Project 1 - MyAutoPano Report.

Shounak Naik
Robotics Engineering Department,
Worcester Polytechnic Institute,
Worcester,MA, USA.
ssnaik@wpi.edu

Venkatesh Mullur
Robotics Engineering Department,
Worcester Polytechnic Institute,
Worcester,MA, USA.
vmullur@wpi.edu

I. Introduction

In this project, we learned about how corners are better feature than edges and why we take into consideration corners in the images. We implemented important concepts in computer vision like Adaptive Non-Maximal Suppression, feature mapping, RANSAC and Homography. The main objective of the project was to stitch two or more images to create a panorama. The images had some amount of overlap and we implemented stitching of panoramas using classical computer vision technique and a modern deep learning technique. Homography is used to orient the images and make a better panoramic effect.

II. Phase I

In this section, we put forth our approach to stitch images together to form a panorama. This is done by extracting feature in all the images and trying to find the best match between the feature points. To implement this we are considering corners as our features since they can be easily distinguished using the neighbouring pixels. These corners are then uniformly distributed using Adaptive Non-Maximal Suppression which are then converted into feature vectors or feature descriptors. Basically, every good feature/corner in all the images are represented as a feature vectors. These feature vectors are then mapped to features in other images to find the best match between them. To discard the incorrect matching between the features, we use RANSAC where we discard the "outliers" and only keep the "inliers". Finally the homography of the second image with respect to the first image is calculated which is then used to warp images and stitch them. The two images having some overlap are shown below which are supposed to be stitched together to form a panorama.

Fig. 1. Images to be stitched.

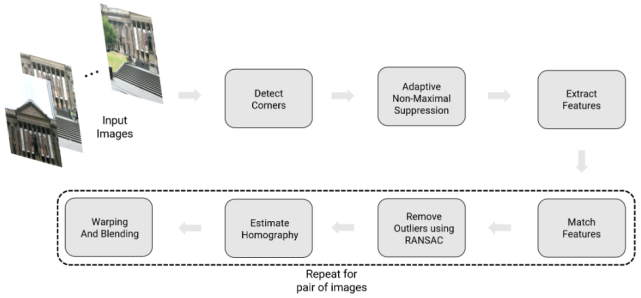

Fig. 2. Overview of the classical method.

A. Corner Detection:

- Corners are supposed to be good features in an image because they have varying gradients in all directions and we can easily detect corners using the neighbouring pixels. Where as edges have different gradients in only two directions.

- CV2 offers two different techniques to detect corners namely- cv2.goodfeaturestotrack and cv2.harriscorners. We have used the Harris Corner detection method to detect corners.

- The output of the corner detector (Harris corners) is shown in the fig. 3 . We can notice that it forms blobs near the corners while if we use the other method from CV2, which has inbuilt ANMS, the output looks well distributed and robust. Since, we have implemented our own ANMS which is described in the next section, we decided to use Harris Corners method.

B. Adaptive Non-Maximal Suppression:

- As mentioned earlier in the report, we see blobs of corners after detecting corners using Harris corner detection. To make it look better, uniformly distributed and bring out local maximas and suppressing the unnecessarily detected corners, we use ANMS or Adaptive Non-Maximal Suppression.

Fig. 3. Corner Detection of first image.

Fig. 4. Corner Detection of second image.

– In some case, due to the orientation of the image captured, the corners might not be very sharp. So these corners might get multiple hits in corner detection. ANMS tries to suppress redundant corners and make them well distributed.

– The harris corner detector gives us the scores of how much the detected corner is actually a corner depending upon the parameters given. Using maximum filter which corresponds to "imregionalmax" in matlab, we only passed the local maximas and called them Nbest corners.

Fig. 5. Corner Detection of third image.

Initialize $r_i = \infty$ for $i = [1 : N_{strong}]$

for $i = [1 : N_{strong}]$ **do**
  for $j = [1 : N_{strong}]$ **do**
    if $(C_{img}(y_j, x_j) > C_{img}(y_i, x_i))$ **then**
      ED $= (x_j - x_i)^2 + (y_j - y_i)^2$
    end
    if $ED < r_i$ **then**
      $r_i =$ ED
    end
  end
end

Sort r_i in descending order and pick top N_{best} points

Fig. 6. Adaptive Non-Maximal Suppression Algorithm.

