ASSIGNMENT - 9

MOSAICING

Problem Statement

We are provided with 6 images of a Blackboard, clicked by an immature photographer.
Our goal is to stitch all of those disoriented and incomplete images into one image containing whole Blackboard.

Theory

- Take two images to be stitched. Consider plane of one of the image (say image 1) as the reference base in which the other image (say image 2) is to be projected.
- Choose 4 corresponding common points from both the images
- Using those 8 points form a Homography matrix
- Choose corner points (say C_2) of the image 2 and apply the Eq. given below to find its corner points in projected transform

$$C`_2 = H * C_2$$

$C`_2$ - Projected points
C_2 - Initial points
H - Homography Matrix

- Find X_min , X_max , Y_min , Y_max using C`2 and C1 (corner points of the reference image)
- | X_min | and | Y_min | are the offsets
- Create a blank canvas of size [Y_max-Y_min , X_max-X_min]
- Insert the reference Image into the canvas at appropriate position
- Iterate over the canvas and for every blank pixel value find the coordinates of the image projected in this plane using Eq.

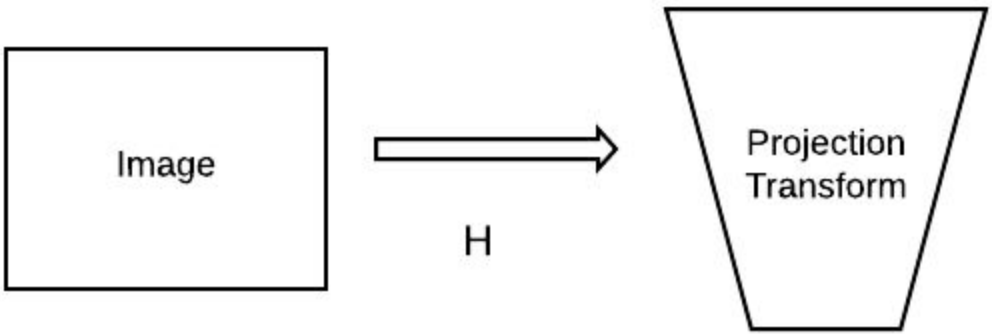
$$Z_2 = H^{-1} * Z_1$$

Z_2 - Original coordinates
Z_1 - Projected coordinates (point of canvas)
H^{-1} - Inverse of Homography Matrix

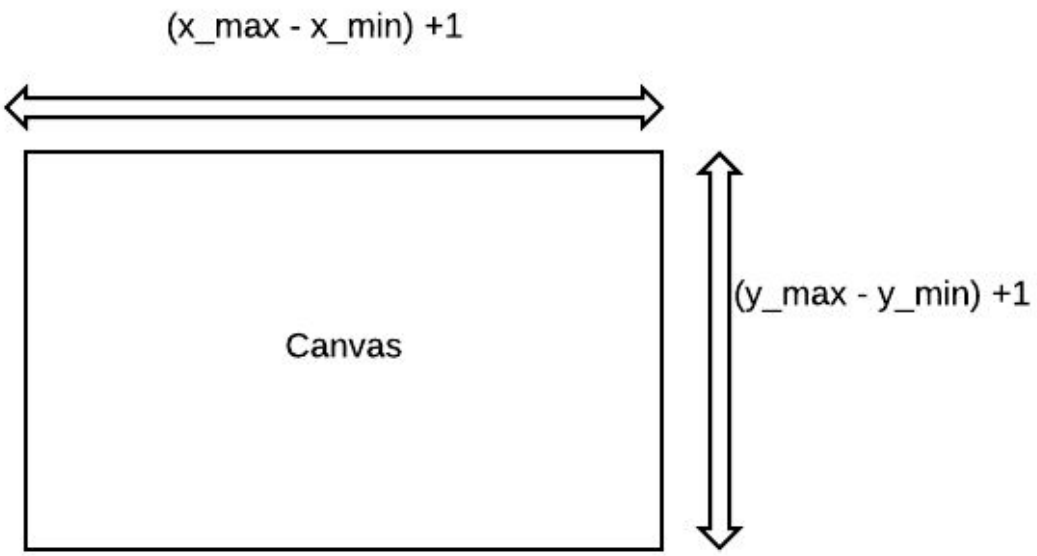
- If Coordinates Z2 lies in the initial image 2, copy the pixel value from that coordinate into canvas

