GPUCorrel: A GPU accelerated Digital Image Correlation software written in Python

Victor Couty, Jean-François Witz, Pauline Lecomte-Grosbras, Julien Berthe, Eric Deletombe, Mathias Brieu

SoftwareX 16 (2021) 100815

Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

GPUCorrel: A GPU accelerated Digital Image Correlation software written in Python

Victor Couty [a,*], Jean-François Witz [a], Pauline Lecomte-Grosbras [a], Julien Berthe [b,a], Eric Deletombe [b,a], Mathias Brieu [c]

[a] Univ. Lille, CNRS, Centrale Lille, UMR 9013 - LaMcube - Laboratoire de Mécanique, Multiphysique, Multi-échelle, F-59000 Lille, France
[b] ONERA, The French Aerospace Lab, Lille, France
[c] Mechanical Engineering Dpt, College ECST, California State University, Los Angeles, USA

ARTICLE INFO

Article history:
Received 14 January 2021
Received in revised form 6 August 2021
Accepted 13 September 2021

Keywords:
DIC
Digital Image Correlation
GPU
Mechanics
CUDA
Python
Strain
Measurement

ABSTRACT

This article presents an open-source Integrated Digital Image Correlation (I-DIC) software written in Python using CUDA-enabled GPUs designed to run at high (1–100 Hz) frequency. The field computation is performed using a global approach and the result is a projection of the real field in a user-defined base of fields. This software can be used in many applications and one use in experimental mechanics is demonstrated by driving a bi-axial tensile test on a cruciform specimen.

Code Metadata

Current code version	v0.2.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00012
Code Ocean compute capsule	
Legal Code License	GPLv2
Code versioning system used	git
Software code languages, tools, and services used	Python3, CUDA
Compilation requirements, operating environments	CUDA, pyCUDA, Numpy
If available Link to developer documentation/manual	Documentation included in comments
Support email for questions	victor.couty@centralelille.fr

1. Motivation and significance

Material behavior is commonly studied by performing series of tests where samples are submitted to chosen loads and their responses are measured with various sensors such as strain gauges or load cells. The most common test is the uniaxial tensile test [1] where a sample of constant section in the region of interest is pulled in a single direction.

Most of the time, the uniaxial tensile test on a homogeneous specimen makes it possible to calculate the imposed stress and plot it as a function of the measured strain. It is therefore easy to plot stress–strain curves from this test to identify material behavior. However, when cross-sections are not constant, loads are complex, the material is heterogeneous or damage or failure cause complex phenomena, this test requires a full analysis or an adequate piloting. Full field measurement methods are a solution to measure the strain state on any point of the specimen, allowing to draw conclusion from this kind of test.

Digital Image Correlation (DIC) is a method to extract a 2D or 3D displacement fields of points on the surface of a sample using

* Corresponding author.
E-mail address: victor.couty@centralelille.fr (Victor Couty).

https://doi.org/10.1016/j.softx.2021.100815

cameras [2]. It is not unusual to perform tests controlled by readings of force, apparent strain or actuator position [3]. However, full-field DIC is mostly used as a post-processing tool even though it carries extremely rich information on the specimen state. This is mainly due to the high computational cost of DIC.

There are numerous open-source DIC projects available, developed for various domains such as experimental mechanics [4–13], fluid dynamics [14,15] or medical purposes [16,17]. The formulation may be different, but the general idea is always to find a displacement field that occurred between two images. In all these scenarios, accuracy prevails over computational speed. This is why DIC is performed offline and usually takes between tens of seconds to several hours for a single pair of images depending on the algorithm, the implementation, the inputs and the hardware. Computer vision is a domain that makes use of time-constrained DIC (or "Optical Flow" to match the terminology used in this domain). This is why some algorithms such a DISflow [18] are designed with computational efficiency in mind and can be tuned to be extremely fast at the expense of precision. The presented algorithm is also coded to be as fast as possible, with the goal to drive tests in real time based on the results from the DIC. As an example, low-end laptop GPU dating from 2013 (K610M) can measure rigid-body motions on a 640 × 480 stream at about 15 fps. Using high-end graphics cards allows higher resolutions and faster processing. Tests on an RTX 3070 showed that the card is able to measure rigid body motions and homogeneous strains at 40 fps on a 4K (3840 × 2160) stream.

To be able to drive a test also implies the capacity to extract meaningful information from the results. This is why the presented algorithm is using Integrated DIC (I-DIC). The term "integrated" refers to the fact that the resulting kinematic fields are included in the research algorithm: the program expects a list of displacement fields to identify from the images. The result will only be a projection of the actual field in the given base. This means that the user can provide a restricted number of fields and the result will only have as many dimensions. This can be used to extract global information such as motion amplitude or global strain, but also more complex behaviors that can be extracted for dense fields of the previous tests or from simulations using a base reduction technique such as Proper Orthogonal Decomposition (POD).

The use of GPU allows a massively parallel processing of the data, giving access to much faster computation. A notable example of GPU-accelerated DIC software is eFOLKI [19].

Full-field displacements represent a large amount of data, especially when using high-resolution cameras. Extracting meaningful values to drive a test from these large arrays may also be computationally expensive. However the ever growing speed of computers and GPU computing made DIC-driven tests accessible. It offers the possibility not only to drive the test based on the position or deformation of any point of the sample during the test, but also to use projection on pre-defined bases of kinematic fields, allowing the extraction of meaningful values from the state of the material. Such bases can be defined prior to the test by numerical simulations, from previous tests or even "on-the-fly" during the test. This paper presents a Python software meant to perform DIC fast enough to process the video feed from a camera in real-time using CUDA-enabled GPUs, giving access to correlation-controlled mechanical tests. The following section will explain the principle of this algorithm.

