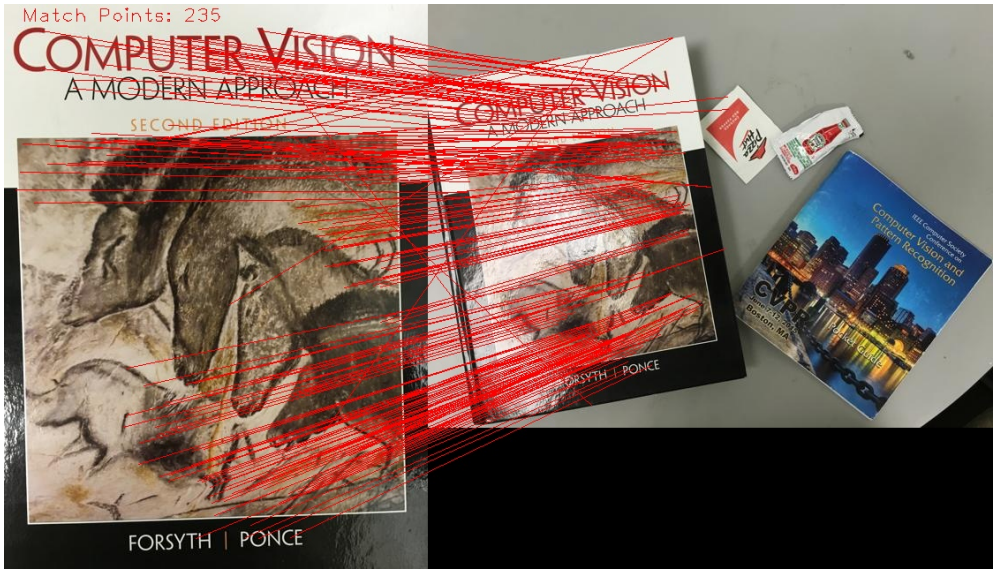
CSCI 574 HW3
Ahmet Can Ozbek

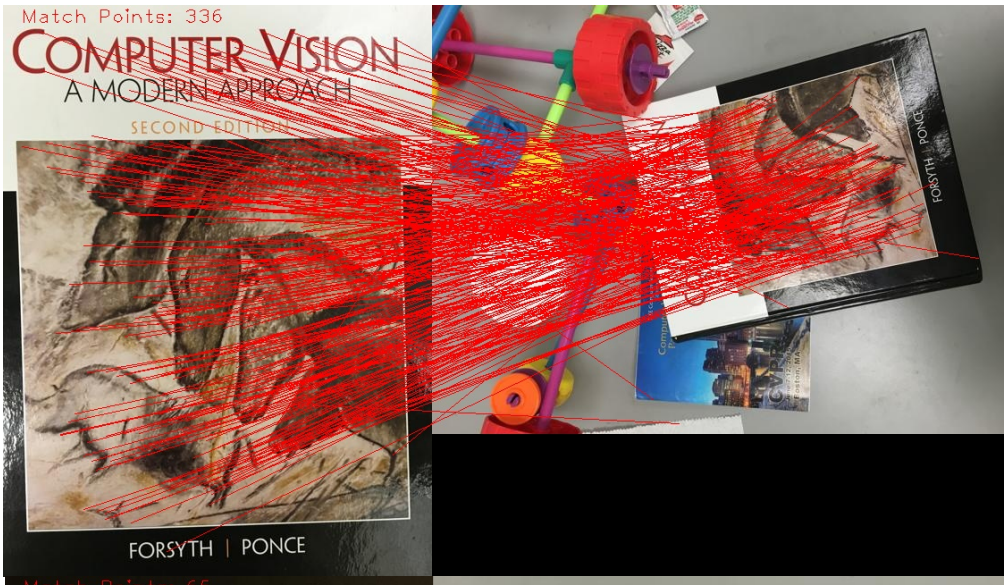
Step 1: SIFT Feature Extraction

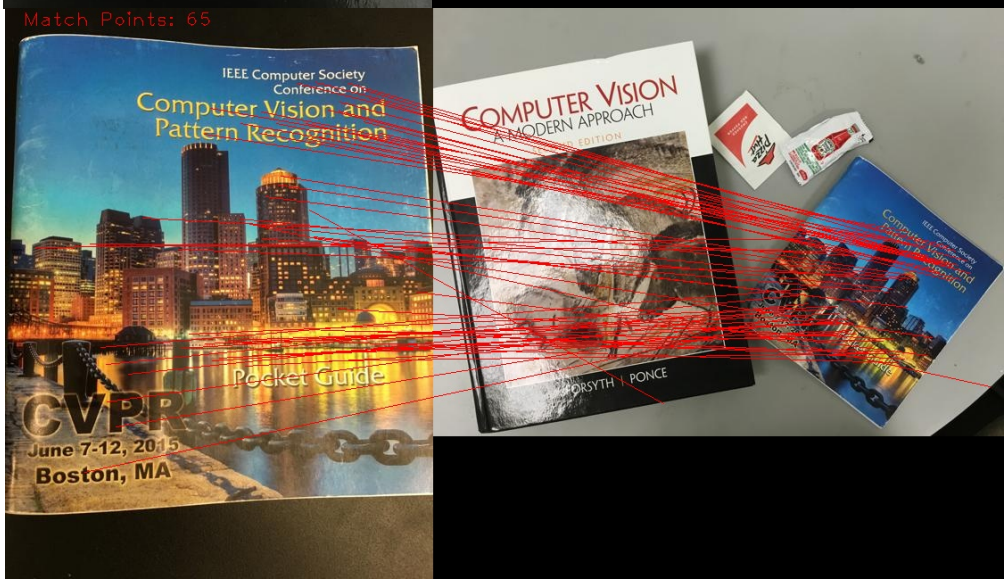
In this step our goal is to extract 128 dimensional SIFT features for each keypoint in the image. For this purpose, we are using OpenCV libraries related to SIFT.

