# Project 2 - Image Stitching Report

Advait Deshmukh - 50292518
Priyanka Pai - 50295903

July 17, 2019

## 1 Introduction

In this project we perform Image Stitching on multiple images to get a single panoramic image. The images are taken in such a way that overlapping fields of view exist between the images on which image stitching can be performed.Image stitching or photo stitching is the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image. Commonly performed through the use of computer software, most approaches to image stitching require nearly exact overlaps between images and identical exposures to produce seamless results, although some stitching algorithms actually benefit from differently exposed images by doing high-dynamic-range-imaging in regions of overlap.In this project we have implemented **all the procedures from scratch** whose equivalent methods in OpenCV are:

- bfMatcher- cv::bfMatch
- FlannMatch- cv::FlannBasedMatcher
- Stitcher- cv::Stitcher
- findHomography()
- RANSac- cv2.RANSAC

## 2 General approach to get a panoramic image

- The first step in panorama stitching is to extract features from both input images. Features are localised at a 'keypoint' and described by a 'descriptor'. Commonly, we use SIFT to detect features and extract feature descriptors. Ensure that descriptors is returned as an array of size n $\chi$ 64

- The find correspondences method finds features with similar descriptors using match flann/bfMatcher, flann uses a kd-tree for efficient correspondence matching. Flann Matcher returns an array with two columns (one for each image), in which each row lists the indices of corresponding features in the two images. This is split into two arrays, points1 and points2, which give the coordinates of corresponding keypoints in corresponding rows.
- Next step is identifying the optimal image size and offset of the combined panorama. For this, we need to consider the positions of each images' corners after the images are aligned. This can be calculated using the homogeneous 2D transform ($3\chi3$-matrix) stored in 'homography'. From this, you can calculate the total size of the panorama as well as the offset of the first image relative to the top-left corner of the stitched panorama.
- Finally, we combine the two images into one panorama using the warpPerspective method.

# 3 Method

## 3.1 Finding Keypoints

We begin by extracting keypoints and descriptors in our input images. To extract keypoints, we have utilised the Scale Invariant Feature Transform (SIFT) [2] provided in OpenCV, which works as:

[1] Constructing a scale space - Create internal representations of the original image to ensure scale invariance. This is done by generating a "scale space".

[2] LoG Approximation - The Laplacian of Gaussian is great for finding interesting points (or key points) in an image, approimated using previous method.

[3] Finding keypoints - These are maxima and minima in the Difference of Gaussian.[1]

[4] Get rid of bad key points - Edges and low contrast regions are bad keypoints. A technique similar to the Harris Corner Detector is used here.

[5] Assigning an orientation to the keypoints - An orientation is calculated for each key point. Further calculations are done relative to this orientation. This effectively cancels out the effect of orientation, making it rotation invariant.

[6] Generate SIFT features - With scale and rotation invariance in place, it uses local image gradients at selected scale and rotation to uniquely describe features.

SIFT image feature coordinates are invariant to translation, rotation, illumination as well as scale.

### 3.2 Selecting Good Keypoints

We take the Euclidean distance between each of the descriptors to sift out the bad keypoints. To do this, we take least Euclidean distances in each iteration of computing these distances, and keep the keypoints who descriptors have least values. This ensures good matching between the images.

### 3.3 Output

### 3.4 Computing Homography Matrix

A Homography matrix is a relation matrix between two images, which in image stitching is useful for computation of camera motion in terms of image rotation and image translation. That is, it helps transform points in one image to the corresponding points in other image using homogeneous coordinates of matching points. The equivalent inbuilt method is - cv2.findHomography() This matrix is

calculated using the following steps:

### 3.4.1 Computing correspondence points

The four correspondence points are calculated as:

$$p_i = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x_i' & y_i x_i' & x_i' \\ 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y_i' & y_i y_i' & y_i' \end{bmatrix}$$

