Camera Calibration with OpenCV

Zinadine Zidan Alsyahana[#1], Jeffrey Nobel Martin[#2], Affa Ndaru Rabbany Wijaya[#3]

[#]*Robotics and AI Engineering*
Faculty of Advanced Technology and Multidiscipline
Airlangga University
Jl. Dr. Ir. H. Soekarno, Mulyorejo, Kec. Mulyorejo, Kota SBY, Jawa Timur 60115
[1]zinadine.zidan.alsyahana-2022@ftmm.unair.ac.id
[2]jeffrey.nobel.martin-2022@ftmm.unair.ac.id
[3]affa.ndaru.rabbany-2022@ftmm.unair.ac.id

***Abstract*— Camera calibration plays a vital role in enhancing the precision of computer vision systems, especially in industrial applications such as quality control, automation, and robotic vision. This paper presents a methodical approach to camera calibration using a ChArUco board, which combines the advantages of checkerboard patterns and ArUco markers to overcome issues such as obstructions and unclear orientation. Using a HuatengVision industrial camera equipped with a 45 mm focal length lens, we captured multiple images of the ChArUco board and processed them to calculate intrinsic camera parameters, including the camera matrix and distortion coefficients. The results of our calibration show a calculated focal length of 49.26 mm on the x-axis and 46.65 mm on the y-axis, closely aligning with the camera's physical settings. The calibration process successfully flattened the convex distortion, as evidenced by the undistorted images. This work demonstrates the effectiveness of the ChArUco board in achieving accurate camera calibration, providing a reliable solution for industrial applications.**

***Keywords*— Camera calibration, ChArUco board, camera matrix, image undistortion, OpenCV.**

I. Introduction

The application of computer vision in the industrial world is very broad, as we often encounter the use of computer vision including quality control, industrial automation, as well as functioning as vision on robots, therefore the accuracy of camera calibration is crucial [1]. Conventional calibration techniques using checkerboard patterns are often hampered by problems such as obstructions and unclear orientation [2]. To improve the accuracy of camera calibration, researchers have combined the advantages of checkerboard patterns and ArUco markers to create the ChArUco board - a system that can overcome these problems [3].

With a 45 mm focal length lens and a HuatengVision industrial camera with a ChArUco board as the object, the present study takes a methodical approach to camera calibration [4]. Through an iterative refinement procedure, in order to obtain high-precision in-camera parameters by taking multiple photos of the ChArUco board at a set distance of 3 meters [5].

The objective of this work is significant, as it has the ability to improve the precision of vision-based measurements and offers a reliable approach to industrial camera calibration. By carefully verifying the focal length parameters and implementing extensive distortion correction, this study aims to validate previous research.

II. Literature Review

A. Camera Models

Camera models help describe how a 3D world is projected onto a 2D image [7-9]. Understanding these models is fundamental in computer vision, photogrammetry, and robotics applications [10-12]. Here, we will discuss two important aspects of camera modeling: the Pin-Hole Camera Model and Distortion Models.

The Pin-Hole Camera Model is the simplest mathematical model used to describe the projection of a 3D scene onto a 2D image plane [13]. It assumes that light from a scene passes through a single small aperture (the pin-hole) and is projected onto an image plane. In the pin-hole model, rays of light pass through the pin-hole and form an inverted image of the scene on the image plane, which is usually behind the aperture. If you imagine an object point in 3D space (X, Y, Z), it will project to a 2D point (x, y) on the image plane. The relationship between the 3D coordinates (X, Y, Z) of a point in space and the 2D coordinates (x, y) of its projection on the image plane is given by:

$$x = f \cdot \frac{X}{Z},\ y = f \cdot \frac{Y}{Z}$$

where:

- f is the focal length (distance between the pin-hole and the image plane).
- (X, Y, Z) are the coordinates of the point in 3D space.
- (x, y) are the coordinates on the 2D image plane.

The projection can also be represented by the camera matrix, K, which encodes intrinsic camera parameters such as focal length, and the principal point (the center of the image). The matrix form of the pin-hole model is:

$$s \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where:

- s is a scaling factor.
- K is the intrinsic camera matrix containing the focal length and principal point

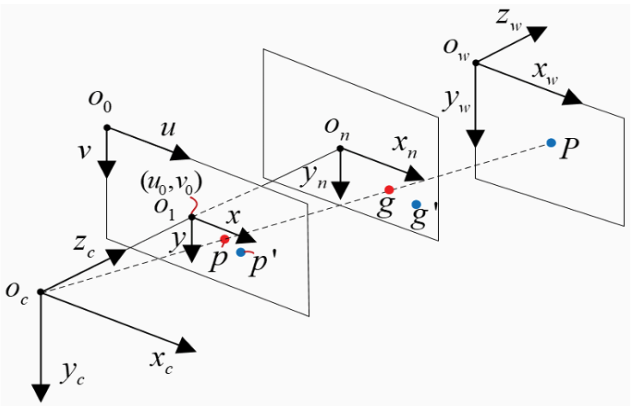

Fig. 1 Camera model [6]

In real-world cameras, the pin-hole model is an idealization. Actual lenses introduce distortion to the captured image, which needs to be corrected for accurate 3D reconstruction or image processing. Two common types of distortion are radial distortion and tangential distortion [14].

The reason behind radial distortion is that light rays are bent more at the edges of the lens than at the center, giving the appearance of curved straight lines. Radial distortion is classified into two categories. Straight lines appear to bend outward due to barrel distortion, which causes the image to appear to bulge outward from the center. A pincushion distortion is when lines in the image bend straight toward the center, giving the impression that the image is pinching inward [15]. Tangential distortion occurs when the lens and the image sensor are not perfectly aligned, causing some parts of the image to appear shifted [16].

