

Lab 6 - Stereo Vision

- Stereo Vision.
- Calibration of a stereo rig.
- Epipolar geometry.
- Rectification of stereo images

6.1 Chessboard calibration

Compile and test the file `chessboard.py` (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 (`Stereo_exe_1.py` 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 3 matrices: `left_corners`, `right_cornes`, `objPoints` 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).

6.2 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.

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

```
np.savez("stereoParams.npz",
        intrinsics1=intrinsics1,
        distortion1=distortion1,
        intrinsics2=intrinsics2,
        distortion2=distortion2,
        R=R, T=T, E=E, F=F)
```

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`).

6.3 Lens distortion

In a new file, read the distortion parameters of the cameras (function `np.load`), 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.

6.4 Epipolar Lines

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.

```
def mouse_handler(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        print("left click")
```

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

```
cv2.setMouseCallback("Window", mouse_handler)
```

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 $ax+by+c=0$.

You may use the following code to access the point coordinates and compute the epilines:

```
p = np.asarray([x,y])
epilineR = cv2.computeCorrespondEpilines(p.reshape(-1,1,2), 1, F)
epilineR = epilineR.reshape(-1,3)[0]
```

and define random colors:

```
color = np.random.randint(0, 255, 3).tolist()
```

6.5 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:

```
R1 = np.zeros(shape=(3,3))
R2 = np.zeros(shape=(3,3))
P1 = np.zeros(shape=(3,4))
P2 = np.zeros(shape=(3,4))
Q = np.zeros(shape=(4,4))
```

For stereo rectification and remapping you might use the following code:

```
cv2.stereoRectify(intrinsics1, distortion1, intrinsics2, distortion2, (width, height), R,
T, R1, R2, P1, P2, Q, flags=cv2.CALIB_ZERO_DISPARITY, alpha=-1, newImageSize=(0,0))
```

```
# Map computation
```

```
print("InitUndistortRectifyMap");
map1x, map1y = cv2.initUndistortRectifyMap(intrinsics1, distortion1, R1, P1,
(width,height), cv2.CV_32FC1)
map2x, map2y = cv2.initUndistortRectifyMap(intrinsics2, distortion2, R2, P2,
(width,height), cv2.CV_32FC1)
```

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.

Open3D installation (homework)

We will use open3D as well as openCV on next lectures. You should have a tutorial example running on your computer. Install open3D as explained in:

http://www.open3d.org/docs/release/getting_started.html

Check if the installation is up and running by running some tutorials in the same page.