Camera calibration using OpenCV

Wilbert Pumacay, *Catholic University San Pablo*, wilbert.pumacay@ucsp.edu.pe
Gerson Vizcarra, *Catholic University San Pablo*, gerson.vizcarra@ucsp.edu.pe

***Abstract*—Camera calibration is an important step in several applications, like augmented reality. To do calibration properly using current calibration methods we need to get features we can related in both 2D camera space and 3D world space to estimate the camera parameters that give this mapping. In this context, the use of grid patterns help by giving us the features we need, being the components in the pattern which we must detect in every frame in video.**

In this paper we use Zhang [2] camera calibration technique implemented by OpenCV, and compare results using feature extraction of chessboard corners, symmetric/asymmetric circle grids, and concentric circle grid patterns, the first two, using funnctions implemented in OpenCV and the third one using our own algorithm by implementing a pipeline that gets these features by combining various techniques from classical image processing.

***Index Terms*—Camera calibration, calibration pattern, circle grid, image processing.**

I. Introduction

THE camera calibration problem consists of finding 11 parameters that describe the mapping between 2D camera space and 3D world space. Six parameters, called extrinsic, come from an homogeneous transform, giving 6 parameters (rotation and translation around the axes). The other 5 parameters, called intrinsic, define some internal properties of the camera. This can be expressed in the following transformation equation:

$$\begin{bmatrix} \mu \\ \nu \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & \mu_0 \\ 0 & \beta & \nu_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{x_1} & r_{y_1} & r_{z_1} & t_x \\ r_{x_2} & r_{y_2} & r_{z_2} & t_y \\ r_{x_3} & r_{y_3} & r_{z_3} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (1)$$

In this equation we can detail each of the variables: x, y, z are original coordinates of an object, r_{ij} and t_{ij} means the rotation and translation values respectively in the model matrix, α and β represents the focal length in x and y axis respectively, μ_0 and ν_0 are the coordinates x and y at the optical center.

To find these parameters, camera calibration methods make use of correspondences between 2D and 3D spaces in order to fit the parameters that best describe this mapping. We achieve this by minimizing the following function:

$$\sum_{i}^{m} \sum_{j}^{n} \| T P_{3D}^{ij} - P_{2D}^{ij} \|^2 \quad (2)$$

Where we are trying to minimize the difference between the expected projection and the actual projection over some set of features. The key idea is that supplying sufficient features that have a correct mapping, we can get the 11 parameters needed that minimize this function. So, a key part is the detection of these features.

In equation 2 we are looping through a set of features n that are found in each frame of a video of m frames, so, we basically need to detect some feature points in each frame of video, which is what we focus in this paper.

II. About the method

To compare results of feature extraction, we implemented the following pipeline.

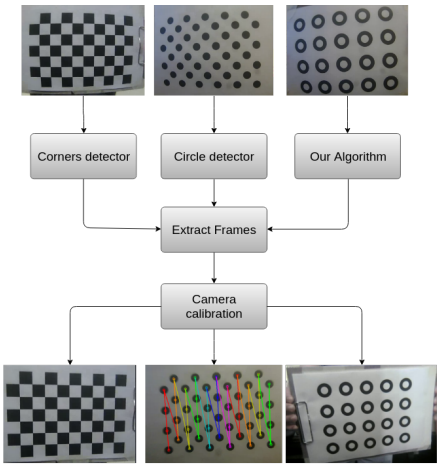

Fig. 1. Comparison pipeline.

A. Pattern detection

We used three principal patterns to calibrate the camera with OpenCV calibration: Chessboard pattern, Circle symmetric/asymmetric pattern and circle concentric pattern. In the first two patterns we applied OpenCV implemented functions, in chessboard pattern we have to recognize corners and save their position, in circle pattern we have to detect circles and save the position of their centers.

B. Circle concentric pattern detection

The pipeline of our pattern detection and tracking algorithm works as follows:

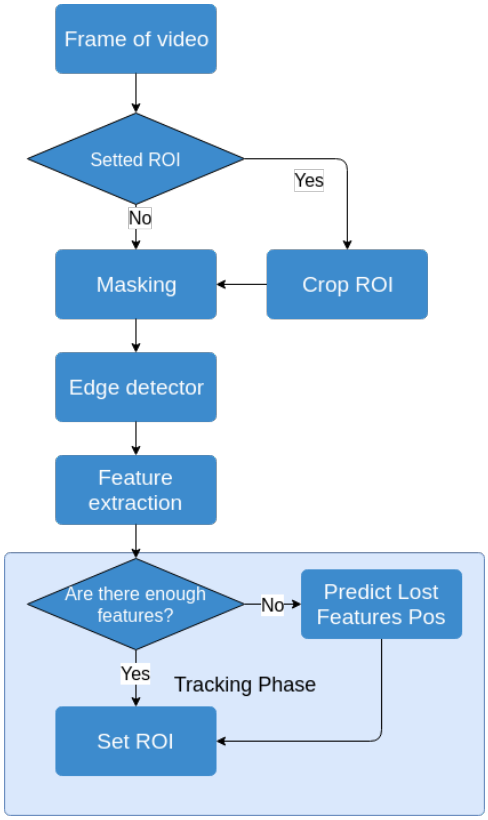

Fig. 2. Pipeline of previous work.

