Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

PYCASO: Python module for calibration of cameras by Soloff's method

Eddy Caron [a,*], Jean-François Witz [a], Christophe Cuvier [b], Arnaud Beaurain [a], Vincent Magnier [a], Ahmed El Bartali [a]

[a] Univ. Lille, CNRS, Centrale Lille, UMR 9013 - LaMcube - Laboratoire de Mécanique, Multiphysique, Multiéchelle, 59655 Villeneuve d'Ascq, France
[b] Univ. Lille, CNRS, ONERA, Arts et Métiers Institute of Technology, Centrale Lille, UMR. 9014-LMFL-Laboratoire de Mécanique des Fluides de Lille-Kampé de Fériet, F-59000, Lille, France

ARTICLE INFO

Article history:
Received 28 October 2022
Received in revised form 9 June 2023
Accepted 11 June 2023

Keywords:
Stereoscopic calibration
Precision measurement
Artificial intelligence
Open-source module
Python

ABSTRACT

The PYthon module for the CAlibration of cameras by SOloff's method (PYCASO) provides an open-source Python-based framework for stereoscopic reconstructions from pairs of 2D images. The determination of a matching function between image and physical coordinates is essential to accurately build 3D representations at any scales. Moreover, at the finest scales, this relationship is strongly non-linear, mainly due to optical distortions of the cameras (such as misalignment of the axis of the lens with that of the objective, refraction of optical windows etc.). Therefore, this paper proposes a general and accurate method based on the Soloff method through the PYCASO module. While the basic method is time-consuming, an acceleration of the procedure is proposed by relying on artificial intelligence (AI).

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00347
Permanent link to reproducible capsule	none
Legal Code License	CC BY-NC-ND 4.0
Code versioning system used	git
Software code languages, tools and services used	Python 3.8.10
Compilation requirements, operating environments and dependencies	numpy, pandas, matplotlib, cv2, sigfig, scipy, sklearn, skimage, joblib, compute_flow or disflow, seaborn
Optional libraries	joblib, GCpu_OpticalFlow
Support email for questions	eddy.caron@centralelille.fr

1. Motivation and significance

Modern experimental mechanics requires the use of instrumentation that is adapted to the finest scales while guaranteeing precise measurements. This is the case in solid mechanics such as plasticity at the microstructure scale [1] or non Newtonian fluid mechanics such as the study of turbulence. The current trend is to use cameras, which provide by definition "planar" information (2D). In this context, digital image correlation (DIC) is often used to quantify kinematic measurements. However, as a result of monotonic or cyclic loading, significant out-of-plane deformation by plasticity occurs that cannot be quantified with in-plane measurements alone. Nevertheless, exploiting stereo correlation at fine scales is difficult due to the optical distortions of the cameras, especially when using high-magnification cameras.

To solve these optical distortions, camera models are developed to compensate for these defects through a calibration step. The most commonly used model is the Pinhole model [2,3]. Unfortunately, this model is not relevant for sophisticated optical systems, especially those with significant misalignment between the optical and lens axis. Li et al. [4] illustrate these problems with their CMO (Common Main Objective) optical system, where the resolution is not achievable with a pinhole model. Other models have been proposed [4,5] but they increase the number of parameters to identify. Additionally, because no initial guess could be used to help with the inversion of the parameter system,

* Corresponding author.
E-mail address: eddy.caron@centralelille.fr (Eddy Caron).

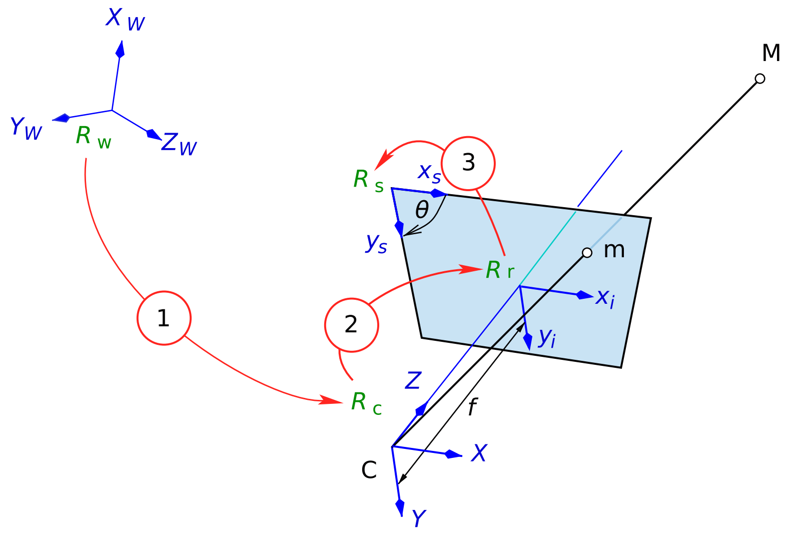


Fig. 1. The three elementary transformations (1, 2 and 3) of the pinhole camera model, and the associated coordinate systems. The first transformation relates the world coordinates of a scene point to coordinates in the camera system. The second one is the projection transformation of this point onto the retinal plane. The third one transform the point into the sensor coordinate system (pixel units).

and because, as we know, no Open-Source implementation exists, it is challenging to use these models in practice. However, the relevance of 2D3C (2 dimensions for 3 components) in mechanical fields is dependent on this calibration. It is then necessary to develop a relevant image reconstruction model.

In this context, two methods can be relevant: Soloff's method [6] and the direct method [7,8]. Both methods use polynomial resolution that links camera coordinates to geocentric coordinates and thus locates the positions of all points in the 3D-space. This paper offers a general view of a camera calibration algorithm: PYCASO. The main advantage of using PYCASO is that, unlike Pinhole model [9], it works on any optical system because it does not make any assumptions beforehand. Fig. 1 shows an example of the Pinhole model, which is defined by three elementary transformations that result in three different parameters: intrinsic parameters (that contain the coordinates of the camera's principal point (x_0, y_0) and its focal lengths (f_x, f_y)), extrinsic parameters (rotations **R** and translations **T**), and distortion parameters.

Since the time to solution with Soloff's method can be long, a third acceleration method based on AI is proposed. Existing methods that use AI for solving stereovision have been reviewed [10, 11] but they are not related to Soloff's method. This paper explains the differences between these three methods and shows that the direct method is less accurate than Soloff's method but much faster. Additionally, the AI method is a good compromise between Soloff's accuracy and speed.

2. Module description

PYCASO is a Python module that performs camera calibration using polynomial methods to precisely identify out-of-plane deformation. Both the direct and Soloff methods are implemented. Fig. 2 illustrates the different viewpoints, including the left camera, right camera, and 3D-space. By knowing the positions of a pattern detected by the cameras (denoted as $\boldsymbol{P_c}$, with $\boldsymbol{c} = \boldsymbol{l}$ for left or $\boldsymbol{r}$ for right) and the corresponding positions $\boldsymbol{x}$ of the same pattern in the global 3D-space, it is possible to fit a 3D coordinates polynomial function $S(\boldsymbol{x})$ ((2.4) in the calibration part). Subsequently, using new positions $\boldsymbol{P}'_{\boldsymbol{c}}$ detected by the cameras and the function $S(\boldsymbol{x})$, it is possible to identify the position $\boldsymbol{x}' = S^{-1}(\boldsymbol{P}'_{\boldsymbol{c}})$ in the global 3D-space (identification part).