According to the research [17], the amount of radial distortion and tangential distortion can be described mathematically by:

$$x_{\text{distorted}} = x + x\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots\right)$$
$$y_{\text{distorted}} = y + \underbrace{y\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots\right)}_{\text{Radial Distortion}}$$

$$x_{\text{distorted}} = x + \left[p_2\left(r^2 + 2\bar{y}^2\right) + 2p_1\bar{x}\bar{y}\right]\left(1 + p_3 r^2 + \ldots\right)$$
$$y_{\text{distorted}} = y + \underbrace{\left[p_1\left(r^2 + 2\bar{x}^2\right) + 2p_2\bar{x}\bar{y}\right]\left(1 + p_3 r^2 + \ldots\right)}_{\text{Tangential Distortion}}$$

where:

- $x_{\text{distorted}}$ and $y_{\text{distorted}}$ are the distorted coordinates.
- x and y are the undistorted coordinates.
- $r = \sqrt{x^2 + y^2}$ is the distance from the center of the image.
- k_1, k_2, k_3 are the radial distortion coefficients.
- p_1, p_2, p_3 are the tangential distortion coefficients.

B. Calibration Parameters

Camera calibration is a critical process in computer vision and robotics for determining the mapping between 3D points in the real world and their 2D projections on an image. This process requires the determination of both intrinsic and extrinsic parameters [18].

Intrinsic parameters describe the internal characteristics of the camera, which govern how the camera transforms 3D points in the world into 2D image coordinates. These parameters are specific to the camera's optical and sensor setup. Intrinsic parameters include key factors such as the focal length, principal points, and distortion coefficients, which collectively define the camera's internal optical and sensor properties [19].

The focal length (f) defines how strongly the lens converges or diverges light. It affects the scale of the image and is usually expressed in pixel units. The focal length is a key part of the camera's intrinsic matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where:

- f_x and f_y represent the focal length in terms of pixels along the x and y axes.
- The focal length is responsible for controlling the field of view (FOV). A larger focal length results in a narrower FOV, while a shorter focal length results in a wider FOV.

The principal point (c_x, c_y) is the point on the image plane where the optical axis intersects the image. It is often, but not always, located at the center of the image.Deviations of the principal point from the center of the image can occur due to imperfections during the camera manufacturing process. The principal point is also included in the camera matrix K, as shown above, and helps translate from image coordinates to real-world coordinates.

Real-world cameras introduce distortions due to the lens, particularly radial and tangential distortions. Distortion coefficients are used to correct these distortions.The radial distortion is commonly modeled with coefficients k_1, k_2, k_3 (for barrel or pincushion distortions), while tangential distortions are modeled using p_1, p_2, p_3, … (for misalignment between the lens and the image sensor). These coefficients are essential for correcting the raw images to obtain accurate geometric representations.

Extrinsic parameters describe the transformation between the camera's reference frame and the 3D world reference frame, which determines the camera's position and orientation relative to the scene. These parameters are crucial in understanding how a camera is positioned in space and how it "sees" the 3D world. They consist of the rotation matrix and the translation vector [20].

The rotation matrix R is a 3x3 matrix that defines the orientation of the camera relative to the world coordinates. It encodes how the camera is rotated around the x, y, and z axes in space. By using this matrix, it is possible to transform 3D world coordinates into the camera's coordinate system. The transformation can be represented as:

$$\mathbf{P}_{\text{camera}} = R \cdot \mathbf{P}_{\text{world}}$$

where $\mathbf{P}_{\text{world}}$ is a 3D point in the world, and $\mathbf{P}_{\text{camera}}$ is the corresponding point in the camera's reference frame. The rotation matrix ensures that the orientation of objects in the scene is correctly aligned with how the camera perceives them.

In addition to rotation, the camera's position in the world is defined by the translation vector T, a 3x1 vector that specifies the camera's displacement along the x, y, and z axes relative to the origin of the world coordinate system. This vector allows the camera's coordinate system to shift in 3D space, representing its actual physical location. The complete transformation from world coordinates to camera coordinates is given by:

$$\mathbf{P}_{\text{camera}} = R \cdot \mathbf{P}_{\text{world}} + T$$

The combination of the rotation matrix and translation vector provides a comprehensive mapping of 3D points from the world coordinate system to the camera's reference frame, making it possible to accurately reconstruct or analyze a scene based on camera data.

C. Calibration Techniques

Camera calibration is the process of estimating the intrinsic and extrinsic parameters of a camera to ensure accurate mapping of 3D points from the world to 2D image coordinates. Various calibration techniques are employed, with the most common patterns being chessboards, ArUco markers, and ChArUco boards. Each method has its own strengths and use cases, influencing their application in computer vision tasks.

The chessboard pattern method is one of the most widely used techniques for camera calibration. It involves capturing multiple images of a flat chessboard pattern from different angles and positions. The chessboard provides a well-defined grid of black and white squares, which allows for easy detection of intersection points or corners. These corners serve as reference points for the calibration process [25].

By analyzing the position of these corners in multiple images, it becomes possible to compute both the intrinsic parameters (like focal length and distortion coefficients) and extrinsic parameters (such as the camera's position and orientation in space). The consistency and simplicity of the chessboard grid make it a reliable method, as the corners can be accurately located even in low-quality images. This method is favored in applications that require high precision, such as 3D reconstruction or robotic vision.

The ArUco marker method utilizes fiducial markers that consist of a unique black-and-white square pattern. These markers are easy to detect and can be recognized even when partially obscured. The detection process involves identifying these markers in the captured images and using their known positions to estimate camera parameters [26].

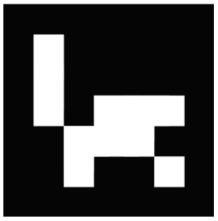

Fig. 2 ArUco marker [26]

ArUco markers are advantageous in scenarios where lighting conditions are variable or when quick detection is necessary. They provide a robust solution for calibration and pose estimation, enabling accurate tracking of the camera's position in real-time applications. By leveraging the unique identifiers of each marker, this method enhances reliability, especially in dynamic environments.

