IN4393 Computer Vision

Group 36

Zheng Liu
Student Number: 4798406
Z.Liu-14@student.tudelft.nl

Yi Li
Student Number: 4953819
Y.Li-52@student.tudelft.nl

1 INTRODUCTION

In this project, we applied some computer vision techniques to fulfil taskes according to the manual, where the 3D reconstruction is also focused. In section 2, we used both Harris detector and VLfeat toolbox to obtain correspondences, and then in section 3, the 8-points RANSAC algorithm was applied to find inliers from those correspondences. In section 4, we introduced how we achieved chaining process, creating the point of view matrix. In section 5, we utilised the previous steps to reconstruct the 3D reconstruction results on both "Teddy Bear" and "Castle" dataset. Section 6 is a simple explanation of how to eliminate the ambiguity of affine results. Section 7 is the conclusion about the project and our gains.

2 INTEREST POINT & CORRESPONDENCE

To find the interest points and correspondences over images, we implemented two method in this part. The first one is simply by applying the *match* function in *vlfeat* toolbox, and the second one makes use of the Harris detector. The resulting images are respectively shown below1,2.

From the images, we can see that the Harris Detector provides a better view of interest points and correspondences. The corresponding line neatly connected between two images, rather than crossing each other randomly. The reason might be that Harris Detector is more robust to rotation, since it is able to filter out all edges and only focuses on corners.

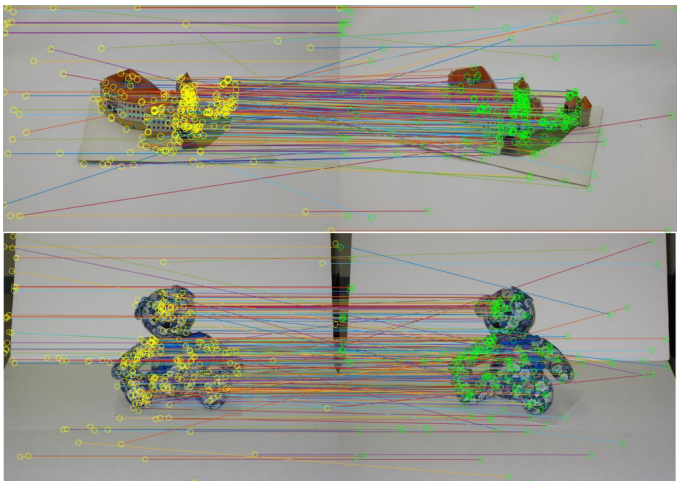

Figure 1: The correspondence detected by applying VLubc-match function.(top: castle model, bottom: teddy bear)

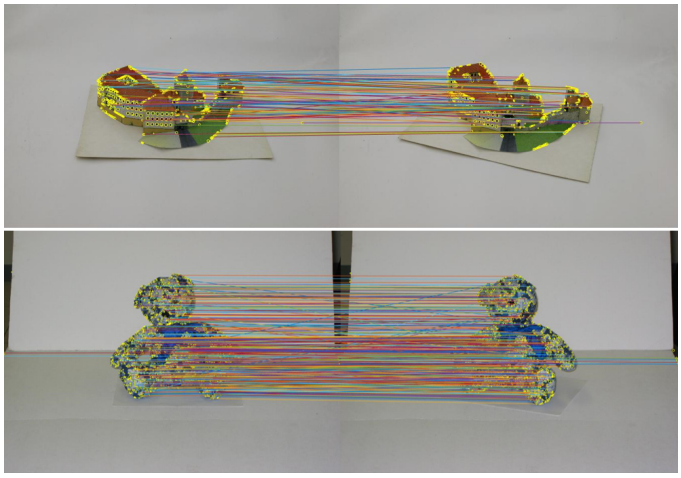

Figure 2: The correspondence detected by applying Harris Detector.(top: castle model, bottom: teddy bear)

3 NORMALISED 8-POINTS RANSAC ALGORITHM

With the interest points and correspondences detected, the 8-point RANSAC algorithm is then applied to find the best matches from them. The resulting matches are show in image3.

