ECE661 Computer Vision Homework 3

Hidekazu Iwaki

October 1, 2008

1 Finding Corresponding Point Pairs

1.1 Harris Corner Detector

Let the intensity image be $I(u,v)$ at coordinate(u,v).

1. Compute the gradient $\mathbf{g}(u,v) = (I_u(u,v), I_v(u,v))^T$ of $I(u,v)$ by the Sovel operator. Where

$$\begin{aligned} I_u(u,v) &\equiv I(u-1,v-1) + 2I(u-1,v) + I(u-1,v+1) \\ &\quad -I(u+1,v-1) - 2I(u+1,v) - I(u+1,v+1) \end{aligned}$$

$$\begin{aligned} I_v(u,v) &\equiv I(u-1,v-1) + 2I(u,v-1) + I(u+1,v-1) \\ &\quad -I(u-1,v+1) - 2I(u,v+1) - I(u+1,v+1). \end{aligned}$$

2. Compute variance and co-variance matrix $G(u,v)$ of the gradient at each pixel using a window of size 5x5. That is

$$G(u,v) = \frac{1}{25} \sum_{-2 \le j \le 2} \sum_{-2 \le i \le 2} \mathbf{g}(u+i, v+j)^T \mathbf{g}(u+i, v+j).$$

3. Compute Harris Image

$$H(u,v) = G_{11}(u,v) * G_{22}(u,v) + 0.04 * (G_{12}(u,v) + G_{21}(u,v))^2.$$

   Where $G_{11}(u,v)$ represent the element of $G(u,v)$ at 1st column and 1st raw and so on.

4. Operate the non-maximum suppression for $H(u,v)$ using a window of size 21x21. Let the result be $H_{sup}(u,v)$.

5. Find the maximum value $H_{max} = \max_{u,v} H(u,v)$.

6. Extract corner points $\mathbf{P}_i = (u_i, v_i)^T$ $(i = 1, 2, ..., N)$ to choose the points that have a larger value of $H_{sup}(u,v)$ than $H_{max} * HarrisThreshold$. Where $HarrisThreshold$ is a adjustable parameter for each images to control the number of corners. It typically has a value between 0.1 and 0.01. A larger value of $HarrisThreshold$ will provide more corner points but weaker corners.

1.2 Corresponding Corners Using Similarity Measures, SSD and NCC

Let the intensity image around a corner in the image A be $A(u,v)$ extracted from the image A. Similarly, let the intensity image around a corner in the image B be $B(u,v)$ extracted from the image B. The sizes of $A(u,v)$ and $B(u,v)$are same as $CorrespondingWindowSize^2$. Where $CorrespondingWindowSize$ is a adjustable parameter for each images to control the matching reliability. It typically has a value between 5 and 41. A larger value of $CorrespondingWindowSize$ will provide more reliable matching but less correspondences. I used two kinds of similarity measures SSD and NCC as follows ($W = CorrespondingWindowSize$):

1. Sum of squared deference(SSD)

$$SSD = \frac{1}{255} \sqrt{\frac{1}{W^2} \sum_{0 \le v \le W} \sum_{0 \le u \le W} \{A(u,v) - B(u,v)\}^2} \quad (0 \le SSD \le 1)$$

2. Normalized Cross Correlation(NCC)

$$NCC = \frac{\overline{AB} - \overline{A} \cdot \overline{B}}{\sqrt{\overline{A^2} - \overline{A}^2}\sqrt{\overline{B^2} - \overline{B}^2}} \quad (-1 \le NCC \le 1)$$

