UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology

Siim Sundla

SEMI-AUTOMATIC DEFLECTION MEASUREMENT USING DIGITAL IMAGE CORRELATION

Master's thesis (30 EAP)

Supervisors:

Assoc. Prof. Gholamreza Anbarjafari

Dr. Andres Punning

Tartu 2015

Abstract

In this thesis, an easy-to-use approach using cubic spline interpolation is proposed for providing initial estimations for digital image correlation (DIC) grid point locations and rotations on objects with major deflection. One of the main problems in using DIC for deflection measurements is that direct correlation fails since image sample rotations can differ considerably across two images. The proposed method uses manually chosen reference points to create a spline that describes deflection. Curve geometries are then taken into account when calculating initial guesses for DIC grid points. The validity and accuracy is demonstrated through a series of tests.

Table of contents

Introduction

Digital image correlation (DIC) is commonly used for deformation measurements, however it is not directly suitable for deflection measurements. This is due to the fact that DIC does not handle larger image rotations well. There exists several methods for compensating this shortcoming, nevertheless it can be difficult to implement them in practice.

The aim of this thesis is to propose an easy-to-use method for providing initial guesses for locations and rotations of DIC grid points on deflected objects. DIC calculations can then in turn serve as basis for research related to specimen deflection.

The paper is divided into three sections. The first chapter gives a historical and mathematical background to DIC and its related algorithms, including their use and limitations. The second chapter specifies the problem to be solved and gives a theoretical overview of the proposed solution. In the final chapter, an implementation is described and evaluated through a series of tests.

1. Digital image correlation

1.1 History and usage

Digital image correlation (DIC) is a widely used image processing algorithm for full-field displacement measurements. The same technique has also been referred to as digital speckle correlation method (DSCM), texture correlation, computer-aided speckle interferometry (CASI) and electronic speckle photography (ESP) [1].

DIC searches for similarities between pixel intensities in two images. Essentially, this involves calculating correlation criteria between the template image and each coordinate in the target image. The simplicity and low cost makes DIC a desirable option for many applications, since it does not require any special equipment other than a camera. Besides conventional cameras, DIC can be used with an optical microscope, laser scanning confocal microscope (LSCM), scanning electron microscope (SEM), atomic force microscope (ATM) and scanning tunneling microscope (STM) to perform micro and nanoscale deformation measurements [1].

Some of the first works in the field of image correlation can be attributed to Gilbert Hobrough. In 1950s Hobrough compared analogue representations of photographs in order to register features from various views. An instrument he designed in 1961 [2] is considered one of the first attempts to extract positional information using a form of DIC. It is believed that Peters and Ranson first proposed the use of computer based DIC-like method to conduct deformation measurements in 1982. However, the original concept involved digitally recording a full-field pattern by using ultrasonic waves. Based on this approach Sutton in 1983 developed numerical algorithms, performed experiments using optically recorded images and showed the feasibility of this approach now known as two-dimensional (2D) DIC [3].

Besides being used in fracture mechanics researchers have used DIC to understand material deformation behaviours on metals, plastics, wood, ceramics, tensile loading of paper and to study damage in composites and concrete. In 1984 Peters demonstrated that the approach can also be used in fluid systems. Material characterization studies have used DIC measurements in films, polymers, metals, heterogeneous materials, wood, bio-materials including skin, shape memory alloys, composites, asphalt, ceramics, glass wool, mineral wool, rock, glass, foams, clay, sands, soils, concrete, paint, electronic components, joints, etc. [3, 4]. Various mechanical properties can be calculated using DIC including: Young's modulus, Poisson's ratio, stress intensity factor, residual stress, thermal expansion coefficient, etc. [1].

DIC has also found use for bridge deflection measurements and for special effects in cinema [5]. Most notably in 1994 movie *Forrest Gump* where various moving elements were replaced and an actor was added to historical video sequences. The basis for these effects required tracking manually the selected features across frames which was done using DIC [6].

1.2 Mathematical definition

DIC works by evaluating similarities between pixel intensities on an image and a template as shown on Figure 1.1 (a) and (b) respectively. Two different approaches are commonly used for achieving this: cross-correlation and least squares matching (LSM)[7].

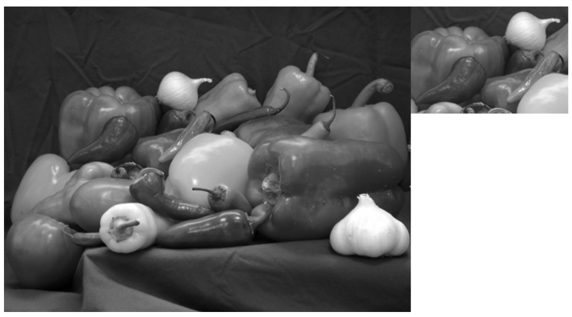


Figure 1.1: Image (a) and template (b) [8].

For cross-correlation a template is shifted over an image and correlation coefficient is calculated for each coordinate that describes how well the overlapping areas match. Figure 1.2 shows a correlation coefficient values calculated for an image and a template seen in Figure 1.1. The peak is at a location where the template originates from. Correlation coefficient shows the certainty of the match.

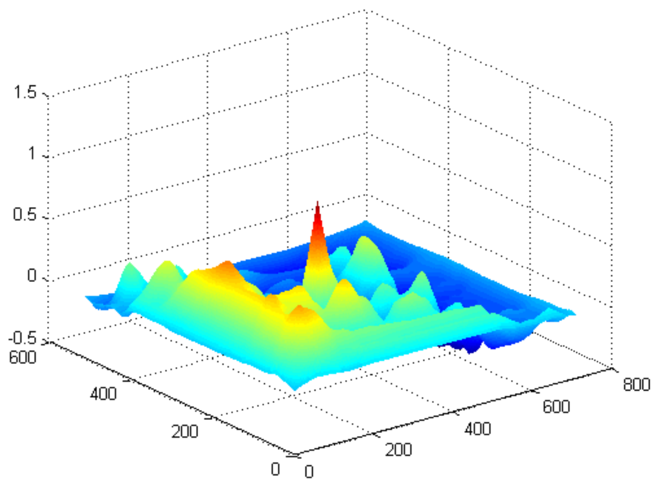


Figure 1.2: Correlation coefficient across image [8].

One of the most used correlation criteria is the normalized cross correlation (NCC) which is defined by:

$$c(u,v)=\frac{\sum_{x,y}[f(x,y)-\bar{f}_{u,v}][t(x-u,y-v)-\bar{t}]}{\sqrt{\sum_{x,y}[f(x,y)-\bar{f}_{u,v}]^2\sum_{x,y}[t(x-u,y-v)-\bar{t}]^2}} \tag{1.1}$$

where

- t is the template;
- $\bar{t}$ is the mean of the template;
- f is the target image;
- $\bar{f}_{u,v}$ is the mean of target image in the region under the template;

Other correlation criteria being used are the cross-correlation (CC) and the zero-normalized cross-correlation (ZNCC). CC in sensitive to both offsets and linear scaling of pixel intensities. NCC can compensate for linear scaling. ZNCC, although more complex, is insensitive to both, making it the most robust choice [1].

The LSM approach tries to minimize the squared differences between the pixels on the template and an image. The simplest LSM criteria is sum of squared differences (SSD) which can be written for a template with size *2Mx2M* as:

$$C=\sum_{i=-M}^{M}\sum_{j=-M}^{M}[f(x_i,y_i)-t(x'_i,y'_i)]^2 \tag{1}$$

Analogously to the cross-correlation criteria, the normalized (NSSD) and zero-normalized (ZNSSD) criteria exist which have the same advantages over SSD as their cross-correlation counterparts [1].

High accuracy of 1/100th of a pixel have been achieved on high quality images using this method. However NCC has been shown to be more accurate when lower quality images are involved. To be able to fully utilize LSM a high quality equipment is required, also colour cameras are not recommended since various post- and preprocessing algorithms can alter the results [7]. An example of LSM is given in section 1.4.2, that describes the gradient-based method for sub-pixel optimization.

Various techniques have been developed to improve the integer displacement search including frequency domain correlation using FFT, a nested searching scheme and sum-table approach [9].

1.3 Speed and accuracy

Accuracy and precision of DIC algorithms can depend on several factors:

- speckle pattern
- subset size
- correlation criterion
- shape function
- sub-pixel interpolation scheme
- sub-pixel registration algorithm

Iterative cross-correlation algorithm based on Newton-Rhapson (NR) method with zero-mean normalized cross-correlation (ZNCC) criterion and bicubic interpolation is considered a standard approach for sub-pixel accurate DIC [9].

Many computer simulations have shown that the DIC algorithm can easily yield accuracy better than 0.001 pixels at a speed faster than 5000 points per second. However, the entire DIC analysis is relatively slow, especially when advanced schemes such as reliability-guided correlation scanning strategy [10].

It has been shown that DIC processing speed can be improved by using Fourier domain correlation. This method is also known as digital speckle displacement measurement (DSDM). However DSDM cannot handle large rotations and deformations well. The accuracy of DIC in spectral domain has shown to be as good as in spatial domain only in some limited cases, due to lack of sub-pixel accuracy [10, 11].

1.4 Sub-pixel tracking

1.4.1 Newton-Rhapson method

Generally, either Newton-Rhapson or Levenberg–Marquardt iterative algorithm is used to match subsets to correlation criterion. These methods require an interpolation process to converge to a sub-pixel accurate solution. Since interpolation is performed many times during the subset matching process, it is evident that great amount of computation time is spent on sub-pixel intensity interpolations [12]. The other downside of these methods is that relatively good initial guess is required for the six unknowns of the shape function shown in equation 1.2 in order for these algorithms to converge at the correct solution [13].

Given for each pair of coordinates (x_i, y_i) in an original image a match can be found in deformed image at (x'_i, y'_i). Such mapping can be described with:

$$x_i'=x_i+u+\frac{\partial u}{\partial x}\Delta x_i+\frac{\partial u}{\partial y}\Delta y_i \qquad y_i'=y_i+v+\frac{\partial v}{\partial x}\Delta x_i+\frac{\partial v}{\partial y}\Delta y_i \tag{1.2}$$

where u is the horizontal and v the vertical displacement of the reference subset, and Δx_i is horizontal and Δy_i vertical distance from the centre of the subset to (x_i, y_i).

