Assignment 2: Structure from Motion

Jeroen van Wely, Niek IJzerman & Jochem Soons
11289988, 11318740, 11327030
Computer Vision 2
MSc Artificial Intelligence
University of Amsterdam

May 3, 2021

1 Introduction

Within this report, we will discuss and implement the Structure-from-Motion algorithm. With the Structure-from-Motion algorithm, we are able to recover three-dimensional structures from 2D images.

First, we will implement a function that takes two images as input and computes the fundamental matrix. We will implement this function such that we are able to obtain the fundamental matrix using (1) the eight-point algorithm, (2) the normalized eight-point algorithm, and (3) the normalized eight-point algorithm with RANSAC. For each of these methods, we will draw epipolar lines which enable us to analyze our results. Second, we will construct the point-view matrix which chains multiple consecutive image views using the matches found across these images. We will visualize the point-view matrix as well for analysis. Third, using the previously constructed point-view matrix, we will implement the Structure-from-Motion algorithm. We will visualize 3d point clouds obtained with the Structure-from-Motion algorithm using (1) our own point-view matrix and (2) a provided PointViewMatrix.txt file. Fourth, we will make some proposals that could improve the Structure-from-Motion algorithm performance. Finally, we will discuss and conclude our report.

2 Data

The dataset we will utilize to perform various experiments and analyze results comprises of 49 consecutive house images. All images are of similar size (512 x 480) and grayscale. For visualization, in figure 1 frames 1, 20, and 40 are shown.

(a) frame 1. **(b)** frame 20. **(c)** frame 40.

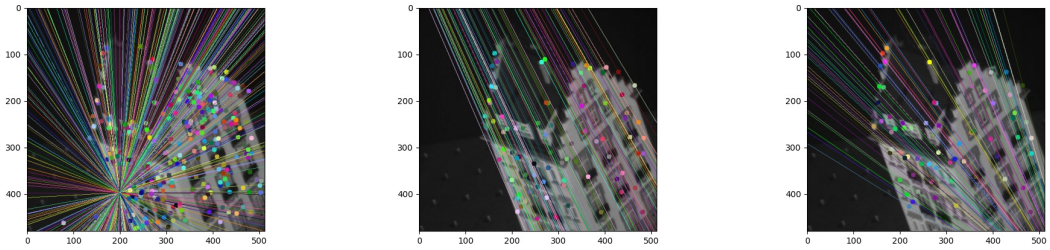
Figure 1: Examples of frames in dataset.

3 Fundamental Matrix & Eight-point algorithm

To apply structure from motion we would have to find the coordinates of a specific point of the scene in both images for each consecutive pair of frames. We could do this exhaustively by checking for a point in the entire second frame. However, this would be computationally very expensive. With the use of the fundamental matrix, we can constraint this search area to a line: the epipolar line. The fundamental matrix is estimated using the eight-point algorithm. In this section, we will analyze the approach for estimating the fundamental matrix using different variants on the eight-point algorithm. These are the vanilla, normalized, and normalized with RANSAC eight-point algorithm. For each figure presented in this section, we have printed an enlargement in appendix C.

3.1 Analysis

In this section, we will analyze the models on their performance and speed. Firstly, we will draw the epipolar lines. These are presented in figure 2. Here each variant is used with its optimal parameter settings (See table 1).



(a) Unnormalized without RANSAC. **(b)** Normalized without RANSAC. **(c)** Normalized with RANSAC.

Figure 2: Epipolar lines plotted with the fundamental matrix computed with different variants of the eight-point algorithm and their optimal parameter settings.

When looking closely we can see that for the unnormalized without RANSAC method the lines do not always go straight through the corresponding point. For the normalized without and with RANSAC methods, we see that this is always the case. Therefore, based on these plots it seems that the eight-point algorithm with and without RANSAC clearly outperforms the unnormalized variant.

Table 1: Optimal parameter-settings for eight-point algorithm variants. Where num_matches is the number of matches used to estimate the fundamental matrix; sample_size is the sample size that is used to sample within the RANSAC algorithm; max_distance is used as a threshold to find inliers within the RANSAC algorithm; and num_iter is the number of iterations used within the RANSAC algorithm. The optimal value for num_matches in the normalized with RANSAC eight-point algorithm tend to jump around, therefore it does not have to be 80 to get the best results. We write 80 because that was the value used in the production of the images in figure 2.

Eight-point algorithm variant	num_matches	sample_size	max_distance	num_iter
Unnormalized without RANSAC	338	_	_	_
Normalized without RANSAC	97	_	_	_
Normalized with RANSAC	80	50	8	$1e^{-8}$

To better understand the performance of the different variants we have implemented a loss function based on the following fundamental matrix constraint:

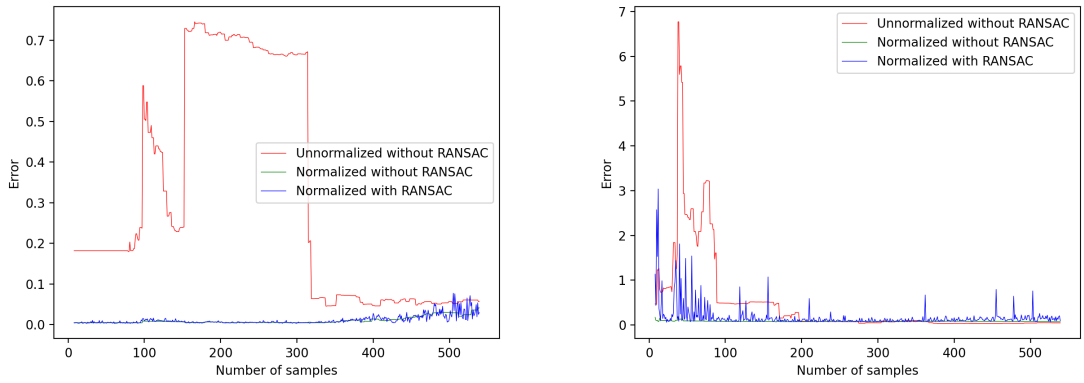
$$p_i'^T F p_i = 0$$

Where F is the fundamental matrix and p and p' are sets of points where for each i p_i is a corresponding point to p'_i. For each model, we compute this error for the points corresponding with the best 100 matches obtained from the brute force matcher from CV2. Note that these are not necessarily the points that have been used to compute the fundamental matrix F. We then take the mean over all 100 error values and use this as a measure of performance. In figure 3a we present a plot of this measure plotted against the number of point-correspondences used to estimate F for all three variants of the eight-point algorithm.

Here it becomes very clear that our previous conclusion based on the plots in figure 2 was correct. The normalized variants do much better relative to the vanilla eight-point algorithm. The normalized with RANSAC model has very similar performance compared to the normalized without RANSAC model. However, if we compare the minimum error over all number_of_samples we find that the normalized with RANSAC model always has a slightly lower minimum relative to the normalized without RANSAC model. This was tested empirically over multiple runs. In the case of 3a the minimum error for the normalized with RANSAC model was 0.00291 and for the normalized without RANSAC model, this was 0.00447. These were both much higher compared to the vanilla model which was 0.0456.

