Report: Project- Mosaicing

Submitted By- Ujjawal Agarwal

16ucs203

Introduction:

This report is about image stitching. In this work, we formulate stitching as a multi-image matching problem, and use manually controlled feature points to find matches between all the images.

Requirements:

Files

This project requires files for matching points explicitly named “new.txt” and “old.txt” for matching points. Input in this file is required to be in <x y> format.

Images

This project requires input images named “m0.jpg”… “m1.jpg” and upto 6 images is hard-coded in the code.

Implementation: Functions used-

getPoints:

This function takes a file as an input and return vector (array) of all the points in <Point2d> format.

computeHomography:

This function computes Homography matrix using two set of points using svd.

getCoordinates:

This function computes points <x',y'> on the transformed image from points <x,y>

getLimits:

This function computes the range of the image for offset matching.

projectImage:

This function computes the warped image using inverse transformation and finally translates it to match with the final panorama image.

stitch:

This function fills the panorama with different image and blend them together.

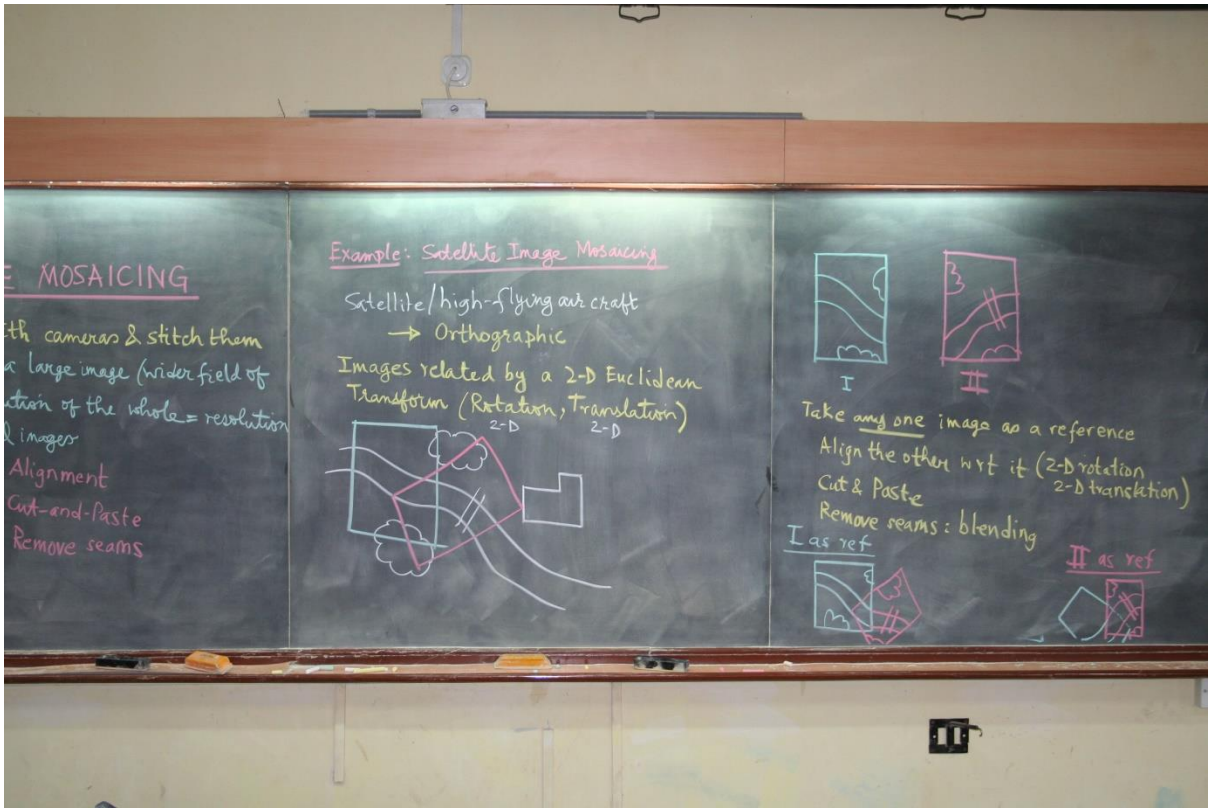
Implementation: Algorithm

In our code, we first compute the Homography matrix and based on this matrix we first find the corner limits for all images. Based on these limits, we conclude size of our final mosaic image.