NR method is commonly used to calculate a root of a polynomial if the analytical solution cannot be calculated directly. Since the correlation coefficient can be found for each coordinate and 1-correlation coefficient is 0 where the match occurs, finding a peak in correlation matrix can be solved as a problem of finding a root of a polynomial.

Unknown parameters that need to be found are:

$$P=\begin{bmatrix} u & \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & v & \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}^T \tag{1.3}$$

Finding a peak is reduced to a minimization problem with the correlation function *C:*

$$\frac{\partial C}{\partial P_i}=0 \quad i=1...6 \tag{1.4}$$

Iterative form of NR algorithm is:

$$\begin{gathered} P^{(0)} \\ \Delta P^{(k)}=-\left\{\frac{\partial^2 C}{\partial P_i \partial P_j}\right\}^{-1}\left\{\frac{\partial C}{\partial P_i}\right\} \quad (i, j=1...6,\ k=0,1,2,...) \\ P^{(k+1)}=P^{(k)}+\Delta P^{(k)} \end{gathered} \tag{1.5}$$

In other words, starting with and initial guess $P^{(0)}$ a derivative is calculated and its root is used to improve the guess for a root of *P*; this is repeated until sufficient accuracy is achieved. Initial guess is an integer value, but for subsequent iterations interpolation of this value and its adjacent values is required [14].

Various researchers have worked to improve the execution speed of DIC using the NR method, these can be often divided into following general categories [15]:

- Hessian matrix simplification. A simplified expression for the matrix of second derivatives has been proposed and shown to reduce computational load significantly. Other works include an algorithm where re-evaluation and inversion of Hessian matrix can be avoided entirely.
- Initial guess improvements. Since the number of steps required to reach convergence depends on the accuracy of the initial guess.
- Faster sub-pixel interpolation.

1.4.2 Gradient-based

Gradient based methods rely on an assumption that image pixel intensities remain unchanged for small subsets and displacements and that mapping between original and deformed subset can be described with a simple translational motion written as follows [14]:

$$f(x_i, y_i)=g(x'_i, y'_i)=g(x_i+u+\Delta u, y_i+v+\Delta v), \tag{1.6}$$

where u and v are horizontal and vertical integer pixel displacements and Δu and Δv are respective sub-pixel displacements. After the integer displacement has been calculated sum of squared differences C need to be minimized in order to find the sub-pixel accurate solution:

$$C(\Delta u,\Delta v)=\sum\sum[f(x_i,y_i)-g(x_i+u+\Delta u,y_i+v+\Delta v)]^2 \qquad (1.7)$$

Taylor expansion of formula 1.6 is:

$$g(x_i+u+\Delta u,y_i+v+\Delta v)=g(x_i+u,y_i+v)+\Delta u\, g_x(x_i+u,y_i+v)+\Delta v\, g_y(x_i+u,y_i+v) \qquad (1.8)$$

After deduction solution to the minimization problem becomes:

$$\begin{bmatrix}\Delta u\\ \Delta v\end{bmatrix}=\begin{bmatrix}\sum\sum(g_x)^2 & \sum\sum(g_x g_y)\\ \sum\sum(g_x g_y) & \sum\sum(g_y)^2\end{bmatrix}^{-1}\cdot\begin{bmatrix}\sum\sum(f-g)g_x\\ \sum\sum(f-g)g_y\end{bmatrix} \qquad (1.9)$$

1.4.3 Iterative and spatial-gradient algorithm

Iterative and spatial gradient also relies on an assumption that image gray-levels do not change with the deformation. An optimization function that has to be minimized is:

$$F_i=f(x_i,y_i)-g(x'_i,y'_i)=0 \quad i=1,2,\ldots n \qquad (1.10)$$

Similarly to NR method relations 1.2 and 1.3 are used to solve:

$$\begin{gathered}P^{(0)}\\ \Delta P^{(k)}=[\nabla F(P^k)^T\nabla F(P^k)]^{-1}\nabla F(P^k)^T\nabla F(P^k) \quad k=0,1,2,..\\ P^{(k+1)}=P^{(k)}+\Delta P^{(k)}\end{gathered} \qquad (1.11)$$

This approach is computationally less expensive than basic NR algorithm since only first order grey derivatives need to be calculated, as opposed to second order used in NR [14].

1.4.4 Curved surface fitting

When the maximum value of the correlation coefficient and its location in the matrix is known, its surrounding values can be used to interpolate a second order polynomial surface, as shown in Figures 1.3 (a) and 1.3 (b).

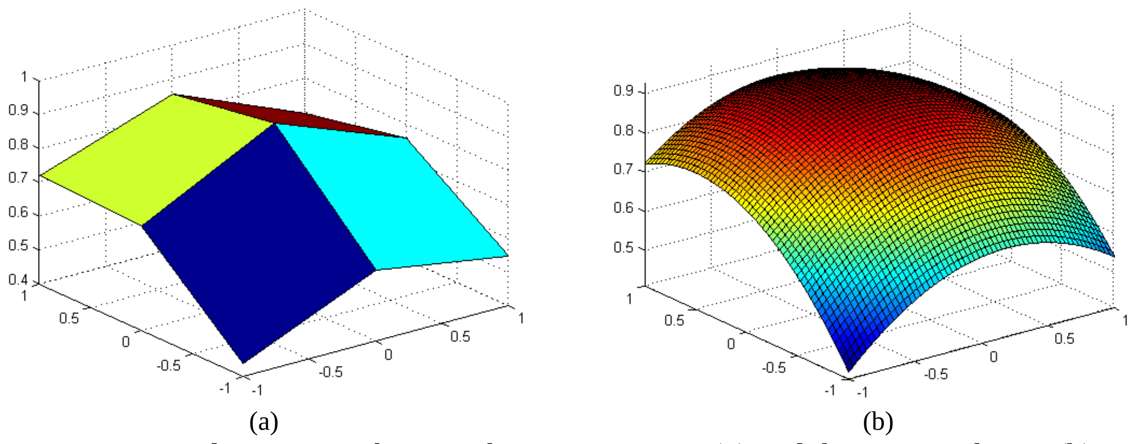


Figure 1.3: Adjacent correlation values to maximum (a) and their interpolation (b).

Second order polynomial is defined by:

$$Z(x,y)=a+bx+cy+dxy+ex^2+fy^2 \tag{1.12}$$

Let $x,y=[-1,1]$.

For a given coordinate where matrix has maximum value $c_{x,y}$ solve for adjacent values:

$$c_{x-1,y-1}=Z(-1,-1)=a-b-y+d+e+f \tag{1.13}$$

$$c_{x,y-1}=Z(0,-1)=a-c+f \tag{1.14}$$

$$c_{x+1,y-1}=Z(1,-1)=a+b-y-d+e+f \tag{1.15}$$

$$c_{x-1,y}=Z(-1,0)=a-b+e \tag{1.16}$$

$$c_{x,y}=Z(0,0)=a \tag{1.17}$$

$$c_{x+1,y}=Z(1,0)=a+b+e \tag{1.18}$$

$$c_{x-1,y+1}=Z(-1,1)=a-b+c-d+e+f \tag{1.19}$$

$$c_{x,y+1}=Z(0,1)=a+c+f \tag{1.20}$$

$$c_{x+1,y+1}=Z(1,1)=a+b+c+d+e+f \tag{1.21}$$

Since all correlation matrix values are already known, the system of equations can be solved to find coefficients *a, b, c, d, e* and *f*.

In case $c_{x,y}$ is at the first or last column or row, $c_{x,y}$ can be used as a peak, otherwise the extremum of the polynomial has to be calculated.

Extremum exists where:

$$\frac{Z'(x,y)}{dx}=b+dy+2xe=0 \qquad \frac{Z'(x,y)}{dy}=c+dx+2fy=0 \tag{1.22}$$

Location offset from the maximum value in matrix can then be calculated using:

$$x=\frac{2bf-dc}{d^2-4ef} \qquad y=\frac{bd-2ec}{4ef-d^2} \tag{1.23}$$

1.4.5 Sub-pixel method comparison

It has been shown that although easy to implement accuracy of curved surface fitting and gradient-based algorithms is lower than the accuracy of NR and spatial-gradient algorithms when complex transformations occur. Precision and stability of last two algorithms is the same, but since spatial-gradient algorithm is less complex to calculate it is recommended for applications where speed and accuracy are required [14].

1.5 Interpolation methods

The original DIC code implemented by Sutton et al. initially included the bilinear interpolation to provide sub-pixel accuracy. Due to the lack of C^1 continuity (first derivatives are not continuous) especially when used on rough topographic profiles, this scheme was later replaced by third-order polynomial interpolation (bicubic spline interpolation). Higher computational cost of bicubic spline interpolation is compensated by faster convergence rate due to improved accuracy [16].

It has been shown that using the B-spline algorithms can further improve the accuracy. Particularly quartic O-MOMS (optimal maximal order minimal support) and quintic B-spline algorithms that with respect to RMSE (root mean square error) can yield hundreds of times better results than the bicubic algorithm. Computational speeds of these algorithms have been shown to be roughly equivalent. However, to be able to benefit from more accurate interpolation algorithms high-pixel-depth, high-quality imaging system is required [12].

1.6 Speckle patterns

DIC relies on the speckle pattern on the surface of the specimen. For effective results these patterns are required to be random, isotropic and high contrast. Camera resolution relative to speckle size is also important. With too fine pattern aliasing will occur which means that surface texture won't be represented accurately. This is related to Nyquist sampling criterion which states that the sampling rate of a signal has to be at least twice the highest frequency component of the signal in order to represent it. In general, the size of the speckles should be at least 3-4 pixels, in order to avoid aliasing [17].

Natural surface textures are rarely sufficient nor optimal for DIC. Several techniques exist for applying a speckle pattern on surfaces. These include [17]:

- **spray paint** – The base is sprayed with white paint and then again with black paint to create speckles.
- **toner** – For small objects (smaller than 12mm). Surface is painted white and toner blown on it.
- **lithography or vapor deposition** – For very small specimens.
- **stencils** – Very large specimens, can be made from thin vinyl with laser or water cutting techniques and then used to roll or spray a pattern.
- **printing** - For medium to large panels. Used for specimens from 25mm to 4m in size.
- **ink** – Placed with a marker. Effect to the surface is minimal which allows to measure very high strains.
- **grid** – Although it is not optimal for DIC the grid can be used when initial guesses and subset sizes are selected so that exactly one grid intersection is contained in each subset.
- **projecting –** With a projector. Only useful for shape measurements.

