A Lightweight Library for Augmented Reality Applications

Luis Gerardo de la Fraga[1](✉), Nataly A. García-Morales[2], Daybelis Jaramillo-Olivares[1], and Adrián J. Ramírez-Díaz[1]

[1] Computer Science Department, Cinvestav, Mexico City, Mexico
fraga@cs.cinvestav.mx

[2] Electrical Engineering Department, Cinvestav, Bioelectronics Unit, Av. IPN 2508, 07360 Mexico City, Mexico

Abstract. To build an Augmented Reality (AR) application it is necessary to recognize a fiducial marker, then to calibrate the camera that is viewing the 3D scene on the marker, and finally to draw a virtual object over the image taken by the camera but in the virtual coordinate system supposed also on the fiducial marker. The camera calibration step give us the transformation matrix from 3D world to 2D on the screen, and the pose of the marker with respect to the virtual coordinate system. An AR application must run interactively with the user, and also in real time. Performing all these calculations in a embedded device such as a Single Board Computer (SBC), a tablet, or a smartphone, is a challenge because a normal numerical analysis library is huge, and it is not designed for such devices. In this article we present a lightweight numerical library, it has been developed thinking in such computing restricted devices. We show results on two AR applications developed for the Raspberry Pi 3 SBC.

Keywords: Augmented reality · Fiducial marker
Camera calibration · Homography estimation

1 Introduction

Augmented reality is a technology which superimposes computer generated images on top of a user's perception of the real world in real time [7]. AR has important applications in fields such as videogamming, interactive marketing and advertising, instructional aids and how-to for use, construction and maintenance, and navigation [7].

Perhaps one of most successful application of AR could be in education [3]. As a new technology, AR could help to students to learn more effectively and increase knowledge retention, relative to traditional 2D desktop interfaces. With the aid of AR could be build the virtual interaction with complex phenomena (showing the magnetic field, or the Earth layers, for example).

In [4] authors analyze an AR application on sixty nine middle-school students. Their results show an increase in student's motivation, attention and motivation factors or the learning environment based on augmented reality technology compared with a more traditional learning environment. Authors also mention that AR is not mature enough to be used massively in education.

Users in AR application need to learn how to build 3D scenes, how to draw virtual objects, and how to manage, although amazing and exciting, a technology that is complex.

OpenCV [2] is the open tool to learn Image Processing, Computer Vision, and 3D user interfaces. This is the facto standard in this field. This library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. But is the OpenCV library is far to be an easy tool, in its homepage there are listed 52 books about how to learn it.

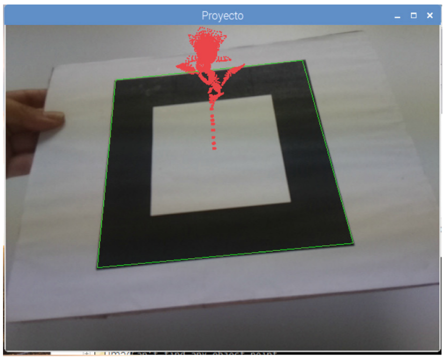
The idea of this paper is to offer a simpler way to process fiducial markers. In Sect. 2 is presented the necessary components to build an AR application. Sect. 3 describes our proposed library that can be used in computationally restricted devices. Sect. 4 shows an AR application build with our library and running on the Raspberry Pi 3, a SBC that uses a 1.2 GHz 64-bits quad-core ARM Cortex-A53 CPU and 1 GB of RAM. Finally, in Sect. 5, some conclusions are drawn.

2 Augmented Reality Application

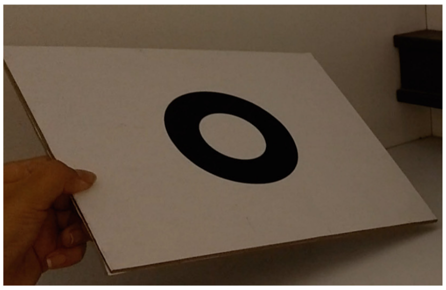
The following steps are performed in a typical AR application: (1) An image processing step to recognize a fiducial marker, (2) A computer vision step to calibrate the camera, this is, to obtain the projection matrix that transforms the 3D world viewing by the camera to a 2D plane that forms the viewed image; this step also obtains the pose, this is, the rotation and translation of the marker with to a virtual and global coordinate system. And finally, the step (3) that draws a virtual object over the coordinate system fixed in the marker. The augmented world is visualized only on the display, where it is possible to see the projection of the virtual objects on the image generated by the camera.