2. Software description

The method presented in this paper is based on the global approach of correlation: the matching algorithm will not work on sub-frames or a finite-element model, but instead process the whole image at once. The problem has twice as many unknowns as pixels in an image. In order to solve the problem, the number of parameters to optimize must be lowered. This implies a restriction of the research direction to a given base. The choice of this base is critical as it will affect both performance and correctness of the result.

The next subsection will present a mathematical representation of the problem and the approach to solve it in this software.

2.1. Principle of global correlation

Let F and G be the two images on which the correlation will be performed i.e. two $\Re^2 \Rightarrow \Re$ functions. The goal of correlation is to find $u(x)$ so that $G(x + u(x)) = F(x)$ for all x in the Region Of Interest (ROI) where u is a $\Re^2 \Rightarrow \Re^2$ function defining the displacement at given coordinates. Because this is an ill-posed problem, we will add constraints on the definition of u [2]. For this algorithm, we will assume that u is a linear combination of chosen base functions b_i. k is the vector of corresponding coefficients and the function for a given k is noted u_k so we have (1) where $\{B\}$ is the vector of all the base fields *i.e.* the base in which the actual field is projected.

$$u_k(x) = \{B\}^T\{k\} \tag{1}$$

Since strict equality between $F(x)$ and $G(x + u_k(x))$ cannot be achieved for all x for multiple reasons including measurement noise, digitization and the evolution of the surface of the sample, the algorithm will seek the closest match by minimizing their difference. Let us introduce a metric to quantify the difference between the two images. Using the Sum of Squared Differences (SSD) over the domain of the images, we have (2).

$$r = \iint_{\Omega} [F(x) - G(x + u_k(x))]^2 \, dx \tag{2}$$

It is asserted that the real displacement field is the minimum of the residual function r. To focus the computation around the ROI, we can use a mask $M(x)$. It is a function that returns a non zero value for every coordinate belonging to the ROI and zero for every other coordinates. Since we are working on discrete data (digital images), the integrals are actually performed as simple sums giving (3).

$$r = \sum_{x \in \Omega} M(x)(F(x) - G(x + u_k(x)))^2 \tag{3}$$

Where x iterates over every pixels of the image. Since the algorithm is working with a sub-pixel precision, an interpolation function is needed to fetch values at non-integer coordinates. To avoid the problem of extrapolation outside the boundaries of the image, the ROI must be smaller than the original image.

This choice of the base $\{B\}$ is critical since the computed field will be a projection of the real field in this base. Different approaches were used to make this choice in the literature. For instance, it can be built using every expected motion such as rigid body motion or homogeneous strain fields. For example the fields represented in Fig. 1 can be used to measure rigid body motions and homogeneous transformations.

In solid mechanics, it is common to use a finite-element approach and use the displacement fields based on the degrees of freedom of a mesh [20,21]. Other methods could be used to choose $\{B\}$, such as Proper Generalized Decomposition (PGD) on a set of measured or simulated fields [22].

Once the base has been defined, k will be sought using an iterative method: it will be updated and computed until a stopping criterion is reached. Differentiating (3) gives (4).

$$\frac{\delta r}{\delta k} = 2 \sum_{x \in \Omega} [(F(x) - G(x + u_k(x))) \, B \, \nabla G(x + u_k(x))] \tag{4}$$

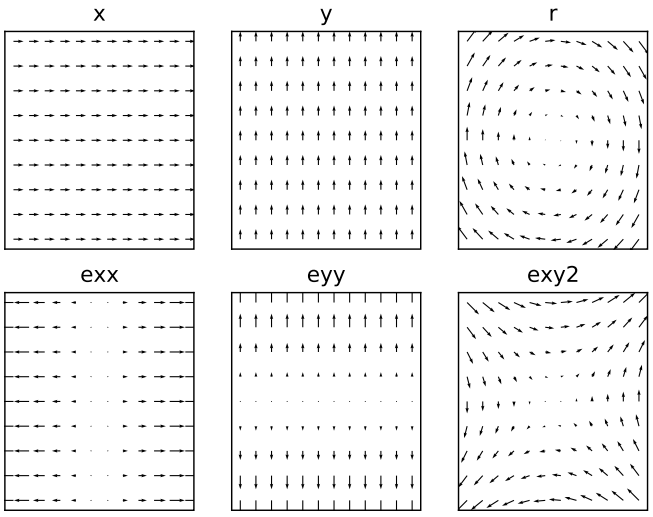

Fig. 1. Illustration of six commonly identified fields.

In order to avoid costly computation of $\nabla G(x + u_k(x))$ at each step, it will be estimated by ∇F. At convergence, we have $F(x) \simeq G(x + u_k(x))$ therefore it will converge towards the same solution [23]. The image gradients ∇F will be computed only when updating the reference image and the base $\{B\}$ is chosen prior to the computation. This means that the direction can be computed for each parameter by making a reduction of the entry wise product of a pre-computed matrix $B_i \cdot \nabla F$ and the difference of the two images at the current step. The minimization is performed using a quasi-Newton method [24].

2.2. Software functionalities

In order to reduce the amount of data to be processed and speed up the algorithm, the computation is performed following a coarse-to-fine (or pyramidal) approach as illustrated in Fig. 2. The images are resampled several times and the first iterations are performed on the lowest resolution, reducing significantly their cost. This approach also has the advantage of smoothing the image on the initial steps. This prevents the algorithm from falling into a local minimum if the initial guess is not close enough to the optimal solution.

