Feature Point Detection

Now & Then: A historic mosaic using Harris Stephens Corner Detector and the SIFT feature descriptor

Anisa Zhurda
Rabbir Bin Rabbani
University of Stavanger, Norway
a.zhurda@stud.uis.no,rb.rabbani@stud.uis.no

ABSTRACT

In this project, we present an implementation and analysis of Harris corner detector and SIFT descriptor to match and overlap images of cities and historic locations in different time periods. The algorithm comprises seven steps, including choosing the right pair of images based on the quality of the images and visual similarities, detecting feature points with Harris Corner Detector,using SIFT to extract the local neighbors around every key point from the pair of images and form descriptors,computing euclidean distances between descriptors from the two images, select the putative matches based on the distance matrix of descriptors and a chosen threshold,using RANSAC algorithm to find the model fitting the matches and combine the images to get the final result. The experiments analyze the parameter combination for Harris and SIFT algorithms,and we take it a step further by exploring RANSAC to compute homography ,a transformation from one projective plane to another.

KEYWORDS

Feature detection, Harris Stephens, Corner Detector, SIFT, RANSAC, Homography, Putative Match, Euclidean distance

1 INTRODUCTION

We all like feeling nostalgic and nothing brings more nostalgia then a look into the past with photos and comparing it to the present. In this project, we have collected pictures of various places in different time periods and have tried to put an image of the past on top of the present location. For doing this we discovered Soren Weber's work on 'Now & Then'[1]. We selected some of his images from his gallery ,that we thought would have higher chance to match and give desirable results.

2 THEORY

This section provides the necessary theory needed for understanding the techniques used for feature point detection and homography transformation to combine the pair of images and represent the concept of how time changes everything.

2.1 Harris Stephen's Corner Detector

The Harris corner detector is an improvement of Moravec detector [3] ,one of the earliest corner detection algorithms that relies on the autocorrelation function of the image for measuring the intensity differences between a patch and windows shifted in different directions.

Feature Point Detection, IDE, UiS
2019.

The popularity of Harris Stephens detector [2] resides in the easy interpretation and the efficiency achieved by considering the differential of the corner score instead of using shifted patches. In order to locate points with strong intensity variations in a local neighborhood ,the algorithm relies on the information of the autocorrelation matrix and the analysis of its eigenvalues.

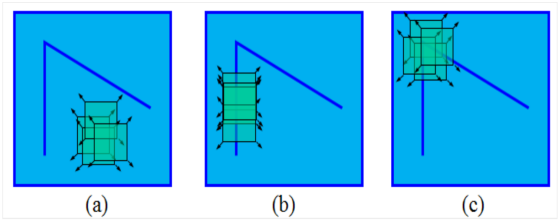

Figure 1: Harris Detector

In a smooth area (a) ,the window does not change in all directions. On the edge as shown in (b), the window does not change in the direction of the edge. At the corner point (c), the window has a change in all directions. Harris corner detection makes use of this intuitive physical phenomenon to determine whether it is a corner point by the degree of change in the window in all directions.[4]

2.2 SIFT Feature Descriptor

Scale Invariant Feature Transform(SIFT)[5] is an image descriptor for matching images and recognizing points in image, which was developed by David Lowe. This descriptor is used alongside other related image descriptors for a large number of purposes in computer vision related to point matching between different views of a 3-D scene and view-based object recognition. The SIFT descriptor is invariant to translations, rotations and scaling transformations in the image domain and robust to moderate perspective transformations and illumination variations. It has been proven to be very useful in practice for image matching and object recognition under real-world conditions.

SIFT transforms an image into a large collection of feature vectors, each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion. These features share similar properties with neurons in primary visual cortex that are encoding basic forms, color and movement for object detection in primate vision.

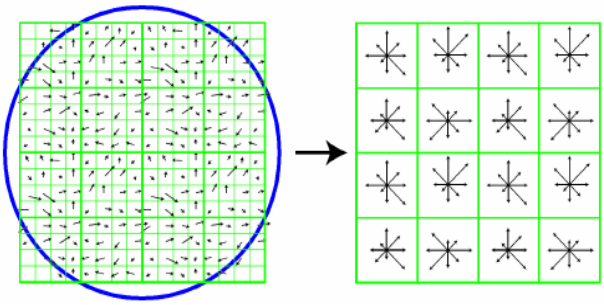

Figure 2: SIFT descriptor

2.3 Random Sample Consensus(RANSAC)

Random sample consensus (RANSAC)[7] is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates. It is a non-deterministic algorithm in the sense that it produces a good result only with a certain probability, this probability increases as more iterations.

The RANSAC algorithm[8] is a learning technique to estimate parameters of a model by random sampling of observed data. Given a dataset whose data elements contain both inliers and outliers, RANSAC uses the voting scheme to find the optimal fitting result. Data elements in the dataset are used to vote for one or multiple models. The implementation of this voting scheme is based on two assumptions: that the noisy features will not vote consistently for any single model (few outliers) and there are enough features to agree on a good model (few missing data). The RANSAC algorithm is essentially composed of two steps that are iteratively repeated:

(1) In the first step, a sample subset containing minimal data items is randomly selected from the input dataset. A fitting model and the corresponding model parameters are computed using only the elements of this sample subset. The cardinality of the sample subset is the smallest sufficient to determine the model parameters.
(2) In the second step, the algorithm checks which elements of the entire dataset are consistent with the model instantiated by the estimated model parameters obtained from the first step. A data element will be considered as an outlier if it does not fit the fitting model instantiated by the set of estimated model parameters within some error threshold that defines the maximum deviation attributable to the effect of noise.