1.7 Limitations

As described in the previous section contrasting surface texture or a speckle pattern has to exists for DIC to be reliable. If this requirement is not met, the DIC cannot be used. It is also required that CCD sensor and object surface should be parallel and out-of-plane motion of the specimen during loading should be small enough. Currently strain measurement accuracy of DIC is lower than that of interferometric techniques and it is not recommended for accurate non-homogeneous small deformation measurements [1].

It has been experimentally shown that correlation is reliable until the difference of sub-image rotations reaches 8°. With larger rotation angles, no obvious peaks can be found from the correlation profile which results in the detection of wrong displacements [18]. Another limitation regarding the use of Lavenberg-Marquardt or Newton-Rhapson sub-pixel optimization is that a good initial guess is required for these algorithms to converge at a correct solution.

When large deformations and rotations are present, a different approach is required in order to provide reliable initial guesses. These include using a lot of intermediate images, making easily detectable special markers or using special knowledge about the experiment. Wang et al. have developed a scheme combining scale invariant feature-transform (SIFT) and improved random sample consensus (iRANSAC) algorithms to provide fast and automatic initial guesses for the DIC method. The SIFT algorithm has been widely used in computer vision to detect and describe features in images [19, 20, 21]. SIFT works by finding image features that remain the invariant despite image rotation, translation or scaling. For each such point descriptors are calculated that help to identify same feature across different images. Feature pairs are found by matching the descriptors on both images and choosing the ones that have minimal Euclidean distance between their locations [22]. iRANSAC algorithm is then used to eliminate wrong feature pairs matches [13].

The downside of such approach is its relative complexity which may make it undesirable for a lot of use cases [23]. Therefore, alternative approaches should be considered when simplifying assumptions can be made based on the application. One possible way to find correlations between highly deflected objects is to extend the search space to a whole image and for each point try different rotations until the best correlation coefficient has been found. This can be very computationally expensive. A possible optimization is to take into account that two adjacent points should end up close to each other with similar rotation. Not only this would considerably increase the complexity of the algorithm, but may also lead to a high number of false positives when image features are not too distinct across the image.

2. Problem definition

2.1 Requirements

Deflection causes parts of material to shift and/or change their orientation as shown in Figure 2.1. It is required to be able to find correlations between areas that are rotated on the deflected image when compared to the original. Since DIC finds matches only when image features in samples overlap, an additional method has to be developed to rotate samples prior correlation.

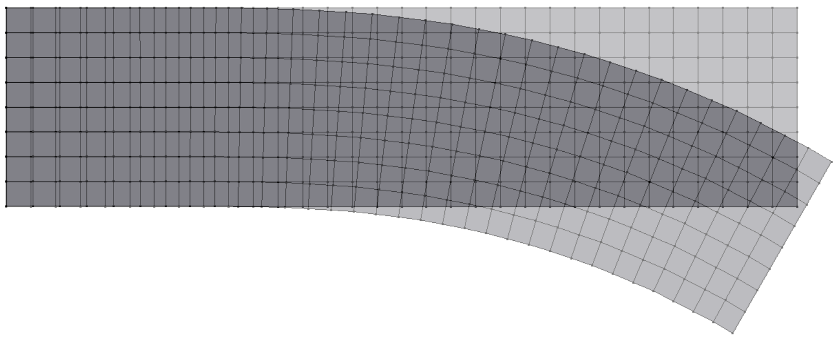


Figure 2.1: Bending geometry.

The developed technique can be adopted to different use cases due to the fact that it is independent from properties such as background features and object size.

2.2 Approach

2.2.1 Rotation tracking

Since it is difficult to implement fully automatic feature detection algorithms like SIFT a semi-automatic method is proposed in this thesis. This works well with few images, but is not suitable for large scale applications that require processing of hundreds or thousands of images.

For efficiency and accuracy, it is necessary to be able to predict the final locations and rotations of correlated image points. This requires knowledge about general deformation that occurred in the image. The proposed method for tracking location and rotation is to use user-defined curves as general guides. Figure 2.2 illustrates such curves, which are used for describing material deflection. In Figure 2.2 (a) curve has been created through two reference points whereas in Figure 2.2 (b) three points have been used. A common curve fitting algorithm such as cubic spline interpolation can be used to construct the aforementioned curves.

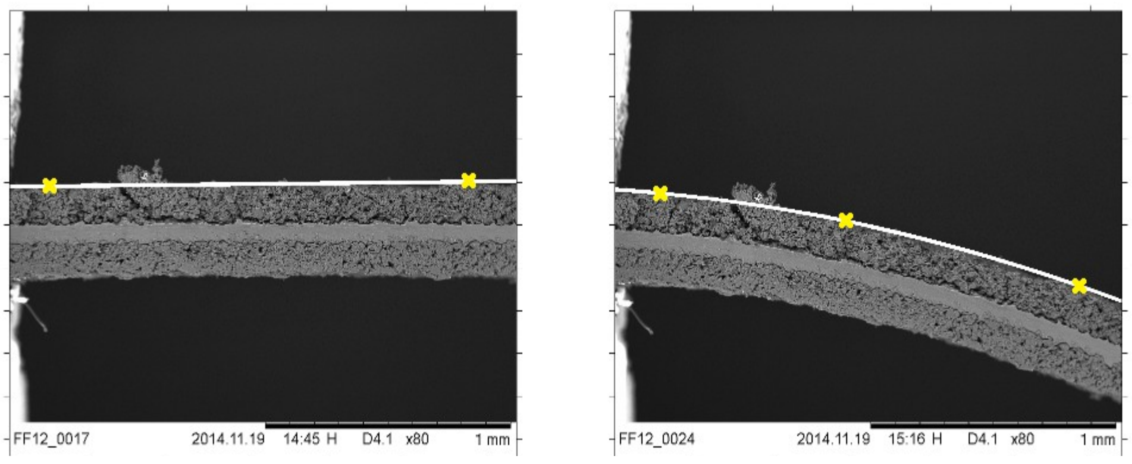

FF12_0017
2014.11.19 14:45 H D4.1 x80 1 mm
FF12_0024
2014.11.19 15:16 H D4.1 x80 1 mm
(a)
(b)

Figure 2.2: Describing general deflection using polynomial functions. Original state (a) and deformed state (b).

A cubic spline with $n+1$ points ($y_0, y_1, \ldots, y_n$) contains a total of n third degree polynomials, defined by:

$$Y_i(t)=a_i+b_i t+c_i t^2+d_i t^3, \tag{2.1}$$

where $t\in[0,1]$ and $i=0,\ldots,n-1$.

Therefore for each i it holds that:

$$Y_i(0)=y_i=a_i \tag{2.2}$$

$$Y_i(1)=y_{i+1}=a_i+b_i+c_i+d_i \tag{2.3}$$

And for derivatives:

$$Y_i'(0)=D_i=b_i \tag{2.4}$$

$$Y_i'(1)=D_{i+1}=b_i+2c_i+3d_i \tag{2.5}$$

For smooth transitions between splines, derivatives have to match at the start and end of consecutive polynomials:

$$Y_{i-1}'(1)=Y_i'(0) \tag{2.6}$$

$$Y_{i-1}''(1)=Y_i''(0) \tag{2.7}$$

For all points it also applies that:

$$Y_i(0)=y_i \tag{2.8}$$

$$Y_{i-1}(1)=y_i \tag{2.9}$$

To complete the equations the following two additional criteria must be added:

$$Y_o''(0)=0 \tag{2.10}$$

$$Y_{n-1}''(1)=0 \tag{2.11}$$

System of equations (2.2-2.11) need to be solved to find variables a_i, b_i , c_i, d_i for each segment. Knowing the sequence of polynomials and their start and end points enable calculation of deflection curve across the image width.

2.2.2 Grid transformation for initial guesses

Creating a rectangular grid with given height, width and step size is a trivial operation. Such a simple grid can then be transformed into more complex one using the predefined paths. Figure 2.3 shows the grid transformation of different curves: linear path (a), a simple curve (b) and a complex curve (c). When the first and the last point on the path are set to roughly the same place on the object, general changes in length and placement can also be approximated.

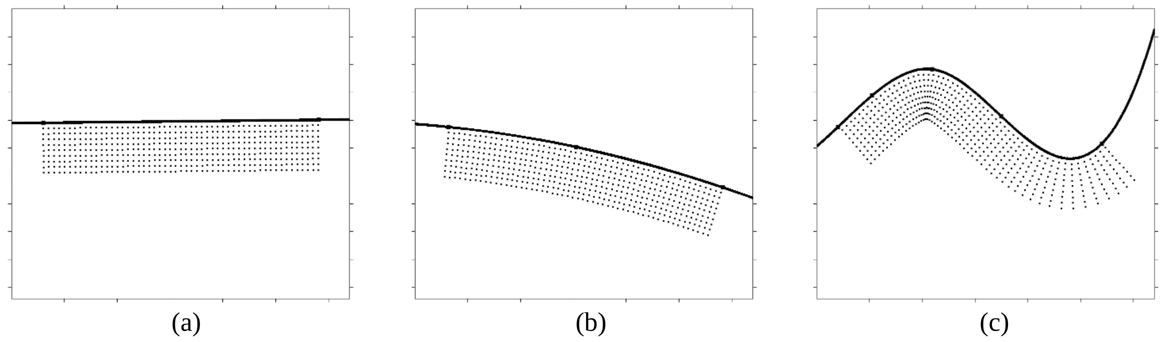


Figure 2.3: Grid transformation using path. Linear (a), simple curve (b) and complex curve (c).

The proposed algorithm assumes the origin of coordinates is in the top left corner. The following example uses two curved paths P_1 and P_2 as shown in Figure 2.4 and 2.5, the same can be applied to the case where either or both of them are linear (straight lines). For this complex example, the curve concavity and length have changed.

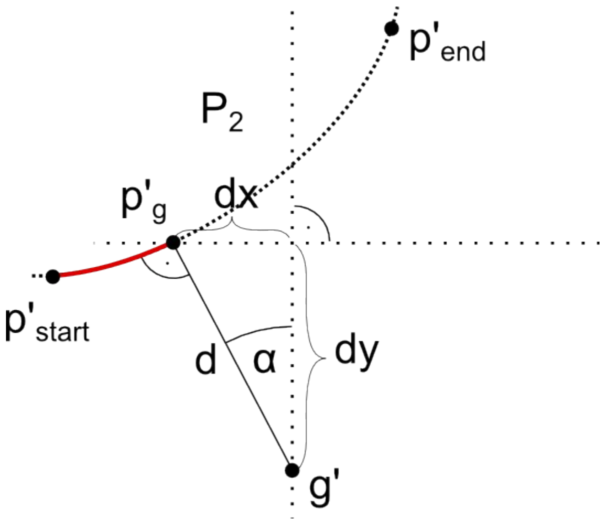


Figure 2.5: Transformation, deflected path.