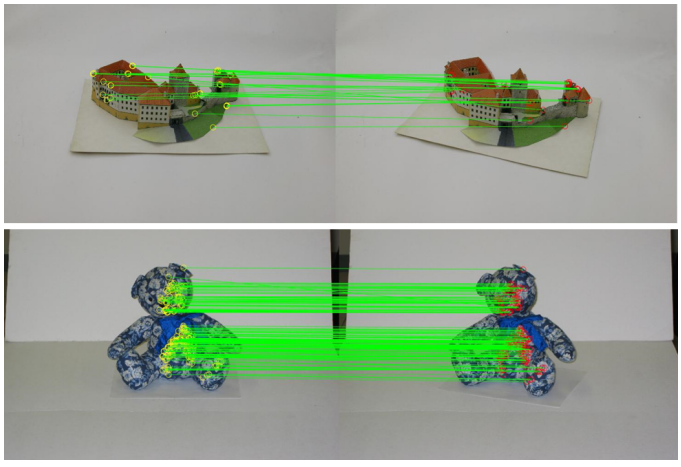

Figure 3: Inliers selected from correspondence by 8 point-RANSAC (top: castle model, bottom: teddy bear)

The algorithm randomly selects 8 interest points to calculate the fundamental matrix, and then use this matrix to calculate the

number of inliers. By repeating this process, the fundamental matrix with most inliers was finally chosen, and used to generate the best matches.

4 CHAINING

Each of our datasets consists of a set of consecutive images, which are also called frames. In this chaining part, we constructed a point-view matrix based on the matches found between each pair of consecutive frames.

The resulted point-view matrices of two datasets are shown in figure 4. The columns and rows represent the tracked points and views/frames respectively. Remarkably, those tracked points are expressed by their index and stored in entries of the matrix(black areas). With the rotation of camera view, some points will disappear in some frames, that is why some columns consist of empty entries(white areas).

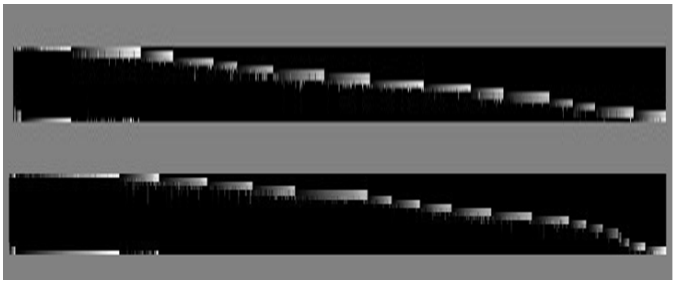

Figure 4: The point-view matrices of teddy-bear(top) and castle(bottom) data-sets

5 STICHING

5.1 Measurements Matrix

We considered to utilise 3 consecutive images to make the blocks and measurement matrix. The basic idea is to apply a loop and the number of iteration is $n-2$, where n is the length of file (number of images) and 2 is 3 ($consecutive\ images$) $-$ 1. Take Teddy bear dataset as example, we choose 3 to be the size of block, such as $1-2-3$, $2-3-4$, etc. There are 16 images thus there should be 14 blocks, the last one is $14-15-16$.

Then we take each block to examine, the points with non-zero values are our interested ones, meaning they exist in all three images we currently considering. If the number of non-zero points in a block is equal or higher than 8, we consider they are significantly enough for the further steps. The measurement matrix is given based on the current block. The size of the matrix is $2m \times n$ (here it is $6 \times n$, n differs from different blocks), where m is the number of frame (consecutive images considered) and n is the number of selected points under the certain block. Later we applied the $sfm()$ function to obtain relevant M and S., we stored them into $Clouds$.

5.2 3D Coordinates

Based on the point-view matrix, we derived a set of three-image blocks. Then we estimated 3D coordinates from each block and visualise them by Tomasi-Kanade factorization. Three visualisations are randomly picked and shown below in figure 5, 6, 7.

As we can see, each block gives us a partial shape of Teddy-bear from a specific view direction, resulting in different coordinates system.