The set of inliers obtained for the fitting model is called the consensus set. The RANSAC algorithm will iteratively repeat the above two steps until the obtained consensus set in certain iteration has enough inliers.

The input to the RANSAC algorithm is a set of observed data values, a way of fitting some kind of model to the observations, and some confidence parameters. RANSAC achieves its goal by repeating the following steps:

(1) Select a random subset of the original data. Call this subset the hypothetical inliers.
(2) A model is fitted to the set of hypothetical inliers.
(3) All other data are then tested against the fitted model. Those points that fit the estimated model well, according to some model-specific loss function, are considered as part of the consensus set.
(4) The estimated model is reasonably good if sufficiently many points have been classified as part of the consensus set.
(5) Afterwards, the model may be improved by reestimating it using all members of the consensus set.

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are part of the consensus set, or a refined model together with a corresponding consensus set size. In the latter case, we keep the refined model if its consensus set is larger than the previously saved model.

3 IMPLEMENTATION

3.1 Image Collection

As we mentioned in introduction we have used Soren Weber's art work [1] who uses this platform to show how the world has changed over time. Since the concept behind his work is based in a more historic viewpoint ,the photos tend to be too complex for feature detection and matching. In addition some transformations are really extreme,and comparison in terms of corner features is not viable. Hence ,the process of choosing the right pair of Then & Now images was a challenge that we solved by intuitive visual comparison and trial-error.

Figure 3: Extreme historical transformation

In the figure above,the post warrior transformation is really extreme and it is impossible to match features and stitch the images together.

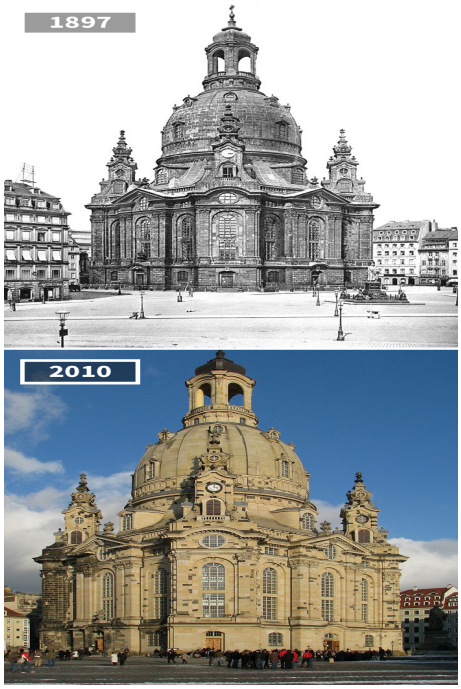

Figure 4: Complex buildings

In figure 4, the building in the present time looks really similar to the old one from our perception,but the complex features in the old building made the matching impossible .

Figure 5: Example of successful match

In the figure above ,even though the images are taken in really large time interval,the position of the bridge in both images is giving us a lot of matching corners and the stitching is successful with most parameter combination.

3.2 Code Structure

For simpler interpretation and flexibility in changing the parameters we have one main function that can be called in the main file. This main function is called 'image_mosaic' and it contains all the main logic for the mosaic operations.There are minor functions to further break this function into modules, e.g. the 'harris' function does the harris corner detection, 'find_sift' finds the sift descriptors and so on.

3.3 Parameter tuning

As expected, each pair of images had its own parameter combination and had to be processed separately. We started by initializing the parameters:

- Harris corner detection function (parameters: sigma , threshold, and radius) to find points of interest.
- SIFT descriptor (parameters: radius, enlarge_factor) used to extract the neighborhood information around the points of interest.
- RANSAC algorithm: 200 matches to fit and 5000 iterations to optimize the homography (H). For each iteration, distances smaller than a threshold of 0.5 are chosen as inliers .

We have put the pair of images in separate folders.

One of the most important steps was the tuning of the parameters and saving all the results (as stitched images) from the parameter combinations in a folder for each of the image pair,and also saving the parameter combinations on a csv file along with the link to the output image. After exploring the parameters and the effects they have on the output image,we decided to tune only the sigma, threshold, radius for the Harris corner detector and the radius for the SIFT descriptor .

3.4 Image Mosaic Function

All the functionalities are structured inside image_mosaic function, following these steps-

(1) Load images and make them gray-scale.
(2) Get corner points using Harris.
(3) Get Descriptors using SIFT and the previously found corner points.
(4) Find the euclidean distances between matching feature descriptors.
(5) Using the distances, find the points that match best.
(6) Use RANSAC to compute transformation from one projective plane to another.
(7) Generate the Stitched image

3.5 Harris Corner detector

This function is done in the 'harris.m' file. It takes the harris parameters and returns the row and column coordinates of the corners. There is a display parameter on this function that, when set as '1' outputs the images with the corner points.

Figure 6: Matching Corners on the new image

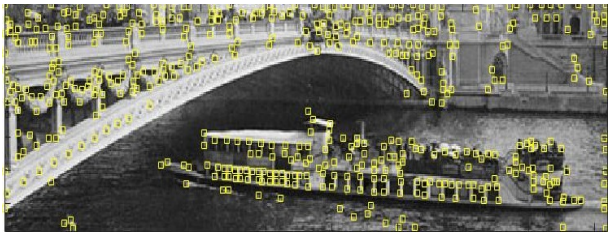

Figure 7: Matching Corners on the old image

3.6 SIFT Descriptor

This function is done in the 'find_sift.m' file.It takes a double gray scale image,the detected corner points,and sift parameters and it forms descriptors by flattening the pixel values in each neighborhood to one dimensional vectors. Then we are using the function select_putative_matches to finding the putative matches using the sift descriptors.

