CSE 559A: Fall 2020

Problem Set 4

Due: Nov 24, 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: 30 points)**

Consider a pair of cameras with projection matrices $K[I|\bar{0}]$ and $K[I|t]$, where I is the 3×3 identity matrix, $\bar{0}$ is a 3-dimensional all zeros vector, t is a three dimensional vector representing translation, and both share the intrinsic matrix,

$$K = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

i.e., they have a focal length of f in pixels, and we are assuming that image co-ordinates are zero-centered.

(a) Consider the horizontal stereo case, where $t = [-t_x, 0, 0]^T$ (where $t_x > 0$), that is the second camera is shifted "right" by t_x (in world length units, say meters). Show that if a point in the world projects to (x, y) in camera 1 and (x', y') in camera 2, then $y' = y$, $x' \leq x$, and that the disparity d is given by

$$d = x - x' = \frac{f t_x}{Z},$$

where Z is the depth (i.e., distance in the view direction from the camera center, or z co-ordinate) of the world point (in meters). **(10 points)**.

(b) Assuming the same pair of camera projection matrices as in part (a), and consider a set of 3D world points that lie on a plane such that the world co-ordinates (X, Y, Z) for all points satisfy the equation $X\alpha + Y\beta + Z\gamma = k$ for some plane parameters $(\alpha, \beta, \gamma, k)$. Show that for all such points, the disparity d is a linear function of the projected (left) image co-ordinates (x, y), as $d = ax + by + c$. Derive an expression for (a, b, c) in terms of the world plane parameters $(\alpha, \beta, \gamma, k)$ and the camera parameters f, t_x. **(10 points)**.

(c) Now, consider the problem when the second camera has a translation vector $t = [0, 0, -t_z]^T$ (where $t_x > 0$). For this case, derive expressions for the projected location (x', y') in camera 2 in terms of the location (x, y) in camera 1, and the depth of the point Z (from the center of camera 1). Assuming that $Z \geq t_z$ for it to be visible in both cameras, what are the set of possible co-ordinates (x', y') that could match to (x, y) (this is when you don't know the depth Z) ? **(10 points)**.

PROBLEM 2 **(Total: 15 points)**

(a) Implement the function `buildcv` in `code/prob2.py` to build a cost-volume. The function will be provided two gray scale images and the maximum possible disparity value input, and should return a three dimensional array of size $H \times W \times (D_{\max} + 1)$, where the the $[y, x, d]$ element is the cost of matching pixel x, y in the left image to $x - d, y$ in the right image (for $0 \leq d \leq D_{\max}$). Use the hamming distance of 5×5 census transform as in the last problem set. For values where $x - d < 0$, set the cost to 24 (corresponding to the maximum possible hamming distance).

The support code will call this function with grayscale versions of the left and right image, and then simply compute the disparity map by doing an $\arg\min$ to produce a color-mapped disparity image `outputs/prob2a.jpg`. Include this in your report. **(5 points)**.

(b) Next, implement `bfilt` to smooth this cost volume using Bilateral filtering with the left RGB image as the guide. Formally, given a cost volume $C[n, d]$, you want to return a smoothed volume $\bar{C}[n, d]$, where

$$\bar{C}[n_1, d] = \sum_{n_2} B[n_1, n_2] C[n_2, d].$$

Here, $B[n_1, n_2]$ is spatially-varying, and defined with respect to the RGB left image X:

$$B[n_1, n_2] \propto \exp\left(-\frac{|n_1 - n_2|^2}{2\sigma^2} - \frac{|X[n_1] - X[n_2]|^2}{2\sigma_I^2}\right), \quad \sum_{n_2} B[n_1, n_2] = 1.$$

This is therefore very similar to the standard Bilateral filter (from Problem Set 1), except that you are computing the kernel based on the RGB image X, and applying to the cost volume C.

The support code will call your function (with appropriate parameters for window size, and spatial and color variances for the kernel), and again do an $\arg\min$ to produce a second disparity map `outputs/prob2b.jpg`. Include this in your report. **(10 points)**.

PROBLEM 3 (Total: 40 points)

(a) Implement the function `viterbilr` in `code/prob3a.py` to implement the forward-backward algorithm described in class to find an exact solution for disparity optimization with a smoothing cost applied only along horizontal neighbors. You will also need to copy the `buildcv` function from problem 2. Given a cost volume C, and values of P_1 and P_2, your function should return a disparity map $d[n]$ that exactly minimizes the cost function:

$$d = \arg\min_d C[n, d[n]] + \sum_{(n,n')} S(d[n], d[n']),$$

where the second summation is over pairs of pixels that are horizontal neighbors, and the smoothness cost is defined as:

$$S(d, d') = \begin{cases} 0 \text{ if } d = d' \\ P_1 \text{ if } |d - d'| = 1 \\ P_2 \text{ otherwise.} \end{cases}$$

Implement this to be efficient as described in class. You will need to do a for loop over the x co-ordinate as you go left-to-right and back, but you should operate on all lines in parallel, and do efficient minimization over disparities.

Running `code/prob3a.py` will call your function and store the returned disparity map in `outputs/prob3a.jpg`. Include this in your report. **(20 points).**

(b) Implement the function `SGM` in `code/prob3b.py` (you will again need to copy the `buildcv` function to this file) to perform semi-global matching as described in class. This function should form "augmented" cost volumes by traversing the image in four directions: left-to-right, right-to-left, top-to-bottom, and bottom-to-top, take their sum, and compute the disparity map as the $\arg\min$ of this summed volume. The smoothness cost function to be used is the same as in part (a).

Running `code/prob3b.py` will generate `outputs/prob3a.jpg`. Include this in your report. **(20 points).**

PROBLEM 4 (Total: 15 points)

Implement the function `lucaskanade` in `code/prob4.py` to perform Lucas Kanade optical flow estimation. Your function will be provided two grayscale image frames, and a window size to aggregate equations / gradient moments from.

Use convolutions with the filters fx and fy defined in the code to compute x and y image derivatives. (Also, apply these derivatives to the 'average' of the two frames).

Running `code/prob3b.py` will call your function on the two frames in the `inputs/` directory, and display a "quiver" plot. Save this plot as an image, and include it in your report.