Given a set of grid points $g \in G$ let $P_1(x)$ be a function that defines base path and $P_1'(x)$ its derivative. Choose two points p_{start} and p_{end} on base path that describe region of interest (where

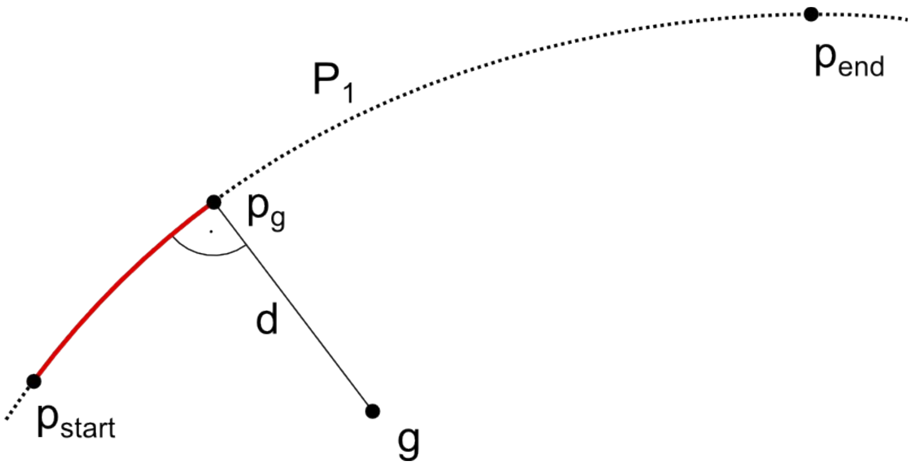


Figure 2.4: Transformation, base path.

the grid will be created). Let $P_2(x)$ be a function that defines second deflected path and $P_2'(x)$ its derivative. Let p'_{start} and p'_{end} be corresponding points on second path.

Euclidean distance from g to $P_1(x)$ for any x is defined by:

$$E(g,x)=\sqrt{(g_x-x)^2+(g_y-P_1(x))^2} \tag{2.12}$$

For each grid point g find closest point p_g with distance d on path P_1:

$$d=min(E(g,x)) \quad \text{for } x=1..\text{image width}, \tag{2.13}$$

assuming that the match is found at $p_g=(x,P(x))$.

Curve length between two points on path P_1 is defined by integral:

$$C_1(p_1, p_2) = \int_{p_{1x}}^{p_{2x}} \sqrt{1+(P_1'(x))^2}\,dx \tag{2.14}$$

Similarly on second path:

$$C_2(p_1, p_2) = \int_{p_{1x}}^{p_{2x}} \sqrt{1+(P_2'(x))^2}\,dx \tag{2.15}$$

Let $p'_g = (p'_{gx}, P(p'_{gx}))$ be corresponding point to p_g on second path that is located at the same distance ratio between the start and endpoint of the path (red curves in Figure 2.4 and Figure 2.5). Therefore distance $C_2(p'_{start}, p'_g)$ without knowing p'_g *is*:

$$l = C_2(p'_{start}, p'_g) = \frac{C_1(p_{start}, p_g)}{C_1(p_{start}, p_{end})} C_2(p'_{start}, p'_{end}) \tag{2.16}$$

p'_g can then be found by solving:

$$l = \int_{p'_{startx}}^{p'_{gx}} \sqrt{1+(P_2'(x))^2}\,dx \tag{2.17}$$

Given $D_p = P_2'(p'_{gx})$ is a derivative of P_2 at p'_g, therefore:

$$\alpha = \tan^{-1}(D_p) \qquad dx = \sin(\alpha)\cdot d \qquad dy = \cos(\alpha)\cdot d \tag{2.18}$$

Final g' can be calculated:

$$g'_x = p'_x - dx \qquad g'_y = p'_y + dy \tag{2.19}$$

2.2.3 Calculating correlated grid location

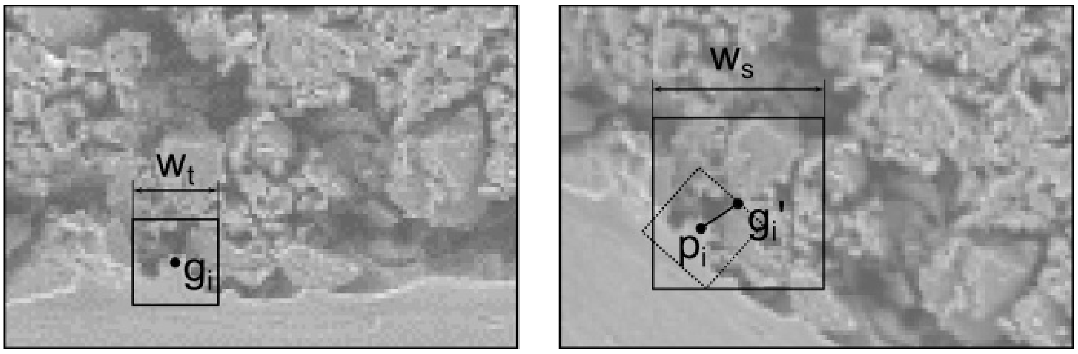
From the transformation step there is the following data available:

- Grid point locations $g_i \in G$ on the base image.
- Estimations for grid point locations $g_i' \in G$ and rotations $r_i' \in R$ on the deflected image.

We choose an odd number for square template width w_t around each grid point on the base image that will be used to search correlations from the deflected image. Similarly we choose an odd number w_s for square template width around each grid point on the deformed sub-image. Also we require that $w_t \le w_s$. Using odd numbers for widths puts the centre of the region into the centre of a pixel, this can later be used to simplify finding the offset.

Template from the first image is used to search correlations with a larger sample on the second image. This common approach works as long as image data from template and deformed sub-image do not differ too much. In case of major deflection straight forward cross-correlation results in poor coefficients across the sample since image features do not overlap in larger regions any more.

Assuming that g_i' in Figure 2.6 (b) is the approximate location of the final point p_i that was found using mathematical transformation described in the previous section, it can be expected that original image data from the template on Figure 2.6 (a) will be rotated near g_i' with a rotation of r_i'.



(a) (b)

Figure 2.6: Correlation template (a) and deflected sub-image (b).

Image area width that has to be cropped from the base image prior rotation has to be able to fit template rotated at 45° where former image diagonal becomes its width. Then the sample size can be calculated as:

$$w_r = \sqrt{w_t^2 + w_t^2} \tag{2.20}$$

After performing the rotation the template can be cropped back to its intended size $w_t \times w_t$.

Normalized 2D cross-correlation between template and deflected sub-image produces a square matrix C with dimensions $(w_t + w_s - 1) \times (w_t + w_s - 1)$ when the template and sub-image are correlated so that the centre of the template "moves" over each pixel on the sub-image. This ensures that the final location will stay inside the search region that is defined by w_s.

A correlation coefficient peak c_p exists in this matrix if correlation criteria have been chosen so that better correlations yield values closer to 1. The easiest way to find the peak is to take the maximum element from the matrix. However, other possibilities exist that improve accuracy as described in chapter 1.4.

The geometrical centre c_c of the correlation matrix C is at row and column:

$$\frac{w_t+w_s-1}{2}+\frac{1}{2}=\frac{wt+ws}{2} \tag{2.21}$$

Displacement from the centre of the matrix would therefore be:

$$d=\left(\frac{wt+ws}{2}-c_{px},\frac{wt+ws}{2}-c_{py}\right) \tag{2.22}$$

The centre of matrix c_c refers to the same location as g_i' on the deflected image. Final grid point location p_i can then be calculated using:

$$p_i=(g_i{'}_x-d_{ix},g_i{'}_y-d_{iy}) \tag{2.23}$$

2.2.3 Sub-pixel accuracy

2.2.3.1 Optimization scheme

Since the relative accuracy of DIC is not the focus of this thesis curved surface fitting algorithm described in section 1.4.4 is used to achieve sub-pixel accuracy due to its simplicity.

2.2.3.2 Compensating base grid fractional coordinates

If the base grid has been constructed using fractional coordinates an additional error is introduced to the results. Image correlation works on an area of pixels as a whole and has no information about the original coordinates.

Each grid point *g* is considered to be in the centre of a pixel *c* by the correlation as can be seen in Figure 2.7 (a). A match is found on the second image at a coordinate *c'* shown on Figure 2.7 (b). However since the original location *g* may have not been in the centre of the pixel the final result has to be corrected so that grid point on the second image would end up at *g'*.

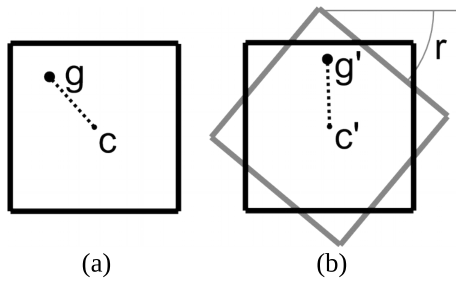


Figure 2.7: Base (a) and rotated grid pixels (b).

Fractional offset on the base grid can be described with a vector $\vec{cg}$. Since rotation *r* is already known from the transformation step, the vector $\overrightarrow{c'g'}$ can be constructed using rotation matrix:

$$\overrightarrow{c'g'}=\vec{cg}*\begin{bmatrix}\cos(r) & -\sin(r)\\ \sin(r) & \cos(r)\end{bmatrix} \tag{2.24}$$

Final location can then be calculated:

$$g'=c'+\overrightarrow{c'g'} \tag{2.25}$$

This method relies on the accuracy of finding the rotation *r*. Since the rotation is calculated using the path defined by the user it can contain errors. The following section describes improving the rotation.

2.2.3.3 Compensating path rotation errors

Assuming that the correct rotation provides the highest correlation coefficient it is possible to improve sub-pixel accuracy by maximizing correlation coefficient. Let *r* be the rotation that is calculated in the grid transformation step. It can be considered to be relatively close to the correct rotation. Figure 2.8 describes finding the best rotation by calculating correlation coefficient between same image samples but by changing the rotation by a constant increment of *c*. This iterative approach ensures that if there are no local maximums the absolute peak will be found, otherwise the local maximum is found. The accuracy can be further improved using interpolation.

