COMPSCI 527 — Homework 7

Due on December 9, 2016, 11:59pm

This assignment is a set of instructions for your final project for COMPSCI 527. In this context, a project differs from a regular homework assignment in that it is more open-ended in its goals, and what constitutes an acceptable "solution" is different.

IMPORTANT INSTRUCTIONS: Work in pairs on this project if you can, and alone only if you must. Working in pairs will help avoid impasses and "local optima" in thinking. Each pair should submit *one* PDF file, with both names on the first page. Do not turn in any code or code listing.

The Software

MATLAB software is provided with this project to create a simulated world with an object, two cameras, and the coordinates of salient object points in images of the object taken by the two cameras. The software then adds Gaussian noise to the image-point coordinates and runs the eight-point algorithm on the noisy input. Finally, it computes several measures of output quality, as described below. You are then asked to design and run some experiments and report about the results.

The code is provided "as is," in the sense that if it does not work for your purposes it is your responsibility to modify it as needed, or to rewrite it if you prefer. You ought to be able to write the code yourself, and what you are given is supposed to save you time.

Unzip the code into some directory, make that directory the current directory for MATLAB, and type `RUNME` at the prompt. The code will then call the following main functions, among others. As you read on, you may want to look at the function definitions and at `README.m` to see how they are called.

`world`**:** Creates a synthetic world with three faces of a cubic box with some squares painted on each face. Only three faces are created, as there is no need to make the faces we cannot see. The box is placed near the origin of the world reference system and its edges are aligned with the coordinate axes. This function also makes two cameras, as well as the coordinates of the visible cube vertices and square corners in the images taken by those cameras. Box coordinates are world coordinates, and each camera has its own reference system, so there are three reference systems in all.

The input `side` to `world` is the side length of the box. The input `t` is a 3×2 matrix whose columns are the vectors from the origin of the world reference system to the centers of projection of the two cameras. Input side length and translations are in an arbitrary unit of measure. However, the function `world` scales camera positions and box, so that the distance between the two camera centers becomes one after scaling. This will make performance evaluation easier. If you change `t` (within reason), then the function `world` will re-orient the cameras so they point towards the box, and adjust the focal distances so that the image of the box fills most of the image. The input `resolution` is a 2-dimensional array with the number of image pixels in the horizontal and vertical direction, in this order. The input `sensorSize` is a 2-dimensional array with the size of the sensor (in millimeters) in the horizontal and vertical direction, in this order.

The output `img` from `world` (third output) contains enough information to draw the two images of the box. So `img` does not just contain point coordinates but also connectivity and color information needed to draw the various polygons that make up the image. Specifically, `img` is a 2×2 array of structures. Two structures describe the corners of the squares painted on the faces of the box (one structure for each image), and the other two describe the visible faces of the box (one structure per image). If you use the display and drawing functions provided, you need not concern yourself with the fields `faces` and `colors`, but just with `img(1, 1).P` (the image coordinates of the square corners in the first image), `img(2, 1).P` (the image coordinates of the visible box vertices in the first image), and the two analogous entries `img(1, 2).P` and `img(2, 2).P` for the second image. Each column in each of these arrays contains (x, y) coordinates in this order: x is the column index and y is the row index, and coordinates are at sub-pixel resolution.

The output `b` from `world` describes the box in world coordinates. This description is formally the same as that of, say, `img(:, 1)`, except that the point coordinates are in three dimensions instead of two (in the world reference frame).

The output `c` is an array of two structures that describe the two cameras, with the following fields:

`resolution` is a copy of the input argument `resolution`.

`T` is the rigid transformation from the world reference system to the camera reference system, in the format described in the class notes. It is a structure with fields `R` and `t`. The world reference system is not attached to either camera, so `T` differs from the identity transformation for both cameras.

`Ks`**,** `Kf` are the internal camera calibration matrices K_s and K_f described in the class notes on the camera model (page 4).

`pi0` is the principal point $\boldsymbol{\pi}_0$.

Feel free to peruse the file `world.m` to understand these structures in greater detail, but lines 39 to 46 of `RUNME` show you how to extract the homogeneous coordinates `PTrue` of all the world points from `b` and the canonical coordinates `p1` and `p2` of all the image points from `img`, and this is really all you need to know about these structures. Of course, **corresponding columns of** `PTrue`, `p1`, `p2` **contain corresponding points.**

`addNoise`**:** Adds Gaussian pseudo-random noise to the images. The noise has standard deviation `sigma` pixels. Currently, `sigma` is set to 0.5 pixels, and you will experiment with different values. The instruction

```
rng('default');
```

on line 5 of `RUNME.m` resets the random number generator, so if you run the whole code twice you obtain *exactly* the same results. This makes debugging easier by making bugs repeatable. However, when you run the same experiment multiple times to create your plots, you of course want different random samples to be used for every run.

`longuetHiggins`**:** Computes the rigid transformation from camera 1 to camera 2 and the scene structure in the reference system of camera 1 from the two images using the eight-point algorithm described in the class notes.[1] Since structure can only be recovered up to a global scale factor, the solution returned by this function is normalized so that the distance between the two centers of projection is equal to one.[2]

`motionError`**:** Returns a measure of dissimilarity between true and computed motion. Please look at the body of the function if you are interested in how the error is computed. Both rotation and translation errors are reported in degrees (translation is a unit vector, so translation errors are angles, and their magnitude is reported in degrees as well).

