CSE 559A: Fall 2020

Problem Set 3

Due: Nov 10, 2020. 11:59 PM

Instructions

Please read the late submission and collaboration policy on the course website:
http://www.cse.wustl.edu/~ayan/courses/cse559a/

Install Anaconda for Python 3.6+ from https://www.anaconda.com/download. We will test all code on this distribution. You can install it locally in a separate directory without interfering with your system install of Python.

1. Complete the code files in `code/` by filling out the required functions.
2. Run each program to generated output images in `code/outputs` directory.
3. Create a PDF report in `solution.pdf` with LaTeXby editing `solution.tex`. In particular, make sure you fill out your name/wustl key (top of the .tex file) to populate the page headers. Also fill out the final information section describing how long the problem set took you, who you collaborated / discussed the problem set with, as well as any online resources you used.
4. The main body of the report should contain responses to any math questions, and also include results and figures for the programming questions as asked for. These figures will often correspond to images generated by your python code in the `code/outputs/` directory.
5. Once you are done, "git add" the completed `solution.pdf` and your updated code files in `code/*.py`. Please do not add the generated output images, as these are already in your report (the git repo is setup to ignore those files). Then do a "git commit", and a "git push". Then, do a "git pull", and a "git log" to verify the timestamp of your submission and the files included. These instructions are also explained in the "problem-sets" section of the course website.

As a general guideline for all problem sets: Write efficient code. While most of the points are for writing code that is correct, some points are allocated to efficiency. Above all, try to minimize the total number of multiplies / adds. For the same number of underlying operations, try to keep the use of `for` loops to a minimum (i.e., over a minimum number of indices). Instead, use convolution, element-wise operations over large arrays, calls to matrix multiply, etc.

PROBLEM 1 **(Total: 15 points)**

(a) Out of a set of N correspondences, assume there are J outliers. If you choose K samples at random, what is the probability that all of them are inliers (i.e., none of them is an outlier) ? **(5 points)**

(b) We are going to do this multiple times, in the hope that *at least once* we have a set of K samples that is completely outlier free. If we want this to guarantee this with probability P, what is the minimum number of times we should sample ? Express your answer in terms of J, K, N, and P. **(5 points)**

(c) Let's say we're dealing with a pair of images that depict the same scene where there are two planes. Given a set of N correspondences, assume that I_1 of these pairs lie on one of the planes and are consistent with the same homography, I_2 lie on the other plane and are consistent with a different homography, and the remaining $J = N - I_1 - I_2$ correspondences are random outliers.

If I draw K samples, what is the probability that all K samples are inliers and come from *the same* plane ? In other words, what is the probability that this using these K samples will give me a correct estimate of the homography for *either* of the two planes ? Note that if my set of K samples contains outliers, or mixes samples from two different planes, my results will be erroneous. **(5 points)**

PROBLEM 2: ROBUST LINE FITTING **(Total: 20 points)**

We will use the idea of robust fitting to the simpler task of fitting lines to (x, y) co-ordinates. Consider the equation of the line in the form $y = mx + b$. Given a set of chosen points (x_i, y_i), we will fit m, b in the least-squares sense as:

$$m, b = \arg\min_{m,b} \sum_i (y_i - mx_i - b)^2$$

(a) Implement the iterative fitting idea by filling out `fitLine` in `prob2a.py`. The goal here is to first find the best fit to all points and then select an inlier set based on samples for which the error is lower than a threshold ϵ (passed to the function). Then, the method iteratively refines the estimate of the line by fitting only to the inlier set, and then rebuilding the inlier set based on the current estimate.

When run, `prob2a.py` will call this function on different point sets with different numbers of outliers. (It will also call the function for only 1 iteration, implying doing a regular least-squares fit, no the set with a moderate number of outliers). Export the generated figure and include it in your report, and comment on the behavior of this method. **(10 points)**

(b) Implement RANSAC by filling out `ransac` in `prob2b.py`. In this case, do N runs of choosing K samples each, fitting a line to each set of K samples, and then building an inlier set for each run. Choose the largest inlier set among all runs, and return the final estimate of the line as the estimate based on this inlier set. When run, `prob2b.py` will call the function with different choices of K and N on a sample set with a large fraction of outliers. Be sure to call the script multiple times, and comment on which choices return the best fit most consistently. **(10 points)**