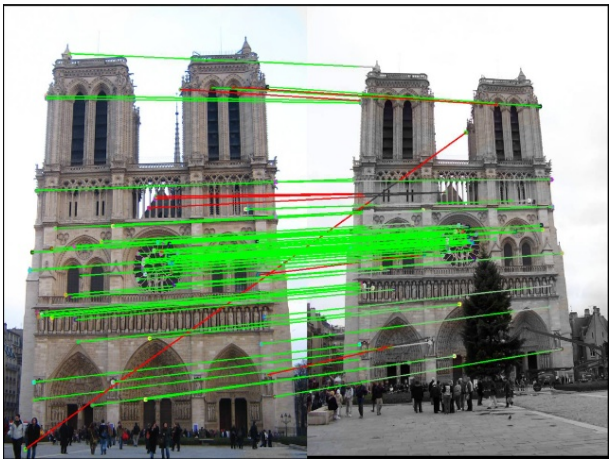

Figure 8: Putative matches

3.7 RANSAC

This function is made in the 'ransac.m' file. It takes the matching coordinates for both the images and the number of iterations and estimates the homography mapping of one image onto the other.

Then we warp one image on top of the other using the estimated transformation from RANSAC and display our result.

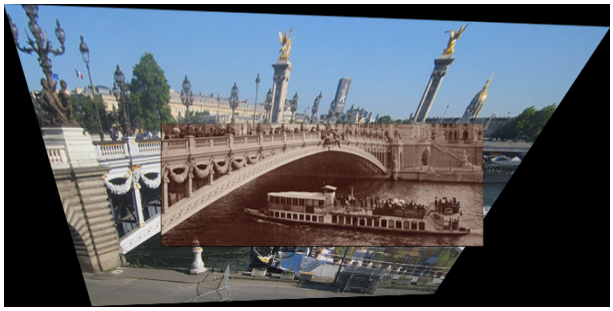

Figure 9: Ransac Output Case 1

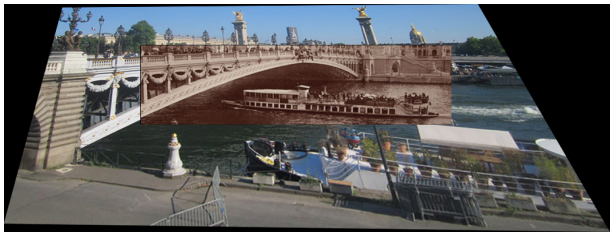

Figure 10: Ransac Output Case 2

4 EXPERIMENTS AND RESULTS

In our experiments, we used 6 pairs of Now and Then images to stitch them together into a historic mosaic.We analyzed the two images and cropped a part of the old image that had a high chance to align with the new one.Then after going through all the implemented steps ,we match these two images by the feature points and have the following final stitched images.

4.1 Working Results

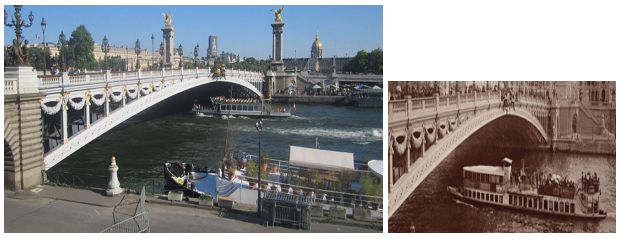

Figure 11: Now & Then:Pair 1

Pair 1 shows Paris, France in the years 1900 and 2017 . The parameter combination that gave the best results in matching the feature points and overlapping the images are : Harris corner detection function (parameters: sigma=1 , threshold=0.02, and radius=1) SIFT descriptor (parameters: radius=3)

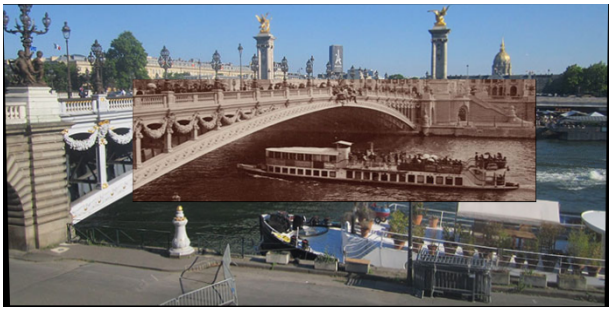

Figure 12: Result 1

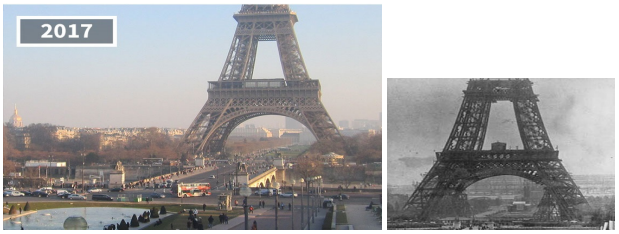

Figure 13: Now & Then:Pair 2

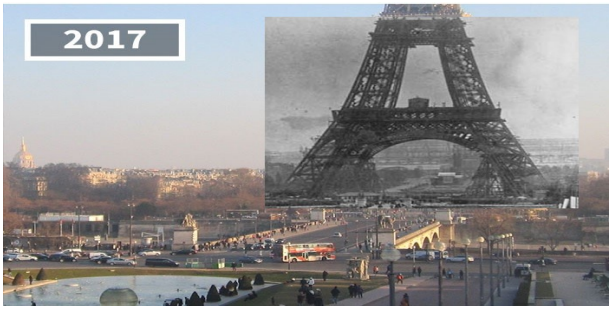

Figure 14: Result 2

Pair 2 shows Paris,the Eiffel Tower in the years 1888 and 2017.There is a high resemblance in the corners around the Eiffel tower which gives us a really good overlap using the following parameters:

- Harris corner detection function (parameters: sigma=1 , threshold=0.02, and radius=2)
- SIFT descriptor (parameters: radius=1)

Figure 15: Now & Then:Pair 3

Figure 16: Result 3

Pair 3 shows Seljestadjuvet,Odda Norway in years 1887 and 2004.The panoramic view has some distinguishable features around the mountain and lake which perfectly harmonize using the parameters:

- Harris corner detection function (parameters: sigma=1.5 , threshold=0.02, and radius=2)
- SIFT descriptor (parameters: radius=3)

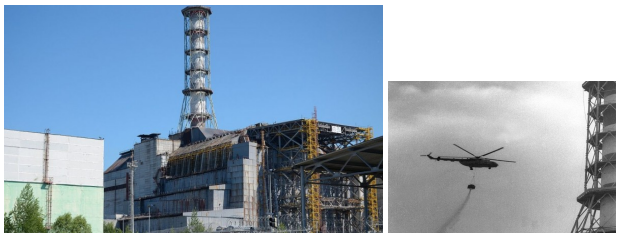

Figure 17: Now & Then:Pair 4

Figure 18: Result 4

In the fourth pair ,the old image shows a historic moment when the helicopter drops concrete onto one of the reactors of the Chernobyl nuclear power plant in the year 1986. We managed to stitch this image to a photo of recent years in Chernobyl.

- Harris corner detection function (parameters: sigma=1 , threshold=0.2, and radius=2)
- SIFT descriptor (parameters: radius=3)

Figure 19: Now & Then:Pair 5

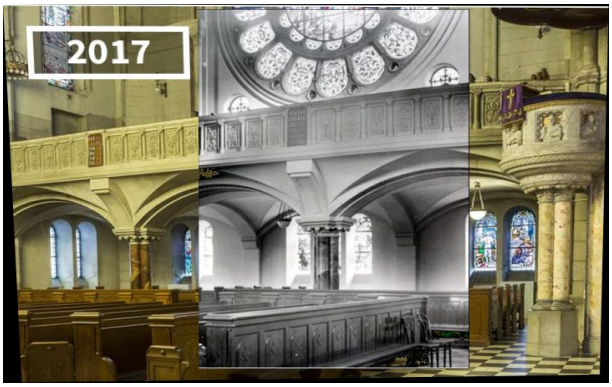

Figure 20: Result 5

Pair 5 shows St. Matthew Evangelical Church in Poland in years 1937 and 2017. This is one of the few churches that did not change much after the difficult war years ,the matching in the stained glass and church archs was quite visible using the parameters:

- Harris corner detection function (parameters: sigma=1 , threshold=0.01, and radius=1)
- SIFT descriptor (parameters: radius=3)

Figure 21: Now & Then:Pair 6

Figure 22: Result 6

Pair 6 shows how a historic building has preserved its architectural characteristics through years in Lodz, Poland, (1887 - 2015).The best result is achieved by :

- Harris corner detection function (parameters: sigma=2 , threshold=0.04, and radius=3)
- SIFT descriptor (parameters: radius=3)

4.2 Irregularities

From all the parameter combinations only a small percentage of the results were accurate and gave us the desirable stitched image. Some of the usual irregular image outputs from the pairs we analyzed above:

Figure 23: Irregular result Pair 1

Figure 24: Irregular result Pair 5

Figure 25: Irregular result Pair 6

5 CONCLUSION

In this work, we presented the implementation and one possible application of the Harris Stephen corner detector and SIFT descriptor. Furthermore we introduced RANSAC to find the optimal fitting result to create the preferred image projection. We implemented the algorithm step by step and analyzed different alternative parameters for each pair of images we wanted to stitch.This project emphasizes even more the importance of parameter selection and combination in more complex images taken from the real life and with different resolutions due to the time the images were taken. Nevertheless ,we wanted to show how a mathematical approach to detect feature points can actually be used in such an exciting application to tell fascinating stories and bring nostalgia from the time gap images stitched in one historic mosaic.

Finally, we have uploaded the code to a github repository which can be found in this link 'https://github.com/RabbirBR/ImageProcessingPr since our output dataset is large we have also uploaded the zipped output in this dropbox link 'https://www.dropbox.com/s/tsmh6tuvtvjblfe/output.zip?dl=0'.

REFERENCES

[1] Soren Weber *81 Before After Pics Showing How The World Has Changed Over Time By Re.Photos.* https://www.boredpanda.com/then-and-now-pictures-changing-world-rephotos, 2018.

[2] Chris Harris Mike Stephens *A COMBINED CORNER AND EDGE DETECTOR* http://www.bmva.org/bmvc/1988/avc-88-023.pdf, 1988.

[3] Hans P. Moravec *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover* https://pdfs.semanticscholar.org/3f20/e72b5e743ec4d85e91029719aac051b0d4e6.pdf, 1980.

[4] University of Washington *Harris Detector* https://courses.cs.washington.edu/courses/cse577/05sp/note

[5] David G. Lowe *Distinctive Image Features from Scale-Invariant Keypoints Volume 60, Issue 2, pp 91âĂŞ110.* https://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94, 2004. https://courses.cs.washington.edu/courses/cse577/05sp/notes/harris.pdf

[6] David G. Lowe *Boosting of factorial correspondence analysis for image retrieval.* https://www.researchgate.net/publication/4351572_Boosting_of_factorial_correspondence_analysis_f July 2008.

