Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Project #2: A Camera Calibration Toolbox

Fall 2016

Overview

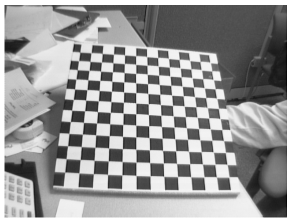
Camera calibration is necessary for almost all robotic vision applications (much to the dismay of roboticists in general). In this project assignment, you will write your own calibration toolbox for standard perspective-lens cameras (i.e., not for fisheye lenses or other lens types). Your toolbox will enable you to calibrate the camera lens intrinsic and radial distortion parameters, using a checkerboard calibration target. The goals are to:

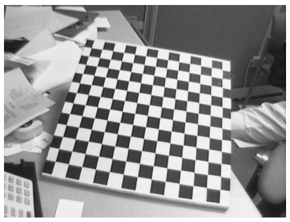
- provide practical experience with the DLT algorithm and process of subpixel feature extraction, and
- assist in understanding nonlinear least squares optimization methods.

The due date for project submission is **Sunday, November 6, 2016, by 11:59 p.m. EDT**. Submission instructions will be posted on Portal prior to this date. To complete the project, you will need to review some material that goes beyond that discussed in the lectures—more details are provided below.

Part 1: Calibrating a Camera for Robotics Applications

(a)


(b)


(c)

(d)

There are several existing calibration toolboxes available for MATLAB. To begin, you should download the most popular toolbox, written by Jean-Yves Bouguet at CalTech, from https://www.vision.caltech.edu/bouguetj/calib_doc/. Give the toolbox a try—the general procedure involves manually drawing a bounding box around each checkerboard pattern (the cross-junctions between the squares are the feature points used for calibration), and then running the feature detection and optimization algorithms. Your task is to implement the same type of interface (although not necessarily as fancy). This portion of the assignment is worth **75 points** (out of 100 points total).

Your software should allow the user to:

- load a set of checkerboard images from a given directory,
- specify the size of each square (useful to obtain absolute scale),
- manually draw an approximate bounding box around each checkerboard pattern, and
- run the calibration algorithm.

The calibration package will need to extract the cross-junction points, estimate the pose of each camera relative to a reference frame defined by the checkerboard, and then perform a nonlinear optimization step to obtain the camera intrinsic and radial distortion parameters (up to the κ_3 radial distortion term; pixel skew may be ignored).

There are variety of possible ways to identify the cross-junctions, which we will discuss in the lecture. Once the cross-junctions are identified, an initial guess for each camera pose (per image) can be found using the Direct Linear Transform (DLT) algorithm (for pose estimation) described in the textbook. Typically, the coarse estimates of the cross-junction positions in each image are then refined using a saddle point detector, such as the one described in the following paper:

> L. Lucchese and S. K. Mitra, "Using Saddle Points for Subpixel Feature Detection in Camera Calibration Targets," in *Proceedings of the Asia-Pacific Conference on Circuits and Systems (APCCAS'02)*, vol. 2, (Singapore), pp. 191–195, December 2002.

Note that *the positioning and ordering of the cross-junction points* relative to the origin of the calibration target are important—the same order is required for each calibration image, for the nonlinear optimization to work correctly.

The final step is to set up, and then solve, the nonlinear system of calibration equations. Your parameter vector should include the pose of each camera relative to the target (using whichever rotation representation you choose) as well as the six or seven calibration parameters (two focal lengths, principal point coordinates, and radial distortion coefficients κ_1, κ_2, and possibly κ_3). You will need to compute the required Jacobians for the nonlinear optimization—these can be determined in steps, using the sequence described on Slide #19 of Lecture #4.

For this portion of the assignment, you should submit:

1. a function, `pose_guess.m`, that computes an initial guess for the camera pose using the DLT algorithm,
2. a function, `checker_points.m`*, that determines the positions of the individual cross-junctions on each checkerboard with subpixel accuracy and orders the points in a standard sequence,
3. a function, `find_jacobians.m`, that computes the Jacobians for all of the elements of the forward projection function (again, see Slide #19),
4. a function, `nonlinear_opt.m`, that accepts a stacked matrix of cross-junction points and an initial guess for the parameter vector, and performs the nonlinear optimization step to produce an updated, optimal estimate of the camera poses and calibration parameters, and
5. a main program, `camera_calib.m`, which loads calibration images from a directory and allows the user to specify both the checkerboard square size and to draw bounding boxes.

