EN2210 Continuum Mechanics

Project

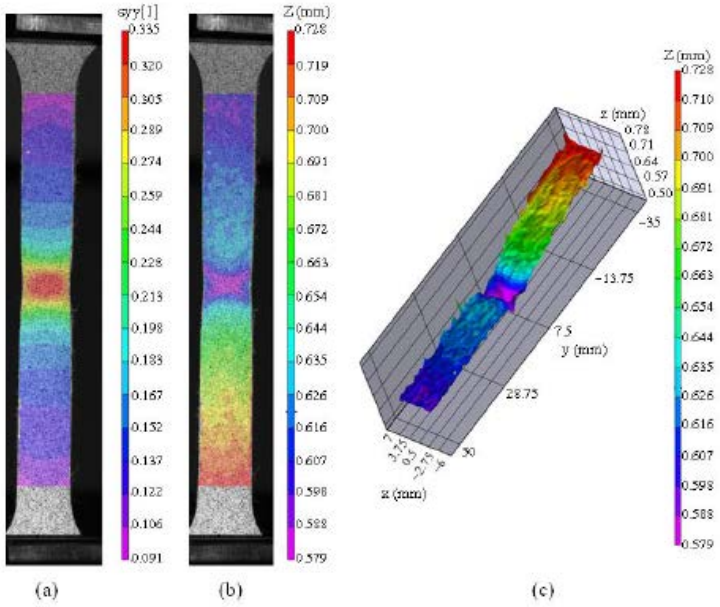
Computing Deformation Measures from Digital Image Correlation Data

Synopsis

You will write a MATLAB code to compute measures of deformation and motion from 2D digital image correlation measures.

1. Introduction

Digital image correlation is a powerful and comparatively inexpensive technique for measuring deformation fields in materials and structures with micron scale resolution. The basic principle is to take a series pictures of a speckle pattern on a surface during deformation, and to determine displacement fields by matching similar regions in two successive images. An example of displacement contours extracted from DIC measurements on the surface of a tensile specimen of steel [1] is shown in the figure.



In the example shown, only the surface of the specimen can be observed. In transparent materials, it is also possible to the full 3D displacement field in a specimen. For example, Professor Franck's lab uses 3D DIC measurements to measure forces exerted by cells on a deformable substrate.

You can download free 2D DIC codes from the web: two examples are Matlab codes written by Elizabeth Jones based on an earlier set of codes by Christopher Eberl. There are also a number of expensive commercial DIC codes on the market.

In this project you will use the image processing toolbox in MATLAB to determine a 2D displacement field from a set of images, and then process this data to calculate and plot 2D measures of deformation.

2. Project Deliverables:

Your mission is to provide:

1. A written report that describes what your code can do, with instructions for its use, and provides some examples of its performance. The examples could include a validation showing that the code gives correct results for a known strain field; and then some examples of fields
2. A MATLAB code that will read the data files provided and plot strain and strain rate measures of interest.

The due date for both is **Wednesday Oct 10.**

3. Data files

You can download some sample images (from the ncorr website) to test your code from the course website. These include two simple examples:

- translation_data.mat : a sample subjected to a small rigid translation
- rotation_data.mat: a sample subjected to a small rigid rotation

To load this data into MATLAB and show the reference and current images use (eg)

```
load('rotation_data.mat')
imshow(ref);
imshow(cur);
```

In addition, there is a more fancy dataset:

- A set of DIC images (tif format) of a plate with a central hole. You can read the Tiff files of the plate with a hole into Matlab using a number of different methods. One way is

```
filedirectory = 'Hole_Plate_Images';
filename = 'ohtcfrp_';
ref = read(Tiff(strcat(filedirectory,'/',filename,'00','.tif')));
ref = double(ref)/256.;
cur = read(Tiff(strcat(filedirectory,'/',filename,'11','.tif')));
cur = double(cur)/256.;
```

  (This reads the zeroth image as the 'reference' and the 11th as the 'current' image)

4. Calculating Displacements using MATLAB image processing toolbox.

Matlab has an extensive image processing toolbox, which includes some built-in functions to perform image correlation. A little background on MATLAB images is helpful:

