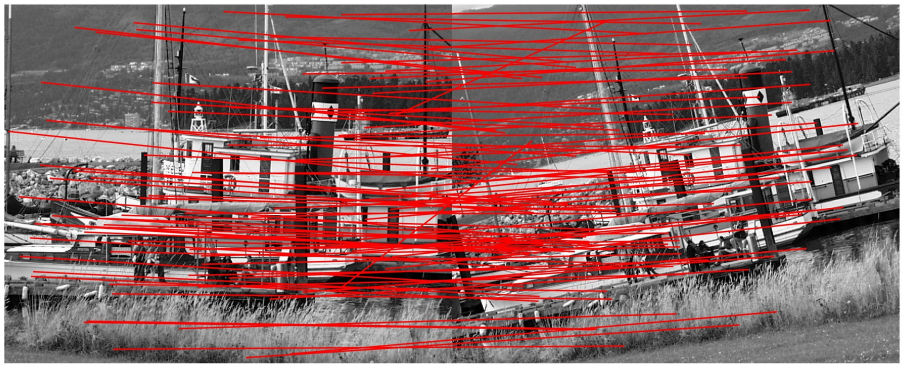
# Image Alignment and Stitching

**Tarun Krishna**
University of Amsterdam
11593040
tarun.krishan@student.uva.nl

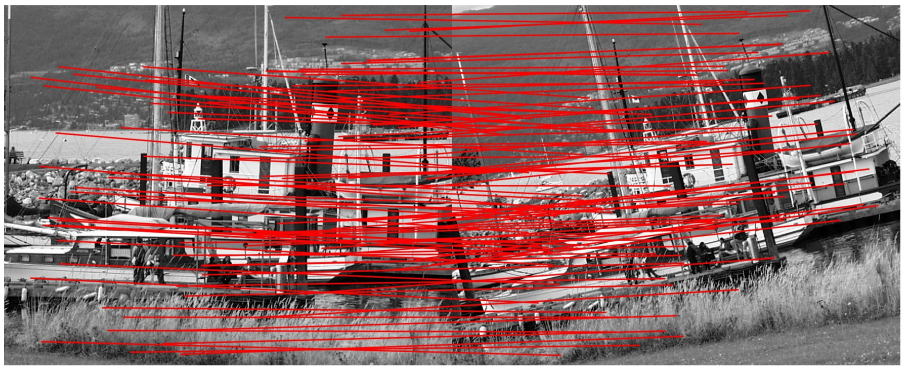
**Dhruba Pujary**
Univeristy of Amsterdam
11576200
dhruba.pujray@student.uva.nl

## 1 Introduction

Image alignment algorithms can discover the correspondence relationships among images with varying degrees of overlap. They are ideally suited for applications such as video stabilization, summarization, and the creation of panoramic mosaics. Image stitching algorithms take the alignment estimates produced by such registration algorithms and blend the images in a seamless manner, taking care to deal with potential problems such as blurring or ghosting caused by parallax and scene movement as well as varying image exposures. In this assignment we try to implement a simple image stitching algorithm based on the transformation achieved through image alignment.



(a) Before RANSAC



(b) After RANSAC

Figure 1: Depiction of matching points before and after RANSAC for inlier detection. As it is clearly observed in Figure1(a) we have 2 outliers. After RANSAC we eliminate those ouliers as shown in Figure1(b)

## 2 Image Alignment

Task of image alignment is, given (stereo)two images both taken from camera rotating with fixed center or a different cameras is to calculate the homography between two given image planes and transform one with respect to another. This could be done by calculating interest points in both images and matching it in images and applying RANSAC to get inliers (Figure1) and finally calculate homogrpahy between the two images using these inliers.

A new image which will be aligned to image 5(b) is obtained by multiplication of affine matrix with the pixel of the image 5(a). It may so happen that after transformation few values will not be integers implies they cannot be directly used as pixel values5(c). There are many ways to correct this like nearest-neighbor interpolation, bilinear interpolation, bicubic interpolation etc. Simplest of all is using nearest-neighbor which is finding the nearest pixel value of the transformed image in the original image. As seen from figure 3(b), using this rounding of technique did gave good results. We tried using bilinear interpolation in which an average of all the four diagonal neighboring pixels but didn't get good results 3(c). Experimenting further, instead of averaging we used median of all the neighboring 8 pixel3(d) in the transformed image. This is similar to using a median filter for removing salt-and-pepper noise but acting only upon the missing pixel values. This technique gave better result that using nearest-neighbor. The edges in a Nearest-neighbor interpolation image 4(a) are coarser than using a median interpolation 4(b).

[1] Well, if we write a transformation of a point using affine transformation it can be represented as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & t_x \\ m_3 & m_4 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Above, we have six unknowns as $m_1, m_2, m_3, m_4, t_x, t_y$ and two equations to solve it i.e under constrained system of equations. To solve for those unknowns we need atleast 6 equations i.e 3 pairs of matching points to solve the affine transformation.