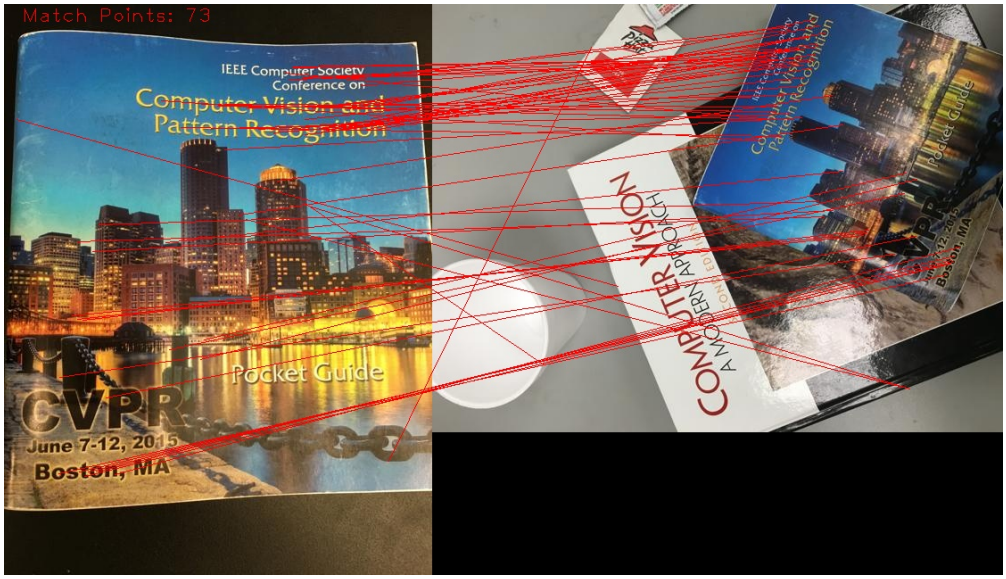
Step 2: Finding Matching Pairs

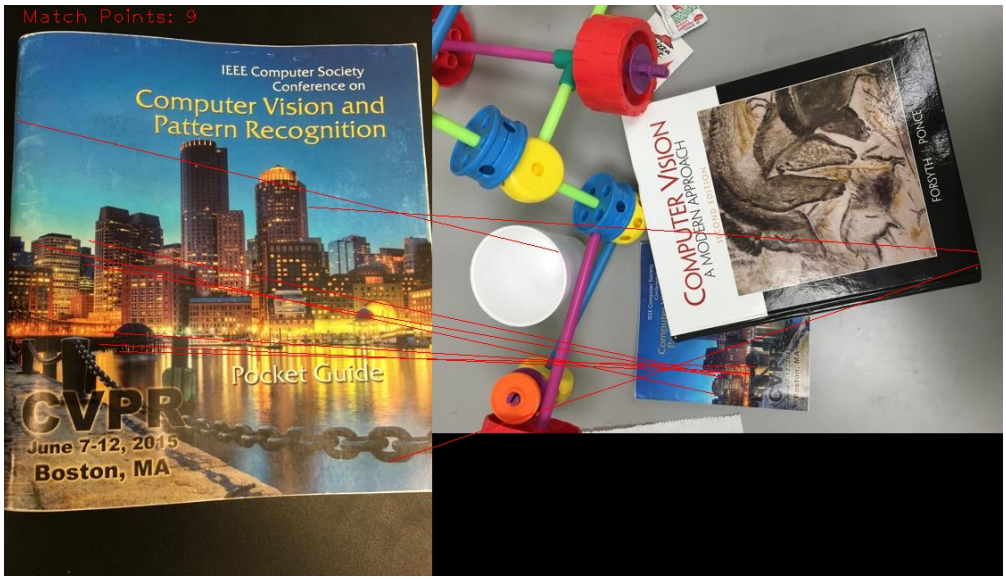
After extracting features of keypoints in both the object and the test(scene) images, we can start matching these keypoints by looking at the 128 dimensional feature vectors. In this question we are asked to use Euclidian distance metric to match relevant keypoints in object image and the test image. Here are the results of this procedure, matching keypoints are marked by red lines:




Match Points: 336
COMPUTER VISION
A MODERN APPROACH
SECOND EDITION
FORSYTH | PONCE


Match Points: 65
IEEE Computer Society
Conference on
Computer Vision and
Pattern Recognition
CVPR
Pocket Guide
June 7-12, 2015
Boston, MA


Match Points: 73
IEEE Computer Society
Conference on
Computer Vision and
Pattern Recognition
CVPR
Pocket Guide
June 7-12, 2015
Boston, MA



In the results above, we can say that SIFT algorithm performs very well however there are some difference in number of match points in different situations. The number of match points are lower when the object of interest is occluded by some other object, this causes to have less number of matching points.

Furthermore, we are also asked to display the results by transforming SIFT feature points on the object to the coordinates of the test image and overlaying the two. This is done by the homography matrix H. Since we know which points in the object image correspond to which points in the test(scene images), we can calculate the H matrix. H matrix can be calculated by the help of the RANSAC algorithm. We can summarize RANSAC algorithm as follows: For a number of iterations a random sample of 4 matching points (from object image to test image) is selected and the homogrophy matrix is calculated using those 4 points. The remaining matching points then are classified as either inlier or outlier depending on the closeness by the calculated H matrix. After we go through all the iterations, the iteration that had the most number of inliers is selected and H is calculated again from those inlier points.

H is a 3x3 matrix that maps points in object image to test image to help us find the desired object, the real values of H matrix is calculated as:

```
H:
 [0.0992954082460203, 0.1300753035981206,     236.9658718969529;
 -0.2004765216581303,    -0.2009788702466251, 439.077996984904;
  0.0002685334494112789, -0.0005307715865735516, 1]
```