Where

$$\overline{A} = \frac{1}{W^2} \sum_{0 \le v \le W} \sum_{0 \le u \le W} A(u,v)$$

$$\overline{A^2} = \frac{1}{W^2} \sum_{0 \le v \le W} \sum_{0 \le u \le W} A^2(u,v)$$

$$\overline{B} = \frac{1}{W^2} \sum_{0 \le v \le W} \sum_{0 \le u \le W} B(u,v)$$

$$\overline{B^2} = \frac{1}{W^2} \sum_{0 \le v \le W} \sum_{0 \le u \le W} B^2(u,v)$$

$$\overline{AB} = \frac{1}{W^2} \sum_{0 \le v \le W} \sum_{0 \le u \le W} A(u,v)\, B(u,v)\,.$$

I corresponded each Harris corner of image A $\mathbf{P}_{Ai}$ to the Harris corner of image B $\mathbf{P}_{Bj}$ that has the maximum similarity more than the *SimilarityThreshold* in $\{\mathbf{P}_{Bk,} : max\left(u_{Ai} - u_{Bk}, v_{Ai} - v_{Bk}\right) < 50\}$. Where I assumed that corresponding points must be closer than 50 pixels to reduce the correspondences and the computation. *SimilarityThreshold* is a adjustable parameter for each images and for each similarity measures, SSD and NCC to control the matching reliability. It typically has a value between 0.05 and 0.15 for SSD and between 0.7 and 0.9 for NCC. A smaller value of *SimilarityThreshold* for SSD and a larger value of *SimilarityThreshold* for NCC will provide more reliable matching but less correspondences.

2 Experimental Result

I will show the parameters I used and the results for three images by Table. 1, Fig. 1 and Fig. 2. I adjusted the parameters to avoid mismatching. In Fig. 1 and Fig. 2, first is given image pair, second and third are taken by myself. Blue circles illustrate extracted Harris corners. Green lines illustrate correspondence by using SSD in Fig. 1 and by using NCC in Fig. 2.

	First Image	Second Image	Third Image
HarrisThreshold	0.08	0.07	0.09
CorrespondingWindowSize	31	51	21
SimilarityThreshold for SSD	0.09	0.14	0.15
SimilarityThreshold for NCC	0.8	0.75	0.8
# of corners of A and B (Blue Circles)	151 and 174	142 and 142	66 and 69
# of correspondences by SSD	21	70	28
# of correspondences by NCC	32	70	30

Table 1: Parameter and Experimental Result

3 C++ Source Code

I attach the source code.