If the peak is at rotation *r*+*ci*, second degree polynomial can be used to approximate the actual extremum. Let *C(r)* be correlation coefficient function with base sample rotation *r*. Second degree polynomial in two-dimensional space is defined by:

$$y=a+bx+cx^2 \tag{2.26}$$

Given that:

$$C(r+c(i-1))=y(-1)=a-b+c \tag{2.27}$$

$$C(r+ci)=y(0)=a \tag{2.28}$$

$$C(r+c(i+1))=y(1)=a+b+c \tag{2.29}$$

Coefficients *a, b, c* can be found by solving this system of equations. Differential of *y* is $y'=b+2xc$. Extremum exists where y'=0. Offset *x* can then be calculated using:

$$x=-\frac{b}{2c} \tag{2.30}$$

The final adjusted rotation is therefore *r+ci+x.*

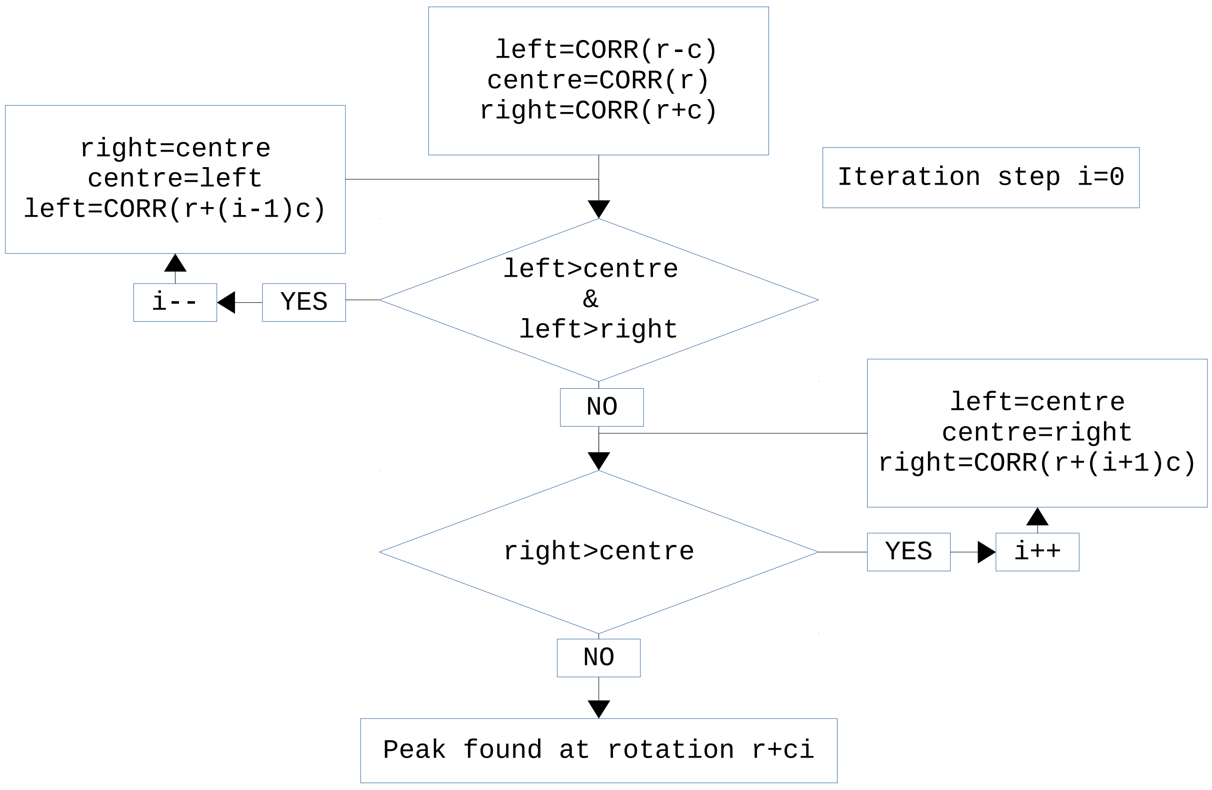


Figure 2.8: Rotation adjustment algorithm.

3. Experimental Results

3.1 Implementation

3.1.1 Improved DIC

Implementation of the proposed approach is based on Matlab Improved DIC library [24]. The following files are used:

- *findpeak.m* – for finding a peak in matrix using interpolation;
- *cpcorr_mod.m* – a modified version of Matlab *cpcorr.m* that enables to tune control-point locations using cross-correlation. This file has been further modified to support rotating the image samples.

See Appendix for comments and source code.

3.1.2 Installation

Rotated DIC library has no other dependencies than standard packages included with Matlab installation. Implementation has been developed and tested on Matlab R2012a.

In order to use the library all the files included have to be added to Matlab execution path. File→Set Path and Add Folder.

3.1.3 Functions

The library contains two executable files *rotated_dic.m* and *rotated_dic_correlate.m* which can be called using the corresponding function names.

rotated_dic(grid_step, search_window, subset, tune_rotation)

Main executable, calculating correlation between two images.

- **grid_step** (integer) - distance in pixels between grid points
- **subset** (integer) - width of the template from the base image used to search correlations with a sub-image from input image, defaults to grid step size in pixels
- **search_window** (integer) - how many times the sub-image width from input image

exceeds subset (template) width

- **tune_rotation** (1/0) - optional, if 1 algorithm will try to find a rotation that gives highest correlation

rotated_dic_correlate(search_window, subset, display, tune_rotation)

For rerunning correlation with different input parameters after *rotated_dic.m* has run once.

- **subset** (integer) - optional
- **display** (1/0) - optional, 1-show images, 0-don't, defaults to 1
- **tune_rotation** (1/0) – optional, defaults to 0

3.1.4 Usage and output

Sample execution using grid_step 20 pixels and search_window 2: *rotated_dic(20, 2)*.

1. User is asked to open a base image using a file dialogue.
2. Clicking on the image sets start and endpoints for the base path. In Figure 2.9 a dialogue appears asking for confirmation weather more points should be added. For this example two points for base path is sufficient. More points can be added until the path matches the desired curve.
3. After path has been set, an another dialogue appears as can be seen in Figure 2.10 and asks for the number of grid rows to be generated, this can be further adjusted until desired result is achieved. Generated grid will be between first and last point on path and follows its geometry. For this example 9 rows are used.
4. User is asked to open the input image.
5. Input path can be defined in the same way as the base path. If the final path is confirmed the program continues with the correlation.
6. Results are displayed, this includes: transformation on Figure 2.11, correlation on Figure 2.12 and visualized changes in relative grid point placements as can be seen on Figure 2.13.

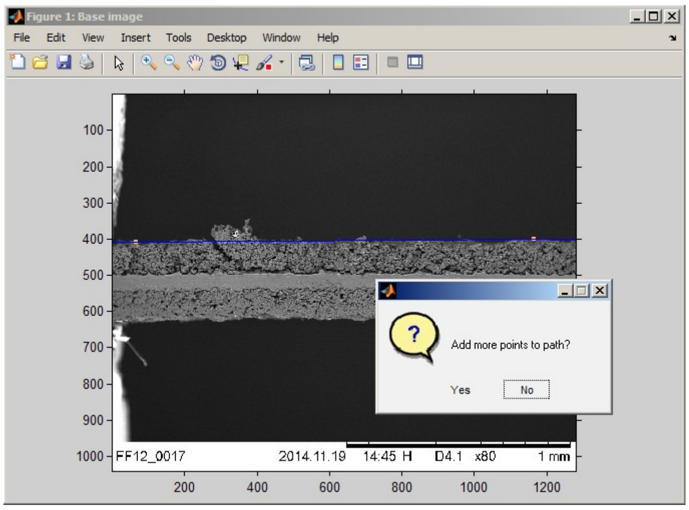


Figure 1: Base image
File Edit View Insert Tools Desktop Window Help
100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
FF12_0017 2014.11.19 14:45 H D4.1 x80 1 mm
Yes
No

Add more points to path?

Figure 2.9: Usage, defining a path.

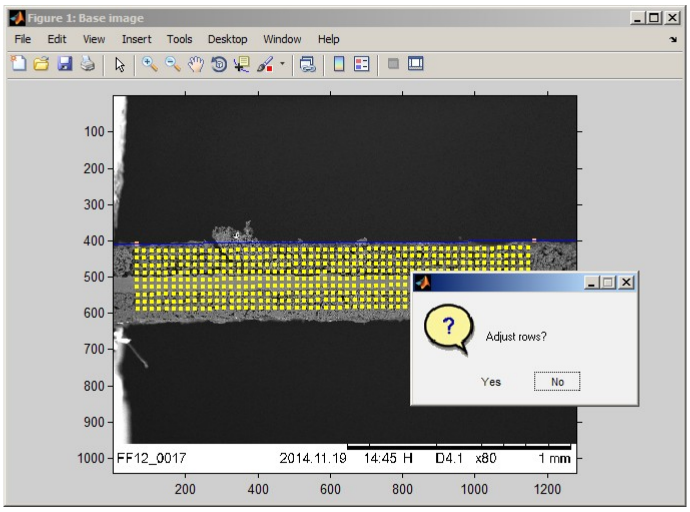


Figure 1: Base image
File Edit View Insert Tools Desktop Window Help
100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
FF12_0017 2014.11.19 14:45 H D4.1 x80 1 mm
Yes
No

Adjust rows?

Figure 2.10: Usage, defining a grid.

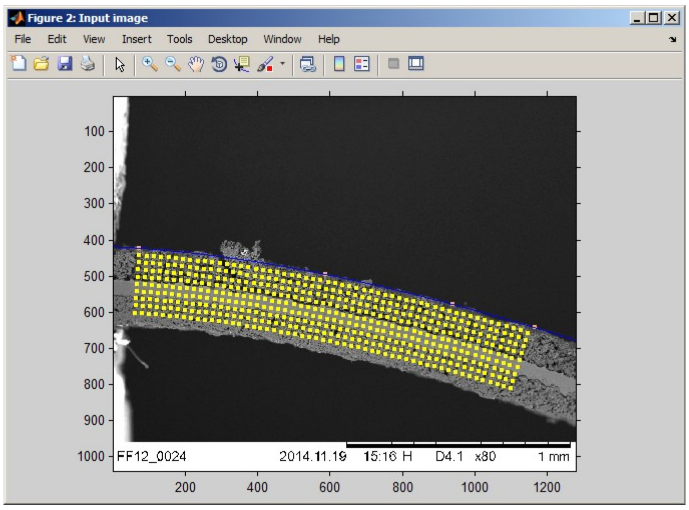


Figure 2: Input image
File Edit View Insert Tools Desktop Window Help
100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
FF12_0024 2014.11.19 15:16 H D4.1 x80 1 mm

Figure 2.11: Results, transformed grid.