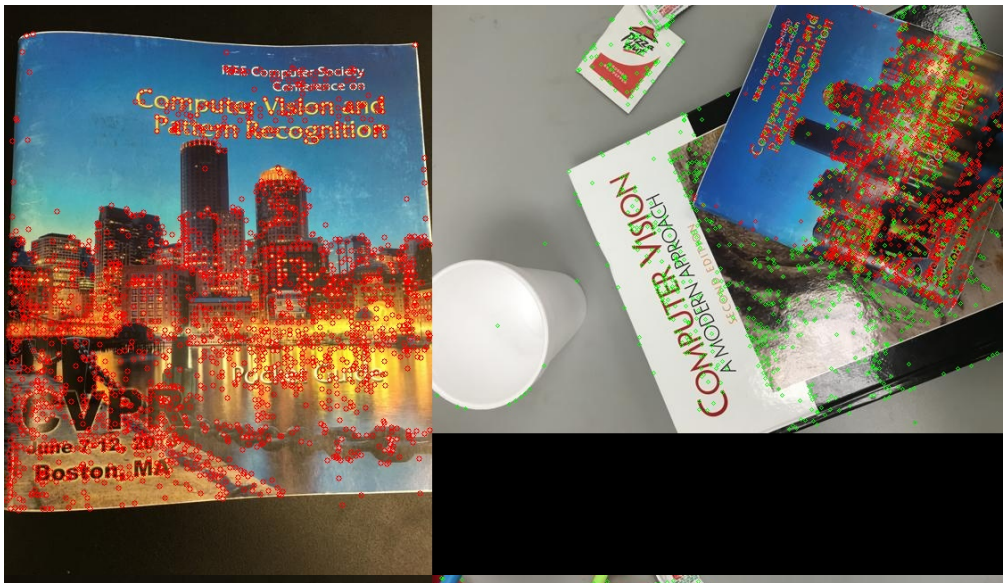
Now we will show the results of transforming SIFT feature points on the object image to the coordinates of the test images by the H matrix and the SIFT features obtained using only the SIFT algorithm on the test images. The points on the test image that are obtained by homography transformation are marked as red, the points that are obtained by SIFT algorithm on test images are marked as green. Here are the overlaid results:


COMPUTER VISION
A MODERN APPROACH
FORSYTH | PONCE


COMPUTER VISION
A MODERN APPROACH
FORSYTH | PONCE


COMPUTER VISION
A MODERN APPROACH
FORSYTH | PONCE

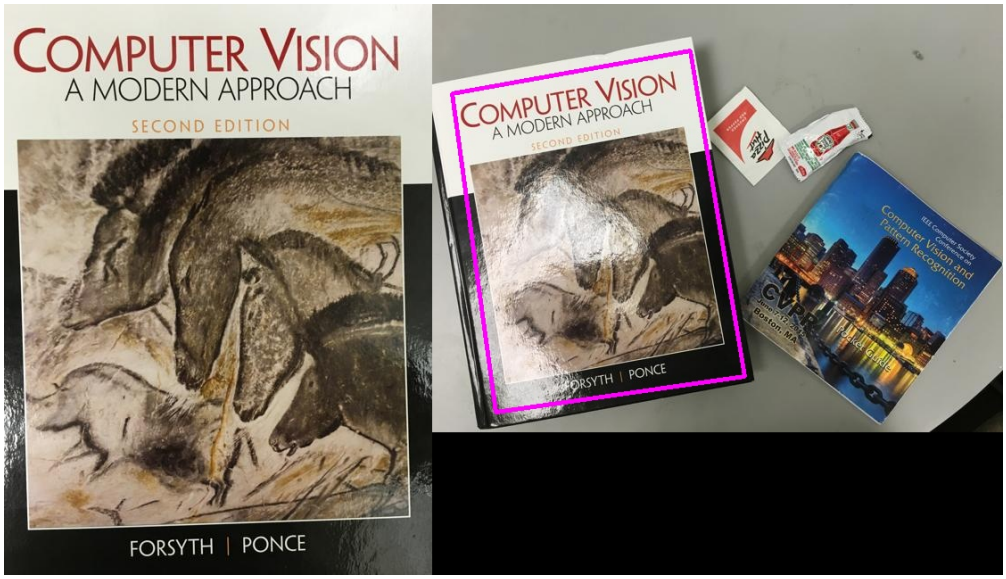

IEEE Computer Society
Conference on
Computer Vision and
Pattern Recognition
CVPR
June 7-12, 20
Boston, MA
COMPUTER VISION
A MODERN APPROACH
SECOND EDITION
FORSYTH | PONCE