The ChArUco method combines the strengths of both chessboards and ArUco markers. A ChArUco board features both ArUco markers and chessboard corners, allowing for reliable detection and high accuracy in calibration. The method works by first detecting the ArUco markers, which then aids in the interpolation of Charuco corners [27].

Fig. 3 A ChArUco board combines a chessboard and ArUco markers [27]

By using both types of features, ChArUco boards achieve higher precision, even in conditions where parts of the board may be obstructed. This method is particularly useful in robotics and augmented reality applications, where accurate pose estimation is crucial.

D. Implementation in OpenCV

OpenCV [24] is a powerful open-source computer vision library that provides a comprehensive suite of functions for camera calibration. These functions allow users to easily implement camera calibration techniques such as the chessboard pattern method, compute intrinsic and extrinsic parameters, and correct image distortions. OpenCV's well-documented API and accessible tools make it the go-to library for camera calibration in research and practical applications.

Key components highlighted include:

- **cv2**: OpenCV library used for computer vision tasks. numpy: For handling arrays and numerical computations.
- **os**: To interact with the operating system for file handling
- **calibrate_and_save_parameters(image_folder)**: Loads images, detects ArUco markers, estimates Charuco corners, and saves camera calibration parameters (camera matrix, distortion coefficients, rvecs, tvecs) to .npy files.
- **calculate_reprojection_error(camera_matrix, dist_coeffs, rvecs, tvecs, object_points, image_points)**: Calculates the reprojection error by comparing detected Charuco corners with projected object points to evaluate calibration accuracy.
- **detect_pose(image, camera_matrix, dist_coeffs)**: Detects ArUco markers in an image, interpolates Charuco corners, and estimates the pose of the board using the saved camera calibration parameters.

For details of the program, please refer to Appendix 1 and Appendix 2.

E. Recent Research and Applications on Camera Calibration in Robotics

Camera calibration continues to play a vital role in various robotic applications, from precision-based industrial manipulations to advanced medical robotics. Recent studies showcase innovative methods that improve the automation, accuracy, and robustness of camera calibration techniques in real-world applications. In this section, we will review several recent research papers and their contributions to the field.

An important advancement in multi-camera calibration is presented in a study on automated calibration for a

multi-camera-robot system [21]. This research focuses on a vision-based control system using four fixed cameras in an eye-to-hand configuration with a collaborative robot. The proposed approach automates the intrinsic camera calibration process by using the robot to capture calibration pattern images from various viewpoints. For extrinsic calibration, the system relies on only two small markers permanently attached to the robot's base, allowing for rapid online calibration. This system enables efficient and precise calibration without manual intervention, making it suitable for laboratory environments. The results from initial experiments demonstrate the method's high level of accuracy and effectiveness in maintaining calibration under controlled conditions. This approach reduces calibration time and makes the process more adaptive to changing environments, which is crucial for real-time robotic operations.

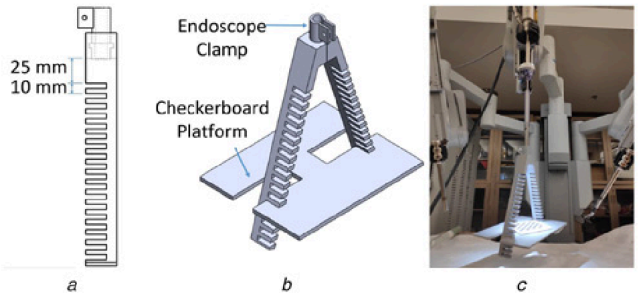

Fig. 4 Apparatus used for preparation of multiple-depth camera calibrationmatrix [21]

A significant application of camera calibration is found in medical robotics, especially in robotic surgery systems like the Da Vinci Surgical System. In recent research, a novel calibration method was introduced to enhance hand-eye coordination between the free-moving endoscopic camera and the patient-side manipulator arms (PSMs) [22]. This process involves image processing and optical tracking to compute the transformation between the camera's frame and an optical marker affixed to the camera body. Similarly, the manipulator arms are calibrated using optical markers to compute the spatial relationship between the PSM and the endoscopic camera. The accuracy of this calibration method was validated by guiding the robot to target points identified through the camera, achieving superior precision compared to earlier calibration techniques. Additionally, the method supports intraoperative calibration updates, which ensures that the system remains highly accurate throughout surgical procedures, an essential feature in medical environments where precision is critical for patient outcomes.

In industrial robotics, where precision and repeatability are paramount, maintaining a high level of accuracy in robotic manipulators is critical [23]. A novel calibration framework was proposed in a study using vision-based techniques with a single camera and ArUco markers. Unlike traditional calibration methods that rely on expensive laser trackers or stereo cameras, this approach leverages a more affordable and flexible camera-based system. The research used a six-degree-of-freedom (6-DOF) UR5 robotic manipulator to test the method, achieving position errors of 2.5 mm and orientation errors of 0.2°. This level of precision demonstrates the framework's applicability to real-world industrial environments where robot accuracy deteriorates over time due to harsh working conditions. The method also uses the product of exponentials for kinematic modeling, which avoids singularities and enhances the system's stability. The result is a practical, low-cost solution for industrial calibration, ensuring that robots maintain their operational accuracy over extended periods.

III. Methods

A. ChArUco Board Setup and Image Capture

The ChArUco board was printed on an A4-sized HVS paper and carefully mounted onto a stable surface, such as a wooden or plastic board, to ensure it remained flat and steady during the image capture process. This setup helps reduce potential distortions caused by bending or movement of the board, which could negatively impact the accuracy of the calibration process. The camera was positioned at a distance of 3 meters from the board and mounted securely to avoid vibrations. A focal length of 45 mm was set on the camera to achieve a clear view, and manual focus adjustments were made to ensure the board's markers were sharp. Uniform lighting across the board was achieved by calibrating brightness settings to prevent shadows or glare that could interfere with detection.