1) Masking : This step is in charge of isolating the pattern by using threhsolding operations.

The approach consists on creating a mask from the grayscale transformed image applying **Adaptive Thresholding algorithm**, this algorithm relies on Integral Image technique especified in [3].

Algorithm 1 Masking

1: *Set up thresholding parameters*
2: $mask = rgb2gray(inputImage)$
3: $mask = AdaptiveThreshold(mask, blockSize)$
4: **return** $masked$

Fig. 3. Image mask applied to ROI.

2) Edge detection: In this stage of the pipeline we extract edges from the result of the previous stage. In order to do this, we applied **Scharr operators** on x and y axis (as described in algorithm 2); Scharr is the result from Sobel algorithm minimizing weighted mean squared angular error in Fourier domain.

Algorithm 2 Edge detection

1: $axis_x = Scharr(masked, 1, 0)$
2: $axis_x = Abs(axis_x)$
3: $axis_y = Scharr(masked, 0, 1)$
4: $axis_y = Abs(axis_y)$
5: $edgesImage = Add(axis_x, axis_y)$
6: **return** $edgesImage$

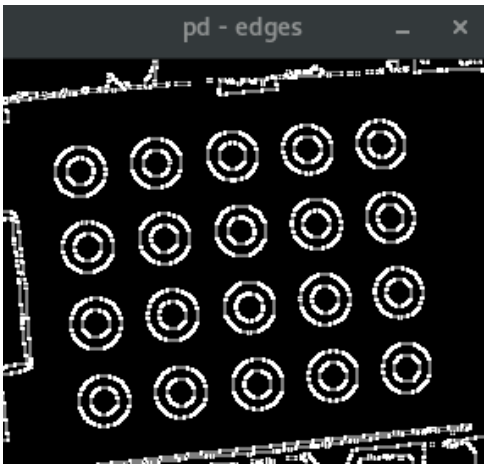

Fig. 4. Edge detection to mask.

3) Feature extraction: The last stage consist of extracting the features needed for the calibration from the edges detected in the previous stage. We used OpenCV's **SimpleBlobDetector** that applies an extra thresholding on the image, applies the **findContours** algorithm to calculate the blob centers, groups centers of several images by their coordinates in blobs and finally, estimates the final centers of the blobs. For detecting

only pattern blobs, we had to apply similar heuristics to above (color blobs, area, aspect ratio, and convexity of points).

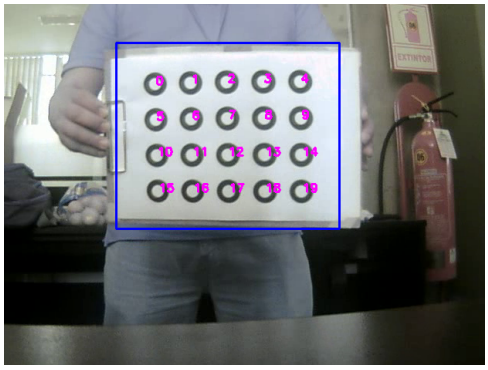

Fig. 5. Result after feature detection.

C. Pattern Matching and Ordering

Once we have the features extracted, we have to order them for them to be used in the calibration algorithm. To achieve this, we made a simple algorithm that matches the best candidate grid to the current features, and correct by the orientation in order to choose a correct ordering.

We first create a bounding box of the current possible corners. Then, we take the outer most corners as the corners of the pattern. With this information, we can compute a candidate grid that represents the positions where the candidate points should be, as seen in figure 7.

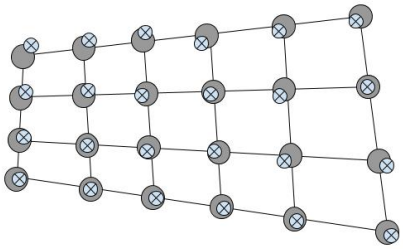

Fig. 6. Grid pattern matching.

We check for each combination of outer most corners, which ordering gives the best fit. This will give us a correct ordering of the outer most corners of the pattern.

