the teddy bear and the model castle (given features were used here)

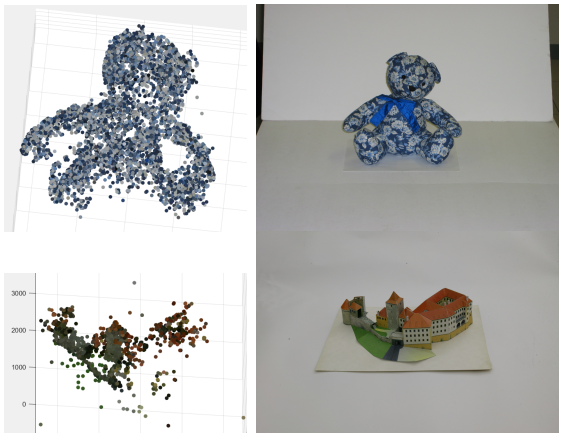

Fig. 8: Comparison between the main view of the point cloud with the first frame of the corresponding video sequence (own features were used here)

VIII. Discussion

In this section, we want to discuss about how to further improve the existing model in terms of quality, computational efficiency and visualization. First, it will be helpful to increase the total number of feature points in order to obtain a denser final point cloud. This can be done by adjusting the parameters of the Harris corner detector, the Difference of Gaussians (DoG) as well as the parameters for SIFT. Furthermore, bundle adjustment can be used to refine the 3D coordinates and the parameters of the motion matrix in order to compensate the increasing errors between the frames.
Secondly, to improve the performance of the algorithm, faster algorithms such as ORB [5] (for SIFT) or LO-RANSAC [6] could be implemented.
One could also decrease the number of maximum RANSAC iterations keeping in mind that it might decrease the quality of the final point cloud. Parallelizing the code, for instance the feature extraction or RANSAC, would highly increase computational efficiency.
Thirdly, as an improvement of the visualization method, vertices between the points in the point cloud could be added to close the gaps and make it appear more like a closed structure.

References

[1] C. Harris and M. Stephens, "A combined corner and edge detector," pp. 23.1–23.6, 1988. doi:10.5244/C.2.23.

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

[3] L. Assirati, N. Rosa, L. Berton, A. Lopes, and O. Bruno, "Performing edge detection by difference of gaussians using q-gaussian kernels," *Journal of Physics Conference Series*, vol. 490, 11 2013.

[4] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: a factorization method," *INTERNATIONAL JOURNAL OF COMPUTER VISION*, vol. 9, no. 2, pp. 137–154, 1992.

[5] K. K. E. Rublee, V. Rabaud and G. Bradski, "Orb: An efficient alternative to sift or surf," pp. 23.1–23.6, 2011. ORB.

[6] O. Chum, J. Matas, and J. Kittler, "Locally optimized ransac," pp. 236–243, 2003.