Using the HuaTengVision Platform software, the camera was connected to a laptop, and a live view of the ChArUco board was established. The camera's frame was adjusted to ensure that the entire ChArUco board was fully visible without any clipping or incomplete coverage. Between 10 to 15 images of the board were captured, ensuring that each image was clear, sharp, and complete. Care was taken to ensure that no image had blurring or loss of details, which would compromise the calibration's precision. The goal was to acquire high-quality images that could be reliably used in the next step of the calibration process.

B. Image Processing and Calibration

After the images were captured, they were saved in a folder on the laptop in either JPG or PNG format, ensuring that the images were compatible with the calibration program. This folder served as the source for the calibration software, which required multiple images to improve the accuracy of the camera parameters. The path to this image folder was input into the calibration program, which read the images to begin the calibration process. The program detected the ArUco markers and ChArUco corners within each image, using these points to calculate the intrinsic camera parameters, such as the camera matrix and distortion coefficients (see Appendix 1).

The camera matrix provides essential information about the camera's focal length along the x and y axes, as well as the optical center, while the distortion coefficients account for lens distortions like radial or tangential distortion. Once these parameters were calculated, they were compared against the physical settings of the camera, particularly the focal length of

45 mm, to ensure consistency. This comparison helped verify the accuracy of the calibration and ensured that the computed values aligned with the camera's physical configuration. If any discrepancies were found, adjustments were made to the camera settings, and the process could be repeated.

C. Validation and Image Undistortion

The final step in the calibration process involved validating the results and applying them to real-world images. Once the calibration process was completed and the camera matrix and distortion coefficients were generated, the values were saved for future use. The next step was to load these parameters into the calibration program (see Appendix 2) to apply undistortion to the captured images. This step is crucial as it removes lens distortions, providing a corrected and accurate representation of the scene, free from common distortions like barrel or pincushion effects. The undistorted images were then analyzed to ensure that all visible distortions were effectively corrected.

If the undistorted images showed no issues, the calibration was deemed successful, and the camera parameters were saved for further use in any pose estimation or 3D reconstruction tasks. However, if any residual distortions or inaccuracies remained, the calibration process would be repeated, possibly with additional images or adjustments to the camera settings, to improve the final results. The entire process of capturing images, calculating the camera parameters, and validating the output is critical in ensuring the accuracy of future computer vision tasks, such as pose detection or 3D modeling, that rely on precise camera calibration.

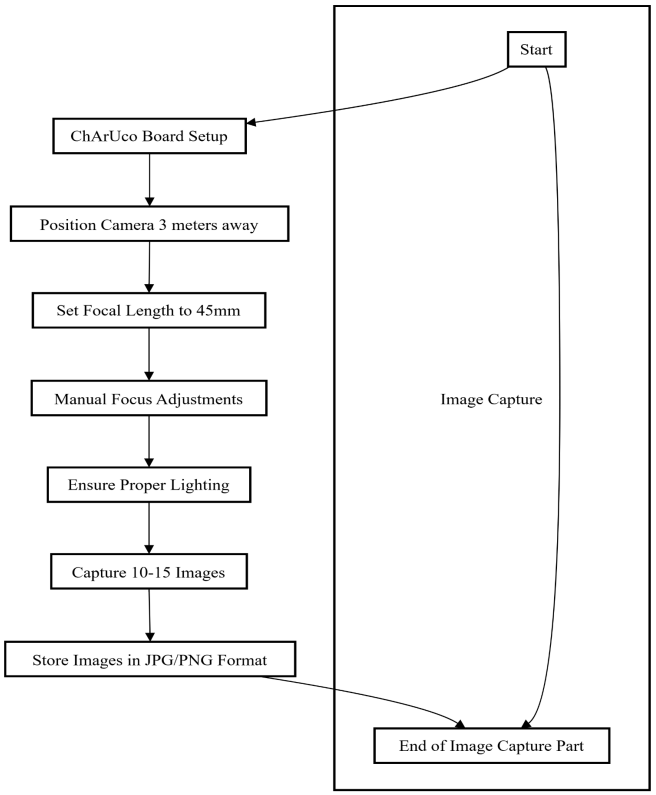

Fig. 5 Capture Process for ChArUco Calibration**:** This flowchart outlines the process of preparing the ChArUco board, setting up the camera, and capturing high-quality images for camera calibration. Each step ensures that the camera and environment are optimized for accurate image acquisition.

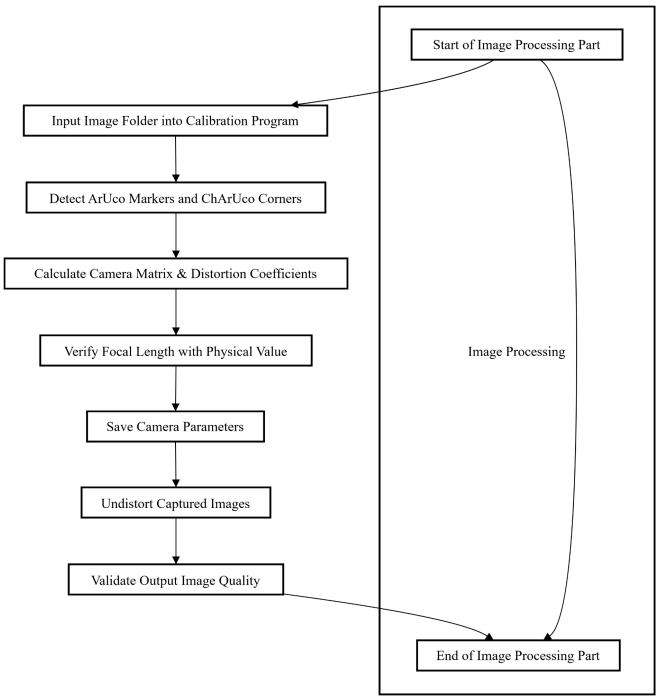

Fig. 6 Image Processing and Calibration Workflow**:** This flowchart illustrates the image processing steps, from reading the captured images to detecting markers, calculating the camera matrix, and undistorting the images. It highlights the calibration process that ensures precise camera parameter estimation.