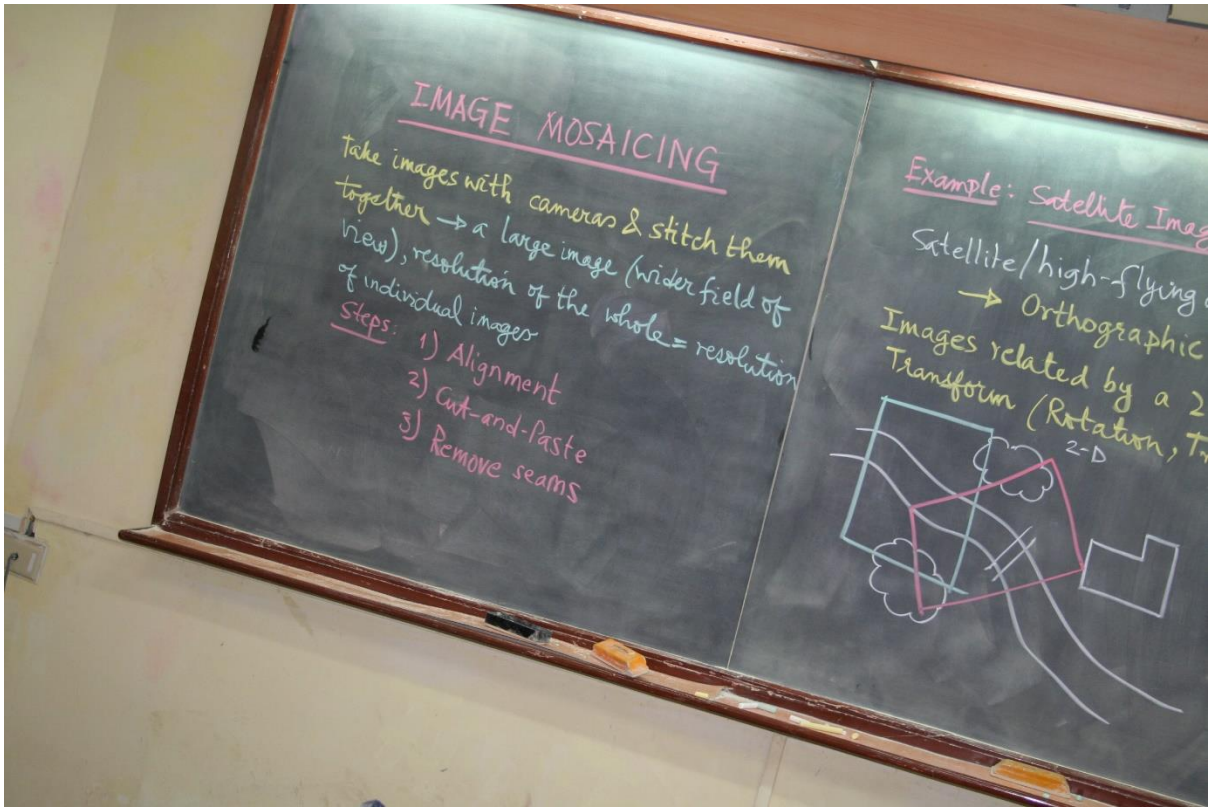
Using Inverse transform, we transform our image to same plane of reference image, and then translate it using offset calculated in above step. Since it's the minimum and maximum bound for all the images, no image can go outside this range, and we finally obtain warped images.

Finally, we blend these images one by one to compute mosaic image.