- The images are stored as two-dimensional arrays of gray-scale values: for example, the image (from rotation_data.mat) shown in Fig 2 is stored in a 375x375 dimensional array called 'cur'. The grayscale value of the pixel at the top left-hand corner is stored in cur(1,1); the bottom left is at cur(375,1); the top right is at cur(1,375).
- You can crop out a sub-region of an image using the syntax

```
cursubimage = cur(vlo:vhi,hlo:hhi);
```

  (here, vlo, vhi, etc are integers - the 'v' and 'h' in the indices stand for vertical and horizontal) You can test this on the 'rotation_data.mat' file if you like: you can use imshow(cursubimage) to display the cropped image.

To calculate displacement and strain fields, you will always load *two* images. The 'reference' image will be the specimen before deformation, and the 'current' will be after deformation. If there are many images, it is usually best to process two consecutive images, and then combine them to track the motion of points of interest from the first image to the last.

You can use the MATLAB 'normxcorr2()' function to calculate an initial approximation to the displacement field (to within +/- 1 or 2 pixels). This calculates the normalized cross-correlation between two matrices, and can be used to determine the location of a small sub-image within a larger image, as described in the Matlab manual. The displacements can then be corrected using the 'cpcorr()' function, which allows you to determine displacements with sub-pixel resolution.

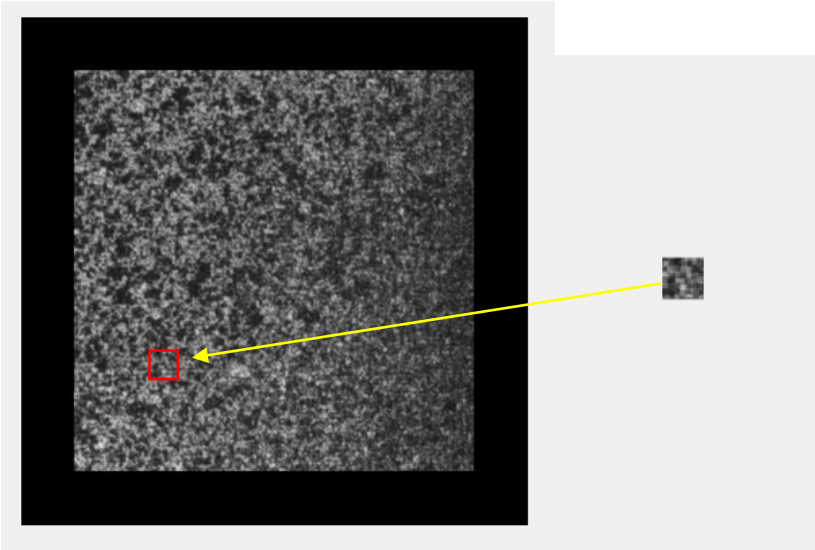


Fig 2: Locating a cropped image from the surface of a deformed specimen in an image of the undeformed specimen

Before writing code to calculate displacements, it is instructive to test the normxcorr2() function by writing a short script that will execute the following operations:

1. Load the rotation_data.mat files
2. Pick a point of interest in the reference image (ref), eg (100,200), and crop a small rectangular region (say 10x10 pixels or so – you can experiment with this size to see what gives the best results) surrounding the point of interest
3. Follow the procedure described in the Matlab manual to use normxcorr2 to locate the coordinates of the top left corner of the cropped region inside the reference image.
4. Superimpose a rectangle on the reference image to identify the region of interest. You can do this with

```
rectangle = [htopleft,vtopleft,width,height];
ref = insertShape(ref,'Rectangle',rectangle,'LineWidth',2,'Color',[1,0,0]);
```

Here 'htopleft' and 'vtopleft' are horizontal and vertical pixel cords of the top left hand corner of the rectangle, and width, height are the width and height (in pixels) of the cropped image.

Check your code by making sure the cropped image matches the region inside the rectangle (your brain is even better at image processing than MATLAB and will be able to see that the patterns match).

