HOMEWORK ASSIGNMENT #3

DUE: Saturday, October 10, 2020, 5:00 P.M.

CSCI 677: Advanced Computer Vision, Prof. Nevatia

Fall Semester, 2020

This is a programming assignment to implement a simplified version of "structure from the motion" pipeline. We will focus on reconstructing from just a pair of calibrated camera images. The goal, given such a pair, is to reconstruct the 3-D positions of a set of matching points in the images and also to infer the camera extrinsic parameters. All results can be in a coordinate frame aligned with the first camera.

A common pipeline for SFM is to compute keypoint features, compute local matches, select globally consistent matches and compute the essential matrix, decompose the essential matrix into a rotation matrix and translation vector, compute camera matrices from these and triangulate to compute 3D point positions. A final step could be to do bundle adjustment for refining the results but we will ignore this step. The output will be a sparse set of 3-D points.

You may implement these steps on your own if you wish, but OpenCV provides functions for most steps so you may want to take advantage of these. There are also many open sources of SFM code available on the web; you are free to consult these, if you find them helpful, but not to use them directly to complete this assignment.

A list of possibly helpful OpenCV functions is provided for your convenience below. You may also use any other functions in OpenCV that you find useful.

1. *Keypoint feature extraction.* Use the SIFT operator by cv2.xfeatures2d.SIFT_create()

   https://docs.opencv.org/4.3.0/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html

   https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html

2. *Feature matchings*:

   - FLANN Matcher:

     https://docs.opencv.org/3.4/dc/de2/classcv_1_1FlannBasedMatcher.html

Matching can be done by using a "brute force" matcher available in OpenCV but we recommend using FLANN; it uses a tree-structure for faster matching but may miss some good matches.

For FLANN you can use the following parameters:
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)

It is suggested that you filter the local matches by performing a **Ratio Test**.
Ratio test means: the best match is better than the second-best match by at least some threshold. For example, second-best match < 0.8 * best-match (you could start from t=0.8 as the threshold). Set an ideal number for `t` so that ratio test can be performed (recommend: 0.5 < t < 1).

Feature matcher takes as input a parameter `k` (https://docs.opencv.org/3.4/db/d39/classcv_1_1DescriptorMatcher.html#a378f35c9b1a5dfa4022839a45cdf0e89) which gives the number of matches per query point. You could set `k=2`.

3. *findEssentialMat* function computes the essential matrix, given a list of match points; it also can use RANSAC for robust matching.
For RANSAC computation, you can use prob=0.999, threshold=1.0.
(https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga13f7e34de8fa516a686a56af119624)

4. *recoverpose* function decomposes an essential matrix into rotation and translation components. It also selects among four possible solutions by considering reconstruction of matched points. The camera matrices can be easily constructed from the recovered rotation and translation parameters.

5. *undistortpoints* may be used to convert image coordinates to normalized coordinates. This is a required step before running triangulation. It is also easy to write this function on your own.

6. *triangulatepoints* may be used to compute intersections of rays, given matching points and camera matrices. As triangulation involves solving linear equations, it may be complicated to write on your own. Note that the results may be output in homogeneous coordinates which should be converted to regular coordinates for output. This function also expects normalized coordinates as input so you need to convert from image coordinates.

Image Data:

Assignment folder contains a HW3_data folder with 3 images. Intrinsic parameters are provided in a file called `intrinsics.txt`. Show results for all 3 pair choices (0-1, 1-2, 0-2); you are not asked to integrate the results of the three pairs.

Assignment Structure:

We have provided `sfm_student.py` which contains the class `SFMSolver`. This contains some useful functions for visualization and some other methods which need to be filled by the student. After everything is filled correctly, the following should be enough to run the sfm pipeline:

```
sfm_solver = SFMSolver(*args)
sfm_solver.run()
```

Note that the `sfm_student.py` is only meant to give you a guideline. You are free to use your own structure if the given template feels restrictive.

Meshlab Installation:
Meshlab ((http://www.meshlab.net/)) is a software to visualize 3D point clouds. Meshlab provides binaries for Windows and Mac. For Linux systems, you can download from snap store using:

```
$ snap install meshlab
$ meshlab
```

Output and Visualization:

Your program should output results at the various intermediate steps. In particular, it should output the computed camera matrices, rotation and translation parameters and point positions.

You should also display the results graphically. You can display some of the matches, before and after applying RANSAC. The reconstructed 3-D points can be displayed from a viewpoint other than that of the original images, say halfway between a pair of cameras. There may be functions in OpenCV to facilitate such visualization but it is also easy to write your own (compute a new projection matrix and apply to the computed points).

We provide a member method `write_simple_obj()` to save output into an .obj file. You may visualize the output by open-source software Meshlab. After you open the output, you might see "nothing" on the screen. Tweak the point size (marked in red circle in the following figure), and the shape is easily visible. As you might need to zoom-in and rotate the object to see the shape, here is a youtube tutorial on how to navigate in meshlab: https://youtu.be/Sl0vJfmj5LQ

What to Submit?

You should submit the following.

1. A brief description of the programs you write (include the source listing). Your program should be commented clearly.

2. Show the numerical (rotation, translation, projection matrix) and visual (matched features, pruned features and reconstructed points) results of intermediate steps. Use your judgment in how to display or record these results. As we provided images in the dataset, please show results for all 3 pair choices (0-1, 1-2, 0-2), and your optimal parameter (t for ratio test).

3. An analysis of your test results: how well does the method work? How would you try to improve the output quality? Provide some thoughts or explanations.