IV. RESULT

The output of the program in Appendix 1 will produce a camera matrix as shown in the following image.

```
  [[0.18749999 0.15149999 0.        ]
   [0.20849998 0.15149999 0.        ]
   [0.20849998 0.17249998 0.        ]
   [0.18749999 0.17249998 0.        ]]]
MEAN ERROR :  451.9231319910604
5
hasil :  90.38462639821208
camera matrix:  [[9.85126039e+03 0.00000000e+00 9.72199014e+01]
 [0.00000000e+00 9.95298503e+03 5.09758165e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Fig. 7 Output of Program 1

In the matrix, the element [1,1] represents f_x, which is the pixel size along the x-axis. The element [2,2] represents f_y, which is the pixel size along the y-axis.

To determine the focal length of the camera using the camera matrix, the following equation can be used.

$$\text{Focal Length } (x) = \frac{f_x \cdot \text{sensor size}(x)}{\text{resolution}(h)}$$

$$\text{Focal Length } (y) = \frac{f_y \cdot \text{sensor size}(y)}{\text{resolution}(w)}$$

From the equation, the camera sensor specifications are 6400 mm x 4800 mm, with a resolution of 1280 x 1024 px.

Therefore, the results of the calculation using the equation are as follows:

1. Focal Length (x) = 49.26 mm
2. Focal Length (y) = 46.65 mm

From these results, it can be observed that the calculated values are close to the focal length set on the camera, which is 45 mm. This difference could be due to inaccuracies during camera settings or an unclear dataset used.

Next, the distortion coefficients and camera matrix produced by the output of Program 1 are input into Program 2, generating the following output.

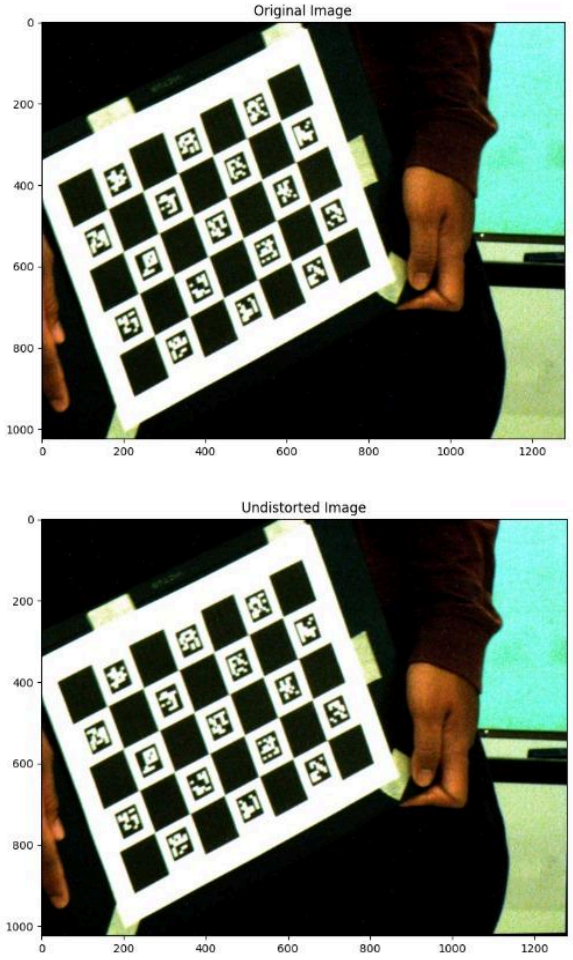

Fig. 8 Output of Program 2

From the program output, there are 2 graphs displaying 2 types of images: the original image and the undistorted image. It can be observed in the undistorted image that there are black pixels in the bottom-right corner of the image, indicating that the image was flattened from the original, which was still convex. This means the calibration was successful.

V. Conclusion

In this camera calibration project, we successfully used the ChArUco board to calibrate an industrial camera with a 45mm lens. Our results show that the calculated focal length is quite close to the results we use as reference - in our experiment, we got results of 49.26 mm for the x-axis and 46.65 mm for the y-axis. While these numbers are not exactly perfect, they are close enough to show that our method works.The results we get do not completely match the reference, this is due to several error factors ranging from the tilt angle at the time of shooting, as well as errors when making settings on the camera.This project is said to be successful because our results show black pixels appearing in the lower right corner of the image, indicating that our calibration successfully flattened the original curved image.

References

[1] 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, doi: https://doi.org/10.1109/34.888718.

[2] J. Heikkila and O. Silven, “A four-step camera calibration procedure with implicit image correction,” *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1106–1112, 1997, doi: https://doi.org/10.1109/cvpr.1997.609468.

[3] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014, doi: https://doi.org/10.1016/j.patcog.2014.01.005.

[4] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, “Speeded up detection of squared fiducial markers,” *Image and Vision Computing*, vol. 76, pp. 38–47, 2018, doi: https://doi.org/10.1016/j.imavis.2018.05.004.

[5] A. Datta, J.-S. Kim, and T. Kanade, “Accurate camera calibration using iterative refinement of control points,” *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, 2009, doi: https://doi.org/10.1109/iccvw.2009.5457474.

[6] P. Sturm and S. J. Maybank, “On plane-based camera calibration: A general algorithm, singularities, applications,” *HAL (Le Centre pour la Communication Scientifique Directe)*, Jan. 2003, doi: https://doi.org/10.1109/cvpr.1999.786974.

[7] G. Ding, T. Chen, L. Sun, and P. Fan, “High Precision Camera Calibration Method Based on Full Camera Model,” *2024 36th Chinese Control and Decision Conference (CCDC)*, vol. 43, pp. 4218–4223, 2024, doi: https://doi.org/10.1109/ccdc62350.2024.10588332.

[8] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 2nd ed. Upper Saddle River: Pearson Education, 2011.

