EN.600.461/661 – Computer Vision
Fall 2020
Homework #2
Due: 11:59 PM, Sunday, October 25, 2020

In this assignment, you will explore 2D and 3D image transformations. In part 1, you will write Python code to detect discriminating features in an image and find the best matching features in other images. Then you will estimate a global transformation between the image pair using your matched features and then warp one image towards the other to stitch them together as a panorama. You will also use the same code to detect and recognize the presence of a small set of examples in a larger image.

In the second part of the assignment, you will perform camera calibration. For extra credit, you can calibrate your own camera and/or show mosaics or object detection from images that you've taken yourself.

You should submit 4 Python notebooks, 2 for each of the two sections of questions in each of the two parts of this assignment.

You will get at most partial credit for using a "magic" function to answer any questions where you should be constructing your own code to answer them. If you are unsure of an allowable function, please ask on Piazza before assuming.

There are no function signatures supplied with this assignment. Please follow the directions for the requested output format. Failure to supply the requested output could result in points taken off.

This assignment also contains extra credit tasks. If you choose to implement functions in these tasks. **The extra credit tasks are completely optional.** You can get perfect grade for the programming assignment as long as your baseline solutions are correct. We suggest you finish implementing the basic tasks before doing any extra credit task.

The data for this assignment can be found here:
https://github.com/qchenclaire/computer-vision-fall-2020/tree/master/HW2

2D Transformations:

Short Answer (5 pts each, 25 points total)

In the following problems, write a short piece of code to answer the following questions. Your code should execute and print the answers to each. Note you may not use Python functions that directly compute the answer, but you are allowed to use all of the basic linear algebra functions. You may reuse the code you've written for the Week 5 Python notebook.

1) Load source points from 1a.npy and matching target points from 1b.npy, all of which are correct, compute the corresponding affine transformations or indicate that it is not possible to and why.

   1a.npy and 1b.npy both contain a 10x3 array where each row is a 3D point. The first row of 1a.npy matches the first row of 1b.npy; The second row of 1a.npy matches the second row of 1b.npy and so forth. You will need numpy.load() to load data points. Note that to check if a value x is equal to 1, you may use abs(x-1)<eps instead of x==1.

2) Load source points from 2a.npy and matching target points from 2b.npy, all of which are correct, compute the corresponding homography or indicate that it is not possible to and why.

3) Load source points from 3a.npy and matching target points from 3b.npy, more than 50% of which are correct, compute the corresponding homography or indicate that it is not possible to and why.

4) Load gradients from gradients.npy, where there are 20 data points, and each data point contains (x_gradient, y_gradient), compute a weighted histogram of gradients, where the contribution of each gradient is proportional to the magnitude of the gradient.

5) Compute the gradients of the bikes1.png, graf1.png,leuven1.png and wall1.png and produce the histogram of gradients for each image.

2D Feature Detection And Matching (30 points)

The "data" subfolder contains the following images for you to test with:

- `bikes1.png, bikes2.png, bikes3.png`
- `graf1.png, graf2.png, graf3.png`
- `leuven1.png, leuven2.png, leuven3.png`
- `wall1.png, wall2.png, wall3.png`

Use the Orb feature detection and matching system you used in the notebook to build a mosaic in the following steps:

Part 1 (10 pts) Use the Orb feature detector together with your RANSAC code and homography detection described above to develop a system to compute the homography that relates two images. Test it on the first two images of the four three image series above and report the resulting transforms.

Part 2 (10 pts) Write a function that, given a series of N images, computes the pairwise matches between them, and uses your inverse warping function from the Python notebook to warp all of the images to the common image plane of the middle image. Return the resulting transforms and the resulting mosaicked image using the four full three image series in the data set.

Note: You can also use the built-in warping function in openCV for partial credit if your inverse warping is either not working or too slow.

Part 3 (10 pts) Extend this to object detection

Read the following OpenCV tutorial on object detection with SIFT:
https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html and
https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html

Using Orb, write a program which builds a dictionary indexed by file name of the first image of each of the series of (bike, graf, leuven, wall) images. It should then read a new unseen image, 'testimage.png' and it should find all of the objects that are present in that image. Note this could include duplicates! I've included an example, but you can easily generate your own test data (hint, start simple and build up to something as complicated as what I've given you).

For full credit, your code should use RANSAC and homography estimation to detect consistent features and thus be able to detect multiple instances of an object in the image. It should print out the identity of the object(s) in the image, the coordinates of the bounding box of the detected object, and it should draw the bounding boxes on the original image and display it.

3D Camera Modeling:

Short Answer: (5 pts each, 25 points)

Consider a camera with a 50mm focal length lens and pixel size of .005mm (5 microns). You may neglect any lens distortion parameters. The sensor size is 1000x1000, and the center of the image is at pixel location 500,500 (consider indexing to start at zero). Submit a Python notebook that prints the answers to the following questions:

1) What is the projection in metric units (i.e. without pixel conversion) for the points (0,0, 1000), (100,0,1000), (150,150,2000), and (300,300,4000), all expressed in mm from the camera optical center?

2) What are the pixel coordinate projections of the same points?

3) Suppose now that we insert a transformation that moves the points 100mm along the positive x direction (i.e. to the right). What are the corresponding metric and pixel

coordinate projections?

4) Now, insert a transform that rotates the points by 5 degrees about the z axis. Apply this transformation to the points *before the image translation* described above. What are the resulting metric and pixel coordinate projections?

5) Now, reserve the order of the rotation and the translation from #4 and report the metric and pixel projections.

Programming (25 points):

For these questions, you will find the following tutorial useful:
OpenCV: Camera calibration With OpenCV

Construct a Python notebook that performs the following:

Part 1 (10 pts): Calibrating a stereo camera: In calib_imgs you will find a series of image pairs taken from a stereo camera setup. The folders “1” and “2” contain the series from each camera. Calibrate each camera individually and report its calibration parameters using the openCV camera calibration function findChessboardCorners and calibrate Camera Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation.

Part 2 (10 pts): Report the following: a) the transform that relates points in camera 2 to camera 1; and 2) the transforms that transform points from both cameras to a common rectified camera frame.

Part 3 (5 pts): Use image warping to produce rectified images of each of the calibration pairs. Prove they are rectified by redoing Part 1 on the rectified images and show that they have the same internal parameters and that the transform between them is now a pure translation along the x axis.

Extra Credit (15 points)

2D Imaging: Part 2a (5 pts EXTRA CREDIT) Improve your mosaic by image blending. Implement the feathering blending function described in the class notes to blend your mosaicked images. Show the results for a blend region of 200% of the overlap region of the images. That is, if the image overlap is roughly 100 pixels, make your blend region 100 pixels on either side of the mid-line.

3D Imaging: Part 1a (5 pts EXTRA CREDIT) Acquire images from your laptop or phone camera and run calibration on them. Report the resulting calibration parameters and the device you used to acquire the images. Provide some evidence that the internal parameters you’ve computed are consistent with the sensor in that device.

SUPERBONUS (5 points): Combine these together! Calibrate your camera, then use the calibration to remove image distortion and produce mosaics from your own camera. Provide your code plus “before” and “after” mosaics showing the effect of distortion correction.