We can also see that the stability of the normalized with RANSAC model starts decreasing after number_of_samples becomes higher than 400. This is most likely because from that point on we start taking bad matches into account, as the matching function returns matches ordered from best to worse. The bad matches are probably throwing off the model, which would explain the instability.

In figure 3b we have plotted the same type of graph only here we haven't used two consecutive frames but instead we skipped 9 frames. Here we can see that even the minimum errors are much higher than the errors in 3a. However, note that as the number of samples increases the vanilla eight-point algorithm eventually starts to outperform the normalized models. This could be something to consider when frames are not so similar to each other.



(a) Frames 1 and 2.

(b) Frames 1 and 10.

Figure 3: Plots of the error against the number of matches used to estimate the fundamental matrix F.

In figure 4 we plotted the time to run the different models against the number of matches used to estimate F. Here it becomes clear that the normalized with RANSAC model is much slower relative to the models without RANSAC. This seems logical as the RANSAC model has to compute multiple matrix decompositions for each iteration it has been set to. The peak at the end of the graph for the normalized with RANSAC model can most likely again be explained by bad matches that are taken into account. The match-keypoints function gradually increases its match threshold (Lowe's ratio test) to ensure the desired amount of matches can be returned. The distance between these last, and thus worse, matches are much bigger compared to the previous matches and thus it takes longer for the

threshold within the match-keypoints has reached that distance.

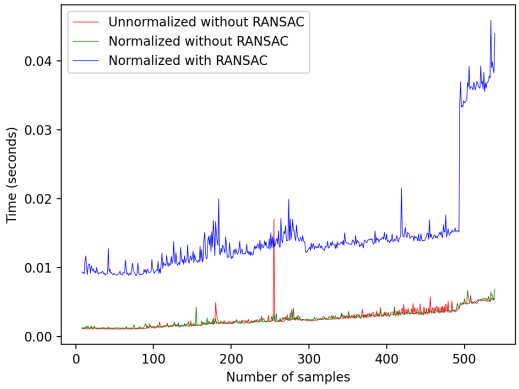


Figure 4: Number of seconds plotted against the number of matches used to estimate the fundamental matrix for all three eight-point algorithm variations.

4 Chaining

The usual matching process is performed across pairs of views. It is however also possible to represent all matches across all consecutive pairs of views in a single matrix. This matrix is referred to as the point-view matrix. Within this matrix, columns represent surface points and rows represent the images they appear in. In this section, we will discuss several techniques we deployed to construct a point-view matrix based on our dataset.

4.1 Point-view matrix

We have implemented different variants of constructing a point-view matrix. We can summarize our construction scheme as follows:

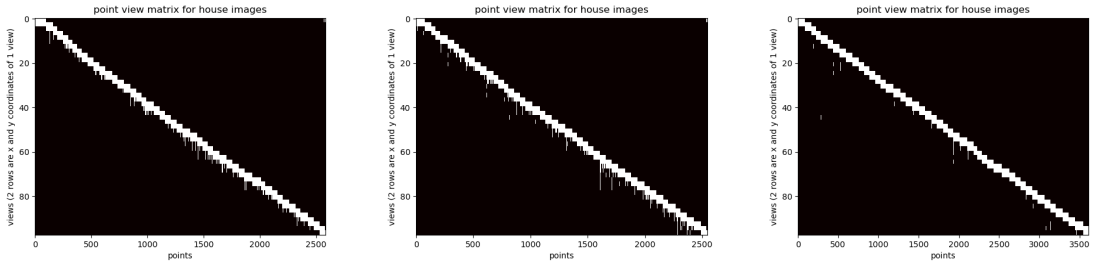
1. Detect interest points with SIFT for two consecutive images and match these points using a Brute Force matcher. Optionally we can filter out good matches using Lowe's ratio test.
2. Apply eight-point RANSAC to the point pairs to detect and select inlier points.
3. (a) **Option 1:** Match and add points to the point-view matrix based on their coordinates. If a matching coordinate is not found, a new column is added to the matrix dedicated to the new point.

   (b) **Option 2:** Match and add points to the point-view matrix based on their coordinates. If a matching coordinate is not found, check if the descriptor of a new point matches the descriptor of the last point in any existing column based on an L1 distance threshold. If yes, add the point to the dedicated column. If not, a new column is added to the matrix dedicated to that point.

   (c) **Option 3:** Match and add points based on their descriptors. If the descriptor matches the descriptor of the last point in any existing column based on an L1 distance threshold, add the new point to that column. If a matching descriptor is not found, a new column is added to the matrix dedicated to the new point.

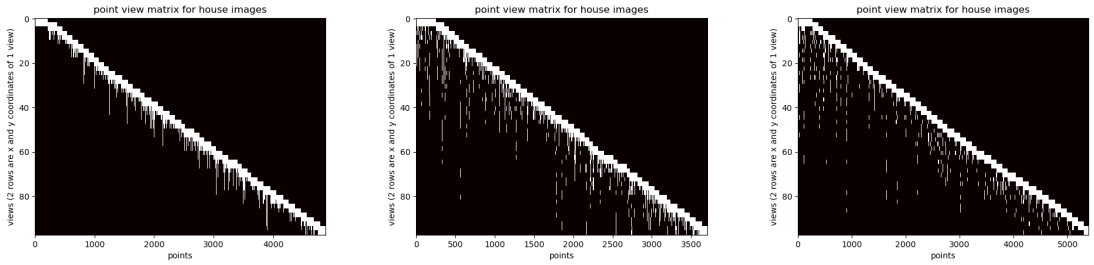
4.2 Analysis

In figure 5 and 6 we show constructed point-view matrices for each option with different RANSAC and L1 threshold values.



(a) Matching with coordinates only (option 1). **(b)** Matching with coordinates and descriptors (option 2). **(c)** Matching with descriptors only (option 3).

Figure 5: Point view matrices for all 3 proposed methods. # RANSAC iterations: 100, RANSAC inlier distance threshold: 0.001, L1 distance threshold: 150.



(a) Matching with coordinates only (option 1). **(b)** Matching with coordinates and descriptors (option 2). **(c)** Matching with descriptors only (option 3).

Figure 6: Point view matrices for all 3 proposed methods. # RANSAC iterations: 100, RANSAC inlier distance threshold: 0.01, L1 distance threshold: 300.

As visible in both figures 5 and 6, all point-view matrices are very sparse. Overall, the point-view matrices in figure 6 are less sparse than in figure 5. This is due to the fact that in figure 6 we use looser threshold values for the maximum distance for points to be considered an inlier (0.001 in figure 5, 0.01 in figure 6) and for the L1 distance threshold to match descriptors of new points with descriptors already in the point-view matrix (150 in figure 5, 300 in figure 6). This results in more matches across frames being included and more descriptors of new and already introduced points being matched. The fact that the point-view matrices in figure 6 are a bit denser than in 5 due to looser thresholds does, however, not necessarily correlate with a better quality of the point-view matrix as more wrong matches across frames are included and descriptors are matched wrong more easily.

