ME/CS 132A_Winter 2015

Assignment 1 - Use OpenCV for camera calibration

Theory

For the distortion OpenCV takes into account the radial and tangential factors. For the radial factor one uses the following formula:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

So for an old pixel point at (x,y) coordinates in the input image, its position on the corrected output image will be (x_{corrected} y_{corrected}). The presence of the radial distortion manifests in form of the “barrel” or “fish-eye” effect.

Tangential distortion occurs because the image taking lenses are not perfectly parallel to the imaging plane. It can be corrected via the formulas:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

So we have five distortion parameters which in OpenCV are presented as one row matrix with 5 columns:

$$Distortion_{coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Now for the unit conversion we use the following formula:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Here the presence of w is explained by the use of homogeneous coordinate system (and w=Z). The unknown parameters are f_x and f_y (camera focal lengths) and (c_x, c_y) which are the optical centers expressed in pixels coordinates. If for both axes a common focal length is used with a given aspect ratio a (usually 1), then f_y=f_x*a and in the upper formula we will have a single focal length f, which you learned in class. The matrix containing these four parameters is referred to as the camera matrix. While the distortion coefficients are the same regardless of the camera resolutions used, these should be scaled along with the current resolution from the calibrated resolution.

The process of determining these two matrices is the calibration. Calculation of these parameters is done through basic geometrical equations. The equations used depend on the chosen calibrating objects. Currently OpenCV supports three types of objects for calibration:

- Classical black-white chessboard
- Symmetrical circle pattern
- Asymmetrical circle pattern

Basically, you need to take snapshots of these patterns with your camera and let OpenCV find them. Each found pattern results in a new equation. To solve the equation you need at least a predetermined number of pattern snapshots to form a well-posed equation system. This number is higher for the chessboard pattern and less for the circle ones. For example, in theory the chessboard pattern requires at least two snapshots. However, in practice we have a good amount of noise present in our input images, so for good results you will probably need at least 20 good snapshots of the input pattern in different positions.

Goal:

Learn to use OpenCV to:

(1)Open("load") an image and display that image;

(2)Take input from Camera, Video or Image file list;

(3)calibrate image for camera calibration;

- Determine the distortion matrix
- Determine the camera matrix
- Save the results into XML/YAML file

The codes for Part A, Part B, Part C and answers for Part D, the screenshot for Part A , the 25 bmp images you saved and used for Part C, and the XML file with parameter information should be archived with name "your name _ME132_HW1" and submitted via Wiki. A hard copy should also be turned in every Thursday before the lecture starts.

Getting Started----Installing the OpenCV Library

Step1: Click on the following link and download **OpenCV-2.2.0-win32-vs2010.exe** http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.2/

When installing, by default, you can make you download path as "C:\OpenCV 2.2". Remember to select "Add OpenCV to the system PATH for all users". If it doesn't work, add to the system path manually by yourself. Finding the following path:

"control panel->system->advanced system setting->system->environmental variables->path".

Then add "opencv/bin" to the path. The bin file should be in your installation file of OpenCV.

Step2: Configure the setup

In Visual Studio, it is necessary to create a project and to configure the setup so that the necessary libraries are *linked*, and the preprocessor will search your OpenCV *include* directories for header files. Once you have done this, you create a new C++ file and start your first program.

To be concrete, you need to create a Win32 Console Application in VS2010 environment.

- In "Configuration Properties-VC++ Directories-Include Directories", add the following 3 directories: C:\OpenCV2.2\include; C:\OpenCV2.2\include\OpenCV; C:\OpenCV2.2\include\OpenCV2.
- In "Configuration Properties-VC++ Directories-Library Directories", add C:\OpenCV2.2\lib.
- In "Linker-Input-Additional Dependencies", add opencv_highgui220d.lib; opencv_core220d.lib; opencv_imgproc220d.lib;

Part A- Open(load) an image and display that image (10%)

This part is relatively easy. The necessary header file you need to include is "highgui.h". Also here are some useful functions in OpenCV Library:

```
cvLoadImage(img_file_name,1);
cvNamedWindow(img_file_name,CV_WINDOW_AUTOSIZE);
cvShowImage(img_file_name,img);
cvWaitKey(0);
cvReleaseImage(&img);
cvDestroyWindow(img_file_name);
```

Requirements: In this first program, you are required to load an image into memory and display it on the screen. Remember to get a screenshot of the window displaying that image. The raw image is provided with the file name "First_Image".