Two very simple fiducial markers are shown in Fig. 1: one is a simple black square, and the other is a pattern of two concentric circles [10]. The white area in the background of both markers help them to be recognized by a computer: it is a high contrast zone on the image. The black zone is supposed to be one of the biggest black objects on the scene. Marker detection is performed with image processing techniques: (1) a global threshold is applied to the input image which produce a binary (blank and white) image, (2) some morphological erosion and dilatation operations are applied to remove noise on the binarized image, and a labeling of the all black components in the image is performed. These image processing tasks were applied using library in [6].

Once we have the markers, the vertices positions must be calculated with the marker in Fig. 1(a), and the points of each ellipse must the extracted from marker in 1(b). With these data the homography can be estimated. Our library helps to perform easily these steps. Now, these Computer Vision steps will be described in detail.



(a) A black square marker



(b) Two concentric circles marker

Fig. 1. Images of two very simple fiducial markers in use.

3 Description of the Lightweight Library

Homography calculation. To obtain a homography by the *normalized DTL* algorithm [9] it is necessary to solve an overdetermined system of equations that forms a matrix of size $2n \times 8$, where n is the number of point correspondences between the marker model and an image of the same model. The easiest way to solve this problem is using the QR decomposition computed with the modified Gram?Schmidt algorithm [8]. In fact, this is the shortest code and easiest codification to solve the QR decomposition. If the overdetermined system of equations is expressed as $A\mathbf{h} = \mathbf{b}$, $\mathbf{h}$ is found using normal equations doing:

$$\begin{aligned} QR\mathbf{h} &= \mathbf{b}, \\ (QR)^{\mathrm{T}}QR\mathbf{h} &= (QR)^{\mathrm{T}}\mathbf{b}, \\ R^{\mathrm{T}}Q^{\mathrm{T}}QR\mathbf{h} &= R^{\mathrm{T}}Q^{\mathrm{T}}\mathbf{b}, \\ R\mathbf{h} &= Q^{\mathrm{T}}\mathbf{b} = \mathbf{c}. \end{aligned} \tag{1}$$

Here the system is already solved because matrix R is upper triangular and $\mathbf{h}$ values are found by back-substitution, staring with $h_{n-1} = c_an/r_{nn}$.

We use here a further way to compact the calculations, such as it was suggested also by Golub and Van Loan [8]: the QR decomposition is calculated to the extended matrix $[A \mid b]$, of size $2n \times 9$, then one obtains $Q[R \mid Q^{\mathrm{T}}b]$, thus the product $Q^{\mathrm{T}}b$ is obtained in the last column. Notice here that matrix Q is not needed, thus the subroutine to solve the $A\mathbf{h} = \mathbf{b}$ is as (using the same notation that in [8]):

A notice here that is important. The calculation of the homography is a very well conditioned problem, then the use of the QR algorithm is justified. The only way to get a pour conditioned problem is to use repeated points correspondences to try to calculate the homography.

Eigendecomposition of a symmetric matrix of size 3×3. The eigendecomposition of a symmetric matrix A results in the matrices $V\ D\ V^{\mathrm{T}}$, where

Algorithm 1. Solving $A\mathbf{h} = \mathbf{b}$ using QR decomposition

```
Require: Matrix $A$ of size $2n \times 8$, vector $\mathbf{b}$ of size $2n$
Ensure: Vector $\mathbf{h}$
 1: $B = [A \mid b]$                                  ▷ $B$ has size $2n \times 9$
 2: for $k = 1:9$ do
 3:     $R(k,k) = \|B(1:2n,k)\|_2$                  ▷ Norm of column vector $k$ of $B$
 4:     $\mathbf{v} = B(1:2n,k)/R(k,k)$
 5:     for $j = k+1:9$ do
 6:         $R(k,j) = \mathbf{v}^{\mathrm{T}} B(1:2n,j)$
 7:         $B(1:2n,j) = B(1:2n,j) - \mathbf{v}R(k,j)$
 8:     end for
 9: end for
10:                                                  ▷ Back substitution stage:
11: $h(8) = B(8,9)/B(8,8)$
12: for $i = 7:1$ do
13:     $sum = 0$
14:     for $j = i+1:8$ do
15:         $sum = sum + B(i,j)\mathbf{h}(j)$
16:     end for
17:     $h(i) = (B(i,9) - sum)/B(i,i)$
18: end for
```

D is a diagonal matrix former with the eigenvalues of A, and V is a orthogonal matrix where its columns are the eigenvectors corresponding to each eigenvalue in D. This problem is simplified with matrices of size 3×3 because is equivalent to find the roots of a cubic equation [6,12].