PROBLEM 3 **(Total: 15 points)**

(a) What is the relationship between the projected co-ordinates of the same point in two cameras, where the projection matrices differ only in terms of the focal length ? In other words, there is no translation or rotation, but we just move the sensor plane closer/farther from the optical center. You can assume that the intrinsic projection matrix is:

$$K = \begin{bmatrix} f & 0 & W/2 \\ 0 & f & H/2 \\ 0 & 0 & 1 \end{bmatrix},$$

with $f = f_1$ for camera 1 and $f = f_2$ for camera two. Derive the expression relating the projected co-ordinates (x_1, y_1) in camera 1 to (x_2, y_2) in camera 2, in terms of f_1, f_2, W, and H. **(5 points)**

(b) Now consider two cameras that are related by both rotation and translation. We know that with translation, points in the two camera images can no longer be related by a homography in general. But now, assume that all points in the 3D world lie on a plane. Show that if you know what that plane is, then you can relate the mapping between two cameras with a homography. (Hint: begin by choosing a world co-ordinate system where the equation for the plane is given by $z = 0$. Both cameras will then have their own arbitrary extrinsic matrices. Then derive the expression for projected homogeneous co-ordinates in one camera in terms of those in another). **(10 points)**

PROBLEM 4 **(Total: 30 points)**

(a) Implement the function `getH` in `prob4.py` which given a number of pairs of 2D co-ordinates (you will get an $N \times 4$ array where each row are the pair of Cartesian co-ordinates $[x, y, x', y']$), computes the Homography that maps from (x, y) to (x', y') (i.e., the matrix H such that $[x', y', 1]^T \sim H\,[x, y, 1]^T$). Convert the pairs of points to Homogeneous co-ordinates, derive the system of equations for the elements of the Homography as discussed in class (retain all three equations per correspondence), and solve using SVD. (You may want to debug this function separately by calling it on arbitrary pairs of quadrilaterals, and looking at how well the Homography maps the provided set of points to each other). **(15 points)**

(b) Now use this function to splice one image into another by filling out the function `splice` in `prob4.py`. This function will be provided a source image, a destination image, and a 4×2 array corresponding of the (x, y) co-ordinates in the destination image of the, in order, top-left, top-right, bottom-left, and bottom-right points of a quadrilateral (which will correspond to the projected image of a rectangle in the scene).

First, fit a homography that goes from these co-ordinates to the corners of your source image (which will be $(0, 0)$, $(W-1, 0)$, $(0, H-1)$, $(W-1, H-1)$ for the source image). Then, use this homography to warp and place the source image into the quadrilateral in the destination image. Note that you don't need to do anything complicated to figure out which points lie inside the quadrilateral. Simply consider the larger rectangle that includes the quadrilateral (going from min to max values of the x and y co-ordinates of the quadrilateral corners). Apply the estimated Homography to all of them, and figure out which points map to inside the boundaries of the source image (i.e., x co-ordinate is between 0 and W-1, y between 0 and H-1). Then transfer these values using bilinear interpolation.

Running `prob4.py` will create a spliced image in `outputs/prob4.png`. Include this in your report. **(15 points)**

PROBLEM 5 **(Total: 20 points)**

(a) Fill out the function `census` in `prob5.py` to compute the 5×5 centered census transform of an image. The function will take a grayscale image as input and should produce a `uint32` array of the same size as output. The lower 24 bits at location (x, y) should be set based on whether the intensity at (x, y) was greater than that at $(x - \delta_x, y - \delta_y)$ (to 1 if it is, 0 otherwise), with each bit corresponding to a different value of δ_x, δ_y going from -2 to 2. Note that you don't need to use a bit for comparing (x, y) to itself. The order of the bits isn't important as long as you use the same convention everywhere. Near the boundaries, set a bit to 0 if the corresponding $(x - \delta_x, y - \delta_y)$ falls outside the image boundary. **(10 points)**

(b) Now given a pair of rectified left and right stereo images, match points based on the hamming distance of their census codes. Given a point (x, y) in the left image, you can assume that it'll match with one of $(x - d, y)$ in the right image where the "disparity" d is an integer with $0 \leq d \leq D_{\max}$. Also for now, assume that the matching point $(x - d, y)$ can not lie outside the right image, so $d \leq x$.

Fill out the function `smatch` in `prob5.py` to take a pair of images and the value of $D_{\max}$, and find the disparity map as the best value of d at each pixel which minimizes the Hamming distance between the census codes at (x, y) in the left image and $(x - y, d)$ in the right. Use the function you wrote in part (a), and we provide a function `hamdist` for computing the element-wise hamming distance between two `uint32` arrays.

Running `prob5.py` will generate a color representation of the estimated disparity map in `outputs/prob5.png`. Include it in your report. **(10 points)**