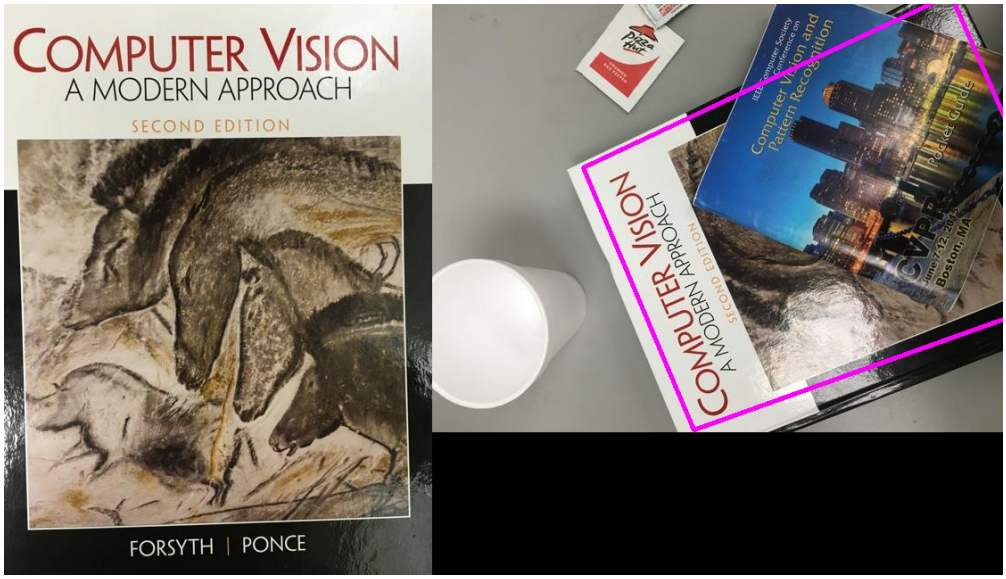

IEEE Computer Society
Conference on
Computer Vision and
Pattern Recognition
CVPR
June 7-12, 20
Boston, MA
COMPUTER VISION
A MODERN APPROACH


IEEE Computer Society
Conference on
Computer Vision and
Pattern Recognition
CVPR
June 7-12, 20
Boston, MA
COMPUTER VISION
A MODERN APPROACH
FORSYTH | PONCE

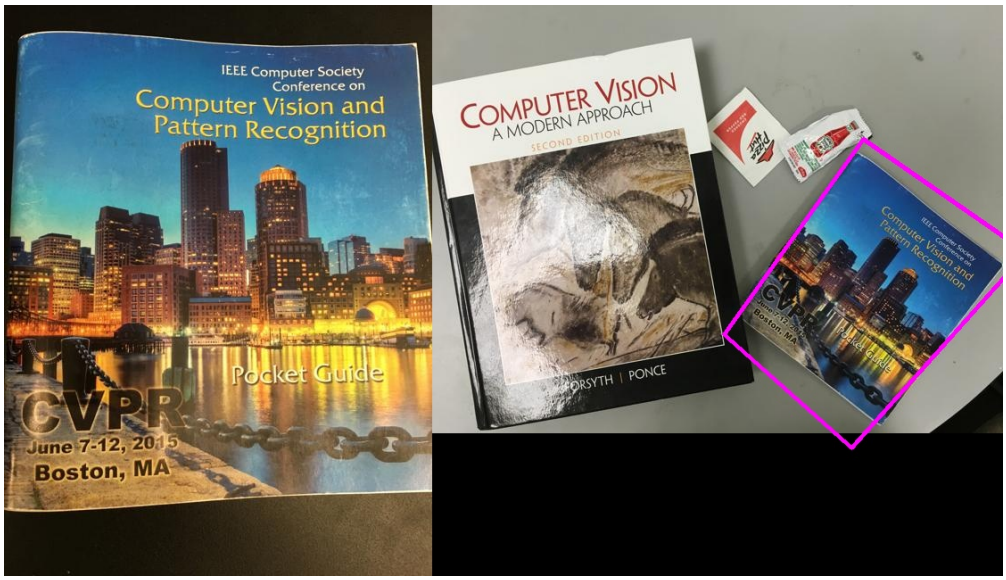
As we can see from the results above, green and red marks overlap on the objects of interest.