A possibility to improve the point-view matrix is by not only considering consecutive image pairs to determine point matches to include in the point-view matrix but also look at non-consecutive pairs (e.g. frame pairs 1-3, 2-4, etc). Following this procedure, overlooked similar points between consecutive images might be included as well. This could result in a denser point-view matrix of higher quality. Furthermore, it is possible to interpolate missing values in the point-view matrix. Although the interpolated values might not exactly correspond to the real values, we do create a denser point-view matrix that could improve results.

5 Structure from Motion

With the constructed point view-matrix that represents point correspondences for different camera views, as explained in section 4, we can perform the final structure from motion (SFM) algorithm. SFM is a technique that serves to recover 3D structure from a sequence of 2D images of a scene [4]. SFM can be applied in various settings, and because of the low costs in collecting data (we just need regular images), it is especially useful when the collection of e.g. terrestrial laser scanning data or GPS data is infeasible or too costly. Examples of domains where SFM is applied regularly are geoscience [5] and documentation and conservation of cultural heritage [1].

5.1 Factorize and Stitch

Given a measurement matrix of point correspondences for different frames or views, SFM essentially works by factorizing this matrix in two parts using singular value decomposition: a motion matrix M and a structure matrix S. When all points would occur in all different views, i.e. when the point view matrix we have is fully dense, we could simply perform this factorization process on the whole matrix. However, because the point-view matrices we construct are relatively sparse, we need to select dense sub-blocks from the matrix to factorize and construct motion and structure. This way, we can iteratively estimate 3D structure point clouds and stitch them together. The general scheme we implemented is as follows:

1. Obtain a point-view matrix using the method we described in section 4.
2. Iterate over a fixed m number of views (which corresponds to 2m rows in the PVM).
3. Select a dense sub block from these views. Our method of selecting this dense block was to drop all columns that contained at least one zero value.
4. Normalize the dense sub-block by subtracting the mean of the points in each view.
5. Apply SVD to the dense block and derive the motion and structure matrices (we will not explain this process in detail, as it is explained already in the lectures)
6. When we are in iteration 2 and higher, we have a 3D point cloud of the previous iteration (S_prev) and a point cloud of the current iteration (S_curr). Because both blocks were normalized differently, we perform Procrustes analysis on the shared points between views to find the transformation matrix R and scale value s that aligns both point clouds.
7. We then apply this transformation matrix and scaling to newly seen points in S_curr and append these transformed points to our main view.

Besides the stitching scheme as described above, we also created 3D point clouds using just one dense sub-block and by using the provided dense point-view matrix. For our implementation of using just one dense block, we iterated over each selection of 2*m rows just as we do in the stitching process, and we kept track of the largest dense sub-block we could find. This largest sub-block was subsequently used for factorization and extracting structure from motion. Moreover, before obtaining results, we varied the hyperparameters used for constructing the point-view matrix (to empirically optimize our results) and we also implemented and compared results of the SFM algorithm when taking 3 or 4 consecutive views into account.

5.2 Analysis

The hyperparameters we used in obtaining our results are displayed in table 2 - so we constructed our point-view matrix based on both coordinates and descriptors. A visualization of the point-view

matrices used for obtaining results can be found in appendix D, figure 17. We have plotted results obtained using the three different methods in figures 7 - 9. All plots have three views of the point cloud: a bottom view (looking from 'within the house'), a side view (looking from 'beneath the house') and a top view (looking from 'before the house'). We will discuss the results of the different methods one by one.

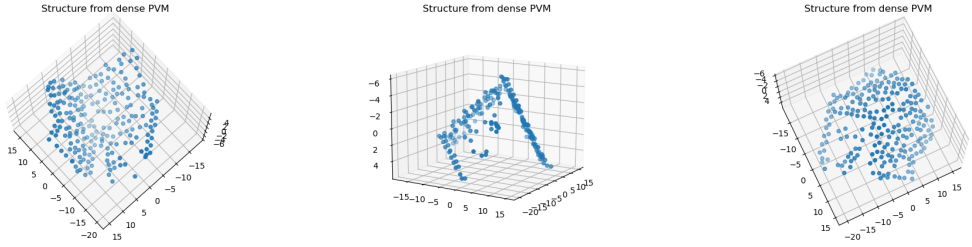
First of all, we can see in figure 7 that, when using the provided dense PVM (the 'super-dataset'), we obtain a 3D point cloud that closely resembles the house of our dataset. This indicates that the SFM algorithm is able to successfully recover structure from a series of 2D images. Especially in figures 7b and 7c, we can clearly see the three main walls or planes that make up the house.

Secondly, from figure 8 we notice that using only one dense sub-block gives surprisingly good results in recovering the house structure. Although it is not that easy to display in 2D plots, as we have a lot fewer points when compared to using the whole point view-matrix, we can see especially in the side view plots (b and e) a shape that looks very similar to the side view plot of the point cloud constructed using the provided PVM. The results show us three things: 1) our own implementation of constructing a PVM seems to be correct, as results are similar to that obtained by using the provided PVM 2) even by just considering the single densest block the SFM algorithm can still recover a house-like structure, and 3) we also notice a small difference between setting m=3 or m=4. The differences are that when m is smaller, we have more points in our point cloud and the plot is also a little bit noisier.

Thirdly, when looking at figure 9, we can see that results obtained through stitching multiple blocks are more disappointing than our previous two results. Both for m=3 and m=4 all plots are very noisy, although for m=4 we can see (again, especially in the side view) a shape that somewhat resembles the wall-like lines we see in the side views of figures 7 and 8. Again, m=3 results in more points in the structure but also more noisy points. Overall though, all plots are quite noisy and in terms of quality, they come nowhere near the results obtained by using just one dense block or the provided dense PVM. Because plots returned using just one dense block look correct but stitched plots look incorrect, we are doubting the correctness of our iterative stitching algorithm (especially the alignment of point clouds). We tried several built-in Procrustes functions and eventually wrote our own adaptation of the SciPy Procrustes function, and we also tried to implement the ICP algorithm for alignment that we wrote in assignment 1. But unfortunately, the results remained unconvincing and disappointing.

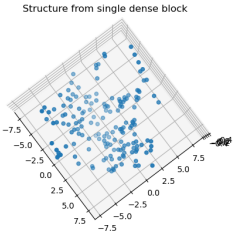
Table 2: Hyperparameters used for constructing the point view matrices for SFM.

Method	Option	num_matches	inlier distance	L1 distance	num_iter
Single dense block	3	300	0.1	250	100
Multiple block stitching	3	200	0.1	250	100

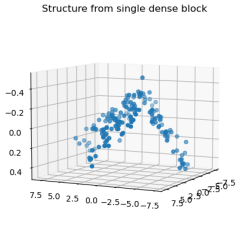


(a) Bottom view **(b)** Side view **(c)** Top view

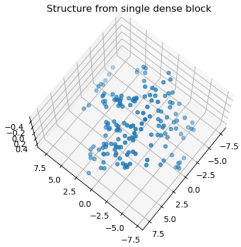
Figure 7: Obtained structure from provided dense point view matrix in (PointViewMatrix.txt). No elimination of affine ambiguity has been performed.