Lastly, we check that the orientation of the board is in some range, as shown in figure 8, which will complete the corner ordering, as shown in figure 9.

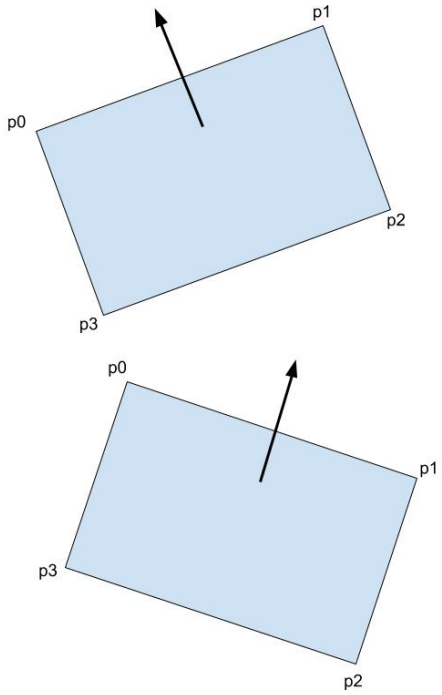

Fig. 7. Pattern orientation check.

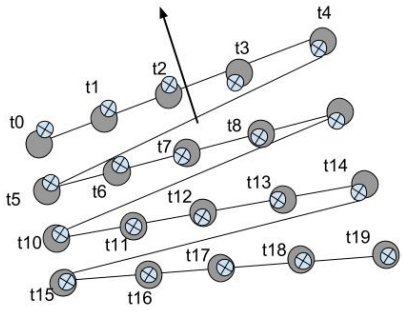

Fig. 8. Pattern final ordering after matching.

D. Calibration process

To do camera calibration process, we took some frames using tools to show the distribution of points in frames in order to improve results: better calibration requires better distribution of points.

We used 30 calibration frames trying to distribute points location as max as possible. In figure 9 we show screenshots of heatmap and point distribution of a calibration process.

Fig. 9. Left: Heatmap. Right: Points distribution.

Also we created a screen that shows the angle histogram of taken frames, like previous tools, it helps us to distribute angles better.

Fig. 10. Angle distribution histogram screen.

III. Results

We calibrated two cameras using as feature detection the three detection algorithms described earlier (OpenCV's chessboard and circle finder and our algorithm)

For each calibration, we took a fixed batch of 30 correctly processed images. The results are the following:

Parameter	PS3 Eye cam			MS lifecam		
	Chessboard	Circles	Rings	Chessboard	Circles	Rings
fx	876.2	933.2	829.5	609.6	609.5	627.0
fy	876.6	935.4	834.8	609.4	612.6	631.8
vx	325.0	342.3	320.0	326.9	351.5	342.7
vy	251.4	220.7	243.5	223.8	227.2	231.2
RMS error	0.313	0.184	0.187	0.74	0.22	0.19
Colinearity before	0.152	0.051	0.17	0.546	0.068	0.05
Colinearity after	0.127	0.054	0.152	0.166	0.011	0.005

TABLE I
PS3 Eye Cam and MS Lifecam calibration results

IV. Conclusions and Future improvements

We see by the results that our algorithm is close enough compared to OpenCV's pattern detectors. We got close to the camera matrix parameters obtained by the other detector algorithms.

Also, the matching and ordering algorithm allows us to get the right ordering, allowing us to pick the right calibration frames to be used for the calibration process.

This process is still quite cumbersome because we have to pick each frame that we want to use. We plan on making this process automatic, by keeping frames at some fixed rate and checking that we pick the right combination of frames from a big batch that give us a good overall distribution over the whole field of view. We have made some tools to help us with this task, as shown in Fig. 9.

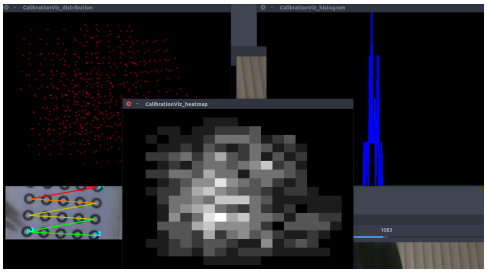

Fig. 11. Pattern final ordering after matching.

We have a heat-map that give us a distribution visualizer, as well as a histogram to be sure we are picking the patterns at some range of orientations.

References

[1] Bradski, G.
OpenCV library. - 2000

[2] Zhengyou Zhang
A Flexible New Technique for Camera Calibration. - 2000

[3] Derek Bradley, Gerhard Roth
Adaptive Thresholding Using the Integral Image. - 2011