Camera calibration using adaptive segmentation and ellipse fitting

Raúl Romaní Flores * **Paul Alonzo, Quio Añamuro** **

* *Universidad Católica San Pablo (e-mail: raul.romani@ucsp.edu.pe).*
** *Universidad Católica San Pablo (e-mail: paul.quio@ucsp.edu.pe).*

Abstract: We describe an algorithm that Detects the control points in the raw images for camera calibration for ring calibration patterns. The algorithm was implemented in c++ 11 and using opencv library. The proposed algorithm for finding the control points consist of four parts, Apply a mask that encloses the control points in order to reduce the work area, Apply adaptive threshold in order segment the image, find contours and find ellipses. We also implemented an algorithm for drawing lines(independent of rotation), in a specific order, starting from the top-left to the top-right and going down. We have conducted an set of experiments with recorded videos and with two cameras in real time.

Keywords: Camera calibration, threshold, integral image, contours, ellipse.

1. INTRODUCTION

In many machine vision applications, a crucial step is to accurately determine the relation between the image of the object and its physical dimension by performing a calibration process. Over time, various calibration techniques have been developed. Nevertheless, the existing methods cannot satisfy the ever-increasing demands for higher accuracy performance. In this paper, we propose an algorithm for control points detection that is used in the first stages of camera calibration techniques. We use a ring calibration patterns (Fig. 1). The proposed algorithm for finding the control points consist of four parts, Apply a mask that encloses the control points in order to reduce the work area, Apply adaptive threshold in order segment the image, find contours and find ellipses. We also implemented an algorithm for drawing lines(independent of rotation), in a specific order, starting from the top-left to the top-right and going down.

We run our algorithm in 4 videos recorded with 5 different cameras (kinnect v2, mslifecam, ps3eyecam and real sense) and two of them was used to test in real time. In order to implement this algorithm we read some theory about Zhang [2000], Datta et al. [2009], and taken Prakash and Karam [2012] as a base for the creating the algorithm. Additionally we made some test of real time pattern detection using the Ps3EyeCam and MsLifeCam getting an average of 48 pfs.

2. PATTERN DETECTION

The proposed algorithm for finding the control points consist of four parts, Apply a mask that encloses the control points in order to reduce the work area, Apply adaptive threshold in order segment the image, find contours and find ellipses. We also implemented an algorithm for drawing lines(independent of rotation), in a specific order, starting from the top-left to the top-right and going down. Those ordered control points will be used later to calibrate the camera. We will describe both algorithms in detail in the next two subsections.

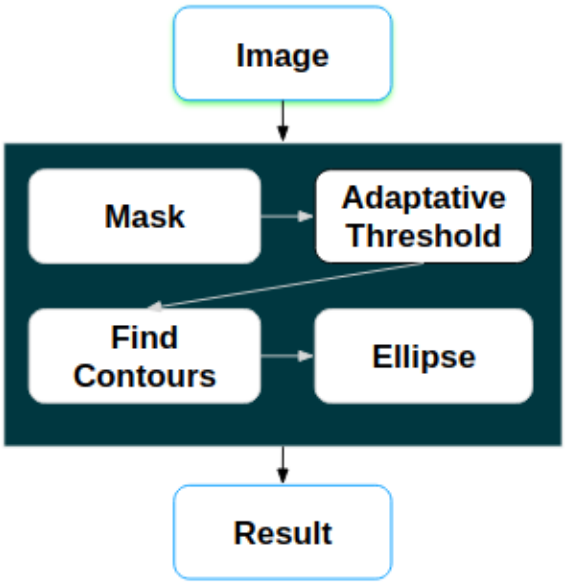

Fig. 1. Pipeline for finding control points

2.1 Mask

Apply a mask that encloses the control points in order to reduce the work area. This mask is computed and applied to the input image after we have found 20 control points, we computed this mask using minAreaRect(opencv function), this function return a rotated rectangle, we use this a mask.

Fig. 2. At left we can see a good masking after found the 20 pattern points. If we didnt find the 20 pattern points, open the mask(right).

2.2 Adaptative Threshold

Preprocess the input image/video by usign adaptive thresholding using the integral image based on Bradley and Roth [2007].

In the paper cited above doesn't say what we should do with the image borders so we usse the opencv threshold to complete that information.

Additionally we calculate the adaptive threshold using opencv to complement the information of the edges with the obtained by the threshold implemented previously.

