# SCREEN-CAMERA CALIBRATION USING A THREAD

*Songnan Li, King Ngi Ngan, Lu Sheng*

Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong SAR

## ABSTRACT

In this paper, we propose a novel screen-camera calibration algorithm which aims to locate the position of the screen in the camera coordinate system. The difficulty comes from the fact that the screen is not directly visible to the camera. Rather than using an external camera or a portable mirror like in previous studies, we propose to use a more accessible and cheaper calibrating object, i.e., a thread. The thread is manipulated so that our algorithm can infer the perspective projections of the four screen corners on the image plane. The 3-dimentional (3D) position of each screen corner is then determined by minimizing the sum of squared projection errors. Experiments show that compared with the previous studies our method can generate similar calibration results without the additional hardware.

***Index Terms—*** Screen-camera calibration, Hough line detection, RANSAC algorithm

## 1. INTRODUCTION

The screen-camera calibration method described in this paper aims to find the position of the screen with respect to the camera. As illustrated in Fig. 1 (a), the objective is to locate the 3D positions $(x_i, y_i, z_i)$, $i \in \{1, 2, 3, 4\}$ of the four screen corners in the camera coordinate system. The screen position information can be used together with a 3D head tracking algorithm (e.g. [1]) to develop a holographic-like displaying system, a demo of which can be found in our project website [2]. Other applications of the screen-camera calibration include using the screen as a controllable planar light source for shape recovery and reflectance measure, enabling the user's interaction with a specific region of the screen.

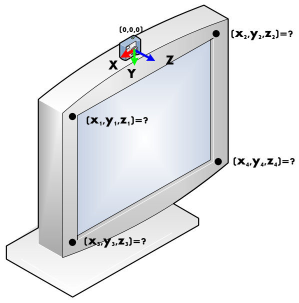
In these applications, the screen and camera usually need to be positioned to face a similar direction. The screen is not directly visible to the camera, which makes the standard calibration algorithms using checkerboard patterns [9] infeasible. To handle this problem, in [3] an external camera which faced towards the screen was added. Since the screen was visible to this external camera, the geometric calibration between them can be done. Observing the same pattern, the extrinsic calibration between the first camera and the external one can be calculated. Then, using the external camera to make the connection, the relative position between the screen and the camera was determined indirectly. Another approach is to use a mirror (planar [4] or spherical [5, 6]) so that the camera can see the screen's reflection. Multiple images were captured by the camera. In each of these images the 3D position of the mirror was estimated, and the screen's reflection region was identified. Screen corners can be found by, for example, intersecting reflected light rays from mirrors in several images. In [7, 8], the authors proposed to use the cornea of the human eye as a mirror. The principle is similar with that of using a spherical mirror [5, 6]. The main advantage is the absence of an external camera or a mirror. However, the camera resolution needs to be high (e.g., $2448 \times 2048$ [8]), since the human eye region usually is very small in an image.

In reality, it is typical that a user has no hardware like an additional camera or a portable mirror, and the resolution of the user's camera is not high enough for calibration. So in this paper, we propose a novel method for screen-camera calibration which uses a very common calibrating object – a thread. Motivation and problem formulation are explained in Section 2. Some implementation issues are discussed in Section 3. The resultant MATLAB program can be downloaded from our project website [2]. Section 4 shows the experimental results. Finally Section 5 draws the conclusion.

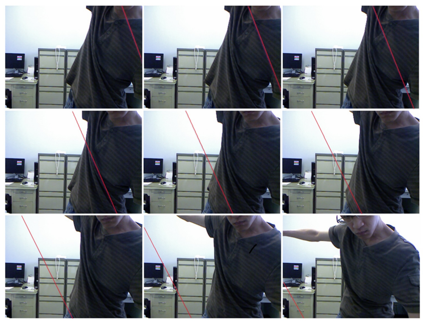
## 2. PROBLEM FORMULATION

As discussed above, the main difficulty of our problem is that the screen is not visible to the camera. Rather than using an external camera or a mirror, the proposed method uses a more common and cheaper calibrating object to infer the screen position. The motivation is based on a simple fact that when using a camera to capture straight lines that intersect at the same 3D point, say $\mathbf{x}$, the projections of the lines will intersect at a 2D point which actually is the projection of $\mathbf{x}$ on the image plane.