[7] Martin A. Fischler & Robert C. Bolles *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography.* https://apps.dtic.mil/dtic/tr/fulltext/u2/a460585.pdf, 1981.

[8] Wikipedia *Random Sample consensus* https://en.wikipedia.org/wiki/Random_sample_consensus.

Appendix

A CODE LISTING

A.1 main.m

```
%% Setting Initial Parameters
clear
clc

harris_params = containers.Map;
harris_params('sigma') = 1;
harris_params('threshold') = 0.05;
harris_params('radius') = 2;

sift_params = containers.Map;
sift_params('radius') = 2;
sift_params('enlarge_factor') = 1.5;

num_putative_matches = 200;
num_ransac_iter = 5000;


folder_name = "images\Bridge";
img_format = 'png';

res = image_mosiac(folder_name,...
    img_format, harris_params, sift_params,
     num_putative_matches,...
    num_ransac_iter, 1);
% imshow(res);


%% Iterating for House3
house3_results = [];

if ~exist('output_images/house3/', 'dir')
   mkdir('output_images/house3/');
end

folder_name = "images\House3";
img_format = 'png';

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.1
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;
```

```
52
53                      res = image_mosiac(folder_name,...
54                          img_format, harris_params,
      sift_params, num_putative_matches,...
55                          num_ransac_iter, 2);
56                      imshow(res);
57
58                      output_file_path = "output_images/
      house3/" + int2str(n);
59                      output_file_path = output_file_path
      +"."+img_format;
60                      imwrite(res, output_file_path);
61
62                      vals = [folder_name i j k l
      output_file_path];
63                      house3_results = [house3_results;
      vals];
64                      n = n + 1
65                  end
66              end
67          end
68      end
69  end
70
71  fid = fopen( 'CSVs/house3_results.csv', 'wt' );
72  writematrix(house3_results, 'CSVs/house3_results.csv');
73  fclose(fid);
74
75
76
77  %% Iterating for House1
78  house1_results = [];
79
80  folder_name = "images\House1";
81  img_format = 'png';
82
83  if ~exist("output_images/house1/", "dir")
84     mkdir("output_images/house1/");
85  end
86
87  n = 1;
88  for i = 1:0.5:3
89      for j = 0.01:0.01:0.1
90          for k = 1:1:3
91              for l = 1:1:3
92                  i = i
93                  j = j
94                  k = k
95                  l = l
96
97                  try
98                      harris_params('sigma') = i;
99                      harris_params('threshold') = j;
100                     harris_params('radius') = k;
101                     sift_params('radius') = l;
102
103                     res = image_mosiac(folder_name,...
104                         img_format, harris_params,
      sift_params, num_putative_matches,...
105                         num_ransac_iter, 2);
106                     imshow(res);
107
108
109                     output_file_path = "output_images/
      house1/" + int2str(n);
110                     output_file_path = output_file_path
      +"."+img_format;
111                     imwrite(res, output_file_path);
112
113                     vals = [folder_name i j k l
      output_file_path];
114                     house1_results = [house1_results;
      vals];
115                     n = n + 1;
116                 end
117             end
118         end
119     end
120 end
121
122 fid = fopen( 'CSVs/house1_results.csv', 'wt' );
123 writematrix(house1_results, 'CSVs/house1_results.csv');
124 fclose(fid);
125
126
127 %% Iterating for House2
128 house2_results = [];
129
130 folder_name = "images\House2";
131 img_format = 'png';
132
133 if ~exist("output_images/house2/", "dir")
134    mkdir("output_images/house2/");
135 end
136
137 n = 1;
138 for i = 1:0.5:3
139     for j = 0.01:0.01:0.1
140         for k = 1:1:3
141             for l = 1:1:3
142                 i = i
143                 j = j
144                 k = k
145                 l = l
146
147                 try
148                     harris_params('sigma') = i;
149                     harris_params('threshold') = j;
150                     harris_params('radius') = k;
151                     sift_params('radius') = l;
152
153                     res = image_mosiac(folder_name,...
154                         img_format, harris_params,
      sift_params, num_putative_matches,...
155                         num_ransac_iter, 2);
156                     imshow(res);
157
158
159                     output_file_path = "output_images/
      house2/" + int2str(n);
160                     output_file_path = output_file_path
      +"."+img_format;
161                     imwrite(res, output_file_path);
162
163                     vals = [folder_name i j k l
      output_file_path];
164                     house2_results = [house2_results;
      vals];
165                     n = n + 1;
166                 end
167             end
168         end
169     end
170 end
171
172 fid = fopen( 'CSVs/house2_results.csv', 'wt' );
173 writematrix(house2_results, 'CSVs/house2_results.csv');
174 fclose(fid);
175
176 %% Iterating for Paris Bridge
177 bridge_results = [];
178
179 folder_name = "images\Bridge";
180 img_format = 'png';
181
182 if ~exist("output_images/bridge/", "dir")
183    mkdir("output_images/bridge/");
184 end
185
186 n = 1;
187 for i = 1:0.5:3
188     for j = 0.01:0.01:0.1
189         for k = 1:1:3
190             for l = 1:1:3
191                 i = i
192                 j = j
```

```
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    bridge/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    bridge_results = [bridge_results;
    vals];
                    n = n + 1;
                end
            end
        end
    end
end

fid = fopen( 'CSVs/bridge_results.csv', 'wt' );
writematrix(bridge_results, 'CSVs/bridge_results.csv');
fclose(fid);


%% Iterating for Eifel Tower 1
eifel_tower_results = [];

folder_name = "images\Tower";
img_format = 'png';

if ~exist("output_images/eifel_tower/", "dir")
   mkdir("output_images/eifel_tower/");
end

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.1
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    eifel_tower/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    eifel_tower_results = [
    eifel_tower_results; vals];
                    n = n + 1;
                end
            end
        end
    end
end

fid = fopen( 'CSVs/eifel_tower_results.csv', 'wt' );
writematrix(eifel_tower_results, 'CSVs/
    eifel_tower_results.csv');
fclose(fid);


%% Iterating for Eifel Tower 2
eifel_tower_2_results = [];

folder_name = "images\EifelTower1";
img_format = 'jpg';

if ~exist("output_images/eifel_tower_2/", "dir")
   mkdir("output_images/eifel_tower_2/");
end

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.1
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    eifel_tower_2/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    eifel_tower_2_results = [eifel_tower_
    2_results; vals];
                    n = n + 1;
                end
            end
        end
    end
end

fid = fopen('CSVs/eifel_tower_2_results.csv', 'wt');
writematrix(eifel_tower_2_results, 'CSVs/eifel_tower_2
    _results.csv');
fclose(fid);


%% Iterating for OddaNorway
odda_norway_results = [];

folder_name = "images\OddaNorway";
img_format = 'jpg';
```

```

if ~exist("output_images/odda_norway/", "dir")
    mkdir("output_images/odda_norway/");
end

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.1
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    odda_norway/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    odda_norway_results = [
    odda_norway_results; vals];
                    n = n + 1;
                end
            end
        end
    end
end

fid = fopen( 'CSVs/odda_norway_results.csv', 'wt' );
writematrix(odda_norway_results, 'CSVs/
    odda_norway_results.csv');
fclose(fid);


%% Iterating for Chernobyl
chernobyl_results = [];

folder_name = "images\Chernobyl";
img_format = 'jpg';

if ~exist("output_images/chernobyl/", "dir")
    mkdir("output_images/chernobyl/");
end

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.1
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    chernobyl/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    chernobyl_results = [
    chernobyl_results; vals];
                    n = n + 1;
                end
            end
        end
    end
end

fid = fopen( 'CSVs/chernobyl_results.csv', 'wt' );
writematrix(chernobyl_results, 'CSVs/chernobyl_results.
    csv');
fclose(fid);


%% Iterating for DresdenGermany
dresden_germany_results = [];

folder_name = "images\DresdenGermany";
img_format = 'jpg';

if ~exist("output_images/dresden_germany/", "dir")
    mkdir("output_images/dresden_germany/");
end

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.1
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    dresden_germany/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    dresden_germany_results = [
    dresden_germany_results; vals];
                    n = n + 1;
                end
            end
        end
    end
end
```

```
fid = fopen( 'CSVs/dresden_germany_results.csv', 'wt' );
writematrix(dresden_germany_results, 'CSVs/
    dresden_germany_results.csv');
fclose(fid);

%% Iterating for ShepherdGlacier
shepherd_glacier_results = [];

folder_name = "images\ShepherdGlacier";
img_format = 'jpg';

if ~exist("output_images/shepherd_glacier/", "dir")
   mkdir("output_images/shepherd_glacier/");
end

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.1
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    shepherd_glacier/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    shepherd_glacier_results = [
    shepherd_glacier_results; vals];
                    n = n + 1;
                end
            end
        end
    end
end

fid = fopen( 'CSVs/shepherd_glacier_results.csv', 'wt' );
writematrix(shepherd_glacier_results, 'CSVs/
    shepherd_glacier_results.csv');
fclose(fid);

%% Iterating for St. Matthews Church
matthew_results = [];

folder_name = "images\St_Matthew";
img_format = 'jpg';

if ~exist("output_images/matthew/", "dir")
   mkdir("output_images/matthew/");
end

n = 1;
for i = 1:0.5:3
    for j = 0.01:0.01:0.10
        for k = 1:1:3
            for l = 1:1:3
                i = i
                j = j
                k = k
                l = l

                try
                    harris_params('sigma') = i;
                    harris_params('threshold') = j;
                    harris_params('radius') = k;
                    sift_params('radius') = l;

                    res = image_mosiac(folder_name,...
                        img_format, harris_params,
    sift_params, num_putative_matches,...
                        num_ransac_iter, 2);
                    imshow(res);


                    output_file_path = "output_images/
    matthew/" + int2str(n);
                    output_file_path = output_file_path
    +"."+img_format;
                    imwrite(res, output_file_path);

                    vals = [folder_name i j k l
    output_file_path];
                    matthew_results = [matthew_results;
    vals];
                    n = n + 1;
                end
            end
        end
    end
end

fid = fopen( 'CSVs/matthew_results.csv', 'wt' );
writematrix(matthew_results, 'CSVs/matthew_results.csv');
fclose(fid);
```