[9] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[10] R. C. Gonzalez and R. E. Woods, *Digital image processing*, 4th ed. New York: Pearson, 2017.

[11] S. Papadopoulos, Ioannis Mademlis, and Ioannis Pitas, “Neural vision-based semantic 3D world modeling,” *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops,* pp. 181–190, 2021, doi: https://doi.org/10.1109/wacvw52041.2021.00024.
[12] D. Cazzato, C. Cimarelli, J. L. Sanchez-Lopez, H. Voos, and M. Leo, “A Survey of Computer Vision Methods for 2D Object Detection from Unmanned Aerial Vehicles,” *Journal of Imaging*, vol. 6, no. 8, p. 78, 2020, doi: https://doi.org/10.3390/jimaging6080078.
[13] P. Sturm, “Pinhole Camera Model,” *Springer eBooks*, pp. 983–986, 2021, doi: https://doi.org/10.1007/978-3-030-63416-2_472.
[14] Y. Liu *et al.*, “Optical distortion correction considering radial and tangential distortion rates defined by optical design,” *Results in Optics*, vol. 3, pp. 100072–100072, 2021, doi: https://doi.org/10.1016/j.rio.2021.100072.
[15] C. Ernould, B. Beausir, J.-J. Fundenberger, V. Taupin, and E. Bouzy, “Measuring elastic strains and orientation gradients by scanning electron microscopy: Conventional and emerging methods,” *Advances in imaging and electron physics*, pp. 1–47, 2022, doi: https://doi.org/10.1016/bs.aiep.2022.07.001.
[16] J. Xu, D.-W. Han, K. Li, J.-J. Li, and Z.-Y. Ma, “A Comprehensive Overview of Fish-Eye Camera Distortion Correction Methods,” *arXiv (Cornell University)*, Jan. 2024, doi: https://doi.org/10.48550/arxiv.2401.00442.
[17] P. Drap and J. Lefèvre, “An Exact Formula for Calculating Inverse Radial Lens Distortions,” *Sensors*, vol. 16, no. 6, p. 807, Jun. 2016, doi: https://doi.org/10.3390/s16060807.
[18] S. Pinto, M. Armando, E. F. Pedrosa, and Santos, “A Camera to LiDAR calibration approach through the optimization of atomic transformations,” *Expert Systems with Applications*, vol. 176, pp. 114894–114894, 2021, doi: https://doi.org/10.1016/j.eswa.2021.114894.
[19] K.-Y. Lin, Y.-H. Tseng, and K.-W. Chiang, “Interpretation and Transformation of Intrinsic Camera Parameters Used in Photogrammetry and Computer Vision,” *Sensors*, vol. 22, no. 24, p. 9602, 2022, doi: https://doi.org/10.3390/s22249602.
[20] S. Verma, J. S. Berrio, S. Worrall, and E. Nebot, “Automatic extrinsic calibration between a camera and a 3D Lidar using 3D point and plane correspondences,” *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, doi: https://doi.org/10.1109/itsc.2019.8917108.
[21] M. Kalia, P. Mathur, N. Navab, and T. Salcudean, “Marker-less Real Time Intra-Operative Camera and Hand-Eye Calibration Procedure for Surgical Augmented Reality,” *Healthcare Technology Letters*, 2019, doi: https://doi.org/10.1049/htl.2019.0094.
[22] S. Lee, S. Shim, H.-G. Ha, H. Lee, and J. Hong, “Simultaneous Optimization of Patient–Image Registration and Hand–Eye Calibration for Accurate Augmented Reality in Surgery,” *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 9, pp. 2669–2682, 2020, doi: https://doi.org/10.1109/tbme.2020.2967802.
[23] H. M. Balanji, A. E. Turgut, and L. T. Tunc, “A novel vision-based calibration framework for industrial robotic manipulators,” *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102248, 2022, doi: https://doi.org/10.1016/j.rcim.2021.102248.
[24] OpenCV, “OpenCV library,” *Opencv.org*, 2019. https://opencv.org/
[25] A. De la Escalera and J. M. Armingol, “Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration,” *Sensors*, vol. 10, no. 3, pp. 2027–2044, 2010, doi: https://doi.org/10.3390/s100302027.
[26] G. Čepon, D. Ocepek, M. Kodrič, M. Demšar, T. Bregar, and M. Boltežar, “Impact-Pose Estimation Using ArUco Markers in Structural Dynamics,” *Experimental Techniques*, 2024, doi: https://doi.org/10.1007/s40799-023-00646-0.
[27] M. C. Roshan, M. Isaksson, and A. Pranata, “A geometric calibration method for thermal cameras using a ChArUco board,” *Infrared Physics & Technology*, vol. 138, p. 105219, 2024, doi: https://doi.org/10.1016/j.infrared.2024.105219.

Appendix

Appendix 1 (ChArUco Board Camera Calibration and Pose Estimation Using OpenCV)

```
import cv2
import numpy as np
import os

ARUCO_DICT = cv2.aruco.DICT_6X6_100
SQUARES_VERTICALLY = 5 # row
SQUARES_HORIZONTALLY = 7 #col
SQUARE_LENGTH = 0.036 # ukuran kotak catur dalam
meter
MARKER_LENGTH = 0.021 # ukuran aruco dalam meter