[2] Generally, *N* iterations are needed in order to find an outlier free set with the probability *p* as defined in [1]:

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}$$

Where $\epsilon$ is the percent of outlier (outlier ratio) we are assuming, *s* is the number of samples drawn each time. So, for $p = 0.99$ , $\epsilon = .1$ and $s = 8$, then w'll need 9 iterations of RANSAC. This is the general heuristic schemes which is followed to get best parameters for the given problem. Alternatively one can also run the algorithm several times and then choose the solution giving the largest set of inliers. The main idea however is to generate a hypothesis from random samples (estimating a model) and then verifying it using all the data (scoring).

**Comparing** results of transformation method implemented by us with the one implemented in Matlab is being shown in Figure2. Well the transformation almost looks the same.

**NOTE:** *run demo_script.m which will display the transformed images*

### 2.1 Image Stitching

In image stitching the step followed is same as image alignment where we consider a reference image, align the other image with respect to reference image and overlay one image upon another. Concretely, considering the reference image $I_1$ we align the other image $I_2$ using the image alignment technique described above to obtain $I_2^t$. To overlay the image $I_2^t$, we need to first estimate the size of the frame on which the images will reside. This can be calculated by obtaining the transformed coordinates of the corners of the image $I_2$ and comparing with the coordinates of image $I_1$. The images are then stitched together by picking pixel by pixel from respective image 6(a) 6(b).

**NOTE:** *run demo_script2.m which will display the stitched images*

[1] How many matches do we need to solve an affine transformation which can be formulated as shown in Figure1

[2] How many iterations in average are needed to find good transformation parameters?

(a) boat1: using our implementation (b) boat1: using imtransform

(c) boat2: using our implementation (d) boat2: using imtransform

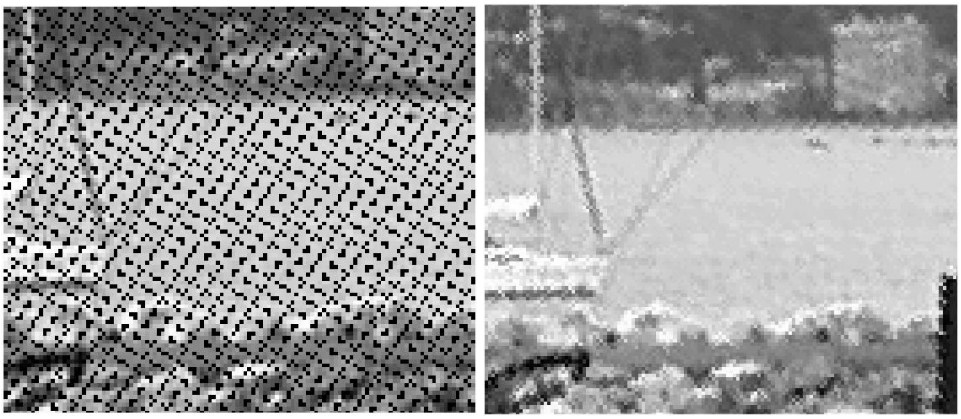
Figure 2: Comparison of transformation generated by our implementation and using in-built Matlab function.

## 3 Conclusion

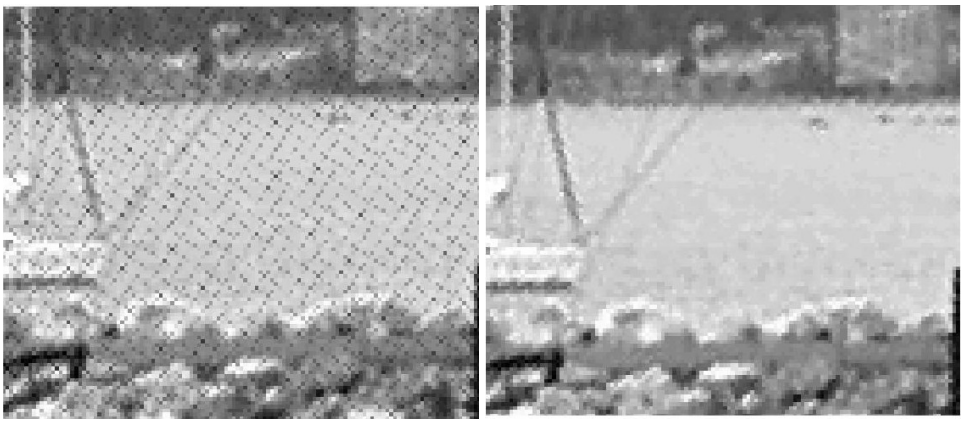
In this report, we used SIFT feature detector and descriptor to identify interest point and find the matching points in another image. These matching points are then used to aliging one image with respect to another by identifying the best affine transformation using RANSAC.Finally, we used all these techniques to obtain a stitched image.