The Homography matrix is a 3x3 matrix but with 8 DoF (degrees of freedom) as it is estimated up to a scale. It is generally normalized with

$$h33 = 1 \ or$$

$$h211 + h212 + h213 + h221 + h222 + h223 + h231 + h232 + h233 = 1$$

### 3.4.2 Code Snippet-Pseudocode[1]

```
def normalize(points):
  """ Normalize a collection of points in
    homogeneous coordinates so that last row = 1. """
  for row in points:
    row /= points[-1]
  return points


def make_homog(points):
  """ Convert a set of points (dim*n array) to
    homogeneous coordinates. """
  return vstack((points,ones((1,points.shape[1]))))
```

### 3.4.3 Output Homography Matrix

[[-4.19124566e-03 3.33650006e-03 8.88243125e-01] [-7.42246704e-04 -4.17223344e-03 4.59267476e-01] [ 1.70333609e-07 4.07848375e-06 -7.13289923e-03]]
[[ 5.13695757e-04 -4.94248673e-04 6.87275352e-02] [-7.06171644e-04 6.45996557e-04 9.97629495e-01] [-1.66110761e-06 -5.53980737e-07 3.23916031e-03]]

### 3.4.4 Good Matches

## 3.5 Robust feature matching through RANSAC Algorithm

RANSAC, short for "RANdom SAmple Consensus," is an iterative method to fit models to data that can contain outliers.[4] Given a model, for example a homography between sets of points, the basic idea is that the data contains inliers, the data points that can be described by the model, and outliers, those that

do not fit the model. We begin by calculating the assumed list of inliers which is produced by each homography matrix, keeping only those whose distance is less than our selected threshold. This helps us in the selection of the best homography matrix which can be further used to stitch images.

1. Select four feature pairs (at random) Compute homography H (exact)
2. Compute inliers where ||pi ', H pi || $< \varepsilon$
   - Pick 2 points
   - Fit line
   - Count inliers
3. Keep largest set of inliers
4. Re-compute least-squares H estimate using all of the inliers

### 3.5.1 Pseudocode for RANSAC

```
class RansacModel(object):
  """ Class for testing homography fit with ransac.py from
    http://www.scipy.org/Cookbook/RANSAC"""

  def __init__(self,debug=False):
    self.debug = debug

  def fit(self, data):
    """ Fit homography to four selected correspondences. """

    # transpose to fit H_from_points()
      # from points
      # target points
       # fit homography and return

  def get_error( self, data, H):
    """ Apply homography to all correspondences,
      return error for each transformed point. """

    data = data.T

    # from points
    fp = data[:3]
    # target points
    tp = data[3:]

    # transform fp
    fp_transformed = dot(H,fp)

    # normalize hom. coordinates
```

```
  for i in range(3):
    fp_transformed[i] /= fp_transformed[2]

  # return error per point
  return sqrt( sum((tp-fp_transformed)**2,axis=0) )
```

### 3.5.2 Output

## 3.6 Stitching the Images Together

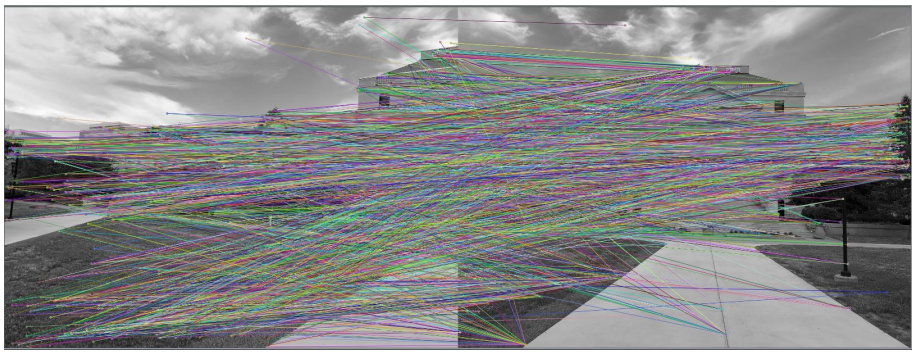
With the homographies between the images estimated (using RANSAC), we now need to warp all images to a common image plane. It makes most sense to use the plane of the center image (otherwise the distortions will be huge). One way to do this is to create a very large image, for example filled with zeros, parallel to the central image and to warp all the images to it. Since all our images are taken with a horizontal rotation of the camera, we can use a simpler procedure: we just pad the central image with zeros to the left or right to make room for the warped images.