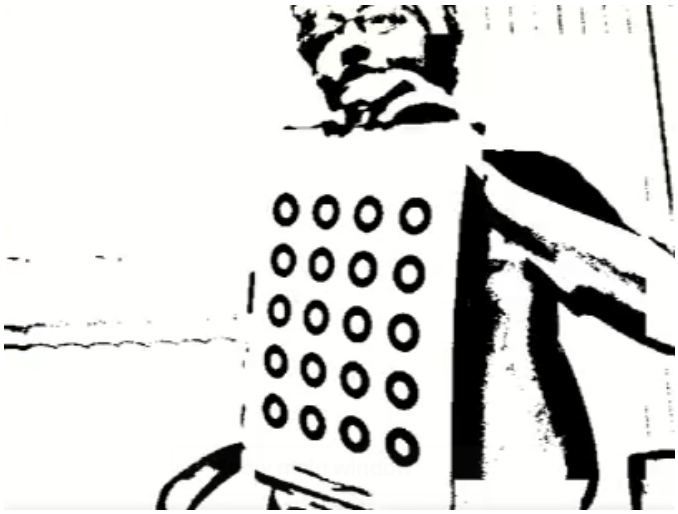

Fig. 3. Thresholding keeping good image information.

We have some troubles when the pattern is moved fast or the camera doesn't focus the pattern correctly.

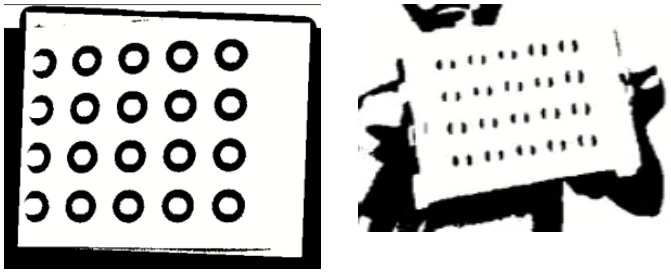

Fig. 4. At left we can see a thresholding error cause by fast movement. At right we can see a thresholding error cause by focus error.

2.3 Find contours

We use the function of openCV findContours to get all the contours from the segmented image in a hierarchy of contours, which will allow us to filter the ellipses in the next step.

Its an important step because if we don't found all pattern ellipses the next step will fail.

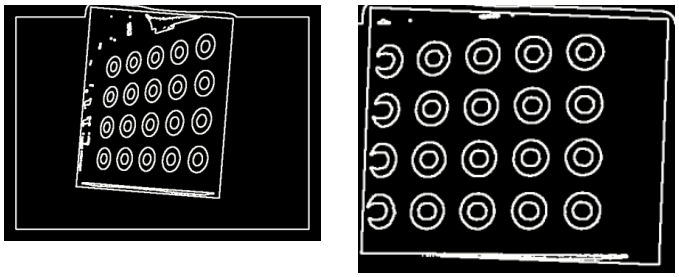

Fig. 5. At left we can see a good contours improved by thresholding. At right we can see a bad contours caused by a fast movement.

2.4 Find ellipse

- From the found contours we look for all the ellipses that have a father and a son, besides that the center of the son is very close to his own.
- From the set of ellipses found we verified that we have at least 2 ellipses near us at a distance no greater than 5 times the radius of the ellipse with which we make the comparison.
- If we have more than 20 ellipses we look for mode based on the hierarchy of the father of each ellipse and we eliminate all those who have a different father.

Fig. 6. Comparison between two ellipses to check if the ellipse c1 belong to the control point. the small Yellow line is the radio of the ellipse c2, the large Yellow line is five times the radio of the ellipse c2. We can see that condition distance(c1, c2) $<$ 5 * radio C2, does not hold, therefore the ellipse c1 does not belong to the control points.

2.5 Line representation

After we have found the rings of the pattern in the image we start to build the lines representation of the pattern. If we lost the pattern we need to order the points again using the next algorithm and if we doesn't lost the pattern

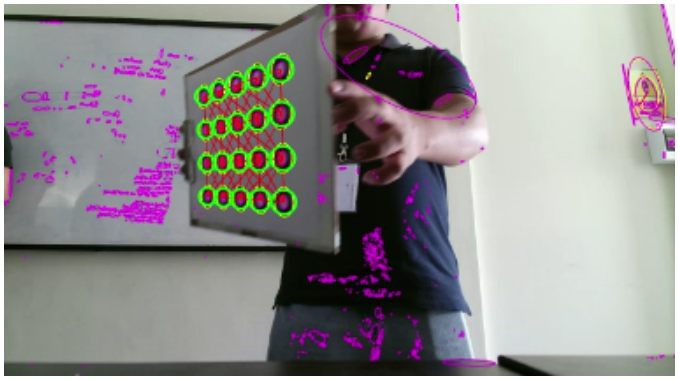

Fig. 7. Discard false positive using the father mode.

