Project 1

Akshat Gupta

October 7, 2009

Contents

1 Platform

The entire project was written in Linux. As a result the skeleton code had to be changed. But I have made sure that I stick to standard C++ to fill in the TODOs. So I guess the same code can be run on Windows, tough I have not tested it. The code can be compiled by going in the Panorama directory and running make. For executing create a dir in Panorama and add the pmg files. Enter the dir and then execute each program.

MatchPair:
../MatchPair/MatchPair -i 21 -j 1
python ../match.py 1 21

AlignPair:
../AlignPair/AlignPair -i 2 -j 1 -tol 1.5 -iter 100
or python ../allign.py

CameraPose:
../CameraPose/CamerPose -ib 1 -ie 22 -u 320 -v 213 -alpha 1 -tol 2

WarpCylinderical:
../WarpCylinderical/WarpCylinderical -ib 1 -ie 22

When running for my data set, I found something really strange. Sometimes the Panorama was really good but sometimes it was really bad. The reason for this is the randomization of allignpair.

2 Code: Matchpair

For computing the matching pair between two features simple euclidean distances are used for all the 128 dimensions. The code can be seen at Listings 1.

Listing 1: FindMatch()

```
int FindMatch ( const CFeature* ft , const CFeatureArray& set )
{
    double dist = -1;
    double min_dist = -1;
    int index = 0;
// ToDo 1: Fill out this function
    for(int i=0; i < set.size(); i++) {
        dist = 0;
        for(int j=0; j < ft->m_numDim; j++) {
            dist += (ft->d[j] - set[i]->d[j])*(ft->d[j] - set[i]->d[j]);
        }
        if(min_dist == -1) {
            min_dist = dist;
            index = i;
        }
        if(dist < min_dist){
            min_dist = dist;
            index = i;
        }
    }
    return index;
}
```

3 Code: RansacHomography

To perform RansacHomography the code has been implemented based on the algorithm provided in the spec.

