BLACKBOARD MOSAICING

Project report submitted in partial fulfillment
of the requirements for the course of

Multimedia Processing and Applications

in

Computer Science and Engineering

by

Paresh Mishra - 16ucs123
Akanksha Maheshwari - 16ucs111

Under Guidance of
Dr. Anukriti Bansal

Department of Computer Science and Engineering
The LNM Institute of Information Technology, Jaipur

November 2018

Acknowledgments

The contributions made to this project could not have been possible without the constant support of few people. We would like to use this opportunity to thank our mentor Dr. Anukriti Bansal for their continual motivating words and the discussions that we had with them.
We are grateful for the constant support and guidance received from both the members who contributed to this project.

ABSTRACT

Image mosaicing is one of the most important subject of research in computer vision. It is one the techniques of image processing which is useful for tiling digital images. Mosaicing is blending together of several arbitrarily shaped images to form one large radiometrically balanced image so that the boundaries between the original images are not seen. We can combine information from many images into one image for better understanding what they represent. For this, we want an algorithm in which the features of the mosaiced object are being viewed and by this we can get more information from the image.

In this project, we have implemented the algorithm by which we can show these features in best possible view. This process used some existing mosaicing algorithm for generating novel views from limited number of views.

IMAGE MOSAICING

What is Mosaicing?

“Mosaic” originates from an old Italian word “mosaico” which means a picture or pattern produced by arranging together small pieces of stone, tile, glass, etc.

Image mosaicing is a process that stitches multiple, overlapping snapshot **images** together in order to produce one large, high resolution composite image.
It is being done in this project such that images taken by normal camera can be used to create a larger field of view using a image mosaicing program.

Need of Image Mosaicing

There are some situations where it is not possible to capture the large documents with the given imaging media there we require the mosaicing techniques to capture all the docs in one field of view
Image mosaicing not only allow us to create a larger field of view using normal camera. The resultant image can also be used for the texture mapping of 3d environment such that the users can view the surrounding scene with real images.

Aim of Project

The user can take images using a hand held camera and then take images at different angles with some overlapping area, then the user can use the program to load the image in and then make the mosaic.

While taking the images there should be some corresponding points between the two images to stitch them together.

IMAGE MOSAICING MODEL

1. Feature Extraction
The first step in image mosaic process is feature detection. Features are the elements in the two input images to be matched. For images to be matched they are taken inside

an image patches. These image patches are groups of pixel in images. Patch matching is done for the input images .

2. Finding Homography

Find the homography matrix and apply inverse homography on the corner points of second image

3. Defining Canvas size

Find the maximum and the minimum value of x and y coordinates in both the images. The final width of the canvas would be the maximum value of x + x_offset and the final height will be maximum value of y + y_offset.

4. Stitching of images

a) Paste the first image as it is on canvas after adding the x and y offset and iterating through whole image.

b) After stitching the first image apply the homography to each pixel of the canvas to get coordinate that matches with a pixel of second image then assign its color value to the pixel in canvas.

Implementation Steps

1. Load the two images

2. Select the corresponding points between the two images

   Here we have used the **cpselect tool** for selecting the corresponding points in images.**Impixeinfo()** can also be used in place of this to select the corresponding points.
   Here we have to choose **4 corresponding points on blackboard only**, because only it is planar.

3. Store the corresponding points in an array

4. Find homography matrix of two images.

5. Find the inverse of homography

6. Apply the inverse homography to the corner points of the second image

7. Store the projected points as well as the corner points of first image in an array.

8. Find the maximum and the minimum value of x and y coordinates in both the images

9. If the minimum values of x and y are less than zero then find the absolute value of x and y and finally find the x offset and y offset.
If the minimum values are not less than zero then offset will be zero.

10. The final width of the canvas would be the maximum value of x + x offset and the final height will be maximum value of y + y offset

11. Create the matrix(2d array) of zeros with the dimensions of final width and final height . e.g. temp=zeros(height,width);

12. Stitch the first image as it is after adding the x offset and y offset and iterating through whole image. E.g.
temp(y_offset+row_number,x_offset+column_no,channel_no) =
img1(row_number,column _no,channel_no);

14. After stitching the first image apply the homography to each pixel of the canvas to get coordinate (a1,b1) (say) and if that (a1,b1) matches with a pixel of second image then assign its color value to the pixel in canvas.