Once the displacement has been computed on a lower resolution, the field is upscaled and used to initialize the next level, all the way to the original image. This reduces the number of iterations to be performed on the higher resolution. By default, the resampling factor is 2 and the image is resampled 5 times, meaning that the first image is 32 times smaller and contains 1024 times fewer pixels than the full resolution.

These parameters can be set to any value in the main class *GPUCorrel* with the arguments *resampling_factor* and *levels* respectively. Each level provides a close estimation of the result of the next one, allowing to perform only a few iterations on each level. This method has been proved to be effective and provides a significant speed-up, reducing the number of iterations to be performed on large images [25]. For the presented software, resampling is accelerated massively thanks to the texture mapping units of the GPU.

The code has a limited set of functionalities: its final goal is to take two images and a set of fields and return the linear combination of the fields that minimizes the differences between the two images. It creates a list of levels that will process the images at different resolutions. Each level is represented by an instance of the class *Correl_level*. The reference image and the fields are downsampled to the correct resolution for each level and the $B_i \nabla F$ matrix are computed. When the second image is given, it is downsampled and the smallest level computes the residual image. It is used with the previously computed tables to compute the research direction. The field is updated and a new iteration begins. If the residual increases, the iteration stop and the level returns the best solution. It is then fed to the next level successively up to the native resolution.

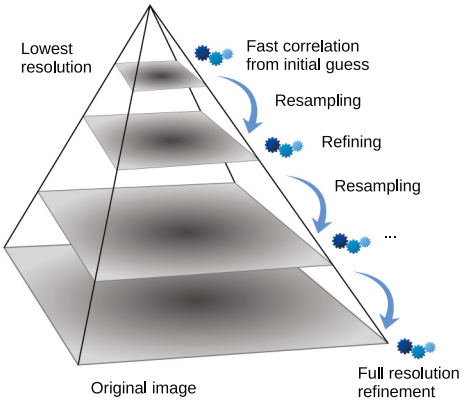

Fig. 2. Illustration of the pyramidal approach.

2.3. Software architecture

The software was designed to be as simple as possible to understand. The goal is for researchers who are not necessarily experimented programmers to be able to read, understand and tweak the code if necessary. This is why Python was chosen as the main language for this correlation program in spite of its relatively poor performances compared to lower-level languages. An other argument in favor of this language is that this software will be integrated to Crappy [26], a Python module developed internally to perform advanced mechanical tests. Only the critical parts were written in CUDA in a template file containing kernels (less than 100 lines excluding comments and blank lines). They will be compiled on-the-fly and used with the Python module PyCUDA [27], which allows most of the operations to be written in Python, even when manipulating data stored on the GPU memory.

The class *GPUCorrel* is the main class for this software. The *__init__* method of this class takes the resolution of the full-size image as its only positional argument. You can provide the fields as a keyword argument when instantiating the class. A field must be a Numpy array of dimension (y,x,2), where field[i,j] holds the displacement along respectively x and y for the pixel at coordinate y = i and x = j. For common fields, helpers are provided and the array can be replaced with a string. For example, one can use "x" for the rigid-body motion along x (i.e. f[:,:,0] = 1 and f[:,:,1] = 0).

Once the class is created and the fields are set, the reference image must be provided using the *set_ref()* method. The image resolution must match the value of img_shape given to *__init__* and the resolution of the fields. With the reference image set, one can call *prepare()* to set the textures and compute the $\{B\}$ matrix. This method computes all the intermediates table used for the computation. It must be called every time the fields or the reference image are changed. Note that if it is not explicitly called, it will be done automatically upon calling *compute()*.

It is now possible to compute the correlation using *compute()*. It takes the second image as argument and returns a 1D Numpy array with as many components as base fields.

The documentation of the GPUCorrel class is included in its docstring in *src/gpucorrel/gpucorrel.py*. A Markdown file *README.md* gives a quick overview of the software and the requirements. Lastly, a text file *LICENSE* contains the license under which this software is published. This file should be read and understood before using, modifying or distributing this software.

The class *GPUCorrel* creates the chosen number of instances of *Correl_level*, each one with a different resolution. When *GPUCorrel.compute()* is called, the given image is resampled to the smallest resolution and fed to the first level. The resulting vector is upscaled and fed to the next level, and so on all the way to the last level. The number of levels and the resampling factor can be chosen respectively by the corresponding keyword arguments at init.

The algorithm will have to read and perform many interpolation operations on the images. To accelerate this as much as possible, the code uses texture objects. They allow a slightly faster access to spatially close 2D data than GPU VRAM but more importantly, texture units perform bilinear interpolation in the time of a single memory read, where performing the bilinear interpolation manually would require four reads. To allow on-the-fly compilation with the PyCUDA module, this code uses bindless textures, which require a device with compute capability 5.0 (starting with Kepler architecture). There are three textures defined in the kernel file: the reference image, the deformed image and the mask. Each level creates these three at their working resolution. The *GPUCorrel* class also creates two textures for each field to accelerate the resampling for all the stages. This option allows a faster setting and updating of the fields, but uses more GPU memory. It is planned to make it optional in the future to allow the use of more fields when GPU Memory is limiting.

2.4. Sample code snippets analysis

The file *example.py* gives a quick insight on how to use this program. It will open a camera on the computer using OpenCV, use the first frame as a reference, continuously read images and perform the correlation between the reference and the new image. It will only compute the rigid body motion between the two frames: displacement along x and y and the rotation. Keep in mind that this method is not meant for large displacement. The best way to appreciate the results is to gently press the table/support on which the camera is resting. The slight movement induced, although invisible to the naked eye, is likely to be detected even by a cheap low-resolution camera.