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
```

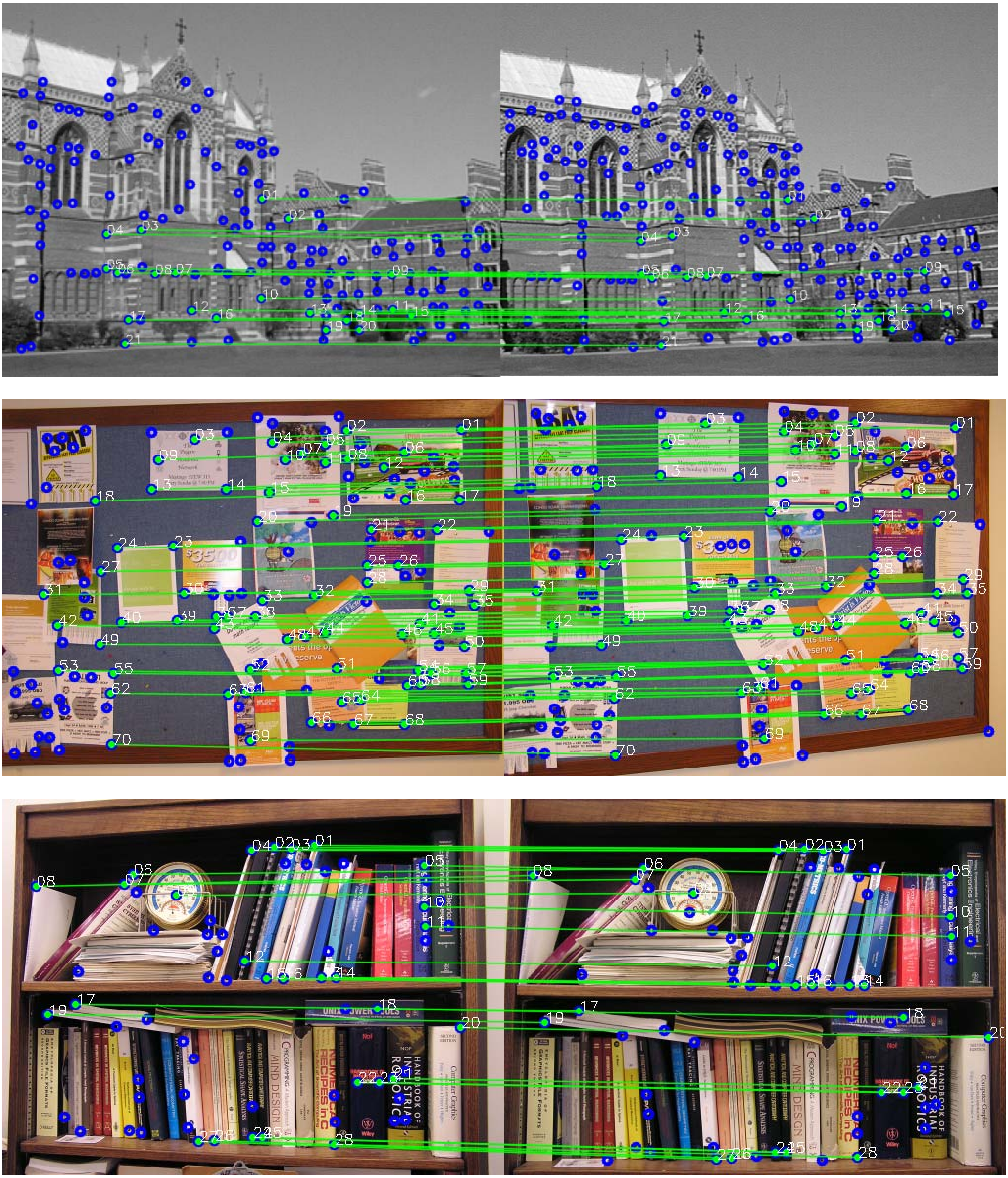

Figure 1: SSD

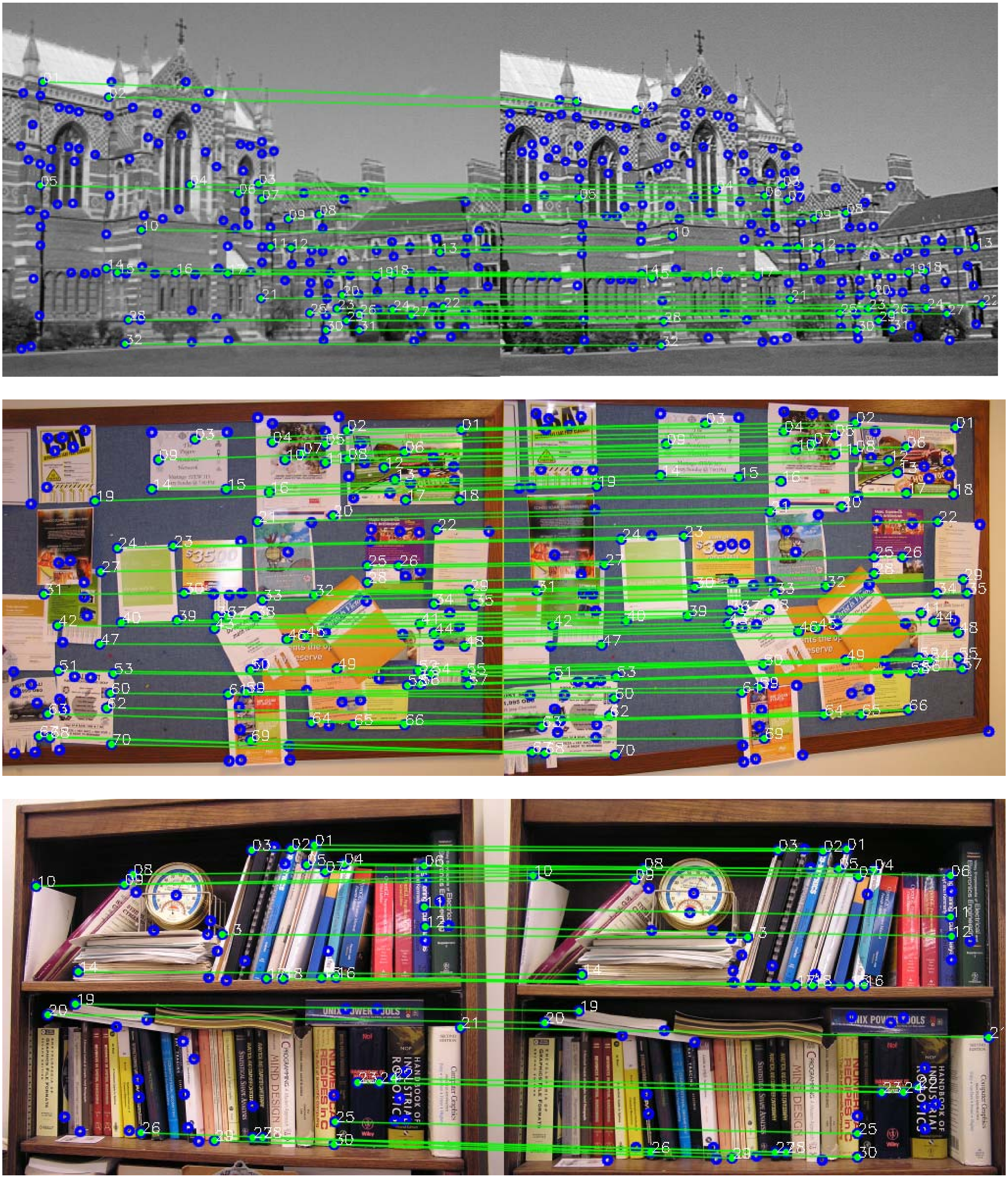

Figure 2: NCC