## 3.7 Pseudocode for Image Stitching

```
def panorama(H,fromim,toim,padding=2400,delta=2400):
  """ Create horizontal panorama by blending two images
    using a homography H (preferably estimated using RANSAC).
    The result is an image with the same height as toim. 'padding'
    specifies number of fill pixels and 'delta' additional translation.
        """

  # check if images are grayscale or color

  # homography transformation for geometric_transform()

  if H[1,2]<0: # fromim is to the right
    print 'warp - right'
    # transform fromim
    if is_color:
      # pad the destination image with zeros to the right


    else:
      # pad the destination image with zeros to the right

  else:
    print 'warp - left'
    # add translation to compensate for padding to the left

    # transform fromim
```

```
  if is_color:
    # pad the destination image with zeros to the left

  else:
    # pad the destination image with zeros


# blend and return (put fromim above toim)
if is_color:
  # all non black pixels
return toim_t
```

### 3.7.1 Output



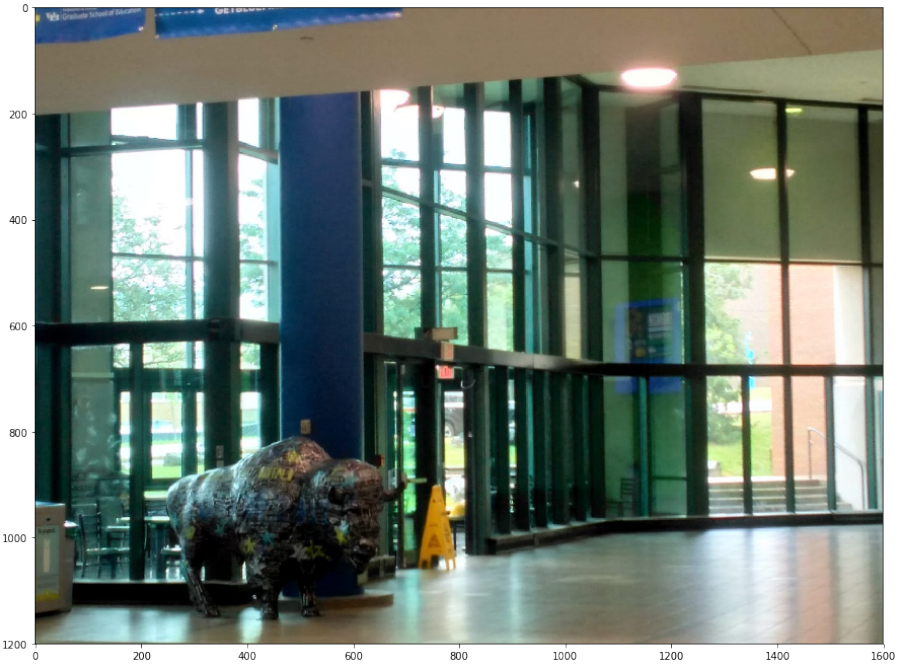
# 4 Contours

# 5 Output

## 5.1 Stitching Planar Images

IMAGE 1

IMAGE 2

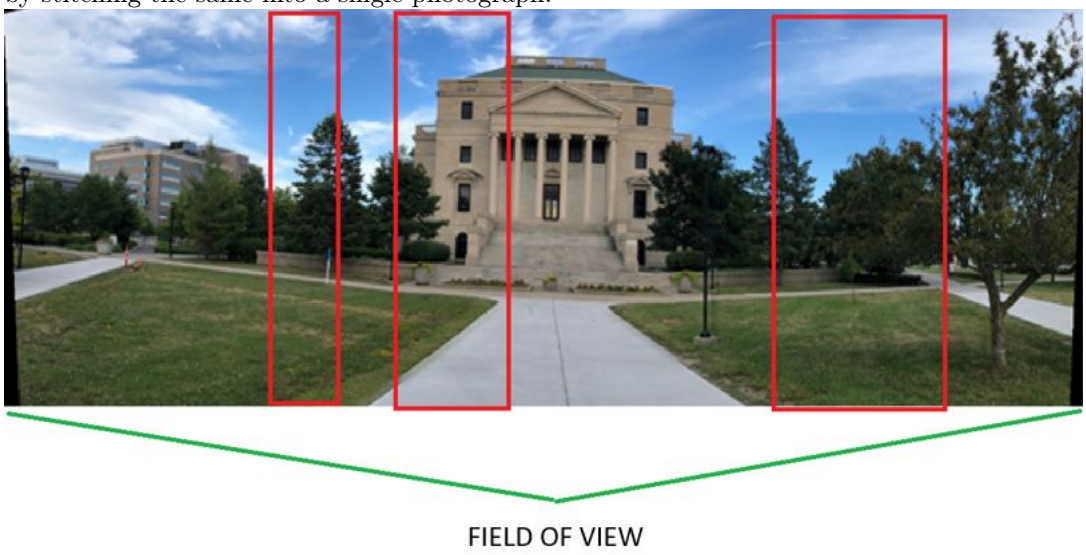


### 5.2 Stitching Images taken at different orientations

This is the Abbott Hall or commonly known as Health Sciences Library.

## 6 Reason For Approach

**Taking Photos** :Manual Mode with Manual Focus-This ensures all the photographs shot have an exact exposure value and focus throughout all the images. Shooting in Automatic, Program or Semi-Automatic modes (like Shutter priority or Aperture priority) will result in different exposure values for each photograph, which in turn may cause the final merged photograph to have varied exposure and color casts in different parts of the photograph. Since two adjacent photographs have at least 20% overlapping areas, we need to understand the common areas in two adjacent photographs and hence we can eliminate duplication of a scene by stitching the same into a single photograph.