Test images are provided with the file *test.py* to run test the program on reference images submitted to only rigid-body motion and no strain. Since the sample is not deformed, the measured strain is only composed of measurement errors. The measurement has a standard deviation below of 9.6 μ strain and extrema below 30 μ strain of error.

3. Illustrative example

To demonstrate the use of the presented program, a tensile test was performed on a cruciform Polymethylmetacrylate (PMMA) specimen shown in Fig. 3. The real-time DIC was used to drive the test based on the strain applied to the sample. The testing machine is designed to apply a load on two perpendicular directions. It was developed internally. It can apply loads up to 10kN using 4 stepper motors allowing tests on a wide variety of materials such as metal, composites and plastics. The machine is shown in Fig. 4.

Just like the presented software, the biaxial machine was designed with simplicity in mind: each arm is composed of a stepper motor, gears and a screw threaded in the jaw of the axis.

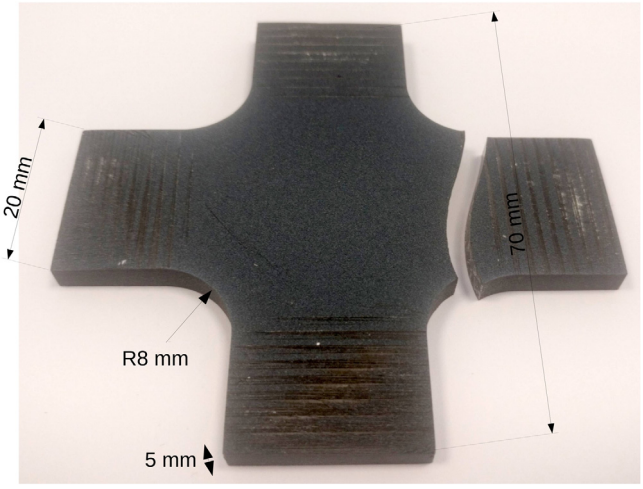

Fig. 3. PMMA specimen after the test.

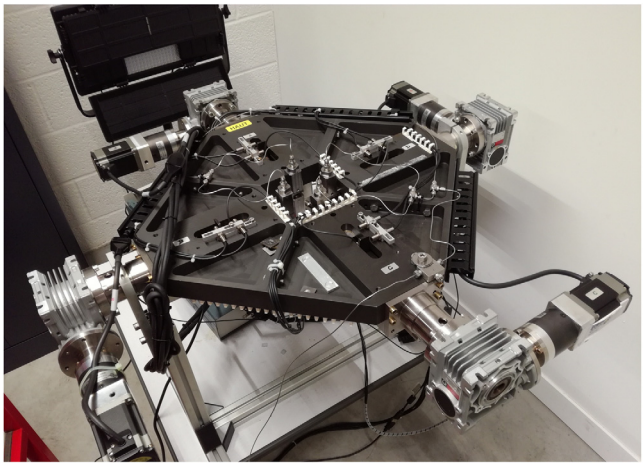

Fig. 4. Biaxial tensile machine.

This conception is easy to assemble and cost-effective, however the high-ratio of the gears induces a significant backlash in the mechanism. Furthermore, the bronze bearings of the axes of the machine have important clearances, inducing displacement in the directions perpendicular to the traction direction. As a consequence, there is no simple relationship between the position of the motor and the actual position of the jaw.

This would be a major issue when driving the test in an open-loop fashion, but real-time DIC allows to measure the actual displacement and strain of the cross-shaped sample in real time during the test. The command can then be adjusted to apply the chosen loading. The major advantage of this methodology is that the test can be driven on the actual state of the material and not based on the measurements of remote sensors. Indeed, the grips position is only an image of the displacement of the tip of the sample, even high-precision machines could miss on events such as specimen slipping out of the jaws or deformations of the parts of the machine. This is no longer an issue when controlling directly the state of the material.

The real time DIC is performed on a square zone of 12.5 $\times$ 12.5 mm at the center of the specimen. The image is a 2500 $\times$ 2500 pixels grayscale image with a depth of 8 bits. The base for the DIC is composed of 3 fields for the rigid body motions (along x, along y and the rotation), the strain along x and y and

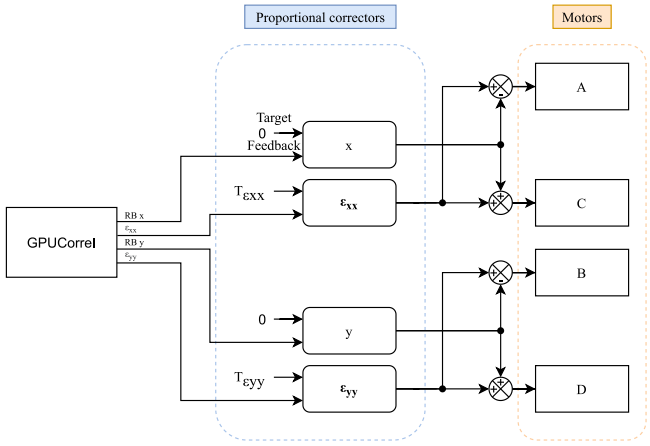

Fig. 5. Diagram presenting the software architecture to drive the test.

the shear strain for a total of 6 fields. The target framerate is 20fps and the DIC runs at this rate.

