CSC420— Introduction to Image Understanding, Fall 2011

Assignment 1: Calibration Checkerboards

Due: 3:00pm, Thurs., Oct. 13
This assignment is worth 20 marks towards your grade in this course.

What to hand in. Both a short written report and the completed Matlab handout code will be submitted electronically. For information on electronic submission see `http://www.cdf.toronto.edu/students/submit.html`. Note you must have physically logged on to a CDF lab machine (and changed your password) before you will be allowed to submit electronically.

Your report must be a pdf file named A1Report.pdf. In addition, make a zip file of your Matlab solution code. The zip file must be called A1Matlab.zip, and should consist of the directory A1Matlab, with two subdirectories Q1, Q2 for the solution of the two questions below. These directories are already in the handout code. (Do not hand in the subdirectory `util` from the handout code.)

In your report you can assume that the marker knows the context of the questions, so do not spend time repeating material in this handout or in the lecture notes.

1. **Camera Parameters [5pts].** Starter code for displaying a 4×4 checkerboard on a ground plane from various points of view is provided in the handout code `A1Matlab/Q1/checkerShow.m`. The ground plane is the $Z_w = 0$ plane in world coordinates. There is a checkerboard lying in this ground plane, centered at the origin of the world coordinate frame, $\vec{X}_w = \vec{0}$. In your solution code, the camera should always be placed above the plane looking down at it. When both parts of this question are completed, the first image frame that is generated should look like the one shown below.

Here the lighter gray denotes sky, while the darker gray denotes the ground plane away from the checkerboard.

(a) Given the intrinsic camera matrix, M_{int}, and the rotation R which forms the first three columns of the extrinsic camera matrix M_{ext}, we wish to solve for the camera's center of projection $\vec{d}_w$ in the world coordinate frame, subject to the following two constraints:

- We want the center of projection $\vec{d}_w$ to be chosen such that the origin of the world coordinates is imaged to the fixed pixel $\vec{p}_0$; and
- We want the distance from the camera's center of projection to the origin of the world coordinates to be $||\vec{d}_w|| = D$.

All the quantities necessary to solve for $\vec{d}_w$ are specified in the starter code `Q1/checkerShow.m`. The variables `boardCenterPx` and `distToCenter` in the starter code refer to the fixed pixel $\vec{p}_0$ and D, respectively. These variables should not be changed.

Given the center of projection which satisfies these two constraints, namely $\vec{d}_w$, you can then fill in the missing fourth column of the extrinsic calibration matrix, M_{ext}.

In your write up, derive equations for $\vec{d}_w$ which are simple to solve in Matlab. Replace the appropriate part of the starter code `checkerShow.m` to implement the solution of $\vec{d}_w$. (The subsequent "sanity check" in the starter code should then be satisfied, although the images generated will still be uniformly sky-gray.)

(b) The starter code generates a $3 \times N$ array `pixCoord`, where each column represents a distinct pixel $\vec{p} = (p_1, p_2, 1)^T$. Here $\vec{p}$ is in pixel coords, with p_1, p_2 integer-valued, so $p_1 = p_2 = 1$ denotes the top left corner of the image, and so on.

In your write up, explain in detail how to compute the scene point that is mapped to each pixel $\vec{p}$. That is, for any given $\vec{p}$, explain how to identify that the sky is imaged to $\vec{p}$. Also, for the case $\vec{p}$ is a non-sky pixel, explain how to identify the X, Y coordinates on the ground plane that projects to $\vec{p}$.

(c) In the Matlab code `checkerShow.m`, implement the above computation of the classification of sky vs non-sky points, and for the non-sky points compute the X, Y coordinates on the ground plane. This coordinate transform should be done as a matrix operation, without you needing to program a Matlab loop. These coordinates X and Y will then be used by the subsequent code (provided in the handout code) to determine the brightness of the ground plane for that pixel.