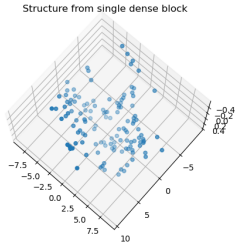
(a) m=3, bottom view



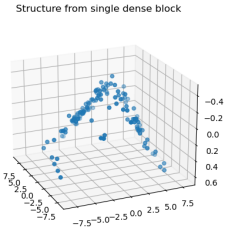
(b) m=3, side view



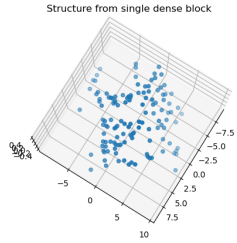
(c) m=3, top view



(d) m=4, bottom view

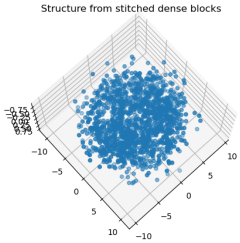


(e) m=4, side view

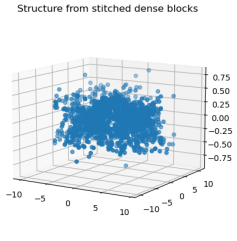


(f) m=4, top view

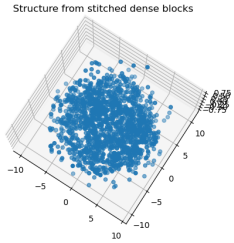
Figure 8: Obtained structure from single dense block. For the top row, the number of subsequent frames (m) used to perform factorization is 3, for the bottom row, m = 4. No elimination of affine ambiguity has been performed.



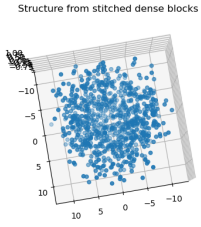
(a) m=3, bottom view



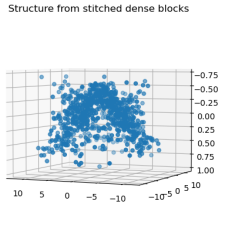
(b) m=3, side view



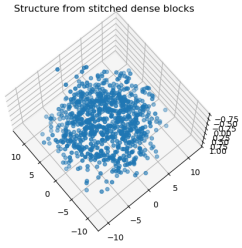
(c) m=3, top view



(d) m=4, bottom view



(e) m=4, side view



(f) m=4, top view

Figure 9: Obtained structure from stitched dense blocks. For the top row, the number of subsequent frames (m) used to perform factorization per iteration is 3, for the bottom row, m = 4. No elimination of affine ambiguity has been performed.

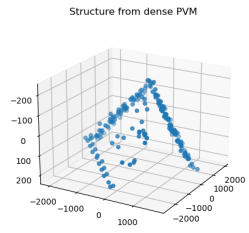
6 Additional Improvements

There are several ideas we implemented to improve our results. We discussed the first improvement already in section 4: trying various criteria for matching and adding new points to our point view matrix (based on coordinates, SIFT descriptors, or a combination of both). When performing the experiments in section 5, we noticed that by using a combination of both criteria we achieved better structure plots than by using just descriptors or just coordinates.

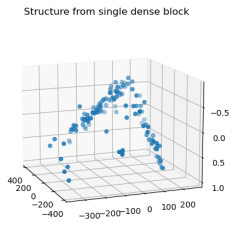
Secondly, because our results of the stitched blocks were not satisfactory, and we suspected that this might come from an error in our alignment method, we tried to implement various Procrustes variants (orthogonal, other modules, etc), and we also implemented our ICP algorithm from assignment 1 to rotate and translate two corresponding structure matrices. However, this did not contribute to improving our plots, and therefore we decided to not include the plots created with the ICP alignment method (the method can still be called upon in our code, though).

Another idea for improving the framework that we implemented is the removal of affine ambiguity of the SFM algorithm. The problem with the basic (affine) SFM algorithm is that the decomposition of the measurement matrix into a motion and structure matrix is not unique: results are subject to affine transformations that can produce and suboptimal plots. To remove this affine ambiguity, we set orthographic constraints to our motion (camera) matrix and update our motion and structure matrices accordingly (the intermediate steps, as discussed in the lecture, can be found in our code). In figure 10, we have plotted the obtained point clouds for the three different methods as described in section 5 (where we used m=4). For simplicity in comparison, we only plotted the side and top views. At first sight, there do not seem to be any differences from our earlier plots in section 5: all figures look more or less equal to their respective counterpart. When we look at the values on the axes, however, we see that the plots are actually very different from those in section 5, their likeliness is simply due to the fact that matplotlib scales axes automatically. We see that for the dense PVM, all axes have increased in range, and the z-axis has increased relatively more, so the house has become less 'flat' (more depth) whereas with the single block and stitched blocks methods, the range of the x and y axes have increased much more when compared to the z-axis, so here the house has become very 'flat' (less depth). It is hard to pinpoint what causes these different changes in scaling of the axes exactly, and therefore it is also difficult to judge if the affine ambiguity removal improved results when compared to the baseline. Overall, apart from the changed axes, no other significant changes or improvements can be noticed from the addition to our framework of ambiguity removal.

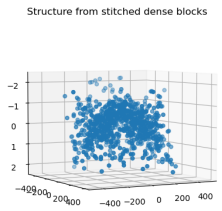
The final idea that we thought of to improve our framework was to increase the density of our point view matrix further through interpolation of missing values, which produces larger dense blocks that we can factorize during SFM. Unfortunately, because of time constraints and workload from other courses, we did not manage to successfully implement a good interpolation pipeline to improve results. We strongly believe, however, that such a technique could contribute to retrieving more accurate structural matrices.



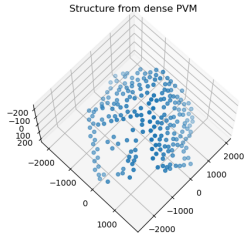
(a) Dense PVM, side view



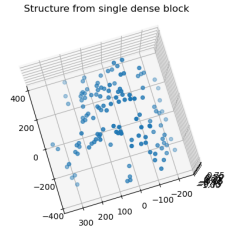
(b) Single block, side view



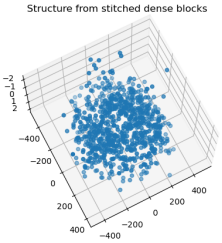
(c) Stitched blocks, side view



(d) Dense PVM, top view



(e) Single block, top view



(f) Stitched blocks, top view

Figure 10: Obtained structure point clouds (side view only) for the three different methods where elimination of affine ambiguity has been performed.