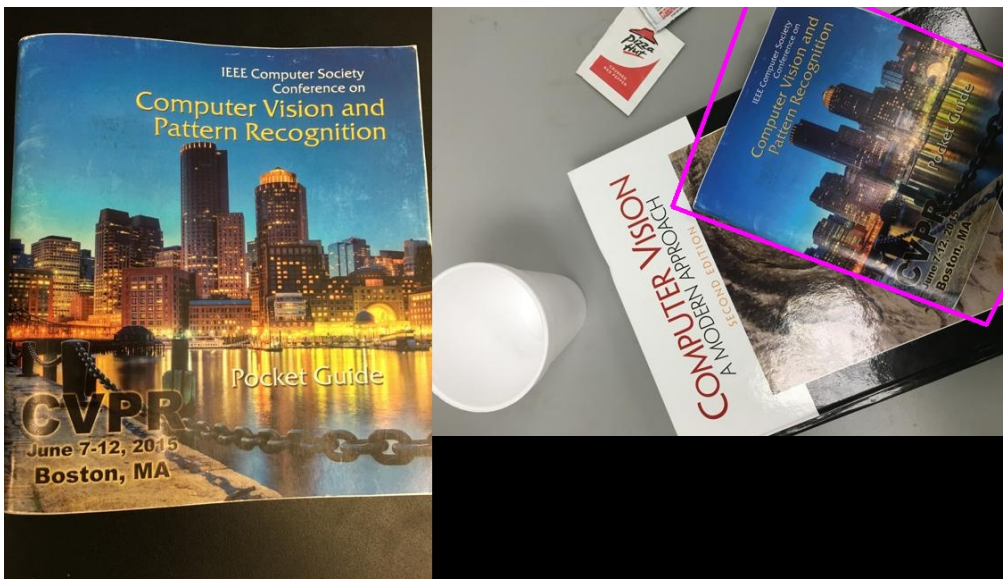
Moreover, as we know the four corners of the object, and as we calculated the homography matrix H, we can find the position and orientation of the object image in the test image. The following results show this concept:

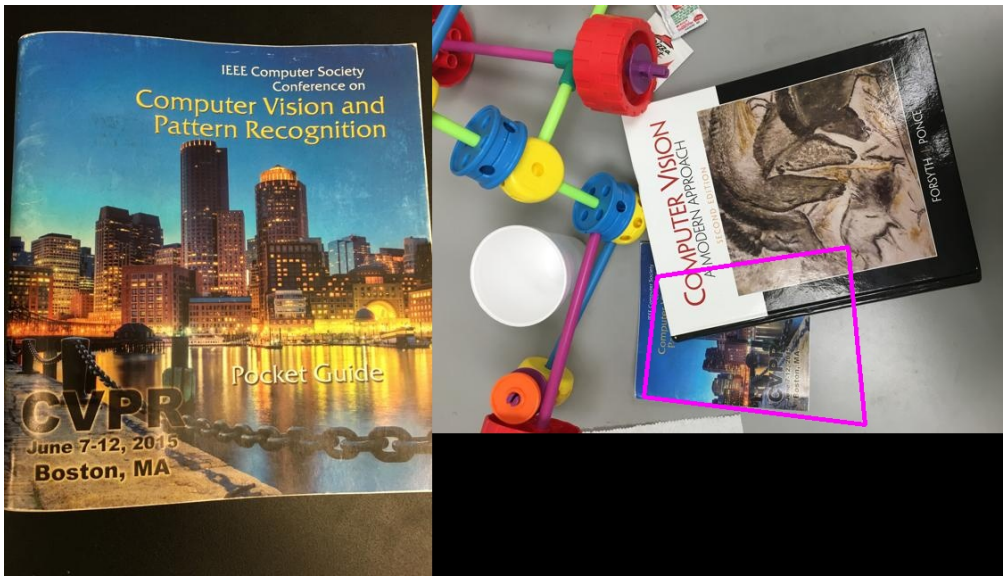





COMPUTER VISION
A MODERN APPROACH
SECOND EDITION
FORSYTH | PONCE


IEEE Computer Society
Conference on
Computer Vision and
Pattern Recognition
Pocket Guide
CVPR
June 7-12, 2015
Boston, MA


