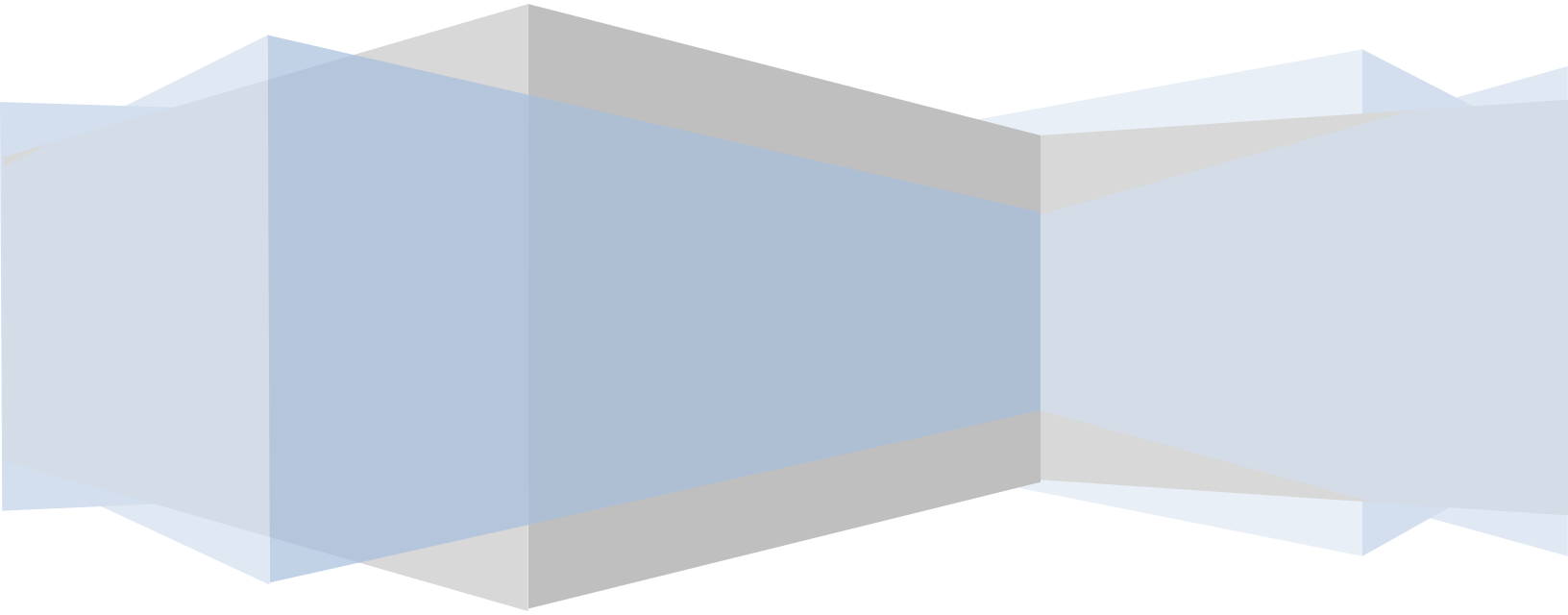
International Institute of Information Technology, Hyderabad

Assignment 2

Mosaicing and Rectification

Mohit Sharma

201505508



DLT (Direct Linear Transformation) based Homography between two images

Technologies used:

1. OpenCV for image operations.
2. Python for homography calculations.
3. MATLAB for SIFT.
   No other in built library call is used.

Implementation steps:

1. Noted 4 same corner points from two different images with some overlap .
2. Created matrix A using these points and ran SVD on the same to find out projection matrix. Code for the same is attached at the end.