You can easily extend these steps to calculate displacement vectors (in pixel coordinates) that map regions from the reference image to the current image. You can compute the displacement vectors

corresponding to a point of interest in the reference image (this would give a Lagrangian description of the displacements) by:

1. Crop a small region surrounding the point in the reference image
2. Crop a larger region around the same point in the current image (you want the region in the reference image to be large enough to contain the region before deformation. Of course, the disadvantage of making the cropped reference image too large is that your code will run very slowly)
3. Locate the coordinates of the top left corner of the cropped current image inside the cropped reference image, as described above.
4. Knowing the sizes of the two images and the coordinates of the corner found in step 3, you can now easily calculate the displacement vector from the center of the reference image to the center of the current image.
5. To check your code, make the current image the same as the reference image – the displacements should then be zero.

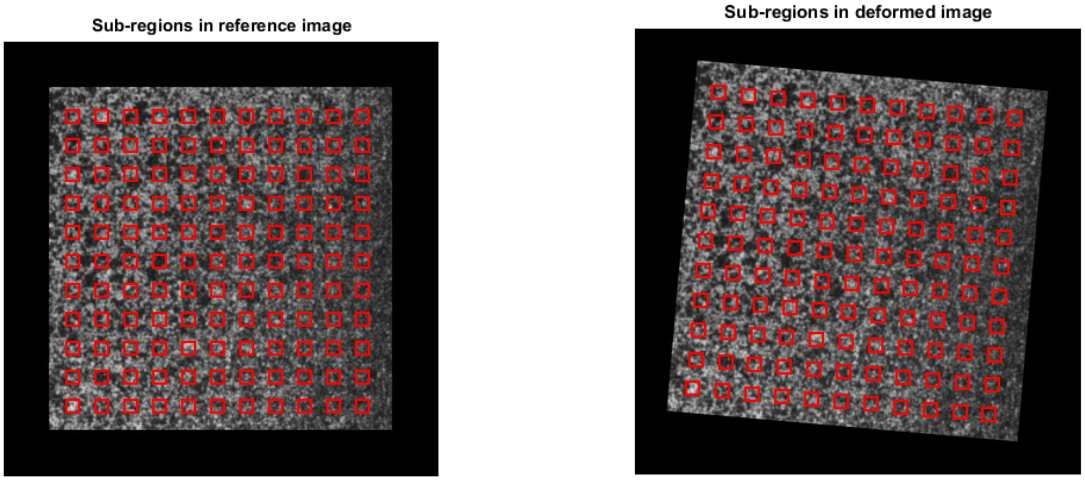


Fig. 3 Illustrating the DIC procedure: (a) Isolate sub-regions in the reference image; (b) Search the deformed solid for regions that best match each square in the reference image. The centers of the squares in the two images define a set of points; the difference between the centers before and after deformation gives the displacement field.

Once you have been able to calculate the displacement of a point of interest, you can extend your code to track the motion of a grid. You can put a uniform grid on the undeformed specimen (to get a Lagrangian description of deformation) or on the deformed specimen (Eulerian). Non-uniform grids can also be used.

To check your computations, try plotting figures that resemble those shown in Figs 3 -5: these illustrate the displacement field in a solid of interest by plotting

- The position of a grid on its surface before and after deformation.
- Contours of horizontal and vertical displacement

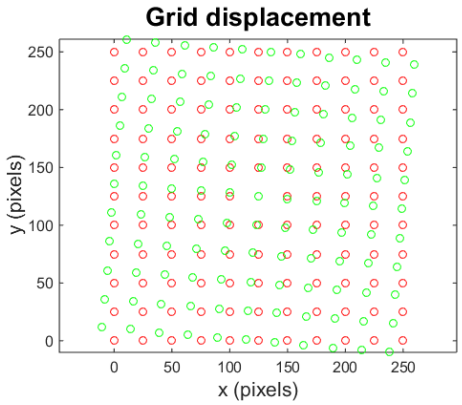


Fig 4: Grid of points before (red) and after (green) deformation

Of course, the MATLAB ‘plot’ and ‘contour’ functions put the origin at the bottom left of the graph, so it is helpful to change the coordinate system to make the displays consistent with what is on the images.