Part B - Take input from Camera, Video or Image file list (20%)

Vision can mean many things in the world of computers. In some cases we are analyzing still frames loaded from elsewhere. In other cases we are analyzing video that is being read from disks. In still other cases, we hope to work with real-time data streaming in from some kind of camera device. In Part A, you know how to load a static image from your computer, in this section, you need to get real-time video stream from a camera, whether a USB webcam from outside your computer or the camera set on your own computer. Also, you need to grasp how to save image files captured by your camera anytime you press a key on your keyboard.

In OpenCV, more specifically, the HighGui portion of the OpenCV library provides us with an easy way to handle this situation. Calling cvCreateCameraCapture() is a routine way to do this. However, if we hope to set multiple cameras in our program and conveniently select any camera we want, it will be much more helpful for Computer Vision problems later. In OpenCV 2.2, we turn to DirectShow to realize our hope to select from multiple cameras.

Download the zip file of CameraDS using the following link. After decompressing the file, put DirectShow folder into your project folder, like \Project\First_Project, add the five files Camera.cpp, CameraDS.h, stdafx.cpp, stdafx.h, targetver.h into \Project\First_Project \First_Project. Then add the cpp file CameraDS.cpp to your project which means your own cpp file and this cpp file should be in the same folder. Finally, remember to include necessary directories. In "Configuration Properties-VC++ Directories-Include Directories", add DirectShow \include.

https://drive.google.com/folderview?id=0B81NqUINfbTYTVJUbl9Ld1Y4ams&usp=drive_web&ddrp=1#

Requirements: In the second program, you need to get real-time video input from the camera on your own computer. While pressing "m" or "M" on your keyboard, the image in that video is captured and saved to your computer. For convenience in Part C, please print out the chessboard picture "Chessboard_10_7" and hold it when you are using your camera to get video input(So we can see that is you). Then save 25 different images with bmp format and name those images as "IMG_SAVED000", "IMG_SAVED001"..."IMG_SAVED024".

A possible version of the whole codes are provided as follows but you still need to program and modify a little bit by yourself.

```
#include"stdafx.h"
#include<cv.h>
#include<highgui.h>
#include<stdio.h>
#include"CameraDS.h"


#define CAP_WIDTH 320
#define CAP_HEIGHT 240

int main()
{
 IplImage* capimg;
 CvCapture* capture;
 int loop =1;
 int file_idx=0;
 int key;
 int DeviceCamCount;

 char filename[256];
 char fileidx[10];

 int CamCount=0;
 int CamIndex[10];
 CCameraDS *pCamDS =NULL;

 pCamDS= new CCameraDS();
 DeviceCamCount=pCamDS->CameraCount();
 if(DeviceCamCount == 0)
 {
   printf("No Camera detected...\n");
            return(-1);
 }

 for( int i=0,j=0; i<DeviceCamCount;i++)
 {
  char szCamName[1024];
```

```
            int retval=pCamDS->CameraName(i,szCamName,sizeof(szCamName));
            if(retval>0)
            {
// select the camera, you probably need to modify codes here.
                        printf("Camera #%d's name is '%s'.\n",i,szCamName);
                         if(!strcmp(szCamName,"USB .."))

                        {
                         CamIndex[j++]=i;
                         CamCount++;
                        }
            }
            else
            {
                        printf("Cannot get camera #%d's name.\n",i);
            }
 }

 if(pCamDS->OpenCamera(CamIndex[0],false,CAP_WIDTH,CAP_HEIGHT)==false)
 {
  printf("Camera do not support w=%d:h=%d\n",CAP_WIDTH,CAP_HEIGHT);
            return(-1);
 }

 cvNamedWindow("camera0",1);

 while(loop)
 {
             capimg = pCamDS ->QueryFrame();
            cvShowImage("camera0",capimg);

            key=cvWaitKey(5);
            if(key==27) // 27 is the ASCII code of "ESC", when you press ESC, you exit the program
            {
             loop=0;
            }
            else if(FILL YOUR CODES) // press "m" or "M" to save the image(bmp file)
            {
             FILL YOUR CODES
            }
 }

 if(pCamDS)
   {
  pCamDS->CloseCamera();
            delete pCamDS;
   }
 cvDestroyWindow("camera0");
 return(0);
}
```

Part C - Use OpenCV for Camera Calibration (50%)

In general, any object can be used as a calibration object. OpenCV opts for a chessboard which ensures that there is no bias toward one side or the other in measurement. Also, the resulting grid corners lend themselves naturally to the subpixel localization functions.