The test is driven using four proportional controllers. The actual position and strains of the sample are measured using the presented DIC algorithm. The two motors of each axis are regulated using two control loops: One of them controls the position of the sample by adjusting the sum of the speed of the two opposing motors. The second controls the applied strains by adjusting the difference of speed of the motors. This setup is applied on both X and Y axis, for a total of 4 controllers. The diagram on Fig. 5 presents the control loop used to drive the test.

The camera used in this test is a Ximea XiB CB500MG-CM, the lens is a Canon EF 100 mm USM. The side camera is a XiC MC124MG-SY with a Laowa 25 mm Ultra macro lens. The DIC, the control of the motors, the acquisition and saving of the images are handled by a single high-end computer. For this test, the specifications are deliberately above the requirements to make sure that all the tasks run exactly at the desired speed. It is equipped with an AMD Threadripper 1950X CPU, 64 GB of 3200 MHz DDR4 RAM and a GTX 1080Ti graphics card. One out of 20 images are written without any compression to a high-speed NVMe SSD.

The sample is a cruciform PMMA specimen, showing a brittle behavior with little viscosity. It is painted with a layer of black paint and a speckle pattern of white paint made using an airbrush. The broken specimen is shown in Fig. 3 and the image processed by the algorithm in Fig. 6. The path was chosen as follow: the strain amplitude, $\sqrt{\epsilon_{xx}^2 + \epsilon_{yy}^2}$, is growing at a constant speed of 0.02%/min and the angle of the command on the $(\epsilon_{xx}, \epsilon_{yy})$ plane varies at the speed of 18°/min. Since the testing machine is not designed to withstand compression, the command is restrained to the positive strain values. The command is represented as the orange line in Fig. 7 and the measured strain as blue dots. The time between two changes of direction is constant and equal to 5 min. The evolution of the strain versus time is shown in Fig. 8.

The position of the center region of the sample versus time is shown in Fig. 9. We can see that the closed loop control effectively kept the displacement of the sample below one pixel which corresponds in our case to 5 μm.

We can see that the strain is following the actual path but within an error that is not negligible. The 95th percentile error is 0.023% strain, meaning that 5% of the points are exceeding 0.023% of error. This is about one order of magnitude larger than expected. This error appears to be mainly originating from the control software: The proportional controllers are having difficulties when facing discontinuous events like the play shown on

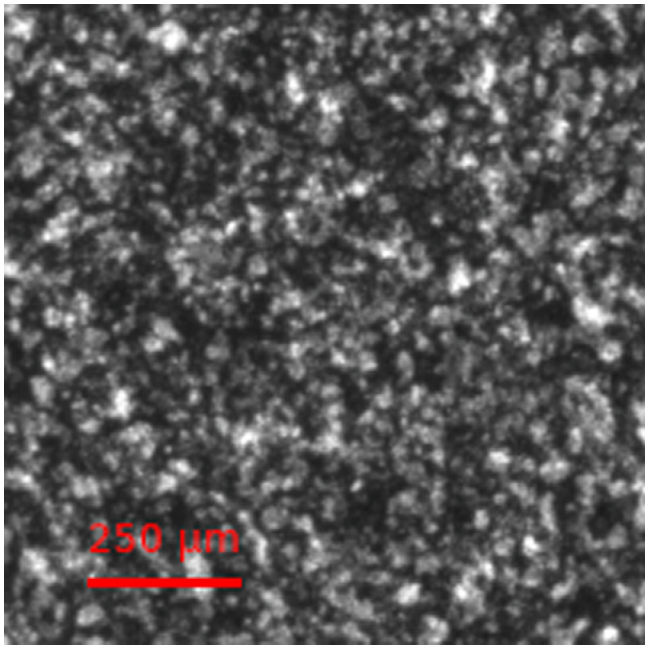

Fig. 6. Zoom on a small region (200 × 200 pixels) of the speckle pattern.

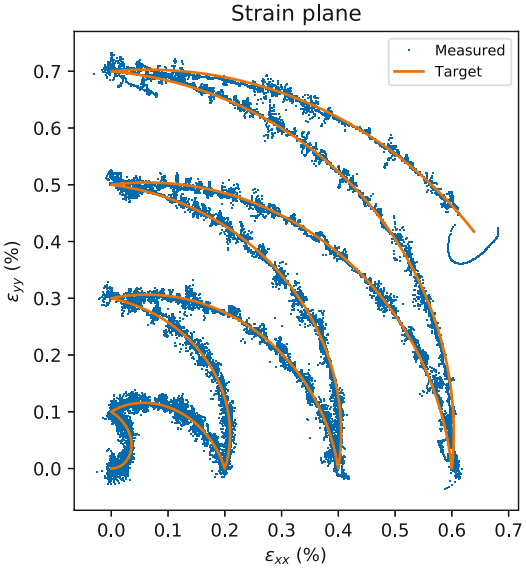

Fig. 7. Strain space explored during the test.

the thread and quantization of the command forced the motor to turn repeatedly on and off because the average speed is below the minimum speed of the motors. Also, the averaging of the measurements causes a delay, preventing a faster control loop.

The fields computed by *GPUCorrel* are not expected to pick up on local events but in the presented scenario, the fields are close to a homogeneous strain state. This allows a direct comparison with a local DIC algorithm. For this study, YaDICs [4] was used as it has been thoroughly tested in the domain of experimental mechanics. The same images were processed after the test in order to give a direct comparison. The image at $t = 1043$ s was chosen as it is almost a biaxial strain state and at a medium level of strain. The fields are shown on Fig. 10 and the residual on Fig. 11.