Listing 1: main.m

A.2 image_mosiac.m

```
function [result] = image_mosiac(folder_name, img_format,
     harris_params, sift_params, num_putative_matches,
    num_ransac_iter, stage)

    folder_images = dir(strcat(folder_name, "\*.",
     img_format));
    images = {};
    images_grey = {};
    harris_points = {};
    sift_discriptors = {};
    XY = {};


    % Find and calculate sift discriptors.
    for i = 1:length(folder_images)
        img_loc = strcat(folder_images(i).folder, '\',
     folder_images(i).name);
        images{i} = imread(img_loc);
        images_grey{i} = im2double(rgb2gray(imread(
     img_loc)));

        [r, c] = harris(...
            images_grey{i},...
            harris_params('sigma'),...
            harris_params('threshold'),...
            harris_params('radius'), lt(stage, 2));

        harris_points{i}{'r'} = r;
        harris_points{i}{'c'} = c;

        sift_discriptors{i} = find_sift(...
            images_grey{i},...
            [c, r, sift_params('radius')*ones(length(c),1
    )],...
```

```
            sift_params('enlarge_factor'));
    end
    matches = {};

    [matches{1}, matches{2}] = select_putative_matches(...
        sift_discriptors{1}, sift_discriptors{2}, num_putative_matches);

    XY{1} = [harris_points{1}{'c'}(matches{1}), harris_points{1}{'r'}(matches{1})];
    XY{2} = [harris_points{2}{'c'}(matches{2}), harris_points{2}{'r'}(matches{2})];
%     disp(length(XY{1}))
%     disp(length(XY{2}))

    [H, num_inliers, residual] = ransac(XY{1}, XY{2}, num_ransac_iter,...
        @homography_fit,@homography_tf);

    result = stitchH(images{1}, H, images{2});
end
```