Take our algorithm as an example. By using a thread we can get a bunch of straight lines that intersect at a screen corner. Specifically, one end of the thread is pressed onto a screen corner. The other end is stretched, and moved around in front of the camera. The camera above/below the screen takes pictures of the thread at different time slots, as shown in Fig. 1 (b). The projections of the thread will intersect, as illustrated in Fig. 1 (c), and the intersection point will be the perspective projection of this screen corner on the image plane. The pro-



(a)



(b)

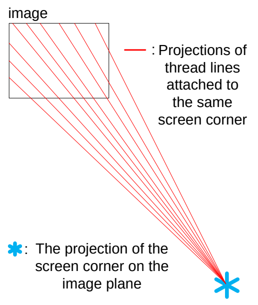


image
: Projections of thread lines attached to the same screen corner
: The projection of the screen corner on the image plane

(c)

**Fig. 1**. (a) Screen-camera calibration is to get the 3D positions of the four screen corners with respect to the camera. (b) One end of a red thread is pressed onto a screen corner. The other end is moved in front of the camera. For illustration, the color of the red thread is enhanced. (c) The projections of the red thread attached to the same screen corner will intersect. The intersection point will be the projection of the screen corner on the image plane.

jections of the other screen corners can be located in a similar way. Since the screen corners are not directly visible to the camera, their projections are outside the image area.

The prior knowledge about the screen includes that its four corners form a rectangle. And its width and height can be measured easily using a ruler. By making them strict constraints, the positions of the four screen corners in the camera coordinate system, i.e., $\mathbf{P}_1$,$\mathbf{P}_2$,$\mathbf{P}_3$,$\mathbf{P}_4$, can be defined as:

**upper-left:** $\mathbf{P}_1 = [x_1, y_1, z_1]^T$

**upper-right:** $\mathbf{P}_2 = \mathbf{P}_1 + \mathbf{R} \times [w, 0, 0]^T$

**bottom-left:** $\mathbf{P}_3 = \mathbf{P}_1 + \mathbf{R} \times [0, h, 0]^T$

**bottom-right:** $\mathbf{P}_4 = \mathbf{P}_1 + \mathbf{R} \times [w, 0, 0]^T + \mathbf{R} \times [0, h, 0]^T$

where $\mathbf{P}_1$ is the 3D position of the upper-left corner; $\mathbf{R}$ is a $3 \times 3$ rotation matrix which has three degrees of freedom, i.e., $\theta_x$,$\theta_y$,$\theta_z$, corresponding to the pitch, yaw, and roll rotations, respectively; $w$ and $h$ are the known width and height of the screen. Therefore, the positions of the four screen corners can be determined by the six parameters, i.e., $x_1$,$y_1$,$z_1$,$\theta_x$,$\theta_y$,$\theta_z$.

In the proposed algorithm, the six parameters are determined by minimizing the sum of squared projection errors of the four screen corners on the image plane. Theoretically, the lines for each corner should intersect at exactly the same point, as illustrated in Fig. 1 (c). However, the positions of the lines will be contaminated by noise. Their intersection point can be calculated by minimizing the Sum of Squared Distances (SSD) between the intersection point and its associated lines

$$min_{\mathbf{m}_i} \sum_{j=1}^{N_i} ||D(\mathbf{m}_i, l_i^j)||^2, i \in \{1, 2, 3, 4\} \tag{1}$$

where $\mathbf{m}_i$ is the 2D intersection point; $N_i$ is the total number of lines attached to corner $i$ and $l_i^j$ is the $j^{th}$ line. Assuming that line $l$ is defined by $Ax + By + C = 0$ and there is a 2D point $\mathbf{p}$=($x_p$,$y_p$), the function $D(\mathbf{p}, l)$ calculates the shortest distance from point $\mathbf{p}$ to line $l$

$$D(p, l) = \frac{|Ax_p + By_p + C|}{\sqrt{A^2 + B^2}} \tag{2}$$

Eq. (1) has a closed form solution.Thus, an intuitive formulation to minimize the sum of squared projection errors is

$$min_{x_1,y_1,z_1,\theta_x,\theta_y,\theta_z} \sum_{i=1}^{4} ||D(\hat{m}(\mathbf{P}_i), \mathbf{m}_i)||^2 \tag{3}$$

where function $\hat{m}(\cdot)$ gets the perspective projection [13] of the screen corner position $\mathbf{P}_i$, $i \in \{1, 2, 3, 4\}$ on the image plane with the camera's intrinsic parameters known. However, by using Eq. (3), it is implicitly assumed that the projections have no directional bias. In fact, the projections tend to locate along the line direction as can be seen from Fig. 2 (a) and (b). Therefore, rather than using Eq. (3), we can calculate the six parameters of the screen position by minimizing the SSD between the projection point of each screen corner and the associated lines using the following equation

$$min_{x_1,y_1,z_1,\theta_x,\theta_y,\theta_z} \sum_{i=1}^{4} \sum_{j=1}^{N_i} ||D(\hat{m}(\mathbf{P}_i), l_i^j)||^2 \tag{4}$$

Eq. (4) is a nonlinear least squares problem, which in our implementation is solved using the Levenberg-Marquardt algorithm [14].