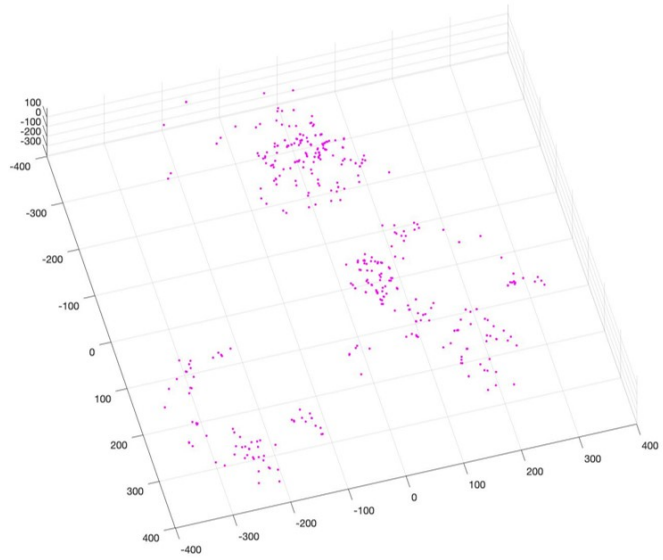

Figure 5: A 3D coordinate of a three-image block from Teddy dataset-No.1

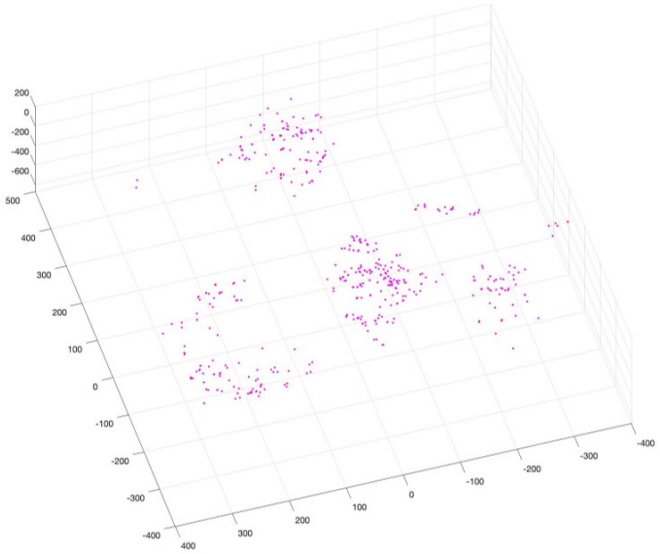

Figure 6: A 3D coordinate of a three-image block from Teddy dataset-No.2

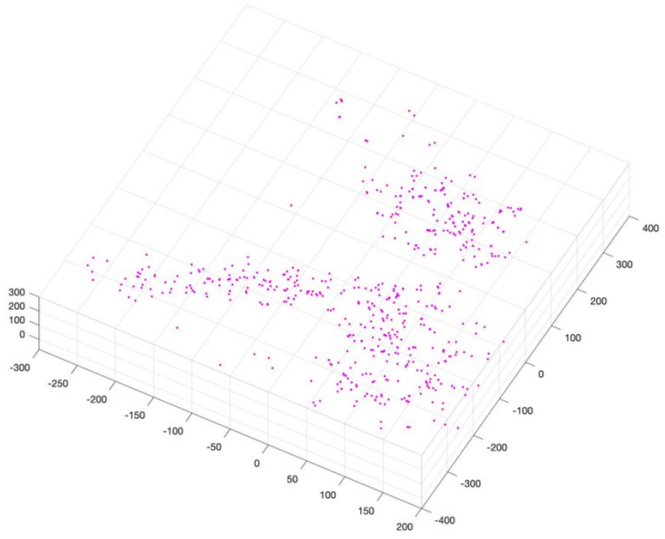

Figure 7: A 3D coordinate of a three-image block from Teddy dataset-No.3

5.3 Optimal Transformation

After obtaining the partial view of the Teddy-bear in different coordinates, it is time to stitch a full view on Teddy-bear by transform all those 3D point into one common coordinate system.
In this process, we made use of the point correspondences derived from the Chaining part, iterative transformed the shared data in the 3D point clouds. The whole Teddy-bear are finally reconstructed in a common 3D coordinate. Figure 8 consists of four different rotation views.