IEEE Computer Society
Conference on
Computer Vision and
Pattern Recognition
Pocket Guide
CVPR
June 7-12, 2015
Boston, MA



OpenCV C++ Source Code:

```
//CSCI574 HW3
//Ahmet Can Ozbek
//USC ID: 8401860152
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/nonfree/features2d.hpp>
#include "opencv2/calib3d/calib3d.hpp"
#include <string>

using namespace std;
using namespace cv;

int getMatchIndex(Mat featureVector,
                  vector<KeyPoint> keypointsScene, Mat descScene);

int main(int argc, const char* argv[])
{
    //Input Image
    String fileLocation =
"/Users/ahmetcanozbek/Desktop/CSCI574/Homeworks/myHW3/CSCI574HW3AhmetOzbek/";
    String fileExtension = ".jpg";
    String fileName1 = "book2";
    String fileName2 = "test3";
    Mat objectImg = imread(fileLocation + fileName1 + fileExtension);
    Mat sceneImg = imread(fileLocation + fileName2 + fileExtension);

    //*SIFT
    //constructing with default parameters
    SIFT mySift;
    //Defining keypoints
    vector<KeyPoint> keypointsObject;
    vector<KeyPoint> keypointsScene;
    //getting keypoints and their locations
    Mat descObject;
    Mat descScene;
    //Initializing
    mySift.operator()(objectImg, noArray(),keypointsObject,descObject);
    mySift.operator()(sceneImg, noArray(),keypointsScene,descScene);
```

```
    //Padding the scene image with zeros so that its row numbers will be equal to
the object image
    Mat paddedSceneImg;
    vconcat(sceneImg, Mat::zeros((objectImg.rows-sceneImg.rows), sceneImg.cols,
sceneImg.type()), paddedSceneImg);
    //Merging the object image and the scene image so that we can see it in one
display
    Mat mergedImage;
    hconcat(objectImg, paddedSceneImg, mergedImage);

    Mat mergedImageSIFT = mergedImage.clone();
    //vector arrays to hold matched points
    vector<Point2f> objMatchPts;
    vector<Point2f> sceneMatchPts;
    //get the match index on the scene
    int matchPoints = 0;
    for(int i=0; i<keypointsObject.size(); i++){
        int fromIndex = i;
        int toIndex = getMatchIndex(descObject.row(i), keypointsScene, descScene);
        if(toIndex != -1){ //If good match
            //Declare the two points of the line
            Point2f fromPoint = keypointsObject[fromIndex].pt;    //From object
image
            Point2f toPoint = keypointsScene[toIndex].pt;         //To scene image

            //Record Match Points
            objMatchPts.push_back(fromPoint);
            sceneMatchPts.push_back(toPoint);

            //Modifying to the appropriate location in the merged image
            toPoint = Point2f(toPoint.x + 480, toPoint.y);

            //Draw the line
            line(mergedImageSIFT, fromPoint, toPoint, Scalar(0, 0, 255));
            //increment
            matchPoints = matchPoints + 1;
        }
    }
    //Display
    putText(mergedImageSIFT, "Match Points: " +
to_string(matchPoints),cvPoint(20,20),
            FONT_HERSHEY_PLAIN, 1.5, cvScalar(0,0,255));
    namedWindow("SIFT"); imshow("SIFT", mergedImageSIFT);
    imwrite(fileLocation + fileName1 + fileName2 + fileExtension, mergedImageSIFT);

    //*Homography Frame
    //Obtain H
    Mat H = findHomography(objMatchPts, sceneMatchPts, CV_RANSAC);
    cout << "H: \n" << H << endl;
    //Define Corners
    vector<Point2f> objectCorners(4);
    vector<Point2f> sceneCorners(4);
    objectCorners[0] = cvPoint(0,0); //top-left