– In the fig.6, a pseudocode of ANMS implementation is given. Figure 7 and 8 actually show the ANMS implemented by us.

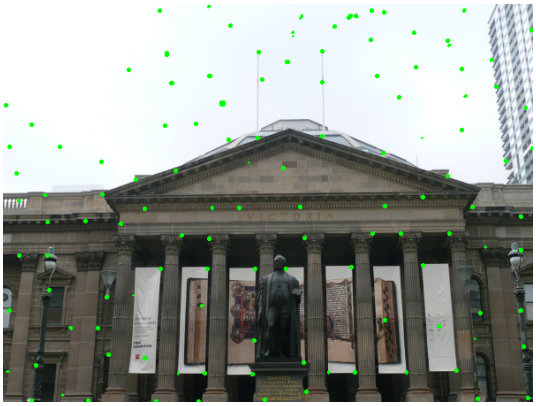

Fig. 7. ANMS on first image

Fig. 8. ANMS on second image

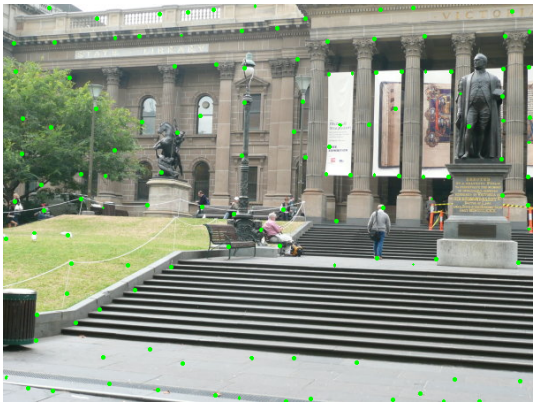

Fig. 9. ANMS on third image

C. Feature Descriptor and Mapping:

- Now we have all the feature points which are the Nbest points in both the images. These points are basically local maxima corners that are given out by ANMS.

- Due to the overlap in the two images and our next objective is to map each point in one image to the corresponding point on the other image, we need to describe every point with some descriptor.

- That is why we are creating a feature descriptor as a feature vector corresponding to every point in all the images.

- These features are taken as a patch of 40*40 with the feature point at the centre and convolving this patch with a gaussian filter. This patch is then sub-sampled to 8*8 patch and then converted into a vector of 64*1. This is shown below in the figure 8.

Fig. 10. Feature Vector/descriptor.

- After getting a feature vector for every Nbest point in both the images, our main task is to map these features. Mapping is basically finding out a match of the feature from one image to the other.

- This mapping is done by choosing one point in the first image and then computing sum of square differences (SSD) between all the points in the second image. We already have initialized the lowest distance and second lowest distance variables as infinity in the beginning. Then for each distance, if the distance is lesser than the lowest distance, then update it; similarly with the second lowest distance.

- Finally by taking the ratio of the lowest distance to the second lowest distance and checking if it is greater than a threshold, then we consider it as best matches.This way we find the matches in the two images which are shown the figures 11, 12 below.

Fig. 11. Feature Mapping from first image to the second.

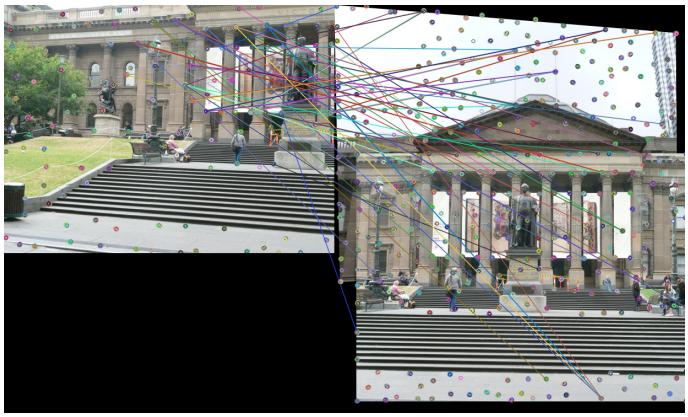

Fig. 12. Feature Mapping from second image to the third.

D. RANSAC (Random Sampling Consensus:

- Our main objective of this step is to calculate homography between the two images. Homography is similar to tranformation but on images. To calculate accurate homography we must check whether we have correctly matched the features.