Implementation

- Upload two Images , Image 1 and Image 2
- Get 4 corresponding common points from each image by user input
- Using findHomography() built in function obtain a Homography matrix of those 8 points
- Obtain 4 corner points of both image 1 (say C1) and image 2 (say C2)
- Apply projective transform equation on the points of image 2 to get 4 corner points (say C`2) of its projection in the plane of image 1



- Now find x_min, x_max, y_min, y_max using C`2 and C1
- Create a blank canvas of size [(Y_max - Y_min) + 1 , (X_max - X_min)+ 1] , extra row-column added just for safety



- Iterating over image 1 for every (i,j) coordinate copy the pixel values in the canvas at position (i - y_min , j - x_min) , as offsets should be subtracted in order to insert image 1 at right position in canvas
- Iterate over canvas, for every coordinate (j + x_min , i + y_min) if there is blank pixel (i.e black color) apply Projective inverse equation on current coordinate to get corresponding coordinates in the original image 2
- Offsets are added during iteration over canvas in order to get the actual coordinates of the projected image
- Check for the bounds of obtained coordinates and copy the pixel value from original image 2
- Repeat this procedure with all the images using previous canvas (result) as the reference image for next iteration and the last canvas will give a complete image

Source Code

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <cmath>

using namespace cv;
using namespace std;
struct coordinate
{
        int x;
        int y;
```

```
};
typedef struct coordinate point;
point src[4];
point dst[4];
int csrc = 0;
int cdst = 0;
void source(int event, int x, int y, int flags, void* param)
{

        if (event == CV_EVENT_LBUTTONDOWN)
        {
                if (csrc < 4)
                {
                        printf("%d,%d\n",
                                x, y);
                        src[csrc].x = x;
                        src[csrc].y = y;
                        csrc++;
                }
        }
}
void destination(int event, int x, int y, int flags, void* param)
{

        if (event == CV_EVENT_LBUTTONDOWN)
        {
                if (cdst < 4)
                {
                        printf("%d,%d\n",
                                x, y);
                        dst[cdst].x = x;
                        dst[cdst].y = y;
                        cdst++;
                }
        }
}
Mat getHomography()//function to calculate homography matrix using pair of 4 corresponding points in
source and destination images
{

        vector<Point2f> points1;
        vector<Point2f> points2;

        //points1 corresponds to 4 points in source image(image to be transformed)

        points1.push_back(Point2f(src[0].x, src[0].y));
        points1.push_back(Point2f(src[1].x, src[1].y));
        points1.push_back(Point2f(src[2].x, src[2].y));
        points1.push_back(Point2f(src[3].x, src[3].y));

        //points2 corresponds to 4 points in destination image(image whose projective plane is taken as
reference)