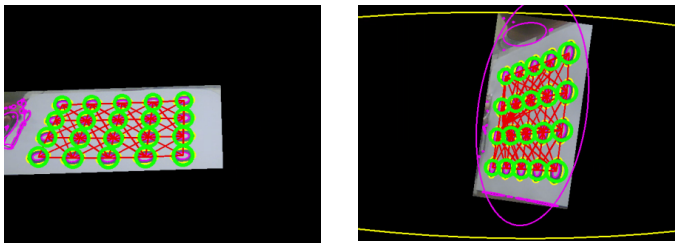

Fig. 8. Good patter points recognition.

Fig. 9. Ellipse not detected correctly cause by a poor image definition.

we perform the tracking process described in the next subsection.

- Using each pair of points we make a line and determine the line equation.
- Using the line equation count the number of points near to the line and store these points in a aux vector.
- If there are 5 points in the aux vector, order these points using the x coordinate and check the distance between the first and last elements if the distance is lower than the radio of circles then sort the aux vector again using the y coordinate
- Draw a line between the first and last elements of the aux vector.
- Keep the last point got from the process described before.
- Remove the points and continue searching process to find the next line until we found the 4 lines.

Whit the ordered points we only need to draw lines from the 0 point to 4 point, 5 point to 9 point etc.

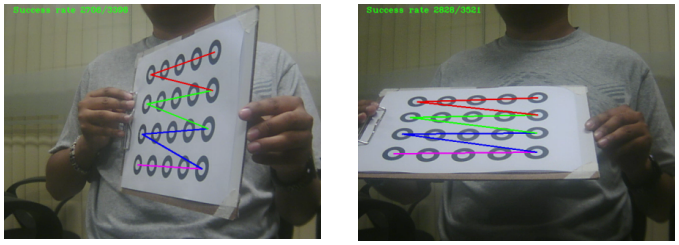

Fig. 10. Results after applying our algorithm on ps3eyecam video.

2.6 Tracking

To keep the order obtained in the previous process we perform a tracking process. From the points found in the previous frame we search in the new points some one that is nearly to these one using the radio as boundary. When the 20 points was found we check if any one has distance bigger than the radio to its near point if this is true we discard old point and consider the pattern was lost.

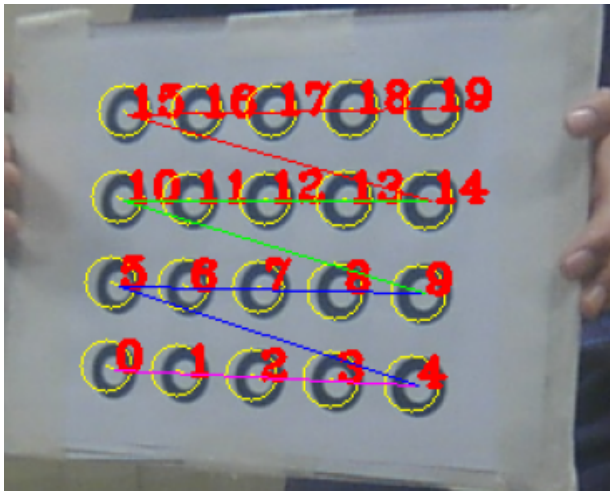

Fig. 11. Tracking process, the yellow circles are the area to search the new position for each pattern point.

3. CAMERA CALIBRATION

To perform the camera calibration we need to select some frames where the pattern was found and use these points to perform the calibration, it's important to choose the set of frames that covers the whole screen to improve the calibration process.

3.1 Frame selection

To perform a good calibration we need to choose some well distributed frames, for this task we develop this algorithm:

1 Define a grid over the image with r x c quadrants.
2 Define the acceptance area for every quadrant, that is a circle with radio the width * height / 6
4 Define the number of frames per quadrant as round ((samples + 4)/(r * c))
5 Per each frame we calculate the center point as (point[7] + point[12]) /2
6 Check the quadrant of the calculated center and if the center stay in the acceptance area.
7 Check if there is possible to add the point in these quadrant.
7.1 Add the frame to the calibration frames and skip 30 frames to avoid add nearly frames.
8 If there are no more space in the quadrant we skip 9 frames to speedup the process.

