Calibration on eye camera and scene camera

h. chi

February 9, 2012

Contents

About this document 8 on page 9.

This document was finished quickly as a draft before paternity leave, and to be updated later.

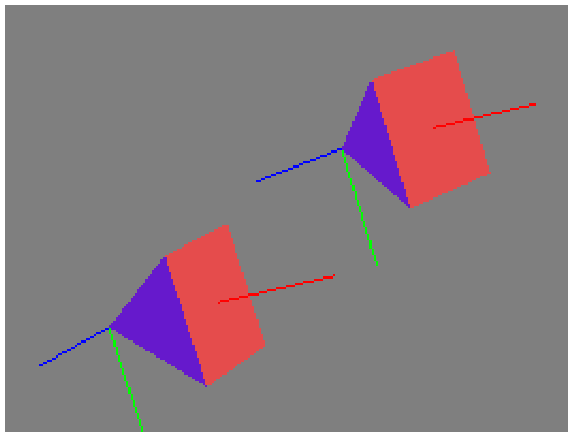

Figure 1: example of two cameras and their coordinates

1 Introduction

The objective of this calibration is to determine the pose of one camera with respect to the other, and in this work, it's chosen arbitrarily to determine the scene camera's pose in the eye camera coordinate system. Figure 1 presents an example of two cameras and their associated right handed coordinate systems, whose X, Y, and Z axises are represented by blue, green, and red lines pointing from origin to X, Y, and Z positive directions, respectively.

A rig made by Protoshop was used as reference in this work, and an OpenGL simulation program was made to mimic the set up, a screen capture from the simulation is shown in figure 2 on the following page . The main purpose of the rig is to hold two circle calibration pattern boards, one small pattern for eye camera and the other big pattern for scene camera. The key point is that Protoshop's high precision manufacturing leads to known geometry of the whole rig, i.e, the pose of one calibration pattern with respect to the other calibration pattern is considered known. On the other hand, we have techniques to estimate pose of eye camera with respect to the small calibration pattern, as well as scene camera pose with respect to the big pattern board. And consequently, we combine those relations to determine the scene cam pose in the coordinate system of the eye camera.

A comment on figure 2 on the next page

The left camera in figure 2 on the following page is eye camera, however, it's pose is out of our interest, due to the use of a mirror, shown as a gray square in the figure. As the figure shows, the eye cam is posed to face the mirror so that it has a good view on the mirror image of the small calibration board. Equivalently, we consider a virtual camera residing in the mirror, facing directly to the small calibration board. Such a virtual eye camera's pose, with respect

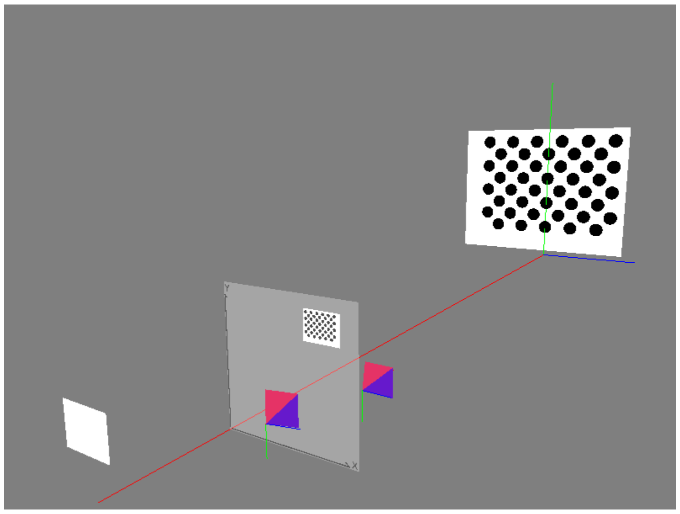

Figure 2: Simulation of Protoshop rig

to the small calibration board, is of our interest.

2 Notation and math expression of the problem

2.1 Camera coordinate system

The coordinate system associated with a camera is shown already in figure 1 on the previous page.

2.2 Calibration pattern coordinate system

The origin of a coordinate system associated with a circle pattern, as shown in figure 3 on the following page, is located at the center of the upper left circle (when the pattern is placed up right), and this circle is index as zero, the next circle to its right (when the pattern is placed up right) is index by one, etc. The X axis leaves center of circle zero, goes to circle one, show as a blue line in the figure, and the Y axis the red line. Z axis is determined by right hand rule.