- Using RANSAC, we are trying to incorrect mapping between two features and then calculating homography. Since we want to calculate homography, we have four unknown variables.

- We take these random 4 points in the both the images and then we calculate homography between them. Now we discard the incorrect matches by taking the points in the transformed first image and the points in the second image then calculate the distance between them. If the distance is below the threshold then call them inliers, if not then these the outlier that are to be discarded.

– Repreating this steps for a number of iterations and keeping the largest set of inliers, we get an accurate homography matrix between the inliers.

Fig. 13. RANSAC between first and second Image

– Now we have the homography between the first and the second image. Now we can warp the first image with respect to the second. The warped Image is shown in the figure 14 below.

Fig. 14. Warped Image

E. Stitching and Blending:

– Now that we have the warped first image, we need to come up with some logic to place the second image so that they stitch properly.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad \begin{aligned} x' &= u/w \\ y' &= v/w \end{aligned}$$

Fig. 15. Coordinates (0,0) of the second image

– Figure 15. shows us the new coordinates of the second image. It will give us minimum and maximum value of the X' and Y'.

– Upon trial and error method, we realised that there is a displacement of the first image to left because of calculating the homography. So we shift it back to the (0,0) using another transformation.

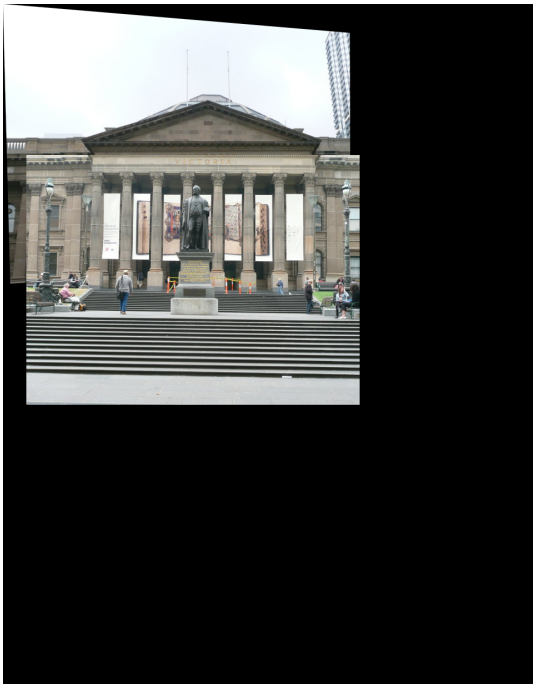

Fig. 16. Stitched Image

– We tried Alpha blending, but due to time constraints, we did not move ahead. But, we can try Poisson Blending too. For stitching the third image,there were no inliers between the stitched and the third image hence we could not stitch it. But, we some of the image relevant to the project are given below. They are the snapshots of what all we tried.

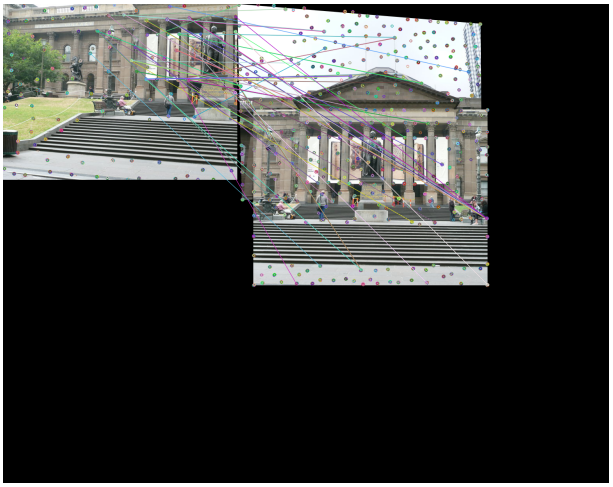

Fig. 17. Feature mapping of the stitched image and the third Image

Fig. 19. Corner detection2

Test Set:

Fig. 18. Corner detection1

Fig. 20. ANMS1

Fig. 21. ANMS2

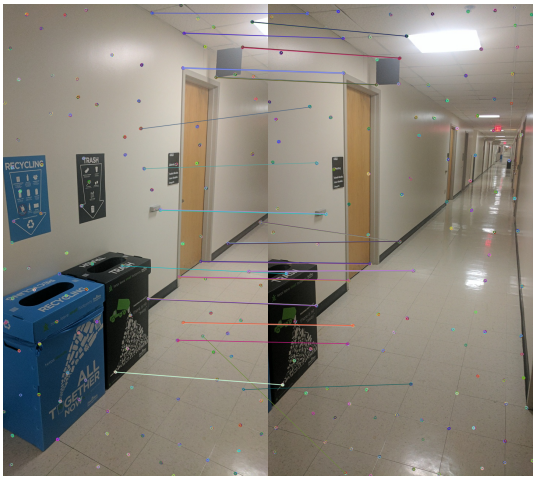

Fig. 22. Feature Mapping

Fig. 23. Stitching

Fig. 24. Warping

III. PHASE II:

With the use of Deep Learning, everything we did in the first phase can be done in one shot without even actually tuning some important parameters. Corner detection, ANMS, Feature Mapping, RANSAC and Blending is done using Deep Learning. Deep Learning enables us to create a robust pipeline to stitch panoramas and with greater speeds. In this section we are implementing two networks - one supervised and one unsupervised. The results and training/testing losses are mentioned below.

A. Data Generation:

- To calculate homography between two images in a dataset can a little heavy because we need to warp one of the images in 3D space which might take a lot of time and it can be computationally expensive. For that we are creating a synthetic dataset taken from MSCOCO dataset. Since, this dataset is a very large, and the training would require a lot of time and memory, we are working on a chunk of this dataset.

- MSCOCO dataset contains a lot of objects in natural images and the network we are implementing needs to have same size of the inputs. We are trying to implement a HomographyNet which does not accept arbitrary sizes of the images. In order to do that we are taking 128*128 patch in the middle of the image so that it does not go out of bound when we perturb it with some threshold.

- As shown in the fig.25 the blue patch is the first patch and the second patch is the perturbed patch. This perturbation is achieved by taking the corners of the blue box and adding/subtraction random values to it. The threshold to these addition/subtraction in our case is set to 16 units.

- To ensure that this perturbation does not go out of bounds, we try to place the patch in the

Fig. 25. The two patches & the perturbations as input to the HomographyNet.

mid portion of the images. This is done on all the images and then resized to the size of the first image.

- This perturbation is done on the fly, that means for every epoch the patch selected is different and hence the perturbations also change. This ensures data augmentation and robust training.

- The two patches on the image in the fig.25 are the inputs to the network. But since the network does not accept arbitrary sizes, we need to come up with some logic to do so.

- To do that, since we have corners of the two patches, we can find the homography between them. Then we do the inverse of that homography matrix, now essentially we can go from the second patch to the first patch.

- Now this inversed homography matrix is multiplied by the first image to get a square patch (which was perturbed earlier) and the coordinates of the corners of the first images will the corners in the second image (perturbed).

- Essentially we warped the image in such a way that the perturbed patch looks square. Now this patch is saved in a pickle array and is given as input to the network with the first random patch generated in the first step.

B. Supervised Approach:

- The basic idea of this network is giving 2 ima as inputs to the network with known homogra between them and the output of the network sh be 8 points (4 corners). These 4 corners are warped corners of the patch.

- Basically, we are doing regression. The first i is a patch, the second is the perturbed version of the patch with the same size. The labels are the

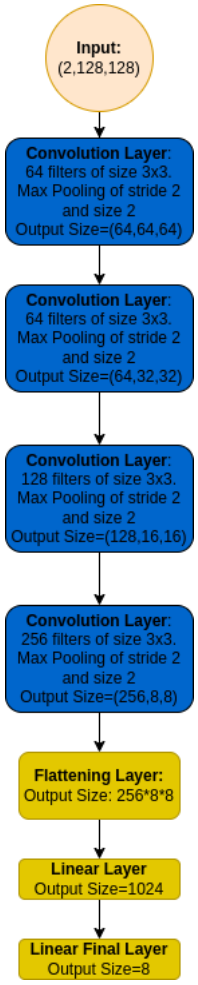

Fig. 26. Architecture of the HomographyNet.

homography between them which is calculated by CA - CB where CA and CB are the corners of the patches A and B. This is called as $H4pt$. It is then converted into a homography matrix by using DLT.