In the Figure 12 we can see the quadrants used to select frames for the PS3 and Lifecam videos, the blue circles are the acceptance area of each quadrant, green circles are the center of the selected frames and the red points are the centers of the evaluated frames.

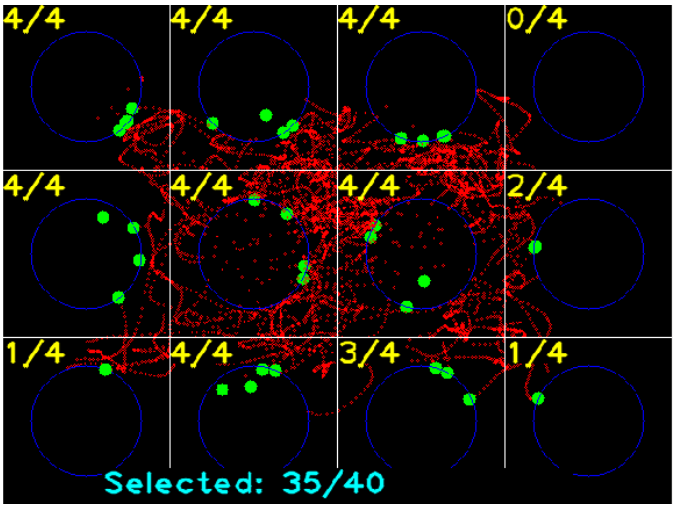

Fig. 12. Visualization of the frame selection using the quadrants distribution.

The Figure 13 show the distribution of points obtained for the algorithm described above.

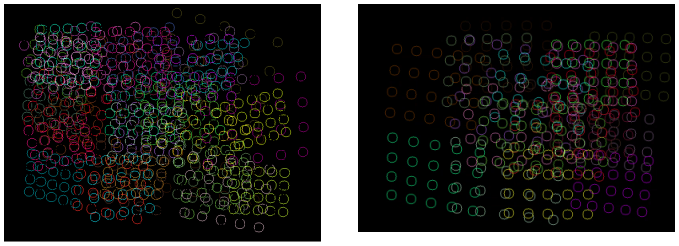

Fig. 13. Points used for the calibration process.

Average of collinearity: In addition to the reprojection error we evaluate the distance from every inner line point to its main line to get the average of collinearity value.

4. REFINEMENT PROCESS

4.1 Undistort

Using the camera matrix and the distortion coefficients we can transform the image input to correct the distortion. In this new image we run the pattern detection algorithm to obtain the pattern points.

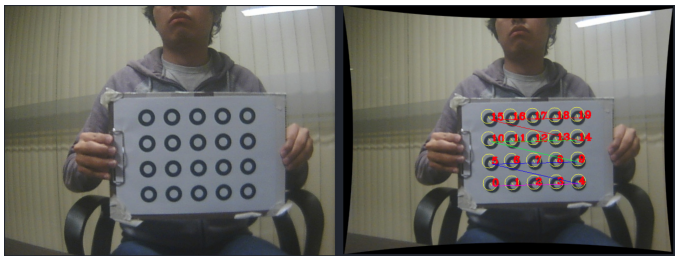

Fig. 14. Image with no distortion using the camera parameters and the distortion coefficients

4.2 Fronto parallel transformation

We use homography transformation from the detected points to an ideal set points to get a fronto parallel view of the image. Then we run the pattern detection algorithm again to find the new centers.

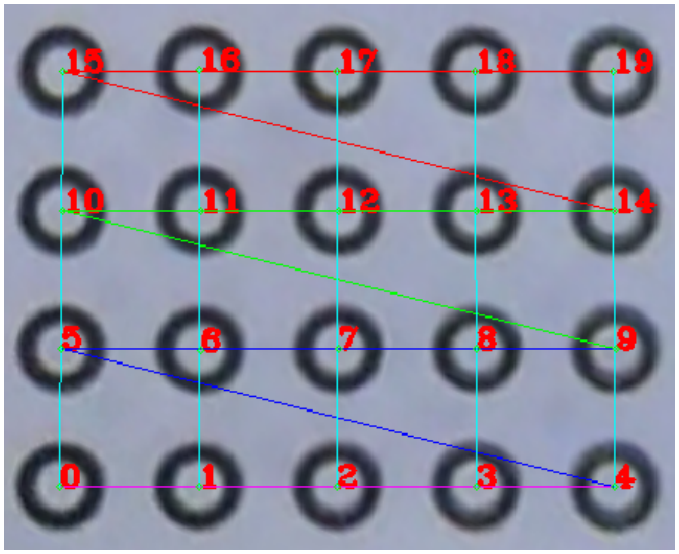

Fig. 15. Image with no distortion using the camera parameters and the distortion coefficients

4.3 Reprojection

Using the new points detected in the fronto parallel image we need to reproject these points to the undistorted image, for this task we use the inverse of the homography matrix and the perspectiveTransform function of opencv.