2.1. Software architecture

PYCASO is a Python-based software tool that can be used to perform camera calibration using Soloff and direct methods. The calibration process involves the following steps, as depicted in Fig. 3.

- Generating a pattern that is suitable for the calibration process using the **'pattern.py'** module (Fig. 4).
- Printing out the pattern and positioning it on the device that needs to be calibrated (Fig. 5).
- Capturing calibration images and feeding them into PYCASO to determine the calibration constants S, S_0 (for Soloff method using Eq. (2.4)), and D (for direct method using Eq. (2.1)).
- Identifying each point in the calibration images using one of the two provided DIC functions, which are based on Open CV or GCpu_OpticalFlow algorithms.
- Connecting the identified coordinates from both cameras and using one of the PYCASO identification functions (direct, Soloff, or AI) to obtain their 3D coordinates.

PYCASO can be downloaded from its GitHub repository.

2.2. Generation of the calibration pattern

The function *pattern.ChAruco_board* (Appendix A.1) is used to create a ChAruco[1] chessboard that can be printed for stereo correlation. It takes three inputs:

- **'ncx'**: The number of squares for the chessboard in the x direction.
- **'ncy'**: The number of squares for the chessboard in the y direction.
- **'pixel_factor'**: The number of pixels per ChAruco pixel (in green in Fig. 4)

The size of the chessboard ($ncx \times ncy$) will depend on the shape of the space you want to calibrate. The pixel_factor will depend on the accuracy of your printer.

[1] ChAruco is a pattern generated with Open CV library. This pattern is a hybrid pattern which mixed good relative detection (chessboard corners) with absolute detection (Aruco patterns).

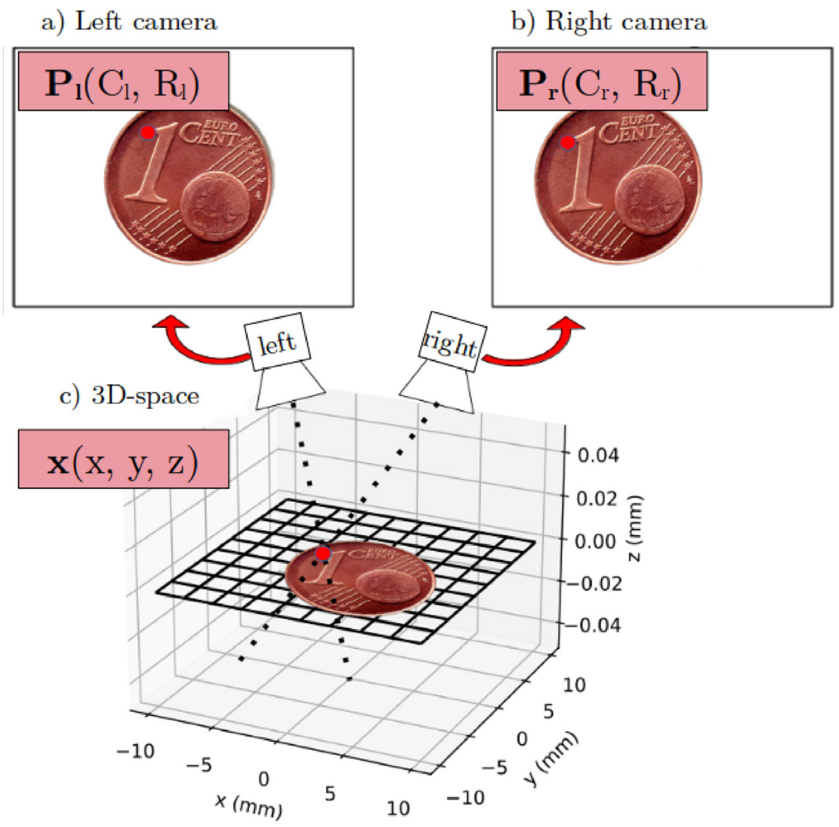


Fig. 2. Same point identified on a coin on (a) left camera, (b) right camera and (c) 3D-space.

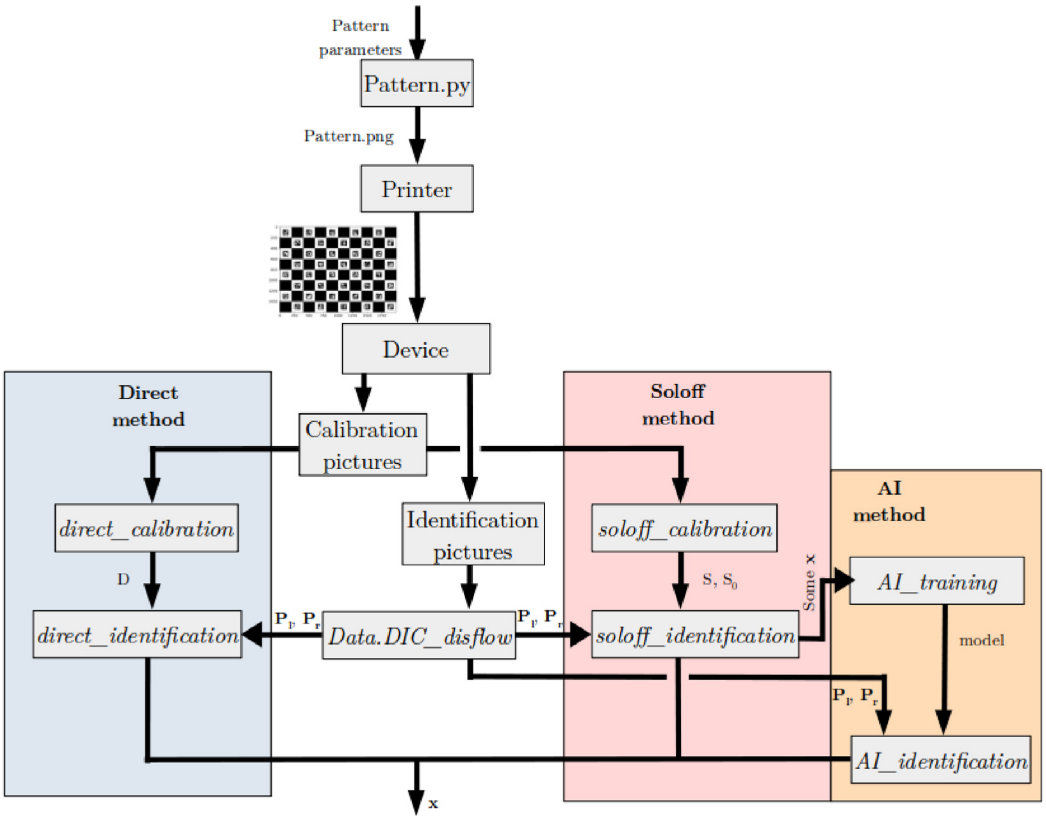
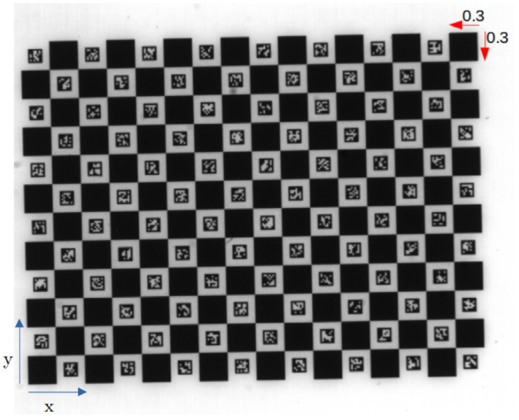
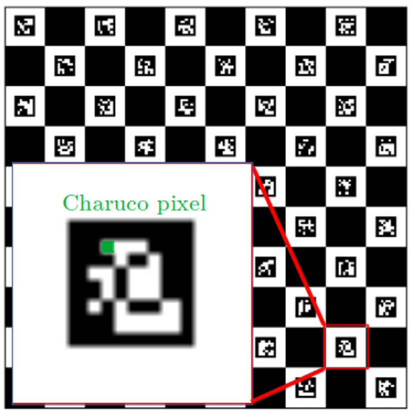