Once you have the code working for a pair of images, you can add steps to process a sequence of images. For this case it is usually simplest to define the grid on the first image, and then track the points on this grid as they move from one image to the next. It is best to correlate two successive images, rather than correlate each image with the first one, however.

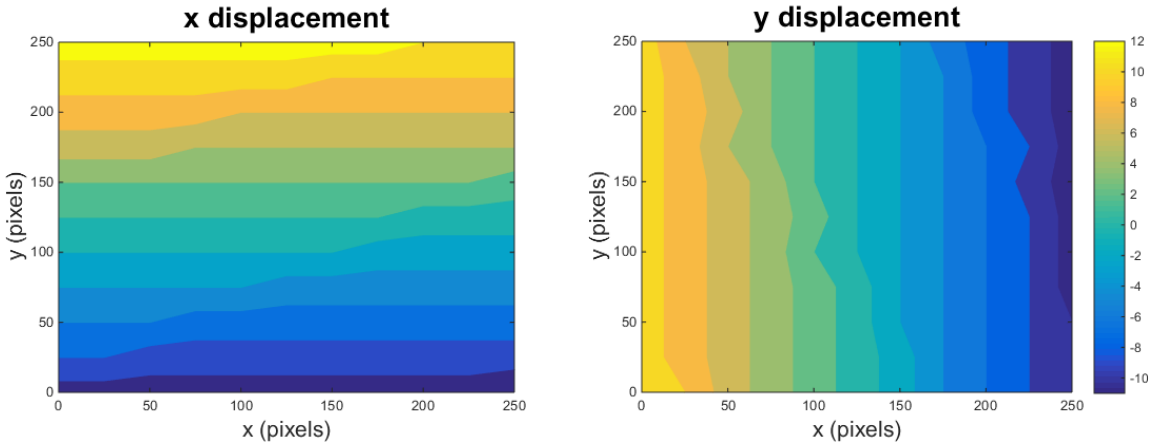


Fig 5: Contours of horizontal (x) and vertical (y) displacement (in pixels), as determined from the normxcorr2 function.

If you have been able to reproduce the plots in Figs 3-5, you will have a code that can generate the coordinates of a set of points in the reference and deformed images. You can improve the resolution of the image correlation process by using the MATLAB ‘cpcorr’ function to adjust the positions of the points in the deformed image slightly. This will give you displacement fields with a resolution of about 0.1 pixels. See the Matlab manual for details. As an example, Fig 6 illustrates the displacements on the rotated sample after processing with cpcorr.

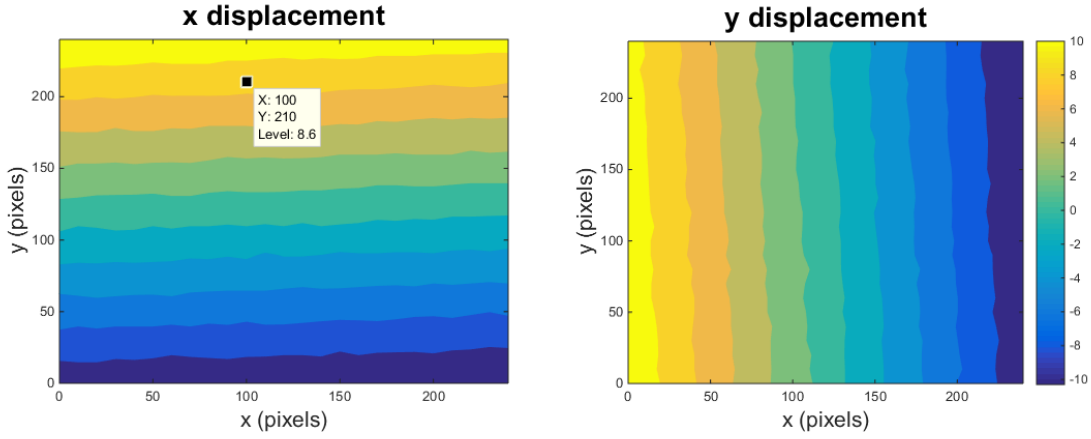


Fig 5: Contours of horizontal (x) and vertical (y) displacement (in pixels), after correction with the cpcorr function.