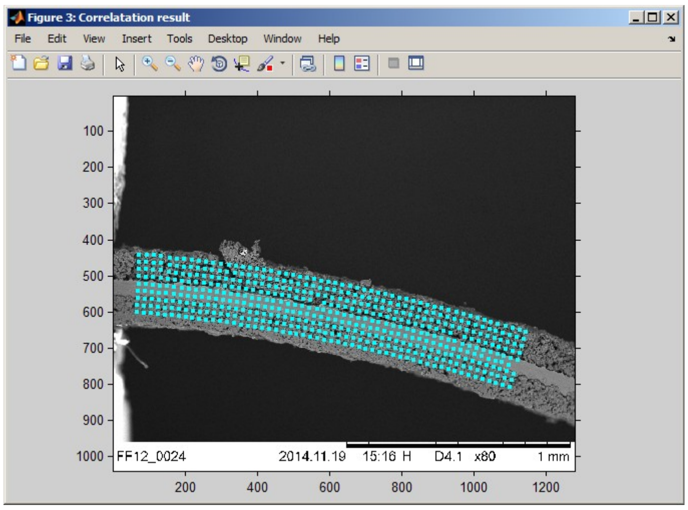


Figure 3: Correlatation result
File Edit View Insert Tools Desktop Window Help
100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
FF12_0024
2014.11.19 15:16 H D4.1 x80 1 mm

Figure 2.12: Results, correlated grid.

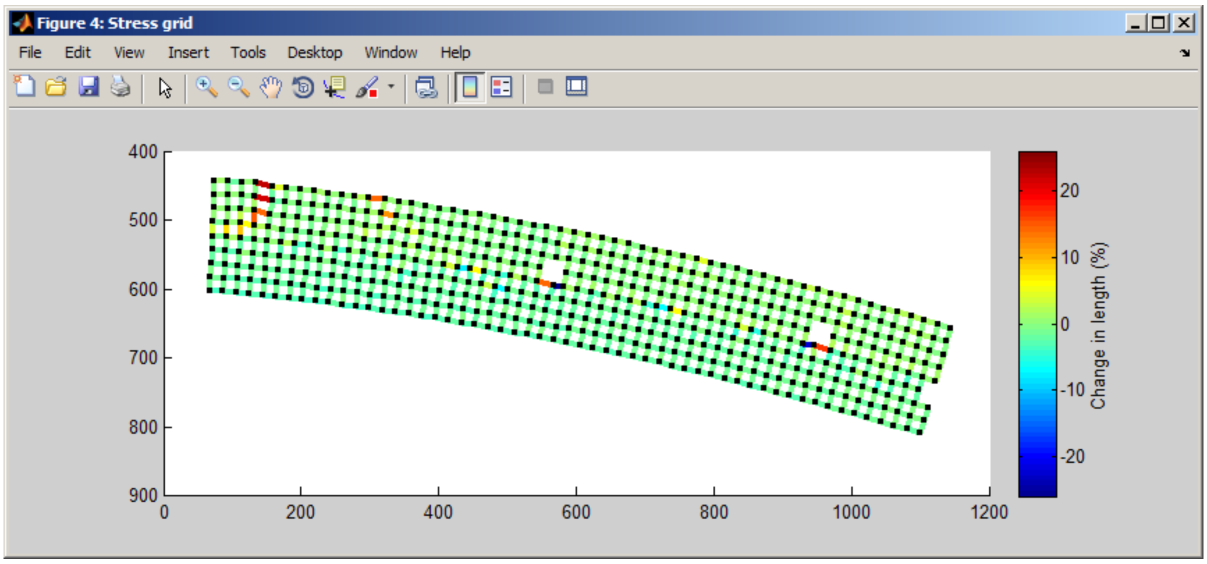


Figure 2.13: Results, stress grid.

As it can be seen from Figures 2.12 and 2.13, majority of original grid points find their match. Correlation coefficients lower than 0.5 are discarded by the library. This is often due to larger changes in the image so that correlation cannot find similar areas any more, the area was too featureless to be distinguishable or the area around transformed grid point (the initial guess) did not contain the correct point. In the latter case it can be tried again with a new path and grid or correlation part can be rerun with the same grid and path data using *rotated_dic_correlate.m* function. This enables to change the correlated area size and look for matches from a larger area.

Correlation and transformation results are written into Matlab data files:

- result-transform.mat
  - base_grid (2xN) - points to be correlated on base image
  - input_grid (2xN) - points to be correlated on input image
  - base_path (2ximage width) - path that describes object deflection on base image
  - input_path (2ximage width) - path that describes object deflection on input image
  - base_rotations (1xN) - rotations in radians for each base grid point
  - rotations (1xN) - rotations in radians for each input grid point
  - base_filename - absolute path to base image file
  - input_filename - absolute path to input image file
  - points_in_row - how many points in a grid row
  - grid_step
  - base_p - x, y coordinate pairs used to create base_path as defined by user
  - input_p - x, y coordinate pairs used to create input_path as defined by user
- result-correlation.mat
  - result (2xN) - adjusted grid points after correlation, x and y coordinates for N grid points
  - corr_coeff - average correlation coefficient (0-1)

3.2 Accuracy and testing

3.2.1 Validation procedure

All tests were performed on an Intel Core i7 Q720 (1.6 GHz) laptop PC with 4GB of RAM installed. Accuracy has been evaluated using mean square error (MSE) based on Euclidean distances between expected and found grid point locations. MSE is commonly used for assessing DIC accuracy [12, 25]. Average correlation coefficient has also been calculated to show the quality of the correlation results. Numerical tests were carried out by applying rigidbody rotation to specimen images [26]. All test images show artificial muscles with contrasting surface texture taken using Hitachi TM3000 tabletop SEM.

3.2.2 Manual testing

Due to the difficulty of generating images that display mathematically predictable deflection, paths with varying curvature have to be tested partially manually on real images. For one such case, 16 distinct points were selected on an image shown in Figure 2.14, and their location was determined on the deflected image by choosing integer coordinates by an eye. It is expected that such method will introduce measurement error of approximately 1 pixel.

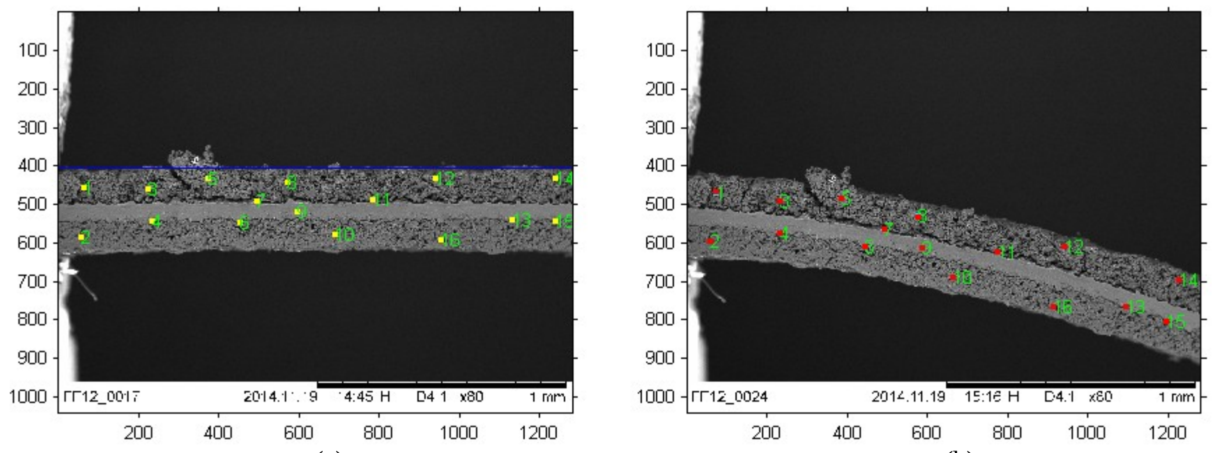


Figure 2.14: Manual testing setup, original image (a) and deflected image (b).

Square of the Euclidean distance from the desired location was calculated for both transformation and correlation (square error). Summary is shown in Table 2.1 and detailed results in Table 2.2.

Table 2.1: General results for manual testing.

Average correlation coefficient	0.908
Transform mean square error (pixels)	10.838
Correlation mean square error (pixels)	0.267

Correlation between transformation and correlation error is -0.0914, this indicates that a bigger errors in setting the initial guess for the grid point do not result in worse accuracy when finding the final location using correlation. However, errors shown here are small and it is clear that larger errors lead to correlation not being found at all due to the point moving out of the search window.

Table 2.2: Detailed results for manual testing.

	Correlation coefficient	Transform square error	Correlation square error
1	0.9114	2.236	0.146
2	0.9312	**26.790**	0.180
3	0.9583	17.071	0.020
4	0.9460	11.646	0.278
5	0.8254	23.471	0.159
6	0.9200	9.797	0.296
7	0.9329	14.418	0.021
8	0.8619	15.159	0.255
9	0.7284	12.065	0.271
10	0.8524	3.576	0.202
11	0.9232	12.487	0.415
12	0.8563	10.050	0.546
13	0.9635	0.498	0.129
14	0.9669	9.164	**0.961**
15	0.9775	1.264	0.262
16	**0.9787**	3.719	0.126

3.2.3 Mathematical testing

More accurate testing can be done on an image with a rectangular grid that will be rotated by an angle as can be seen on Figure 2.15. In this setup, all grid points on the deflected image can be precisely calculated. Paths were still set manually as they would by the user of the library. Errors were calculated by measuring the Euclidean distance between pre-calculated and correlated location. Test parameters are shown in Table 2.3.

