CS 663 Project - Document Scanner

Meet Udeshi - 14D070007
Arka Sadhu - 140070011
Sravan Patchala - 14D070012

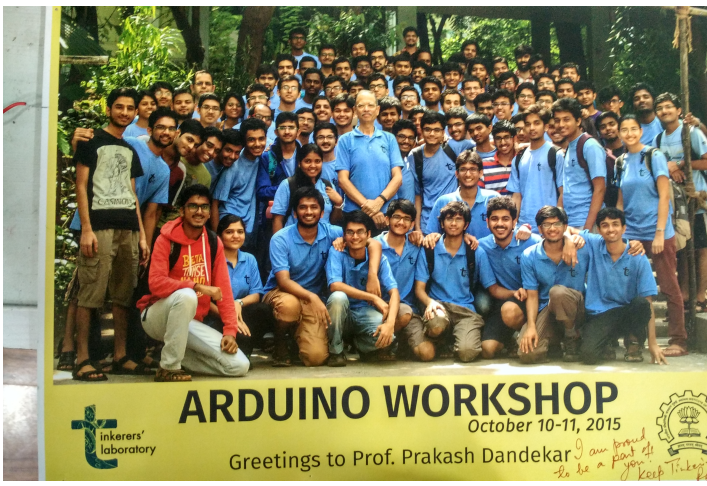
November 2016

Problem Statement

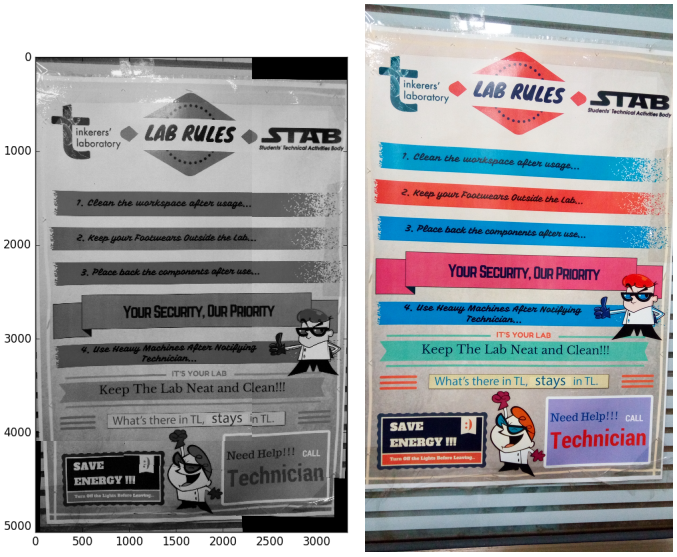
We capture parts of a document, with overlapping regions between the images and then reconstruct the whole document by stitching the images.