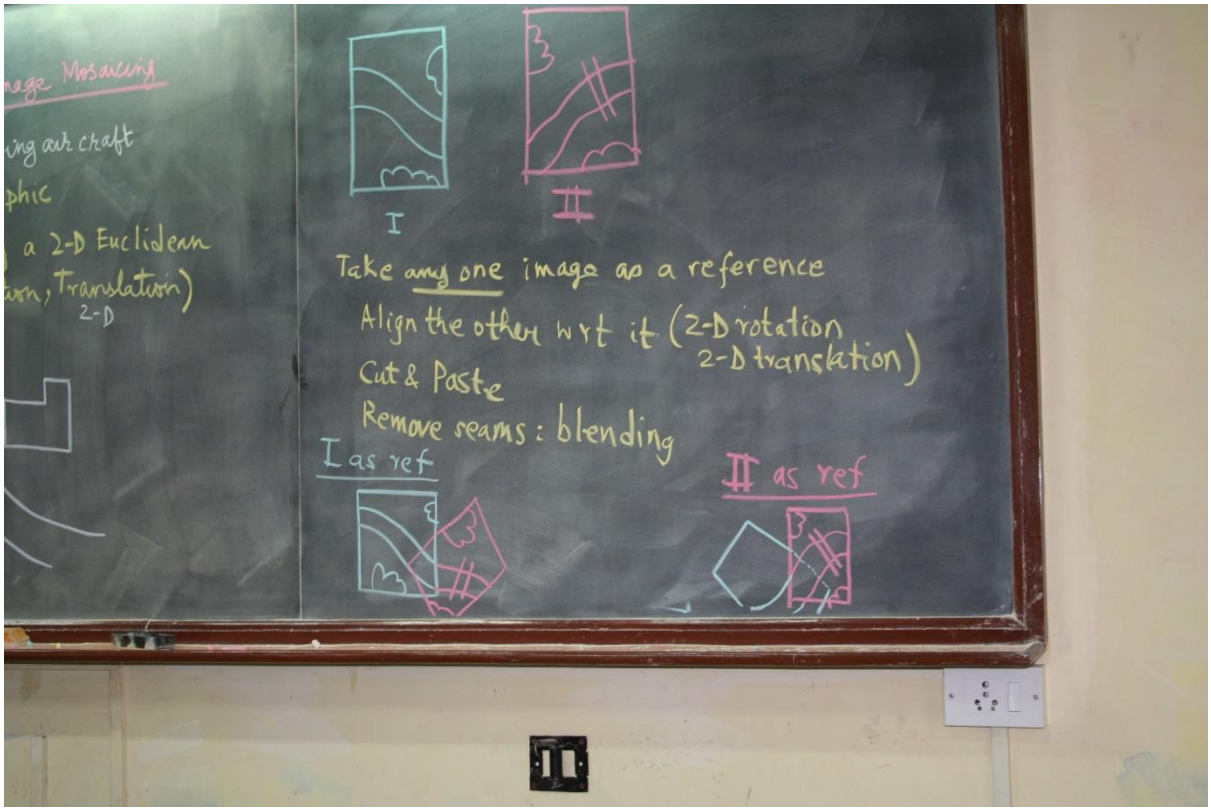
Input Image:



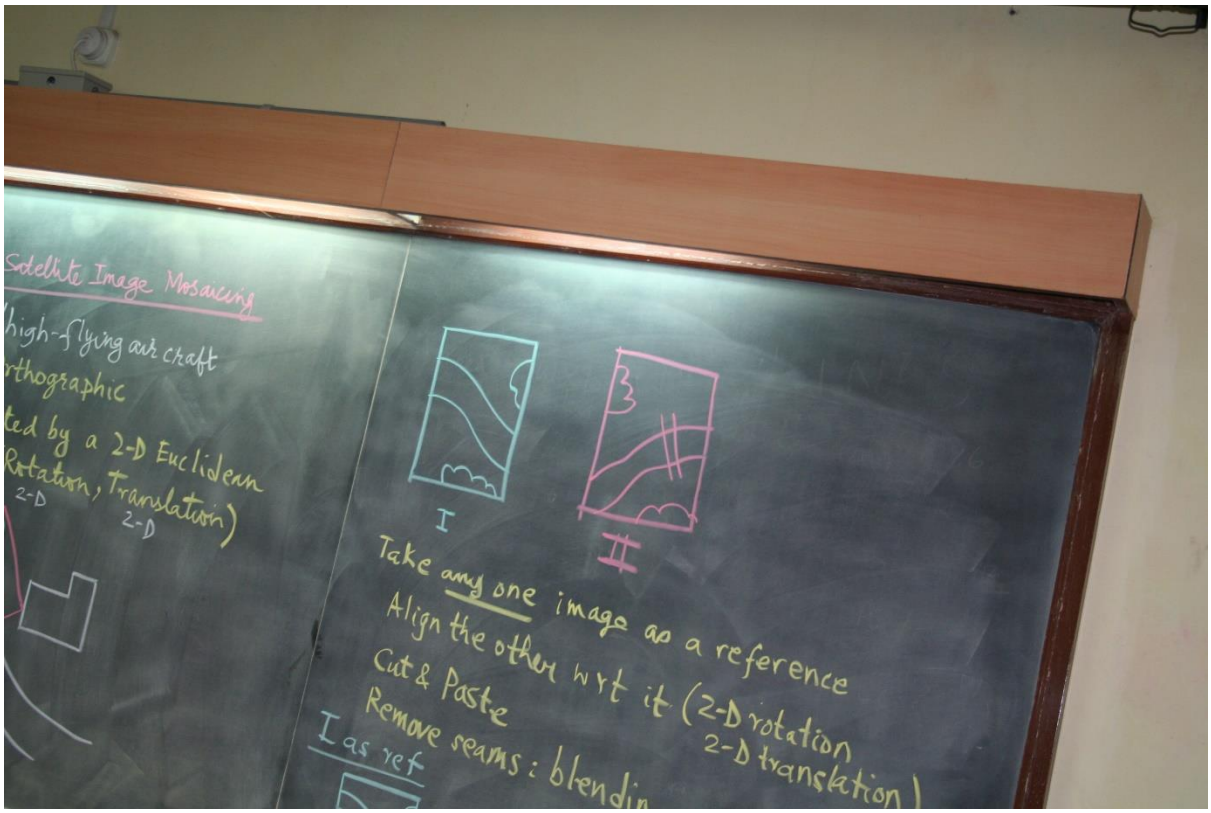
mref.jpg



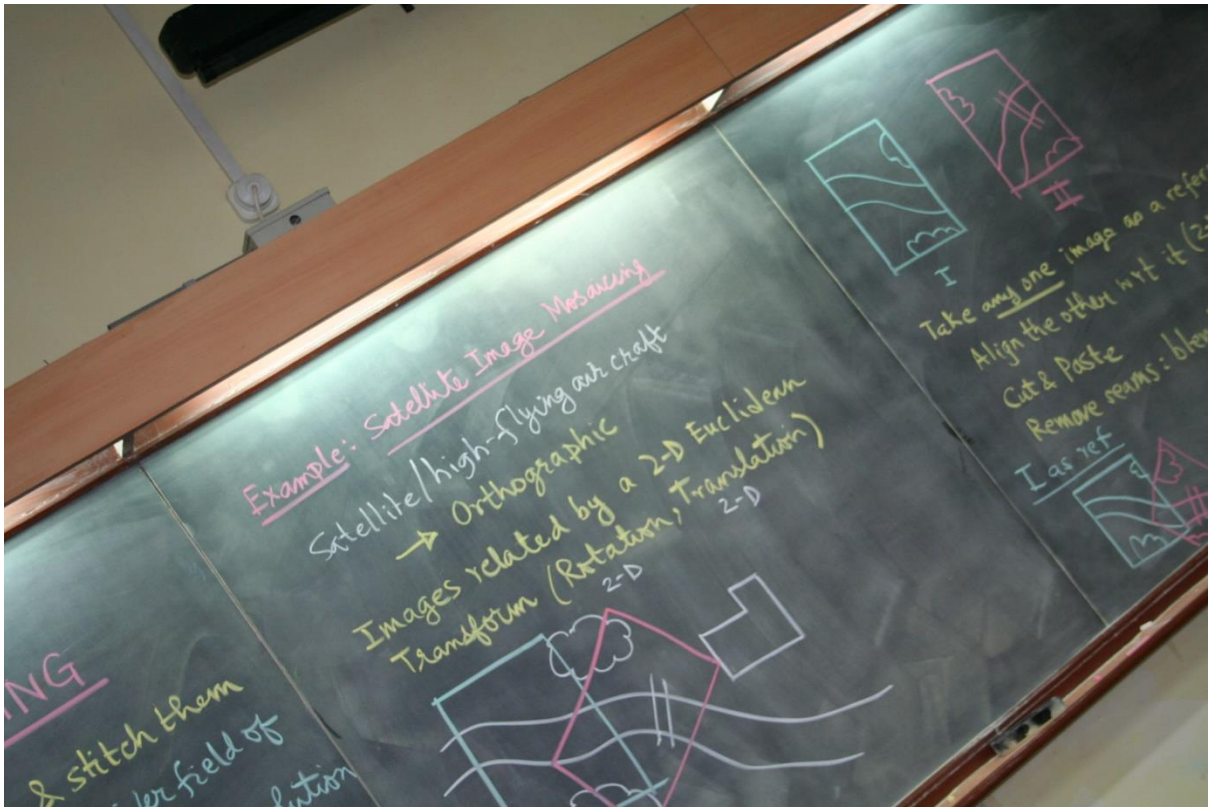
m1.jpg


I
II
Take any one image as a reference
Align the other wrt it (2-D rotation
2-D translation)
Cut & Paste
Remove seams : blending
I as ref
II as ref

m2.jpg


I
II
Take any one image as a reference
Align the other wrt it (2-D rotation
2-D translation)
Cut & Paste
I as ref

m3.jpg


Example: Satellite Image Mosaicing
Satellite/high-flying aircraft
→ Orthographic
Images related by a 2-D Euclidean
Transform (Rotation, Translation)
2-D
2-D
Take any one image as a refer
Align the other wrt it (2
Cut & Paste
Remove seams: blen
Last ref