- We are expecting the output to be the 4 corners of the H4pt as shown in the fig.27. As shown in the fig.26 we have implemented a similar deep learning model with half the layers.

- Batch size = 64
- Number of epochs = 50
- Optimizer = Adam Optimizer.

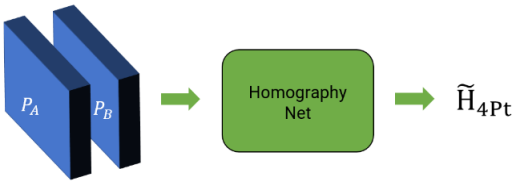

Fig. 27. Flow of the HomographyNet.

– The network will throw out 8 values of the H4pt or 4 point homography. The loss function would be the L2 loss of the predicted 4 point homography and the ground truth of the homography.
$||pred(H4pt) - H4pt||2$

C. Results:

- The output of the network is the 4 point homography, when we add the original corner points to the output of the network to get the homography matrix. Our model is a little bit overfitted but, by changing some hyperparameters and number of epochs, we can achieve a better accuracy.

- The training loss of our model is shown in the fig.28. And the results are shown in the fig.29 and fig.30

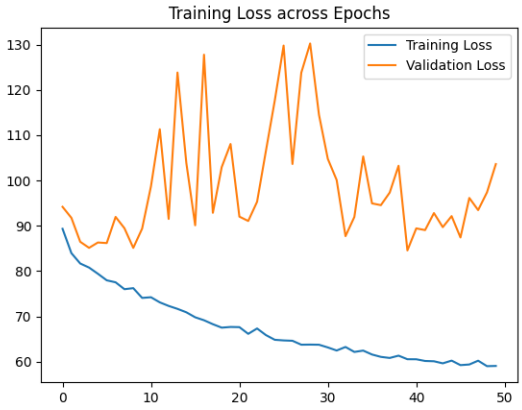

Fig. 28. Training Loss.

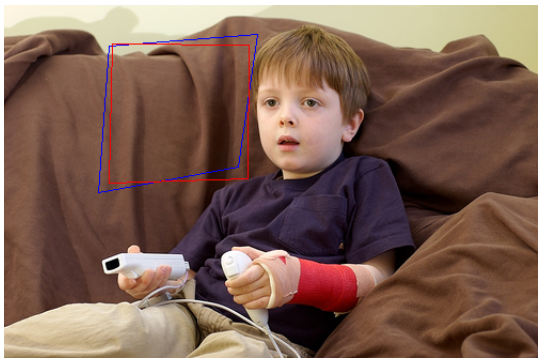

Fig. 29. Test Image 1.

- The blue box in the fig.29 and fig.30 is the ground truth of the Patch B and the red one is the predicted one.

D. Unsupervised

The unsupervised learning pipeline is an extension of the supervised learning pipeline. The overview of the pipeline

Fig. 30. Test Image 2.

can be seen in Figure 31. The Homography Net that we used in the unsupervised learning is the same as the one used in the Supervised Learning part. After we get the $H4pt$, we use the TensorDLT method to convert $H4pt$ into the H (homography matrix). This Homography matrix is further used in the *Spatial Transformer Network* to warp the *Patch A*. This warped *Patch A* is then compared with the the *Patch B*. They are compared with each other with a photometric loss which essentially is a pixel-wise $L1 - norm$. This loss is backpropogated to optimize the entire network.

1) TensorDLT: TensorDLT does the similar function as the *cv2 getPerspectiveTransform*. The difference between these two methods is that the TensorDLT gives a differentiable output. TensorDLT converts the 8 dimensional $H4pt$ into $3*3$ H matrix. This is done by formulating this problem as a $Ax = B$. By solving for x, we get the Homography matrix. The A matrix is obtained by stacking A_i like in the Figure 32.