Listing 2: RansacHomography

```
void RansacHomography( CHomography& homo,
                       const MatchArray& aryMatch,
                       const CFeatureArray& set1,
                       const CFeatureArray& set2,
                       float inlierTol, int numIter )
{
        const float SQR_TOL = inlierTol*inlierTol;
        const int NUM_SAMP = 6;
        int maxInlier = 0;
        double dA[ NUM_SAMP*2*8 ];
        double dB[ NUM_SAMP*2 ];
        double dH[8];
        struct timeval tv;
        gettimeofday(&tv, NULL);
        srand48(tv.tv_usec);
        long int index;
        int flag;
        vector < int > dup_index;
        int test_index[] = {100, 150, 200, 250, 300, 350};
// ToDo2: Find homography using RANSAC

        for(int i=0; i < NUM_SAMP; i++) {
            dB[i] = 0;
        }

        for(int i=0; i< numIter; i++) {
            dup_index.clear();
            for(int j =0 ; j< NUM_SAMP; j++ ) {
                index = lrand48();
                index = index % aryMatch.size();
//                  index = test_index[j];
                flag = 1;
                for(unsigned int k=0; k < dup_index.size(); k++) {
                    if(dup_index[k] == index) {
                        flag = 0;
                    }
                }
                if(!flag) {
                    --j;
                    continue;
                }
//                std::cout << "Index " << index << std::endl;
                int ind1 = aryMatch[index].first;
                int ind2 = aryMatch[index].second;

                dA[16*j + 0] = -1 * set1[ind1]->x;
                dA[16*j + 1] = -1 * set1[ind1]->y;
                dA[16*j + 2] = -1;
                dA[16*j + 3] = 0;
                dA[16*j + 4] = 0;
                dA[16*j + 5] = 0;
                dA[16*j + 6] = set2[ind2]->x * set1[ind1]->x;
                dA[16*j + 7] = set2[ind2]->x * set1[ind1]->y;

                dA[16*j + 8 + 0] = 0;
                dA[16*j + 8 + 1] = 0;
                dA[16*j + 8 + 2] = 0;
                dA[16*j + 8 + 3] = -1 * set1[ind1]->x;
                dA[16*j + 8 + 4] = -1 * set1[ind1]->y;
                dA[16*j + 8 + 5] = -1;
                dA[16*j + 8 + 6] = set2[ind2]->y * set1[ind1]->x;
                dA[16*j + 8 + 7] = set2[ind2]->y * set1[ind1]->y;
                dB[2*j] = -1*set2[ind2]->x;
                dB[2*j + 1] = -1*set2[ind2]->y;

            }

            for(int j = 0; j < 8; j++)
                dH[j] = 0;
            CvMat mA = cvMat( NUM_SAMP * 2, 8, CV_64FC1, dA);
            CvMat mB = cvMat( NUM_SAMP * 2, 1, CV_64FC1, dB);
            CvMat mH = cvMat( 8, 1, CV_64FC1, dH);
            int ret = cvSolve(&mA, &mB, &mH, CV_SVD);
```

```
//              std::cout << ret << std::endl;
            for(int j = 0; j < 8; j++) {
                dH[j] = cvmGet(&mH, j, 0);
//                  std::cout << "H is " << std::endl;
//                  std::cout << dH[j] << std::endl;
            }
            //continue;
            int inlier_count = 0;
            for(unsigned int j = 0; j< aryMatch.size(); j++) {
                int ind1 = aryMatch[j].first;
                int ind2 = aryMatch[j].second;
                double x2 = set2[ind2]->x;
                double y2 = set2[ind2]->y;
                double x1 = set1[ind1]->x;
                double y1 = set1[ind1]->y;
                double _x1p = dH[0] * x1 + dH[1] * y1 + dH[2]*1;
                double _y1p = dH[3] * x1 + dH[4] * y1 + dH[5]*1;
                double h1p = dH[6] * x1 + dH[7] * y1 + 1;
                double x1p = _x1p / h1p;
                double y1p = _y1p / h1p;
                double dist = (x1p - x2)*(x1p - x2) + (y1p - y2)*(y1p - y2);
                if( sqrt(dist) < SQR_TOL) {
                    //  std::cout << "Coming here" << std::endl;
                    inlier_count++;
                }
            }

            if(inlier_count > maxInlier) {
                maxInlier = inlier_count;
                for(int k = 0; k < 8; k++)
                    homo.d[k] = dH[k];

                homo.d[8] = 1;
            }
        }


        printf( "homography inliers: %d(%d)\n", maxInlier, aryMatch.size() );
}
```

4 Code: PoseFromHomography

Listing 3: PoseFromHomography()

```
void PoseFromHomography( double* R, const CHomography& homo, double* K1, double* K2 )
{
    // Get the initial rotation (relative to cam1) from the homography.
    // This can be derived from:
    //
    //      R = inv(K2) x H x K1 (ref).
    //
    // However, since K1 and K2 may not be very accurate, the rotation matrix
    // can be of poor quality. We will use bundle adjustment to fix it later.

    //ToDo3: Compute R
    double iK2[9], temp1[9], temp2[9], temp3[9];
    for(int i = 0; i< 9; i++) {
        iK2[i] = 0;
        temp1[i] = 0;
        temp2[i] = 0;
        temp3[i] = 0;
    }
    CvMat mK1 = cvMat( 3, 3, CV_64FC1, K1);
    CvMat mK2 = cvMat( 3, 3, CV_64FC1, K2);
    CvMat mH = cvMat( 3, 3, CV_64FC1, (double *)homo.d);
    CvMat imK2 = cvMat( 3, 3, CV_64FC1, iK2);
    CvMat mtemp = cvMat( 3, 3, CV_64FC1, temp1);
    CvMat mR = cvMat( 3, 3, CV_64FC1, temp2);

    cvInvert(&mK2, &imK2);
    cvMatMul(&imK2, &mH, &mtemp);
    cvMatMul(&mtemp, &mK1, &mR);

    for(int j = 0; j < 9; j++) {
        R[j] = cvmGet(&mR, j/3, j%3 );
    }

}
```

5 Code: ImageBlending

For Image blending a simple algorithm of Luminosity mathching has been used. Intensities for two consecutive images are averaged and the diffrence is taken out. From the higher luminosity image we subtract the difference / 2 and from the lower luminosity we add. This makes the consecutive image of the same intensity level.