```
//Calculate NCC between pA and pB
// -1 <= NCC <= 1 ( if NCC=1 then pA and pB are exactly same)
double NCC(int aWinHalfSize, unsigned char *pA, unsigned char *pB) {
    int winWidth=2*aWinHalfSize+1;
    int winSize=winWidth*winWidth;
    double avgA=0, avgB=0, avgA2=0, avgB2=0, avgAB=0;
    for (int i = 0; i < winSize; ++i) {
        avgA+=pA[i];
        avgB+=pB[i];
        avgA2+=pA[i]*pA[i];
        avgB2+=pB[i]*pB[i];
        avgAB+=pA[i]*pB[i];
    }
    avgA/=(double)winSize;
    avgB/=(double)winSize;
    avgA2/=(double)winSize;
    avgB2/=(double)winSize;
    avgAB/=(double)winSize;
    double varA=avgA2-avgA*avgA;
    double varB=avgB2-avgB*avgB;
    double covarAB=avgAB-avgA*avgB;
    return covarAB/sqrt(varA)/sqrt(varB);
}

//Calculate SSD between pA and pB
// 0 <= SSD <= 1 ( if SSD=0 then pA and pB are exactly same)
double SSD(int aWinHalfSize, unsigned char *pA, unsigned char *pB) {
    int winWidth=2*aWinHalfSize+1;
    int winSize=winWidth*winWidth;
    double ssd=0;
    for (int i = 0; i < winSize; ++i) {
        ssd+=(pA[i]-pB[i])*(pA[i]-pB[i]);
    }
    ssd=sqrt(ssd/(double)winSize)/255;
    return ssd;
}