Listing 2: image$_{m}osiac.m$

A.3 harris.m

```
function [r, c] = harris(im, sigma, thresh, radius, display_corners)

    dx = [-1 0 1; -1 0 1; -1 0 1];
    dy = dx';

    Ix = conv2(im, dx, 'same');
    Iy = conv2(im, dy, 'same');

    g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);

    Ix2 = conv2(Ix.^2, g, 'same');
    Iy2 = conv2(Iy.^2, g, 'same');
    Ixy = conv2(Ix.*Iy, g, 'same');

    cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); % Harris corner measure

    if nargin > 2


  sze = 2*radius+1;
  mx = ordfilt2(cim,sze^2,ones(sze));
  cim = (cim==mx)&(cim>thresh);

  [r,c] = find(cim);                  % Find row,col coords.

  if nargin==5 & display_corners      % overlay corners on original image
    figure, imagesc(im), axis image, colormap(gray), hold on
    plot(c,r,'ys'), title('corners detected');
  end

    else  % leave cim as a corner strength image and make r and c empty.
  r = []; c = [];
    end
```

Listing 3: harris.m

A.4 find$_{s}ift.m$

```
function sift_desc_arr = find_sift(I, circles, enlarge_factor)

    num_angles = 8;
    num_bins = 4;
    num_samples = num_bins * num_bins;
    alpha = 9;

    if nargin < 3
        enlarge_factor = 1.5;
    end

    angle_step = 2 * pi / num_angles;
    angles = 0:angle_step:2*pi;
    angles(num_angles+1) = []; % bin centers

    [hgt wid] = size(I);
    num_pts = size(circles,1);

    sift_desc_arr = zeros(num_pts, num_samples * num_angles);

    % edge image
    sigma_edge = 1;


    [G_X,G_Y]=gen_dgauss(sigma_edge);
    I_X = filter2(G_X, I, 'same'); % vertical edges
    I_Y = filter2(G_Y, I, 'same'); % horizontal edges
    I_mag = sqrt(I_X.^2 + I_Y.^2); % gradient magnitude
    I_theta = atan2(I_Y,I_X);
    I_theta(isnan(I_theta)) = 0; % necessary????

    % make default grid of samples (centered at zero, width 2)
    interval = 2/num_bins:2/num_bins:2;
    interval = interval - (1/num_bins + 1);
    [grid_x grid_y] = meshgrid(interval, interval);
    grid_x = reshape(grid_x, [1 num_samples]);
    grid_y = reshape(grid_y, [1 num_samples]);

    % make orientation images
    I_orientation = zeros(hgt, wid, num_angles);
    % for each histogram angle
    for a=1:num_angles
        % compute each orientation channel
        tmp = cos(I_theta - angles(a)).^alpha;
        tmp = tmp .* (tmp > 0);


        I_orientation(:,:,a) = tmp .* I_mag;
    end

    % for all circles
    for i=1:num_pts
        cx = circles(i,1);
        cy = circles(i,2);
        r = circles(i,3) * enlarge_factor;

        % find coordinates of sample points (bin centers)
        grid_x_t = grid_x * r + cx;
        grid_y_t = grid_y * r + cy;
        grid_res = grid_y_t(2) - grid_y_t(1);

        % find window of pixels that contributes to this descriptor
        x_lo = floor(max(cx - r - grid_res/2, 1));
        x_hi = ceil(min(cx + r + grid_res/2, wid));
        y_lo = floor(max(cy - r - grid_res/2, 1));
        y_hi = ceil(min(cy + r + grid_res/2, hgt));

        % find coordinates of pixels
        [grid_px, grid_py] = meshgrid(x_lo:x_hi,y_lo:y_hi);
        num_pix = numel(grid_px);
        grid_px = reshape(grid_px, [num_pix 1]);
        grid_py = reshape(grid_py, [num_pix 1]);

        % find (horiz, vert) distance between each pixel and each grid sample
```

```
        dist_px = abs(repmat(grid_px, [1 num_samples]) -
    repmat(grid_x_t, [num_pix 1]));
        dist_py = abs(repmat(grid_py, [1 num_samples]) -
    repmat(grid_y_t, [num_pix 1]));

        % find weight of contribution of each pixel to
    each bin
        weights_x = dist_px/grid_res;
        weights_x = (1 - weights_x) .* (weights_x <= 1);
        weights_y = dist_py/grid_res;
        weights_y = (1 - weights_y) .* (weights_y <= 1);
        weights = weights_x .* weights_y;

        % make sift descriptor
        curr_sift = zeros(num_angles, num_samples);
        for a = 1:num_angles
            tmp = reshape(I_orientation(y_lo:y_hi,x_lo:
    x_hi,a),[num_pix 1]);
            tmp = repmat(tmp, [1 num_samples]);
            curr_sift(a,:) = sum(tmp .* weights);
        end
        sift_desc_arr(i,:) = reshape(curr_sift, [1
    num_samples * num_angles]);
    end


%% normalize the SIFT descriptors more or less as
    described in Lowe (2004)
    tmp = sqrt(sum(sift_desc_arr.^2, 2));
    normalize_ind = find(tmp > 1);

    sift_desc_arr_norm = sift_desc_arr(normalize_ind,:);
    sift_desc_arr_norm = sift_desc_arr_norm ./ repmat(tmp
    (normalize_ind,:), [1 size(sift_desc_arr,2)]);

    % suppress large gradients
    sift_desc_arr_norm(find(sift_desc_arr_norm > 0.2)) =
    0.2;

    % finally, renormalize to unit length
    tmp = sqrt(sum(sift_desc_arr_norm.^2, 2));
    sift_desc_arr_norm = sift_desc_arr_norm ./ repmat(tmp
    , [1 size(sift_desc_arr,2)]);

    sift_desc_arr(normalize_ind,:) = sift_desc_arr_norm;



function [GX,GY]=gen_dgauss(sigma)

f_wid = 4 * floor(sigma);
G = normpdf(-f_wid:f_wid,0,sigma);
G = G' * G;
[GX,GY] = gradient(G);

GX = GX * 2 ./ sum(sum(abs(GX)));
GY = GY * 2 ./ sum(sum(abs(GY)));
```