Image1 used for DLT, 4 corner points are marked in red

Image2 used for DLT

Results:

Homography matrix:

[[3.74593677e-03 1.01456967e-03 9.99167810e-01]
 [-4.89964652e-04 4.82531104e-03 4.00600116e-02]
 [-1.65324985e-06 1.72202497e-06 4.50535846e-03]]

Stitched Image:

Observations:

Image is not stitched properly because of following reasons:

1. Image is not planar and camera center is not exactly fixed.
2. Human error in noting down the common points.
3. While mapping the pixels from image2 to image1, float values are encountered which are rounded off to integers.

Ransac based method for estimating homography:

Implementation steps:
Image coordinates for 18 common points are taken from both the images and followed these steps to calibrate:

1. Randomly select 4 pair of points out of 18.
2. Find H matrix for above selected points using DLT.
3. Using this H matrix, Find out image1 coordinates (xi, yi) for all the 18 points.
4. Calculate the error in terms of Euclidean distance from calculated values (xi, yi) to actual values.
5. If this error is lesser than the threshold (taken as 0.1) then return P matrix as the final result.
6. Repeat above 5 steps until we reach threshold.

Same images as shown above are used for RANSAC too.

Results:

Homography matrix:

```
[[-5.07645106e-04   1.11822860e-03  -9.94672411e-01]
[3.32904069e-04  -1.66414422e-03  -1.03023751e-01]
[2.45570295e-06   2.27950479e-06  -2.91764230e-03]]
```

Stitched Image:

Observations:

As we can see that stitching of images is better in this case because of following reasons Image is not planar and camera center is not exactly fixed.

1. Because of increased no. of points chances of human error have decreased.
2. But as we can see there are some gaps between the pixels because of rounding off.

SIFT implementation to find matching points and calculate homography using those points

Used MATLAB's SURF package for finding the common points between two images as SIFT had some installation issues. Found 18 different points and ran the same RANSAC method to compute homography as mentioned above. Code for the same is at the end.
Same images as shown above are used for RANSAC too.

Results:

Homography matrix:

```
[[-0.0024   -0.0001   -0.9966]
 [0.0004   -0.0033   -0.0820]
 [0.0000   -0.0000   -0.0037]]
```

Stitched Image:

Observations:

As we can see that stitching of images is best in this case because of no human interaction for noting the common points. The remaining error in stitching is just because of non planar world with camera center not fixed accurately.

Panorama image

Used the same code iteratively to stitch all the five images one by one with common points detection using SURF.

Perspective distortion correction:

Downloaded following 3 images with a number plate in rotated position. For correcting the perspective distortion, calculated homography between 4 corners of number plate to 4 corners of a rectangle.