15. Show the final image using **imshow()** function

16. Save the image using **imwrite('img',filename).**

CODE

MAIN FUNCTION

```
% code for stitching images
close all;
clear all;

%%%%%%%%%%  FOR m0 + m5  %%%%%%%%%%

% loading of images
img1=imread('m0.jpg');
img2=imread('m5.jpg');

%Store the corresponding points in an array
a=[1003,916,963,1505,1722,1536,1735,1115];
b=[1130,236,948,712,1881,1045,2065,617];

%Call a function to do the remaining process and get final result
result=mosaic_123(img1,img2,a,b,0);
imwrite(result,'m0m5.jpg');

%%%%%%%%%%  FOR m0m5 + m4  %%%%%%%%%%

img2 = imread('m4.jpg');
img1 = imread('m0m5.jpg');

%Store the corresponding points in an array
b=[95,576,22,1105,804,1022,1196,1082];
a=[1628,955,1575,1753,2333,1535,2627,1579];

%Call a function to do the remaining process and get final result
result=mosaic_123(img1,img2,a,b,2);
imwrite(result,'m0m5m4.jpg');
figure,imshow(result);

%%%%%%%%%%  FOR m0m5m4 + m2  %%%%%%%%%%

img1 = imread('m0m5m4.jpg');
img2 = imread('m2.jpg');
```

```
%Store the corresponding points in an array
a= [2579,928,2629,1580,2957,956,2976,1178];
b= [349,583,157,1339,1105,701,1035,1025];

%Call a function to do the remaining process and get final result
result=mosaic_123(img1,img2,a,b,0);
imwrite(result,'m0m5m4m2.jpg');
figure,imshow(result);

%%%%%%%%%%  FOR m0m5m4m2 + m3  %%%%%%%%%%

img1 = imread('m0m5m4m2.jpg');
img2 = imread('m3.jpg');

%Store the corresponding points in an array
a= [2572,905,2879,950,2591,1496,2935,1616];
b= [127,129,719,73,85,827,733,1011];

%Call a function to do the remaining process and get final result
result=mosaic_123(img1,img2,a,b,0);
imwrite(result,'m0m5m4m2m3.jpg');
figure,imshow(result);

%%%%%%%%%%  FOR m0m5m4m2m3 + m1  %%%%%%%%%%

img1 = imread('m0m5m4m2m3.jpg');
img2 = imread('m1.jpg');

%Store the corresponding points in an array
a= [2255,1347,2591,1495,2879,949,2976,1178];
b= [899,1449,1539,1259,1667,325,1993,473];

%Call a function to do the remaining process and get final result
result=mosaic_123(img1,img2,a,b,0);
imwrite(result,'m0m5m4m2m3m1.jpg');
figure,imshow(result);
```

MOSAICING FUNCTION

```
function [canvas] = mosaic_123(img1,img2,a,b,c)

img1=im2double(img1);

img2=im2double(img2);
Homo=homography(a(1),a(2),a(3),a(4),a(5),a(6),a(7),a(8),b(1),b(2),b(3),b(4),b(5),b(6),b(7),b(8));
%Find the inverse of homography
invHomo=inv(Homo);

%Apply the inverse homography to the corner points of the second image
projected_point=[1 1 1] * invHomo;
l1=projected_point(1,1)/projected_point(1,3);
m1=projected_point(1,2)/projected_point(1,3);

projected_point=[1 size(img2,1) 1] * invHomo;
l2=projected_point(1,1)/projected_point(1,3);
m2=projected_point(1,2)/projected_point(1,3);

projected_point=[size(img2,2) 1 1] * invHomo;
l3=projected_point(1,1)/projected_point(1,3);
m3=projected_point(1,2)/projected_point(1,3);

projected_point=[size(img2,2) size(img2,1) 1] * invHomo;
l4=projected_point(1,1)/projected_point(1,3);
m4=projected_point(1,2)/projected_point(1,3);

%Store the projected points as well as the corner points of first image in an array.
corners=[l1 m1;l2 m2;l3 m3;l4 m4;1 1;1 1664;2496 1;2496 1664];

%Find the maximum  and the minimum value of x and y coordinates  in both the images
MAX_X=corners(1,1);
for i=1:8
   if MAX_X<= corners(i,1)
       MAX_X=corners(i,1);
   end
end
disp(MAX_X)

MAX_Y=corners(1,2);
for i=1:8
```

```
    if MAX_Y<= corners(i,2)
        MAX_Y=corners(i,2);
    end
end
disp(MAX_Y)

MIN_X=corners(1,1);
for i=1:8
    if MIN_X>= corners(i,1)
        MIN_X=corners(i,1);
    end
end
disp(MIN_X)

MIN_Y=corners(1,2);
for i=1:8
    if MIN_Y>= corners(i,2)
        MIN_Y=corners(i,2);
    end
end
disp(MIN_Y)

%Find offset values for x
if(MIN_X<0)
    x_offset=round(abs(MIN_X));
else
    x_offset=0;
end

%Find offset values for y
if(MIN_Y<0)
    y_offset=round(abs(MIN_Y));
else
    y_offset=0;
end

%calculating final height and width of canvas
final_width=round(MAX_X+x_offset);
final_height=round(MAX_Y+y_offset);

disp(x_offset)
disp(y_offset);
```