//Calculate SSD and NCC between at aPa in apImageA and at aPb in apImageB
//window size is 2*aWinHalfSize+1 X 2*aWinHalfSize+1
bool Sim(IplImage *apImageA, CvPoint2D32f aPa, IplImage *apImageB,
        CvPoint2D32f aPb, int aWinHalfSize, double *aSSD, double *aNCC) {
    int whs=aWinHalfSize;
    if ( (aPa.x - whs ) < 0 ||(aPa.x + whs ) >= apImageA->width ||
         (aPa.y - whs ) < 0 ||(aPa.y + whs ) >= apImageA->height ||
         (aPb.x - whs ) < 0 ||(aPb.x + whs ) >= apImageB->width ||
         (aPb.y - whs ) < 0 ||(aPb.y + whs ) >= apImageB->height) {
        return false;
    }
    int winWidth=2*whs+1;
    int winSize=winWidth*winWidth;
    unsigned char *a=new unsigned char [winSize];
    unsigned char *b=new unsigned char [winSize];
    int U0a=aPa.x-whs, V0a=aPa.y-whs;
    int U0b=aPb.x-whs, V0b=aPb.y-whs;

    //Extraction of templates
    for (int v = 0; v < winWidth; ++v) {
        int vStepA=(V0a+v)*apImageA->widthStep+U0a;
        int vStepB=(V0b+v)*apImageB->widthStep+U0b;
```

```
        for (int u = 0; u < winWidth; ++u) {
            a[v*winWidth+u]=(unsigned char)apImageA->imageData[vStepA+u];
            b[v*winWidth+u]=(unsigned char)apImageB->imageData[vStepB+u];
        }
    }
    *aSSD=SSD(whs, a, b);
    *aNCC=NCC(whs, a, b);
    return true;
}

//Find the corner in apImageB corresponding to the corner at aPa in apImageA
bool Corresponding(IplImage *apImageA, CvPoint2D32f aPa, IplImage *apImageB,
        int nPb, CvPoint2D32f *aPb, int *aIndexSSD, int *aIndexNCC,
        int aWinHalfSize, int aSearchDis, double aSSDTresh, double aNCCTresh) {
    double minssd=10000000;
    *aIndexSSD=-1;
    double maxncc=-10000000;
    *aIndexNCC=-1;
    for (int i = 0; i < nPb; ++i) {
        double ssd, ncc;
        if (fabs(aPa.x-aPb[i].x)<aSearchDis && fabs(aPa.y-aPb[i].y)<aSearchDis)
            if (Sim(apImageA, aPa, apImageB, aPb[i], aWinHalfSize, &ssd, &ncc)) {
                if (ssd>=0)
                    if (minssd>ssd) {
                        minssd=ssd;
                        *aIndexSSD=i;
                    }
                if (ncc>=-1)
                    if (maxncc<ncc) {
                        maxncc=ncc;
                        *aIndexNCC=i;
                    }
            }
    }
    if (minssd > aSSDTresh)
        *aIndexSSD=-1;
    if (maxncc < aNCCTresh)
        *aIndexNCC=-1;
    return true;
}

//Extract Harris Corner
bool Harris(IplImage *apOrgImage, int *aNPoint, CvPoint2D32f *apP,
        int aWinHalfSize, int aMinDistance, double aRatioMaxMin) {
    int winWidth=2*aWinHalfSize+1;
    int winSize=winWidth*winWidth;
    int w=apOrgImage->width;
    int h=apOrgImage->height;
    float *GImage = new float[apOrgImage->imageSize*3];
    float *HarrisImage = new float[apOrgImage->imageSize];
    float *HarrisImageV = new float[apOrgImage->imageSize];
    int iws=apOrgImage->widthStep;
    int gws=w*3;