## References

[1] A. Hast, J. Nysjo, and A. Marchetti. Optimal ransac - towards a repeatable algorithm for finding the optimal set. *Journal of WSCG*, no.1:21–30, 2013.



(a) Transformed image (b) Nearest-neighbor Interpolation

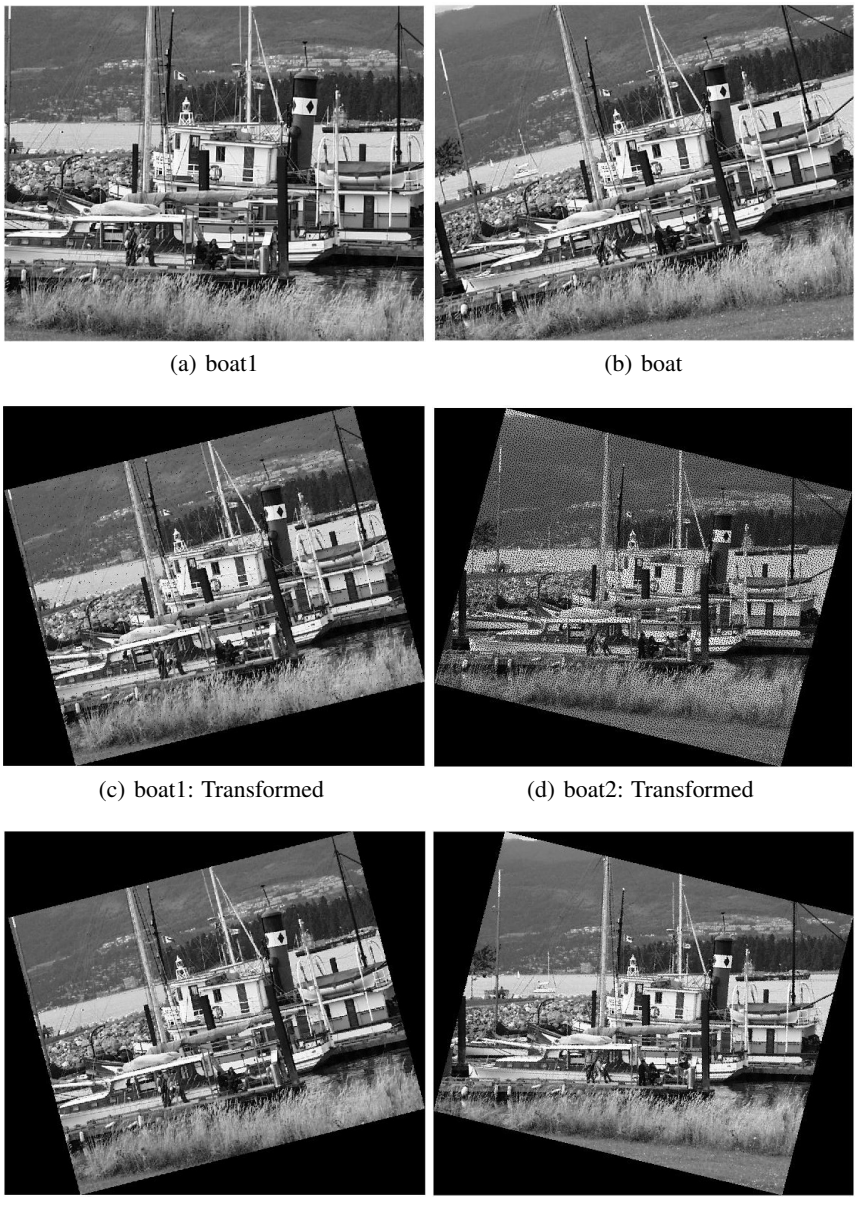


(c) Average Interpolation (d) Median Interpolation

Figure 3: Figure3(a) shows pixel values after transformation and Figure 3(b) Figure3(c) and Figure3(d) different interpolation to cope up with unfilled pixels in Figure3(a).

(a) Nearest-neighbor interpolation (b) Median interpolation

Figure 4: Comparison between nearest-neighbor interpolation Figure4(a) and median interpolation Figure4(b).



(a) boat1 (b) boat

(c) boat1: Transformed (d) boat2: Transformed

(e) boat1: Nearest-neighbor interpolation (f) boat2: Nearest-neighbor Interpolation

Figure 5: Left column of above subplot represents transformation on boat1 with right image as reference image and vice-versa for right subplot.

(a) Left image as reference (b) Right image as reference

Figure 6: Represents stitching of images Figure6(a) considers left image as reference image and transforms the the right and stitch it accordingly and Figure6(b) does the opposite of that.