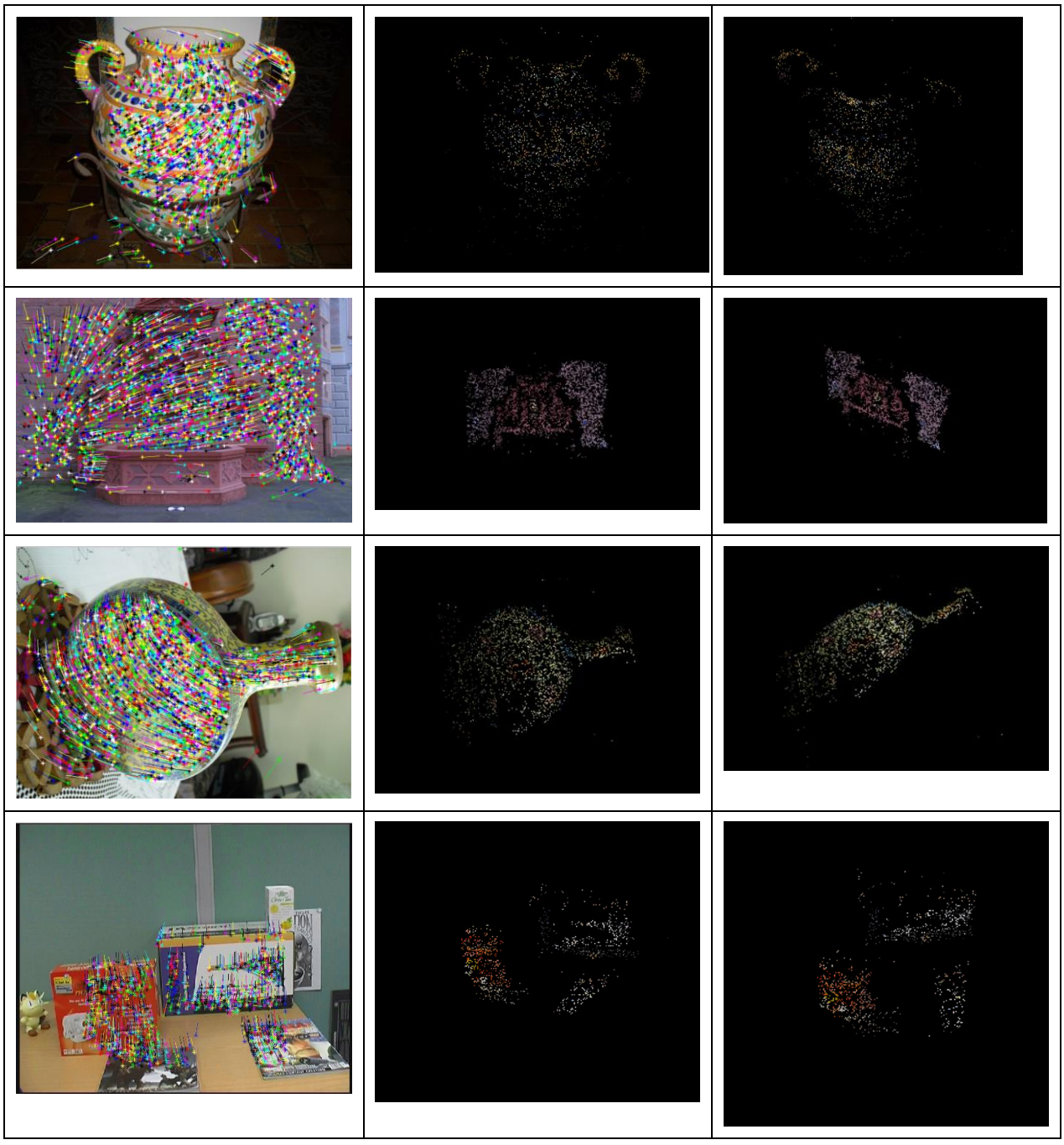
Homework Assignment #5

DUE: 5:00pm, Sunday May 1st (Electronic turnin required)



You are to implement a 2-view, sparse 3D reconstruction algorithm. Your program takes two photos taken by the same camera, with the camera executing a general motion. With a general camera movement, epipolar lines most likely will not correspond to the image scan lines.

Furthermore, you will not be able to recover the absolute scale of the 3D scene. A rough estimate of the camera's intrinsic parameters will be provided with the test data sets. So you can attempt a similarity 3D reconstruction.

Sample test images can be found in http://www.cs.ucsb.edu/~cs281b/testimages/prog5/ (or follow the local image archive link from the class web page). Make sure that your programs work on images in that directory. Again, you are welcome to use your own photos, but do heed some simple instruction for taking pictures, e.g., the baseline should not be too large nor too small, the camera's aim shouldn't rotate too much, environmental lighting must be good, and the object of interest must be heavily textured and be of a good size.

Your Matlab code should be named: SfM2.m (structure from motion using 2 views) and should accept two image filenames, and output the 3D point cloud to a file in this format:

x y z r g b

That is, each line should correspond to a recovered 3D point, with coordinates (x,y,z) and color (r,g,b). (x,y,z) should be in the floating-point format and (r, g, b) should be integers, ranging from 0 to 255. You can use the pixel color from either input image.

You should add the following header to your point-cloud file and rename your point-cloud file with a "ply" extension.

```
ply
format ascii 1.0
element vertex 1336
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
end_header
```

The last number in the 3rd line beginning with "element vertex" should correspond to the number of 3D points in your point-cloud file (in the above example, there are 1,336 3D points recovered). Your point-cloud file is now in the ply format that can be viewed by most 3D visualization software (e.g., Meshlab).

BONUS: You need to implement only a sparse, feature-based 3D reconstruction algorithm. It is also possible to perform dense reconstruction if you rectify the image pairs into a standard stereo configuration or recover the epipolar relations mathematically. Be warned that dense reconstruction based on un-calibrated and un-rectified image pairs is considered a tough problem. Featureless regions are hard to match and, without camera calibration, it may not be possible to recover accurate epipolar geometry. Again, turn in your images with your program if you are attempting something extra that the TA should take a look at.

A word on the camera's intrinsic parameters:

In each directory there are two files: intrinsic.txt.backup and intrinsic.new that record the camera's intrinsic parameters. These files have the same format: The first three lines contain the 3x3 intrinsic matrix **K,** which is:

$$\mathbf{K} = \begin{bmatrix} f \cdot \alpha & 0 & u_o \\ 0 & f \cdot \beta & v_o \\ 0 & 0 & 1 \end{bmatrix}$$

Where (u_o, v_o) are the optical center, *f* is the focal length, and α and β are the length conversion factors (from mm to pixel). The 4th line contains four numbers: CCD width, CCD height, and the first two terms of the lens' spherical distortion parameters (defined in http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html)

The difference is that intrinsic.txt.backup often contains a rough estimate of these parameters, where u_o and v_o are half the CCD width and height, respectively, and the two spherical distortion parameters are zero. On the other hand, intrinsic.new contains calibrated intrinsic parameters, estimated automatically in the 3D reconstruction process. Sometime, they are the same if the camera was calibrated in advance.