2.3 Change of coordinate

Figure 1 on the previous page can be used to illustrate change of coordinate systems. The right side camera's pose, with respect to the left side camera, can be described by a translation component and a rotation component, and both the translation and rotation are described in the left camera's coordinate. Translation is commonly given by translation vector $T = [t_x, t_y, t_z]^T$, and rotation

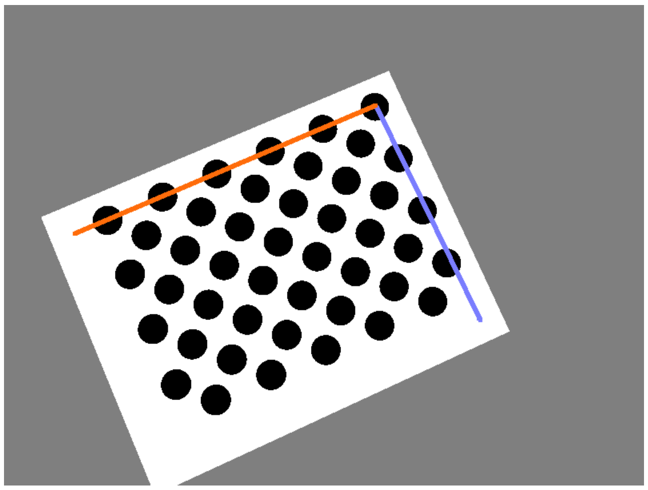

Figure 3: Pattern coordinate system

matrix

$$R = \begin{bmatrix} r_{1,x} & r_{2,x} & r_{3,x} \\ r_{1,y} & r_{2,y} & r_{3,y} \\ r_{1,z} & r_{2,z} & r_{3,z} \end{bmatrix}$$

Obviously, T gives the position of the right camera's coordinate system's origin in the left camera's coordinate system, and columns of R are unit vectors on X, Y, and Z axises of the right side camera's coordinate system, described in the left side coordinate.

As homogeneous coordinates are often used, T and R are combined into one 4×4 transformation matrix $M_{l,r} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$, where subscript l, r indicates translation/rotation from $l$eft camera coordinate system to the $r$ight.

2.4 Translation and rotation from eye virtual camera to scene camera

Using above notation, the Protoshop-rig-assistant calibration is to compute

$$M_{evc,sc} = M_{evc,b} \cdot M_{b,s} \cdot M_{s,sc} \tag{1}$$

where $M_{evc,sc}$ is the 4×4 transformation matrix from $e$ye $v$irtual $c$amera to $s$cene $c$amera, $M_{evc,b}$ from eye virtual camera to $b$ig calibration pattern, $M_{b,s}$ from big pattern to small pattern, and $M_{s,sc}$ from small pattern to $s$cene $c$amera.

3 Method

According to equation (1), key task in this calibration problem is to find relative pose between a camera and a calibration pattern facing to it. In our current solution, OpenCV function solvePnP is used for this purpose.

4 C++ implementation

to be written later.

5 Result

The work resulted in two programs, one relies on OpenCV for calibration, and the other relies OpenGL to simulate the rig and cameras, etc. Images and coordinates are exported from simulator and passed to calibrator for validation and debugging purpose.

5.1 OpenGL simulator

In simulation, the poses of the two cameras, referring to figure 2 on page 3, are first initialized and then kept constant, in other words, the two camera's poses are fixed, after initialization, with respect to the mirror coordinate system, whose X and Y axises are shown in figure 2 on page 3. The mirror is moved and rotated properly so that the scene camera produces good images of the big calibration pattern, meanwhile, the eye virtual camera produces pictures of the mirror images of the small pattern. A pair of such images are used to obtain the pose of scene camera in terms of eye camera's coordinate system, given as equation (1).

5.2 OpenCV calibrator

to be written later

5.3 Camera parameters

Parameters of a camera include focal length in X pixels, f_x, in Y pixels, f_y, and center of image (u_x, u_y), as well as distortion parameters. For ideal camera, parameters can be derived easily according to OpenGL perspective model. In real case, they are obtained by camera calibration, which is a part of the overall calibration of this work. The third means is to run this calibration codes on images exported from OpenGL simulator.

All those three approaches are used in the work.