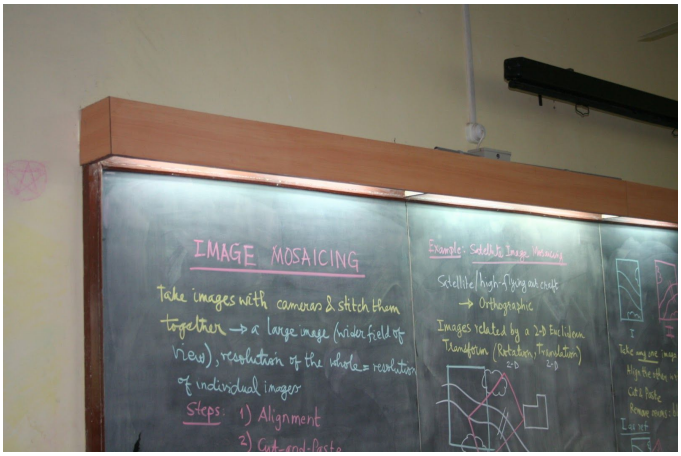
```
%Create the matrix(2d array) of zeros with the dimensions of final width and final height
canvas=zeros(final_height,final_width);
canvas=im2double(canvas);

%Stitch the first image as it is after adding the x offset and y offset and iterating through whole image.
for i=1:size(img1,1)
   for j=1:size(img1,2)

       canvas(y_offset+i,x_offset+j,1)=img1(i,j,1);    %Red channel
       canvas(y_offset+i,x_offset+j,2)=img1(i,j,2);    %Green channel
       canvas(y_offset+i,x_offset+j,3)=img1(i,j,3);    %Blue Channel

   end
end

%figure,imshow(canvas);

%For second image
for i=1:size(canvas,1)
   for j=1:size(canvas,2)
       % Applying homography on each pixel of canvas
       projected_point= [j, i, 1]*Homo;
       a1=projected_point(1,1)/projected_point(1,3);
       b1=projected_point(1,2)/projected_point(1,3);
       a1=round(a1); % rounding values
       b1=round(b1);
       %check if these values matches with pixel in image 2, if yes,
       %assign color values
     if(a1>=1 && b1>=1 && a1<=size(img2,2) && b1<=size(img2,1))
       canvas(i+y_offset,j+x_offset,1)=img2(b1,a1,1);    %red channel
       canvas(i+y_offset,j+x_offset,2)=img2(b1,a1,2);   % green channel
       canvas(i+y_offset,j+x_offset,3)=img2(b1,a1,3);  % blue channel
     end
   end
end



%show final output
figure,imshow(canvas);
```

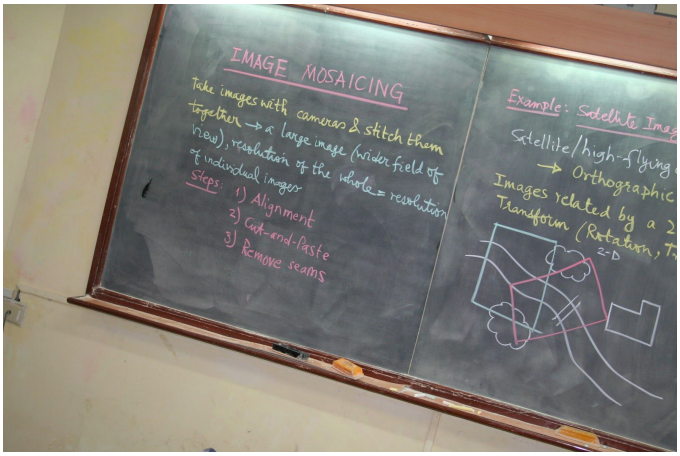
HOMOGRAPHY FUNCTION

```
function H=homography(x1,y1,x2,y2,x3,y3,x4,y4 , xp1,yp1,xp2,yp2,xp3,yp3,xp4,yp4)
%This function will find the homography betweeb 4 points using svd



A=[
-x1  -y1  -1   0    0    0   x1*xp1   y1*xp1   xp1;
 0    0    0 -x1   -y1  -1   x1*yp1   y1*yp1   yp1;
-x2  -y2  -1   0    0    0   x2*xp2   y2*xp2   xp2;
 0    0    0 -x2   -y2  -1   x2*yp2   y2*yp2   yp2;
-x3  -y3  -1   0    0    0   x3*xp3   y3*xp3   xp3;
 0    0    0 -x3   -y3  -1   x3*yp3   y3*yp3   yp3;
-x4  -y4   -1  0    0    0   x4*xp4   y4*xp4   xp4;
 0    0    0  -x4  -y4  -1   x4*yp4   y4*yp4   yp4];

[U,S,V] = svd(A);



H=V(:,end);
H=reshape(H,3,3);
```

