Assignment 3
CAP 6419 - 3D Computer Vision

Abhianshu Singla

Image Mosaic Creation using Feature-Based Registration

The main steps which I used to create an image Mosaic are as:

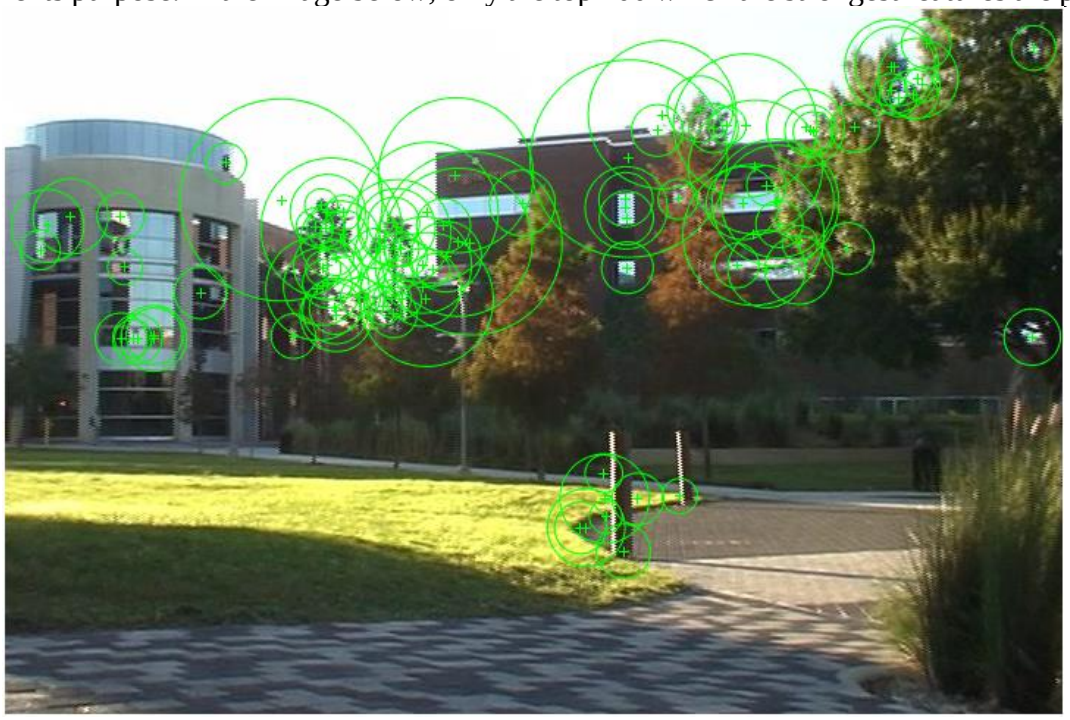
1. Extract the image features such as BRISK, SURF, SIFT, etc.
2. Match these features among a pair of images to track point correspondences.
3. Eliminate outliers which are not fit to these pair of images using Peter Koveski's RANSAC method.
4. Compute the homography using RansacFitHomography.
5. Warped (or Stitched) the image using the computed homography to create a panorama.

Details:

1. Feature Extraction

- **SURF: Speeded-Up Robust Features**

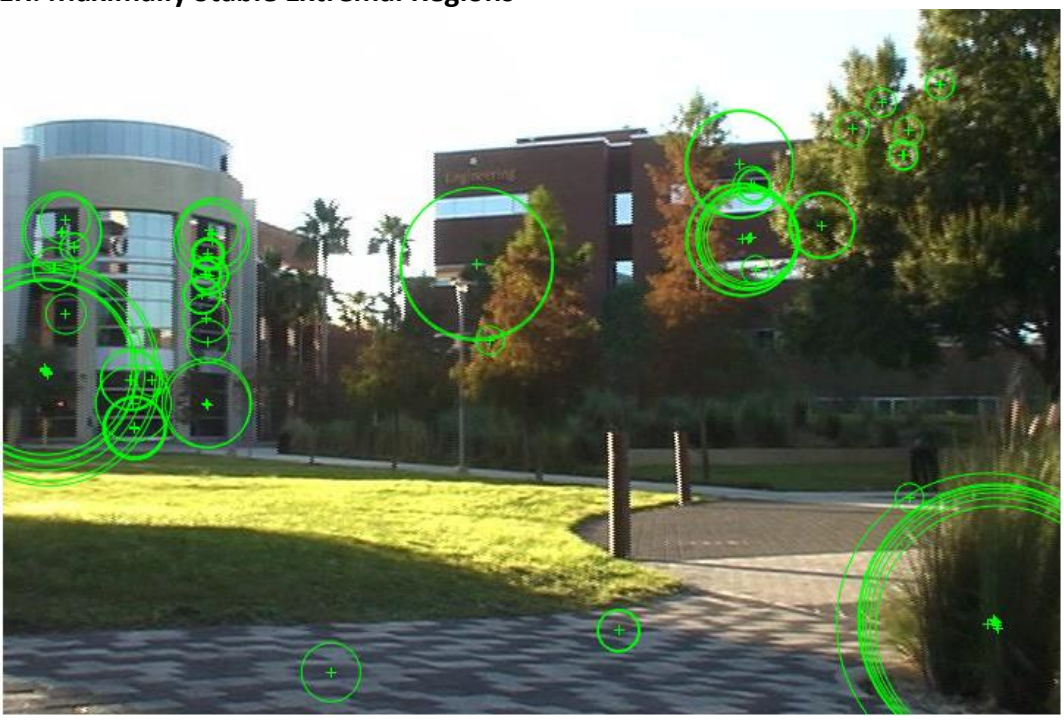
It generates scale and rotation invariant keypoints and their descriptors. It is highly time efficient as compared to other feature extraction techniques such as SIFT (Scale Invariant Feature Transform). However, in situations of extreme intensity difference, it is less effective than SIFT. But I don't have the SIFT library working with my MATLAB. So, I am using SURF features and I am not taking the images with large intensity difference. So, SURF features solve this assignments purpose. In the image below, only the top 100 which are strongest features are plotted.



```
points = detectSURFFeatures(grayImg);
[features, points] = extractFeatures(grayImg, points);
```

- **BRISK**: **Binary Robust Invariant Scalable Key-Points**
  Although SIFT and SURF features are invariant to scale, rotation and viewpoint, their descriptors take a lot of time to compute. So, SURF detector can be used with BRISK descriptor. In the image below, the top 100 keypoints are plotted on the same image as above to see the comparison between these features.

- **MSER: Maximally Stable Extremal Regions**



2. Feature Mapping

After the keypoint extraction, these features (keypoints) are matched to each other at their corresponding locations in different image pairs.


```
indexPairs = matchFeatures(features1, features2);
```

As there are some random point correspondences as shown in the above image using red and black markers. So, the additional parameters are name-value pair in matchFeatures, i.e., ‘Unique’. If the value of ‘unique’ is true, then, it performs the forward-backward match to select unique match. It first matches feature1 to feature2 and then matches features2 to features1 and selects the best match.


```
indexPairs = matchFeatures(features1, features2, 'Unique', true);
```

3. RANSAC: Random Sample Consensus

After obtaining the point correspondences, we need only four-point correspondences which will result in computing a homography with minimum error.

RANSAC problem basically consists of two problems which are fitting a model to the data and classification of data into inliers and outliers. RANSAC can easily scale to a large proportion of outliers. The basic algorithm of RANSAC from the textbook is shown below:

Objective

Robust fit of a model to a data set S which contains outliers.

Algorithm

(i) Randomly select a sample of s data points from S and instantiate the model from this subset.
(ii) Determine the set of data points S_i which are within a distance threshold t of the model. The set S_i is the consensus set of the sample and defines the inliers of S.
(iii) If the size of S_i (the number of inliers) is greater than some threshold T, re-estimate the model using all the points in S_i and terminate.
(iv) If the size of S_i is less than T, select a new subset and repeat the above.
(v) After N trials the largest consensus set S_i is selected, and the model is re-estimated using all the points in the subset S_i.

Peter Koveski's RANSAC function:

```
[M, inliers] = ransac(x, fittingfn, distfn, degenfn s, t, feedback,
maxDataTrials, maxTrials)
```

- x - It is the data on which we want to fit a model. Here, we have point correspondences (x1 and x2) from the two images. Hence, [x1.location; x2.location] is our x.
- s - It is the minimum number of samples required from x to fit the model. s= 4 for 4 point correspondences.
- t - It is the threshold distance between a data point and a model to classify it as an inlier or outlier. I used t as 0.005
- fittingfn - It is the function to fit the model. It can return multiple models or an empty array if no model is possible on the given data set.
- distfn - It is the distance function to compute the distance between the model and the data point.
- degenfn – a function to determine if the set of data points will give degenerate model or not. It checks the collinearity of all the three point sets of the two images possible with s.