        points2.push_back(Point2f(dst[0].x,dst[0].y));
        points2.push_back(Point2f(dst[1].x, dst[1].y));
```

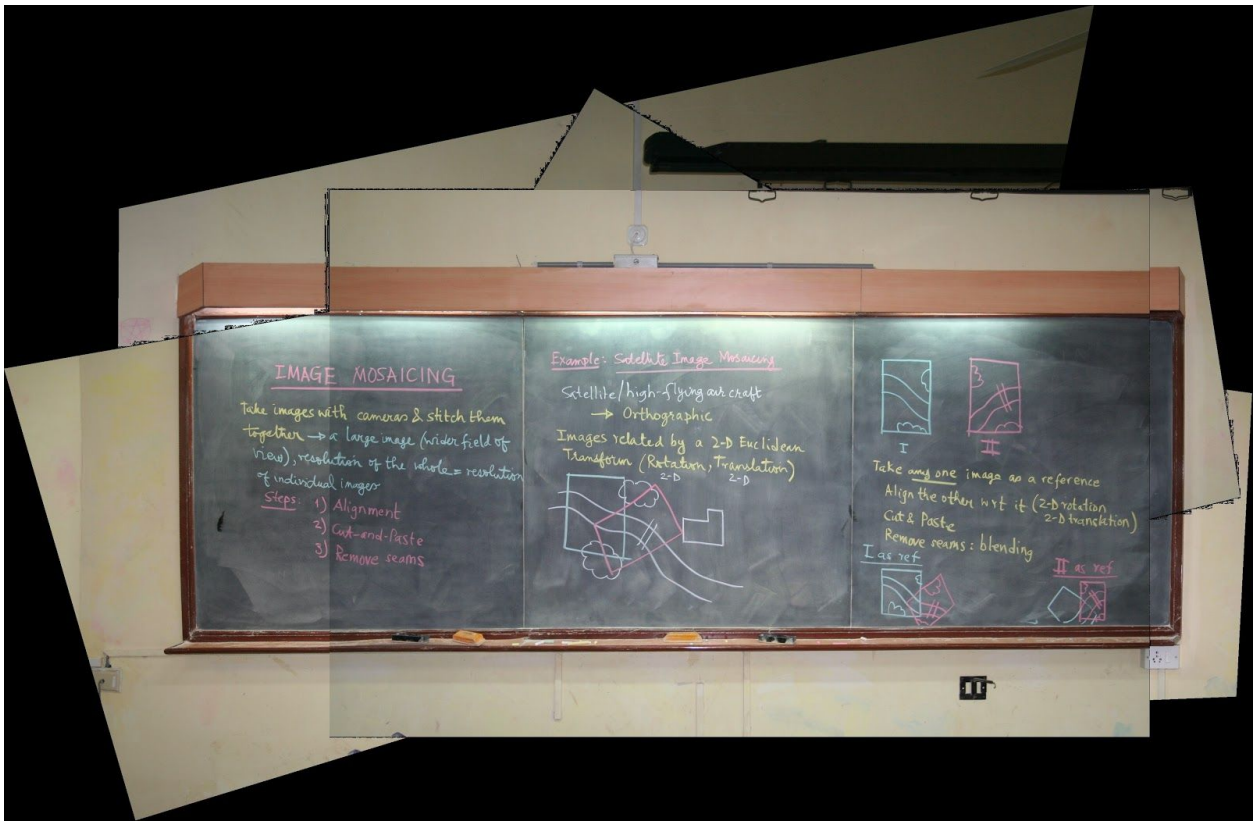
```
        points2.push_back(Point2f(dst[2].x, dst[2].y));
        points2.push_back(Point2f(dst[3].x, dst[3].y));

        Mat H = findHomography(Mat(points1), Mat(points2), 8, 3.0);
        return H;
}

void Compute(Mat img, Mat ref)
{

        /*double homo[8][9];
        for (int i = 0; i < 4; i++)
        {
                homo[2 * i][0] = (-1)*(src[i].x);
                homo[2 * i][1] = (-1)*(src[i].y);
                homo[2 * i][2] = -1;
                homo[2 * i][3] = 0;
                homo[2 * i][4] = 0;
                homo[2 * i][5] = 0;
                homo[2 * i][6] = (dst[i].x)*(src[i].x);
                homo[2 * i][7] = (dst[i].x)*(src[i].y);
                homo[2 * i][8] = dst[i].x;

                homo[(2 * i) + 1][0] = 0;
                homo[(2 * i) + 1][1] = 0;
                homo[(2 * i) + 1][2] = 0;
                homo[(2 * i) + 1][3] = (-1)*(src[i].x);
                homo[(2 * i) + 1][4] = (-1)*(src[i].y);
                homo[(2 * i) + 1][5] = -1;
                homo[(2 * i) + 1][6] = (dst[i].y)*(src[i].x);
                homo[(2 * i) + 1][7] = (dst[i].y)*(src[i].y);
                homo[(2 * i) + 1][8] = dst[i].y;

