ARTICLE IN PRESS

J. Vis. Commun. Image R. xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Automatic, fast, online calibration between depth and color cameras

Ilya V. Mikhelson [a,*], Philip G. Lee [a], Alan V. Sahakian [a,b], Ying Wu [a], Aggelos K. Katsaggelos [a]

[a] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA
[b] Department of Biomedical Engineering, Northwestern University, Evanston, IL 60208, USA

ARTICLE INFO

Article history:
Received 13 September 2012
Available online xxxx

Keywords:
Calibration
Depth cameras
Color cameras
Automatic
Online
Fast
Point cloud
Point correspondence

ABSTRACT

Automatic camera calibration has remained a hard topic in computer vision since its inception due to its reliance on the image correspondence problem. This problem becomes even more pronounced when calibrating a depth image with a color image due to a lack of simple correspondences between the two modalities. In this work, we develop a completely automatic, very fast, online algorithm that demonstrates how a consumer-grade depth camera can be calibrated with a color camera with minimal user interaction.

1. Introduction

1.1. Background

Camera calibration is a basic problem in computer vision, consisting of finding the relative position and pose of all the cameras to one another. This is important for inferring 3D structure [1], aligning camera arrays in computational photography [2], and more. Due to the recent release of the consumer-grade Microsoft Kinect™ depth sensor, there are also many vision algorithms that aim to take advantage of the fusion of RGB and point cloud scene structure. For example, in the problem of remote vital sign monitoring, careful aim has to be maintained at the subject's chest [3]. Computer vision along with depth information have been used for this purpose [4].

The basic technique for conventional camera calibration is to image a 2-D target or a 3-D target in distinct orientations where image-to-image point correspondences can be easily obtained either manually or semi-automatically [5]. This target is commonly the checkerboard pattern, for which numerous corner extraction algorithms exist, given the rough manually selected location of the target region [6]. Recent improvements have also made checkerboard extraction automatic, e.g. [6,7].

For point cloud to color image calibration, the pinhole camera model gives the mapping from a homogeneous point in the world $\boldsymbol{P} = [x, y, z, 1]^T$ to homogeneous image coordinates $\boldsymbol{p} = [u, v, 1]^T$ as

$$w\boldsymbol{p} = \mathbf{K}[\mathbf{R}, \mathbf{t}]\boldsymbol{P} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & -f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_1^T & t_x \\ \mathbf{r}_2^T & t_y \\ \mathbf{r}_3^T & t_z \end{bmatrix} \boldsymbol{P}, \quad (1)$$

where $\mathbf{K}$ is the intrinsic camera matrix, $\mathbf{R}$ is the camera rotation matrix, and $\boldsymbol{t} = -\mathbf{R}\boldsymbol{c}$ with $\boldsymbol{c}$ being the pinhole location in world coordinates. To model the radial and tangential distortion function introduced by the lens, this linear model is usually concatenated with a truncated Taylor expansion.

Since depth cameras, like the Kinect™, can directly give values for world coordinates $\boldsymbol{P}$, it seems at first glance that it should be easier to obtain the calibration parameters of the model, but depth cameras present their own problems. First of all, it is not easy – even manually – to provide point cloud to RGB correspondences due to lack of visual information in the point cloud. These correspondences must come from structural corners that are present in the point cloud that are also visible in RGB. However, the second major problem with depth-camera-generated point clouds is that they provide extremely noisy and incomplete information at such corners and depth edges due to the measurement method (structured light). This creates the paradox that points which should be visible in both the color and depth images are exactly those whose information is missing in the point cloud.

Many algorithms have been developed to solve these problems and to calibrate a depth and color camera pair. In the field of time-of-flight (ToF) cameras, which measure distance by pulsing a light source and measuring the reflection delay, recent work includes [8,9]. In the field of structured light cameras, such as the Kinect™, recent work includes [10–13]. All of these methods require user

* Corresponding author.
E-mail address: i-mikhelson@u.northwestern.edu (I.V. Mikhelson).

1047-3203/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jvcir.2013.03.010

interaction in some form, usually marking points in the color or depth image manually.

1.2. Motivation for our work

The main drawback with all of the methods that we have found is that it is a tedious process. In our experience, calibration, especially extrinsic calibration, has to be performed often due to the sensitivity of camera systems to even the slightest nudge. This is exacerbated when the setup is on a mobile platform. In addition, recalibration has to be performed if the setup of the cameras is changed altogether. In our applications with remote vital sign monitoring, the setup is moved around a lot since it is on a mobile platform which is transported to various environments for testing, creating small changes in the orientation of the cameras which is problematic.

We recognize that once the intrinsic parameters have been found, there is no need to recalibrate them. Therefore, we present a calibration technique where the extrinsic parameters can be found very quickly, usually in under a minute, with no tedious work on the part of the user.

Using our method, not only can we mitigate the aforementioned paradox, wherein we find direct point correspondences between the point cloud and the color image to perform calibration, but this happens very rapidly, online, and without requiring the user to manually specify *any* point correspondences.