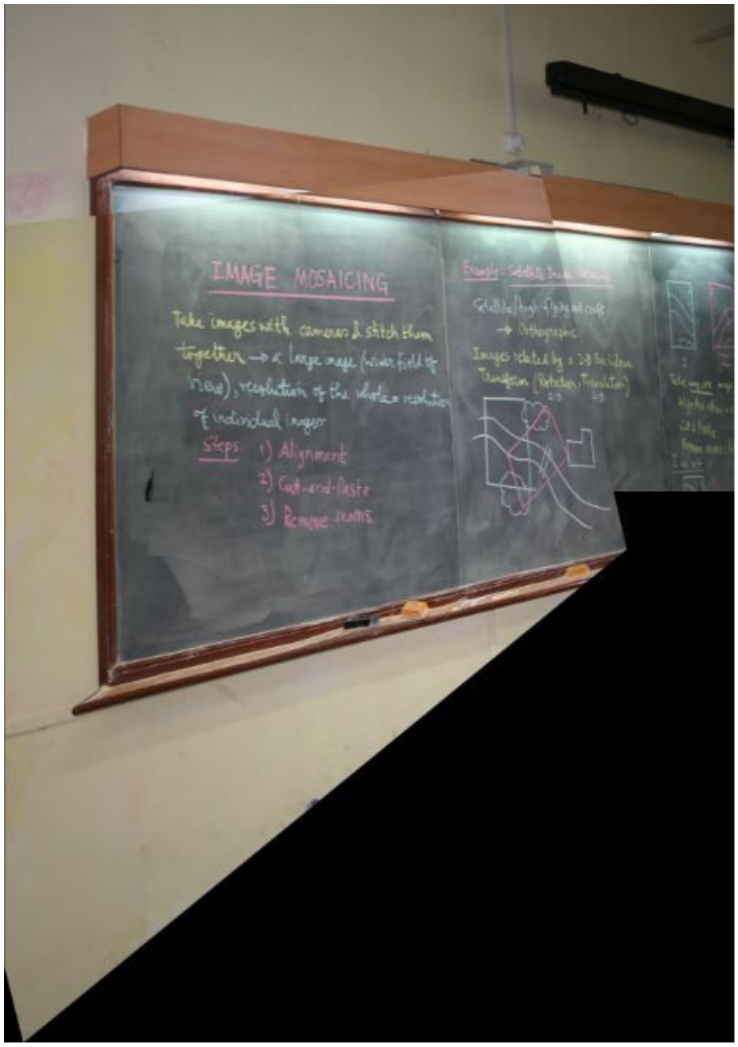
RESULTS

INPUT IMAGES : Img0 + Img5

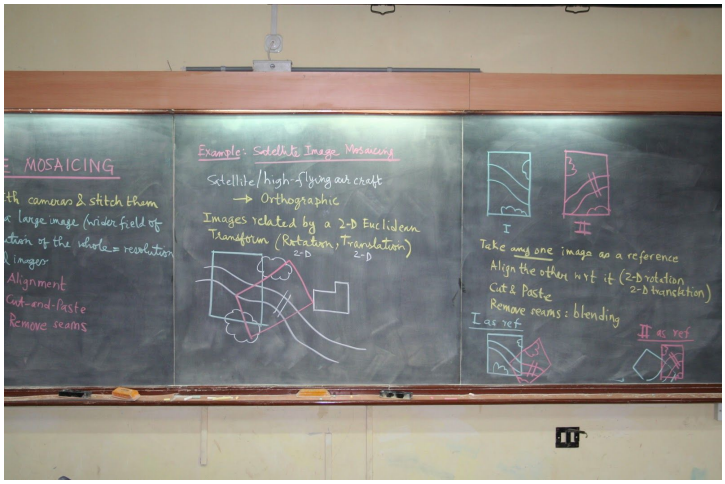




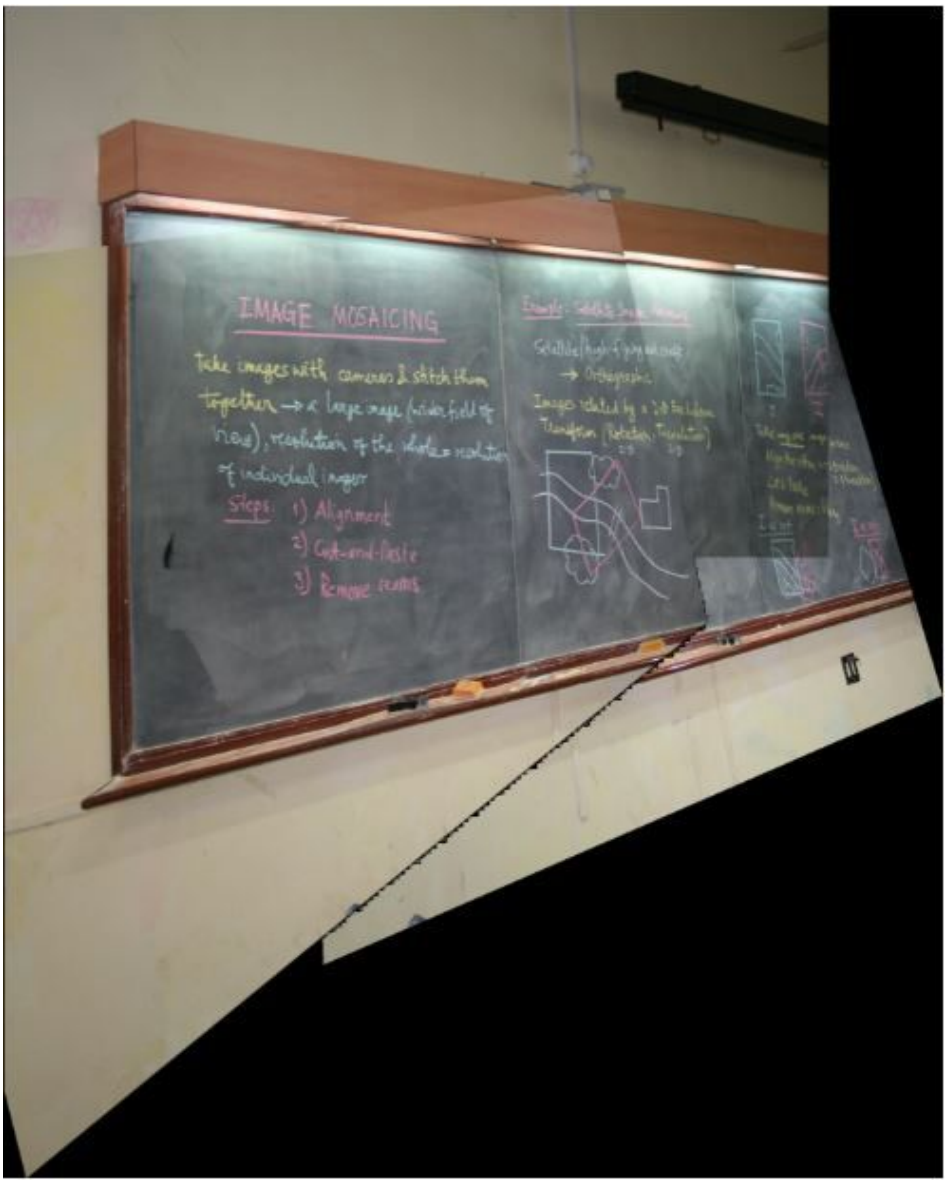
OUTPUT IMAGE



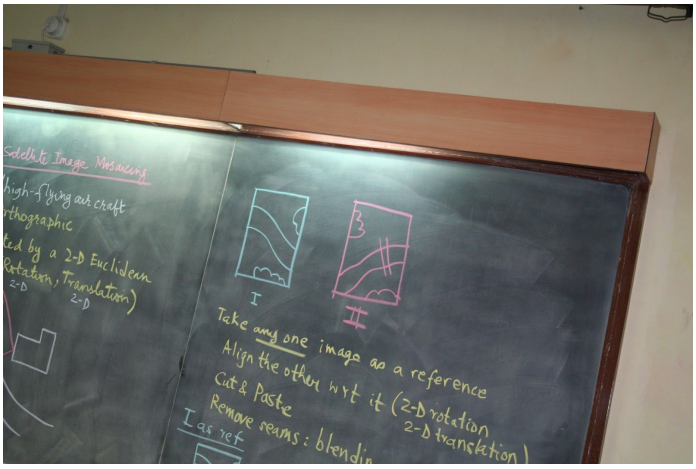
INPUT IMAGE : m0m5 + m4



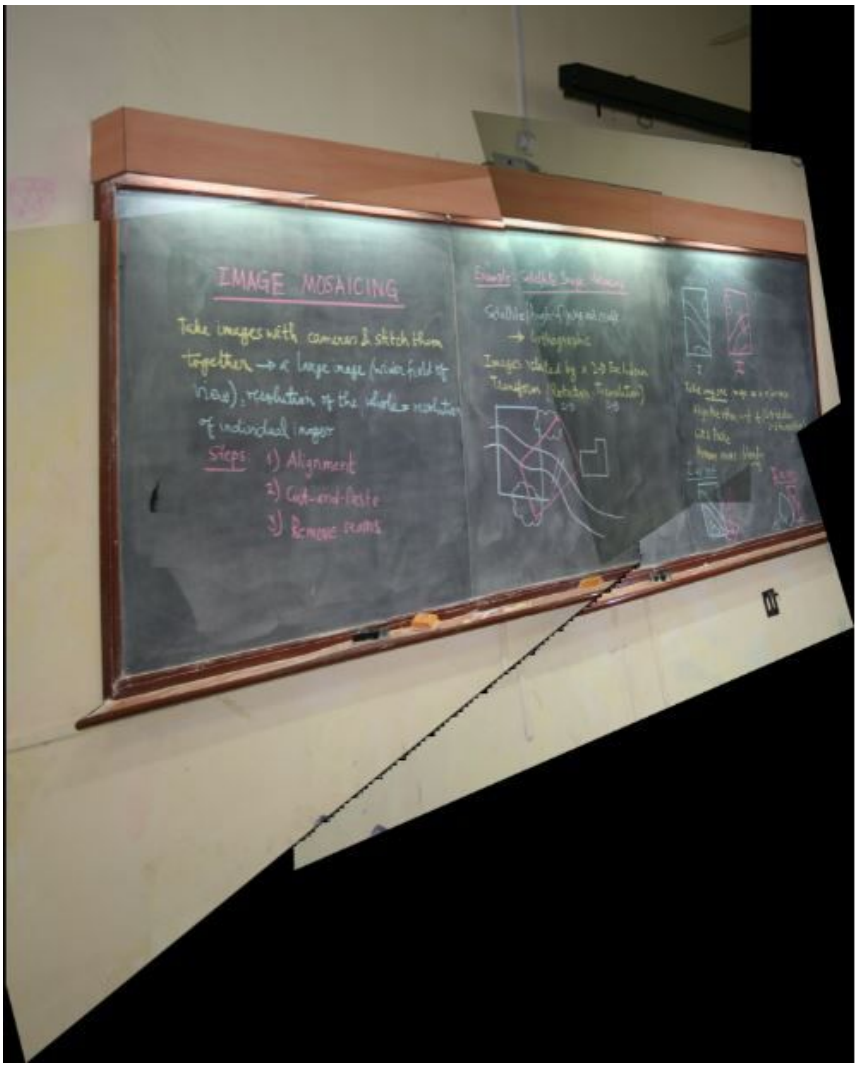
