Intro to Image Understanding (CSC420)

Assignment 3

Posted Oct. 18, 2017; Submission Deadline: Oct. 27, 2017 at 11:59pm

Instructions for submission: Please write a document and submit a **PDF** with your solutions (include pictures where needed). Include your code inside the document, and submit through **MarkUS**.

For full marks you must show your work, not just your final answer.

Max points: 12, max extra credit points: 3

1. [**2 points**] A robber left his/her shoe behind. The police took a picture of it (see **shoe.jpg**). Estimate the width and length in centimeters of the shoe from the picture as accurately as possible! Show your work.

2. You've joined a CSI unit and our suspect **mugShot.jpg** has been implicated in a grisly crime. Raiding his apartment, we recovered a photograph of him sitting with his accomplice, but he knew we were on to him and shredded the photograph! We need you to implement RANSAC and re-assemble it. You may also use a downloaded implementation of SIFT and any code you wrote for Assignment 2. Python users check `opencv-contrib`.

   (a) [**2 points**] Create a controlled test case between two affine transformed images, and develop your RANSAC algorithm to calculate the affine transformation via least-squares. Recall that the minimum number of correspondences necessary to solve for a 2D affine transform is 3. Visualize the best transformation for the best matching image just like you did for Assignment 2, exercise 2(d). You can use any tutorial code to help create your test-case image, or draw one, or take some pictures.

   (b) [**2 points**] The **shredded** directory contains the shredded picture pieces. Using the mugShot, try reassembling the image in random permutations and keep the one that best fits your model. Rank your models by the mean residual SSD. You could alternatively try a greedy or dynamic programming approach. Show the re-constructed image. Display your final, best reassembled image. *Hint: For speed use down-sampling.*

3. For this question you do not need to write code, but you do need to write the equations and show your work (i.e. how you derived them). If you need to calculate the location of any points or lines in the image plane, state those as givens; but use the minimal

set. You can assume that the ground plane is orthogonal to the image plane. *Hint: Draw the scenes on paper from the side.*

(a) **[2 points]** Examine image **tracks.jpg**. Assume you are given K, R, t. Are you able to estimate the distance between adjacent railway ties in world coordinates? If so, write the necessary equations as a function of the pixel locations, camera intrinsic and/or extrinsic matrices.

(b) **[2 points]** Examine image **man.jpg**. The camera centre is 95 centimetres off the ground. You can assume the ground is planar. Are you able to estimate the height of the man in centimetres without K? If so, derive and write the necessary equations as a function of the pixel locations in the image, and estimate the height.

4. Attached is an image **um_000038.png** recorded with a camera mounted on a car. The focal length of the camera is 721.5, and the principal point is $(609.6, 172.9)$. We know that the camera was attached to the car at a distance of 1.7 meters above ground.

(a) **[0.5 points]** Write down the internal camera parameter matrix K.

(b) **[0.5 points]** Write the equation of the ground plane in the camera's coordinate system. You can assume that the camera's image plane is orthogonal to the ground.

(c) **[1 point]** How would you compute the 3D location of a 2D point (x, y) in the image by assuming that the point lies on the ground? Express your answer in the camera's coordinate system. You can assume that the camera's image plane is orthogonal to the ground. No need to write code, math is fine.

Extra Credit (Easy)

You are given an image with depth captured with Microsoft Kinect. The file **rgbd.mat** contains a variable `im` which is the RGB image and `depth` that contains depth information for each pixel (such an image is typically called an RGB-D image). Depth is nothing else but the Z coordinate in the camera's coordinate system. To get familiar with it, you can plot it with e.g., `imagesc(depth)` (in Matlab). In this plot, pixels that are red are far away, blue ones are close to the camera, the rest are somewhere in between. Further, you can find a function **camera_params.m** which contains the camera's parameters.

(a) **[1 point]** Compute a 3D coordinate for each pixel (with non-zero depth) in camera coordinate system. Plot the computed point cloud (all 3D points). You can use the function `plot3` (in Matlab). For visually more appealing plots you could also use the function `surf`. Include the plot in your solution document.

(b) **[2 points]** The file **rgbd.mat** also contains a variable called `labels`. This variable encodes four objects of interest. For example, `imagesc(labels==1)` will visualize the first object of interest and `imagesc(labels==4)` the fourth one. Thus, all pixels in `labels` that have value 1 belong to the first object, all pixels that have

value 2 belong to the second object, etc. To get the x and y coordinates of all pixels that belong to the first object, you can do: `[y,x] = find(labels==1);`. For each object, compute the 3D location for all of its pixels. Now compute the geometric center of each object by simply averaging its computed 3D coordinates. Write code that finds the object (among the labeled four) that is farthest from the camera (i.e. its distance to camera center is the largest). Write also code that finds the object that is the highest above floor. Here you can assume that the image plane is orthogonal to the floor.

Sources

- **man.jpg**: Chris Ford, `https://www.flickr.com/photos/chrisschoenbohm/4817576839/`
- **tracks.jpg**: Ryan Voetsch, `https://www.flickr.com/photos/voetshy/`