437 images were processed instead of all the 2193 to save computing time and disk space, they were sampled regularly

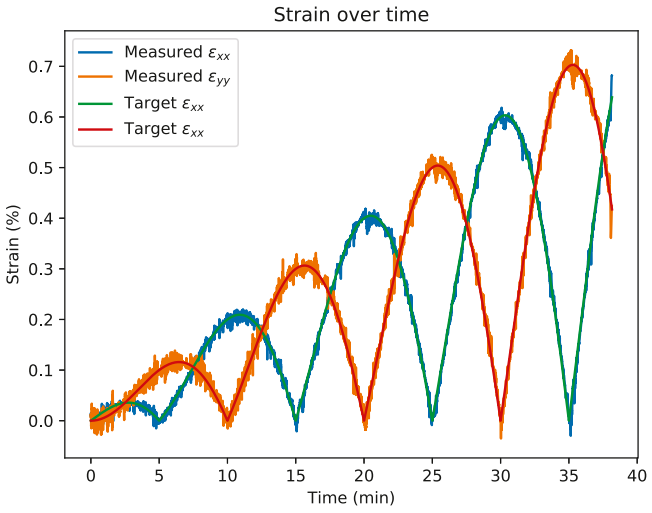

Fig. 8. Strains versus time.

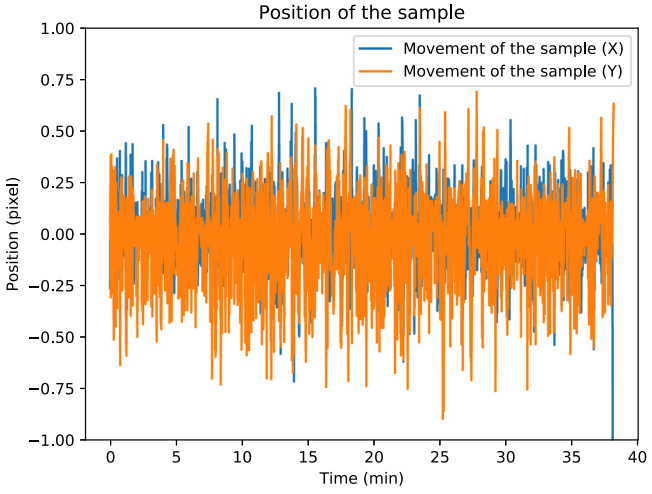

Fig. 9. Position of the specimen versus time.

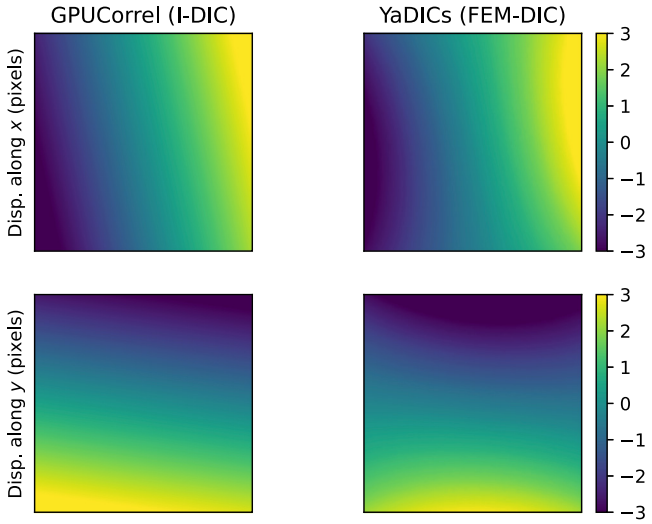

Fig. 10. Displacement fields computed by GPUCorrel (left) and YaDICs (right).

over the entire test. The YaDICs sequence follows a coarse-to-fine

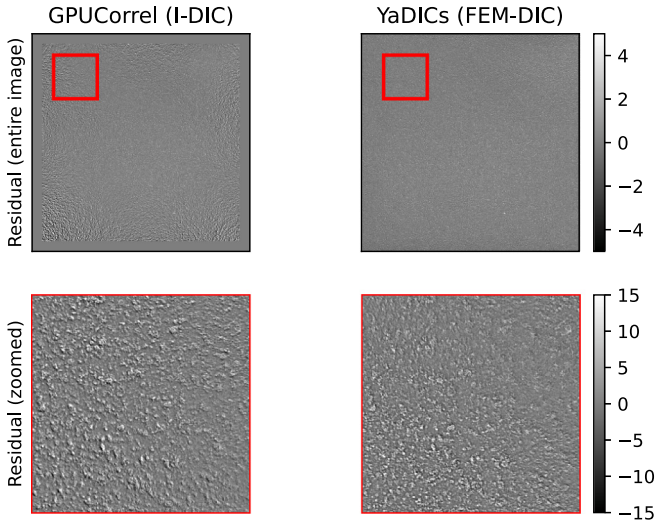

Fig. 11. Residual of the computation for GPUCorrel (left) and YaDICs (right) expressed in difference of digital level.

Table 1
Statistical indicators of the absolute difference between the projected YaDICs fields and GPUCorrel.