7 Discussion and Conclusion

When working on the assignment and this report, we learned a lot about a variety of interesting computer vision subjects, related to the estimation of a fundamental matrix using variants of the eight-point algorithm, epipolar geometry, construction of point-view matrices, and recovering structure from a series of images through 3D affine SFM. From this report, we can conclude several things. Firstly, we saw that we can successfully estimate fundamental matrices using the eight-point algorithm and that using RANSAC and normalization have many benefits over their vanilla counterpart. Secondly, we implemented and discussed different techniques to construct a point view matrix, which can introduce more density but also more noise. Lastly, we discussed the SFM algorithm and our results on the house images dataset. In the end, not all results were equally convincing, but we still managed to successfully recover 3D structural information from a series of 2D images, both with a provided matrix but also through our own framework. Overall, we enjoyed working on the assignment, because it encouraged us to find solutions or improvements that were not always provided in the assignment or the lectures. A particular example of such an original idea is our deployment of the ICP algorithm for aligning the structural matrices, which we had to write for the previous assignment. This solution-oriented thinking using the knowledge you already have has certainly made us more enthusiastic about the field of computer vision.

A Self Evaluation

All members contributed equally to the assignment. Each group member was tasked with creating the code for one of the three sections that required coding, but we also read and checked each others implementations to ensure correctness. Other matters like the analysis of the results and writing the report were done jointly by discussions over Zoom.

B Fundamental Matrix

In this section, we will discuss the approach for estimating the fundamental matrix using different variants on the Eight-point algorithm. These are the vanilla, normalized, and normalized with RANSAC Eight-point algorithm. Firstly, we will discuss why we use the fundamental matrix. Then in sections 3.1, 3.2, and 3.3, we will briefly discuss the theory behind the three Eight-point algorithm variants. Finally, in section 3.4 we will present and discuss the results obtained from using the three algorithms.

To apply structure from motion we would have to find the coordinates of a specific point in the scene in both images for each consecutive pair of frames. Take the scenario where we have selected a point in frame 1 which we want to find in frame 2. We could then check every single point in frame 2 and this would give us the correct result. However, this would be computationally expensive. Using the fundamental matrix would be a better alternative. The fundamental matrix sets an epipolar constraint on where a point in one frame of a scene can occur in another frame of that same scene. Now we will only have to search within this constraint, which is a line. This is computationally much less expensive than checking each point. The pseudocode of the different strategies for estimating the fundamental matrix is given in the next three sections.

B.1 Eight-point algorithm

To find the 3x3 fundamental matrix we will use the eight-point algorithm. The eight-point algorithm was first proposed in 1981 [3]. It was originally used to estimate the essential matrix, however, it is easily extended to the estimation of the fundamental matrix. Below we present the pseudo-code of our implementation of the vanilla eight-point algorithm.

```
1: Collect at least 8 point-correspondences using SIFT by:
   1.1: Find matches using SIFT and OpenCV library
   1.2: Remove points in the background using Lowes ratio test
   1.3: Sort by distance corresponding with quality of the match
   1.4: Return only the best matches/point-correspondences
2: Place the point-correspondences coordinates in matrix A (see below for more information
    on matrix A)
3: Compute the SVD of matrix A resulting in: A = U D Vt
4: Compute the fundamental matrix F by reshaping the last row of Vt into shape 3X3
5: Set rank of F to 2 by the following steps:
   5.1: Compute the SVD of fundamental matrix F resulting in: F = FU FD FVt
   5.2: Correct the smallest singular value of FD to be 0. This gives FD'
   5.3: Recompute fundamental matrix F resulting in: F = FU FD' FVt
```

Where Matrix A:

$$\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_nx_n' & x_ny_n' & x_n & y_nx_n' & y_ny_n' & y_n & x_n' & y_n' & 1 \end{bmatrix}$$

B.2 Normalized eight-point algorithm

The problem with the original eight-point algorithm [3] is that the algorithm is very sensitive to noise, which made its use limited. In 1997 R.I. Harley proposed a variation that significantly increased the accuracy. This was the normalized eight-point algorithm and in this section, we will outline the implementation used in this study [2].

The normalized eight-point algorithm is defined by the same steps presented in the pseudocode in the previous section with adding two preceding steps and one final step. **Step 1** is computing matrix T for both sets of points p and p'. We do this by first computing the mean over both coordinates x and y in the corresponding set of points:

$$m_x = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \& \qquad m_y = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Use the means to compute d:

$$d = \frac{1}{n}\sum_{i=1}^{n} \sqrt{(x_i - m_x)^2 + (y_i - m_y)^2}$$

Finally, compute matrix T:

$$T = \begin{bmatrix} \sqrt{2}/d & 0 & -m_x\sqrt{2}/d \\ 0 & \sqrt{2}/d & -m_y\sqrt{2}/d \\ 0 & 0 & 1 \end{bmatrix}$$

Step 2 Now that T and T' have been computed for both sets of points p and p' we can use these to normalize the points:

$$\hat{p} = Tp \qquad \& \qquad \hat{p'} = T'p'$$

Now we will execute steps 1 to 5 from the pseudocode in the previous section and finalize with the final step which is using T and T' to compute the final denormalized fundamental matrix F to return:

$$F = T'^{T} F T$$

The complete pseudocode for the normalized eight-point algorithm:

```
1: Compute matrix T and T' for both sets of points p and p':
    1.1: Compute means of x, y, x' and y' coordinates
    1.2: compute d and d'
    1.3: Compute T and T'
2: Normalize both sets of points p and p' resulting in: p = Tp, p' = T'p'
3: Collect at least 8 point-correspondences using SIFT by:
    3.1: Find matches using SIFT and OpenCV library
    3.2: Remove points in the background using Lowes ratio test
    3.3: Sort by distance corresponding with quality of the match
    3.4: Return only the best matches/point-correspondences
4: Place the point-correspondences coordinates in matrix A (see below for more information
     on matrix A)
5: Compute the SVD of matrix A resulting in: A = U D Vt
6: Compute the fundamental matrix F by reshaping the last row of Vt into shape 3X3
7: Set rank of F to 2 by the following steps:
    7.1: Compute the SVD of fundamental matrix F resulting in: F = FU FD FVt
    7.2: Correct the smallest singular value of FD to be 0. This gives FD'
    7.3: Recompute fundamental matrix F resulting in: F = FU FD' FVt
8: Denormalization off f resulting in: F = T'^T F T
```

B.3 Normalized Eight-point Algorithm with RANSAC

We have also implemented the normalized eight-point algorithm with RANSAC. This requires one more additional step to the steps used to perform the normalized eight-point algorithm. After normalizing the points in steps 1 and 2 we call upon our RANSAC function where we select eight random point-correspondences from the entire set of point-correspondences. With these points, we compute the fundamental matrix. Then using a given threshold and a distance measure we compute the number of inliers. The distance measure to compute the distance between two points p and p' is the Sampson distance defined by the following equation:

$$d_i = \frac{\left(p_i'^T F p_i\right)^2}{(Fp_i)_1^2 + (Fp_i)_2^2 + \left(F^T p_i'\right)_1^2 + \left(F^T p_i'\right)_2^2}$$

Where F is the fundamental matrix and $(Fp)_j^2$ is the square of the j^{th} entry of the vector Fp. If d_i is smaller than some threshold, the match is said to be an inlier. When we find a fundamental matrix F with a new maximum of inliers we save this matrix F and the inliers. We repeat these steps for a given number of iterations and then return the fundamental matrix F that had the most inliers together with those inliers. We use those inliers as the point-correspondences to compute the final fundamental matrix F using the remaining steps discussed in the previous section. The complete pseudocode for the normalized eight-point algorithm with RANSAC:

The complete pseudocode for the normalized eight-point algorithm with RANSAC:

```
1: Compute matrix T and T' for both sets of points p and p':
    1.1: Compute means of x, y, x' and y' coordinates
    1.2: compute d and d'
    1.3: Compute T and T'
2: Normalize both sets of points p and p' resulting in: p = Tp, p' = T'p'
3: Use RANSAC to find the best sample of point-correspondences/matches:
    3.1: For given number of iterations:
        3.2: Randomly choose 8 random point-correspondences
        3.3: Compute fundamental matrix F based on the sample
        3.4: Compute the number of inliers using Sampson distance and given threshold
        3.5: if the number of inliers is more than the current max:
            3.6: Set the current number of inliers as new max
            3.7: Save inliers and corresponding F
    3.8: return inliers en fundamental matrix F corresponding to the max number of inliers
4: Collect at least 8 point-correspondences using SIFT by:
    4.1: Find matches using SIFT and OpenCV library
    4.2: Remove points in the background using Lowes ratio test
    4.3: Sort by distance corresponding with quality of the match
    4.4: Return only the best matches/point-correspondences
5: Place the point-correspondences coordinates in matrix A (see below for more information
     on matrix A)
6: Compute the SVD of matrix A resulting in: A = U D Vt
7: Compute the fundamental matrix F by reshaping the last row of Vt into shape 3X3
8: Set rank of F to 2 by the following steps:
    8.1: Compute the SVD of fundamental matrix F resulting in: F = FU FD FVt
    8.2: Correct the smallest singular value of FD to be 0. This gives FD'
    8.3: Recompute fundamental matrix F resulting in: F = FU FD' FVt
9: Denormalization off resulting in: F = T'^T F T
```

C Enlarged figures section 3

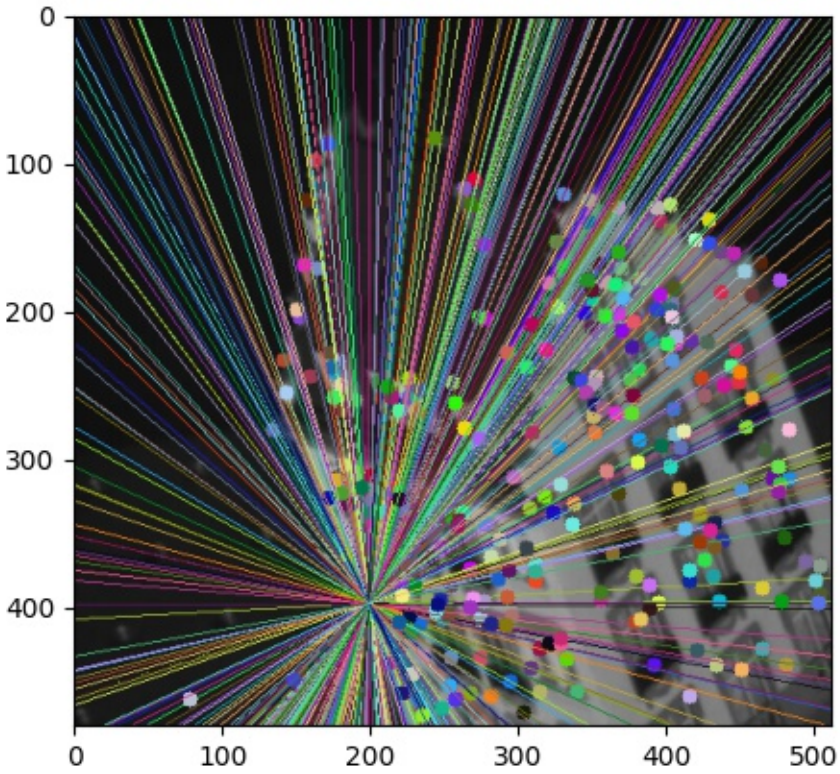


Figure 11: Epipolar lines plotted with the fundamental matrix computed with the unnormalized without RANSAC eight-point algorithm and its optimal parameter settings.

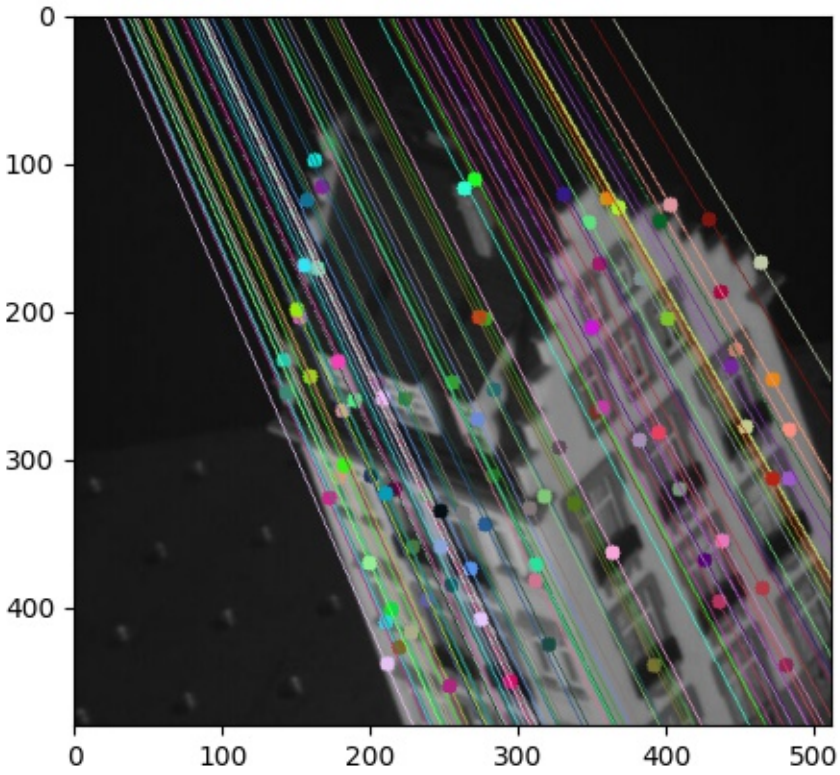


Figure 12: Epipolar lines plotted with the fundamental matrix computed with normalized without RANSAC eight-point algorithm and its optimal parameter settings.