PATH_TO_YOUR_IMAGES =
"/home/sulthan/Documents/coolyeah/Computer
Vision/jepri/"

all_rvecs = []

all_tvecs = []
all_corners = []
all_ids = []
dictionary =
cv2.aruco.getPredefinedDictionary(ARUCO_DICT)
board =
cv2.aruco.CharucoBoard(size=(SQUARES_HORIZONTALL
Y, SQUARES_VERTICALLY),
squareLength=SQUARE_LENGTH,
```

```
markerLength=MARKER_LENGTH,
dictionary=dictionary)
params = cv2.aruco.DetectorParameters()
params.cornerRefinementMethod =
cv2.aruco.CORNER_REFINE_NONE
print('params: ', params)

def calibrate_and_save_parameters():
    # Define the aruco dictionary and charuco
board

    # all_charuco_corners = []
    # all_charuco_ids = []
    rvecs = []
    tvecs = []
    camera_matrix = None
    dist_coeffs = None
    # board = None
    # Load PNG images from folder
    print([i for i in
os.listdir(PATH_TO_YOUR_IMAGES)])
    # image_files = [PATH_TO_YOUR_IMAGES+"\\"+f
for f in os.listdir(PATH_TO_YOUR_IMAGES) if
f.endswith(".jpeg")]
    image_files =
[os.path.join(PATH_TO_YOUR_IMAGES, f) for f in
os.listdir(PATH_TO_YOUR_IMAGES) if
f.endswith(".jpeg")]

    image_files.sort()  # Ensure files are in
order

    all_charuco_corners = []
    all_charuco_ids = []
    print(image_files)

    for image_file in image_files:
        #print nama file .jpg di image_files
setiap for loop jalan
        print("image_files:", image_files)
        print(f"image_file:{image_file}")
        image = cv2.imread(image_file)
        # alpha = 1
        # beta = 0
        # image =
cv2.convertScaleAbs(image,alpha,beta)
        # print(image)
        image_copy = image.copy()
        [marker_corners, marker_ids, _] =
cv2.aruco.detectMarkers(image,
board.getDictionary(), parameters=params)
        print(f"markercorner:{marker_corners}")
        # If at least one marker is detected
        if marker_ids is not None and
len(marker_ids) >= 4 : # MARK: MARKER ID

cv2.aruco.drawDetectedMarkers(image_copy,
marker_corners, marker_ids)
            retval,charuco_corners, charuco_ids
=
cv2.aruco.interpolateCornersCharuco(marker_corne
rs, marker_ids, image, board)
            if retval:

all_charuco_corners.append(charuco_corners)

all_charuco_ids.append(charuco_ids)
            if charuco_ids is not None and
len(charuco_ids) > 4:

cv2.aruco.drawDetectedCornersCharuco(image_copy,
charuco_corners, charuco_ids, (255, 0, 0))
        cv2.imshow('image', image_copy)
        cv2.waitKey(100)
    cv2.destroyAllWindows()

    print("Corner Count:",
len(all_charuco_corners))
    print("ID Count:", len(all_charuco_ids))

    # Verifikasi bahwa setiap entri memiliki ID
yang sesuai
    for corners, ids in zip(all_charuco_corners,
all_charuco_ids):
        print("Corners:", len(corners), "IDs:",
len(ids))
        assert len(corners) == len(ids), "Jumlah
sudut dan ID tidak cocok."

        retval, camera_matrix, dist_coeffs,
rvecs, tvecs =
cv2.aruco.calibrateCameraCharuco(all_charuco_cor
ners, all_charuco_ids, board, image.shape[:2],
None, None)
        all_rvecs.append(rvecs)
        all_tvecs.append(tvecs)

    # Save rvecs and tvecs

np.save('/home/sulthan/Documents/coolyeah/Comput
er Vision/rvecs.npy', rvecs)

np.save('/home/sulthan/Documents/coolyeah/Comput
er Vision/tvecs.npy', tvecs)

np.save('/home/sulthan/Documents/coolyeah/Comput
er Vision/cam-mat-NYKtest.npy', camera_matrix)
```

```
np.save('/home/sulthan/Documents/coolyeah/Computer Vision/dist-coeffs-NYKtest.npy', dist_coeffs)

    return all_charuco_corners, all_charuco_ids, camera_matrix, dist_coeffs, board

def calculate_reprojection_error(all_charuco_corners, all_charuco_ids, camera_matrix, dist_coeffs, board, all_rvecs, all_tvecs):

    print("charuco")
    mean_error = 0
    print("len: ", len(all_charuco_corners))

    for i in range(len(all_charuco_corners)):
        print('gambar ke-: ', i)
        charuco_points = all_charuco_corners[i]
        charuco_ids = all_charuco_ids[i]
        rvecs = all_rvecs[i]
        tvecs = all_tvecs[i]
        rvecs = np.array(rvecs)
        tvecs = np.array(tvecs)
        status, rvecs, tvecs = cv2.aruco.estimatePoseCharucoBoard(charuco_points, charuco_ids, board, camera_matrix, dist_coeffs, rvecs, tvecs)
        print("estimate pose charuco ","berhasil" if status else "gagal")
        obj_points = np.array(board.getObjPoints(), dtype=np.float32)
        print("obj points: ",obj_points)
        obj_points = obj_points.reshape(-1, 3) # Menyesuaikan dimensi obj_points
        # print("sebelum project points")
        imgpoints, _ = cv2.projectPoints(obj_points, all_rvecs[i][0], all_tvecs[i][0], camera_matrix, dist_coeffs)
        # print("stelah project points")
        # Mengubah all_corners[i] dan imgpoints ke format 2D
        corners_2d = all_charuco_corners[i].reshape(-1, 2)

        imgpoints_2d = imgpoints.squeeze()
        # print("squeeze done")

