CS 223b: Introduction to Computer Vision
Assignment 1: Camera Calibration
Solutions

1 Calibration Software

The solution handed in by Kurt Miller, Kayur Patel and Justin Tansuwan turned out to be so good that we decided to publish is as a reference program for this assignment. It is attached at the bottom of this solution paper and can be downloaded as `hw1_sol.zip` from the class homepage.

Some nice bonuses included by some groups were:

- **Painting the corners found in the image**. This is a very useful tool to find errors in the corner finding algorithm - OpenCV cannot find them in all images. An additional numbering or color-coding of the corners makes sure that corner ordering is the same in the image and the scene (see Fig. 1).

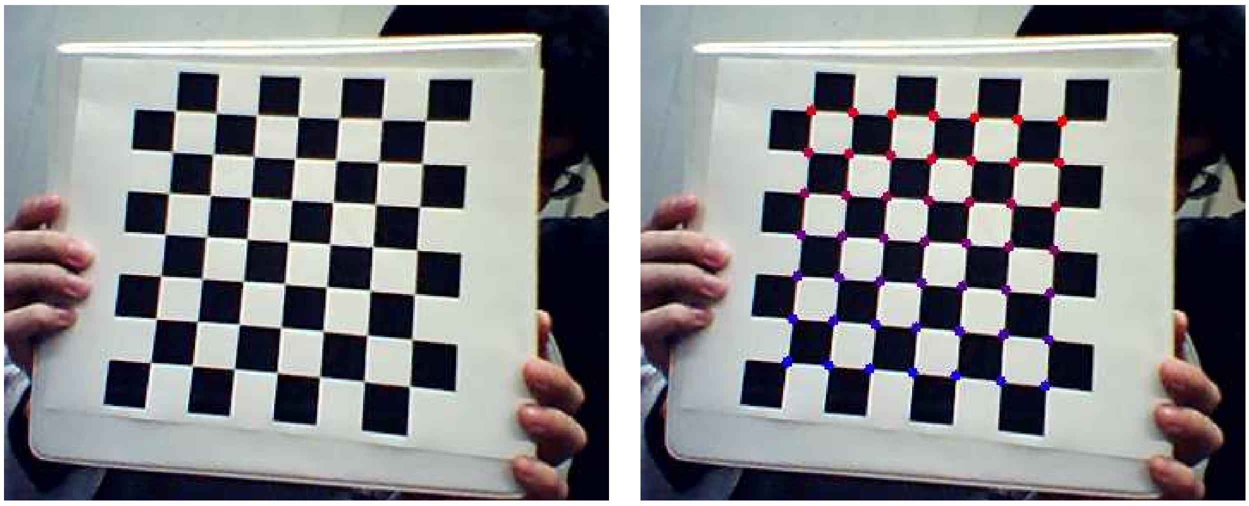
Figure 1: Original image and image overlayed with detected corners

- **Backprojecting the corners**. Since `cvCalibrate()` returns not only the camera parameters, but also the rotation/translation information for every image, we can easily project 3D scene coordinates into the image. This is a very good test whether the calibration data is correct, since the backprojected corner points from the scene should match the ones in the image. The left part of Figure 2 shows the result for a proper calibration.

- **Undistorting the image**. The math for the backprojection is linear (except for the final projection step) and therefore assumes that the image is not distorted. To compute undistorted images, we can feed the camera parameters into an OpenCV function called `cvUndistortOnce`. This function warps the image in a way that parallel lines in the image are parallel in the scene (Fig. 2, right).