Fig. 3. PYCASO process, choosing the direct method (blue box), Soloff method (red box) or Soloff and AI method (red and yellow boxes).

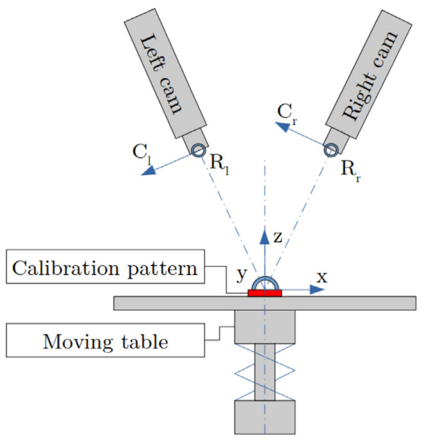


(a)

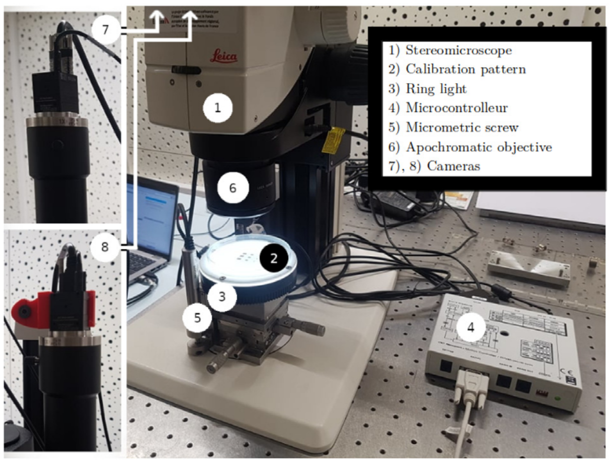


(b)

Fig. 4. (a) Pattern generation example. A 10 by 10 chessboard; pixel_factor = 10. (b) Picture of a 16 by 12 chessboard (ncx = 16 and ncy = 12); pixel_factor = 20; square size = 0.3 × 0.3 mm^2.



(a) Device scheme



(b) Real device picture

Fig. 5. Schematic (a) and real (b) device used for calibration part.

The output of the *pattern.ChAruco_board* function is a PNG figure (Fig. 4(b)) that needs to be printed for the device on a flat surface with higher resolution than the camera resolution (Fig. 4(a)).

2.3. Calibration device construction

To achieve accurate calibration, it is necessary to determine the real-world dimensions of the calibration pattern and to create a suitable device for translation of the target. It is also crucial to identify sufficient points in all directions ($\overrightarrow{x}$, $\overrightarrow{y}$, $\overrightarrow{z}$). Several papers in the literature have already explained how to create this device, and interested readers can refer to papers [7,8] for further information. Fig. 5 shows a setup of the device.

The device consists of a printed calibration pattern (ChAruco chessboard) mounted on a z-displacement motion system, two cameras, a light source, and the object that will be identified. Additionally, Fig. 6 shows an example of a coin photographed by the two cameras.

Once mounting and tested, PYCASO only requires the following results for 3D reconstruction:

- A pair of right and left pictures of the pattern for each z coordinate during the calibration step.
- A pair of right and left pictures of the object during identification step.

2.4. Calibration and identification of the points in 3D space

After constructing the calibration device 2.3, all necessary numerical materials are available. For the next step, two calibration methods can be employed in PYCASO. Both methods link the coordinates of 3D space points $\boldsymbol{x} = (x, y, z)$ (as shown in Fig. 6(b)) with the coordinates of the same points on the left camera $\boldsymbol{P_l} = (C_l, R_l)$ and the right camera $\boldsymbol{P_r} = (C_r, R_r)$ (as illustrated in Fig. 6(a)) using a polynomial relationship. (C_c is for column, R_c is for row)

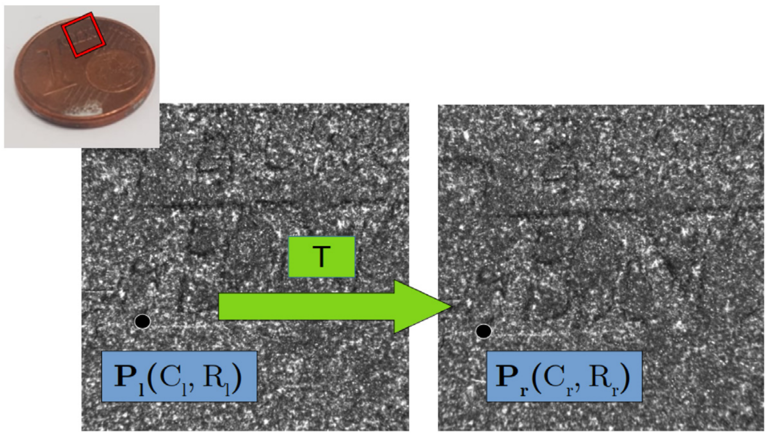
2.4.1. Direct method

The first method is a direct method that links the 2D and 3D coordinates with the polynomial equations:

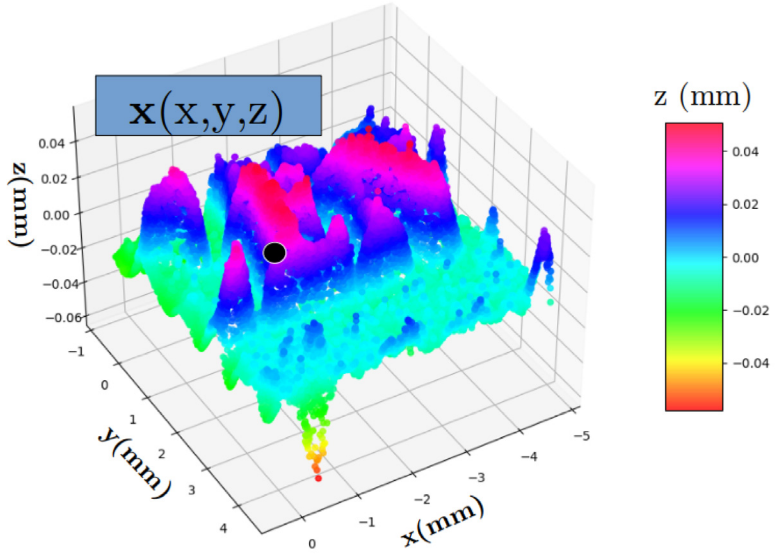
$$x = D_x(\boldsymbol{P_l}, \boldsymbol{P_r}) = d_0 + d_1 C_l + d_2 R_l + d_3 C_r + \cdots\cdots + d_{n-2} C_r R_r^{p-1} + d_{n-1} R_r^p \tag{2.1}$$

(*Respectively* Dy, Dz *for* y *and* z *coordinates*)

Here, n is the number of calibration constants and p is the degree of the polynomial. All coordinate combinations (C_l, R_l, C_r, R_r) are crossed.



(a)



(b)

Fig. 6. (a) Left and right pictures of a 1 cent coin. (b) 3D image. T represents the transformation field between left and right cameras.

The matrix form of these equations for any point k detected in all views is:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_k = \begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix}_k = d \cdot M_k = \begin{pmatrix} d_{0x} & \dots & d_{(n-1)x} \\ d_{0y} & \dots & d_{(n-1)y} \\ d_{0z} & \dots & d_{(n-1)z} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ C_l \\ \vdots \\ R_r^p \end{pmatrix}_k \tag{2.2}$$

Here, d is the calibration constants and M_k is the polynomial matrix coordinates. The method can be divided into three steps.

- The first step resolves the constants d of the polynomial functions D (D_x, D_y and D_z) with the Python function *pycaso.direct_calibration* (Appendix A.2).
  During this step, one of the problems is the detection of ChAruco patterns. Indeed, it happens that some of them cannot be detected. However, additional image processing is added to the basic Open CV detection, which finds all the missing points through the function *data_ library.complete_ missing_ points* (Appendix A.4).
- The second step is the identification of the same points in both cameras. In the illustrative example in Section 3, this identification is made with the function *data_library.DIC_ disflow* (Appendix A.5). This function first calculates the displacements fields U and V by DIC. The transformations U and V are grouped inside the T transformation in Fig. 6(a)) with the Open CV computer vision library. Then, any point $\boldsymbol{P_l}$ on the left camera is associated with its twin $\boldsymbol{P_r}$ on the right camera with the relations:

$$\boldsymbol{P_l} = \begin{pmatrix} C_l \\ R_l \end{pmatrix} ; \boldsymbol{P_r} = \begin{pmatrix} C_r \\ R_r \end{pmatrix} = \begin{pmatrix} C_l + U \\ R_l + V \end{pmatrix} \tag{2.3}$$

- The last step involves identifying the new points in 3D space with the Python function *pycaso.direct_identification* (Appendix A.3). This function puts the new coordinates $\boldsymbol{P_r}$ and $\boldsymbol{P_l}$ in the polynomial function and calculates the solution $\boldsymbol{x}$. It takes as arguments the points identified on the left and right cameras, the constants D of direct polynomials calculated during the first step, and the polynomial maximum degree used during the first step.

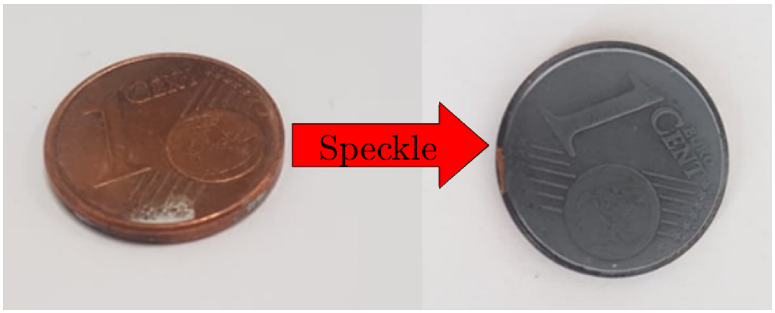


Fig. 7. Coin painted in black and speckled with a chalk solution (1/3 chalk–2/3 ethanol).

2.4.2. *Soloff method*

The second method is a Soloff method which links the 2D and 3D coordinates with the polynomial equations:

$$C_l = S_{Cl}(\boldsymbol{x}) = s_0+s_1x+s_2y+s_3z+\cdots\cdots+s_{n-2}yz^{p-1}+s_{n-1}z^p \quad (2.4)$$

(Respectively S_{Rl}, S_{Cr}, S_{Rr} for R_l, C_r and R_r coordinates)

Here, n is the number of calibration constants and p the degree of the polynomial. All of the coordinate combinations (x, y, z) are crossed.

The matrix form of those equations for any point k detected in all views is:

$$\begin{pmatrix} C_l \\ R_l \\ C_r \\ R_r \end{pmatrix}_k = \begin{pmatrix} S_{Cl} \\ S_{Rl} \\ S_{Cr} \\ S_{Rr} \end{pmatrix}_k = s \cdot M_k = \begin{pmatrix} s_{0Cl} & \dots & s_{(n-1)Cl} \\ s_{0Rl} & \dots & s_{(n-1)Rl} \\ s_{0Cr} & \dots & s_{(n-1)Cr} \\ s_{0Rr} & \dots & s_{(n-1)Rr} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \\ \vdots \\ z^p \end{pmatrix}_k \quad (2.5)$$

Here, s is the calibration constants and M_k the polynomial matrix coordinates. The method can be divided into three steps.

- The first step resolves the constants s of the polynomial functions $S(S_{Cl}, S_{Rl}, S_{Cr}$ and $S_{Rr})$ with the Python function *pycaso.soloff_calibration* (Appendix A.6).
- The second step is the same as the direct method, which involves identifying the same points in both cameras using the function *data.DIC_disflow*.
- The last step involves identifying the new points in 3D space with the Python function *pycaso.soloff_identification* (Appendix A.7).

The function used for the last step is more sophisticated than that used for the direct identification method. In this case, a Levenberg–Marquardt [12] iterative algorithm is used from the scipy library[2] to solve the polynomial function, which cannot be used directly as in the direct method because the solution is on the right side of the function (Eq. (2.4)).