$$Im = H * Ir$$

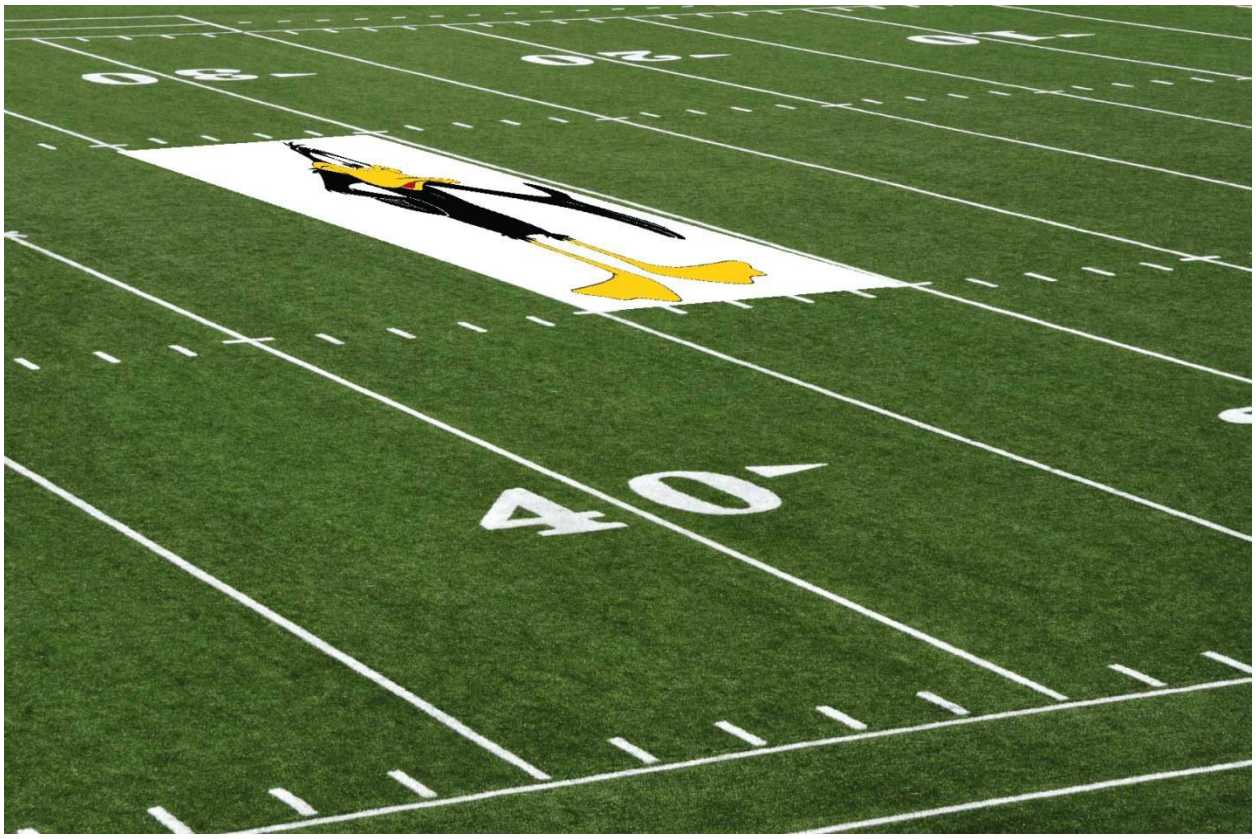
Here Im is the actual image while Ir represents the rectangle. Now Multiplication of Ir's points with H will give us a point of Im. This resultant point is copied to rectangle. eg: if (x,y) is rectangle's point which gives (u,v) after multiplying with H, then copy the intensity value at u,v to x,y in rectangle. Corresponding perspective distortion correction is also shown in following images.