m4.jpg

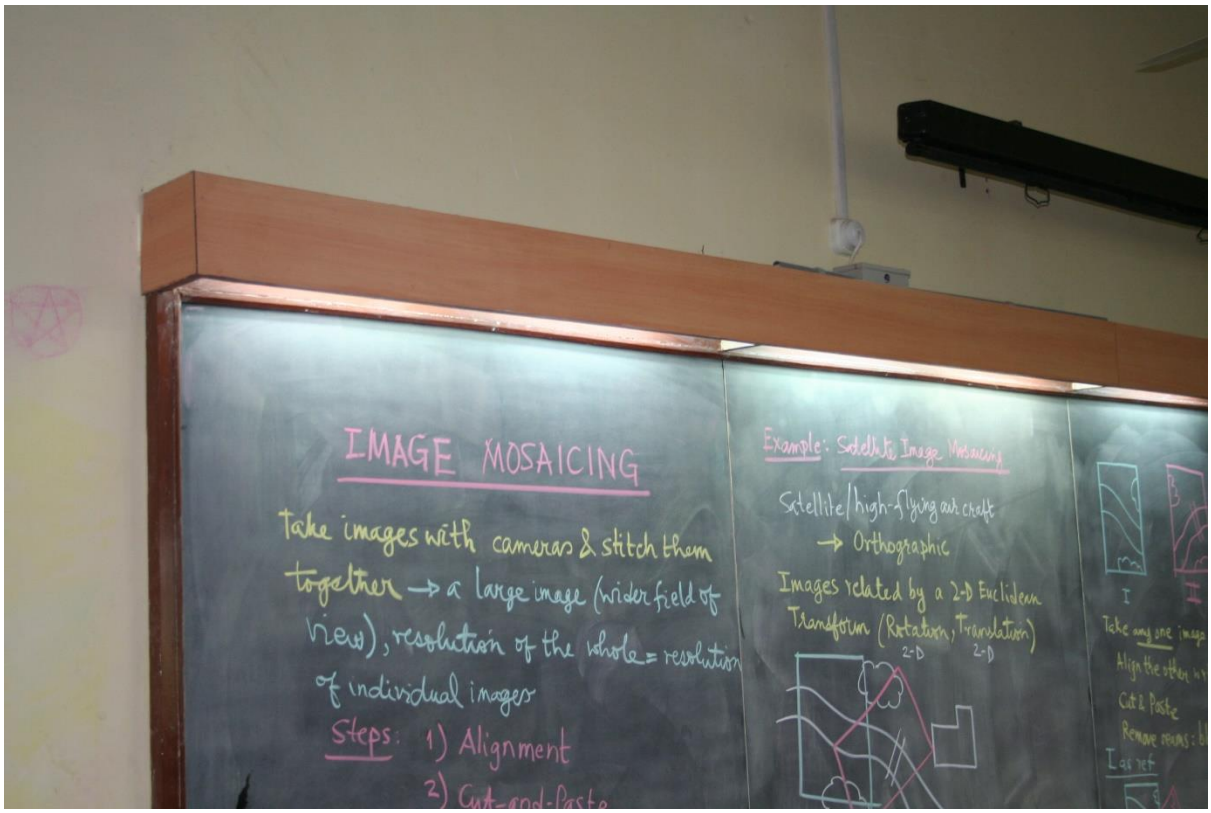
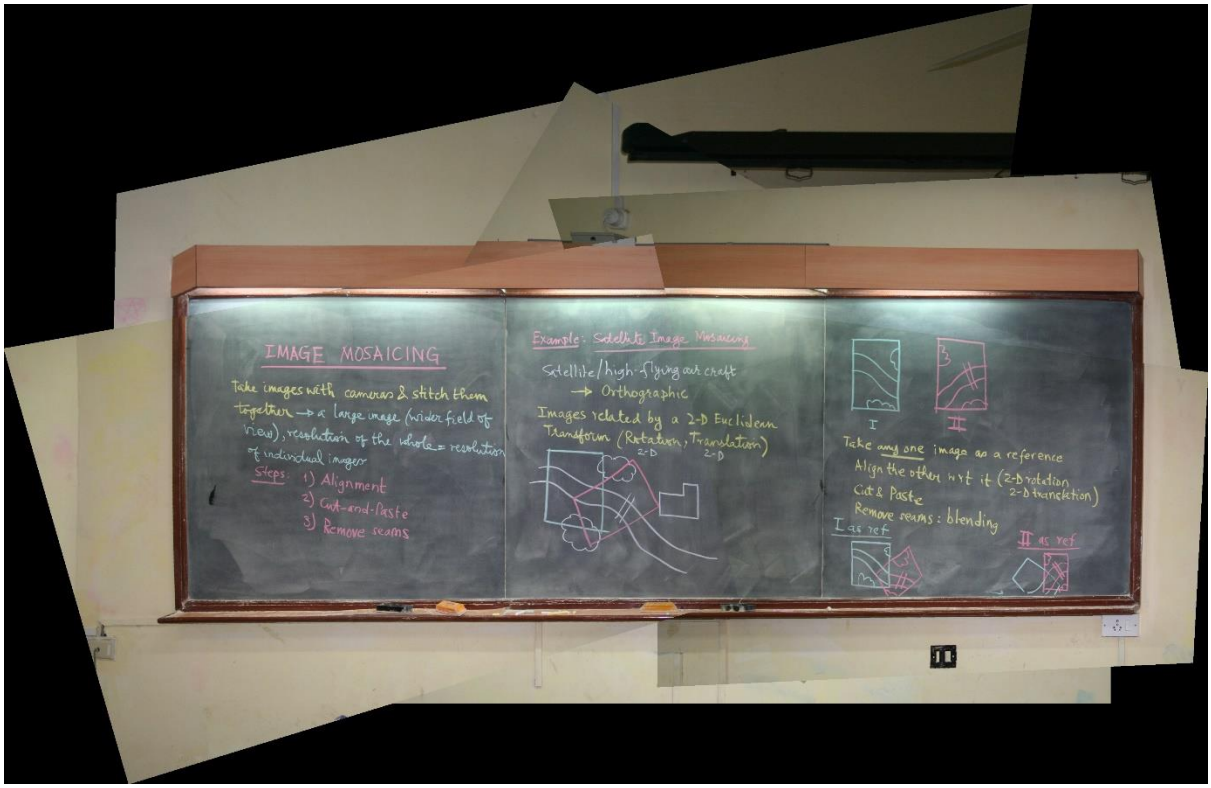