Experimental Results

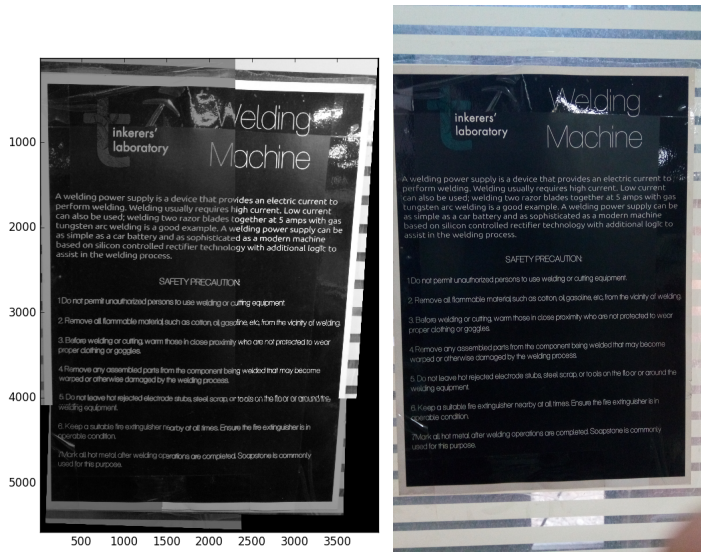
Image Poster



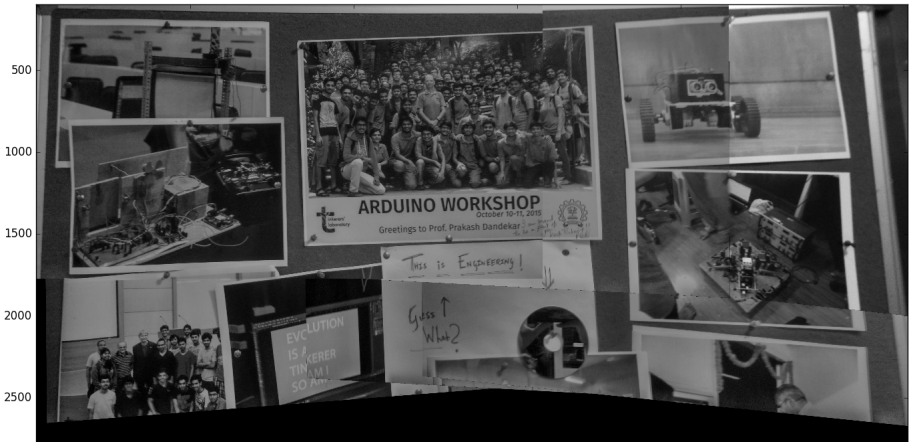
Poster with Image and Text



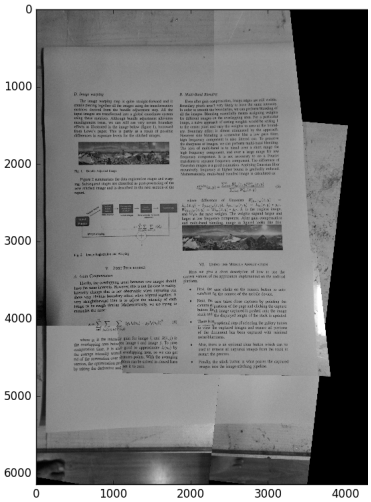
White Text on Dark Background

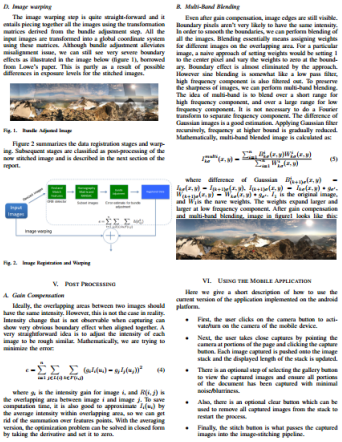


Multiple Posters with both text and images



Printed text document with images





D. Image warping

The image warping step is quite straight-forward and it entails piecing together all the images using the transformation matrices derived from the bundle adjustment step. All the input images are transformed into a global coordinate system using these matrices. Although bundle adjustment alleviates misalignment issue, we can still see very severe boundary effects as illustrated in the image below (figure 1), borrowed from Lowe's paper. This is partly as a result of possible differences in exposure levels for the stitched images.

Fig. 1. Bundle Adjusted Image

Figure 2 summarizes the data registration stages and warping. Subsequent stages are classified as post-processing of the now stitched image and is described in the next section of the report.

Fig. 2. Image Registration and Warping

V. Post Processing

A. Gain Compensation

Ideally, the overlapping areas between two images should have the same intensity. However, this is not the case in reality. Intensity change that is not observable when capturing can show very obvious boundary effect when aligned together. A very straightforward idea is to adjust the intensity of each image to be rough similar. Mathematically, we are trying to minimize the error:

$$e = \sum_{i=1}^{N} \sum_{j \in I(i)} \sum_{n \in F(i,j)} (g_i I_i(u_n) - g_j I_j(u_n))^2 \quad (4)$$

where g_i is the intensity gain for image i, and $I(i, j)$ is the overlapping area between image i and image j. To save computation time, it is also good to approximate $I_i(u_n)$ by the average intensity within overlapping area, so we can get rid of the summation over features points. With the averaging version, the optimization problem can be solved in closed form by taking the derivative and set it to zero.

