Panoramic Image Stitching with SIFT

EE417 - Computer Vision Term Project
Supervisor: Prof. Mustafa Unel

Omer Burak Aladag and Berkant Deniz Aktas
{aladagomer | berkantdeniz} @sabanciuniv.edu

Faculty of Engineering and Natural Sciences
Sabancı University

January 11, 2019

Abstract

The problem considered in this project report is the fully constructed panoramic images from ordered consecutive images of the same scene. The core of this problem is feature extraction and matching between images. Then, RANSAC model is used to estimate mapping of pixels between two images. Finally, blending is done to construct panoramic images. Mathematical fundamentalsbehind the ideas of the principles used in this paper are reviewed. Our method is sensitive to the order of the images and will not work with unordered images. After a short overview of the project, we go over the concepts we have implemented or used in this perspective and then we show our results and present our conclusion.

Content

1 Introduction

An image is worth a thousand words. And yet, a single image may not be sufficient to fully capture the scenery. In such cases, panoramic images can be constructed from a few samples of the view. Panorama is a popular way to create mosaic images to simulate visually rich telepresence or virtual reality experience. In computer vision, image mosaics are part of a larger recent trend, namely the study of visual scene representations[1]. Various ways have been developed throughout the years to construct panoramic images of real-world sceneries. One of the oldest ways is to record images onto a long film strip using a panoramic camera [7]. Another method is consisting of regular images which cover the whole view scenery. Then, these images should be aligned and combined with the stitching algorithm previously proposed by various works [7]. For decent stitching, controlled camera trajectory is required. In this paper, we also show how uncontrolled trajectory affects output panoramic images.

In our work, we represent panoramic images by using the SIFT algorithm to extract and match features between ordered consecutive n images. Each transformation, homography, corresponds to one pixel from the first image to another pixel in the second image. After matchings are acquired, we blend images accordingly to get our output image by stitching. During the stitching process, our approach is linear. Thus we do not have cylindrical or spherical coordinates in our implementations. But it also creates a drawback when we have cylindrical image sequences in our data set.

Algorithm : Panoramic Image Stitching with SIFT

Input: n ordered images

I. Extract SIFT features from all images

II. For each image

   a. Find corresponding matching points between I(i) and I(i+1)

   b. Use RANSAC to solve for the homography between images

III. Warp images on the panoramic frame with their estimated homography

IV. Blend the panoramic image with alpha intensities

Output: Panoramic image mapped to the reference view

2 Feature Extracting and Feature Matching

There are various algorithms for extracting features from images. Extracting features are useful for different areas of computer vision applications such as object tracking, stereo matching, automated surveillance and more. A straightforward method for extracting features is detecting corner features of the image. Harris [4] proposed a novel algorithm for extracting the corner features from the image. However, the detected Harris Corners are not scale invariant. In other words, the detected features are strongly depended on the position of the camera.

In order to achieve a robust solution for feature matching, Lowe [2] proposed Scale Invariant Features (SIFT). Every SIFT point has its unique key values and descriptors. In order to detect key points, SIFT algorithm creates the scale-space using Gaussian Filter and Difference of Gaussians (DoG). Then algorithm detects extrema points and discards selected points which are lower than a threshold value.

After detection step is done by detecting the key points, algorithm computes their corresponding descriptors. Computing these descriptors is very useful for feature matching. A SIFT descriptor is a 3-D spatial histogram of the image gradients which characterizes the apparance of a key point [8]. The gradient value computed for each pixel is a three-dimensional feature vector constructed with the pixel's location and corresponding gradients orientation. Gaussian weighting function is used for giving fewer weights to the pixels which are away from key points.

The matching procedure is formed by comparing the square of the euclidean distances between the patch of pixels. The point pairs which have a euclidean distance smaller than a certain threshold will be selected and others will be discarded.