IMAGE MOSAICING
Take images with cameras & stitch them
together → a large image (wider field of
view), resolution of the whole = resolution
of individual images
Steps: 1) Alignment
2) Cut-and-Paste
Example: Satellite Image Mosaicing
Satellite/high-flying aircraft
→ Orthographic
Images related by a 2-D Euclidean
Transform (Rotation, Translation)
2-D
2-D
I
Take any one image
Align the other wrt
Cut & Paste
Remove seams: bl
Last ref

m5.jpg

Output Image:



mosiac.jpg

Source Code:

```
//compilation command: g++ -std=c++11 -g mosaicing.cpp -o mosaicking.out -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_imgcodecs
//execution command: ./mos.out


#include <stdio.h>
#include <iostream>
#include <algorithm>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/imgcodecs/imgcodecs.hpp"
#include "opencv2/opencv.hpp"

//number of pictures
#define nop 6

using namespace cv;
```

```
using namespace std;

//get points from files
vector<Point2d> getPoints(string filename){
        vector<Point2d> points;
        ifstream file(filename);

        double x,y;

        while(file>>x>>y){
                Point2d dot(x,y);
                points.push_back(dot);
        }

        return points;
}

//computing Homography matrix from points based on svd
Mat computeHomography(const vector<Point2d> &points1, const vector<Point2d>
&points2){

        int numofPoints= min(points1.size(),points2.size());

        Mat A= Mat(numofPoints*2,9,CV_64FC1);

        for(int i=0; i<numofPoints; i++){

                double data1[9]= {-1*points1[i].x,-1*points1[i].y,-
1,0,0,0,points1[i].x*points2[i].x,points1[i].y*points2[i].x,points2[i].x};
                double data2[9]= {0,0,0,-1*points1[i].x,-1*points1[i].y,-
1,points1[i].x*points2[i].y,points1[i].y*points2[i].y,points2[i].y};

                for(int j=0;j<A.cols;j++){
                        A.at<double>(2*i,j)= data1[j];
                        A.at<double>(2*i+1,j)= data2[j];
                }

        }

        Mat S,U,V;
        SVD::compute(A,S,U,V);
        V = V.row(V.rows-1);

        Mat H= V.clone();
        H= H.reshape(1,3);
        H=H/H.at<double>(2,2);
        return H;
}
```

```
//get coordinates after applying transformation on given points
Point2i getCoordinates(int x, int y, const Mat &H){
        double pt[3]= {(double)x, (double)y, 1.0};
        Mat A(3,1,CV_64FC1,pt);
        Mat B= H*A;

        Point2i
dot((int)(B.at<double>(0)/B.at<double>(2)),(int)(B.at<double>(1)/B.at<doubl
e>(2)));
        return dot;
}

//compare two points based on x (row)
bool compareX(const Point2d &a, const Point2d &b){
        return a.x<b.x;
}

//compare two points based on y (column)
bool compareY(const Point2d &a, const Point2d &b){
        return a.y<b.y;
}

//get corners for image after transformation
void getLimits(const Mat &img, const Mat &Hmat, int* limits){
        Point2i corners[4];
        corners[0]= getCoordinates(0,0,Hmat);
        corners[1]= getCoordinates(img.rows-1,0,Hmat);
        corners[2]= getCoordinates(img.rows-1,img.cols-1,Hmat);
        corners[3]= getCoordinates(0,img.cols-1,Hmat);

        int minrow= min_element(corners,corners+4,compareX)->x;
        int maxrow= max_element(corners,corners+4,compareX)->x;
        int mincol= min_element(corners,corners+4,compareY)->y;