Points refinement To improve the calibration process and the collinearity we use the undistorted set of points(old points) and the reprojected set of points(new points) the following algorithm describe the process to update the new set of points in the undistorted space.

1 Per each row we fit a line using the 5 points of the new set of points.
2 Per each point in the row
2.1 Calculate the distance from the old point to the line.
2.2 Calculate the distance from the new point to the line.
2.3 Calculate the blend factor as the proportion of each distance to the line giving a priority to the closest one
2.4 Update the new point using the blend factor as newPoint = newPoint * factor + oldPoint * (1-factor)

In the Figure 16 we can see in green the original points and in red the new set of point after the refination process.

Fig. 16. Original points(green) and refinated points(red).

4.4 Distort

To perform the calibration process using the new points we first need to distort the found points. The algorithm to distort the points is shown in the Figure 17 we can see the result of the distort over the found points, in red we can see the original centers, green are the centers with no distortion and blue the points with distortion.

We use the new set of distorted points to perform the camera calibration process.

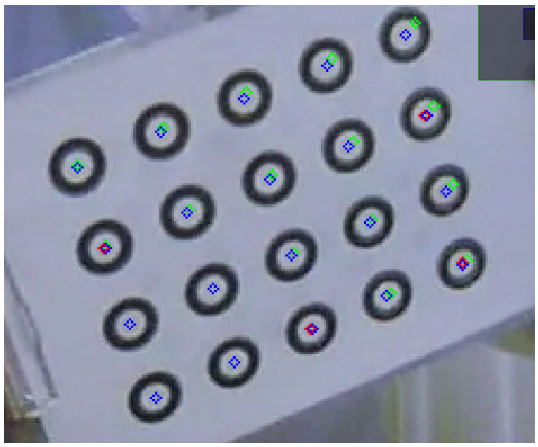

Fig. 17. Result of distort the found points

5. RESULTS

We used the camera calibration algorithm provided by the OpenCV library to perform a comparison with the method implemented in this paper. We disabled the flags FixAspectRatio, AssumeZeroTangentialDistortion and FixPrincipalPointAtTheCenter. See Tables 6.2 and 6.2.

5.1 PS3 calibration

For the PS3 camera we now present the comparison between the chessboard, circles and rings patterns.

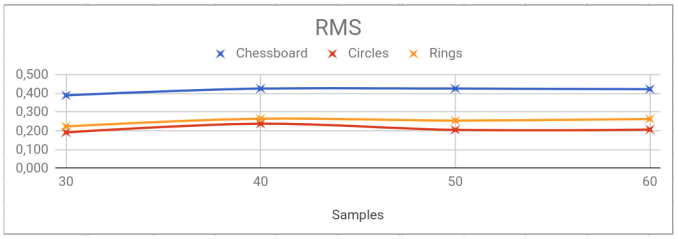

Fig. 18. RMS results using three differents kinds of patterns and differents numbers of samples.

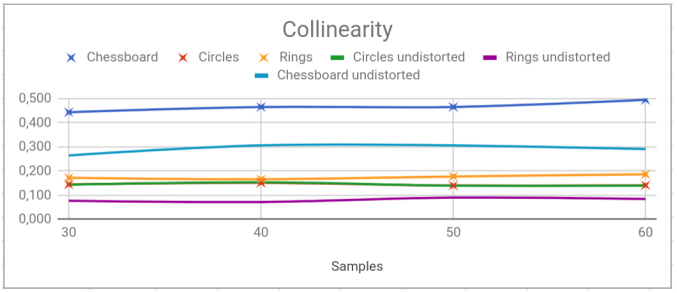

Fig. 19. Collinearity results using three differents kinds of patterns and differents numbers of samples.