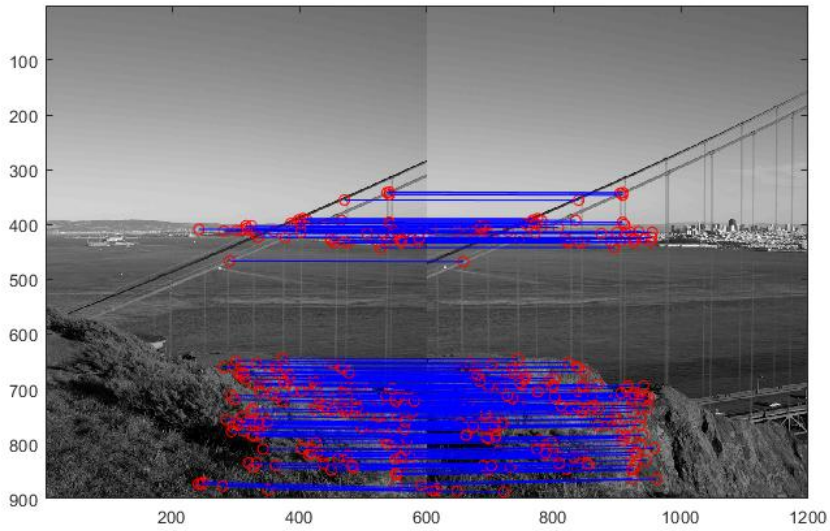

Figure 1. Matched features

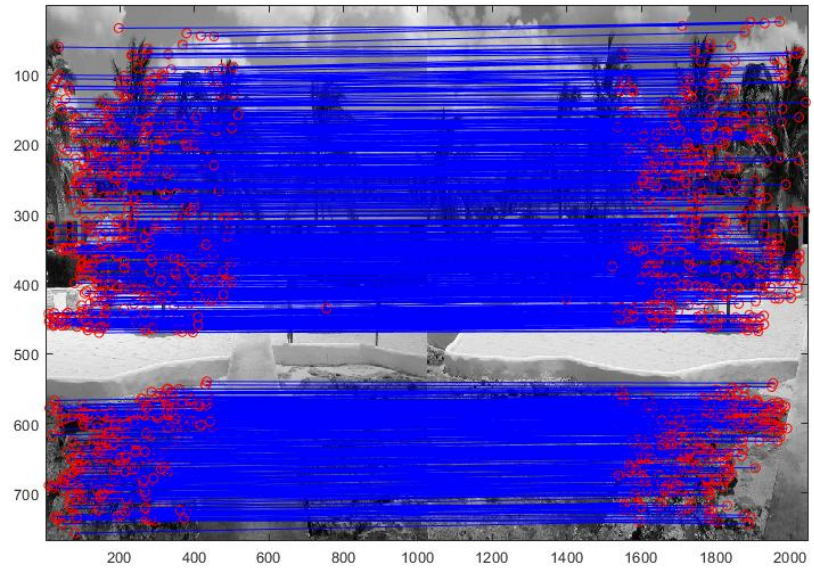

Figure 2. Match features with a lower threshold

3 Homography Estimation

After matching the key points, the transformation matrix of the second key points with respect to first keypoints should be computed. This computation can be done by homography estimation.

Equation 1

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$$

As we need a homography, four points can be used for both images. With RANSAC four random points can be extracted and their matches can be inserted in the below equations number 2, 3 and 4.

Equation 2

$$x = \frac{x_i}{z_i} = \frac{h_{11} * X_i + h_{12} * Y_i + h_{13}}{h_{31} * X_i + h_{32} * Y_i + 1}$$

Equation 3

$$\Leftrightarrow \begin{cases} h_{31} * x_i * X_i + h_{32} * x_i * Y_i + x_i - h_{11} * X_i - \\ \quad h_{12} * Y_i - h_{13} = 0 \\ h_{31} * y_i * X_i + h_{32} * y_i * Y_i + y_i - h_{21} * X_i - \\ \quad h_{22} * Y_i - h_{23} = 0 \end{cases}$$