0007

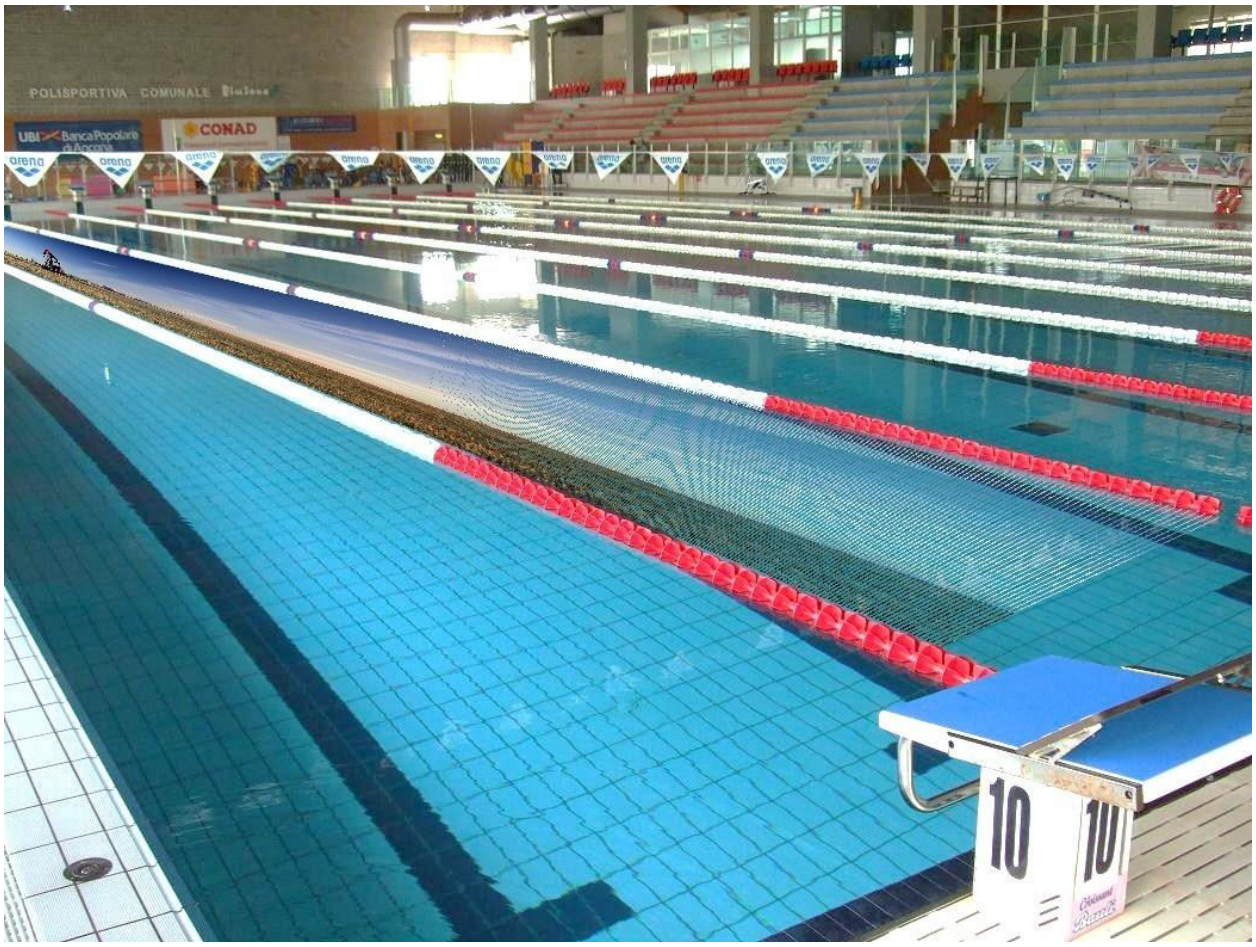

0007

Graphic image overlay over the sports image:

Used the similar idea of homography as mentioned in perspective distortion correction. In this case graphic image or we can think of rectangle in above case is copied on the sports image i.e. we introduced perspective distortion in the graphic image. Please check the code at the end. Following are the images generated as a result of code:



Daffy duck in the field



Land or pool?

Tom and jerry on pool table

Codes

```
# program to implement DLT method to find homography between two
images

from numpy import *
import numpy as np
from scipy import linalg

A =[]
#x1 = [[304, 306], [347, 368], [432, 352], [300, 58]]
      # I1
#x2 = [[38, 322], [90,388], [188, 364], [39, 44]]
      # I2

#x1 = [[160,43], [165,269], [498,374],[511,103]]
      #perspective projection, p1
#x2 = [[0,0], [0,400], [600,400], [600,0]]

#x1 = [[250,750], [608,747], [843,100],[685,93]]
      #perspective projection, p2
#x2 = [[0,0], [0,400], [600,400], [400,6]]

#x1 = [[219,382], [230,461], [510,568],[518,486]]
      #perspective projection, p3
#x2 = [[0,0], [0,400], [600,400], [600,0]]

#x1 = [[494,177],  [154,200], [809,418], [1231, 382],]
      #perspective projection, green field
#x2 = [[498,0], [0,0], [0,504],[498,504]]

#x1 = [[0,186], [0,214], [790,539], [1026,431]]
      #perspective projection, pool
#x2 = [[0,0], [0,334],[1170,334], [1170,0]]

x1 = [[245,29], [79,76], [508,187], [639,105]]
      #perspective projection, pool table
x2 = [[1920,0], [0,0],       [0,1200],[1920,1200]]

def addPointToA(x1,x2):
      # I1 = H*I2
      l = [x2[0],x2[1],1,0,0,0,-x1[0]*x2[0], -x1[0]*x2[1], -x1[0]]
      A.append(l)
      l = [0,0,0,x2[0],x2[1],1,-x1[1]*x2[0], -x1[1]*x2[1], -x1[1]]
      A.append(l)

def prepareA():
      #prepare matrix A for selected points
      for i in range(len(x1)):
            addPointToA(x1[i], x2[i])
```

```
prepareA()

U,s,V = linalg.svd(A)

print 'Homography matrix is:'
H = V[-1].reshape((3,3))
print H
x2=[[0,0], [800,450]]
print 'Image 1 calculated using this homography matrix:'
for i in range(len(x2)):
      y = np.dot(H, x2[i] + [1])
      print 'Image 2 point: %s' % x2[i]
      print 'Image 1 point: %s' % x1[i]
      l = [y[0]/y[2], y[1]/y[2]]
      print 'Calculated image 1 point: %s' % l
      print
```