5.4 Simulation result

An experiment result is shown in figure 4 on the following page. The experiment settings are:

- 61 image pairs are captured

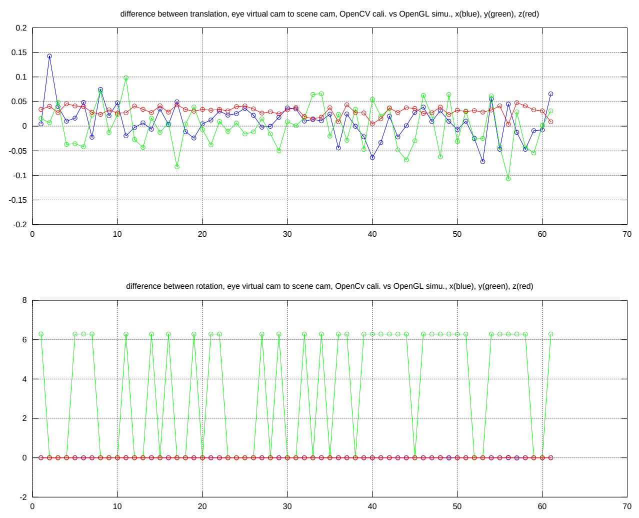

Figure 4: simulation vs calibration, translation and rotation, 61 image pairs

- Camera parameters are obtained by calibration using images captured separately from OpenGL simulation

5.4.1 Interpreting the figure

- There is a difference around magnitude of 6 in the rotation part (subplot in the bottom of the figure). It's not an error, but a result of a matter of notation. The reason is that, a rotation vector $[0, 0, \pi]$ is equivalent to vector $[0, 0, -\pi]$, but they produce a distance of 2π.
- For translation part, noise-like effect is observed, because the results on different pairs of images vary around certain fixed value.

5.4.2 Averaging to remove noise

The noise-like effect implies use of a running average. Figure 5 on the next page shows the result after running average is applied to translation component.

Running average method is effective for X and Y component, but not for Z. The reason is that, implied by figure 4, the estimate of Z component is subjected to a bias (offset).

6 Discussion

The bias in estimates of Z component of the translation is an interesting source of error, as it's not removable by averaging. The OpenGL simulator and OpenCV

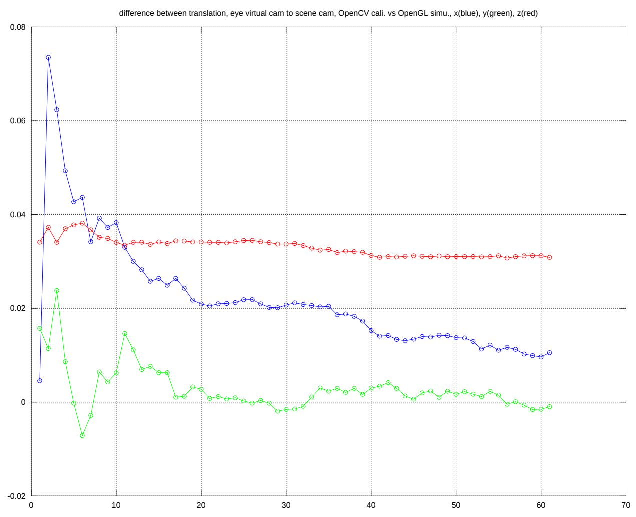

Figure 5: Simulation vs calibration, translation, 61 image pairs (averaging)

calibrator are studied carefully to trace the error. It's found that OpenCV function solvePnP is sensitive to noise contained in its input.

6.1 SolvePnP

In order to test solvePnP function, a OpenGL simulator program was made to capture calibration pattern and to export images and coordinates, and the OpenCV program was made to test solvePnP function which takes images captured from simulator and produces coordinates.

The coordinates produced by OpenCV program are compared with those coordinates form OpenGL program (considered true value) . Figure 6 on the next page presents the result (ideal camera parameters used), it shows that the estimates in X and Y components of translation have small error, but the Z components have a noticeable bias. This is highly likely the source of error in forgoing discussion.

6.1.1 Noise in image points

The SolvePnP function has 3 input:

1. ***object points***: coordinates of the centers of the circles of a calibration pattern, with respect to the pattern coordinate system.

   (a) object points coordinates are integers and do not have any chance to contain errors.