Listing 4: find$_sift.m$

A.5 dist.m

```
function distance = dist(x, c)
[ndata, dimx] = size(x);
[ncentres, dimc] = size(c);
if dimx ~= dimc
  error('Data dimension does not match dimension of
    centres')
end

distance = (ones(ncentres, 1) * sum((x.^2)', 1))' + ...
  ones(ndata, 1) * sum((c.^2)',1) - ...
  2.*(x*(c'));

% Rounding errors occasionally cause negative entries in
    distance
if any(any(distance<0))
  distance(distance<0) = 0;
end
```

Listing 5: dist.m

A.6 select$_putative_matches.m$

```
function [m1, m2] = select_putative_matches(d1, d2, num)
    d1 = zscore(d1')';
    d2 = zscore(d2')';

    distance = dist(d1,d2);

    [h,w] = size(distance);

    % Lowe's ratio test
    m1 = [];
    m2 = [];
    for i = 1:h
        [~,idx]= sort(distance(i,:));
        if distance(i,idx(1))/distance(i,idx(2)) < 0.5
            m1 = [m1;i];
            m2 = [m2;idx(1)];
            if length(m1) == num
                break;
            end
        end
    end
end
```

Listing 6: select$_putative_matches.m$

A.7 ransac.m

```
function [H, num_inliers, residual] = ransac(XY_src,
    XY_des, num_ransac_iter,...
    homography_fit, homography_tf)

    num_inliers = 0;
    plot_num_inliers = zeros(1,num_ransac_iter);
    plot_residual = zeros(1,num_ransac_iter);
    for i = 1:num_ransac_iter
        ind = randperm(size(XY_src,1));
        ind_s = ind(1:4);
        ind_r = ind(5:end);


        tmp_H = homography_fit(XY_src(ind_s,:), XY_des(
    ind_s,:));
        predict = homography_tf(XY_src(ind_r,:),tmp_H);

        dists = sum((XY_des(ind_r,:) - predict).^2,2);

        inlier_idx = find(dists < 0.3);
        tmp_num_inliers = length(inlier_idx);

        if tmp_num_inliers > num_inliers
            H = tmp_H;
            num_inliers = tmp_num_inliers;
            residual = mean(dists(inlier_idx));
        end
        plot_num_inliers(i) = num_inliers;
    end
end
```

Listing 7: ransac.m

A.8 homography$_fit.m$

```
function H = homography_fit(XY, XY_)
    A = [];
    for i = 1:size(XY,1)
        Xi = [XY(i,:) ';1];
        xi_ = XY_(i,1);
        yi_ = XY_(i,2);

        A = cat(1,A,cat(2,zeros(1,3),Xi',-yi_*Xi'));
        A = cat(1,A,cat(2,Xi',zeros(1,3),-xi_*Xi'));
    end
    [~,~,V]=svd(A);
    H = V(:,end);
    H = reshape(H,[3 3])';
end
```

Listing 8: homography$_fit.m$

A.9 homography$_tf.m$

```
function XY_ = homography_tf(XY,H)
    Xi = cat(1, XY', ones(1, size(XY, 1)));
    Xi_ = H*Xi;
    XY_(:,1) = (Xi_(1,:)./Xi_(3,:))';
    XY_(:,2) = (Xi_(2,:)./Xi_(3,:))';
end
```

Listing 9: homography$_tf.m$

A.10 stitchH.m

```
function result = stitchH(img_src, H, img_des)
    [~, xdata, ydata] = imtransform(img_src, maketform('
     projective',H'));

    xdata_out=[min(1,xdata(1)) max(size(img_des,2), xdata
     (2))];
    ydata_out=[min(1,ydata(1)) max(size(img_des,1), ydata
     (2))];

    result1 = imtransform(img_src, maketform('projective'
     ,H') ,...
        'XData',xdata_out,'YData',ydata_out);
    result2 = imtransform(img_des, maketform('affine',eye
     (3)) ,...
        'XData',xdata_out,'YData',ydata_out);
    result = result1 + result2;
    overlap = (result1 > 0.0) & (result2 > 0.0);
    result_avg = (result2);

    result(overlap) = result_avg(overlap);
end
```

Listing 10: stitchH.m