B. Multi-Band Blending

Even after gain compensation, image edges are still visible. Boundary pixels aren't very likely to have the same intensity. In order to smooth the boundaries, we can perform blending of all the images. Blending essentially means assigning weights for different images on the overlapping area. For a particular image, a naive approach of setting weights would be setting 1 to the center pixel and vary the weights to zero at the boundary. Boundary effect is almost eliminated by the approach. However sine blending is somewhat like a low pass filter, high frequency component is also filtered out. To preserve the sharpness of images, we can perform multi-band blending. The idea of multi-band is to blend over a short range for high frequency components, and over a large range for low frequency component. It is not necessary to do a Fourier transform to separate frequency component. The difference of Gaussian images is a good estimation. Applying Gaussian filter recursively, frequency at higher bound is gradually reduced. Mathematically, multi-band blended image is calculated as:

$$I_{dst}^{multib}(x, y) = \frac{\sum_{i=1}^{n} I_{dst}^{i}(x, y) W_{dst}^{i}(x, y)}{\sum_{i=1}^{n} W_{dst}^{i}(x, y)} \quad (5)$$

where difference of Gaussian $I_{dst}^{i}(x, y) - I_{dst}^{i+1}(x, y) = I_{dst}^{i}(x, y) - I_{dst}^{i}(x, y) * g_{\sigma^i} = W_{dst}^{i}(x, y)$. I_i is the original image, and W_{dst}^{i} is the mere weights. The weights expand larger and larger at low frequency component. After gain compensation and multi-band blending, image in figure2 looks like this:

VI. Using the Mobile Application

Here we give a short description of how to use the current version of the application implemented on the android platform.

- First, the user clicks on the camera button to activate/turn on the camera of the mobile device.
- Next, the user takes close captures by pointing the camera at portions of the page and clicking the capture button. Each image captured is pushed onto the image stack and the displayed length of the stack is updated.
- There is an optional step of selecting the gallery button to view the captured images and ensure all portions of the document has been captured with minimal noise/blurriness.
- Also, there is an optional clear button which can be used to remove all captured images from the stack to restart the process.
- Finally, the stitch button is what passes the captured images into the image-stitching pipeline.

Procedure

Procedure described in the paper Badlani Akinola

- Feature Matching
- Homography Matrix computation using RANSAC
- Bundle Adjustment
- Gain Compensation
- Multi-Band Blending

What We Have Implemented

- We have successfully implemented Feature Matching. We used ORB feature detection function inbuilt in OpenCV Python.
- Next we implemented RANSAC on our own.
- We adjusted the parameters in probabilistic modelling to get the correct number inliers
- We experimented with different ways to merge the image. We tried simple merging, and then merging sequentially. The latter gave much better homographies.
- We tried doing Bundle Adjustment. There is no direct implementation in OpenCV Python, so we created our own Bundle Adjuster. But this doesn't give much better results compared to normal merging, so we are not using this in the final code. Also scipy cannot be ported to Android so we did not go much further with it.
- We implemented gain compensation.
- We also implemented Multi-Band Blending using laplacian pyramids

Interesting Observations

We did the following changes from the paper:

1. We are calculating homography after scaling down. But we can't directly apply to the original image. Applying this to the original image and then upsampling will cause a lot of errors. So instead we find the equivalent Matrix Transformation $S * H * S^{-1}$. This causes no error due to discretization.
2. We also tried out Image Merging by adding image one by one. We are doing this by maintaining a queue. So suppose Image2 matches with Image1, then we compare Image3 with the composite Image2 and Image1, this returns much better result than the one got by simple merging by projecting everything to Image 1.
3. We experimented with ORB features and found that the feature matching is much better for Binary Image rather than Grayscale. Hence we use Binary Image to calculate the Homography, then use the Scaling Transformation to get the Homography Matrix for the Original Image.
4. We used Probabilistic Modelling to know how many inliers should we focus on, which resulted in much better results.
5. We used scipy.optimize for Bundle Adjustment, since we needed solution for non-linear least square minimization. The suggested method in the paper is Levenberg-Marquardt algorithm but unfortunately there is no opencv python implementation for the same. But scipy cannot be migrated to Android, hence we are not using it.

6. Gain Compensation: We find average of both the images i.e. warped image and merged image in the common region. We scale the warped image such that the intensity in the common region is the same.

7. MultiBand Blending : Again there is no existing implementation of it in OpenCV python. Hence we implemented it using Laplacian Pyramids. We took K = 6 levels in our implementation. So basically we first find Gaussian Pyramids till K=6 levels, then we start from the last level use Pyramid Up function and then subtract it with the previous level to get a laplcian Pyramid of that level. We do it for all levels. Then we add the laplacian pyramids for all the images. This is stored in a different array Ls. This contains summed images of different levels. Then we add all the levels, which gives us the final multi-band blended image.