    //Compute g(u,v)^T*g(u,v)
    for (int v = 1; v < apOrgImage->height-1; ++v) {
        unsigned char *img=(unsigned char*)apOrgImage->imageData;
        int vIStep=v*iws;
        int vGStep=v*gws;
        for (int u = 1; u < apOrgImage->width-1; ++u) {
```

```
        int intPos=vIStep+u;
        double sobleU= img[intPos-iws-1] + 2*img[intPos-1]
                + img[intPos+iws-1] -img[intPos-iws+1] - 2*img[intPos+1]
                - img[intPos+iws+1];
        double sobleV= img[intPos-iws-1] + 2*img[intPos-iws]
                + img[intPos-iws+1] -img[intPos+iws-1] - 2*img[intPos+iws]
                - img[intPos+iws+1];
        int tmp=vGStep+u*3;
        GImage[tmp+0]=sobleU*sobleU;
        GImage[tmp+1]=sobleV*sobleU;
        GImage[tmp+2]=sobleV*sobleV;
    }
}
double max=-1000000000;

//Compute G(u,v) and H(u,v) then find H_max
for (int v = aWinHalfSize; v < h-aWinHalfSize; ++v) {
    int vGStep=v*gws;
    for (int u = aWinHalfSize; u < w-aWinHalfSize; ++u) {
        int intPos=vGStep+u*3;
        double g11=0, g12=0, g22=0;
        for (int vv = -aWinHalfSize; vv < aWinHalfSize; ++vv) {
            for (int uu = -aWinHalfSize; uu < aWinHalfSize; ++uu) {
                int tmp=intPos+vv*gws+uu*3;
                g11+=GImage[tmp+0];
                g12+=GImage[tmp+1];
                g22+=GImage[tmp+2];
            }
        }
        g11/=(double)winSize;
        g12/=(double)winSize;
        g22/=(double)winSize;
        HarrisImage[v*w+u]=(g11*g22-g12*g12)+0.04*(g11+g22)*(g11+g22);
        if (max<HarrisImage[v*w+u])
            max=HarrisImage[v*w+u];
    }
}

//Non-maximum suppression
for (int v = aMinDistance; v < h-aMinDistance; ++v) {
    int vHStep=v*w;
    for (int u = aMinDistance; u < w-aMinDistance; ++u) {
        int intPos=vHStep+u;
        float intVal=HarrisImage[intPos];
        HarrisImageV[vHStep+u]=intVal;
        for (int vv = -aMinDistance; vv < aMinDistance; ++vv) {
            for (int uu = -aMinDistance; uu < aMinDistance; ++uu) {
                if (intVal<HarrisImage[intPos+vv*w+uu]) {
                    HarrisImageV[vHStep+u]=0;
                    break;
                }
            }
        }
    }
}
int count=0;

//Extract corners
for (int v = 0; v < h; ++v) {
    int vHStep=v*w;
```

```
        for (int u = 0; u < w; ++u) {
            if ((HarrisImageV[vHStep+u]!=0)&&(HarrisImageV[vHStep+u]>max
                    *aRatioMaxMin)) {
                if (count < *aNPoint) {
                    apP[count].x=u;
                    apP[count].y=v;
                    count++;
                }
            }
        }
    }
    *aNPoint=count;

    return true;
}

int main(int argc, char **argv) {
    std::string folderName = "./";
//  std::string fileName="Book";
//  std::string fileName="sample";
      std::string fileName="Board";

    double HarrisMaxMinRatio=0.1;
    int HarrisWin=2;
    int HariisMinDis=10;

    int CorrespoindingMaxDis=50;
    int CorrespoindingWin=10;
    double threshSSD=0.1;
    double threshNCC=0.8;

    //Read parameter
    std::string tmp;
    std::ifstream ifs(( fileName + ".dat").c_str());
    ifs >> tmp >> HarrisMaxMinRatio;
    ifs >> tmp >> HarrisWin;
    ifs >> tmp >> HariisMinDis;
    ifs >> tmp >> CorrespoindingMaxDis;
    ifs >> tmp >> CorrespoindingWin;
    ifs >> tmp >> threshSSD;
    ifs >> tmp >> threshNCC;