OUTPUT IMAGE



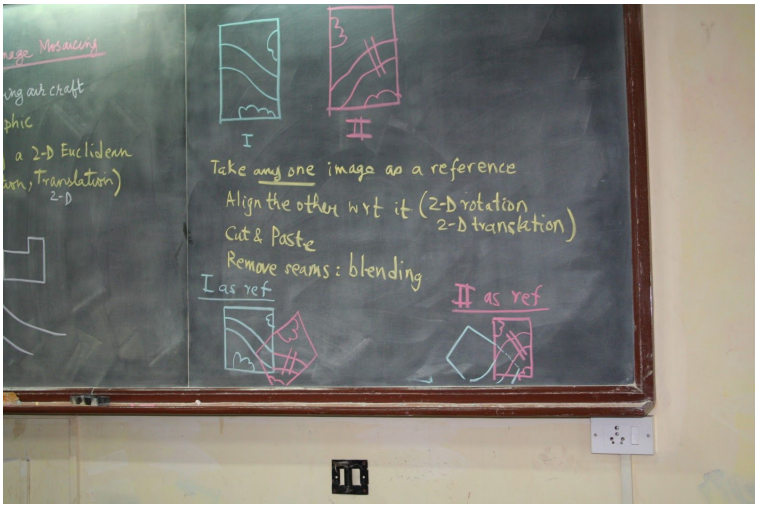
INPUT IMAGES : m0m5m4 + m2



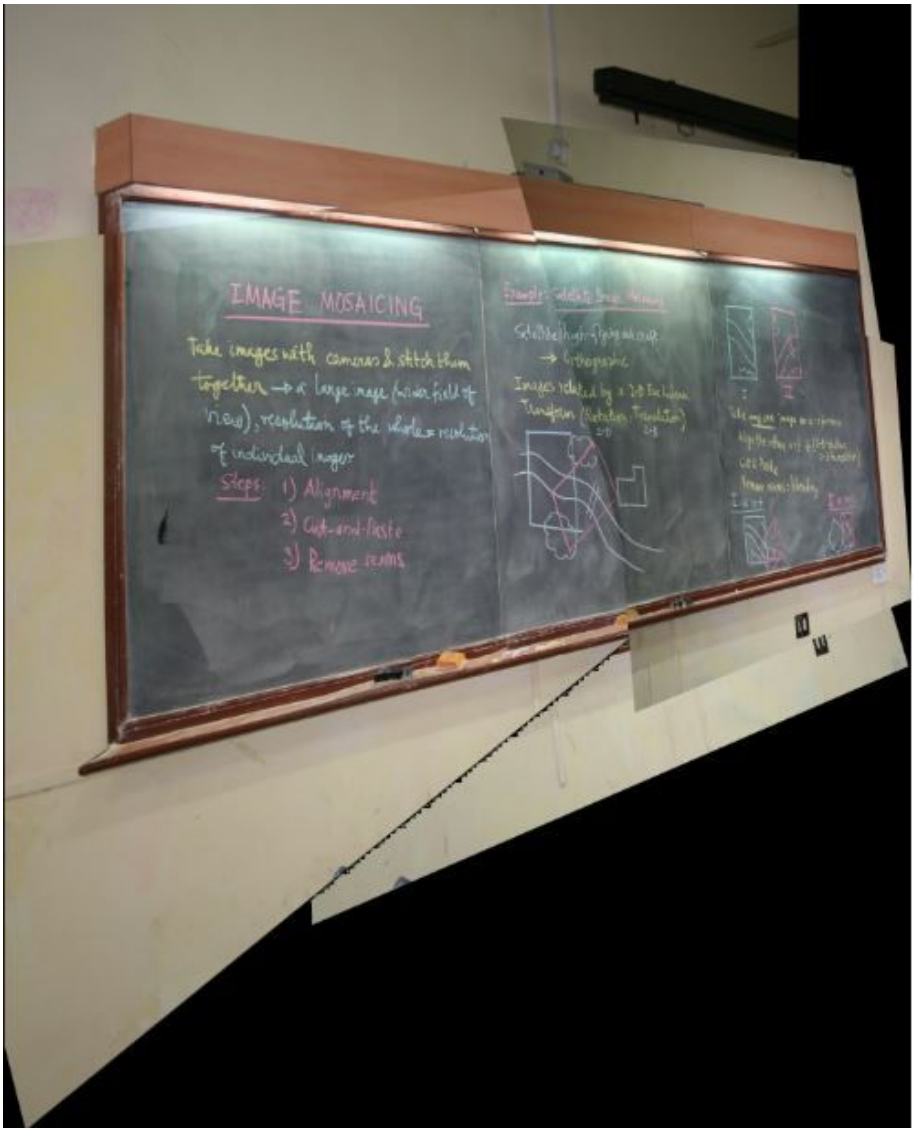
OUTPUT IMAGE



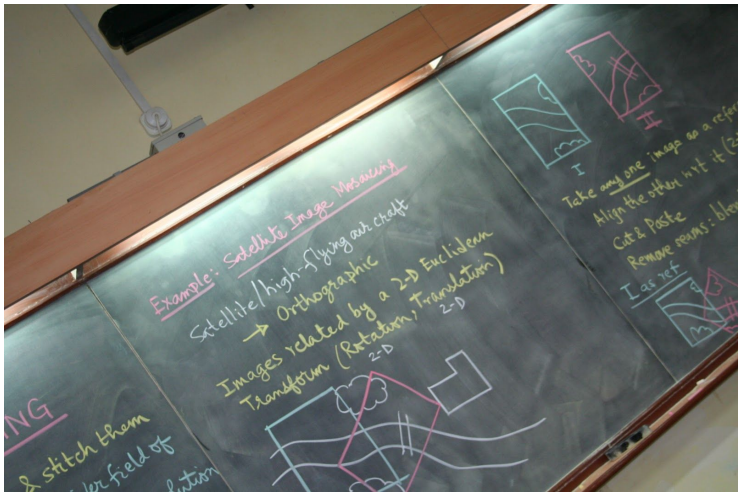
INPUT IMAGE : m0m5m4m2 + m3



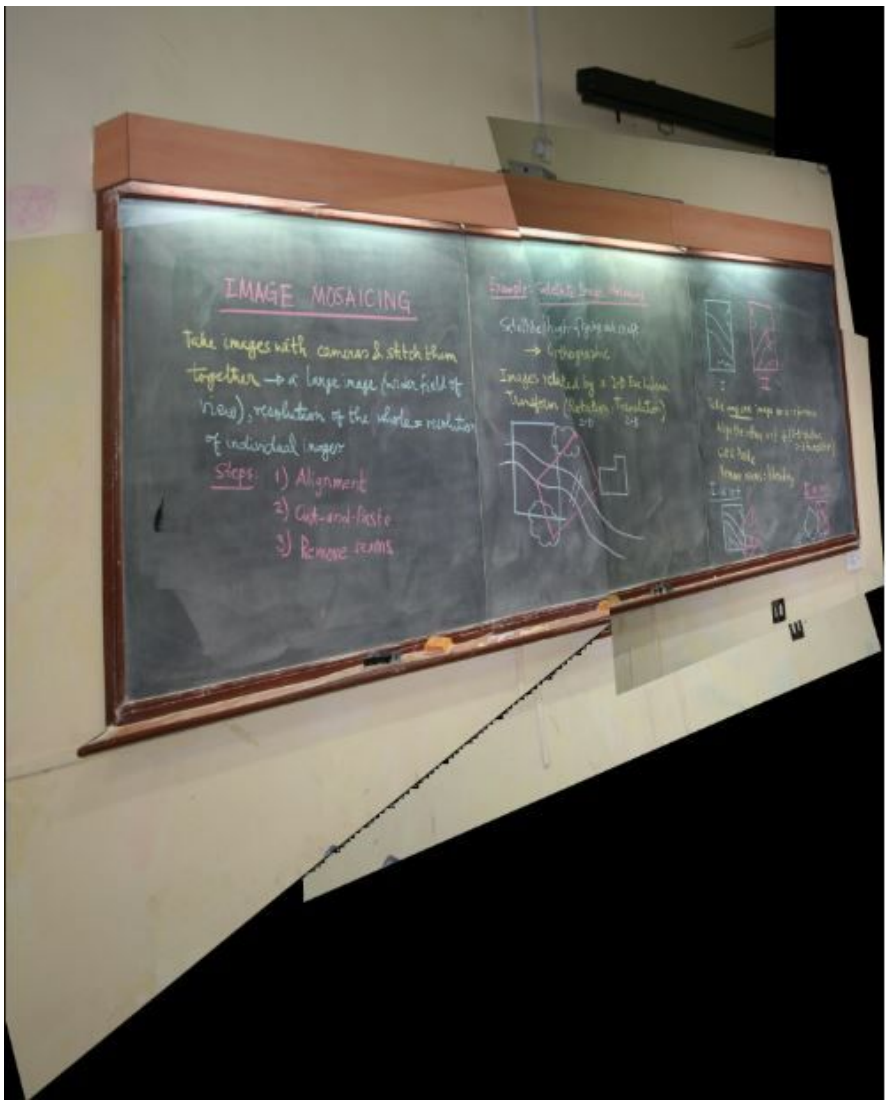
OUTPUT IMAGE



INPUT IMAGE : m0m5m4m2m3 + m1



OUTPUT IMAGE



Applications

- Constructing high resolution images that cover an unlimited field of view using inexpensive equipment.
- Creating immersive environments for effective information exchange through the internet.
- Using image mosaicing to make a significant impact in video processing.

Observations

- This algorithm works only for 2d images for three dimensional images this algorithm will not work.
- After stitching the images we get the large field of view
- Selection of images in correct order is most important, a bit change in order can distort the resultant to a large extent.
- This method works fine for all types of images but they consume time.
- Mosaicing of multiple images cannot be achieved by repeatedly warping new images to reference image. Hence after mosaicing 5 images to the reference image the image alignment does not look good.