Given an image of a chessboard(or a person holding a chessboard picture), use OpenCV function cvFindChessboardCorners(img,boardsize,imageCorners) to locate the corners of the chessboard. The first argument "image" is an 8-bit grayscale(single-channel) image. It is

actually the input image with a chessboard on it. The second argument "boardsize" indicates how many corners are in each row and column. It is counting the interior corners. Its type is Cvsize(M,N), Here is the function definition:

```
int cvFindChessboardCorners(  const void* image, Cvsize boardsize,CvPoint2D32f* imagecorners);
```

The corners returned by cvFindChessboad Corners() are only approximate, and you should use OpenCV function cornerSubPix() to get exact values of locations of corners. Neglecting to call this refinement after you first locate the corners will probably cause substantial errors in calibration. You will probably write exactly the following codes when calling this function.

```
cornerSubPix(image, imageCorners,
                    cvSize(11,11),
                    cvSize(-1,-1),
                    cvTermCriteria(cv::TermCriteria::MAX_ITER + cv::TermCriteria::EPS,30, 0.1));
```

Next, OpenCV function cvDrawChessboardCorners() will label the corners onto an image you provide. Here is the function definition:

```
void   cvDrawChessboardCorners( CvArr* image, CvSize boardsize, CvPoint2D32f* corners, int pattern_found);
```

Finally, call calibrateCamera() for camera calibration. This OpenCV function will take the chessboard images and output the camera intrinsic matrix and distortion coefficient vector. The following codes are what you need to write when calling this function.

```
calibrateCamera(objectPoints, // the 3D points
                imagePoints, // the image points
                imageSize,   // image size
                cameraMatrix,// camera intrinsic matrix
                distCoeffs,  // distortion coefficient matrix
                rvecs, tvecs,// rotation and translation vector
                0);
```

After you have done with the following functions, you need to save your parameter information(camera intrinsic matrix and distortion coefficient vector) to a XMY file. Codes are as follows:

```
FileStorage fs("test.yml", FileStorage::WRITE);
fs << "intrinsic" << cameraMatrix;
fs << "distcoeff" << distCoeffs;
```

Requirements: In the third and last program, use the 25 images in get in Part B which are "IMG_SAVED000"..."IMG_SAVED024" and run your codes you write to get calibrated images and XMY file which includes the parameter information. Just take the 25 images as input and see what happens when you successfully run you codes in Part C.

Putting all above together, a possible version of the codes of the main part without the header files is as follows. The highlighted red part of the codes is where you need to write codes for above OpenCV functions or make some changes to make the whole program work.

```
#define CAP_WIDTH  640
#define CAP_HEIGHT 480

std::vector<std::vector<cv::Point3f>> objectPoints;
```

```
std::vector<std::vector<cv::Point2f>> imagePoints;

cv::Mat cameraMatrix;
cv::Mat distCoeffs;
cv::Mat map1,map2;

int main()
{
  cv::Mat img;
  cv::Mat img2;
  IplImage* capimg;
  int  DeviceCamCount = 0;
   CCameraDS *pCamDS   = NULL;

  int       CamCount = 0;
  int       CamIndex[10];

   pCamDS  = new CCameraDS();
   DeviceCamCount = pCamDS->CameraCount();

   int i;
   int loop  = 1;
   int key;
   int file_idx = 0;
   char filename[256];
   char fileidx[10];

        //Define the variable "boardsize" here;
   cv::Size imageSize(CAP_WIDTH, CAP_HEIGHT);
   std::vector<cv::Point2f> imageCorners;
   std::vector<cv::Point3f> objectCorners;

  for (int i=0; i<boardSize.height; i++)
   {
      for (int j=0; j<boardSize.width; j++)
      {
         objectCorners.push_back(cv::Point3f(i, j, 0.0f));
      }
   }

   while ( loop )
   {
        //Find the chessboard bmp images here;

      printf("IMG = %s\n", filename);
      img = imread( filename, 0 );
      if ( img.empty())
         loop = 0;
      else
      {
        //Find the chessboard corners here;
        //Find exact values of locations of corners here;

        if (imageCorners.size() == boardSize.area())
         {
            imagePoints.push_back(imageCorners);
            objectPoints.push_back(objectCorners);
         }
         img.copyTo( img2 );

        //Draw chessboard corners here;

         imshow( "image", img2 );
```

```
            key = cvWaitKey(400);
            if ( key == 27 )
            {
                loop = 0;
            }
        }
    }

    std::vector<cv::Mat> rvecs, tvecs;
        //Use calibrateCamera() for camera calibration here;
        //Save parameter information to XML file here;
}
```

Part D - Understand better the camera parameters (20%)

Based on what you get from Part C, please answer the following questions:
a-What is the aspect ratio of the image?
b-What is the horizontal and vertical resolution of your camera?
c-What is the field of view of the camera?
d-What does each element mean in your 1*5 distortion coefficient vector? What can you know about the radial distortion and tangential distortion of your camera?