    //Get image
    IplImage *pOrgImageA = cvLoadImage((folderName + fileName + "A.png").c_str());
    IplImage *pOrgImageB = cvLoadImage((folderName + fileName + "B.png").c_str());
    int wa=pOrgImageA->width;
    int ha=pOrgImageA->height;
    int sa=pOrgImageA->widthStep;

    //Harris Corner Detection of Image A
    int corner_countA = 1000;
    IplImage *src_img_grayA = cvCreateImage(cvGetSize(pOrgImageA), 8, 1);
    CvPoint2D32f *cornersA = (CvPoint2D32f *) cvAlloc(corner_countA
            * sizeof(CvPoint2D32f));
    cvConvertImage(pOrgImageA, src_img_grayA);
    Harris(src_img_grayA, &corner_countA, cornersA, HarrisWin, HariisMinDis,
            HarrisMaxMinRatio);

    //Harris Corner Detection of Image B
    int corner_countB = 1000;
```

```
IplImage *src_img_grayB = cvCreateImage(cvGetSize(pOrgImageB), 8, 1);
CvPoint2D32f *cornersB = (CvPoint2D32f *) cvAlloc(corner_countB
        * sizeof(CvPoint2D32f));
cvConvertImage(pOrgImageB, src_img_grayB);
Harris(src_img_grayB, &corner_countB, cornersB, HarrisWin, HariisMinDis,
        HarrisMaxMinRatio);

//Preparation for Visualization
CvScalar ssdMatched=CV_RGB(0, 255, 0);
CvScalar harrisCorner=CV_RGB(0, 0, 255);
CvScalar nccMatched=CV_RGB(0, 255, 0);
CvScalar fontColor=CV_RGB(255, 255, 255);
CvFont font;
cvInitFont(&font, CV_FONT_HERSHEY_SCRIPT_SIMPLEX, 0.5, 0.5);

//Preparation for Visualization
IplImage *pImageSSD = cvCreateImage(cvSize(wa+pOrgImageB->width, ha),
IPL_DEPTH_8U, 3);
IplImage *pImageSSD2 = cvCreateImage(cvSize(wa+pOrgImageB->width, ha),
IPL_DEPTH_8U, 3);
IplImage *pImageNCC = cvCreateImage(cvSize(wa+pOrgImageB->width, ha),
IPL_DEPTH_8U, 3);
IplImage *pImageNCC2 = cvCreateImage(cvSize(wa+pOrgImageB->width, ha),
IPL_DEPTH_8U, 3);
for (int v=0; v<pOrgImageA->height; v++)
    memcpy(&(pImageSSD->imageData[v * pImageSSD->widthStep]),
            &(pOrgImageA->imageData[v * sa]), wa * 3);
for (int v=0; v<pOrgImageB->height; v++)
    memcpy(&(pImageSSD->imageData[v * pImageSSD->widthStep + wa * 3]),
            &(pOrgImageB->imageData[v * sa]), wa * 3);
for (int v=0; v<pImageNCC->height; v++)
    memcpy(&(pImageNCC->imageData[v * pImageNCC->widthStep]),
            &(pOrgImageA->imageData[v * sa]), wa * 3);
for (int v=0; v<pImageNCC->height; v++)
    memcpy(&(pImageNCC->imageData[v * pImageNCC->widthStep + wa * 3]),
            &(pOrgImageB->imageData[v * sa]), wa * 3);