 # Program to implement RANSAC to to find homography between two images

```
from random import randint
from numpy import *
import numpy as np
from scipy import linalg
import math

A =[]

# 18 points from both the images
x1 = [[304, 306], [347, 368], [432, 352], [300, 58], [346, 381], [342,
393], [341, 406], [447, 360], [462, 367], [478,376], [712, 178],
[674,220], [715,124],[718, 69], [720, 11], [589, 214], [494, 93],
[594, 163]]
x2 = [[38, 322], [90,388], [188, 364], [39, 44], [88,400], [85,415],
[82,430], [206, 371], [222,377], [238, 385], [458,187],[426,226],
[459, 138], [461, 87],[462, 35],[351, 218], [258, 97], [355, 170]]

def addPointToA(x1,x2):
      # I1 = H*I2
      l = [x2[0],x2[1],1,0,0,0,-x1[0]*x2[0], -x1[0]*x2[1], -x1[0]]
      A.append(l)
      l = [0,0,0,x2[0],x2[1],1,-x1[1]*x2[0], -x1[1]*x2[1], -x1[1]]
      A.append(l)

def prepareA():
      d ={}
      for i in range(4):
            t = randint(0,17)
            while t in d:
                  t = randint(0,17)
            d[t] = 1
            addPointToA(x1[t], x2[t])

H = []

threshold = 0.1
def getTotalError():                              # return error in
calculated points using current homography matrix
      dist = 0
      for i in range(0,8,2):
            WP = [A[i][0], A[i][1]]
            y = np.dot(H, WP + [1])
            dist += (-1 * A[i][8] - y[0]/y[2]) * (-1 * A[i][8] -
y[0]/y[2]) + (-1 * A[i+1][8] - y[1]/y[2]) * (-1 * A[i+1][8] -
y[1]/y[2])
      return math.sqrt(dist)

while 1:
      A = []
```

```
      prepareA()
      U,s,V = linalg.svd(A)
      H = V[-1].reshape((3,3))
      print getTotalError()
      if(getTotalError() <= threshold):
            break;


print 'Homography matrix is:'
print H

print 'Image points calculated using this camera matrix:'
for i in range(6):
      y = np.dot(H, x2[i] + [1])
      print 'Image 2 point: %s' % x2[i]
      print 'Image 1 Image point: %s' % x1[i]
      l = [y[0]/y[2], y[1]/y[2]]
      print 'Calculated Image 1 point: %s' % l
      print
```