Equation 4

$$\begin{pmatrix} -X_i & -Y_i & -1 & 0 & 0 & \cdots & x_i*X_i & x_i*Y_i & x_i \\ 0 & 0 & 0 & -X_i & -Y_i & -1 & y_i*Xi & y_i*Y_i & y_i \end{pmatrix} * \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ 1 \end{bmatrix} = 0$$

Least Square solutions of the last equation give the unknowns of the homography matrix. After successfully computing transformations for key points, the transformed point should be compared with previous matches. This will measure the distance between points. This can be done with setting a maximum deviation with labeling transformed key points with a distance under this threshold as inliers. Homography is computed again and again as long as the number of inliers are sufficient. If number of inliers are not sufficient the algorithm will stop after the fixed number of iterations.

4 Warping and Blending

After computing homography between images, warping can be done in sequential order. The first step constructs panoramic image frame which has a maximum size. After constructing the image frame, images can be warped into a frame one by one with corresponding homographies.

When the warping is done, a simple approach to blending is taking the weighted sum of the image intensities along each intersection line [3]. However, this can cause unclear images due to noise. Because of this reason, image color channels are changed to grayscale. We calculated alpha values for each pixel in the range of epsilon and one. For each new pixel, we calculated the weighted sum of pixel intensity values with corresponding alpha values.

5 Results

In this section, we demonstrate four different panorama image results from the dataset of Adobe Panorama Images [9]. Each experiment is done to show the effects of source images on the panorama stitching. Each stage of our algorithm has been affected by these image attributes. A detailed discussion of these results will be done in conclusion part. The one thing should be pointed out here, is that all our source images are in grayscale.

Figure 3. Stitched Bridge Image

Figure 4. Stitched Shanghai Bay Image

Figure 5. Stitched Mountain View Image

Figure 6. Stitched Office Image

6 Conclusion

In this paper, we tried to propose a very simple algorithm to construct Panoramic Images using SIFT. One thing we couldn't finish is to construct Panoramic Images with colors. In our blending script, the weighted sum of old pixel intensities to construct new pixel intensities is based on alpha values, opacity. Thus, we were not able to separate three channels of colors to anticipate the effects of R, G, B values on new pixel individually. Instead, we used grayscale input sets. However, since we were able to construct panoramic images that show our algorithm is correct but needs to be improved in terms of blending for colored images.

For decent stitching of images, small step sizes required to create larger overlapping areas. This is shown well in mountain scene (Figure 4) and bridge (Figure 3) panorama images. Other thing for taking pictures in order to increase quality of results is to prevent motion parallax of camera. Since we are not mapping our images onto a spherical coordinate system, it created wave shaped output

panoramas like in Shanghai example. The system is robust to camera zoom, orientation of the input images, and changes in illumination due to flash and exposure/aperture settings. However, our implementation fails with unordered image sets.

Final note about our algorithm is, if the two consectuive images has no enough corresponding features to estimate homography matrix RANSAC estimation fails and our program stops. But one improvement can be done on this to discard the less similar image until an image found which has similar features in it.