Finally, you could smooth the displacement fields. For a rectangular grid, you could use this open source matrix smoothing code.

You can test your code by applying a known transformation to a reference image. The following MATLAB functions are useful:

- B = imrotate(A,angle) rotates the image A through angle (in degrees) counterclockwise
- B = imresize(A,size) scales the image by a factor 'size'. Note that this increases the number of pixels in the image, so you will need to crop the new image to use the cpcorr function
- B = imtranslate(A,translation_vector) translates an image
- B = imwarp(A,tform) applies a geometric transformation (which can include stretching) to an image. The 'tform' variable is a MATLAB 'geometric transformation' object that must be defined using tform = affine2d(T), where T is a transformation matrix. See the Matlab manual for details. For example, you can apply a homogeneous deformation to an image using

```
tform = affine2d([Fxx Fxy 0; Fyx Fyy 0; 0 0 1])
cur = imwarp(ref,tform);
```

  Fxx, Fyy,Fxy,Fyx are the components of the deformation gradient. Your code should recover the deformation gradient correctly, as well as deformation measures derived from the deformation gradient.

Rectangular grids don't work well for solids with complicated shapes. If you are an experienced coder or matlab user you might like to write a more sophisticated code that can handle unstructured grids. Here are some suggestions for using unstructured grids:

1. For the images provided, the background is always black. This means that you can easily generate an unstructured grid on the surface by covering the solid with a rectangular grid, and then discarding any points with intensity less than a threshold (eg ref(i,j)<0.1).
2. You can calculate displacements at each point in an unstructured grid using the same procedure as for a structured grid.
3. You can plot contours on an unstructured grid by (i) triangulating the grid using the Matlab delaunay() function (this connects all the points with a set of triangles) (ii) Removing any triangles whose centroids lie outside the specimen (check the image intensity at the center of each triangle, and remove it from the), then (iii) using the open-source 'tricontf' function to plot the contours.
4. You can smooth data on an unstructured grid using a version of 'Laplacian smoothing' – in this procedure, the value of a function at each point in the grid is replaced by the average of all the other points in the grid that share an edge of a triangle with the point of interest. For example, in the grid shown in Fig 6, the value at vertex 9 would be computed as the average of grid points 2,3,4,11 and 10. You need some trickery to make this process effective: for example it usually gives bad results for grid points on the edge of the region, such as 1,2,3 etc. These can be omitted from the smoothing (fairly easy), or for something a bit more sophisticated use only line smoothing around the edge, i.e. replace the value at grid point 1 by the average of 8 and 2 (a bit harder).

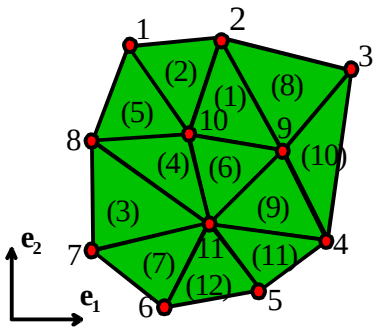


Fig 6: A simple unstructured grid

5. Calculating strain measures

Your next mission is to compute, and plot, deformation measures in the specimen. You can choose what deformation measures you would like to compute – but you will most likely want to start by calculating the deformation gradient or displacement gradient.

To do this, you will have to interpolate the displacement data generated in part 4. There are several different ways you could do this, which are described briefly in the sections to follow.

5.1 Using the Matlab gradient function

If you calculate displacements at a set of points on a rectangular grid, you can use the Matlab 'gradient' function to calculate the displacement gradient, eg

```
[uxx,uxy] = gradient(hdisp,hspacing,vspacing);
```

Here, hdisp is a matrix of horizontal displacements at each grid point (the points have to be ordered with hdisp(1,1) at the bottom left corner, and with row and col in hdisp(row,col) specifying y and x position in the grid, respectively)

4.2 Computing displacement gradients using finite-element interpolation

If you would like to use an unstructured grid (harder) you can compute gradients using finite element interpolation functions. In this approach, sets of grid points are taken to lie at the corners of triangular element, as shown in Fig 6. The displacement field inside each triangle is interpolated between the values at the corners, which then allows the gradients to be computed.