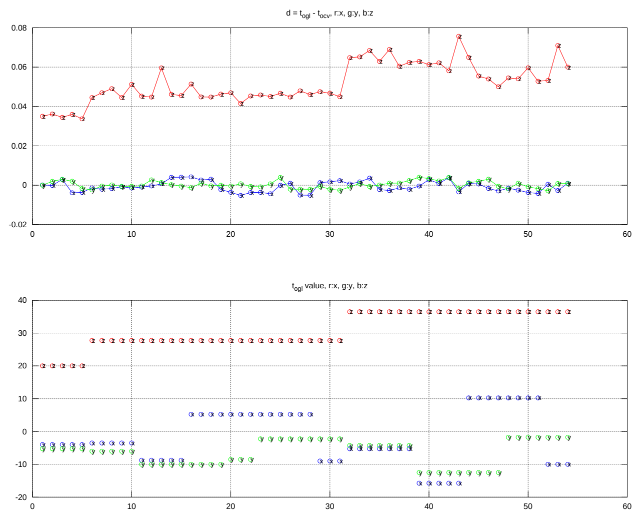

Figure 6: test solvePnP

2. ***image points***: pixel coordinates of the projections of those above centers onto the image plane of the camera, at sub pixel accuracy

3. camera parameters, including f_x, f_y, u_x, u_y and distortion vector.

   (a) Further study shows that using ideal camera parameters to replace parameters given by calibration does not improve the result.

Therefore, it must be the noise in image points that produces errors. The testing uses 54 image pairs, in each pair, OpenGL compute projections of circle centers onto image plane. When OpenCV takes an image, it identifies circles, and then determine their centers to accuracy in sub pixel level. The OpenGL projections are considered as true values, and the difference between true values and the estimates from OpenCV are considered noise. The noise is not obvious, as shown in figure 7 on the following page, red circles represents image points returned by OpenCV and blue points by OpenGL, it shows that they almost overlap.

Data analysis shows that the maximum difference in X and Y is less than one pixel.

6.1.2 replace SolvePnP's input of image points by true values

When the true values of image points are computed in OpenGL and used to replace those estimated by OpenCV procedure, the result becomes much better, as shown in figure 8 on the next page.

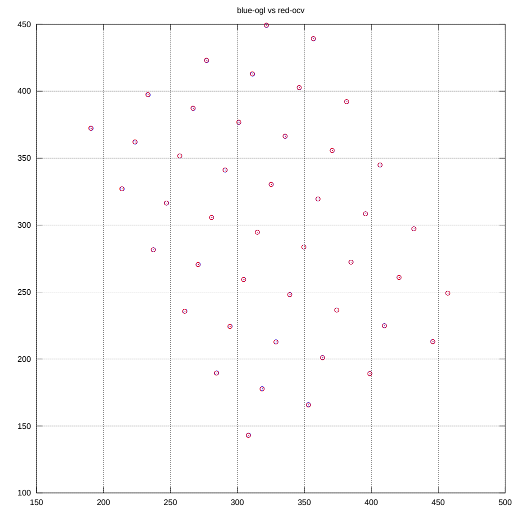

Figure 7: noise: image points, OpenCV vs. OpenGL

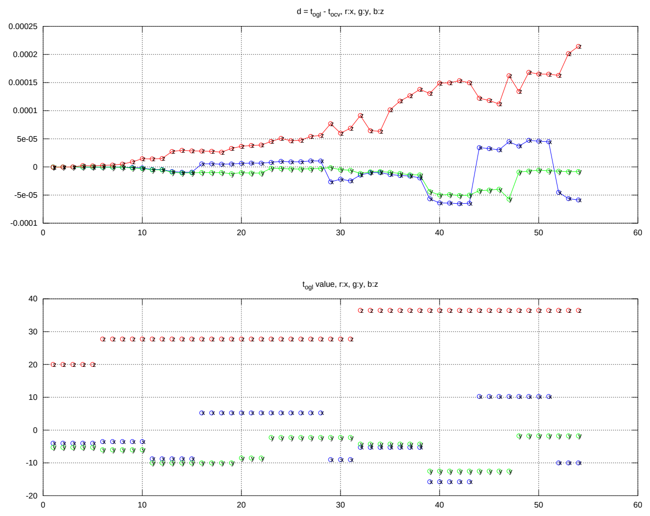

Figure 8: use of true values for image points

6.2 Error and distance between camera and pattern

Figure 8 on the preceding page implies that the shorter the distance between camera and pattern board, the more accurate the estimates.

7 Past and future work