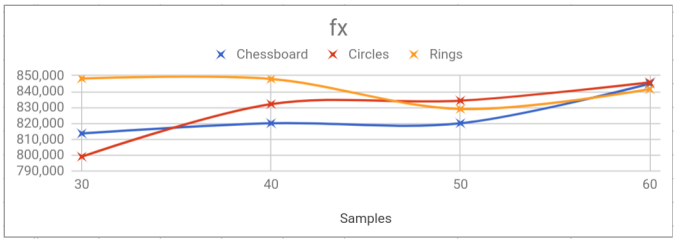

Fig. 20. Focal length X results using three differents kinds of patterns and differents numbers of samples.

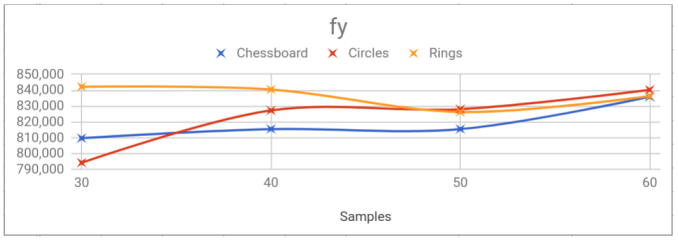

Fig. 21. Focal length Y results using three differents kinds of patterns and differents numbers of samples.

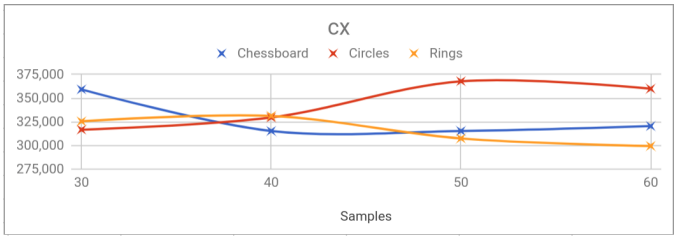

Fig. 22. Optic center X results using three differents kinds of patterns and differents numbers of samples.

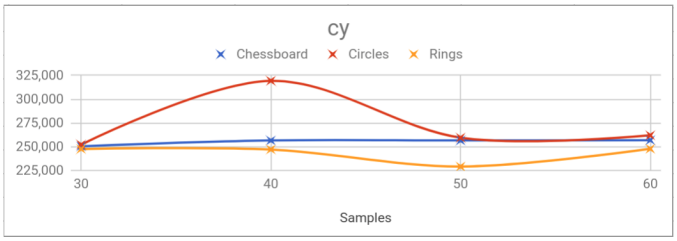

Fig. 23. Optic center Y results using three differents kinds of patterns and differents numbers of samples.

5.2 Lifecam calibration

For the Lifecam camera we now present the comparison between the chessboard, circles and rings patterns.

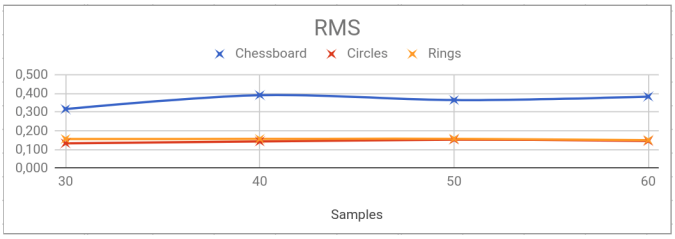

Fig. 24. RMS results using three differents kinds of patterns and differents numbers of samples.

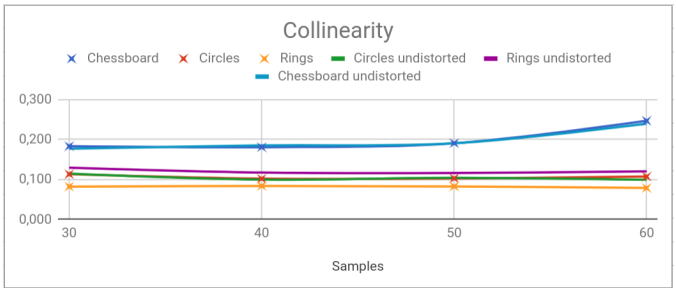

Fig. 25. Collinearity results using three differents kinds of patterns and differents numbers of samples.

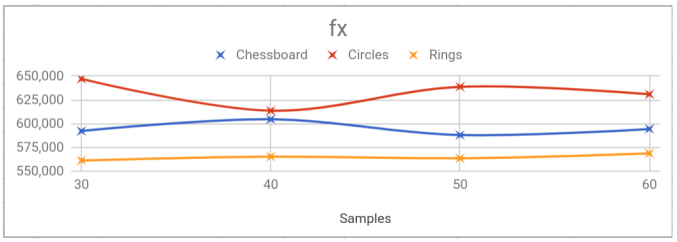

Fig. 26. Focal length X results using three differents kinds of patterns and differents numbers of samples.