 # Program to implement ransac using matched points found out from SIFT/SURF to find homography between two images

```
from random import randint
from numpy import *
import numpy as np
from scipy import linalg
import math

A =[]

# 18 points from both the images
x1 = [[489, 50], [518, 203], [541, 95], [528, 49], [556, 126], [322,
209], [478, 154], [552, 122], [548, 117], [400,216], [555, 184],
[602,178], [617,16],  [493, 57], [532, 221], [483, 215], [388, 104],
[509, 96]]
x2 = [[253, 54], [283,208], [305, 102], [294, 56], [319,134],
[64,215], [241,159], [315, 130], [311,124], [156, 221],
[319,190],[363,186], [374, 31],   [257, 61],[296, 226],[247, 221],
[144, 102], [273, 101]]

def addPointToA(x1,x2):
      # I1 = H*I2
      l = [x2[0],x2[1],1,0,0,0,-x1[0]*x2[0], -x1[0]*x2[1], -x1[0]]
      A.append(l)
      l = [0,0,0,x2[0],x2[1],1,-x1[1]*x2[0], -x1[1]*x2[1], -x1[1]]
      A.append(l)

def prepareA():
      d ={}
      for i in range(4):
            t = randint(0,17)
            while t in d:
                  t = randint(0,17)
            d[t] = 1
            addPointToA(x1[t], x2[t])

H = []

threshold = 0.1
def getTotalError():                              # return error in
calculated points using current homography matrix
      dist = 0
      for i in range(0,8,2):
            WP = [A[i][0], A[i][1]]
            y = np.dot(H, WP + [1])
            dist += (-1 * A[i][8] - y[0]/y[2]) * (-1 * A[i][8] -
y[0]/y[2]) + (-1 * A[i+1][8] - y[1]/y[2]) * (-1 * A[i+1][8] -
y[1]/y[2])
      return math.sqrt(dist)
```

```
while 1:
      A = []
      prepareA()
      U,s,V = linalg.svd(A)
      H = V[-1].reshape((3,3))
      print getTotalError()
      if(getTotalError() <= threshold):
            break;


print 'Homography matrix is:'
print H

print 'Image points calculated using this camera matrix:'
for i in range(6):
      y = np.dot(H, x2[i] + [1])
      print 'Image 2 point: %s' % x2[i]
      print 'Image 1 Image point: %s' % x1[i]
      l = [y[0]/y[2], y[1]/y[2]]
      print 'Calculated Image 1 point: %s' % l
      print
```

// Program to stich two images with overlapping regions (Use homography matrices from DLT, RANSAC, RANSAC using SIFT points) and display

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

#define IMAGE1   "I1.jpg"
#define IMAGE2   "I2.jpg"

//double H1[] = {3.74593677e-03,   1.01456967e-03,   9.99167810e-01};
           //From DLT
//double H2[] = {-4.89964652e-04,   4.82531104e-03,   4.00600116e-02};
//double H3[] = {-1.65324985e-06,   1.72202497e-06,  4.50535846e-03};

//double H1[] = {-5.07645106e-04,   1.11822860e-03,  -9.94672411e-01};
           //From RANSAC
//double H2[] = {3.32904069e-04,  -1.66414422e-03,  -1.03023751e-01};
//double H3[] = {2.45570295e-06,   2.27950479e-06,  -2.91764230e-03};

double H1[] = {-0.0024,   -0.0001,   -0.9966};
     //From SURF
double H2[] = {0.0004,   -0.0033,   -0.0820};
double H3[] = {0.0000,   -0.0000,   -0.0037};



Mat A1 = Mat(3, 1, CV_32FC1, H1 );
Mat A2 = Mat(3, 1, CV_32FC1, H2 );
Mat A3 = Mat(3, 1, CV_32FC1, H3 );

double multiply(double d[], double A[], int l)
{
     double sum = 0;
     for(int i = 0; i<l; i++ )
     {
          sum += d[i] * A[i];
     }
     return sum;
}
void stichImages()                                        //Stich both
images
{
     Mat image1 = imread(IMAGE1, CV_LOAD_IMAGE_COLOR);
     Mat image2 = imread(IMAGE2, CV_LOAD_IMAGE_COLOR);
     double pt[3] ={0};
     pt[2] = 1;
     if(!image1.data || !image2.data)
```