SVD of a matrix of size 3×3. This problem is used to solve the orthogonalization of a matrix, specifically when this matrix is a matrix obtained with a linear method using a plane [13] or a circular marker [10]. The calibration of a camera with both linear methods produces a rotation matrix R far of being orthogonal, then to become orthogonal such matrix its Singular Value Decomposition is applied: $R = UD_1V^{\mathrm{T}}$, and $R' = UV^{\mathrm{T}}$ is obtained, where R' is already orthogonal. In [13] it is demonstrated that the obtained R' is the best orthogonal matrix which minimizes the Frobenius norm of $R' - R$.

Fitting an ellipse to a set of points. The fastest algorithm to fit a set of point to an ellipse is by solving a least square problem that minimizes the sum of squared algebraic distances [5]. Instead to solve this problem as an eigendecomposition of a 6×6 matrix, as it is in [5], this problem can be transformed easily to solve three times a eigendecomposition of 3×3 matrices, or to find three times the roots of three cubic equations [6].

Camera calibration using a marker of two concentric circles. This marker is presented in [10]. First, it is necessary to recognize two ellipses. For this task the previous method in the last paragraph can be used. With the two recognized ellipses, the homography can be obtained directly as it is explained in [10]. Here it is necessary to invert a 3×3 matrix, thus its eigendecomposition can be used because the used matrices in this method are the representation of a conic equation, which is symmetric and of 3×3 size.

4 Experiments and Results

To test our library against the results produced with a high level language, such as Octave, to demonstrate the correctness of its calculations. We generate two simulated images, each one from a square and two ellipses as the planar models of the markers. From these models we use the pinhole camera model in (2) with a known camera position to generate the two images shown in Fig. 2(b) and (c). The used pinhole camera model is

$$\lambda \mathbf{p} = KR[I| - \mathbf{c}]\mathbf{P}, \tag{2}$$

where $\mathbf{p} = [u, v, 1]^{\mathrm{T}}$ is a point over the image, $\mathbf{P} = [x, y, z, 1]\mathrm{T}$ is a point in the 3D scene, $\mathbf{c}$ is the position of the camera, and R is a rotation matrix. Markers are situated on the xy-plane, then $\mathbf{P} = [x, y, 0, 1]^{\mathrm{T}}$ and the camera model is reduced as:

$$\begin{aligned} \lambda \mathbf{p} &= KR[\mathbf{e}_1,\ \mathbf{e}_2,\ -\mathbf{c}]\mathbf{P}, \\ \lambda \mathbf{p} &= K[\mathbf{r}_1,\ \mathbf{r}_2,\ -R\mathbf{c}]\mathbf{P}, \\ \lambda \mathbf{p} &= H\mathbf{P}, \end{aligned}$$

where I is the identity matrix $I = [\mathbf{e}_1,\ \mathbf{e}_2,\ \mathbf{e}_3]$, and $R = [\mathbf{r}_1,\ \mathbf{r}_2,\ \mathbf{r}_3]$, and we are abusing the notation on $\mathbf{P}$ to still denote a homogeneous 2D point on the model $\mathbf{P} = [x, y, 1]\mathrm{T}$. The used camera intrinsic parameters are in matrix K:

$$K = \begin{bmatrix} 1000 & 0 & -300 \\ 0 & 1000 & -200 \\ 0 & 0 & -1 \end{bmatrix},$$

to generate images of size 600×400 pixels, the principal point is situated in its center at $(300, 200)$.

Then we apply both codes, in C with out light library and with Octave, to estimate both homographies H, and recover the rotation matrix and camera position. Results are shown in Table 1.

Table 1. Results of camera calibration and pose estimation using the markers in Fig. 2

Language, marker	f	$R = R_z(\theta_3)R_y(\theta_2)R_z(\theta_1)$ $(\theta_3, \theta_2, \theta_1)$	Camera center
Ground truth	1000.0	(90.00, 57.69, -108.43)	$[20.00, -60.00, 40.00]^{\mathrm{T}}$
Octave (square)	1043.1	(89.44, 58.44, -107.70)	$[19.97, -62.71, 40.45]^{\mathrm{T}}$
C (square)	1043.1	(89.44, 58.44, -107.31)	$[19.74, -63.43, 40.44]^{\mathrm{T}}$
Octave (circles)	1012.0	(89.95, 57.70, -114.19)	$[26.20, -58.40, 40.50]^{\mathrm{T}}$
C (circles)	1005.5	(90.04, 57.67, -114.37)	$[26.27, -57.98, 40.30]^{\mathrm{T}}$