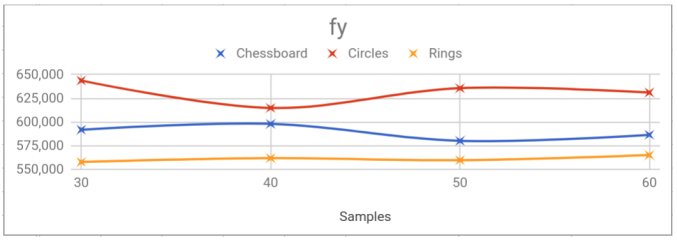

Fig. 27. Focal length Y results using three differents kinds of patterns and differents numbers of samples.

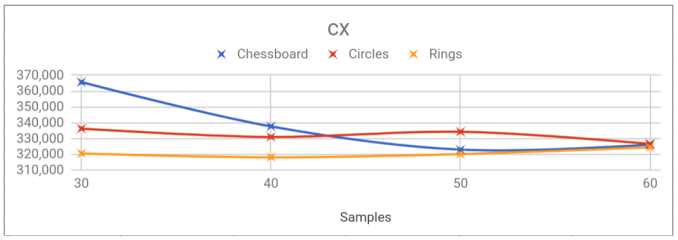

Fig. 28. Optic center X results using three differents kinds of patterns and differents numbers of samples.

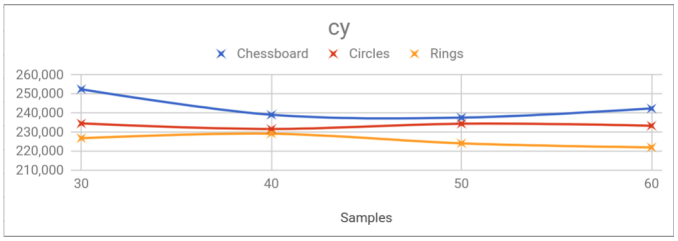

Fig. 29. Optic center Y results using three differents kinds of patterns and differents numbers of samples.

6. EXPERIMENTS

6.1 OpenGL pattern 3d visualization

We use the camera matrix obtained in the calibration process to implement an Opengl application to show the pattern movement in the 3D space.

To achieve this, we use the solvePnP opencv's function to calculate the rotation and translation vectors of a frame using the camera matrix and distortion coefficients found in the calibration process, then we get the euler angles from the rotation vector to rotate a plane with the pattern texture.

In the Figures 30 and 31 we can see the results obtained using this idea.

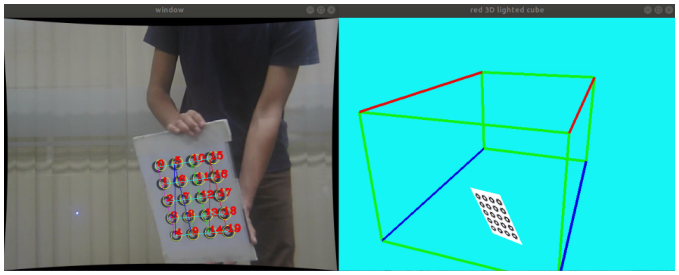

Fig. 30. Experiments using Opencv + Opengl

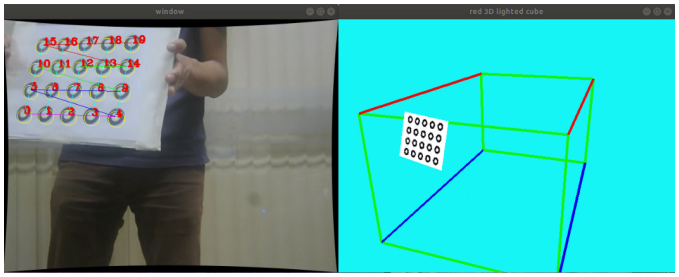

Fig. 31. Experiments using Opencv + Opengl

6.2 Simple augmented reality application

Another experiment that we perform was insert a 3d model in the video using the rotation calculated using the solvePnP Opencv's function.

The Figure 32 shows the execution of our simple augmented reality app.