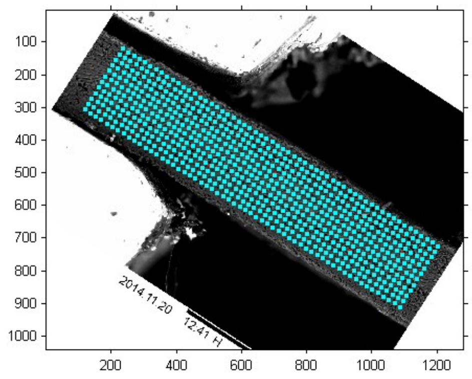


Figure 2.15: Linear path testing setup.

Table 2.3: Test parameters.

Grid step size	20 pixels
Subset size	20 pixels
Search window	3
Rotation adjustment	off
Correlation peak interpolation	on

Three distinct test cases were observed shown in Tables 2.4, 2.5 and 2.6. Since in real-life applications only the final DIC accuracy is important, transformation accuracy is no longer observed.

Table 2.4: Test case 1, original base image and 33° rotated input image.

Grid points (nr)	696
Mean correlation coefficient	0.974
Mean square error (pixels)	0.004

Table 2.5: Test case 2, 20° rotated base image and -45° rotated input image.

Grid points (nr)	855
Mean correlation coefficient	0.964
Mean square error (pixels)	0.006

Table 2.6: Test case 3, base image equals input image.

Grid points (nr)	696
Mean correlation coefficient	0.974
Mean square error (pixels)	0.001

It can be seen that even if images are the same the correlation still introduces errors, this due to interpolation and disappears when interpolation is removed. Highest possible accuracy using this implementation with respect to mean square error is therefore 0.001 while with actual rotations 0.004 which proves that curve based transformation can provide initial guesses for an accurate DIC.

3.2.4 Sub-pixel accuracy

3.2.4.1 Peak interpolation

Same test cases were used as described in section 3.2.3. For each case, mean square error (MSE) was calculated shown in Table 2.7.

Table 2.7: Interpolation effect on accuracy.

Test case nr.	MSE (peak interpolation)	MSE (no peak interpolation)
1	0.004	0.180
2	0.006	0.175
3	0.001	0.000

When correlating the same image (test case nr 3) peak interpolation creates a small error, but for actual use cases when images need to be rotated and image data changes, it can improve sub-pixel accuracy.

3.2.4.2 Rotation adjustment

For the following results, previously defined test cases 1, 2, 3 were run and mean square error (MSE) and mean correlation coefficients (MCC) were calculated.

Table 2.8: Rotation adjustment effect on accuracy and execution speed.

	No rotation adjustment			**With rotation adjustment**		
Test case nr.	**MSE**	**MCC**	**Time**	**MSE**	**MCC**	**Time**
1	0.004	0.974	5s	0.004	0.974	20.6s
2	0.006	0.964	6.1s	0.005	0.964	25.4s
3	0.001	0.974	2.5s	0.001	0.975	18.5s

It can be seen from the results in Table 2.8 that the effect of rotation adjustment is negligible for these cases, but execution time is considerably increased. Correlation coefficient has improved marginally, but this does not have a positive effect on the accuracy. This can be due to the fact that paths are well defined and rotations accurate enough. However when a rotation error of 5.72° (0.1 radians) is manually added, the results can differ considerably as can be seen from Table 2.9.

Table 2.9: Rotation adjustment effect on accuracy and speed when additional error introduced.

	No rotation adjustment			**With rotation adjustment**		
Test case nr.	**MSE**	**MCC**	**Time**	**MSE**	**MCC**	**Time**
1	0.086	0.867	5.1s	0.003	0.974	49.5s
2	0.123	0.860	5.8s	0.006	0.964	58.8s
3	0.095	0.857	4.7s	0.001	0.975	46.6s

Adjusting rotation improves both correlation coefficient and mean square error. Interestingly, accuracy for the first test case has improved over the result with unaltered rotation. Since peak interpolation is used, accuracy may vary based on input data, which adds randomness to the results. Execution times have further increased.

It is clear that when path is not well defined, rotation adjustment can improve accuracy. However this increases the execution time. Mean correlation coefficient can be used as an indication when the adjustment should be applied. Lower correlation coefficients may indicate that it is necessary to tune rotation. When the result with adjustments gives higher mean correlation coefficient it can be assumed that accuracy was improved as well.

Summary

The aim of this thesis was to develop an easy-to-use method for providing initial guesses for DIC grid point locations and rotations on objects where major deflection occurs. Study of literature revealed that special approach is required to be able to perform DIC on subsets that are rotated, although such methods exist they are difficult to implement. A method was proposed that employs manual input for creating a curve across image width that describes the specimen deformation. Deflection curve could then be used to provide initial guesses and rotations prior DIC.

The approach was implemented using MATLAB and several tests were made to evaluate the algorithm. The results showed that curve based grid transformation can provide sufficient estimation to be able to carry out DIC measurements and converge to sub-pixel accurate solution. The magnitude of displacement error caused by transformation was shown to be irrelevant when search windows are big enough for the correct location to be included. However rotational errors were proven to be significant for sub-pixel accuracy, additional method was proposed and shown to work for compensating errors in rotation estimations.

In conclusion the results have shown that the described method can be used for providing initial guesses for grid point locations on deflected objects. Therefore DIC can be used with this approach to carry out deflection measurements.

Poolautomaatne painde mõõtmine kasutades digital image correlation meetodit

Siim Sundla

Kokkuvõte

Käesoleva magistritöö eesmärgiks oli arendada välja lihtsalt kasutatav meetod esialgsete *digital image correlation* (DIC) meetodi koordinaatpunktide asukohtade ning rotatsiooni määramiseks suure paindega objektidel. Kirjanduse uuringust selgus, et vajalik on erinev lähenemine võrreldes tavapärase DIC kasutamisega. Kuigi sellelaadsed algoritmid on olemas on nad reaalsuses raskesti kasutatavad. Magistritöös pakutud meetod rakendab kasutaja poolset sisendit, et koostada kõverjoon objekti painde kirjeldamiseks, mida on võimalik kasutada DIC koordinaatpunktide asukoha ning rotatsiooni hindamiseks.

Kirjeldatud algoritm realiseeriti kasutades MATLAB programmerimiskeelt. Selleks, et tõestada lähenemise korrektsust viidi läbi mitmed testid. Tulemused näitasid, et kõverjoontel põhinev koordinaadipunktide muundamine annab piisava täpsusega sisendi, et läbi viia alam-piksli täpsusega DIC mõõtmine. Ühtlasi näidati, et asukoha määramise viga on ebaoluline kui DIC otsinguaken on piisavalt suur, et sisaldada õiget punkti. Teisalt rotatsiooni hinnangu vead mõjutasid otseselt tulemuse kvaliteeti. Vea kompenseerimiseks esitati meetod, mille toimimist näidati ka testide tulemustes.

Kokkuvõtvalt võib tulemustest järeldada, et esitatud meetodit saab kasutada DIC punktide asukohtade ning rotatsioonide määramiseks paindega objektidel, mis võimaldab omakorda läbi viia mõõtmisi painde määramiseks.

Acknowledgements

I would like to thank my supervisors Gholamreza Anbarjafari and Andres Punning for advice and guidance on writing this thesis. Additional thanks goes to Andres Punning for providing SEM images of artificial muscles that were used for evaluating the developed approach.

References

[1] B. Pan, K. Qian, H. Xie and A. Asundi, “Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review”, *Measurement Science and Technology*, 2009, **20**, DOI:10.1088/0957-0233/20/6/062001.

[2] Hobrough, G. L. (1971). U.S. Patent No. 3,595,995. Washington, DC: U.S. Patent and Trademark Office.

[3] M. A. Sutton, J. Orteu and H. W. Schreier, “Image Correlation for Shape, Motion and Deformation Measurements”, Springer Science+Business Media, New York, 2009.

[4] J. D. Krehbiel and T. A. Berfield, “Applying Digital Image Correlation to Biological Materials”, http://sottosgroup.beckman.illinois.edu/papers/undergrads/REU_Krehbiel.pdf. 08.05.2015, 12:00(UTC).

[5] S. Yoneyama and H. Ueda, “Bridge Deflection Measurement Using Digital Image Correlation with Camera Movement Correction”, *Materials Transactions*, 2012, **53(2)**, 285- 290, DOI:10.2320/matertrans.I-M2011843.

[6] J. P. Lewis, “Fast Normalized Cross-Correlation”, Industrial Light & Magic, http://scribblethink.org/Work/nvisionInterface/nip.html, 08.05.2015, 12:00(UTC).

[7] R. Roncella, E. Romeo, L. Barazzetti, M. Gianinetto and M. Scaioni, “Comparative Analysis of Digital Image Correlation Techniques for In-plane Displacement Measurements”, *5th International Congress on Image and Signal Processing*, 2012, 721-726, DOI:10.1109/CISP.2012.6469731.

[8] Mathworks. MATLAB Normalized 2-D cross-correlation function description. http://se.mathworks.com/help/images/ref/normxcorr2.html 14.05.2015, 17:00(UTC).

[9] B. Pan and K. Li, “A fast digital image correlation method for deformation measurement”, *Optics and Lasers in Engineering,* 2011, **49**, 841–847, DOI:10.1016/j.optlaseng.2011.02.023.

[10] Z. Wang, H. Kieu, H. Nguyen and M. Le, “Digital image correlation in experimental mechanics and image registration in computer vision: Similarities, differences and complements”, *Optics and Lasers in Engineering*, 2015, **65,** 18–27,

DOI:10.1016/j.optlaseng.2014.04.002.

[11] S. Yaofeng, T. Y. Meng, J. H. L. Pang and S. Fei, “Digital Image Correlation and its Applications in Electronics Packaging”, *Electronic Packaging Technology Conference. Proceedings of 7th*, 2005, DOI:10.1109/EPTC.2005.1614380.

[12] L. Luu, Z. Wang, M. Vo, T. Hoang and J. Ma, “Accuracy enhancement of digital image correlation with B-spline interpolation”, *Optics Letters*, 2011, **36(16),** 3070-3072, DOI:10.1364/OL.36.003070.

[13] Z. Wang, M. Vo, H. Kieu and T. Pan, “Automated Fast Initial Guess in Digital Image Correlation”, *Strain*, 2014, **50**, 28–36, DOI:10.1111/str.12063.

[14] L. Xiong, X. Liu, G. Liu, J. Liu, X. Yang and Q. Tan, “Evaluation of Sub-pixel Displacement Measurement Algorithms in Digital Image Correlation”, *International Conference on Mechatronic Science, Electric Engineering and Computer,* 2011, 1066-1069, DOI:10.1109/MEC.2011.6025650.