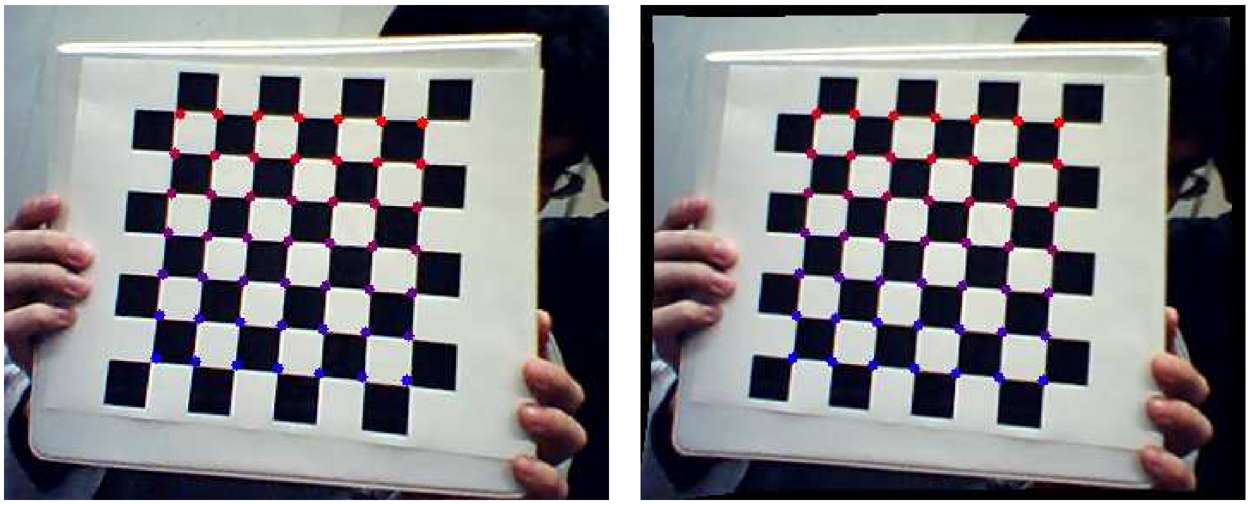

Figure 2: 3D Scene coordinates backprojected into the image using the intrinsic and extrinsic matrices. The left image shows the result before correcting for image distortion, the right one afterwards.

2 Calibration Results

Every camera is slightly different, so a lot of values can be correct calibration results. Your solution is o.k. as long as it does not deviate too much from these average values:

	Average	Standard Deviation
Focal length in x pixels f/s_x	459.0	9.8
Focal length in y pixels f/s_y	458.7	8.8
Optical Center c_x	165.0	23.5
Optical Center c_y	125.9	21.7
Radial Distortion k_1	0.6	0.2
Radial Distortion k_2	-2.2	0.7
Tangential Distortion t_1	0	0.1
Tangential Distortion t_2	0	0.1

```
/**
 Syntax: calibrate <image_basename> <image_ending> <number of images>
    <corners per row> <corners per column>

 The program then reads the images
 image_basename0image_ending, ... , image_basenameNimage_ending
 (N=number of images) and outputs:
 - the camera calibration
 - the camera image overlayed with the corners found
 - the corners backprojected using their 3D scene corrdinates,
   the extrinsic and intrinsic camera matrix
 - the backprojected corners in undistorted images
**/



#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <ctype.h>
#endif

IplImage *image = 0, *threshold = 0, *undistort = 0;
IplImage *bw = 0;
CvPoint2D32f *cornersArray = 0;
char filename[128], base[128], end[8];
int  numImages, imageNum;
CvPoint3D32f *corners3d = 0;
CvVect32f distortion = 0;
CvMatr32f camera_matrix = 0;
CvVect32f translation_vectors = 0;
CvMatr32f rotation_matrices = 0;

void DrawCircles(CvArr *img, CvPoint2D32f *ptArray, int numPts, char
*windowName, int wait);
void InitCorners3d(CvPoint3D32f *corners3d, int xCorners, int yCorners, int
numImages);
void PrintIntrinsics(CvVect32f distortion, CvMatr32f camera_matrix);
void PrintMatrix(CvMatr32f matrix, unsigned int rows, unsigned int cols);

/*
 * Backprojection from World to Image coordinates
 */
CvPoint2D32f *ConvertWorldToPixel(CvPoint3D32f *pts3d, int numImages, int
*numPts, CvMatr32f cam, CvVect32f t, CvMatr32f r)
{
        int i, j, k;
        CvPoint2D32f *pts2d    = new CvPoint2D32f[numImages * 49];

        CvMat *C                 = cvCreateMat(3, 3, CV_32FC1);
        CvMat *point3D  = cvCreateMat(3, 1, CV_32FC1);
        CvMat *R        = cvCreateMat(3, 3, CV_32FC1);
        CvMat *T        = cvCreateMat(3, 1, CV_32FC1);

        CvPoint3D32f *pts3dCur = pts3d;
        CvPoint2D32f *pts2dCur = pts2d;
        for (i = 0; i < 3; i++)
                for (j = 0; j < 3; j++)
                {
                        CV_MAT_ELEM(*C, float, i, j) = cam[3 * i + j];
                }

        for (k = 0; k < numImages; k++) {
                for (j = 0; j < 9; j++) {
                        if (j < 3) CV_MAT_ELEM(*T, float, j, 0) = t[k*3 + j];
                        CV_MAT_ELEM(*R, float, j / 3, j%3) = r[k*9 + j];
                }

                for (i = 0; i < numPts[k]; i++)
                {
                        CV_MAT_ELEM(*point3D, float, 0, 0) = pts3dCur[i].x;
                        CV_MAT_ELEM(*point3D, float, 1, 0) = pts3dCur[i].y;
                        CV_MAT_ELEM(*point3D, float, 2, 0) = pts3dCur[i].z;


                        cvMatMulAdd(R, point3D, T, point3D); //rot and
translate
                        cvMatMul(C, point3D, point3D);     //camera

                        pts2dCur[i].x = CV_MAT_ELEM(*point3D, float, 0, 0) /
CV_MAT_ELEM(*point3D, float, 2, 0);
                        pts2dCur[i].y = CV_MAT_ELEM(*point3D, float, 1, 0) /
CV_MAT_ELEM(*point3D, float, 2, 0);
                }
                pts3dCur += numPts[k];
                pts2dCur += numPts[k];
        }

        cvReleaseMat(&point3D);
        cvReleaseMat(&T);
        cvReleaseMat(&R);

        return pts2d;
}

int main(int argc, char* argv[])
{
        int xCorners = 0;
        int yCorners = 0;
        int iImg;
        int width;
        int height;

        if (argc == 6) {
                sprintf(base, "%s", argv[1]);
                sprintf(end, "%s", argv[2]);
                numImages = atoi(argv[3]);
                xCorners = atoi(argv[4]);
                yCorners = atoi(argv[5]);
        } else {
                sprintf(base, "good\\image");
                sprintf(end, ".bmp");
                numImages = 12;
                xCorners = 7;
                yCorners = 7;
        }
        cornersArray = new CvPoint2D32f[xCorners*yCorners*numImages];
        corners3d = new CvPoint3D32f[xCorners*yCorners*numImages];
        int *cornersFound = new int[numImages];
        distortion = new float[4];
        camera_matrix = new float[9];
        translation_vectors = new float[3*numImages];
        rotation_matrices = new float[9*numImages];

        InitCorners3d(corners3d, xCorners, yCorners, numImages);

        printf( "Hot keys: \n"
        "\tESC - quit the program\n"
        "\tm - Mark corners\n"
                );

    cvNamedWindow("Image", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Undistorted", CV_WINDOW_AUTOSIZE);

        CvPoint2D32f *nextCornersArray = cornersArray;
        for (iImg = 0; iImg < numImages; iImg++)
        {
                // Load all the new images into there data structures
                sprintf(filename, "%s%d%s", base, iImg, end);
                image = cvLoadImage(filename);
                width = image->width;
                height = image->height;

                if (image->depth != IPL_DEPTH_8U) {
                        fprintf(stderr, "image, %s, depth not IPL_DEPTH_8U\n",
filename);
                        exit(1);
                }

                bw = cvCreateImage(cvSize(image->width, image->height),
IPL_DEPTH_8U, 1);
                cvConvertImage(image, bw);
                threshold   = cvCreateImage(cvSize(image->width,
image->height), IPL_DEPTH_8U, 1);

                // Feedback of images loaded for sanity
                cvShowImage("Image", image);

                // Start calibration of points
                cornersFound[iImg] = xCorners * yCorners;