7 Appendices

7.1 Main Script

```
%% Initialize images

clear all; close all; clc;
outputDirect =
'D:\Burak\Senior_402\EE417\Project\Project_v1';
imageDir = 'D:\Burak\Senior_402\EE417\Project\Project_v1\4';
imageSet = imageDatastore(imageDir);
%% Extract feature points with SIFT
% Compute feature points
for i = 1 : length(imageSet.Files)
    inputImage = single(readimage(imageSet,i));
    [keyPoints{i}, descriptors{i}] = vl_sift(inputImage);
end
%% Compute homographies
for i=1:length(imageSet.Files)-1
    % Assign relevant data structures to variables for
convenience
    descriptors1 = descriptors{i};
    descriptors2 = descriptors{i + 1};
    keySet1 = keyPoints{i};
    keySet2 = keyPoints{i + 1};
    % Find matching feature points between current two
images.
    [matches, scores] = vl_ubcmatch(descriptors1,
descriptors2) ;
    img1_FeaturePTS = keySet1([2 1], matches(1, :))';
    img2_FeaturePTS = keySet2([2 1], matches(2, :))';
    homoList{i} = homoRANSAC(img1_FeaturePTS,
img2_FeaturePTS);
end
%% Warp images
homoMatrix(1) = {eye(3)};
for i = 2: size(homoList, 2) + 1
```

```
    homoMatrix{i} = homoMatrix{i-1} * homoList{i - 1};
end
homoMap = homoMatrix;
% Compute the size of the output panorama image
minRow = 1;
minCol = 1;
maxRow = 0;
maxCol = 0;
% for each input image
for i=1:length(homoMap)
    currentImage = readimage(imageSet, i);
    [rows,cols,~] = size(currentImage);
    pointMatrix = cat(3, [1,1,1]', [1,cols,1]', [rows, 1,1]',
[rows,cols,1]');
    % Map each of the 4 corner's coordinates into the
coordinate system of
    % the reference image
    for j=1:4
        result = homoMap{i}*pointMatrix(:,:,j);
        minRow = floor(min(minRow, result(1)));
        minCol = floor(min(minCol, result(2)));
        maxRow = ceil(max(maxRow, result(1)));
        maxCol = ceil(max(maxCol, result(2)));
    end
end
% Calculate output image size
panoramaHeight = maxRow - minRow + 1;
panoramaWidth = maxCol - minCol + 1;
% Calculate offset of the upper-left corner of the reference
image relative
% to the upper-left corner of the output image
rowOffset = 1 - minRow;
colOffset = 1 - minCol;
% Perform inverse mapping for each input image
for i=1:length(homoMap)
    % Create a list of all pixels' coordinates in output
image
    [xCord,yCord] = meshgrid(1:panoramaWidth,
1:panoramaHeight);
    % Create list of all row coordinates and column
coordinates in separate
    % vectors, x and y, including offset
    xCord = reshape(xCord,1,[]) - colOffset;
    yCord = reshape(yCord,1,[]) - rowOffset;
    % Create homogeneous coordinates for each pixel in output
image
    panoramaPTS(1,:) = yCord;
    panoramaPTS(2,:) = xCord;
    panoramaPTS(3,:) = ones(1,size(panoramaPTS,2));
    % Perform inverse warp to compute coordinates in current
input image
    imageCord = homoMap{i}\panoramaPTS;
    rowCord = reshape(imageCord(1,:),panoramaHeight,
panoramaWidth);
    colCords = reshape(imageCord(2,:),panoramaHeight,
panoramaWidth);
```

```
    currentImage = im2double(readimage(imageSet, i));

    % Bilinear interpolate
    currentWarpedImage = zeros(panoramaHeight,
panoramaWidth);
    currentWarpedImage(:, :) =
interp2(currentImage(:,:),colCords, rowCord, 'linear', 0);
    warpedImages{i} = currentWarpedImage;
end
%% Blend images
panoramaImage = zeros(panoramaHeight, panoramaWidth, 3);
panoramaImage = warpedImages{1};
for i = 2 : length(warpedImages)
    panoramaImage = blendg(panoramaImage, warpedImages{i});
end
imshow(panoramaImage)
imwrite(panoramaImage, fullfile(outputDirect,
'EE417_Project4.jpg'));
```