Field	Mean	STD	Median	MAD	95th percentile
x (pix)	1.945e−03	2.300e−03	1.489e−03	7.837e−04	4.940e−03
y (pix)	2.931e−03	2.862e−03	2.429e−03	1.288e−03	6.629e−03
r (°)	1.121e−04	1.158e−04	7.831e−05	4.913e−05	3.158e−04
ϵ_{xx} (%)	8.551e−04	4.641e−04	8.039e−04	3.745e−04	1.676e−03
ϵ_{yy} (%)	1.016e−03	5.366e−04	9.579e−04	3.627e−04	1.985e−03
ϵ_{xy} (%)	4.061e−04	3.344e−04	3.199e−04	2.194e−04	9.944e−04

approach starting at scale 3 (one eighth of the original resolution) and resampling with a factor two after each step until the correlation is performed on the full-resolution. A median filter is used with a width a 3 pixels. The mesh is a FEM-grid with 16 pixels of spacing along X and Y. The pixel-wise field is then computed and the projection on the 6 fields base (3 rigid body motion, strain along X and Y and shear strain) is performed to extract 6 values to be compared directly with the results from GPUCorrel on the same images.

The values obtained by projection of the fields from YaDICs are very close to the value from GPUCorrel. The Table 1 shows common statistical indicators of the difference between the two measurements. MAD refers to the mean of the absolute deviation. These indicators show that the results from GPUCorrel are almost identical to the projected local fields from YaDICs. This shows that the level of accuracy is more than what is required to drive a test in real time. This is illustrated by the error from the control loop, which is orders of magnitude larger than the error from the measurement.

4. Impact

The actual position of the motors is shown in Fig. 12. It is clear when comparing it with Fig. 8 that the relationship between the motors positions and the actual strain is not straightforward. Therefore performing this test without using a closed-loop control would have been impossible. This figure also shows oscillations originating from the controller, further demonstrating that the control loop was not optimally set up.

This test shows that real-time DIC can be used to control biaxial mechanical tests based on the average strain of a region of the sample and on the absolute position of the sample to keep

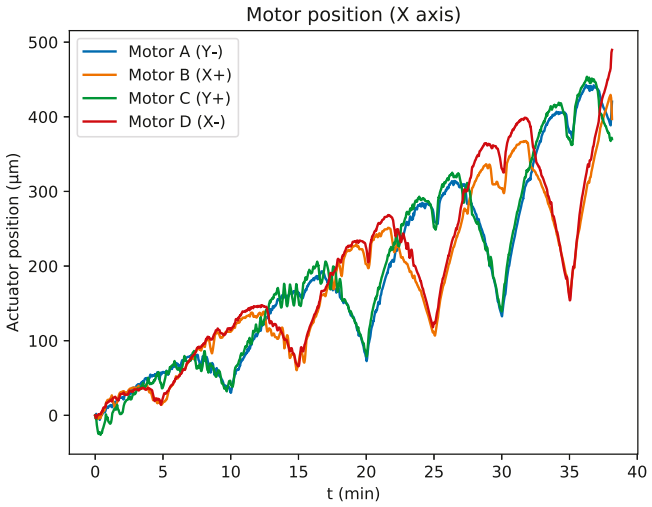

Fig. 12. Positions of the arms based on motors positions.

it centered and aligned. This demonstrates that real-time global DIC using chosen kinematic bases is possible and can be used to drive tests. The base used in this test is only composed of rigid body motion and homogeneous strain. It has the advantage of giving not only the global strain in all directions including shear strain but also the exact position of the sample relative to the camera, which could not have been measured using strain gauges. The next step is to use fields originating from simulations to measure quantities related to the inner state of the material that could not be directly extracted instantaneously during a test using conventional sensors.

5. Conclusions

This software allows the measurement of the strain of a sample without needing any sensors other than a camera. Unlike force or position sensors, the measurement is a direct image of the state of the material while remaining non intrusive. This measurement is fast enough to be operated in real time on a video feed. The residual can quantify the error and easily inform of correlation errors, making this software adequate to monitor and control mechanical tests.

This software will allow the development of a new generation of mechanical tests, making characterization faster and more accurate, leading to a better description of materials, ultimately leading to better sizing of mechanical parts.

One application of this software for experimental solid mechanics have been shown in this article, but DIC has many other applications in fluid dynamics where it is known as Particle Image Velocimetry (PIV), computer vision where it is known as image registration and many others. This software could be used for different purposes as it was developed with the intent to be generic and not meant specifically for solid mechanics. There could be room for improvements, for example local fields computation and dynamically updated list of fields. Any suggestion to improve this software will be happily considered.

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.

Acknowledgments

We would like to thank the “Région Hauts-de-France”, the “Agence de l’Inovation de Défense, France” and the “ELSAT2020” project for their financial and material support.

References

[1] Hart E. Theory of the tensile test 15 (2) 351–355. http://dx.doi.org/10.1016/0001-6160(67)90211-8. URL: https://linkinghub.elsevier.com/retrieve/pii/0001616067902118.

[2] Horn BKP, Schunck BG. Determining optical flow. URL: https://dspace.mit.edu/handle/1721.1/6337.

[3] Lautenschlager EP, Brittain JO. Constant true strain rate apparatus 39 (10) 1563–1565. http://dx.doi.org/10.1063/1.1683160. URL: http://aip.scitation.org/doi/10.1063/1.1683160.

[4] Dahdah N, Limodin N, Bartali AE, Witz JF, Seghir R, Charkaluk E, Buffiere JY. Damage investigation in a319 aluminium alloy by x-ray tomography and digital volume correlation during in situ high-temperature fatigue tests 52 (4) 324–335, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/str.12193. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/str.12193.

[5] Réthoré J. UFreckles, language: eng. http://dx.doi.org/10.5281/zenodo.1433776. URL: https://zenodo.org/record/1433776.

[6] Blaber J, Adair B, Antoniou A. Ncorr: Open-source 2d digital image correlation matlab software 55 (6) 1105–1122. http://dx.doi.org/10.1007/s11340-015-0009-1. URL: http://link.springer.com/10.1007/s11340-015-0009-1.