- A first approximation close to the final solution is calculated.
  - The polynomial constants S_0 are obtained by repeating the first step but for a polynomial degree of one.
  - Since there are no crossed coordinates in a polynomial degree of one, it becomes possible to invert (2.5) to get a preliminary estimation $\boldsymbol{x_0}(x_0, y_0, z_0)$ of $\boldsymbol{x}(x, y, z)$.
- Finally, the Levenberg–Marquardt algorithm leverages this initial estimate $\boldsymbol{x_0}$ to determine the best solution.

2 scipy.optimize.curve_fit

All of these calculations are implemented in the Python function *pycaso.soloff_identification*. This function takes as input the points identified on the left and right cameras, the constants S and S_0 calculated during the first step, and the maximum polynomial degree used during the first step.

2.4.3. *AI method*

In order to speed up the Soloff method's time to solution (Section 4.3), an AI method has been developed. The process involves three steps.

First, the Soloff method is applied to a subset of points, typically $k = 50000$, to perform a full calibration, a DIC step, and an identification step.

Second, an AI model is trained using these data, along with the *pycaso.AI_training* function from Appendix A.8. The *RandomForestRegressor* function from the *sklearn* Python module is used to generate the model. This function takes the four coordinates (C_l, R_l, C_r, R_r) as inputs and outputs the three coordinates (x, y, z).

The final step involves using new cameras coordinates (inputs) and the trained model to generate unknown 3D coordinates (outputs) with the function *pycaso.AI_identification* (Appendix A.9). This function takes as arguments the points identified on the left and right cameras and the model built in the previous step.

3. Illustrative example — Coin identification by Digital Image Correlation (DIC)

To illustrate the software, let us try to identify the shape of a coin speckled with a chalk solution (Fig. 7).

3.1. *Pattern*

The pattern chosen to identify the coin is a 16 × 12 ChAruco chessboard (Appendix B). This pattern is a chromium deposit on an alumina support made by beam lithography. This support needs a backlight (Fig. 5(b)).

3.2. *Equipment*

The optical system is composed of two XiQ cameras and a stereo microscope Leica M205C.

The mechanical system to move the pattern during the calibration part is composed of a motion controller SMC100CC used to drive a motorized actuator TRA6PPD fixed on a 3-axes support M-562F-XYZ.

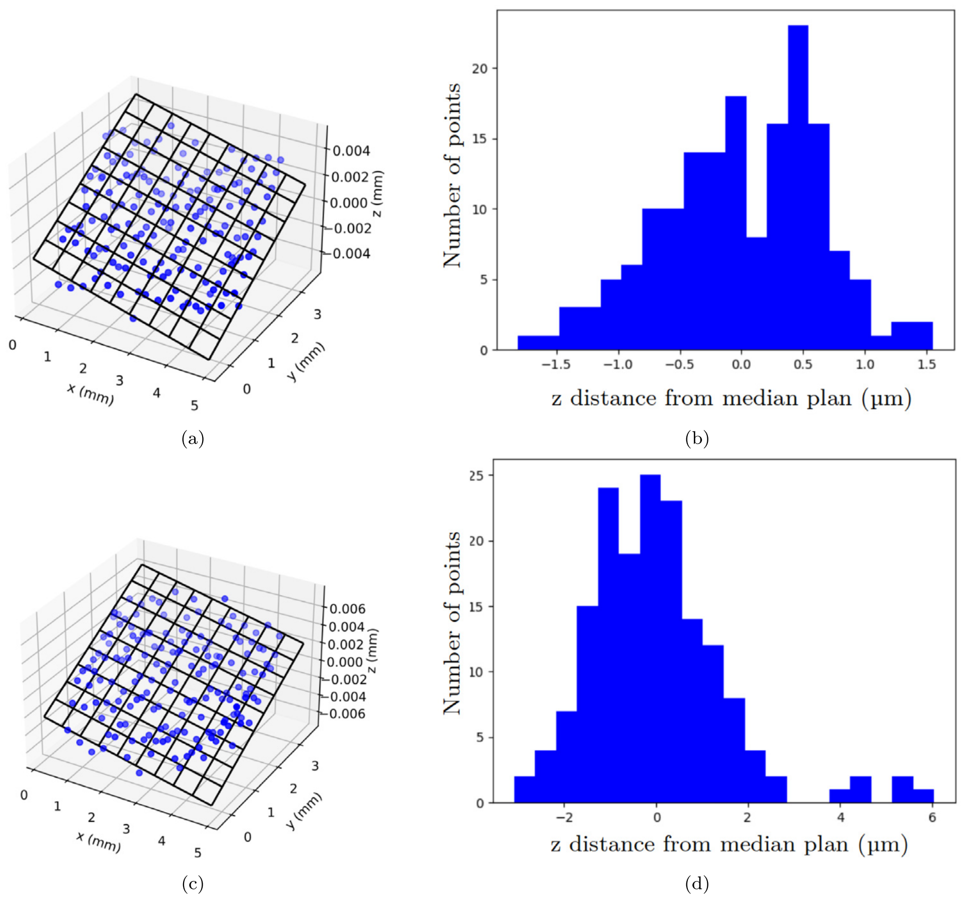


Fig. 8. Soloff (a) and direct (c) pattern points identify in 3D space. Knowing that the points detected on the pattern are all on the same plan, it is possible to estimate the reconstruction quality by fitting the median plan (plot in black) and look to the distribution around it. Soloff (b) and direct (d) repartition of z distance from points identify to median plan.

3.3. Processing

The global processing is divided into two parts:

- The calibration part is composed of the ChAruco chessboard, the backlight, all the mechanical and optical systems. One hundred pictures are taken in the right and left cameras at each z-coordinate from 2.96 mm to 3.04 mm and the calibration parameters are calculated (Appendix C.1).
- The identification part is composed of the speckled coin, the direct light, all the optical system and the 3-axes support. The DIC is used between left and right pictures (Appendix C.2 and Fig. 6). The 3D profile is then calculated by PYCASO and projected onto the left camera (z-coordinate viewed from the left camera referential) with direct, Soloff and AI methods and compared with a profilometric measurement (Fig. 9).
  Finally, the results of the z-coordinate projected onto the left camera with all methods are available in Fig. 9.

4. Performance

4.1. Accuracy

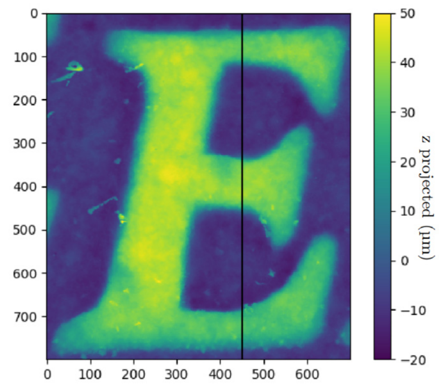
The accuracy of each calibration method is compared with a flat surface evaluation. With a rotated identification of the pattern (Fig. 4) considered as a flat surface, it is possible to evaluate the method by fitting a plan and determining the distance distributions around it. According to Fig. 8, the Soloff's method (a, b) is better than the direct one (c, d) to identify the z position with a standard deviation of 0.65 µm (0.24 pixels) against 1.48 µm (0.55 pixels).