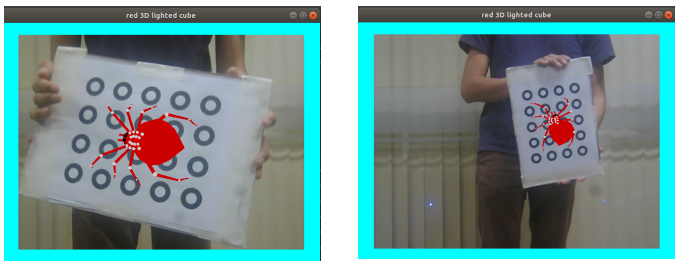

Fig. 32. Simple augmented reality application

7. CONCLUSIONS

- It is important to clean irrelevant information in the videos in order to capture relevant characteristics.
- The algorithm needs to be further improved to get better performance on other videos.
- The algorithm depends on how well the ellipses of the calibration pattern are detect. Depends basically on fitEllipse method of opencv.
- For the camera calibration process is important to perform a good tracking algorithm to keep the order of the points.
- The frames selected for the calibration process should cover the four quadrants of the image plane in similar proportions and have good quality to improve the results.

REFERENCES

Bradley, D. and Roth, G. (2007). Adaptive thresholding using the integral image. *Journal of graphics tools*, 12(2), 13–21.

Datta, A., Kim, J.S., and Kanade, T. (2009). Accurate camera calibration using iterative refinement of control points. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, 1201–1208. IEEE.

Prakash, C.D. and Karam, L.J. (2012). Camera calibration using adaptive segmentation and ellipse fitting for localizing control points. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, 341–344. IEEE.

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11), 1330–1334.

	Chessboard					Circles					Rings				
Samples	**20**	**30**	**40**	**50**	**60**	**20**	**30**	**40**	**50**	**60**	**20**	**30**	**40**	**50**	**60**
Rms	0,390	0,426	0,426	0,423	0,423	0,192	0,239	0,206	0,207	0,226	0,224	0,265	0,255	0,264	0,260
fx	813,725	820,161	820,161	845,004	845,004	799,140	832,183	834,295	845,796	838,066	848,202	847,880	829,118	841,530	832,514
fy	809,813	815,512	815,512	835,884	835,884	794,294	827,273	828,058	840,258	835,511	842,190	840,456	826,297	836,341	827,218
cx	359,364	315,476	315,476	320,676	320,676	316,854	329,727	367,877	360,050	360,822	325,649	331,315	307,625	299,637	303,603
cy	250,695	256,814	256,814	257,091	257,091	252,407	319,372	259,705	262,200	276,342	247,831	247,318	229,413	248,058	250,308
Collinearity	0,443	0,465	0,465	0,495	0,495	0,145	0,152	0,140	0,141	0,145	0,173	0,167	0,178	0,187	0,198
Collinearity undistorted	0,265	0,306	0,306	0,291	0,291	0,144	0,154	0,140	0,140	0,145	0,077	0,072	0,090	0,085	0,082

Fig. 33. PS3 camera calibration results using three differents kinds of patterns and differents numbers of samples.

	Chessboard					Circles					Rings				
Samples	**20**	**30**	**40**	**50**	**60**	**20**	**30**	**40**	**50**	**60**	**20**	**30**	**40**	**50**	**60**
Rms	0,317	0,392	0,365	0,384	0,447	0,133	0,143	0,153	0,146	0,146	0,156	0,157	0,157	0,150	0,149
fx	592,424	604,732	588,222	594,538	577,396	647,129	613,707	638,809	630,977	644,893	561,505	565,586	563,842	568,962	568,983
fy	591,990	598,184	580,326	586,648	572,452	643,747	614,923	635,728	631,196	644,245	558,071	562,136	560,000	565,347	565,909
cx	365,692	337,721	323,025	326,155	333,391	336,306	331,074	334,358	326,686	326,254	320,645	318,235	320,239	324,489	323,410
cy	252,566	239,212	237,711	242,498	240,347	234,667	231,740	234,522	233,461	232,634	226,959	229,333	224,278	222,154	217,304
Collinearity	0,183	0,181	0,191	0,247	0,272	0,113	0,102	0,103	0,107	0,106	0,082	0,084	0,083	0,079	0,080
Collinearity undistorted	0,177	0,185	0,191	0,239	0,256	0,115	0,100	0,104	0,099	0,111	0,130	0,117	0,116	0,120	0,112

Fig. 34. Lifecam camera calibration results using three differents kinds of patterns and differents numbers of samples.