```
	{
		cout<<"Failed to read the images.\n";
		return;
	}
	Size sz1 = image1.size();
	Size sz2 = image2.size();
	Mat result(sz1.height + sz2.height, sz1.width+sz2.width,
CV_8UC3);
	vector<int> compression_params;
	compression_params.push_back(CV_IMWRITE_JPEG_QUALITY);
	compression_params.push_back(100);
	for(int i = 0; i<image1.rows; i++)
	{
		for(int j = 0; j<image1.cols; j++)
		{
			result.at<Vec3b>(i, j) = image1.at<Vec3b>(i, j);
		}
	}
	for(int i = 0; i<image2.rows; i++)
	{
		for(int j = 0; j<image2.cols; j++)
		{
			pt[0] = j;
			pt[1] = i;
			Mat p = Mat( 3,1,  CV_32FC1, pt);
			double x = multiply(H1,pt,3);
			double y = multiply(H2,pt,3);
			double w = multiply(H3,pt,3);
			if( w == 0)
				w = 1;
			if(x/w>=0 && y/w>=0 && x/w<result.cols && y/w<
result.rows)
				result.at<Vec3b>(abs(y/w), abs(x/w)) =
image2.at<Vec3b>(i, j);

		}
	}
	if(!imwrite( "stiched.jpg", result,  compression_params))
	{
		cout<<"Failed to write the image\n";
	}
	namedWindow( "Stiched Image", WINDOW_AUTOSIZE );// Create a
window for display.
	imshow( "Stiched Image", result );
	waitKey(0);
}

int main(int argc, char* argv[])
{
	stichImages();
	return 0;
```

```
}
```

```
// Program to change perspective of images to a rectangle

#include "opencv2/highgui/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

#define IMAGE2   "p3.jpg"

//double H1[] = {-3.87592060e-03,  -1.92119386e-04,  -9.65700909e-01};
      //p1
//double H2[] = {-6.19051904e-04,  -3.60034531e-03,  -2.59532119e-01};
//double H3[] = {2.77954075e-06,  -7.07115167e-07,  -6.03563068e-03};

//double H1[] = {-3.91986198e-03,  -2.19890156e-03,  -3.16223569e-01};
      //p2
//double H2[] = {1.74720015e-03,  -1.30122945e-03,  -9.48670706e-01};
//double H3[] = {-3.73612270e-06,  -1.75464010e-06,  -1.26489427e-03};

double H1[] = {8.95217273e-04,   1.46564478e-04,   4.97359733e-01};
      //p3
double H2[] = {1.71737751e-04,   6.17119139e-04,   8.67540722e-01};
double H3[] = {-4.56606470e-07,   3.65698394e-07,   2.27104901e-03};


Mat A1 = Mat(3, 1, CV_32FC1, H1 );
Mat A2 = Mat(3, 1, CV_32FC1, H2 );
Mat A3 = Mat(3, 1, CV_32FC1, H3 );

double multiply(double d[], double A[], int l)
{
      double sum = 0;
      for(int i = 0; i<l; i++ )
      {
            sum += d[i] * A[i];
      }
      return sum;
}
void perspectiveImage()                                          //change
image persepective
{
      Mat image1(400, 600, CV_8UC3, Scalar(0));
      Mat image2 = imread(IMAGE2, CV_LOAD_IMAGE_COLOR);
      double pt[3] ={0};
      pt[2] = 1;
      if(!image1.data || !image2.data)
      {
            cout<<"Failed to read the images.\n";
            return;
      }
```