## 3. IMPLEMENTATION DETAILS

The MATLAB implementation of the proposed algorithm can be downloaded from the project website [2], where a video demonstrating its usage can also be found. Some implementation details are highlighted in this section.

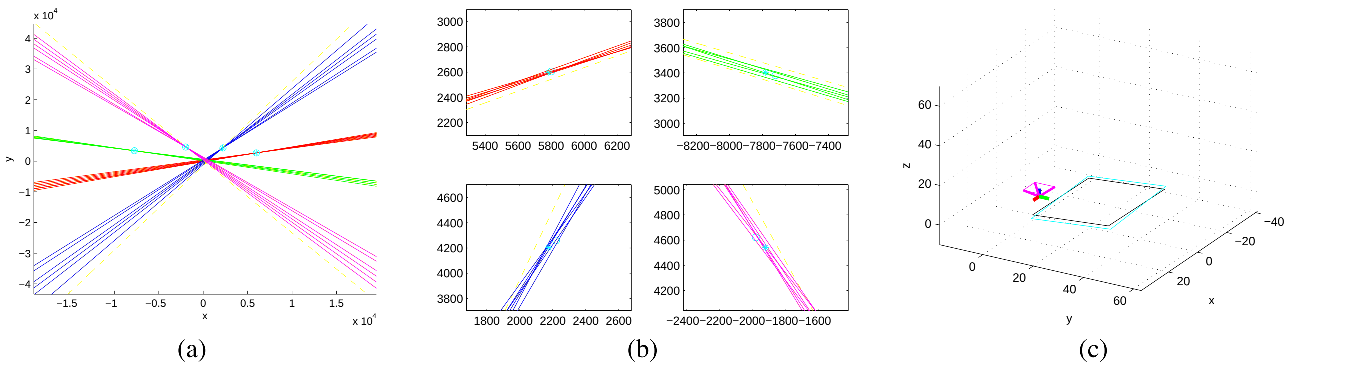


**Fig. 2**. (a) The red, green, blue, and magenta colored lines are associated with the upper-left, upper-right, bottom-left, and bottom-right corners, respectively. The dashed yellow lines are the outliers rejected by RANSAC. The cyan stars indicate the intersection points of the lines. The cyan circles are the perspective projections of the 3D corner positions generated by the proposed method. (b) Zoom-in images of (a) around the intersection point for each screen corner. (c) Calibration results comparison between the proposed method (black) and that of [3] (cyan).

### 3.1. Line detection using Hough transform

As shown Fig. 1 (b), we need to locate the thread in the images. A semi-automatic method is employed which requires the user to approximately indicate the line's position. This initial position will be refined automatically using the Hough line detection method.

More precisely, the user only needs to provide the rough positions of the left and right ends of the thread by clicking on the image. To refine the user's inputs, firstly the color difference between each pixel and its neighbors is calculated in the CIELAB color space. By thresholding this color difference image, a binary edge map is obtained. Hough transform [10] is then applied to this edge map. Finally, a local window search is performed on the Hough transform histogram with the initial line position as the window's center. The local maximum within the window indicates the number of edge pixels on the final line.

We found that this empirical line detection method works well for both thin and thick lines (caused by e.g. motion blur or out-of-focus blur). The thread can be replaced by other calibrating objects as long as they can be conveniently attached to the screen corner and produce images of straight lines. A more automatic line detection method that needs less user's interaction will be our future work.

### 3.2. Outlier rejection using RANSAC

The line detection can fail occasionally due to reasons like complex background, motion blur, out-of-focus blur, etc. Therefore, there will be outliers in the line detection results. To reject outliers, our program shows the detection result to the user, and allows the user to decide whether or not to (1) redo the line detection, (2) manually determine the line position, or (3) skip the current image.

