Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Project #2: Camera Pose Estimation

Fall 2017

Overview

Camera pose estimation is an exceedingly common task for robotic vision applications—we typically wish to know the pose (position and orientation) of the camera in a visual scene at all times. The general problem of pose estimation can be tricky (because you need to know something about scene geometry), and we will explore it later in the course. In this project, you will learn how to very accurately estimate the pose of the camera relative to a known object, in this case a planar checkerboard camera calibration target of fixed size. The goals are to:

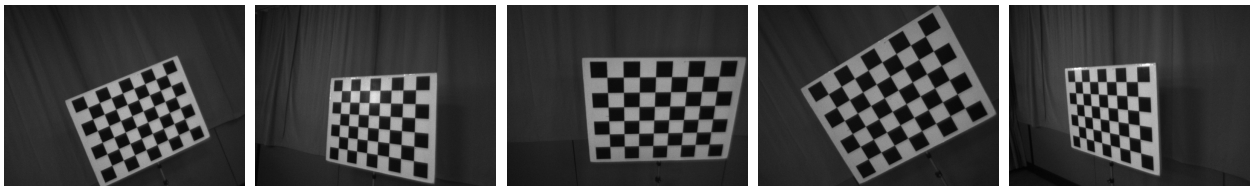
- provide practical experience with image smoothing and subpixel feature extraction, and
- assist in understanding nonlinear least squares optimization methods.

The due date for project submission is **Sunday, October 22, 2017, 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: Image Smoothing and Subpixel Feature Extraction

For problems in which the pose of the camera must be known with a high degree of accuracy (e.g., camera tracking for high fidelity 3D reconstruction), it is not unusual to insert a known calibration object into the scene; knowledge of the geometry of the calibration object can then be used to assist in pose estimation (assuming the object is visible). To determine the camera lens parameters, it is very common to use a planar checkerboard target of a known size—features on the checkerboard that lie at specific positions can be extracted reliably for pose estimation.

In this project, you will estimate the camera pose relative to a checkerboard target whose squares are **63.5 mm** in size (on a side). You may assume that the checkerboard is perfectly flat, that is, the z coordinate of any point lying on the board is exactly zero (in the frame of the target). Sample images are shown at the bottom of the page. You may also assume that each image has already been unwarped to remove any lens distortion effects.



(a) (b) (c) (d) (e)

Figure 1: Sample images for use in testing your pose estimation algorithm.

To determine the camera pose, you will need to carefully extract a set of known feature points from an image of the target. Correspondences between the observed 2D image coordinates and the known 3D coordinates of the feature points then allows the pose to be determined (see Part 2). Typically, for a planar checkerboard target, *cross-junction* points are used, for two reasons, 1) they are easy to extract, and 2) they are *invariant* to perspective transformations. A cross-junction is defined as the (ideal) point at which the diagonal black and white squares meet. In the sample images, the number of cross-junctions is $8 \times 6 = 48$. This portion of the assignment is worth **60 points** (out of 100 points total).

There are variety of possible ways to identify the cross-junctions, which we will discuss in the lecture. Usually, 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)*, vol. 2, (Singapore), pp. 191–195, December 2002.

The saddle point is the best subpixel estimate of the true position of the cross-junction. Note that images are smoothed with a Gaussian filter prior to computing the saddle points—this is done because, in many cases, the cross-junction points are not uniquely defined (due to, e.g., image quantization errors). If you zoom in on one of the sample images, you may notice this effect.

Your first task for the project is to implement a MATLAB function that returns an ordered list (row-major) of the cross-junction points on the target, to subpixel precision as determined by the saddle point detector. In every case, we will provide a bounding polygon that encloses the target; the upper left cross-junction should be taken as the origin of the target frame, with the x-axis pointing to the right in the image and the z-axis extending out of the page. Using this information, you may compute the metric coordinates of each (ideal) cross-junction.