        int maxcol= max_element(corners,corners+4,compareY)->y;

        limits[0]= min(limits[0],minrow);
        limits[1]= max(limits[1],maxrow);
        limits[2]= min(limits[2],mincol);
        limits[3]= max(limits[3],maxcol);
}

//warping image based on transformation matrix and then translating to fit
in final panorama
Mat projectImage(const Mat &img, const Mat &Hmat, int limits[]){
        Mat H= Hmat.inv();
        Mat rslt(limits[1]-limits[0]+1, limits[3]-limits[2]+1, CV_8UC3,
Scalar(0,0,0));

        for(int i=limits[0];i<=limits[1];i++){
```

```
                for(int j=limits[2];j<=limits[3];j++){
                        Point2i dot= getCoordinates(i,j,H);
                        if(dot.x>=0 && dot.x<img.rows && dot.y>=0 &&
dot.y<img.cols)
                                rslt.at<Vec3b>(i-limits[0],j-limits[2])=
img.at<Vec3b>(dot.x,dot.y);
                }
        }

        return rslt;
}

//checking if pixel color is black
bool isnotblack(Vec3b pixel){
        return pixel[0]==0 && pixel[1]==0 && pixel[2]==0?false:true;
}

//stitching two images
void stitch(Mat &mosaic, const Mat &img){

        for(int i=0;i<mosaic.rows;i++)
                for(int j=0;j<mosaic.cols;j++)
                        if(isnotblack(img.at<Vec3b>(i,j)))
                                mosaic.at<Vec3b>(i,j)= img.at<Vec3b>(i,j);
}

int main(int argc, const char** argv){

        //initial datasets for input and output images with homoraphy
        Mat imgin[nop], Homography[nop], imgout[nop];

        //reading reference image
        imgin[0]= imread("mref.jpg",CV_LOAD_IMAGE_UNCHANGED);
        assert(&imgin[0]);

        //identity matrix for reference image
        Homography[0]= Mat::eye(3,3,CV_64FC1);

        //reading input images and files and computing homography matrix
        for(int i=1;i<nop;i++){
                string imagename= "m" + to_string(i) + ".jpg";
                imgin[i]= imread(imagename,CV_LOAD_IMAGE_UNCHANGED);
                assert(&imgin[i]);

                string oldFile= "old" + to_string(i) + ".txt";
                vector<Point2d> oldpoints= getPoints(oldFile);

                string newFile= "new" + to_string(i) + ".txt";
                vector<Point2d> newpoints= getPoints(newFile);
```

```
                Homography[i]= computeHomography(oldpoints,newpoints);
        }

        //panorama size
        int limits[4]= {INT_MAX,INT_MIN,INT_MAX,INT_MIN};
        for(int i=0;i<nop;i++)
                getLimits(imgin[i],Homography[i],limits);

        namedWindow("Image",CV_WINDOW_NORMAL);
        resizeWindow("Image",1600,800);

        //warping image and saving
        for(int i=0;i<nop;i++){
                imgout[i]= projectImage(imgin[i],Homography[i],limits);
                string imagename= "mnew" + to_string(i) + ".jpg";
                imwrite(imagename,imgout[i]);
        }

        //final image
        Mat mosaic(limits[1]-limits[0]+1, limits[3]-limits[2]+1, CV_8UC3,
Scalar(0,0,0));

        //stitching all the images and saving
        for(int i=0;i<nop;i++){
                stitch(mosaic,imgout[i]);
                imshow("Image",mosaic);
                waitKey(0);
                string imagename= "mosaic" + to_string(i) + ".jpg";
                imwrite(imagename,mosaic);
        }

        destroyAllWindows();
        return 0;
}
```