Another strategy to automatically reject outliers using the RANSAC algorithm has also been implemented. A RANSAC toolbox [11, 12] is used. The RANSAC algorithm contains an iterative process. In each iteration, it randomly selects a small subset of the lines to calculate their intersection, and using this intersection point to distinguish outliers. After a large number of iterations, it becomes very likely that in one of these iterations, the selected subset contains only inliers. And the intersection point produced by this subset of lines should lead to the minimum number of outliers.

### 3.3. Parameter initialization

The minimization given by Eq. (4) requires a good initialization of the parameters, which in our implementation has a analytical solution. Take the position of the upper-left corner as the origin, and make the upper-right and bottom-left corner lie on the positive X and Y axis of the world coordinate system, respectively. The positions of the four screen corners in the world coordinate system can be given by

$$\mathbf{M}_1 : [0,0,0]^T, \mathbf{M}_2 : [w,0,0]^T, \mathbf{M}_3 : [0,h,0]^T, \mathbf{M}_4 : [w,h,0]^T$$

Let $\mathbf{R}$ and $\mathbf{t}$ denote the camera's rotation and translation matrix w.r.t. the world coordinate system. Then we have

$$\mathbf{P}_i = \mathbf{R} \times \mathbf{M}_i + \mathbf{t}, i \in \{1,2,3,4\} \quad (5)$$

Therefore, to get the 3D positions of the screen corners in the camera coordinate system is equivalent to calculating $\mathbf{R}$ and $\mathbf{t}$. To this end, we use a technique similar to that of [9] to get a closed form solution for $\mathbf{R}$ and $\mathbf{t}$. Generally, the first step is to compute the homography matrix $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$ that correlates $\mathbf{M}_i$s with their projections $\mathbf{m}_i$s

$$s\tilde{\mathbf{m}}_i = \mathbf{H} \times \tilde{\mathbf{M}}_i, i \in \{1,2,3,4\} \quad (6)$$

where $s$ is a scale factor. The projection $\mathbf{m}_i$ is calculated by Eq. (1), and $\tilde{\mathbf{m}}_i$ denotes the homogeneous representation of $\mathbf{m}_i$, i.e., adding 1 as the last element of $\mathbf{m}_i$. On the other hand, $\tilde{\mathbf{M}}_i$ denotes a vector that replaces the last element of $\mathbf{M}_i$ (i.e., 0) with 1. In the next step, we can use $\mathbf{H}$ and the camera's intrinsic parameter matrix $\mathbf{A}$, which is assumed to be known, to calculate $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]$ and $\mathbf{t}$

$$\mathbf{r}_1 = \lambda\mathbf{A}^{-1}\mathbf{h}_1, \mathbf{r}_2 = \lambda\mathbf{A}^{-1}\mathbf{h}_2, \mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2, t = \lambda\mathbf{A}^{-1}\mathbf{h}_3$$

where $\lambda = 1/||\mathbf{A}^{-1}\mathbf{h}_1||$ in our implementation. $\mathbf{R}$ calculated in this way does not in general satisfies the properties of a rotation matrix. We use the method described in [9] to estimate the best rotation matrix from $\mathbf{R}$. For more implementation details please refer to our code. It should be noted that similar with Eq. (3) this closed form solution does not consider the directional bias of the projections. Therefore, a further optimization using Eq. (4) is necessary.

## 4. EXPERIMENT

A Dell 1908FPb 19" LCD display was used in our experiment. The camera was a Microsoft Kinect for XBOX360 which was fixed on top of the display using a bracket. The resolution of the color camera is $640 \times 480$. Kinect also has a depth camera which is not used by our algorithm. To facilitate the line detection, we suggest selecting the color of the thread to be distinguishable from the background. In our experiment, a red thread was used.

For each corner, seven images like the ones shown in Fig. 1 (b) were used for calibration. Fig. 2 (a) illustrates the line detection and outlier rejection results. The red, green, blue, and magenta colored lines are associated with the upper-left, upper-right, bottom-left, and bottom-right corners, respectively. All lines were detected using the semi-automatic method described in Section 3.1. The outliers were selected by the RANSAC algorithm. The dashed yellow lines shown in Fig. 2 (a) are the rejected outliers. We calculated the intersection point of the lines (inliers only) associated with each corner by minimizing SSD, as given by Eq. (1). The cyan stars of Fig. 1 (a) indicate the four intersection points. Using the proposed algorithm, the 3D positions of the screen corners can be determined. By perspectively projecting them onto the image plane, we obtained the cyan circles in Fig. 2 (a). As expected, the cyan stars and circles are close to each other, which is consistent with the objective of the optimization given by Eq. (4). Fig. 2 (b) shows zoom-in images of Fig. 2 (a) around the intersection point for each screen corner.