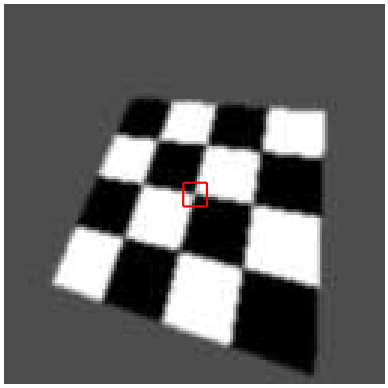
2. **Calibration Patterns [15pts].** It is common to use planar checkerboard patterns for camera calibration. Examples of images of such calibration objects are given in the subdirectory `Q2/data` of the handout code. (Synthetic examples are generated by the solution of Question 1.)

Camera calibration depends of accurately locating the image locations of the corner points of the calibration pattern. Here, a corner point is a point in the corner of **four neighbouring checks**, that is, points on the outer boundary of the checkerboard are not counted as corner points. For example, in the previous figure there are exactly nine corner points.

Here we begin the task of identifying this type of calibration pattern in images. This whole task is too big for a first assignment, so we focus instead on a possible first step. In particular, we investigate the use of HoG features to quickly identify potential corner points. This quick identification stage will suffer from several false positive (FP) and false negative (FN) identifications (that is, respectively, (FP) points in the image that are identified as checkerboard corner points but are not, and (FN) points in the image that are indeed the image location of a checkerboard corner point but are not identified as such).

Here we will try to minimize the occurrences of incorrect corner detections. However, we won't be able to avoid them all. The idea is that subsequent processes will use our results as input data, along with the image, and obtain a final match for the checkerboard target. In order to be useful for these subsequent processes, it is important that *we minimize the number of false negatives* (i.e., corner points in the image that are not detected). Having extra false positives after this first stage turns out to be necessary.

(a) **HoG at Corner Points.** Assume that a histogram of gradients is computed within a single cell, say consisting of an 8×8 square image patch, and this histogram forms our HoG response. Suppose the cell is centered on the image of a checkerboard corner point, and that only the four neighbouring checks appear in this 8×8 patch. An example is shown below, where the red square indicates the single HoG cell.



Describe the expected structure of this histogram. That is, how many peaks do you expect, where are the peaks in relationship to other peaks, and so on? To help you with this task, if you set the flag `PT_SAMPLE` to `true`, the handout code `calibHoG.m` allows you to select single image points with your mouse and then displays the histogram for the HoG cell centered on this point.

(b) **Single-Cell HoG Detection.** In the handout code, implement `Q2/findCornerPts.m` so that it checks the properties that you identified in part (a) above and produces a non-negative score for each point. The score should increase as you become more confident that the point corresponds to a corner point.

The handout code is set up so that (with `PT_SAMPLE` set to `false`) you can run your detection method on a subsampled grid over the whole image with one call to `findCornerPts`. If you run out of memory, simply process the sample points in the variable `z` in several smaller batches.

Adjust your detection method so that it identifies almost all the corner points in the checkerboard images, and has the minimum number of false positives that you can manage. (See the JPEG files in the `Q2` directory for sample results that I obtained using the first three images in the data set.)

For pixels at which you have some confidence that there is a checkerboard corner, your implementation of `findCornerPts.m` should also return the two main image orientations (modulo π) for this potential corner point. See the handout code `findCornerPts.m` for further specifications.

(c) Your write up must clearly describe the detection algorithm you used in part (b). That is, given a HoG histogram, describe precisely what do you do and why. Also, show some printed samples of the results you obtained. Moreover, discuss the reasons for incorrect responses. Why are a few corners missed (if they are)? What sorts of image structures cause non-corners to be identified as corners? Briefly explain why these false positives arise.

(d) **Second Stage.** Describe what you could implement for a subsequent stage of processing that you expect will improve these detection results. This second stage must make significant use of your current output, both the detected points and their orientations. Moreover, be clear about what aspect(s) of your current results you are aiming to improve. Finally, if you are curious and have time, try programming this second stage and adding it to `calibHoG.m`.