        }
        for (int i = 0; i < 8; i++)
        {
                printf("\n");
                for (int j = 0; j < 9; j++)
                {
                        printf("  %f", homo[i][j]);
                }

                printf("\n");
        }
        Mat H(8, 9, CV_64FC1, homo);
        Mat w, u, vt;
        SVDecomp(H, w, u, vt);
        printf("\n");
        printf("%d   %d", vt.rows, vt.cols);
        double a[3][3];
        int counter = 0;
        for (int i = 0; i < 3; i++)
        {
                printf("\n");
                for (int j = 0; j < 3; j++)
                {
```

```
                        a[i][j] = (vt.at<double>((vt.rows) - 1, counter))/(vt.at<double>((vt.rows) - 1,
(vt.cols)-1));
                        printf("   %f", a[i][j]);
                        counter++;
                }

        }
        */
        Mat A = getHomography();//get homography matrix
        double a[3][3];
        for (int i = 0; i < 3; i++)//convert Mat into Array
        {
                printf("\n");
                for (int j = 0; j < 3; j++)
                {
                        a[i][j] = A.at<double>(i,j);
                        printf("   %f", a[i][j]);

                }

        }
        point srcedges[4] = {{0,0},{img.cols - 1,0},{0,img.rows - 1},{img.cols - 1,img.rows - 1}};
        point dstedges[4];
        for (int i = 0; i < 4; i++)//obtain corner points of resultant quadrilateral after
transformation of original image
        {
                dstedges[i].x = ceil(((srcedges[i].x)*a[0][0] + (srcedges[i].y)*a[0][1] + a[0][2]) /
((srcedges[i].x)*a[2][0] + (srcedges[i].y)*a[2][1] + a[2][2]));
                dstedges[i].y = ceil(((srcedges[i].x)*a[1][0] + (srcedges[i].y)*a[1][1] + a[1][2]) /
((srcedges[i].x)*a[2][0] + (srcedges[i].y)*a[2][1] + a[2][2]));
        }
        //calculate x and y offset ,as well as dimension of final image after stitching the two
overlapping snapshots of blackboard
        int xmin = min(dstedges[0].x, min(dstedges[1].x, min(dstedges[2].x, dstedges[3].x)));
        if (xmin > 0)
        {
                xmin = 0;
        }
        int xmax = max(dstedges[0].x, max(dstedges[1].x, max(dstedges[2].x, dstedges[3].x)));
        if (xmax < ref.cols)
        {
                xmax = ref.cols;
        }
        int ymin = min(dstedges[0].y, min(dstedges[1].y, min(dstedges[2].y, dstedges[3].y)));
        if (ymin > 0)
        {
                ymin = 0;
        }
        int ymax = max(dstedges[0].y, max(dstedges[1].y, max(dstedges[2].y, dstedges[3].y)));
        if (ymax < ref.rows)
        {
                ymax = ref.rows;
        }
        printf("\n%d Size of Resultant Image   - %d", ymax - ymin + 1, xmax - xmin + 1);
        Mat res(ymax - ymin + 1, xmax - xmin + 1, CV_8UC3, Scalar(0, 0, 0));//creating Mat for final
stitched image , intially its a black image
```

```
        for (int i = 0; i < ref.rows; i++)//pasting destination image/reference image directly on final
canvas
        {
                for (int j = 0; j < ref.cols; j++)
                {
                        for (int c = 0; c < 3; c++)
                        {
                                res.at<Vec3b>(i - ymin, j - xmin)[c] = ref.at<Vec3b>(i, j)[c];//copy
and paste reference/base image directly to canvas at offset