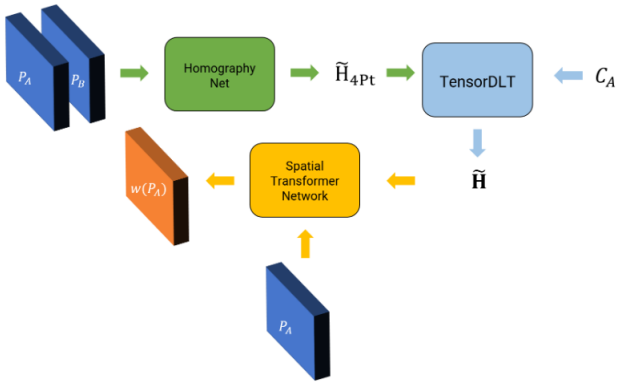

Fig. 31. Unsupervised Learning Pipeline

$$\hat{\mathbf{A}}_i = \begin{bmatrix} 0 & 0 & 0 & -u_i & -v_i & -1 & v_i'u_i & v_i'v_i \\ u_i & v_i & 1 & 0 & 0 & 0 & -u_i'u_i & -u_i'v_i \end{bmatrix},$$

Fig. 32. Stack A_i's like this for every point pair to form A

2) Spatial Transformer Network: This network helps us warp a given image with the homography matrix. This layer in the network also has to be differentiable since the backpropogation flows through this. After getting the warped image, we use photometric loss to find our error between the *predicted Patch B* and *actual Patch B*. This part was not implemented by us and we directly used a third party API - `kornia` to implement this.

3) Results and Implementation Problems: While generating the batches for corners, we faced interesting issues. By tackling these issues we learned how to collate and shape the batch so that we can use it in the structure of our code. After fixing errors, we saw that the loss is not decreasing for our unsupervised pipeline. We could not understand how we could solve this issue in the given amount of time. Nonetheless, assuming that our network would have been trained correctly, we wrote the Testing code. The testing code is similar to the supervised part where we take the $H4pt$ predicted and then we get the *predicted patch B*.

The training loss graph of this section can be seen in Figure 33. A sample test image of this technique can be seen in Figure 34

Fig. 33. Training Loss for unsupervised technique.

IV. Conclusion:

In the first phase we stitched the overlapped images into a seemless panorama, but when stitching the third image, there were no inliers found so we could not show the final results. But we could stitch 2 images from the test set and the result is good. We can improve these results by detecting better corners and making the hyperparameters tune. We can also find a better logic in stitching and blending that might make the output look even better.

The results obtained from the deep learning which we implemented were not very encouraging because we could not get much time to tune the hyperparameters. With the tuning, we are sure to get better results. The mean loss

Fig. 34. Test image for unsupervised method. Blue lines mean the *patch B* and Red Lines mean the *predicted Patch B*

obtained in the supervised learning was 81.44 and in the unsupervised part was 84.26. In the unsupervised learning model, the loss is not varying over epochs and we suspect that the backpropogation is not being implemented. Nonetheless, we got to learn and experience a lot over this project and since we have implemented it from scratch we can say we know the concept behind it.

This project was a great experience for us as we got to learn both the classical and neural network pathway to solve the image stichting problem. We found the geometric concepts very fascinating. We also learnt a lot on how to use the cluster and to do remote computing. We also learnt to use `vim` which is a very good skill to have. We look forward to the next projects in this course.

REFERENCES

[1] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/1606.03798.pdf.
[2] https://cocodataset.org/home.
[3] https://arxiv.org/abs/1709.03966.
[4] https://github.com/kornia/kornia/tree/master/examples/homographyregression.
[5] https://github.com/abhi1625/CMSC-733/blob/462fab67e46246794bdbda4c84d2bd77b54432af/Abhi1625p1/Phase1/Code/MyPano.pyL204
[6] https://github.com/p-akanksha/MyAutoPano/blob/051912d9c9656af57bd90db75f7bfff94f27225c/Phase1/Wrapper.py.
[7] https://github.com/advaitp/My-AutoPano/blob/9960f44b6be65a3dd1c6c28216add5e1ee52d3a0/Report.pdf
[8] https://cs231n.github.io/.
[9] https://medium.com/@navekshasood/image-stitching-to-create-a-panorama-5e030ecc8f7.
[10] https://ignitarium.com/use-of-homography-matrix-for-image-stitching/.
[11] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://momohuang.github.io/assets/img/Panorama/panoramareport.pdf.
[12] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://courses.cs.washington.edu/courses/cse576/16sp/Slides/10ImageStitching.pdf.