```
	vector<int> compression_params;
	compression_params.push_back(CV_IMWRITE_JPEG_QUALITY);
	compression_params.push_back(100);
	for(int i = 0; i<image1.rows; i++)
	{
		for(int j = 0; j<image1.cols; j++)
		{
			pt[0] = j;
			pt[1] = i;
			Mat p = Mat( 3,1,  CV_32FC1, pt);
			double x = multiply(H1,pt,3);
			double y = multiply(H2,pt,3);
			double w = multiply(H3,pt,3);
			if( w == 0)
				w = 1;
			if(x/w>=0 && y/w>=0 && x/w<image2.cols && y/w<
image2.rows)
				image1.at<Vec3b>(i,j) =
image2.at<Vec3b>(abs(y/w), abs(x/w));
		}
	}
	if(!imwrite( "stiched.jpg", image1,  compression_params))
	{
		cout<<"Failed to write the image\n";
	}
	namedWindow( "Persepective Image", WINDOW_AUTOSIZE );
	imshow( "Persepective Image", image1 );
	waitKey(0);
}

int main(int argc, char* argv[])
{
	perspectiveImage();
	return 0;
}
```

```
// Overlay a graphic image on sports image

#include "opencv2/highgui/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

#define IMAGE1   "b3.jpg"
#define IMAGE2   "t3.jpg"



//double H1[] = {2.79277865e-03,   3.66389838e-03,   6.10082058e-01};
      //green field
//double H2[] = { -1.51401163e-04,   9.46477090e-04,   7.92314361e-01
};
//double H3[] = { 1.78322017e-07,  -1.83507158e-06,   3.96157180e-03};

//double H1[] = { 6.43662982e-04,   5.95565215e-18,   0.00000000e+00};
      //pool
//double H2[] = { -5.84299389e-04,   5.91195960e-04,   9.99984995e-
01};
//double H3[] = {-3.96774510e-06,   6.56501131e-07,   5.37626341e-03};

double H1[] = {9.93976017e-04,   2.24413193e-03,   7.20625502e-01};
      //pool table
double H2[] = {-1.98992283e-04,   4.69429308e-04,   6.93259976e-01};
double H3[] = {8.38027667e-07,  -2.00182384e-06,   9.12184179e-03};

Mat A1 = Mat(3, 1, CV_32FC1, H1 );
Mat A2 = Mat(3, 1, CV_32FC1, H2 );
Mat A3 = Mat(3, 1, CV_32FC1, H3 );

double multiply(double d[], double A[], int l)
{
      double sum = 0;
      for(int i = 0; i<l; i++ )
      {
            sum += d[i] * A[i];
      }
      return sum;
}
void perspectiveImage()                                          //change
image persepective
{
      Mat image1 = imread(IMAGE1, CV_LOAD_IMAGE_COLOR);
      Mat image2 = imread(IMAGE2, CV_LOAD_IMAGE_COLOR);
      double pt[3] ={0};
      pt[2] = 1;
      if(!image1.data || !image2.data)
```

```
	{
		cout<<"Failed to read the images.\n";
		return;
	}
	Mat result(image1);
	vector<int> compression_params;
	compression_params.push_back(CV_IMWRITE_JPEG_QUALITY);
	compression_params.push_back(100);
	for(int i = 0; i<image2.rows; i++)
	{
		for(int j = 0; j<image2.cols; j++)
		{
			pt[0] = j;
			pt[1] = i;
			Mat p = Mat( 3,1,  CV_32FC1, pt);
			double x = multiply(H1,pt,3);
			double y = multiply(H2,pt,3);
			double w = multiply(H3,pt,3);
			if( w == 0)
				w = 1;
			if(x/w>=0 && y/w>=0 && x/w<result.cols && y/w<
result.rows)
				result.at<Vec3b>(abs(y/w), abs(x/w)) =
image2.at<Vec3b>(i, j);
		}
	}
	if(!imwrite( "stiched.jpg", result,  compression_params))
	{
		cout<<"Failed to write the image\n";
	}
	namedWindow( "Persepective Image", WINDOW_AUTOSIZE );
	imshow( "Persepective Image", result );
	waitKey(0);
}

int main(int argc, char* argv[])
{
	perspectiveImage();
	return 0;
}
```