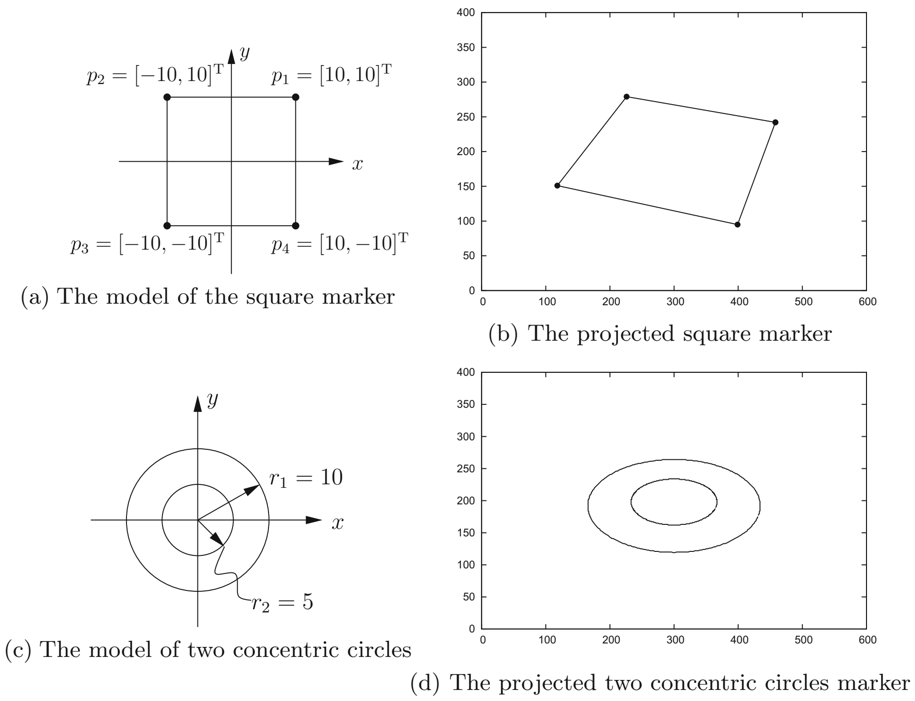

$p_2 = [-10, 10]^T$
$p_1 = [10, 10]^T$
$p_3 = [-10, -10]^T$
$p_4 = [10, -10]^T$
x
y
(a) The model of the square marker
(b) The projected square marker
$r_1 = 10$
$r_2 = 5$
(c) The model of two concentric circles
(d) The projected two concentric circles marker

Fig. 2. The square and two concentric circles markers and their projections used in the experiment. Images in (b) and (d) are generated with the parameters detailed in text.

Details for this first experiment are as follows: The square model has vertices $\{(10, 10), (-10, 10), (-10, -10), (10, -10)\}$. The two circles models have radius 10 and 5. Both markers are shown in Fig. 2(a) and (c), respectively. Both codes, in C and Octave, for this experiment are available in http://cs.cinvestav.mx/~fraga/LightLib.tar.gz.

The results shown in Table 1 are not the best with respect to the ground truth. This is because two reasons: (1) The marker images in Fig. 2(a) and (c) where generated rounding the pixels locations to the nearest integers, and (2) The pose calculated using the homography is based in a linear method which is not the best solution. The solutions obtained and shown in Table 1 must be refined with a non-linear method to improve their values. An alternative that will be explored as future work is to use a PnP algorithm such as the one in [11,14] to calculate the pose. The inconvenience of using a PnP algorithm is that the camera must be calibrated in advance.

With the library were programmed two augmented reality applications shown in Fig. 3. A virtual object is shown above the corresponding marker. It is possible to move the marker in the real world, or to move, carefully, the camera. The virtual object follows the marker on the monitor screen. We used images with a resolution of 640×480 pixels, and then we obtained a processing frame rate of 30 frames per second. The Raspberry Pi 3 camera has a buildin autofocus feature [1], thus camera must be recalibrated at time to time. In our applications we recalibrate the camera at every frame because it is possible that focus change due this autofocus characteristic. In applications in Fig. 3 OpenGL 2.1 was used (this is the version that supports the Raspberry Pi 3) using glut for input/output. The glut function `glutTimerFunc` implements the timer that guide the main loop at 30 fps. Still could be possible to add a intelligent behavior to the application: perhaps it is not necessary to recalibrate if the reprojection error of the marker vertices is not to high. We are going to check this last possibility in the near future.