    objectCorners[1] = cvPoint(objectImg.cols, 0); //top-right
    objectCorners[2] = cvPoint(objectImg.cols, objectImg.rows); //bottom-right
    objectCorners[3] = cvPoint(0, objectImg.rows); //bottom-left

    //Obtain the corners in the scene image for the object by the transform of H
    perspectiveTransform(objectCorners, sceneCorners, H);
    //Find the object (Draw the frame in the scene image around the detected
object);
    Mat mergedImageLINES = mergedImage.clone();
    sceneCorners[0] = Point2f(sceneCorners[0].x + 480, sceneCorners[0].y);
    sceneCorners[1] = Point2f(sceneCorners[1].x + 480, sceneCorners[1].y);
    sceneCorners[2] = Point2f(sceneCorners[2].x + 480, sceneCorners[2].y);
    sceneCorners[3] = Point2f(sceneCorners[3].x + 480, sceneCorners[3].y);
    line(mergedImageLINES, sceneCorners[0], sceneCorners[1], Scalar(255,0,255),4);
```

```
    line(mergedImageLINES, sceneCorners[1], sceneCorners[2], Scalar(255,0,255),4);
    line(mergedImageLINES, sceneCorners[2], sceneCorners[3], Scalar(255,0,255),4);
    line(mergedImageLINES, sceneCorners[3], sceneCorners[0], Scalar(255,0,255),4);
    //Display
    namedWindow("Frame"); imshow("Frame", mergedImageLINES);
    String resultType = "Frame";
    imwrite(fileLocation + fileName1 + fileName2 + resultType + fileExtension,
mergedImageLINES);

    //*Homography Matching Points
    vector<Point2f> siftKeypointsObject;
    for(int i=0; i<keypointsObject.size();i++){
        siftKeypointsObject.push_back(keypointsObject[i].pt);
    }
    vector<Point2f> siftKeypointsSceneNormal;
    for(int i=0; i<keypointsScene.size();i++){
        siftKeypointsSceneNormal.push_back(keypointsScene[i].pt);
    }

    Mat outDraw = mergedImage.clone();
    for(int i=0; i<siftKeypointsObject.size();i++){
        circle(outDraw,siftKeypointsObject[i], 3, Scalar(0,0,255));
    }
    vector<Point2f> siftKeypointsSceneByH;
    perspectiveTransform(siftKeypointsObject, siftKeypointsSceneByH, H);
    //Red by H (the points on the object image transformed by H matrix onto the
test(scene) image)
    for(int i=0; i<siftKeypointsSceneByH.size(); i++){
        siftKeypointsSceneByH[i] = Point2f(siftKeypointsSceneByH[i].x + 480,
siftKeypointsSceneByH[i].y);
        circle(outDraw,siftKeypointsSceneByH[i], 2, Scalar(0,0,255),1);
    }

    //Green (SIFT feature points) (the feature points calculated by SIFT algorithm
on the test(scene) image)
    for(int i=0; i<siftKeypointsSceneNormal.size(); i++){
        siftKeypointsSceneNormal[i] = Point2f(siftKeypointsSceneNormal[i].x + 480,
siftKeypointsSceneNormal[i].y);
        circle(outDraw,siftKeypointsSceneNormal[i], 2, Scalar(0,255,0),1);
    }

    namedWindow("Homography"); imshow("Homography", outDraw);
    resultType = "Homography";
    imwrite(fileLocation + fileName1 + fileName2 + resultType + fileExtension,
outDraw);


    waitKey(0);
    return 0;
}

int getMatchIndex(Mat featureVector,
                  vector<KeyPoint> keypointsScene, Mat descScene){
    double minEucDist = 1e6; //some large number
    int matchIndex = -1;
    for(int i=0; i<keypointsScene.size(); i++){
        double eucDist = norm(featureVector, descScene.row(i));
        if(eucDist < minEucDist){
            minEucDist = eucDist;
            matchIndex = i;
        }
    }
    if(minEucDist > 110){
        return -1;
    }
    return matchIndex;
}
```