The main program should output all of the camera poses (suitable for plotting), as well as the calibration parameters (intrinsic and radial distortion terms). You should also answer the following question: *does including the κ_3 improve your results (in terms of squared residual error), or make the results worse?*

For testing, you may use the set of images included in the project .zip file (examples above), which are from the MATLAB Toolbox page. This will allow you to directly compare your output with the existing Toolbox output. Once submitted, we will test your program on a *hold out* set of images, with different parameters. We'll rank your solutions based on performance (as a kind of mini-competition) and post the results in class. Please include an anonymous 'secret identifier' you would like us to use to identify your submission when posting results.

A final note: the function above (marked with a '*') that detects and orders cross-junction points on the checkerboard can be difficult to write (the ordering process may be harder than you think). This function (only) can be written collaboratively by the class, if desired—that is, students can and should work together through the Portal Discussion Forum (and informal discussions offline) to come up with a workable solution.

Please clearly comment your code and ensure that the filenames above are used, as this will help to speed up marking. *We will run your code.*

Part 2: Undistorting Images

Your second task, now that you have an operational calibration tool, is to write a short program that undistorts or *unwarps* images, based on a set of supplied calibration parameters. The program should accept a single image (greyscale) and the calibration parameters from Part 1, and generate an undistorted image in which straight lines (in the world) appear straight (in the image). This portion of the assignment is worth **15 points** (out of 100 points total).

For this portion of the assignment, you should submit:

1. a main program, `undistort_image.m`, that accepts either an image pixel array or image filename, and vector of calibration parameters, and produces an undistorted result.

Part 3: Inserting Virtual Objects into the Real World

Your third (fun) task is to insert a virtual 'reference frame' (three coordinate axes) into one of the undistorted images of the calibration target, to visually see just how straight the lines on the target actually are, and hence evaluate the quality of your calibration result. This portion of the assignment is worth **10 points** (out of 100 points total).

We'll leave this portion of the assignment fairly open. You will need to provide a single program that overlays the reference frame on top of the image, similar to the example shown below. With knowledge of the absolute size of the calibration target, you may pick a suitable size for the reference frame axes (typically red for x, green for y, and blue for z).

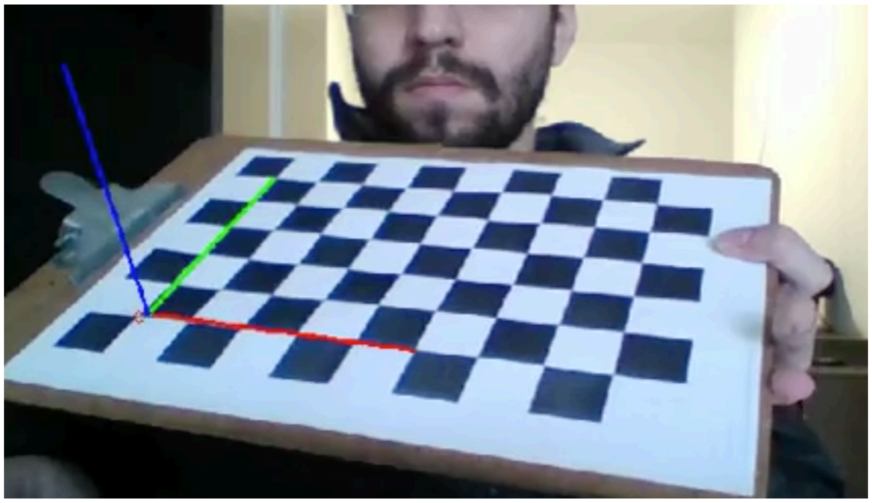


Figure 1: Virtual reference frame overlaid on image of calibration target.

Grading

Points for each portion of the project will be assigned as follows:

Part 1: Calibrating a Camera for Robotics Applications

- Perspective camera pose DLT function – **10 points**
- Checkerboard cross-junction detection and ordering function – **15 points**
- Jacobian function (for all parameters) – **15 points**
- Nonlinear optimization routine – **15 points**
- Calibration Toolbox program – **15 points**
- Coding style, appearance, and adequate commenting – **5 points**

Total: **75 points**

Part 2: Undistorting Images

- Accurate inverse distortion model that produces 'straight' image results – **10 points**
- Proper resizing of the input image to incorporate all undistorted image pixels – **5 points**

Total: **15 points**

Part 3: Inserting Virtual Objects into the Real World

- Successful insertion of virtual reference frame of proper size into image – **5 points**
- Correct alignment of frame axes with calibration target – **5 points**

Total: **10 points**

The total number of points for the assignment is **100 points**. Grading criteria include: correctness and succinctness of the implementation of support functions, proper overall program operation, and correct calibration results (subject to some variation). Note that there is a (limited) subjective to component to the grading scheme—we will judge your reference frame insertion results based on appearance. Please also note that we will test your code *and it must run successfully*, so please do not forget to include all required program files in your submission.