UNIVERSITY OF AVEIRO
DEPARTAMENT OF ELECTRONICS, TELECOMUNICATIONS AND INFORMATICS

Computer Vision (2015/2016)

Lecture 06

Stereo Vision

- Calibration of a stereo rig
- Epipolar geometry
- Rectification of stereo images

1. Chessboard calibration

Compile and test the file chessboard.cpp (similar to the one used in the last lecture). This code detects corners in a chessboard pattern using `openCV` functions and shows the results of the detection for a series of images.

Rename the file (cheesboard1.1cpp for example) and modify the code to allow for detection of corners in a series of stereo pair images (use the provided right images with name rightxx.jpg).

Fill the necessary matrices with the correct value to calibrate the stereo pair. The objective is to define 4 matrices: `ipts1`, `ipts2`, `opts` and `npoints` with, respectively, 2D pixel coordinates of the corners in the left and right image (2 coordinates per row), 3D point coordinates of the chessboard corner (3 coordinates per row) and number of points to consider for each pair (scalar value per row).

Analyze the code for filling the 3D coordinates of the pattern. Imagine that you use another pattern with a different size values what should you do to get the distance correctly evaluated?

Note: It is possible to improve the precision of the corner detection by using the `cvFindCornerSubPix` function after the call to `cvFindChessboardCorners`.

2. Stereo calibration

Calibrate the stereo pair using the function `cvStereoCalibrate`. You can access the documentation of the 3D reconstruction and calibration functions in `openCV` help:

http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Use the default parameters presented in the documentation for the stereo calibration, except for the last parameter that you should set to `CV_CALIB_SAME_FOCAL_LENGTH`, meaning that the algorithm will consider the same focal length for both camera and that no guess is provided for the other parameters.

The output of the stereo calibration will be the following matrices:

```
cv::Mat intrinsic_matrix_1;
cv::Mat distortion_coeffs_1;
cv::Mat intrinsic_matrix_2;
cv::Mat distortion_coeffs_2;
cv::Mat R;
cv::Mat T;
cv::Mat E;
cv::Mat F;
```

After a successful calibration, save the matrices in an xml file using the file storage functions to avoid recalibration of the stereo rig each time.

```
cv::FileStorage fs("stereoParams.xml", cv::FileStorage::WRITE);

if (!fs.isOpened())
    {
    std::cout << "Failed to open stereoParams.xml" << std::endl;
    return 1;
    }
else
    std::cout << "Writing camera parameters" << std::endl;

fs << "cameraMatrix_1" <<  intrinsic_matrix_1;
fs << "distCoeffs_1" <<  distortion_coeffs_1;
fs << "cameraMatrix_2" <<  intrinsic_matrix_2;
fs << "distCoeffs_2" <<  distortion_coeffs_2;
fs << "rotation" << R;
fs << "translation" <<  T;
fs << "essential" << E;
fs << "fundamental" << F;
fs.release();
```

Try to repeat the process with the other set of images available.

Note: Given the large number of parameters to be evaluated the stereo calibration process might not always give reliable results depending on the images. It is possible to ease the process by calibrating individually each camera (intrinsics and extrinsics parameters) with the function `cvCalibrateCamera` (see previous lecture) and indicate the stereo calibration algorithm to use these values as guess or as fixed by changing the last parameter of the function (`CV_CALIB_USE_INTRINSIC_GUESS` or `CV_CALIB_FIX_INTRINSIC`**)**.

3. Lens distortion

In a new file, read the distortion parameters of the cameras (function `cvLoad`), select a stereo pair of images from the pool of calibration images and present the undistorted images (image with the lens distortion removed) using the function `cvUndistort` to compute the new images.

4. Epipolar lines (opcional / Homework)

Modify the previous example to show only the undistorted images. Add the possibility to select a pixel in each image using the following code to set a callback to be called for handling mouse events.

```
void mouseHandler(int event, int x, int y, int flags, void* param)
  {
    switch(event){
      case CV_EVENT_LBUTTONDOWN:
       break;
```

```
    }
  }
```

Do not forget to associate the callback to each window using the following code:

```
cv::SetMouseCallback( "Window", mouseHandler);
```

Start by adding a callback to each window and writing down the coordinates of the selected pixel. Do not forget to add a `cvWaitKey(-1)` at the end of the program.

Use the function `computeCorrespondEpilines` to draw the corresponding epipolar line for each selected point (use `cvLine`). The epipolar line of points in the left image should be drawn in right image and vice versa. To compute the epipolar lines, use the fundamental matrix estimated during the stereo calibration. Note that the function `computeCorrespondEpilines` returns the 3 coefficients `(a,b,c)` of the corresponding epipolar line for a given point define as `ax+by+c=0`.

5. Image rectification

Select a pair of stereo images and use the following OpenCV functions to generate the rectified images (corresponding epipolar lines in the same rows in both images):

`cvStereoRectify`: this function computes the rotation and projection matrices that transform both camera image plane into the same image plane, and thus with parallel epipolar lines. The size of the output matrices R1, R2, P1, P2 is respectively 3x3 and 3x4.

`cvinitUndistortRectifyMap`: This function computes the transformation (undistortion and rectification) between the original image and the rectified image. The output arrays mx1 and mx2 are a direct map between the two images, for each pixel in the rectified image, it maps the corresponding pixel in the original image.

`cvRemap`: apply the transformation between two images using the provided map of x/y coordinates.

The several matrices to be used can be defined as:

```
cv::Mat map1x;
cv::Mat map1y;
cv::Mat map2x;
cv::Mat map2y;

cv::Mat R1;
cv::Mat R2;

cv::Mat P1;
cv::Mat P2;
cv::Mat Q;
```

Visualize the resulting images, and draw lines in rows (for example at each 25 pixels) to evaluate visually if corresponding pixels are in corresponding lines.

Homework:

Modify the code to make it interactive as in problem 4. By clicking in a point in an image, the corresponding row will appear in the other image.

6. PCL installation (homework)

We will use `pcl` as well as `openCV` on next lectures. You should have a tutorial example running on your computer. Install `pcl` in your linux distribuition as explained in:

http://pointclouds.org/downloads/linux.html

Check if the installation is up and running and configure the necessary makefiles with the tutorial in:

http://pointclouds.org/documentation/tutorials/using_pcl_pcl_config.php#using-pcl-pcl-