4.2. Comparison with a profilometry

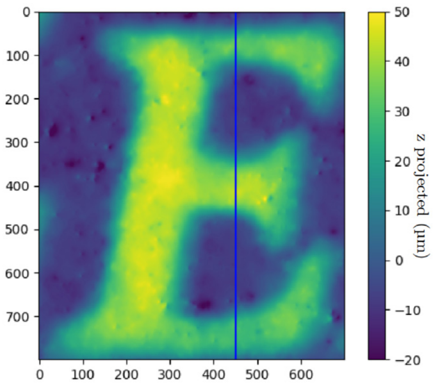
A comparison with an accurate profilometry method is available in Fig. 9. A reset is applied on the profilometric view and a same line is drawn in the middle of the pictures. Topographies are compared through those lines on Fig. 9(e). This comparison shows that the peaks are detected at the same position with a similar size (40–50 µm). The comparison between the AI and the Soloff curves shows a good match, although some local differences can be detected (orange circle in Fig. 9(e)). In fact, the mean distance between Soloff and AI methods is 1.40 µm (0.55 pixels). This result is better than the direct method where the mean distance between Soloff and direct methods is 5.1 µm (2.01 pixels). Also, all methods smooth the shapes compared with the profilometric curve. This smoothing can be explained by the DIC software.

Indeed, by changing the regularization parameter (Tikhonov[3]) and the median filter size of the GCpu_OpticalFlow algorithm, it is possible to have a better fitting with the profilometric curve (Fig. 9(e)). A crossed study with a variation of Tikhonov parameter from 1000 to 100000 and a variation of the median filter size from

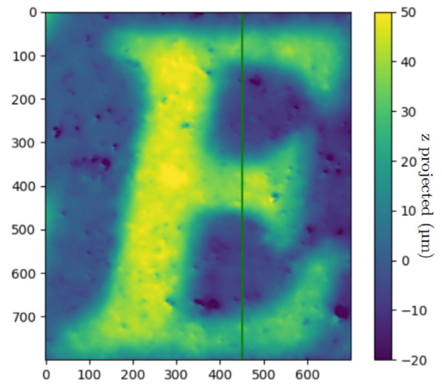
[3] Tickonov is a regularization parameter used in optical flow method [13] and has the same role as the element size in DIC methods. It allows obtaining optical flow fields.



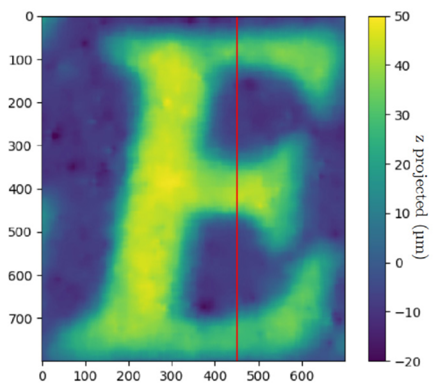
(a) Profilometric measurement



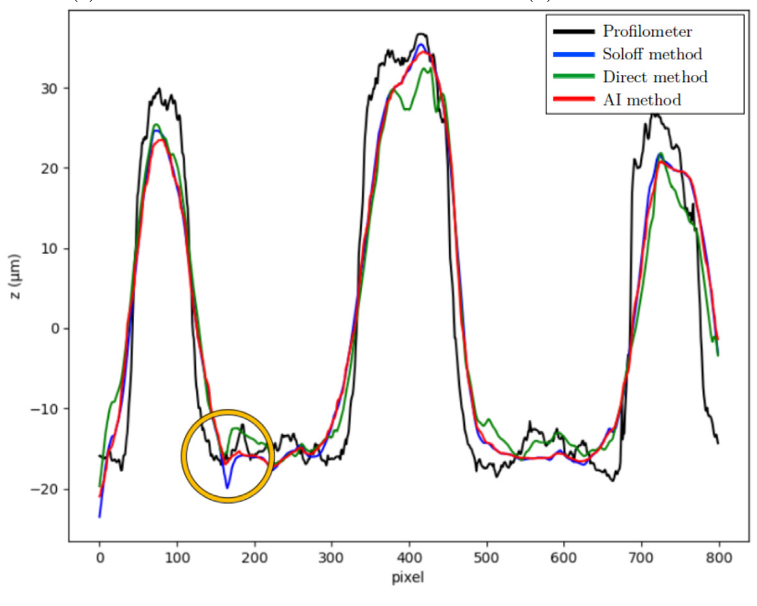
(b) Soloff method



(c) Direct method



(d) AI method



(e) Comparison of z distribution over lines

Fig. 9. (a) Profilometric topography. (b) Soloff topography. (c) Direct topography. (d) AI topography. (e) *z*-distribution of the 450th column.

1 to 5 gave us the best solution for a Tikhonov parameter equal to 20000 and a median filter size equal to 3. This configuration is shown in Fig. 10.