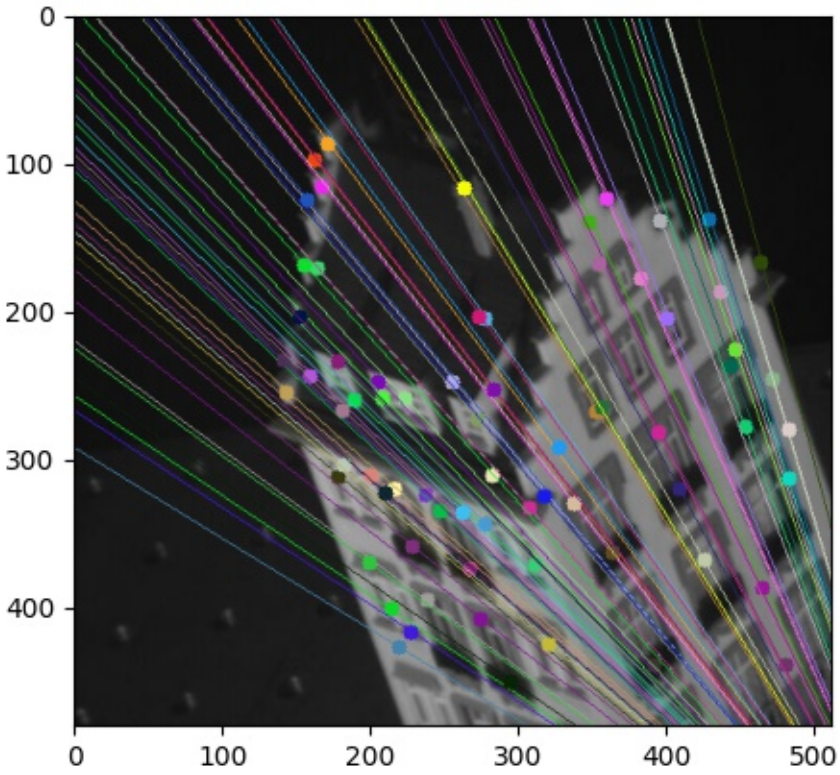


Figure 13: Epipolar lines plotted with the fundamental matrix computed with normalized with RANSAC eight-point algorithm and its optimal parameter settings.

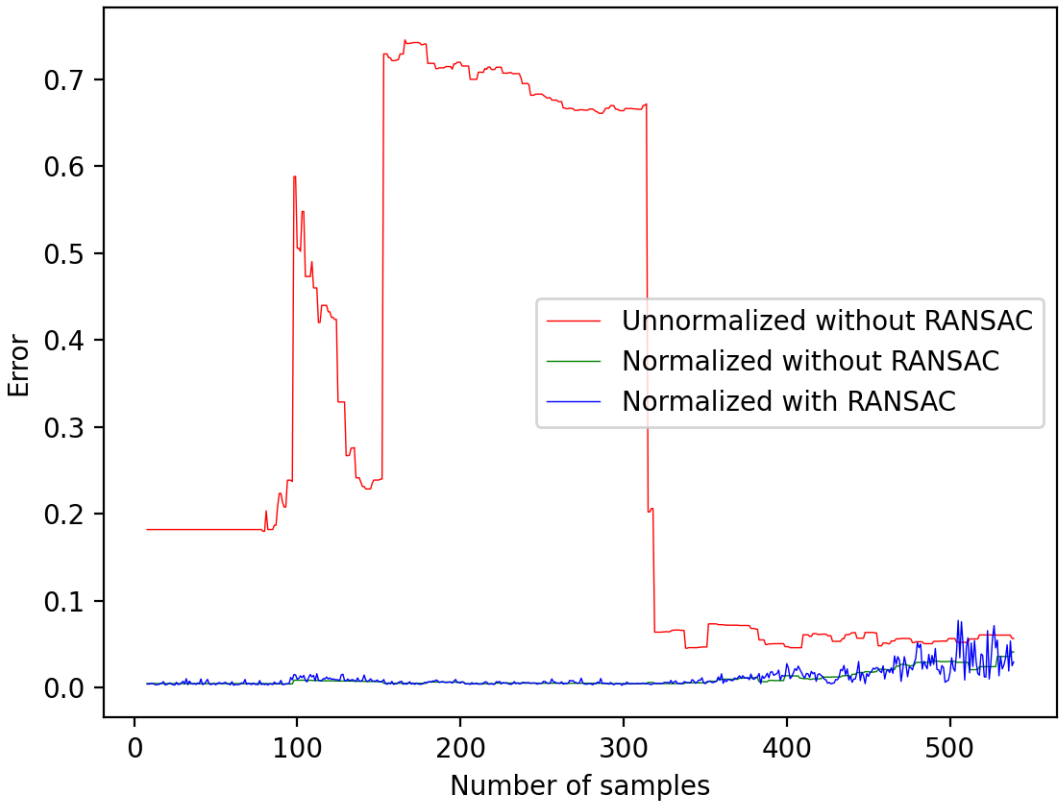


Figure 14: Plots of the error against the number of matches used to estimate the fundamental matrix F for frames 1 and 2.

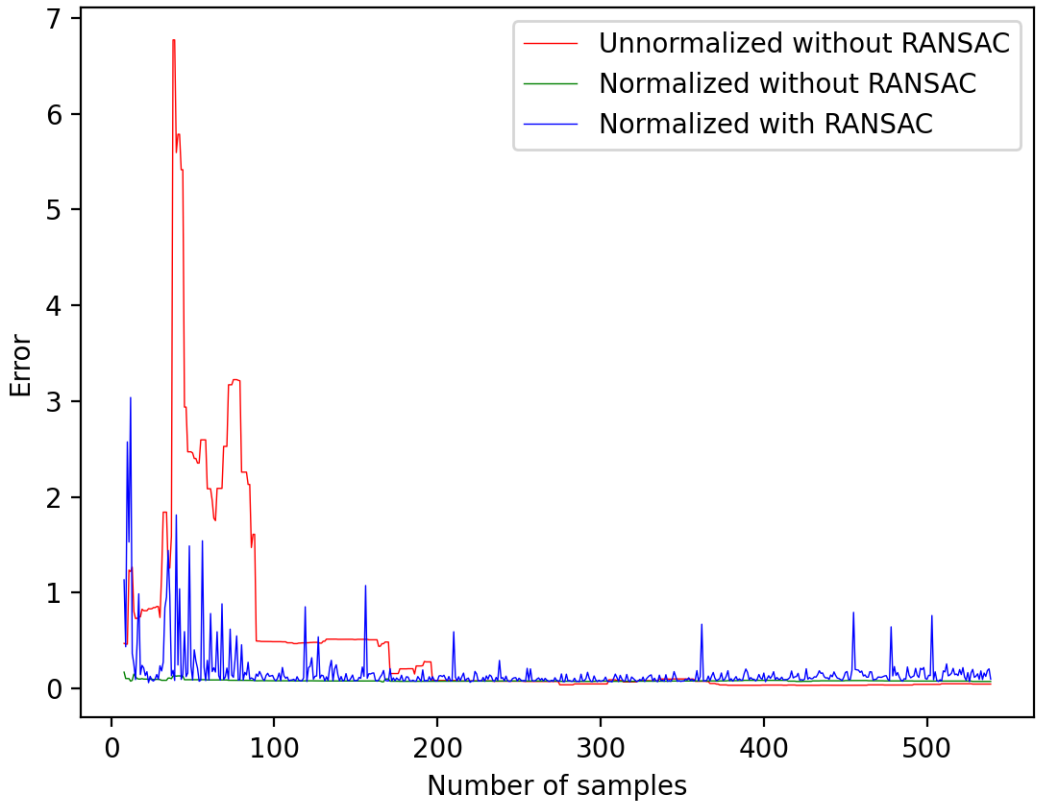


Figure 15: Plots of the error against the number of matches used to estimate the fundamental matrix F for frames 1 and 10.