fittingfn: homography2d is used as a fitting function. In this function, first the data points are normalized to get the origin at the centroid. This 2d homography is computed using:

$$\begin{bmatrix} 0^T & -wX^T & yX^T \\ wX^T & 0^T & -xX^T \\ -yX^T & -xX^T & 0^T \end{bmatrix} \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix} = 0$$

$$where\ the\ j^{th}\ row\ of\ H\ is\ denoted\ by\ h^{j\,T}\ and\ X\ is\ the\ point$$

This is obtained by taking the cross-product of X' and HX. X ←→ X' are the point correspondences. The above equation is of the form Ah = 0 which is linear in h and it is solved using Singular Value Decomposition (svd).

$$[U \quad D \quad V] = svd(A, 0)$$

The last column of V gives the result and it is then reshaped to a 3*3 matrix to fill the nine values of the Homography matrix H. In the last step, it is denormalized.

Gist of this function:

```
for n = 1:Npts
    X = x1(:,n)';
    x = x2(1,n); y = x2(2,n); w = x2(3,n);
    A(3*n-2,:) = [  0  -w*X  y*X];
    A(3*n-1,:) = [ w*X   0  -x*X];
    A(3*n  ,:) = [-y*X  x*X   0 ];
end
[U,D,V] = svd(A,0);
H = reshape(V(:,9),3,3)';
```

distfn: This function evaluates the symmetric transfer error as in the error in both the images using geometric distance. In this, both the forward and the backward transformations are considered and the geometric errors for both the transformations are summed.

```
% Calculate, in both directions, the transfered points
Hx1    = H*x1;
invHx2 = H\x2;
% Normalise so that the homogeneous scale parameter for all coordinates is 1.
x1     = hnormalise(x1); x2     = hnormalise(x2);
Hx1    = hnormalise(Hx1); invHx2 = hnormalise(invHx2);

d2 = sum((x1-invHx2).^2)  + sum((x2-Hx1).^2);
inliers = find(abs(d2) < t);
```

4. **Compute the Homography:**

After the 4 point correspondences are obtained which will give the best result, the homography is computed on these 4 points using the same homography2d function which we used as a fitting function.

```
[H, inliers] = ransacfithomography((matchedPoints.Location)',
(matchedPointsPrev.Location)', 0.005);
```

5. Image Warping:

In this step, images are warped together to create a panorama. For this, backward mapping is used. In backward mapping, initially the source image is separated into three different channels, i.e., red, blue and green. Then for each pixel in the destination image, backward find the corresponding points in the source image. The mask is obtained using the corner coordinates of the destination image which are obtained by applying the homography on the corner coordinates

of the source image. The backward mapping is obtained using the inverse of the tforms (2D homography) struct with the Image. The bilinear interpolation is obtained by setting the name-value pair ‘interp’ as ‘bilinear’.

```
Tinv = invert(tforms(centerImageIdx));
for i = 1:numel(tforms)
    tforms(i).T = tforms(i).T * Tinv.T;
end
……….
panoramaView = imref2d([height width], xLimits, yLimits);
warpedImage = imwarp(I, tforms(i), 'OutputView', panoramaView,...
'interp', 'bilinear');
```

Results:

Set 1

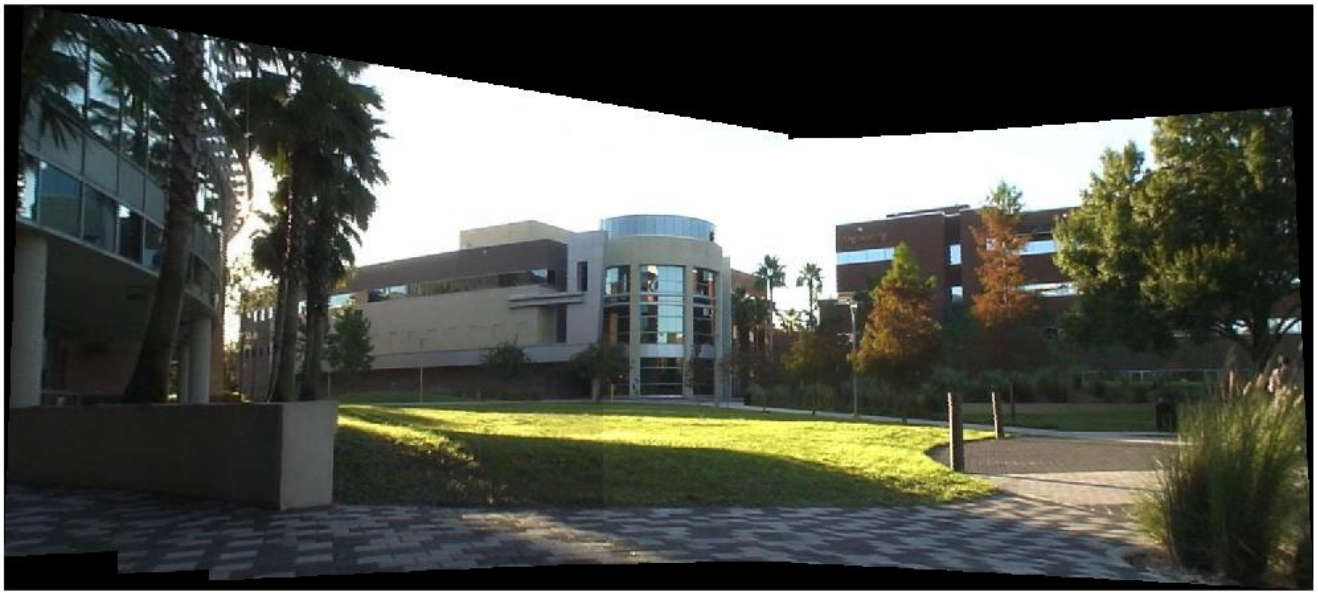
Panorama

Set 2






Panorama



Set 3









Panorama

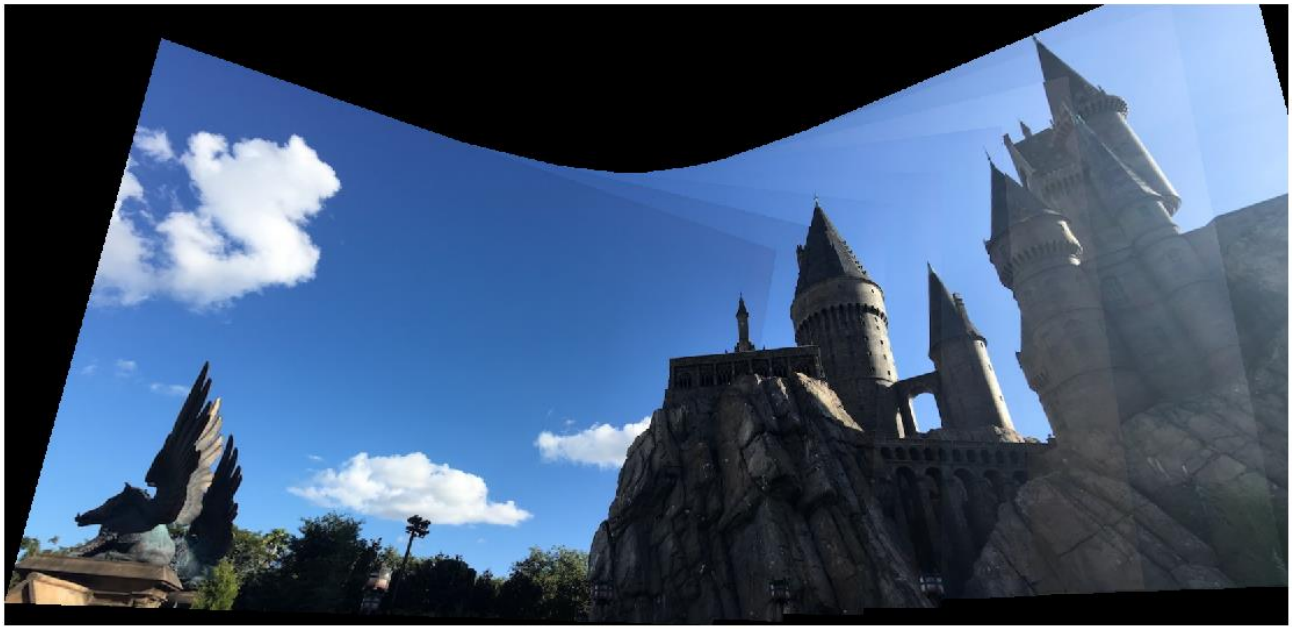
Set 4







Panorama



Set 5



Panorama

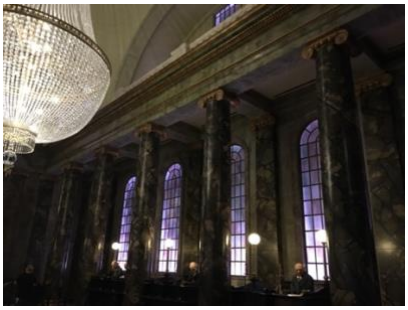
Set 6

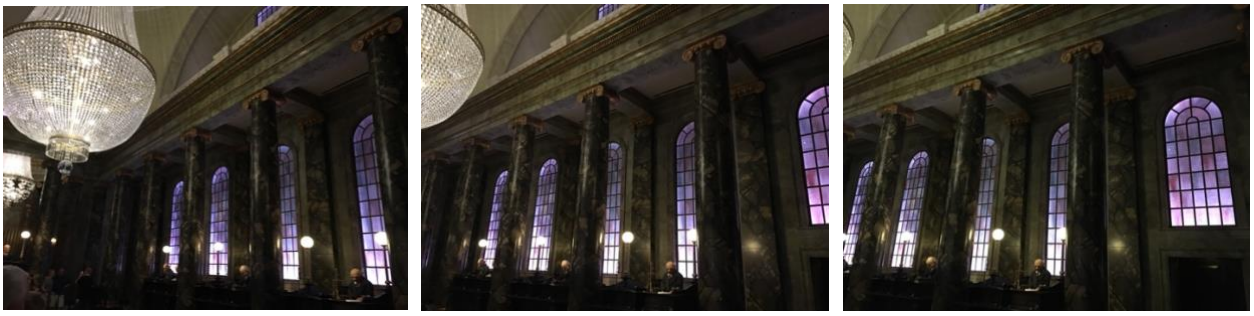