`structureError`**:** Returns a measure of dissimilarity between true and computed structure (the shape of the box). To this end, the function centers the true and computed cloud of 3D points around their centroids, scales them so that their RMS scale is 1, rotates one of the two clouds so as to match the other as well as possible by solving the Procrustes problem described in the class notes[3], and then measures the RMS residual between the transformed clouds. The result is divided by the square root of the number of points, so that the unit of measure for the resulting error is in units of length per point, where the overall (RMS) size of the object is one unit. For instance, a structure error of 0.015 means that the average error is 1.5 percent of the overall size of the object. The value computed by this function is a forgiving definition of structure error, because significant efforts are made to align true and computed shape before measuring their discrepancy.

`reprojectionError`**:** Measures the RMS error between the original *image* points and the points projected from the solution of 3D reconstruction. The units are pixels per point.

`bundleAdjust`**:** Refines the solution obtained with the Longuet-Higgins algorithm. The latter solves the overall optimization problem in stages, and therefore yields a suboptimal solution. The function `bundleAdjust` performs a local optimization by minimizing the mean reprojection error and using the output of Longuet-Higgins as a starting point. This function uses the builtin MATLAB function `lsqnonlin` ("minimize a least-squares nonlinear function"). It returns both camera transformation and scene structure, plus image residuals before and after bundle adjustment.

Other functions provided with this assignment display results in various ways, as you will see when you run the code. These figures are only drawn as sanity checks, to see what happens as you run your code. Please try to understand what these figures mean, and read the code when needed. Keep in mind that structure is displayed in different reference systems for different figures, so you should drag the mouse around in the structure figures to rotate the results and view them from useful viewpoints. You will likely turn off these displays when you run your experiments.

(Continues on the next page)

[1] If you supply a third output argument, then this function also returns structure in the system of reference of camera 2.
[2] As stated earlier, this is also how `world` normalizes everything.
[3] Appendix A of the notes on the eight-point algorithm.

Your Task

Your are to modify the code provided (or write your own, if you prefer) to analyze empirically what happens as you vary the noise level and other parameters. There are several questions one could address, and I leave it to you to choose which questions to ask. Here are some examples. *These are examples only: It is plausible to earn a top score without answering any of these questions, and answering your own instead.* Words in square brackets list alternatives for you to pick from:

- How do [rotation, translation, structure, reprojection] errors depend on image noise?
- How does the [angle between the camera optical axes, distance from the camera to the scene] affect the errors above?
- Noise is in pixels. What happens if you change image resolution but keep `sigma` fixed?
- Does image quantization (coordinates truncated to pixel accuracy) affect the results, how, and depending on what parameters?
- To what extent does bundle adjustment improve the results from the Longuet-Higgins algorithm?
- If you feel adventurous, modify the code in `box` or `world` to squish the box into an increasingly flatter object. Does that affect results? How and to what extent?
- If you feel even more adventurous, ask new questions of your own instead of the ones given above. For instance, you could modify the code to include lens distortion and see how that affects reconstruction. Alternatively, you could write calibration code (the function `initialize.m` on the syllabus page, December 1 may be a starting point) and use the functions provided for this assignment (plus a distortion model) to analyze that instead of the reconstruction code.

Keep in mind that 3D reconstruction is an ill-posed computation, in which small input errors can lead to very large output errors. In particular, large values (say, more than 3 pixels standard deviation) of noise may lead to meaningless results. So you will have to find reasonable noise ranges and values by trial and error.

In addition, each point on each of your plots should be the average (or perhaps, even better, the median) of the results obtained by running the experiment multiple times (say 30 times or more) with the same parameter values (same `sigma` as well), but different instances of pseudo-noise. Without this, your plots will look very random. You may want to add error bars to your plots to show the significance of the results. All these considerations mean that **you may need some time to design and run your experiments, so do not start this project late.** Write a single script file that generates all the plots and figures, so you can make modifications and run everything again without confusion.

Some questions that appear intriguing at first may give uninteresting answers in practice, so give yourself some time to explore alternatives. Asking good questions is as important as answering them well. If something surprises you, describe it and try to explain: An open mind, curiosity, and the technical ability to follow up are what makes a good scientist.

What to Hand In

The main product of your work should be a clear, well-organized document that answers questions similar to the ones above—or different ones if you like—mainly in the form of few readable plots and clear, succinct text. I am not looking for answers to many questions. Just pick two or three questions but answer those carefully, with well-designed plots and good descriptions and explanations in your text, written in good English. You will be evaluated on how well you reason about the problem, rather than how many pages you write. A document that asks and answers new, original questions that make sense, instead of the ones in the examples, would be wonderful. Even a "failed" experiment is rewarded with a high grade if the reasons for the failure are well explained.

It is important to design plots that support your reasoning well, and convey information clearly and succinctly. Once you have enough plots to say what you need to say, adding more plots will likely make your document worse. Thus, you want as few plots as possible, but no fewer, given the argument you want to make. Try to superimpose plots on the same diagram when this makes sense, and as long as the result is readable. Make sure you label all axes and plot lines, and add meaningful, detailed legends and figure captions. Always specify units for all axes, either with the axis labels on in the figure captions. *No unit, no credit.*

Your document should have a brief introduction that explains what you set out to discuss. You need not explain how the eight-point algorithm works, except to the extent that your explanation is useful for your arguments. Then there should be a main section with your results and discussion, followed by a brief but clear and non-perfunctory conclusion that states what you have learned from your work, good or bad. When referring to a figure in your text, do so by also including the figure number: "*Figure 3 shows that...*"

A good document with 4-5 pages of high-quality text is close to optimal. Good thinking about the problem and a good design of experiments are the main criteria, but good structure, syntax, and grammar matter as well, and will be part of the evaluation metrics.

Do not hand in any code. All I will read is a single PDF file that you should submit by midnight of the deadline. It is highly advisable to make the PDF with LaTeX, unless you can create a document of comparable typographical consistency by other means.