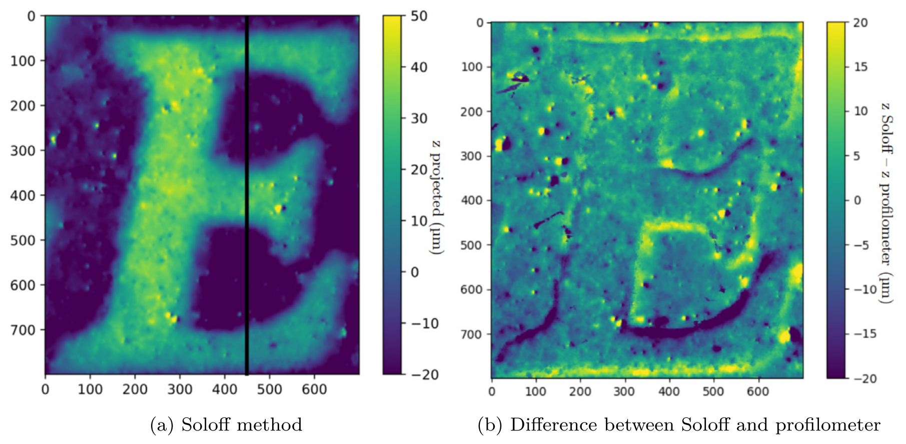
4.3. Time to solution

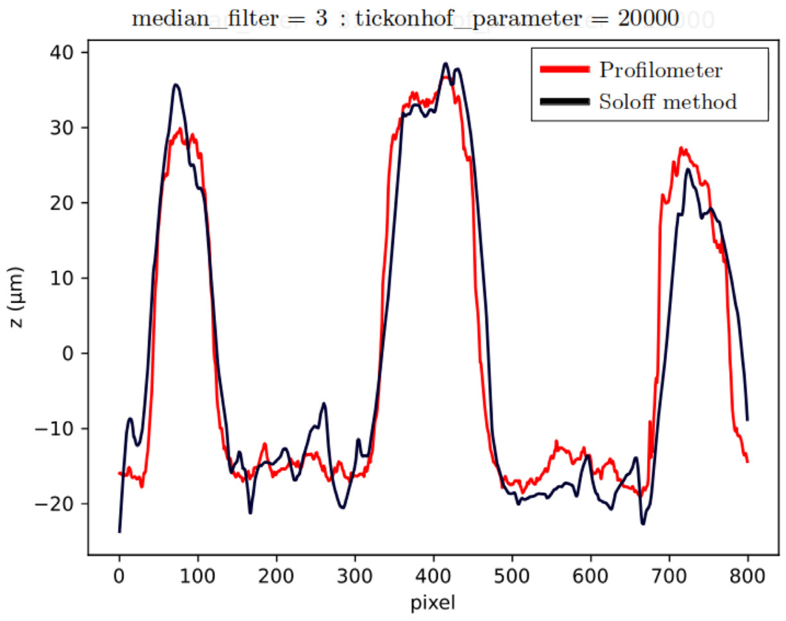
The main problem of the Soloff method arises from the Levenberg–Marquardt loop used for each pair of points to calculate the 3D position $\boldsymbol{x}$.

For a pair of images with a resolution of 2048 × 2048 pixels^2 (4 million points), the Soloff method takes 609 s to solve, while the direct method requires only 2 s. The AI method (Trained on 100,000 points calculated using Soloff) needs 210 s, including the training part.

5. Impact

The 3D DIC is an accurate method that is essential for measuring out of plane displacement fields. While some industrial





(c) Soloff method and profilometry comparison

Fig. 10. Topographic result with Soloff method for a Tikhonov parameter = 20000 and a median filter size = 3 (a). Comparisons of the topography (b) and the line distribution with profilometry (c).

solutions exist, they can be expensive, and not all laboratories can afford them. Other Open-Source solutions exist for 2D [14] and 3D [15,16] correlation, but they are based on a pinhole model. The PYCASO module offers a unique way to perform 3D DIC at a microscale with any optical system, especially when the pinhole model cannot be used.

6. Conclusion

This paper proposes a new software package called PYCASO that builds a processing pipeline for planar image calibration that includes three different methods: direct, Soloff, and AI-assisted Soloff. All of these methods output the topography of the sample using Digital Image Correlation (DIC). PYCASO allows users to follow in-situ topography during testing. This stereoscopic method is available for any optical system, making it possible to use the same protocol at the appropriate scale.

However, there are some limitations to be aware of. First, the detection of the ChAruco pattern is not perfect but additional picture processing (Section 2.4.1) resolves this issue. Second, all results depend on the accuracy of the calibration, so it is essential to follow all the steps and use accurate equipment. Finally, all methods are not equivalent; while the Soloff method is more accurate, the direct method is much faster, even if Soloff's method can be accelerated using the AI method.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All datas are on Github.

Acknowledgments

The authors would like to acknowledge the CNRS Research Federation on Ground Transports and Mobility, in articulation with the Elsat2020 and RITMEA projects supported by the European Community, the French Ministry of Higher Education and Research and the Hauts de France Regional Council.

Appendix A. Functions signatures

A.1. ChAruco board

```
ChAruco_board (ncx : int,
               ncy : int,
               pixel_factor : int = 1)
               —> np.ndarray :
```

A.2. Direct calibration

```
direct_calibration (z_list : np.ndarray,
                    direct_pform : int,
                    left_folder : str = 'left_calibration',
                    right_folder : str = 'right_calibration',
                    name : str = 'calibration',
                    saving_folder : str = 'results',
                    ncx : int = 16,
                    ncy : int = 12,
                    sqr : float = 0.3,
                    hybrid_verification : bool = False,
                    multifolder : bool = False,
                    plotting : bool = False)
                    —> (np.ndarray,
                        np.ndarray) :
```

A.3. Direct identification

```
direct_identification (Xc1_identified : np.ndarray,
                       Xc2_identified : np.ndarray,
                       direct_constants : np.ndarray,
                       direct_pform : int) —> np.ndarray :
```

A.4. Complete missing points

```
complete_missing_points (corners_list : np.ndarray,
                         im : str,
                         ncx : int = 16,
                         ncy : int = 12,
                         sqr : float = 0.3,
                         hybrid_verification : bool = False)
                         —> list :
```

A.5. DIC disflow

```
DIC_disflow (DIC_dict : dict,
             flip : bool = False,
             image_ids : list = [False])
             —> (np.ndarray,
                 np.ndarray) :
```

A.6. Soloff calibration

```
Soloff_calibration (z_list : np.ndarray,
                    Soloff_pform : int,
                    left_folder : str = 'left_calibration',
                    right_folder : str = 'right_calibration',
                    name : str = 'calibration',
                    saving_folder : str = 'results',
                    ncx : int = 16,
                    ncy : int = 12,
                    sqr : float = 0.3,
                    hybrid_verification : bool = False,
                    multifolder : bool = False,
                    plotting : bool = False)
                    —> (np.ndarray,
                        np.ndarray,
                        np.ndarray):
```

A.7. Soloff identification

```
Soloff_identification (Xc1_identified : np.ndarray,
                       Xc2_identified : np.ndarray,
                       Soloff_constants0 : np.ndarray,
                       Soloff_constants : np.ndarray,
                       Soloff_pform : int,
                       method : str = 'curve_fit')
                       —> np.ndarray :
```

A.8. AI training

```
AI_training (X_c1 : np.ndarray,
             X_c2 : np.ndarray,
             xSoloff_solution : np.ndarray,
             AI_training_size : int = 1000,
             file : str = 'Soloff_AI_training.csv',
             method : str = 'simultaneously')
             —> sklearn.ensemble._forest.
                RandomForestRegressor :
```

A.9. AI identification

```
AI_identification (X_c1 : np.ndarray,
                   X_c2 : np.ndarray,
                   model : sklearn.ensemble.
                           _forest.RandomForestRegressor,
                   method : str = 'simultaneously')
                   —> np.ndarray:
```

Appendix B. Pattern generation

```
import sys
import pathlib
import pattern
ncx = 16
ncy = 12
pixel_factor = 10
Pattern = ChAruco_board(ncx,
                        ncy,
                        pixel_factor)
```

