EN.601.461/661

Computer Vision

Assignment #6

Due date: October 29th 11:59PM

5 marks (10% per day late submission)

Instructions: Submit your Python files and images on Gradescope. Please follow the instructions for each file. Test images can be found on Piazza (Ressources). An autograder will be available to ensure that the input/output of your functions complies with the guidelines. Use it to validate your functions. The autograder will only test if your code uses correct input/output. It will not test if your code is produces correct results. You will need to use OpenCV to for the following: Image reading, corner detection (including subpixel accuracy), camera calibration, SIFT (detection and matching), fundamental matrix and epipolar lines. Refrain from displaying images in your functions. There is no “main” function, you should be able to call your functions yourself (from you own main file or from a prompt).

You will need to zip all your files according to the following directory structure:

$$\begin{tikzcd}
\text{assignment6} \arrow[r] \arrow[dr] \arrow[ddr] & \text{you\_python\_files\_here} & \\
& \text{calibration\_images} \arrow[r] & \text{your\_calibration\_images\_here} \\
& \text{test\_images} \arrow[r] & \text{your\_test\_images\_here}
\end{tikzcd}$$

Collect Data (1pt)

You will need to collect your own data from cameras provided in Wyman 170. Wyman 170 has six UR-5 robots. Three of them will be mounted with grippers and with Intel R200 RGBD cameras. Although, you don’t really need the robots for this assignment (a camera is fine), you will need to use them for the other assignments and the sooner you familiarize yourself with them, the better. These robots are easy to use. A video will be posted on Monday (the lab was being used by another course until Friday) on how to use them and the Linux computers attached to them.

For those who want to start early, a practice data set will be posted on Piazza. You can use them for now, but you will need to collect your own images at some point. In fact, I encourage you to implement and validate your calibration code before heading to the lab and collect your own data (I’ll explain why below).

You will need to collect between 30-40 images of a checkerboard, but I recommend you collect a few more in cases that some images may need to be discarded. The checkerboards in the lab have 12x9 corners separated by 20mm squares. If you have your calibration code working, I recommend you try it while you are in the lab to make sure that your results are good. Otherwise you may have to go back to the lab if you realize that your images have poor quality (and produce poor results).

Expert Challenge (0.5pt)

You will earn these marks if the images you collected (at least 30 images) produce better camera calibration results than the dataset provided. Your dataset will need to produce a lower overall root mean squared error (RMS). See the OpenCV documentation on how to obtain overall RMS on camera calibration.

CalibrateCamera.py (1pt)

This function will load calibration images and calculate the calibration matrix of the camera. The function has the following definition:

```
def CalibrateCamera( path, width, height, size ):
```

`path`: relative path (string) to the directory containing the images. For example, "calibration_images".
`width`: the width of the checkerboard (number of internal corners). For example 12.
`height`: the height of the checkerboard (number of internal corners). For example 9.
`size`: The size of the squares on the checkerboard (in meter). For example 0.02.
`return`: The function returns the overall RMS, the camera calibration matrix K and the coefficient of the non-linear distortion (see OpenCV documentation).

This function must 1) read the image, 2) find the corners to subpixel accuracy (read the OpenCV documentation) and 3) calibrate the camera and return the parameters. Do not hardcode the filename of the images or the number of files. Instead use `os.listdir` to obtain the list of files in a given directory (that's why the `path` argument is there for).

RectifyImage.py (0pt)

This method is provided to remove the non-linear distortion from images. Use this method to rectify the images before extracting keypoints. From this point, you should not use any of the calibration images. Instead capture two images of a scene (i.e. in the lab) by moving the robot and save these images in the test_images directory.

def Rectify(img, K, dist):

`img`: The camera image.
K: The calibration matrix. Returned from CalibrateCamera.
`dist`: The non-linear distortion coefficients. Returned from CalibrateCamera.
`return`: The rectified image.

ExtractKeypoints.py (0.5pt)

As in the previous assignment, you will need to detect and match SIFT keypoints. You can reuse some of the code from your previous assignment, with the difference that the detection and matching must be implemented in the same function. Do not use calibration images (they are not "SIFT friendly"). The definition must be

```
def ExtractKeypoints(img1, img2, N):
```

`img1`: First rectified image. An numpy.ndarray image of uint8.
`img2`: Second rectified image. An numpy.ndarray image of uint8.
N: The number of matches to return.

`return`: The 2D image coordinates of the matched points (i.e. x, x'). These should be two 2xN numpy.ndarray of int32 values.

This function detects and matches SIFT keypoints between two *rectified* images. It returns the N strongest matches. Unless for debugging, do not display any image. The SIFT keypoints should be created with default parameters (3 octaves, 0.04 contrast threshold, 10 edge threshold and 1.6 sigma).

FundamentalMatrix.py (0.5pt)

This function takes two sets of points, (i.e. x, x') and calculate the fundamental matrix.

def FundamentalMatrix(x, xp):

`x`: The 2D image coordinate from the first image.
`xp`: The 2D image coordinates from the second image.
`return`: The fundamental matrix.

This is a straightforward method that calculate and returns the fundamental matrix.

EpipolarLines.py (0.5pt)

Calculate and return the epipolar lines of the points in both images.

```
def EpipolarLines( x, xp, F ):
```

`x`: The 2D image coordinate from the first image.
`xp`: The 2D image coordinates from the second image.
F: The fundamental matrix.
`return`: The epipolar lines. These should be two 3x1xN numpy.ndarray of type float32 (see OpenCV Documentation).

DisplayKeypointsandLines.py(1pt)

Display the keypoints and epipolar lines of both images.

```
def DisplayKeypointsandLines( img1, img2, x, xp, lines1, lines2 ):
```

`img1`: The first image.
`img2`: The second image.
`x`: The 2D image coordinate from the first image.
`xp`: The 2D image coordinates from the second image.
`lines1`: epipolar lines of the first image.
`lines2`: epipolar lines of the second image.

Reminder: By submitting your assignment you acknowledge that you have read the JHU Academic Misconduct Policy and that you are the author of 100% of the code submitted. Furthermore, distributing or sharing your assignments (whereas online, with classmates or students that will take this course in future semesters) is not authorized and will be sanctioned with a course grade “F”.