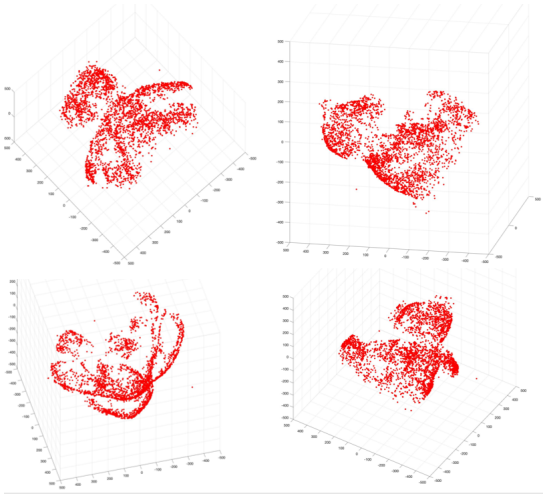

Figure 8: Stitched 3D point set to the main view coordinate of the Teddy dataset(top left: front view, top right: side view, bottom left: top view, bottom right: side view)

Also, we apply the same procedure from point-view matrix to 3D coordinates on the Castle dataset. Two rotated views on 3D reconstruction of castle are put below9, which imply the real shape of the castle model.

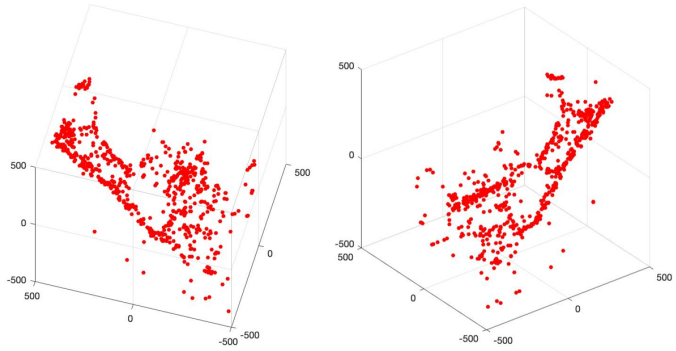

Figure 9: Stitched 3D point set to the main view coordinate of the Castle dataset(left: side view No.1, right: side view No.2)

6 ELIMINATE AFFINE AMBIGUITY

Although we have solved the measurement matrix before, the $D = MS$ does not have unique solution, which is recognised as affine ambiguity. We need to impose image axes ($a1$ and $a2$) are perpendicular and their scale is 1, where $a1$ is column vector and $a2$ is row vector, the projection of x and y respectively. We defined the starting value for L, L_0 as: $A_i L0 A_i^T = Id$. We solved L by iterating through all images and finding L one which minimises $A_i L A_i^T = Id$, for all i, by applying $MATLAB$ function $lsqnonlin()$.

We applied the Choleshy decomposition to cover middle variable C by $L = CC^T$ to update M and S by $M = MC$ and $S = C^{-1}S$. Then, the ambiguity could be eliminated.

7 3D SURFACE RENDERING

In the previous sections, we have already reconstructed the 3D space coordinates of the objects, which are presented as red points in the three-dimensional system of coordinate, although they are able to be recognised by us intuitively clearly, we still expect that we can reconstruct the colour and make a surface to render them more realistic.

In $surfaceRender()$ function, the input are the merged points cloud, matrix M from the measurement matrix $D = MS$, mean value of the first frame and the relevant image, say the reference image, which reflect which side people look at the object (Teddy bear or castle). Similarly as we select the first frame with its non-zero M, we have to make sure that this condition holds on.

We calculated the norm vector, say in a way as the view direction and normalise it. Then we calculated the dot product and remove the negative values, which means these points are behind the surface that we can visualise the object. For a certain view from people, points cannot be observed should be deleted.

For the following steps, we applied the $MATLAB$ functions which offered by the lab manual to create the surface under mesh grid. The surface function is obtained by the $MATLAB$ function and based on which the mesh grid points are interpolated. After transforming

to the main view, we add the mean values back to move points to correct positions.

The last step is to extract colours from the original image and apply them onto the previous surface we achieved. We extracted colours based on the *RGB* channels (if the image is gray-scale, then just utilise the only dimension intensity), selected based on the indexes of points. $surf()$ is used to display coloured surface.

