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 in what constitutes an acceptable “solution.” Written instructions can only go so far, so I encourage you to ask questions in class for clarification.

The Software

Some software is provided with this project to get you started. 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.

Unzip the code into some directory, make that directory the current directory for MATLAB, and type `RUNME` at the prompt. The code will then do the following (item headers below are the names of the functions that perform each step):

**world:** Create a synthetic world with a calibration box (the same you used for homework 6) and two cameras, and also return the images taken by those cameras. The position of the cameras relative to the box is determined by the input vectors in `tau`. The two cameras always point towards the box, and their focal distances are set automatically so that the image of the box fills most of the image.

The output `img` contains enough information to draw the two images of the box, so it does not just contain point coordinates but also connectivity information to draw the various patches that make up the image, and their colors. Specifically, `img` is a $2 \times 2$ array of structures. Two of these 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 homogeneous image coordinates of the square corners in the first image), `img(2, 1).P` (the homogeneous 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.

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

The output `c` is an array of two structures that describe the two cameras. The names of the fields should be self-explanatory, given the notation used in the textbook.

Feel free to peruse the file `world.m` to understand these structures in greater detail.

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

**motion:** Compute the rigid transformation between the two cameras from the two images. Note that points in the image are assumed to correspond to each other in the order in which they are listed in the image data structures, so correspondence is known for these experiments. To do its work, `motion` uses the so-called eight-point algorithm in table 5.1 (page 121) of the textbook. To this end, `motion` calls the function `essentialDiscrete` from [http://cs.gmu.edu/~kosecka/bookcode.html](http://cs.gmu.edu/~kosecka/bookcode.html). All code from the subdirectory `Ma Soatto` comes from this site. Feel free to use more code from that site or from anywhere else.

**triangulate:** Given motion and image points, this function recovers the scene structure, that is, the 3D coordinates of the points in the frame of reference of the first camera. Recall that structure (and the amount of translation between cameras) can only be recovered up to a global scale factor.

**bundleAdjust:** Using the reconstruction from the eight-point algorithm, this function refines the solution by optimizing the full reprojection error in expression (5.25) of the textbook with the numerical optimization routine `lsqnonlin` available in MATLAB. In so doing, `bundleAdjust` enforces the constraints that rotation is an orthogonal matrix and translation is a unit vector. This refinement, called `bundle adjustment`, sometimes results in better estimates of both motion and structure.

**motionError:** This function compares true and computed motion, and is run both before and after bundle adjustment. Please look at the body of the function to see how the error is computed. Both rotation and translation errors are reported in degrees (translation is a unit vector, so errors are degrees for that as well).

**structureError:** This function centers the true and computed cloud of 3D points around their centroids, scales them so that their RMS scale is 1, rotates one so as to match the other as well as possible, 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. So a structure error of 0.015 is an average error of 1.5 percent of the overall size.
**reprojectionError**: This function 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.

Other functions 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 frames for different displays, so you should drag the mouse around in the structure figures to rotate the results and view them from useful viewpoints.

**Your Task**

Your are to modify this code to analyze empirically what happens as you vary the noise level and other parameters. There are several questions to address, and I leave to you the choice of which questions to ask. Here are some examples. Words in square brackets list alternatives for you to pick from:

- For what noise levels do we stop getting meaningful results, and which types of error fail first?
- How do [rotation, translation, structure, reprojection] errors depend on image noise, either before or after bundle adjustment?
- How does the [angle between the camera, distance from the camera to the scene] affect the errors above?
- To what extent does bundle adjustment help, and what type of error does it help reduce the most? In other words, compare errors before and after bundle adjustment for the various types of error.
- What happens if you change image resolution but keep sigma fixed?
- 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?

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.

When you answer questions like the ones above, 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 from 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. This means also that you may need some time to 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.

**What to Hand In**

The main product of your work should be a clear, well-organized document that answers some of the questions above, mainly in the form of readable plots and clear, succinct text. You will be evaluated on how well you reason about the problem, rather than how many pages you write. A “failed” experiment is still rewarded with a good grade if the reasons for the failure are well explained.

It is important to design plots that answer some of the questions above—or others like them—with the purpose to convey information clearly and succinctly. Once you have enough plots to say what you need to say, the more plots you have and the worse you will likely do. Try to superimpose plots on the same diagram, as long as the result is readable. Make sure you label all axes and plot lines, and add meaningful figure captions.

Your document should have a brief introduction that explains what you set out to discuss. 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.

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

Do not hand in any code. All I will read is a single PDF file that you should email to me (tomasi@cs.duke.edu) by midnight of the deadline.