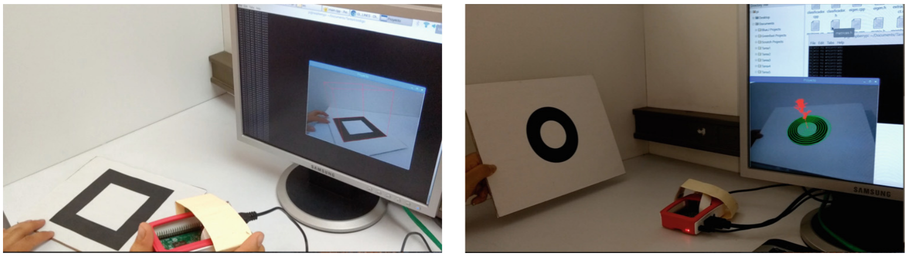


Fig. 3. Two pictures from two applications of augmented reality working on the Raspberry Pi 3

In OpenCV documentation in [2] it is available the SVD but not the QR decomposition. It seems now that OpenCV uses its own SVD implementation. In previous versions it used old Blas/Lapack numerical libraries (these libraries are used also by Octave). These old numerical libraries are written in Fortran language. We believe our small library uses memory more efficiently than Blas/Lapack and because of this reason, it can run very fast.

5 Conclusions

We have developed a light numerical library which can be applied in augmented reality application. We can calibrate a camera and calculate the pose for two different markers.

This library is programmed in C language and is intended for computing restricted devices such as smartphones, tablets and single board computers.

Results with our library are almost the same that the ones obtained with Octave that is a high level numerical language.

We tested our library in the Raspberry Pi 3 SBC. A frame rate of 30 frames per second with a camera resolution of 640 × 480 pixels was obtained.

As a future work we think it is necessary a function to calculate the marker pose solving the PnP problem [14]. PnP problem solves the pose if camera intrinsic parameters are known. The method to obtain the pose based in the homography, described in this article, is a linear method which has not the best results.

Otherwise, it is necessary to refine the solution using a non-linear method, which could be prohibitive in a real time application. It is necessary to investigate more about this problem of pose detection.

References

1. Camera model v2 documentation. https://www.raspberrypi.org/documentation/hardware/camera/. Accessed 9 Oct 2017
2. Open source computer vision library (OpenCV). http://opencv.org. Accessed 18 Sept 2017
3. Billinghurst, M., Dünser, A.: Augmented reality in the classroom. IEEE Mag. Comput. **45**, 56–63 (2012)
4. Di Serio, A., Blanca Ibáñez, M., Delgado Kloos, C.: Impact of an augmented reality system on students motivation for a visual art course. Comput. Educ. **68**, 586–596 (2013)
5. Fitzgibbon, A., Pilu, M., Fisher, R.: Direct least square fitting of ellipses. IEEE Trans. Pattern Anal. Mach. Intell. **21**(5), 476–480 (1999). https://doi.org/10.1109/34.765658
6. de la Fraga, L., Cruz Díaz, C.: Fitting an ellipse is equivalent to find the roots of a cubic equation. In: 2011 8th International Conference on Electrical Engineering, Computer Science and Automatic Control, pp. 1–4. IEEE (2011)
7. Gervautz, M., Schmalstieg, D.: Anywhere interfaces using handheld augmented reality. IEEE Mag. Comput. **45**, 26–31 (2012)
8. Golub, G., Van Loan, F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
9. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2003)
10. Kim, J.S., Gurdjos, P., Kweon, I.S.: Geometric and algebraic constraints of projected concentric circles and their applications to camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. **27**(4), 637–642 (2005)
11. Kneip, L., Scaramuzza, D., Siegwart, R.: A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In: 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)(2011). https://doi.org/10.1109/CVPR.2011.5995464
12. Kopp, J.: Efficient numerical diagonalization of hermitian 3×3 matrices. Int. J. Mod. Phys. C **19**, 523–548 (2008)
13. Zhang, Z.: A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. **22**(11), 1330–1334 (2000)
14. Zheng, Y., Kuang, Y., Sugimoto, S., Astrom, K., Okutomi, M.: Revisiting the PnP problem: a fast, general and optimal solution. In: The IEEE International Conference on Computer Vision (ICCV), December 2013