[7] Turner D, Crozier P, Reu P. Digital image correlation engine. URL: https://www.osti.gov/biblio/1245432-digital-image-correlation-engine.

[8] Belloni V, Ravanelli R, Nascetti A, Di Rita M, Mattei D, Crespi M. py2dic: A new free and open source software for displacement and strain measurements in the field of experimental mechanics 19 (18) 3832, number: 18 Multidisciplinary Digital Publishing Institute. http://dx.doi.org/10.3390/s19183832. URL: https://www.mdpi.com/1424-8220/19/18/3832.

[9] Passieux J-C. jcpassieux/pyxel, original-date: 2018-10-06T17:55:07Z. URL: https://github.com/jcpassieux/pyxel.

[10] André D. Damien andré/ pydic. URL: https://gitlab.com/damien.andre/pydic.

[11] Olufsen SN, Andersen ME, Fagerholt E. μ DIC: An open-source toolkit for digital image correlation 11 100391. http://dx.doi.org/10.1016/j.softx.2019.100391. URL: https://www.sciencedirect.com/science/article/pii/S2352711019301967.

[12] Das PP, Elenchezhian MRP, Vadlamudi V, Reifsnider K, Raihan R. RealPi2ddic: A low-cost and open-source approach to in situ 2d digital image correlation (DIC) applications 13 100645. DOI: http://dx.doi.org/10.1016/j.softx.2020.100645. URL: https://www.sciencedirect.com/science/article/pii/S2352711020303587.

[13] Miikki K, Karakoç A, Rafiee M, Lee DW, Vapaavuori J, Tersteegen J, Lemetti L, Paltakari J. An open-source camera system for experimental measurements 14 100688. http://dx.doi.org/10.1016/j.softx.2021.100688. URL: https://www.sciencedirect.com/science/article/pii/S2352711021000339.

[14] Thielicke W, Stamhuis E. PIVlab–Towards User-Friendly, Affordable and Accurate Digital Particle Image Velocimetry in MATLAB 2 (1), ISBN: 2049-9647, Ubiquity Press.

[15] Liberzon A, Lasagna D, Aubert M, Bachant P, Käufer T, jakirkham, Bauer A, Vodenicharski B, Dallas C, Borg J. OpenPIV/openpiv-python: OpenPIV - python (v0.22.2) with a new extended search PIV grid option. http://dx.doi.org/10.5281/zenodo.3930343. URL: https://zenodo.org/record/3930343.

[16] Genet M, Stoeck C, von Deuster C, Lee L, Kozerke S. Equilibrated warping: Finite element image registration with finite strain equilibrium gap regularization 50 1–22. http://dx.doi.org/10.1016/j.media.2018.07.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S1361841518305346.

[17] Klein S, Staring M, Murphy K, Viergever MA, Pluim JPW. elastix: A toolbox for intensity-based medical image registration 29 (1) 196–205, IEEE Transactions on Medical Imaging http://dx.doi.org/10.1109/TMI.2009.2035616.

[18] Kroeger T, Timofte R, Dai D, Van Gool L. Fast optical flow using dense inverse search. In: Leibe B, Matas J, Sebe N, Welling M, editors. Computer Vision – ECCV 2016, vol. 9908. Springer International Publishing; 2016, p. 471–88. http://dx.doi.org/10.1007/978-3-319-46493-0-29, URL: http://link.springer.com/10.1007/978-3-319-46493-0_29.

[19] Plyer A, Le Besnerais G, Champagnat F. Massively parallel lucas kanade optical flow for real-time video processing applications 11 1–18. http://dx.doi.org/10.1007/s11554-014-0423-0.

[20] Hild F, Roux S. Digital image correlation: From displacement measurement to identification of elastic properties - a review 42. http://dx.doi.org/10.1111/j.1475-1305.2006.00258.x.

[21] Sun Y, Pang JHL, Wong CK, Su F. Finite element formulation for a digital image correlation method 44 (34) 7357-7363. http://dx.doi.org/10.1364/AO.44.007357. URL: https://www.osapublishing.org/ao/abstract.cfm?uri=ao-44-34-7357.

[22] Passieux J-C, Périé J-N. High resolution digital image correlation using proper generalized decomposition: PGD-DIC 92 (6) 531–550. http://dx.doi.org/10.1002/nme.4349.

[23] Neggers J, Blaysat B, Hoefnagels JPM, Geers MGD. On image gradients in digital image correlation: On image gradients in digital image correlation 105 (4) 243–260. http://dx.doi.org/10.1002/nme.4971. URL: https://onlinelibrary.wiley.com/doi/10.1002/nme.4971.

[24] Lucas BD, Kanade T. An iterative image registration technique with an application to stereo vision, in: IJCAI81, 674–679. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?.

[25] Black MJ, Anandan P. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields 63 (1) 75–104. http://dx.doi.org/10.1006/cviu.1996.0006. URL: http://www.sciencedirect.com/science/article/pii/S1077314296900065.

[26] Couty V. CRAPPY : Un module de pilotage et d'acquisition pour l'expérimental, PyConFR 2018. URL: https://youtu.be/KKvJFfN-3Lc.

[27] Klöckner A, Pinto N, Lee Y, Catanzaro B, Ivanov P, Fasih A. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation 38 (3) 157–174. http://dx.doi.org/10.1016/j.parco.2011.09.001. URL: http://www.sciencedirect.com/science/article/pii/S0167819111001281.