**Keypoint and Descriptors** : Even though there are several ways to tackle the keypoint selection and finding the feature descriptor(like GLOH,SURF), we went for SIFT implementation as it is more robust,more accurate than any

other descriptors also, rotation and scale invariant. Some of the alternatives to SIFT were Multi-scale histogram of oriented gradients, but that approach would result in features that are non fully-scale oriented. Additionally, this approach is already a part of the way SIFT extracts features. Another alternative is ORB, which is more robust than the Multi-scale histogram approach and a faster approach than SIFT. However, **SIFT is traditionally a more robust approach and as it is not as sensitive to in-plane rotations as ORB.**

**Distance Calculation** - There are many approaches to computing the distance : Euclidean Distance, Manhattan Distance, Hamming Distance but we opted for Euclidean distance as the approximation is accurate in comparison with the other approaches. **The reason for selecting Euclidean Distance as our approach is that we are working on a continuous space where most of the dimensions are properly scaled and relevant.**

**RANSAC** :All the inliers will agree with each other on the translation vector, thus lie on the same fit line. All outliers are out of the boundary as it is robust to noise and outliers. One of the main advantages of using RANSAC is that it can do robust estimation of the parameters, which is its ability to estimate parameters even when significant amount of outliers are present. **This was the main reason for us choosing RANSAC.**

**Blending** - Approaches we tried - Alpha Blend, Adding two images with illumination Differences.

# 7 Libraries Used

- numpy - For Matrix operations.
- OpenCV 3.4.1.2 For basic image utilities like reading, writing and applying basic transformations to images.
- random - For selecting random numbers in RANSAC.
- sys - For taking arguments (eg. via command line).
- glob - For pathname pattern expression.
- os - For interacting with the Operating System.

# References

[1] Programming Computer Vision - Cambridge, MA: MIT Press

[2] OpenCV Documentation - https://opencv.org/

[3] Richard Szeliski - Computer Vision: Algorithms and Applications http://szeliski.org/Book/

[4] RANSAC https://en.wikipedia.org/wiki/Random_sample_consensus