//Draw Detected Harris Corners
for (int i = 0; i < corner_countA; i++) {
    CvPoint pa=cvPointFrom32f(cornersA[i]);
    cvCircle(pImageSSD, pa, 4, harrisCorner, 2, CV_AA);
    cvCircle(pImageNCC, pa, 4, harrisCorner, 2, CV_AA);
}
for (int i = 0; i < corner_countB; i++) {
    CvPoint pb=cvPoint((int)cornersB[i].x+wa, (int)cornersB[i].y);
    cvCircle(pImageSSD, pb, 4, harrisCorner, 2, CV_AA);
    cvCircle(pImageNCC, pb, 4, harrisCorner, 2, CV_AA);
}

//Matching and Visualization
int counter=0;
int nSSD=0, nNCC=0;
for (int i = 0; i < corner_countA; i++) {
    int indexSSD, indexNCC;
    Corresponding(src_img_grayA, cornersA[i], src_img_grayB, corner_countB,
            cornersB, &indexSSD, &indexNCC, CorrespoindingWin,
            CorrespoindingMaxDis, threshSSD, threshNCC);
    if (indexNCC>=0) {
        nNCC++;
        CvPoint pa=cvPointFrom32f(cornersA[i]);
```

```
            CvPoint pb=cvPoint((int)cornersB[indexNCC].x+wa,
                    (int)cornersB[indexNCC].y);
            char c[3];
            sprintf(c, "%02d", nNCC);
            cvPutText(pImageNCC2, c, pa, &font, fontColor);
            cvPutText(pImageNCC2, c, pb, &font, fontColor);
            cvCircle(pImageNCC, pa, 3, nccMatched, -1, CV_AA);
            cvLine(pImageNCC, pa, pb, nccMatched, 1, CV_AA);
            cvCircle(pImageNCC, pb, 3, nccMatched, -1, CV_AA);
        }
        if (indexSSD>=0) {
            nSSD++;
            CvPoint pa=cvPointFrom32f(cornersA[i]);
            CvPoint pb=cvPoint((int)cornersB[indexSSD].x+wa,
                    (int)cornersB[indexSSD].y);
            cvCircle(pImageSSD, pa, 3, ssdMatched, -1, CV_AA);
            cvLine(pImageSSD, pa, pb, ssdMatched, 1, CV_AA);
            cvCircle(pImageSSD, pb, 3, ssdMatched, -1, CV_AA);
            char c[3];
            sprintf(c, "%02d", nSSD);
            cvPutText(pImageSSD2, c, pa, &font, fontColor);
            cvPutText(pImageSSD2, c, pb, &font, fontColor);
            counter++;
        }
    }
    cvNamedWindow(fileName.c_str(), CV_WINDOW_AUTOSIZE);
    cvOr(pImageSSD, pImageSSD2, pImageSSD);
    cvShowImage(fileName.c_str(), pImageSSD);
    cvWaitKey(-1);
    cvOr(pImageNCC, pImageNCC2, pImageNCC);
    cvShowImage(fileName.c_str(), pImageNCC);
    cvWaitKey(-1);

    //Save Result
    cvSaveImage((fileName+"NCC"+".png").c_str(), pImageNCC);
    cvSaveImage((fileName+"SSD"+".png").c_str(), pImageSSD);
    std::ofstream ofs(( fileName + ".dat").c_str());
    ofs << "HarrisMaxMinRatio= "<< HarrisMaxMinRatio << std::endl;
    ofs << "HarrisWin= "<< HarrisWin << std::endl;
    ofs << "HariisMinDis= "<< HariisMinDis << std::endl;
    ofs << "CorrespoindingMaxDis= "<< CorrespoindingMaxDis << std::endl;
    ofs << "CorrespoindingWin= "<< CorrespoindingWin << std::endl;
    ofs << "threshSSD= "<< threshSSD << std::endl;
    ofs << "threshNCC= "<< threshNCC << std::endl;
    ofs << "corner_countA= "<< corner_countA << std::endl;
    ofs << "corner_countB= "<< corner_countB << std::endl;
    ofs << "nNCC= "<< nNCC << std::endl;
    ofs << "nSSD= "<< nSSD << std::endl;

    return 0;
}
```