As some tweaks I also tried some minor enhancements. Since I am taking two images at a time and modifying both, every image gets modified twice. Thus, the results are not that great. It is great if the evg intensities of all the images are close to each other but if one image acts as an outlier, that image does not get blended very well. To improve the results I tried repeating the logic multiple times and them did MWA(Moving window average). The code can be seen in listing 4.

Apart from averaging the intensity, weighted average was taking using a mask. The mask is a simple on the center as 1 and corners of 0.1. Since the code has been mixed up with the skeleton code, The entire function is presented here.

Listing 4: ImageBlending

```
int main(int argc, char* argv[])
{
    // the size of the the panorama image
    CvSize szPano = cvSize( 8000, 480 );

    // the radius
    const double r = (double)szPano.width / (2*M_PI);

    // the pan angle per pixel
    const double angPixel =  2*M_PI / (double)szPano.width;

    // the number of images in the sequence
    int ib = 0;
    int ie = 0;
    int numImage = 0;

    int arg = 0;
    while( ++arg < argc)
    {
        if( !strcmp(argv[arg], "-ib") )
            ib = atoi( argv[++arg] );

        if( !strcmp(argv[arg], "-ie") )
            ie = atoi( argv[++arg] );
    }

    // the indices of the image sequence
    numImage = ie-ib;

    int* imgId = new int[ numImage ];

    for( int i=0; i<numImage; ++i )
        imgId[i] = ib+i;
    // the output panorama
    IplImage* imgPano = cvCreateImage( szPano, IPL_DEPTH_8U, 1 );

    IplImage* weightPano = cvCreateImage( szPano, IPL_DEPTH_32F, 1 );
    IplImage* imgFloatPano = cvCreateImage( szPano, IPL_DEPTH_32F, 1 );

    cvSetZero( imgPano );
    cvSetZero( weightPano );
    cvSetZero(imgFloatPano);

    cvSetZero( imgPano );

    IplImage* cv_image[numImage];

    for( int i=0; i<numImage; ++i) {
        char strName[128];
        sprintf( strName, "image%04d.pgm", imgId[i] ); // modify according to your image na
        std::cout << strName << std::endl;
        cv_image[i] = cvLoadImage( strName, 0 );
    }

    int count;
    double avg_int1 = 0, avg_int2 = 0;
//        for( int temp =0; temp < 20; temp++) {
    for( int i=0; i<numImage; i++) {
        count = 0;
        int _avg_int1 = 0;
        int _avg_int2 = 0;

        for(int j=0; j < cv_image[i]->height; j++) {
            for(int k=0; k < cv_image[i]->width; k++) {
                _avg_int1 += cvGetReal2D( cv_image[i], j, k );
                count ++;
            }
        }
        _avg_int1 /= (double)count;
        avg_int1 = 0.0 * avg_int1 + 1*_avg_int1;
        //avg_int1 /=2;
        count = 0;
        for(int j=0; j < cv_image[(i + 1)%numImage ]->height; j++) {
            for(int k=0; k < cv_image[(i+1)%numImage ]->width; k++) {
                _avg_int2 +=
                    cvGetReal2D( cv_image[(i+1)%numImage ], j, k );
                count ++;
            }
        }
        _avg_int2 /= (double)count;
        avg_int2 = 0.0 * avg_int2 + 1.0*_avg_int2;
        //avg_int2 /=2;
        for(int j=0; j < cv_image[i]->height; j++) {
            for(int k=0; k < cv_image[i]->width; k++) {
                double diff = avg_int2 - avg_int1;
//                      double diff = 128 - avg_int1;
                double val = cvGetReal2D( cv_image[i], j, k );
                val += diff/2.0;
                cvSetReal2D( cv_image[i], j, k, val );
            }
        }

        for(int j=0; j < cv_image[(i + 1)%numImage ]->height; j++) {
            for(int k=0; k < cv_image[(i+1)%numImage ]->width; k++) {
                double diff = avg_int1 - avg_int2;
//                      double diff = 128 - avg_int2;
                double val = cvGetReal2D( cv_image[(i+1)%numImage ], j, k );
                val += diff/2.0;
                cvSetReal2D( cv_image[(i+1)%numImage ], j, k, val );
            }
        }