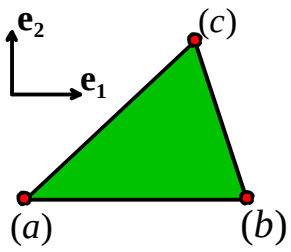


Fig 6: Triangular element

Calculating displacement gradients in triangular elements is particularly simple. For this case, the displacements can be interpolated between the values at the three corners. Consider a triangular element, with nodes *a*, *b*, *c* at its corners. Let $x_i^{(a)}$, $x_i^{(b)}$, $x_i^{(c)}$ denote the coordinates of the corners, and define

$$N_a(x_1,x_2)=\frac{\left(x_2-x_2^{(b)}\right)\left(x_1^{(c)}-x_1^{(b)}\right)-\left(x_1-x_1^{(b)}\right)\left(x_2^{(c)}-x_2^{(b)}\right)}{\left(x_2^{(a)}-x_2^{(b)}\right)\left(x_1^{(c)}-x_1^{(b)}\right)-\left(x_1^{(a)}-x_1^{(b)}\right)\left(x_2^{(c)}-x_2^{(b)}\right)}$$

$$N_b(x_1,x_2)=\frac{\left(x_2-x_2^{(c)}\right)\left(x_1^{(a)}-x_1^{(c)}\right)-\left(x_1-x_1^{(c)}\right)\left(x_2^{(a)}-x_2^{(c)}\right)}{\left(x_2^{(b)}-x_2^{(c)}\right)\left(x_1^{(a)}-x_1^{(c)}\right)-\left(x_1^{(b)}-x_1^{(c)}\right)\left(x_2^{(a)}-x_2^{(c)}\right)}$$

$$N_c(x_1,x_2)=\frac{\left(x_2-x_2^{(a)}\right)\left(x_1^{(b)}-x_1^{(a)}\right)-\left(x_1-x_1^{(a)}\right)\left(x_2^{(b)}-x_2^{(a)}\right)}{\left(x_2^{(c)}-x_2^{(a)}\right)\left(x_1^{(b)}-x_1^{(a)}\right)-\left(x_1^{(c)}-x_1^{(a)}\right)\left(x_2^{(b)}-x_2^{(a)}\right)}$$

The displacement at an arbitrary point in the triangle can then be expressed as

$$u_i(x_1,x_2)=u_i^{(a)}N_a(x_1,x_2)+u_i^{(b)}N_b(x_1,x_2)+u_i^{(c)}N_c(x_1,x_2)$$

You can differentiate these to calculate the displacement gradient. Note that the displacement gradients are constant inside each triangle.

This procedure gives gradients inside the triangles, but what you really want is values for the gradients at the grid points. Again, there are various ways to find these. The simplest procedure is to average the values inside all the triangles connected to each grid point. For example, for the grid in Fig 6, the value at grid point 6 would be the average of the values in the two triangles (7) and (12). The value at grid point 12 would be the average of triangles (1,2,4,5,6). Once you have grid point data, you can smooth the field using Laplace smoothing.

5. Data processing and plotting

Once you have computed displacement gradients, it is straightforward to compute other quantities that characterize deformations. Examples might include

- The Jacobian $J = \det(\mathbf{F})$
- Left and right stretch tensors, and/or their principal values and directions (you could display these using a quiver plot); strain invariants
- The rotation tensor, and its axis/angle (Rodriguez representation) (can you calculate the misorientation between the initial and rotated specimens in Fig 3?)
- The velocity gradient; the stretch rate; the spin tensor, the vorticity vector (or its magnitude and direction) (you will have to make up a time interval between the images, of course).

You can test the parts of your code that compute strains by creating a grid of points, and assigning each point a displacement corresponding to a uniform deformation gradient. Your code should return the correct values for deformation gradient at each grid point. You can also apply affine transformations to an image – your code should be able to work back and calculate the transformation (plus some noise).

As a demonstration, the figures below show the unstructured correlation grid before and after deformation; and contours of vertical displacement (pixels) and vertical Lagrange strain for a pair of images for the hole-in-a-plate example.