        # Menyesuaikan ukuran array jika perlu
        min_len = min(len(corners_2d), len(imgpoints_2d))
        corners_2d = corners_2d[:min_len]
        imgpoints_2d = imgpoints_2d[:min_len]

        # Menghitung norma dengan format yang benar
        error = cv2.norm(corners_2d, imgpoints_2d, cv2.NORM_L2) / len(corners_2d)
        mean_error += error
        print("MEAN ERROR : ", mean_error)
        print(len(all_charuco_corners))

    return mean_error/len(all_charuco_corners) if len(all_charuco_corners) > 0 else 0

def detect_pose(image, camera_matrix, dist_coeffs):
    # Undistort the image
    undistorted_image = cv2.undistort(image, camera_matrix, dist_coeffs)

    # Define the aruco dictionary and charuco board
    dictionary = cv2.aruco.getPredefinedDictionary(ARUCO_DICT)
    board = cv2.aruco.CharucoBoard((SQUARES_VERTICALLY, SQUARES_HORIZONTALLY), SQUARE_LENGTH, MARKER_LENGTH, dictionary)
    params = cv2.aruco.DetectorParameters()

    # Detect markers in the undistorted image
    marker_corners, marker_ids, _ = cv2.aruco.detectMarkers(undistorted_image, dictionary, parameters=params)

    # If at least one marker is detected
    if len(marker_ids) > 0:
        # Interpolate ChArUco corners
        charuco_retval, charuco_corners, charuco_ids = cv2.aruco.interpolateCornersCharuco(marker_corners, marker_ids, undistorted_image, board)

        # If enough corners are found, estimate the pose
        if charuco_retval:
            retval, rvec, tvec = cv2.aruco.estimatePoseCharucoBoard(charuco_corners, charuco_ids, board, camera_matrix, dist_coeffs, None, None)

            # If pose estimation is successful, draw the axis
            if retval:

cv2.drawFrameAxes(undistorted_image, camera_matrix, dist_coeffs, rvec, tvec, length=0.1, thickness=15)
```

```
    return undistorted_image

def main():
    # Load calibration data
    # camera_matrix =
np.load['camera_matrix.npy']
    camera_matrix =
np.load('/home/sulthan/Documents/coolyeah/Comput
er Vision/cam-mat-NYKtest.npy')
    # dist_coeffs = np.load['dist_coeffs.npy']
    dist_coeffs =
np.load('/home/sulthan/Documents/coolyeah/Comput
er Vision/dist-coeffs-NYKtest.npy')

    # Iterate through PNG images in the folder
    image_files =
[os.path.join(PATH_TO_YOUR_IMAGES, f) for f in
os.listdir(PATH_TO_YOUR_IMAGES) if
f.endswith(".jpeg")]
    image_files.sort()  # Ensure files are in
order

    output_folder =
"/home/sulthan/Documents/coolyeah/Computer
Vision/output/"
    output_file_path =
os.path.join(output_folder,
os.path.basename(image_file))
    cv2.imwrite(output_file_path, pose_image)


    for image_file in image_files:
        # Load an image
        image = cv2.imread(image_file)

        # Detect pose and draw axis
        pose_image = detect_pose(image,
camera_matrix, dist_coeffs)




        # Show the image
        cv2.imshow('Pose Image', pose_image)
        cv2.waitKey(1000)
        cv2.destroyAllWindows()

if _name_ == "_main_":
    all_charuco_corners, all_charuco_ids,
camera_matrix, dist_coeffs, board =
calibrate_and_save_parameters()
    hasil =
calculate_reprojection_error(all_charuco_corners
, all_charuco_ids, camera_matrix, dist_coeffs,
board, all_rvecs, all_tvecs)
    print("hasil : ",hasil)
    print("camera matrix: ", camera_matrix)

    # img =
cv2.imread(PATH_TO_YOUR_IMAGES+"/cal0.jpg")
    # alpha = 2
    # beta = 0.5
    # img =
cv2.convertScaleAbs(img,alpha=alpha,beta=beta)
    # cv2.imshow("ori", img)
    # undistorted =
cv2.undistort(img,camera_matrix,dist_coeffs)
    # cv2.imshow("undist",undistorted)
    # cv2.waitKey(0)
```

Appendix 2 (Camera Image Undistortion Using re-calibrated Camera Matrix and Distortion Coefficients)

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from matplotlib.pyplot import imread

mtx =
np.load("/home/sulthan/Documents/coolyeah/Comput
er Vision/cam-mat-NYKtest.npy")
dist =
np.load("/home/sulthan/Documents/coolyeah/Comput
er Vision/dist-coeffs-NYKtest.npy")


print(mtx)
print("================================")
print(dist)

# img =
imread("/home/sulthan/Documents/coolyeah/Compute
r Vision/calibration data/WhatsApp Image
2024-09-17 at 06.31.21 (2).jpeg")
# img =
imread("/home/sulthan/Documents/coolyeah/Compute
r Vision/calibration data/WhatsApp Image
2024-09-17 at 06.31.22.jpeg")
img =
imread("/home/sulthan/Documents/coolyeah/Compute
r Vision/jepri/WhatsApp Image 2024-10-03 at
22.59.08 (3).jpeg")

# img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# cv2.imshow("img",img)
# cv2.waitKey(0)
```

```
# cv2.destroyAllWindows()

undistorted = cv2.undistort(img, mtx, dist)
# cv2.imshow("undist",undistorted)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.figure(figsize=(20,10))
```

```
plt.subplot(1,2,1)
plt.imshow(img)
plt.title("Original Image")
plt.subplot(1,2,2)
plt.imshow(undistorted)
plt.title("Undistorted Image")
plt.show()
```