7.2 homoRANSAC Function

```
function H = homoRANSAC(p1, p2)
assert(all(size(p1) == size(p2)));  % input matrices are of
equal size
assert(size(p1, 2) == 2);  % input matrices each have two
columns
assert(size(p1, 1) >= 4);  % Need at least 4  points
numIter = 100;
maxDist = 3;
maxInliers = 0;
bestH = zeros(3,3);
for i = 1:numIter %loops numIter = 100 times
    inds = randperm(size(p1, 1), 4); %inds is a vector of 4
random unique integers in [1, n]
    H1 = homographyCalc(p1(inds,:),p2(inds,:)); %calculate
homography with 4 inds
    inliers = 0;
    for j = 1:size(p1,1)
        A = H1 * [p2(j,:),1]'; %A is 3xn matrix and H is a
3x3 matrix
        dist = sqrt(sum((A-[p1(j,:),1]').^2)); %A is nx3
matrices and dist is a 1xn matrix
        if(dist < maxDist)
            inliers = inliers + 1;
        end
    end
    if(inliers > maxInliers)
        bestH = H1;
        maxInliers = inliers;
    end
end
inlierpts = zeros(maxInliers,1);
k = 1;
```

```
for l = 1:size(p1,1)
    C = bestH * [p2(l,:),1]';
    dist = sqrt(sum((C-[p1(l,:),1]').^2));
    if(dist < maxDist)
        inlierpts(k) = l;
        k = k + 1;
    end
end
H = homographyCalc(p1(inlierpts,:),p2(inlierpts,:));
end
function H = homographyCalc(p1, p2)
assert(all(size(p1) == size(p2)));
assert(size(p1, 2) == 2);
n = size(p1, 1);
if n < 4                                    %% To estimate
Homography matrix we need at least 4 points
    error('Need at least 4 matching points');
end
H = zeros(3, 3);  % Homography matrix to be returned
M = zeros(n*3,9);
Lambda = zeros(n*3,1);
for i=1:n
    M(3*(i-1)+1,1:3) = [p2(i,:),1];
    M(3*(i-1)+2,4:6) = [p2(i,:),1];
    M(3*(i-1)+3,7:9) = [p2(i,:),1];
    Lambda(3*(i-1)+1:3*(i-1)+3) = [p1(i,:),1];
end
x = (M\Lambda)';
H = [x(1:3); x(4:6); x(7:9)];
end
```

7.3 Blending Function

```
function [ imgBlended ] = blendg( img1, img2 )
assert(all(size(img1) == size(img2)));
imgBlended = zeros(size(img1), 'like', img1);
alpha1 = grey2alpha(img1);
alpha2 = grey2alpha(img2);
outChannel = (alpha1.*img1(:,:) + alpha2.*img2(:,:)) ...
    ./(alpha1 + alpha2);

imgBlended(:,:) = outChannel;
end
function imgAlpha = grey2alpha(img, epsilon)
switch nargin
    case 1
        epsilon = 0.001;
end
binaryImg = im2bw(img, epsilon);
complementImg = imcomplement(binaryImg);
imgDistance = bwdist(complementImg,'euclidean');
imgAlpha = rescale(imgDistance, epsilon, 1);
end
```

8 References

1. Anandan, P. et al., editors. (1995), IEEE Workshop on Representations of Visual Scenes, IEEE Computer Society Press, Cambridge, Massachusetts.

2. Brown, M. and Lowe, D.G (2003), "Recognising panoramas," Proceedings Ninth IEEE International Conference on Computer Vision, Nice, France, pp. 1218-1225vol.2.doi:10.1109/ICCV.2003.1238630

3. Brown, M. and Lowe, D.G. (2006), "Automatic Panoramic Image Stitching using Invariant Features." International Journal of Computer Vision 74, no. 159-73. doi:10.1007/s11263-006-0002-3.

4. Harris, C. & Stephens, M. (1988), "A combined corner and edge detector". 50.

5. Shum, H.-Y., and Szeliski, R. (2001), "Construction of Panoramic Image Mosaics with Global and Local Alignment." Panoramic Vision, 227-268. doi:10.1007/978-1-4757-3482-9_13.

6. SIFT Keypoint Detector Demo: https://www.cs.ubc.ca/~lowe/keypoints/

7. Shum, H.-Y., and Szeliski, R. (2000), "Creating Full View Panoramic Image Mosaics and Environment Maps". Computer Graphics (Proceedings of Siggraph '97). 10.1145/258734.258861.

8. Vedaldi, A. (2007), "An open implementation of the SIFT detector and descriptor." UCLA.

9. Dataset: https://sourceforge.net/adobe/adobedatasets/panoramas/home/Home/