On *TeddyBear* dataset, we could obtain the surface rendering result on all certain single views, meaning that we could render each single view but did not achieve the merged results under all 16 images. The results of two examples (face and back of Teddy respectively) are shown in Figure 10, 11, 12 and 13.

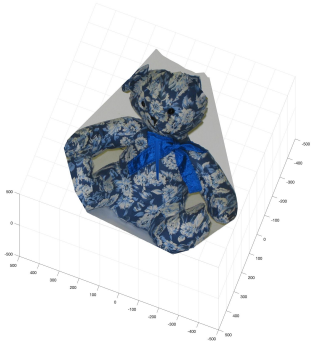

Figure 10: Surface rendering effect of Teddy bear face 1

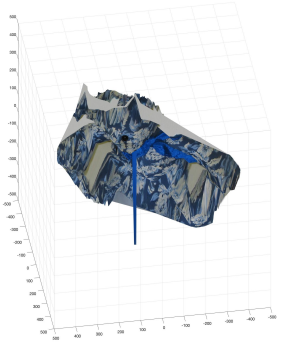

Figure 11: Surface rendering effect of Teddy bear face 2

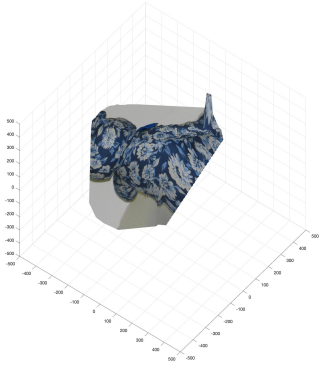

Figure 12: Surface rendering effect of Teddy bear back 1

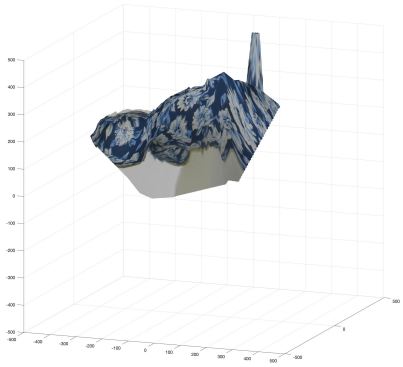

Figure 13: Surface rendering effect of Teddy bear back 2

The results on Castle dataset is not as good as what we expect, shown in Figure 14 and 15.

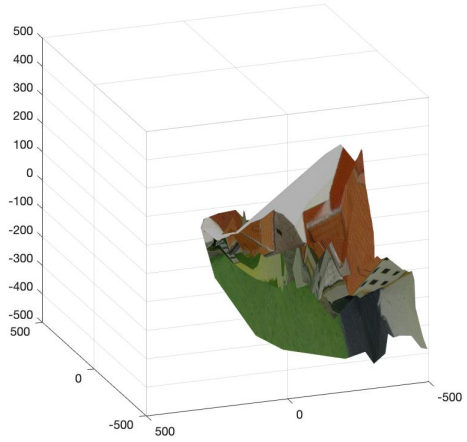

Figure 14: Surface rendering effect of Castle angle 1

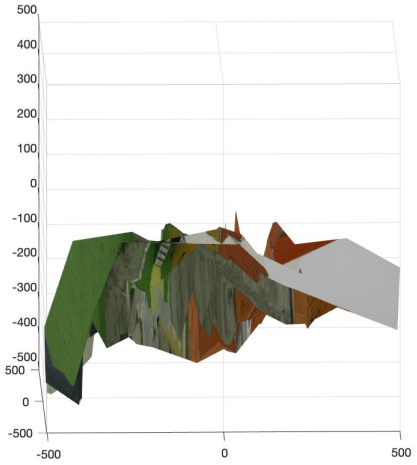

Figure 15: Surface rendering effect of Castle angle 2

8 CONCLUSION

By applying the practical project on both "Teddy Bear" and "Castle" dataset, the results are satisfied and shown by immediate steps in this report. Also, We practised the mathematical and theoretical knowledge of computer vision course especially for the 3D reconstruction part. We obtained a deeper understanding of those techniques and proficient skills to formulate theories into code. We cooperated very well and both of us made great efforts for both encoding and report writing.