    }

    double maxWt=0;
    double **weight;

    weight = (double **)malloc(sizeof(double*) * cv_image[0]->height);
    for(int i=0; i<cv_image[0]->height; i++) {
        weight[i] = (double *)calloc(cv_image[0]->width, sizeof(double));
    }
    for(int i = 0; i < cv_image[0]->height; i++)
        for(int j = 0; j < cv_image[0]->width; j++)
            weight[i][j] = 0.0;

    int cr = cv_image[0]->height/2;
    int cc = cv_image[0]->width/2;

    for(int i = 0; i < cv_image[0]->height; i++)
    {
        for(int j = 0; j < cv_image[0]->width; j++)
        {
            double value = (1.0 - (abs(cr - i)/((float)cr)) + 1.0 - (abs(cc - j)/((float)cc
            if(value > maxWt)
                maxWt = value;
            weight[i][j] = value;
        }
    }

    for( int i=0; i<numImage; ++i ) // for each image
    {
        // load the iamge
        // char strName[128];
        // sprintf( strName, "image1%d.pgm", imgId[i] ); // modify according to your image
        // IplImage* img = cvLoadImage( strName, 0 );
        IplImage* img = cv_image[i];
        // load the camera
        CCamera cam;
        LoadCamera( cam, imgId[i] );

        //
        double dir[3];

        CvPoint2D64f corner[4] = {
            cvPoint2D64f( 0, 0 ),
            cvPoint2D64f( 0, img->height ),
            cvPoint2D64f( img->width, 0 ),
            cvPoint2D64f( img->width, img->height ) };

        int lm = INT_MAX; // left most
        int rm = -INT_MAX; // right most

        // find the left end of the image in the panorama
        for( int j=0; j<2; ++j )
        {
            cam.GetRayDirectionWS( dir, corner[j] );
            double theta = atan2( dir[0], dir[2] );
            lm = ( lm < floor( szPano.width *theta /(2*M_PI) ) ) ? lm : floor( szPano.width
        }

        // find the right end of the image in the panorama
        for( int j=0; j<2; ++j )
        {
            cam.GetRayDirectionWS( dir, corner[j+2] );
            double theta = atan2( dir[0], dir[2] );
            rm = ( rm > ceil( szPano.width *theta /(2*M_PI) ) ) ? rm : ceil( szPano.width *
        }
        if( lm > rm ) // 180 degree crossing
            lm -= szPano.width;

        printf( "L:%d, R:%d\n", lm, rm );

        // ToDo 4: Image blending
        for( int m=0; m<szPano.height; ++m )
            for( int n=lm; n<rm; ++n )
            {
                double theta = n * angPixel;
                double tan_phi = ( m-szPano.height/2 )/r;

                //
                CvPoint3D64f pt = cvPoint3D64f( r*sin( theta ), r*tan_phi, r*cos( theta ) )

                // check if the point is in front of the camera
                CvPoint2D64f ptProj = cam.GetProjection( pt );

                int bx = ptProj.x +0.5;
                int by = ptProj.y +0.5;

                if( bx >=0 && bx < img->width && by >=0 && by < img->height )
                {
                    double v = cvGetReal2D( img, by, bx );
                    double wt = weight[by][bx];
                    cvSetReal2D( imgPano, m, n<0? n+szPano.width: n, v );
                    double temp =cvGetReal2D( imgFloatPano, m, n<0? n+szPano.width: n);
                    temp += (v/255)*wt;
                    cvSetReal2D( imgFloatPano, m, n<0? n+szPano.width: n, temp );

                    // Used for linear method.
                    double pano_wt =cvGetReal2D( weightPano, m, n<0? n+szPano.width: n);
                    cvSetReal2D( weightPano, m, n<0? n+szPano.width: n, pano_wt+wt );

                }
            }
    }

    IplImage* wresult = cvCreateImage(szPano,IPL_DEPTH_32F,1);
    cvDiv(imgFloatPano,weightPano,wresult);

    cvSaveImage( "panorama.jpg", wresult );
    cvReleaseImage( &imgPano );

    return 0;
```

```
}
```

6 Code: OptimizeSet

This code has been inspired by optimize pair and has been extended to support all the images at one time. This removes the black spaces and the bumps from the Panorama.