Appendix C. 3D reconstruction

C.1. Calibration

```
import numpy as np
import sys
import pathlib
import os
from glob import glob

import pycaso as pcs
import data_library as data
import matplotlib.pyplot as plt

# Define the inputs
calibration_dict = {
     'left_folder' : 'Images_example/left_calibration',
     'right_folder' : 'Images_example/right_calibration',
     'name' : 'micro_calibration',
     'saving_folder' : 'results/main_exemple',
     'ncx' : 16,
     'ncy' : 12,
     'sqr' : 0.3}

DIC_dict = {
     'left_folder' : 'Images_example/left_identification',
     'right_folder' : 'Images_example/right_identification',
     'saving_folder' : 'results/main_exemple',
     'name' : 'micro_identification',
     'window' : [[300, 1700], [300, 1700]]}

# Create the list of z plans
Folder = calibration_dict['left_folder']
Imgs = sorted(glob(str(Folder) + '/*'))
z_list = np.zeros((len(Imgs)))
for i in range (len(Imgs)) :
    z_list[i] = float(Imgs[i][len(Folder) + 1:-4])

# Chose the degrees for Soloff and direct polynomial fitting
Soloff_pform = 332
direct_pform = 4

# Create the result folder if not exist
saving_folder = 'results/main_exemple'
if os.path.exists(saving_folder) :
    ()
else :
    P = pathlib.Path(saving_folder)
    pathlib.Path.mkdir(P, parents = True)

# Direct method
direct_A, Mag = pcs.direct_calibration (z_list,
                                        direct_pform,
                                        **calibration_dict)

# Soloff method
A111, A_pol, Mag = pcs.Soloff_calibration (z_list,
                                           Soloff_pform,
                                           **calibration_dict)
```

C.2. DIC calculation

```
# On the Pycaso codes, the notation of Pl = Xl (resp Pr = Xr)
Xl, Xr = data.DIC_get_positions(DIC_dict)
```

C.3. Direct, Soloff and AI reconstruction

```
# Chose right and left coordinates
Xl = Xl[0]
Xr = Xr[0]


# Direct identification
xDirect_solution = pcs.direct_identification (Xl,
                                              Xr,
                                              direct_A,
                                              direct_pform)
xD, yD, zD = xDirect_solution
wnd = DIC_dict['window']
zD = zD.reshape((wnd[0][1] - wnd[0][0], wnd[1][1] - wnd[1][0]))

plt.figure()
plt.imshow(zD)
plt.title('Z projected on left camera with direct calculation')
cb = plt.colorbar()
cb.set_label('z in mm')
plt.show()


# Soloff identification
xSoloff_solution = pcs.Soloff_identification (Xl,
                                              Xr,
                                              A111,
                                              A_pol,
                                              Soloff_pform)
xS, yS, zS = xSoloff_solution
zS = zS.reshape((wnd[0][1] - wnd[0][0], wnd[1][1] - wnd[1][0]))

plt.figure()
plt.imshow(zS)
plt.title('Z projected on left camera with Soloff calculation')
cb = plt.colorbar()
cb.set_label('z in mm')
plt.show()


# AI identification
AI_training_size = 50000
model = pcs.AI_training (Xl,
                         Xr,
                         xSoloff_solution,
                         AI_training_size = AI_training_size)
xAI_solution = pcs.AI_identification (Xl,
                                      Xr,
                                      model)
xAI, yAI, zAI = xAI_solution
zAI = zAI.reshape((wnd[0][1] - wnd[0][0], wnd[1][1] - wnd[1][0]))

plt.figure()
plt.imshow(zAI)
plt.title('Z projected on left camera with Soloff calculation')
cb = plt.colorbar()
cb.set_label('z in mm')
plt.show()
```

References

[1] Basseville S, Cailletaud G, Ghidossi T, Guilhem Y, Lacoste E, Proudhon H, Signor L, Villechaise P. Numerical analysis on the local mechanical fields in polycrystalline 316ln stainless steel under cyclic fatigue loading: Comparison with experimental results. Mater Sci Eng A 2017;696:122–36. http://dx.doi.org/10.1016/j.msea.2017.04.023.

[2] Zhang Z. A flexible new technique for camera calibration. IEEE Trans Pattern Anal Mach Intell 2000;22(11):1330–4. http://dx.doi.org/10.1109/34.888718.

[3] Sutton MA, Orteu JJ, Schreier HW. Image correlation for shape motion and deformation measurements : basic concepts, theory and applications. Springer; 2009.

[4] Li W, Wei Z, Zhang G. Affine calibration based on invariable extrinsic parameters for stereo light microscope. Opt Eng 2014;53(10):102105. http://dx.doi.org/10.1117/1.OE.53.10.102105.

[5] Gaudenz D, Kubler O. Calibration of cmo-stereo-microscopes in a micro robot system. Int Arch Photogram Remote Sens 1995.

[6] Soloff SM, Adrian RJ, Liu Z-C. Distortion compensation for generalized stereoscopic particle image velocimetry. Meas Sci Technol 1997;8(12):1441–54. http://dx.doi.org/10.1088/0957-0233/8/12/008.

[7] Leandry I, Brèque C, Valle V. Calibration of a structured-light projection system: Development to large dimension objects. Opt Lasers Eng 2012;50(3):373–9. http://dx.doi.org/10.1016/j.optlaseng.2011.10.020.

[8] Valle V, Laou L, Léandry I, Yotte S, Rossignol S, Hedan S. Crack analysis in mudbricks under compression using specific development of stereo-digital image correlation. Exp Mech 2018;58(3):475–86. http://dx.doi.org/10.1007/s11340-017-0363-2.

[9] Xu G, Zhang Z. Epipolar Geometry in Stereo Motion and Object Recognition: A Unified Approach, vol. 30. 1996.

[10] Yin JE, Eisenstein DJ, Finkbeiner DP, Stubbs CW, Wang Y. Active optical control with machine learning: A proof of concept for the vera c. Rubin Observ. 2021;23. http://dx.doi.org/10.3847/1538-3881/abe9b9.

[11] Hamid MS, Manap NA, Hamzah RA, Kdmin AF. Stereo matching algorithm based on deep learning: A survey, 23. 2020, http://dx.doi.org/10.1016/j.jksuci.2020.08.011.

[12] Madsen K, Nielsen HB, Tingleff O. Methods for non-linear least squares problems. 2004.

[13] Horn BKP, Schunck BG. Determining optical flow. 1981, https://doi-org.ressources-electroniques.univ-lille.fr/10.1016/0004-3702(81)90024-2.

[14] Filho D, Nunes L, Xavier J. ICorrVision-2D: An integrated python-based open-source digital image correlation software for in-plane measurements (part 1). SoftwareX 2022;19(1):101131. http://dx.doi.org/10.1016/j.softx.2022.101131.

[15] João F, Nunes L, Xavier J. ICorrVision-3D: An integrated python-based open-source digital image correlation software for in-plane and out-of-plane measurements (part 2). SoftwareX 2022;19(1):101132. http://dx.doi.org/10.1016/j.softx.2022.101132.

[16] Dana S, Moerman KM, Jaeger AM, Genovese K, Herr HM. MultiDIC: An open-source toolbox for multi-view 3D digital image correlation, 4. 2018, http://dx.doi.org/10.1109/ACCESS.2018.2843725.