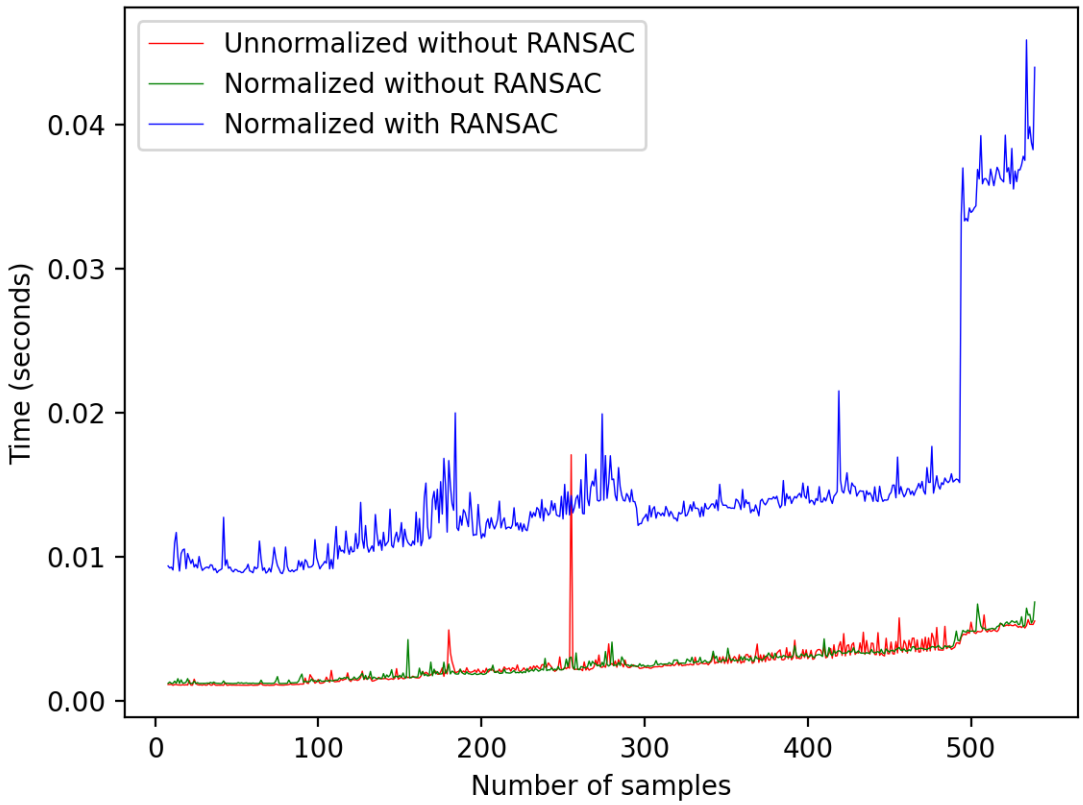
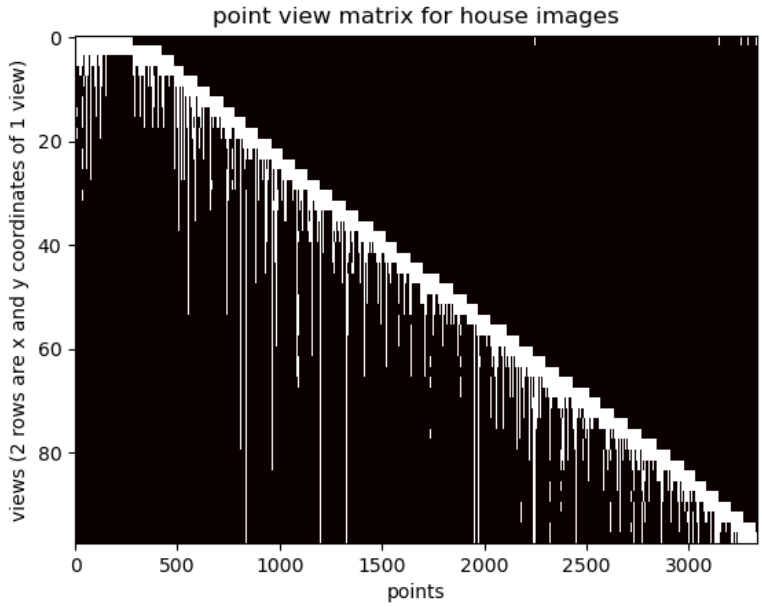
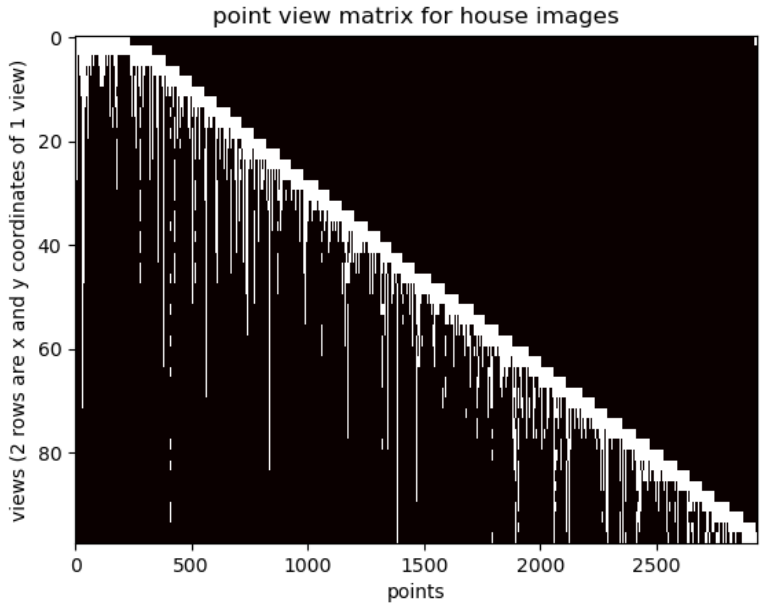


Figure 16: Number of seconds plotted against the number of matches used to estimate the fundamental matrix for all three eight-point algorithm variations.

D Point view matrices used in section 5



(a) The PVM used when using a single dense block.



(b) The PVM used when stitching dense blocks.

Figure 17: Visual example of how our point view matrices looked when applying Structure from Motion in section 5 for both methods, after trying and optimizing hyperparameters.

References

[1] Gabriele Guidi, J-A Beraldin, and Carlo Atzeni. High-accuracy 3d modeling of cultural heritage: the digitizing of donatello's" maddalena". *IEEE Transactions on image processing*, 13(3):370–380, 2004.

[2] Richard I Hartley. In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 19(6):580–593, 1997.

[3] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135, 1981.

[4] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979.

[5] Matthew J Westoby, James Brasington, Niel F Glasser, Michael J Hambrey, and Jennifer M Reynolds. 'structure-from-motion'photogrammetry: A low-cost, effective tool for geoscience applications. *Geomorphology*, 179:300–314, 2012.