Listing 5: OptimizeSet

```
void OptimizeSet( CCamera* cam,
                  const CHomography* homo,
                  const CFeatureArray* set,
                  const MatchArray* aryMatch,
                  int numImage )
{
    //ToDo5: Extra credit

    CBundleAdjust ba( numImage, BA_ITER );

    for( int i = 0; i < numImage; i++ )
        ba.SetCamera(&cam[i], i);

    for( int j = 0; j<numImage; j++)
    {
        MatchArray aryInlier = aryMatch[j];

        int pre_j = (j==0) ? numImage-1 : j-1;

        CFeatureArray set1 = set[pre_j];
        CFeatureArray set2 = set[j];

        for( int k=0; k<aryInlier.size(); k++ )
        {
            const CFeature* ft1 = set1[ aryInlier[k].first ];
            const CFeature* ft2 = set2[ aryInlier[k].second ];

            double dir[3];
            cam[pre_j].GetRayDirectionWS( dir, cvPoint2D64f( ft1->x, ft1->y ) );

            CvPoint3D64f pt3 = cvPoint3D64f( dir[0]*radius, dir[1]*radius, dir[2]*radius );

            ba.SetPointProjection( pt3, j, cvPoint2D64f( ft2->x, ft2->y ) );
            ba.SetPointProjection( pt3, pre_j, cvPoint2D64f( ft1->x, ft1->y ) );
        }
    }
    ba.DoMotionAndStructure();

    for(int i=0; i < numImage ; i++)
    {
        ba.GetAdjustedCamera( &cam[i], i );
    }

}
```

7 CameraPose

```
Camera 1:
826.987 320 213 1
0.00802427 0.637215 0.0057802
0 0 0

Camera 2:
825.881 320 213 1
-0.00591917 0.23473 0.0111685
0 0 0

Camera 3:
825.532 320 213 1
-0.0169962 -0.10312 0.0140472
0 0 0

Camera 4:
825.891 320 213 1
-0.0305269 -0.436302 0.0127993
0 0 0

Camera 5:
826.089 320 213 1
-0.0459558 -0.780591 0.0111643
0 0 0

Camera 6:
827.548 320 213 1
```

```
-0.058889 -1.06962 0.0094564
0 0 0

Camera 7:
831.989 320 213 1
-0.0705965 -1.44125 0.00983642
0 0 0

Camera 8:
833.62 320 213 1
-0.0782128 -1.73201 0.00902734
0 0 0

Camera 9:
833.843 320 213 1
-0.0851043 -2.02545 0.00900462
0 0 0

Camera 10:
831.043 320 213 1
-0.0890686 -2.37263 0.00764895
0 0 0

Camera 11:
830.526 320 213 1
-0.1048 -2.74868 -0.00145914
0 0 0

Camera 12:
825.55 320 213 1
-0.107268 -3.09712 -0.00730762
0 0 0

Camera 13:
828.738 320 213 1
0.0939474 2.86215 0.0115522
0 0 0

Camera 14:
830.05 320 213 1
```

```
0.0842323 2.65522 0.0133004
0 0 0

Camera 15:
824.788 320 213 1
0.0644153 2.30608 0.0168021
0 0 0

Camera 16:
832.116 320 213 1
0.0500192 1.9995 0.0133254
0 0 0

Camera 17:
832.54 320 213 1
0.0355285 1.59389 0.0157644
0 0 0

Camera 18:
832.585 320 213 1
0.028412 1.31976 0.0156021
0 0 0

Camera 19:
833.091 320 213 1
0.022699 1.09011 0.0121814
0 0 0

Camera 20:
830.781 320 213 1
0.0152297 0.845381 0.0103597
0 0 0

Camera 21:
829.55 320 213 1
0.00912097 0.656128 0.0112686
0 0 0
```