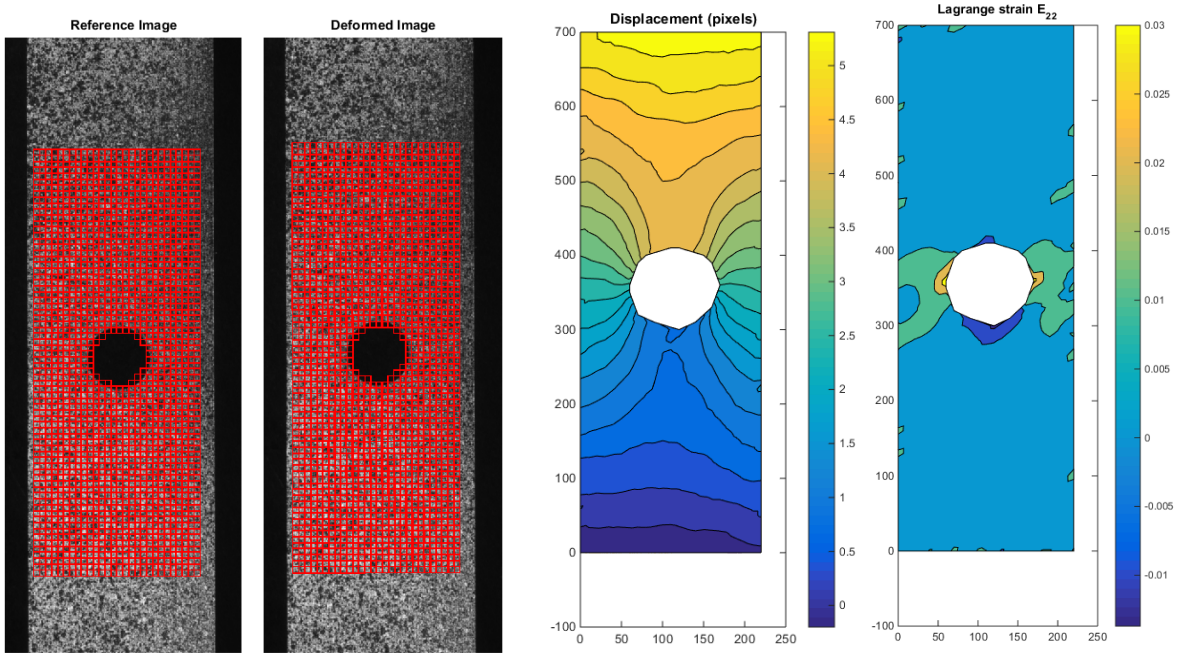


Fig 8: Examples of displacements and strains on the surface of a strained specimen using an unstructured grid.

6. References

1. Jason Coryell, Josh Campbell, Vesna Savic, John Bradley, Sushil Mishra, Shashank Tiwari, Louis Hector, Jr. "Tensile deformation of quenched and partitioned steel - a third generation high strength steel," Supplemental Proceedings, Vol **2**, Materials Properties, Characterization and Modeling, 555-562 2012

2. Franck, C., Hong, S., Maskarinec, S.A., Tirrell, D.A., Ravichandran, G., "Three-Dimensional Full-Field Measurements of Large Deformations in Soft Materials using Confocal Microscopy and Digital Volume Correlation," Experimental Mechanics, **47**, 427-438, 2007.

Appendix 1: Grading Rubric

1. Report: 10 points.
    - Report describes an extensive series of tests to verify that a substantial set of deformation measures are calculated correctly (including comparisons with hand-calculations) (10 points)
    - Good report describing a range of tests 8 points
    - Good report, minor errors or omissions
    - Adequate report, but difficult to follow; incomplete 6 points
    - Major errors or omissions 4 points
2. Matlab Code 15 points
    - Correct, well commented, user-friendly code with extensive capabilities; capable of calculating deformation measures using both structured and unstructured grids 15 points
    - Good, correct, well commented and user friendly MATLAB capable of handling regular grids 12 points
    - Functioning code, but difficult to read or use, or limited capabilities for data processing or visualization 9
    - Major errors, or only minimal capabilities 6