For this portion of the assignment, you should submit:

1. a function, `gaussian_blur.m`, which filters the supplied image with a symmetric Gaussian kernel of fixed standard deviation. The filtered image should be the same size as the original, and of the same MATLAB class (e.g., integer). Hint: you may consider using the built-in MATLAB function `conv2.m`,
2. a function, `saddle_point.m`, which accepts a small image patch and attempts to find a saddle point using the algorithm described in the paper above (i.e., you should implement the algorithm). Note that the image coordinates of the saddle point should be returned to subpixel precision (i.e., as doubles), and
3. a function, `cross_junctions.m`, which determines the images coordinates of the individual cross-junctions on each checkerboard with subpixel accuracy and orders the points in sequence (row-major order).

Note that *the ordering of the cross-junction points* relative to the origin of the calibration target is critically important—the same order is required for each calibration image, for the next step (nonlinear optimization) to work correctly. Note also that, as with Project #1, for this portion of the assignment you may make use of an external feature detector.

Please clearly comment your code, and ensure that you use the filenames/function stubs specified exactly. We will run your code.

Part 2: Camera Pose Estimation

Upon completing Part 1 of the project, you should have a function (`cross_junctions.m`) that produces a series of 2D-3D feature correspondences, which can be used for pose estimation. You will implement pose estimation using a nonlinear least squares procedure that incorporates all of the available data. This portion of the assignment is worth **40 points** (out of 100 points total).

The solution to the **PnP problem** (i.e., Perspective-n-Points) is described in Section 6.2 of the Szeliski text. Although there is a linear algorithm for the problem, it does not work when all points are coplanar. Instead, we will provide you with an initial guess for the camera pose ($\pm 10^\circ$ and 20 cm, approximately). You will need to know the camera intrinsic calibration matrix, which in this case is:

$$\mathbf{K} = \begin{bmatrix} 564.9 & 0 & 337.3 \\ 0 & 564.3 & 226.5 \\ 0 & 0 & 1 \end{bmatrix}$$

where the focal lengths and principal point values are in pixels. This is sufficient information to obtain the rotation and translation of the camera relative to the target.

The final step is to set up, and then solve, the nonlinear system of equations for pose estimation, using nonlinear least squares. Your parameter vector should incorporate updates to the translation parameters, and also a *minimal representation* of the updates to the orientation parameters. To assist with this, the `.zip` archive for the project includes a series of functions to parameterize the camera orientation using roll-pitch-yaw Euler angles (and to compute the required Jacobians). You may use this parameterization, or another parameterization if you prefer.

For this portion of the assignment, you should submit:

1. a function, `pose_estimate_nlopt.m`, which accepts a set of 2D-3D correspondences (image-target) and an initial guess for the camera pose, and performs a nonlinear least squares optimization procedure to compute an updated, optimal estimate of the camera pose, and
2. a function, `find_jacobian.m`, which computes the Jacobian matrix required for nonlinear least squares. If you choose to use an orientation parametrization other than Euler angles, you should *very clearly document this in the comments in this file.*

For testing, you may use the set of images included in the project archive (examples above). Once submitted, we will evaluate your estimator on a *hold out* set of images. If you wish, to assist in debugging, you may use the MATLAB plotting tools available on Portal (under `Projects/`) to visualize your pose estimates, etc.

Please clearly comment your code, and ensure that you use the filenames/function stubs specified exactly. We will run your code.

Grading

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

Part 1: Image Smoothing and Subpixel Feature Extraction

- Gaussian image filtering function – **10 points**
- Accurate saddle point detector implementation – **15 points**
- Cross-junction function finds 'correct' set of target points (outlier-free) – **15 points**

- Cross-junction function produces set of subpixel correspondences – **5 points**
- Cross-junction correctly orders point correspondences — **10 points**
- Coding style, appearance, and adequate commenting – **5 points**

Total: **60 points**

Part 2: Camera Pose Estimation

- Correct Jacobian matrix for each iteration of nonlinear least squares – **15 points**
- Nonlinear least squares implementation that computes optimal camera pose – **25 points**

Total: **40 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.