                // nextCornersArray is a pointer into cornersArray to the last
slot open
                // this will overflow if cvchessboard ever gives us more
corners then expected
                int bResult = cvFindChessBoardCornerGuesses(bw, threshold,
NULL, cvSize(xCorners, yCorners), nextCornersArray, &cornersFound[iImg]);
                cvFindCornerSubPix(bw, nextCornersArray, cornersFound[iImg],
cvSize(5,5), cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_ITER, 100, 0.1));

                // more visual feedback to confirm correctness
                // cvWaitKey(0);
                DrawCircles(image, nextCornersArray, cornersFound[iImg],
"Image", -1);
                cvShowImage("Image", image);
                sprintf(filename, "%s%d%s%s", base, iImg,
"_findChessBoardCornerGuesses", end);
                cvSaveImage(filename, image);

                if (!bResult)
                {
                        fprintf(stderr, "Did not find expected number of
points... bailing\n");
                        cvWaitKey(0);
                        exit(1);
                }
                cvWaitKey(0);

                // update so that nextCornersArray points to the next open slot
in cornersArray
                nextCornersArray += cornersFound[iImg];
                cvReleaseImage(&image);
                cvReleaseImage(&bw);
                cvReleaseImage(&threshold);
        }

//      cvWaitKey(0);
        cvCalibrateCamera(numImages, cornersFound, cvSize(width, height),
                cornersArray, corners3d, distortion, camera_matrix,
translation_vectors,
                rotation_matrices, 0);

        PrintIntrinsics(distortion, camera_matrix);

        CvPoint2D32f *backprojPts2d = ConvertWorldToPixel(corners3d, numImages,
cornersFound, camera_matrix, translation_vectors, rotation_matrices);

        for (iImg = 0; iImg < numImages; iImg++) {
                // Load all the new images into there data structures
                sprintf(filename, "%s%d%s", base, iImg, end);
                image = cvLoadImage(filename);
                undistort = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U,
3);
        cvUnDistortOnce(image, undistort, camera_matrix, distortion, 1);

                DrawCircles(image, &backprojPts2d[iImg*cornersFound[iImg]],
cornersFound[iImg], "Image", 1);

                DrawCircles(undistort, &backprojPts2d[iImg*cornersFound[iImg]],
cornersFound[iImg], "Undistorted", 1);

                sprintf(filename, "%s%d%s%s", base, iImg, "_backproject", end);
                cvSaveImage(filename, image);

                sprintf(filename, "%s%d%s%s", base, iImg,
"_backproject_undistorted", end);
                cvSaveImage(filename, undistort);

                cvWaitKey(0);
                cvReleaseImage(&image);
                cvReleaseImage(&undistort);
        }

        cvWaitKey(0);

    cvDestroyWindow("Image");
        delete cornersArray;
        delete corners3d;
        delete cornersFound;
        delete distortion;
        delete camera_matrix;
        delete translation_vectors;
        delete rotation_matrices;

    return 0;
}
/* DrawCircles
 * Takes number of corners on x axis and y axis + number of images
 * Buids an array of corners.  Assumes all corners on z = 0 plane.
 */
void DrawCircles(CvArr *img, CvPoint2D32f *ptArray, int numPts, char
*windowName, int wait=0)
{
        CvPoint pt;

        for (int i = 0; i < numPts; i++)
        {
                int clr = (int)((float)i / numPts * 255.0);
                pt.x = (int)ptArray[i].x;
                pt.y = (int)ptArray[i].y;
                cvCircle(img, pt, 3, CV_RGB(255-clr,0,clr), CV_FILLED);
                if (windowName)
                {
                        cvShowImage(windowName, img);
                        if (wait != -1)
                                cvWaitKey(wait);
                }
        }
}

/* InitCorners3d
 * Takes number of corners on x axis and y axis + number of images
 * Buids an array of corners.  Assumes all corners on z = 0 plane.
 */
void InitCorners3d(CvPoint3D32f *corners3d, int xCorners, int yCorners, int
numImages)
{
        int n, i, j;
        int nStride = xCorners * yCorners;

        for (n = 0; n < numImages; n++)
        {
                for (i = 0; i < xCorners; i++)
                {
                        for (j = 0; j < yCorners; j++)
                        {
                                corners3d[n*nStride + i*7+j].x = (float)i;
                                corners3d[n*nStride + i*7+j].y = (float)j;
                                corners3d[n*nStride + i*7+j].z = (float)0;
                        }
                }
        }
}

void PrintIntrinsics(CvVect32f distortion, CvMatr32f camera_matrix)
{
        int i;
        if (distortion)
        {
                printf("Distortion Coefficients:\n");
                printf("%4.4f ", distortion[0]);
                printf("%4.4f ", distortion[1]);
                printf("%4.4f ", distortion[2]);
                printf("%4.4f ", distortion[3]);
                printf("\n");
        }

        if (camera_matrix)
        {
                printf("Camera Matrix:\n");
                PrintMatrix(camera_matrix, 3, 3);
        }
}

void PrintMatrix(CvMatr32f matrix, unsigned int rows, unsigned int cols)
{
        int m, i, j;
        if (matrix)
        {
                m = 0;
                for (j = 0; j < rows; j++)
                {
                        for (i = 0; i < cols; i++)
                        {
                                printf("%4.4f ",matrix[m++]);
                        }
                        printf("\n");
                }
        }
}
```