Panorama

Set 7

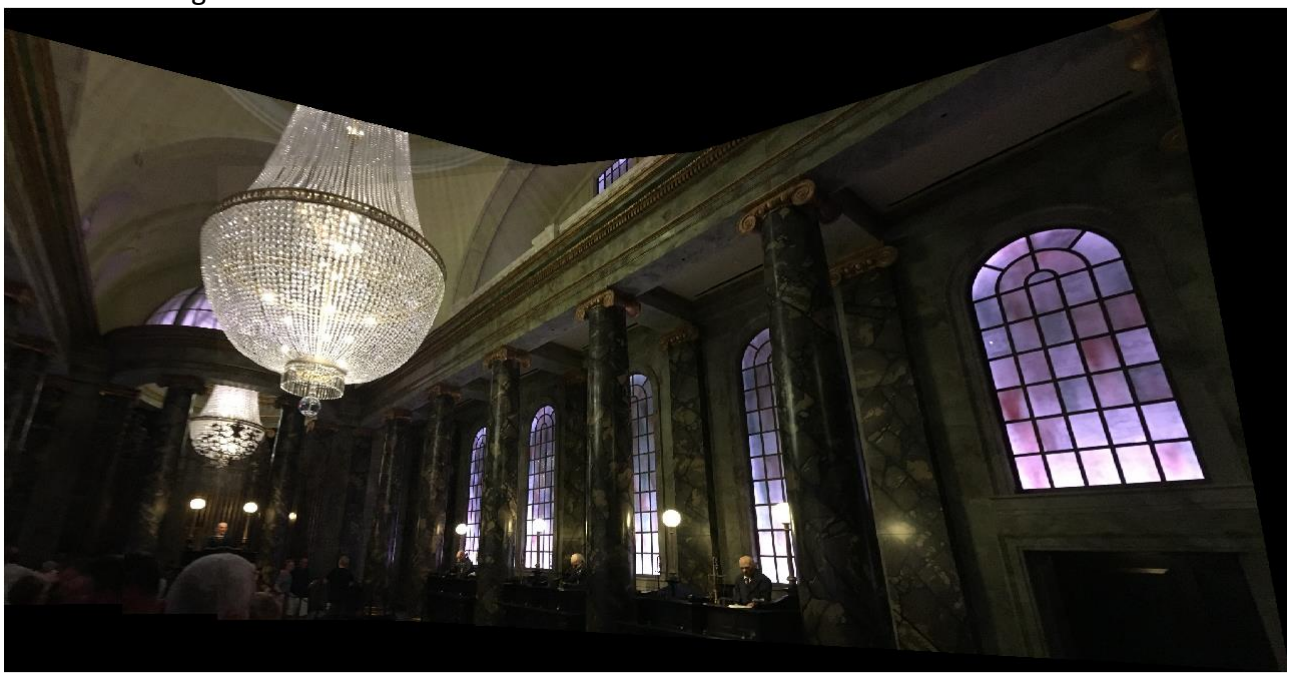

Panorama

Set 8





Panorama Image



References:

1. https://www.mathworks.com/help/vision/examples/feature-based-panoramic-image-stitching.html
2. Multiple View Geometry in Computer Vision – Richard Hartley and Andrew Zisserman