                        }
                }
        }
        for (int i = 0; i < res.rows; i++)//stiching transformed image upon final canvas
        {
                for (int j = 0; j < res.cols; j++)
                {
                        if (res.at<Vec3b>(i, j)[0] == 0 && res.at<Vec3b>(i, j)[1] == 0 &&
res.at<Vec3b>(i, j)[2] == 0)//if the pixel under consideration devoids color then only go through with
the process
                        {
                                double b[3][1] = { {j + xmin}, {i + ymin}, {1} };
                                Mat B(3, 1, CV_64FC1, b);
                                Mat X(3, 1, CV_64FC1);
                                solve(A, B, X, DECOMP_SVD);//solve for coordinates of original image X
= A`B
                                double x = (X.at<double>(0, 0)) / (X.at<double>(2, 0));
                                double y = (X.at<double>(1, 0)) / (X.at<double>(2, 0));
                                if (x < img.cols&&y < img.rows&&x>0 && y>0)//check whether computed
coordinated are within bound
                                {
                                        for (int c = 0; c < 3; c++)
                                        {
                                                res.at<Vec3b>(i, j)[c] = img.at<Vec3b>(y, x)[c];//copy
pixel values to canvas

                                        }
                                }
                        }
                }
        }
        String result = "result";
        namedWindow(result, WINDOW_NORMAL);
        imshow(result, res);
        imwrite("exp1.jpg", res);//store the stitched image
        printf("\n\nDONE");


}

int main()
{
        Mat img;
        Mat ref;
        img = imread("1.jpg");
        ref = imread("4.jpg");
```

```
        String srcimg = "source";
        namedWindow(srcimg, WINDOW_NORMAL);
        String dstimg = "destination";
        namedWindow(dstimg, WINDOW_NORMAL);
        setMouseCallback(srcimg, source, NULL);
        imshow(srcimg, img);
        setMouseCallback(dstimg, destination, NULL);
        imshow(dstimg, ref);
        waitKey(0);
        Compute(img, ref);//compute mosaicing
        waitKey(0);
        return 0;
}
```

Result



Conclusion/Observation

- For every pair of 4 corresponding/common points selected, calculated homography matrix comes out to be different

- OpenCV C++ function ' *SVDecomp(H, w, u, vt)'* gives different homography matrix than derived from '*findHomography()*' , hence transformation results are different as well as odd if we use 'SVDecomp()', unfortunately there is no other way to perform SVD decomposition in OpenCV C++.
- Following forum links also discuss same issue regarding SVDecomp() in C++ version of OpenCV
  http://answers.opencv.org/question/32932/svd-with-different-solution-as-wolframalpha/
  https://stackoverflow.com/questions/16053380/svd-computing-different-result-in-matlab-and-opencv?rq=1
  https://stackoverflow.com/questions/23707755/opencv-svd-returns-different-result-than-matlab?rq=1
- For the most accurate transformed result , choose common points such that they lie inside blackboard but are as far apart as they can be.
- Final Stitched image still have some geometric distortion,discontinuities and black/pepper noise around certain areas.These imperfection are caused due to erroneously calculated homography matrix, rounding off decimal values etc.
- Uneven exposure which is caused due to changing lighting conditions, automatic controls of cameras, printing/scanning devices, etc. can be removed by histogram equalization.

Motivation

We have learn how to perform projective transformation of an image using homography matrix, using it we have assembled series of image which were overlapping snapshots of a common view and then produced a continuous seamless photographic representation or simply panorama. We have been able to create a program which enables a user to take images using a hand held camera and then take images at different angles with some overlapping area and then simply congregate a mosaic.This project has provided us with great insights behind working and solving of real world problems and thus has been a good learning experience for us as students.