2. Methods

Our goal is to project points from the point cloud created by any depth camera into the color camera frame of reference. In order to do this, we need to estimate the 14 parameters of the pinhole projection model with lens distortion (up to 4th order lens distortion approximation): $\Pi = \{f_u^C, f_v^C, c_u^C, c_v^C, t_x^{D\to C}, t_y^{D\to C}, t_z^{D\to C}, \phi_R^{D\to C}, \theta_R^{D\to C}, \psi_R^{D\to C}, \kappa_1^C, \kappa_2^C, \nu_1^C, \nu_2^C\}$, where f_u, f_v, c_u, and c_v make up the intrinsic matrix $\mathbf{K}$ in Eq. (1); t_x, t_y, t_z make up the extrinsic translation vector $\boldsymbol{t}$ in Eq. (1); ϕ_R, θ_R, and ψ_R are (ZXY) Tait–Bryan rotation angles that parameterize $\mathbf{R}$ in Eq. (1); and κ_1 and κ_2 are radial distortion coefficients, and ν_1 and ν_2 are tangential distortion coefficients. The superscript C denotes a parameter of the color camera, a superscript D denotes a parameter of the depth camera, and a superscript $D \to C$ denotes a transformation from the depth camera frame to the color camera frame.

In order to perform calibration, we use a color image and a point cloud obtained simultaneously. The point cloud can come from any source, e.g. a ToF camera, a laser range finder, a Kinect™. The target we use is a checkered, planar (as opposed to textured) square. We use such a target because it is simple to create and does not require any special materials. A flowchart of the automatic calibration process can be seen in Fig. 1.

2.1. Intrinsic parameters

2.1.1. Color camera intrinsic parameter estimation

Since color camera calibration is well-studied topic, we use the Bouguet toolbox [14] for MATLAB [15] with an automatic checkerboard-finding algorithm [7]. This allows us to keep the whole algorithm automatic while obtaining very precise intrinsic parameters and distortion coefficients for the color camera. This step eliminates the need to find $f_u^C, f_v^C, c_u^C, c_v^C, \kappa_1^C, \kappa_2^C, \nu_1^C$, and ν_2^C, since these are returned by the toolbox.

2.1.2. Depth camera intrinsic parameter estimation

For our work, we use OpenNI™ drivers, which create a point cloud from the Kinect's™ native disparity map. Since our work is relevant to any point cloud, we did not directly work with the Kinect's™ disparity map. The point cloud we retrieve is created from a depth image using the Kinect's™ intrinsic parameters:

$$x = \left(\frac{(u^D - c_u^D)}{f^D}\right)z, \tag{2}$$

$$y = \left(\frac{(\nu^D - c_\nu^D)}{f^D}\right)z, \tag{3}$$

$$z = z, \tag{4}$$

where $[x, y, z]^T = \boldsymbol{P}$ is a real-world coordinate, $[u^D, \nu^D]^T = \boldsymbol{p}^D$ is a pixel coordinate in the depth frame of reference, c_u^D and c_ν^D define the principal point in the depth frame of reference, and f^D is the focal length of the depth camera.

The OpenNI™ drivers return a 3-dimensional image as a 3-channel matrix, where each (u, ν) point in the matrix has a corresponding (x, y, z) value in the 3 channels. Since the Kinect™ is only able to directly acquire the z-channel, this means that the x- and y-channels are calculated according to Eqs. (2) and (3).

Since we get an image of x, y, z, u^D, and ν^D values, we estimate the intrinsic depth camera parameters (c_u^D, c_ν^D, and f^D) that were used to create the point cloud. We do this by minimizing the reprojection error between points in the point cloud that we retrieve and points in a point cloud that we create using only the z values of the retrieved point cloud and Eqs. (2) and (3):

$$\operatorname{argmin}_{c_u^D, c_\nu^D, f^D} \sum_{i,j} \left[\left(\frac{(u_{ij}^D - c_u^D)}{f^D}\right) z_{ij} - x_{ij} \right] + \left[\left(\frac{(\nu_{ij}^D - c_\nu^D)}{f^D}\right) z_{ij} - y_{ij} \right], \tag{5}$$

where i and j index the image.

We assume for our work that the distortion coefficients for the depth camera are all zero. In the work of Smisek et al. [10] and Herrera et al. [12], it was shown that there is indeed a distortion in the depth image. However, we did not find significant improvement

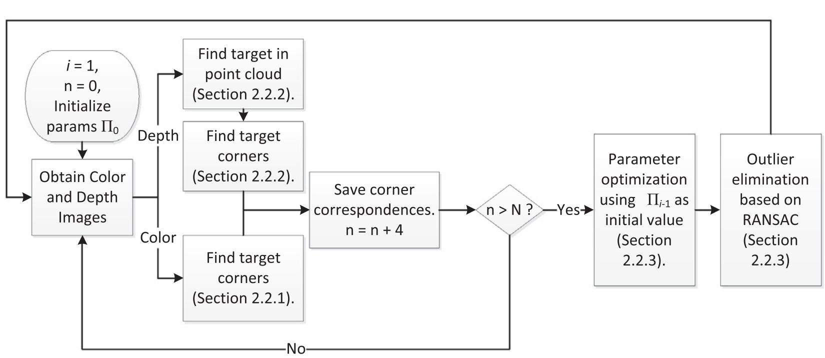

Fig. 1. Flowchart of algorithm.