For comparison, we implemented another screen-camera calibration method in which an external camera was required [3], the principle of which has been introduced in Section 1. Fig. 2 (c) shows the calibration results comparison between the two methods. The red, green and blue bars indicate the XYZ directions in the camera coordinates. The magenta lines indicate the position of the camera. The black colored lines represent the screen position located by using the proposed method, while the cyan colored lines represent the screen position located by using the method of [3]. It can be observed that these two methods generated similar results. However, our method does not need the additional hardware. The distance between corner positions calculated by the two methods is listed in Table 1. The maximum distance is around 2 centimeters (cm). It should be noted that the results of [3] cannot be considered as the ground truth. Actually getting the ground truth is extremely difficult if possible. A more comprehensive comparison should include as many screen-camera methods as possible. This will be one of our future works.

**Table 1**. The distance between corner positions calculated by the proposed method and that by [3].

| | Upper-left | Upper-right | Bottom-left | Bottom-right |
|---|---|---|---|---|
| Distance (cm) | 1.19 | 1.93 | 1.94 | 1.81 |

## 5. CONCLUSION

We propose a novel screen-camera calibration algorithm which uses a thread to infer the screen's position. One end of the thread is pressed onto the screen corner, and the other end is moved in front of the camera. Using a semi-automatic line detection method, the thread lines are located on the image plane. The outliers of the line detection results are automatically rejected using the RANSAC algorithm. The inliers are used to infer the perspective projections of the screen's corners. By enforcing the rectangular shape and measuring the physical width and height of the screen, the screen position can be determined with six parameters, whose values are derived by minimizing the sum of squared projection errors. The merit of our algorithm is that it uses more accessible and cheaper calibrating object while generating similar calibration results as shown in the experiments.

## 6. REFERENCES

[1] S. Li, K.N. Ngan, L. Sheng, "A head pose tracking system using RGB-D camera", in Proc. International Conference on Computer Vision Systems, pp. 153-162, Russia, 2013.

[2] Project website, available at: `http://www.ee.cuhk.edu.hk/~snli/screen_camera_cali.htm`

[3] C. Zhang, Q. Cai, P.A. Chou, Z. Zhang, R. Martin-Brualla, "Viewport: A Fully Distributed Immersive Teleconferencing System with Infrared Dot Pattern", IEEE Transactions on Multimedia, vol. 20, no. 1, pp. 17-27, 2013.

[4] N. Funk and Y.H. Yang, “Using a raster display for photometric stereo”, in Proc. Canadian Conference on Computer and Robot Vision, pp. 201207, 2007.

[5] Y. Francken, C. Hermans, and P. Bekaert, “Screencamera calibration using a spherical mirror”, in Proc. Canadian Conference on Computer and Robot Vision, pp. 1120., 2007.

[6] Y. Francken, C. Hermans, and P. Bekaert, “Screencamera calibration using Gray codes”, in Proc. Canadian Conference on Computer and Robot Vision, pp. 155161, 2009.

[7] C. Nitschke, A. Nakazawa, and H. Takemura, “Display-camera calibration using eye reflections and geometry constraints”, Comput. Vis. Image Underst., vol. 115, no. 6, pp. 835853, 2011.

[8] C. Nitschke, A. Nakazawa, and H. Takemura, “Practical DisplayCamera Calibration from Eye Reflections using Coded Illumination”, in Proc. Asian Conference on Pattern Recognition, pp. 550 554, 2011.

[9] Z. Zhang, “A flexible new technique for camera calibration”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.22, no.11, pp. 1330-1334, 2000.

[10] R. C. Gonzalez, R. E. Woods, “Digital Image Processing 2nd Edition”, Prentice Hall, 2002.

[11] RANSAC toolbox, available at: `http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/RANSAC.shtml`

[12] M. Zuliani, C. S. Kenney, B. S. Manjunath, “The Multiransac Algorithm and its Application to Detect Planar Homographies”, in Proc. IEEE International Conference on Image Processing, pp. 153-156, Italy, 2005.

[13] R. Hartley, A. Zisserman, “Multiple View Geometry in Computer Vision Second Edition”, Cambridge University Press, March 2004.

[14] J. More, “The Levenberg-Marquardt Algorithm: Implementation and Theory”, Numerical Analysis, G.A. Watson, ed., Springer-Verlag, 1977.