[15] Z. Wang, S. Wang and Z. Wang, “An analysis on computational load of DIC based on Newton–Raphson scheme”, *Optics and Lasers in Engineering*, 2014, **52,** 61–65, DOI:10.1016/j.optlaseng.2013.07.019.

[16] G. Vendroux and W. G. Knauss, “Submicron Deformation Field Measurements II: Improved Digital Image Correlation”, *Experimental Mechanics*, 1998, **38(2),** 86-92, DOI:10.1007/BF02321649.

[17] “Speckle Pattern Fundamentals”, CSI Application Note AN-525, https://melab.wikischolars.columbia.edu/file/view/AN525+-+Speckle+Pattern+Fundamentals.pdf, 08.05.2015, 12:00(UTC).

[18] P. Hung and A. S. Voloshin, “In-plane Strain Measurement by Digital Image Correlation”, *J. of the Braz. Soc. of Mech. Sci. & Eng*. 2003, **25(3)**, 215-221, DOI:10.1590/S1678-58782003000300001.

[19] J. Luo, Y. Ma, E. Takikawa, S. Lao, M. Kawade and B. L. Lu. “Person-specific SIFT features for face recognition”, *Acoustics, Speech and Signal Processing*, *ICASSP 2007. IEEE International Conference*, 2007, **2**, II-593-II-596, DOI:10.1109/ICASSP.2007.366305.

[20] T. Sun, S. Ding and Z. Ren. “Novel image recognition based on subspace and SIFT”, *Journal of Software*, 2013, **8(5)**, 1109-1116, DOI:10.4304/jsw.8.5.1109-1116.

[21] K. H. Ali and T. Wang, “Recognition of Human Action and Identification Based on SIFT and Watermark”. *Intelligent Computing Methodologies*, 2014, **8589,** 298-309. DOI:10.1007/978-3-319-09339-0_31.

[22] D. G. Lowe, “Object recognition from local scale-invariant features”, *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, **2**, 1150-1157, DOI:10.1109/ICCV.1999.790410.

[23] A. Punning, V. Vunder, I. Must, U. Johanson, G. Anbarjafari and A. Aabloo, “In situ scanning electron microscopy study of strains of ionic electroactive polymer actuators.”, *Journal of Intelligent Material Systems and Structures*, 2015 DOI:10.1177/1045389X15581520.

[24] Mathworks. MATLAB Improved Digital Image Correlation library documentation. http://www.mathworks.com/matlabcentral/fileexchange/43073-improved-digital-image-correlation--dic- 15.05.2015, 17:00(UTC).

[25] C. Cofaru, W. Philips and W. Van Paepegem, “Evaluation of digital image correlation techniques using realistic ground truth speckle images”, *Measurement Science and Technology*, 2010, **21**, DOI:10.1088/0957-0233/21/5/055102.

[26] A. M. R. Sousa, J. Xavier, M. Vaz, J. J. L. Morais and V. M. J. Filipe, “Measurement of displacement fields with sub-pixel accuracy by combining cross-correlation and optical flow”. *8º Congresso Nacional de Mecânica Experimental*, 2010.

Appendix

The following section contains primary source files for the rotated DIC MATLAB library. Full source with test code and data is provided on the CD included with the thesis.

README.md

Rotated DIC Matlab library

```
====================================
Searches for correlations between two images taking into account changes in general geometry that are
represented by two curves that are defined by the user. Makes it possible to find same images points
from images that are deflected.
Files
-----
### Source ###
* rotated_dic.m - executable code
* rotated_dic_correlate.m - executable code, only correlation part
* transform_grid.m - grid trasform based on paths for initial guesses
* draw_colored_strain_grid.m - strain visualization
* cpcorr_mod.m - from Matlab "Improved Digital Image Correlation" library, modified to rotate
correlated samples
* findpeak.m - from Matlab "Improved Digital Image Correlation" library
* iterate_path.m - finds current path length and differencial for each point on path
* get_path_length.m - gets path length between given points
* construct_base_grid.m - creates grid based on given path

### Test ###
* test/test_errors.m - function for calculating transform and correlation errors base on predefined data
* test/test_transform.m - transform errors test
* test/test_cor_linear_errors.m - correlation errors test
* test/test_cor_linear_errors2.m - correlation errors test, both images rotated
* test/test_cor_same_errors.m - correlation errors test with same image
* test/calc_and_plot_errors.m - test helper function
* test/calc_expected_grid.m - test helper function
* test/calc_square_errors.m - test helper function

* test/test-setup.mat - path and grid data for tests
* test/test-setup-transform.mat - path and grid data for tests
* test/test-setup-cor-linear.mat - data for correlation errors test
* test/test-setup-cor-linear2.mat - data for correlation errors test 2

* test/test-base.tif
* test/test-input.tif
* test/test-linear-base.tif
* test/test-linear-33deg.tif
* test/test-linear-2-20deg.tif
* test/test-linear-2-45deg.tif
```

Execution

### General terms and parameters
- **grid_step** (int) - distance in pixels between grid points
- **subset** (int) - width of the template from the base image used to search correlations with a sub-image from input image, defaults to grid step size in pixels
- **search_window** (int) - how many times the sub-image width from input image exceeds subset (template) width
- **tune_rotation** - optional, if 1 algorithm will try to find a rotation that gives highest correlation. Improves subpixel accuracy when rotations calculated from path have errors, but slows down execution speed considerably

### rotated_dic(grid_step, search_window, subset, tune_rotation)
Main executable, calculating correlation between two images

* subset - optional
* tune_rotation - optional, defaults to 0

1. open base image
2. click on base image to create curve that matches general material deformation
3. enter how many grid rows should be generated (grid starts from first path point and end with last)
4. open input image
5. click on input image to create curve that matches general material deformation

NB! First and last path points on base and input images must be roughly at the same spot on material so that general length strain and shift could be calculated for the initial guesses

Output:

result-transform.mat

* base_grid (2xN)- points to be correlated on base image
* input_grid (2xN)- points to be correlated on input image
* base_path (2ximage width)- path that describes object deflection on base image
* input_path (2ximage width)- path that describes object deflection on input image
* base_rotations (1xN) - rotations in radians for each base grid point
* rotations (1xN) - rotations in radians for each input grid point
* base_filename - absolute path to base image file
* input_filename - absolute path to input image file
* points_in_row - how many points in a grid row
* grid_step - (see input parameters)
* base_p - x,y coordinate pairs used to create base_path as chosen by user
* input_p - x,y coordinate pairs used to create input_path as defined by user

result-correlation.mat

* result (2xN)- adjusted grid points after correlation
* corr_coeff - average correlation coefficient (0-1)

After rotated_dic result has been saved correlation part can be repeated with rotated_dic_correlate

### rotated_dic_correlate(search_window, subset, display, tune_rotation)
For rerunning correlation with different input parameters after rotated_dic has run once

* subset - optional

* display - optional, 1-show images, 0-dont, defaults to 1
* tune_rotation - optional, defaults to 0

Final correlation step will save results as: 'result-correlation.mat'

Testing
----
* display - optional, 1-show images, 0-dont, defaults to 1

### test_errors(display)
Calculates and displays mean square errors for grid transform and correlation based on preset images and data in the test folder

### test_transform
Transforms rectangular grid to bent one and back to rectangular, checks errors

### test_cor_linear_errors(display)
Calculates differences between correlation results and mathematically expected final gridpoint locations. Plots errors and correlation coefficients. Input image is under 33deg rotation

### test_cor_linear_errors2(display)
Calculates differences between correlation results and mathematically expected final gridpoint locations. Plots errors and correlation coefficients. Base image is under 20deg and input under -45deg rotation. Rotation tuning is applied

### test_cor_same_errors(display)
Calculates differences between correlation results base gridpoint locations. Uses same image as base and input. Plots errors and correlation coefficients.

Limitations

- only grayscale images
- images must be same size
- only horizontal paths supported (x coordinates monotonically increasing)
- all gridpoints must be below path

construct_base_grid.m

```
% CONSTRUCT_BASE_GRID - creates grid based on given path
%
% Parameters:
%   base_path (2ximage width)- path that describes object deflection
%       on base image
%   base_p - x,y coordinate pairs used to create base_path as chosen by user
%   grid_step (int) - distance in pixels between grid points
%   rows - how many rows should grid have
%
% Returns:
%   base_grid (2xN)- points to be correlated on base image
%   base_rotations (1xN) - rotations in radians for each base grid point
%   points_in_row - how many gridpoint in one row
function [base_grid, base_rotations, points_in_row] = construct_base_grid(base_path,
base_p, grid_step, rows)
    % find first and last point on path and distance between them
    [~,idx] = min(base_p(1,:));
    first_point = base_p(:, idx);
    [~,idx] = max(base_p(1,:));
    last_point = base_p(:, idx);
    p_dist = calc_dist_on_path(base_path, first_point, last_point);

    % construct rectangular grid between first and last point
    points_in_row = ceil(p_dist/grid_step);
    reference_grid = create_reference_grid(first_point, grid_step, rows, points_in_row);

    % construct straight horizontal path
    image_width = size(base_path,2);
    reference_path = [1:image_width; repmat(first_point(2), 1, image_width)];

    % transform reference grid using path on base image
    [base_grid, base_rotations] = transform_grid(reference_grid, reference_path,
base_path, base_p, [], []);

% CALC_DIST_ON_PATH - gets distance on path between two points
%
% Parameters:
%   path (2ximage width)- path that describes object deflection
%   first_point - x,y coordinates for first point
%   second_point - x,y coordinates for second point
%
% Returns:
%   distance - distance in pixels
function distance = calc_dist_on_path(path, first_point, second_point)
    [~, path_length] = iterate_path(path);
    distance = get_path_length(path_length, [first_point, second_point]);

% CREATE_REFERENCE_GRID - creates basic rectangular grid
%
% Parameters:
%   start_point - x,y coordinates where to generate grid from going right
%       and down
%   grid_step - distance in pixels between grid points
%   rows - how many rows should grid have
%   columns - how many columns should grid have
%
% Returns:
%   grid (2xN)- arrays of coordinates
function grid = create_reference_grid(start_point, grid_step, rows, columns)
    grid_start_x = start_point(1);
    grid_start_y = start_point(2) + grid_step;
    grid_end_x = grid_start_x + (grid_step) * columns-1;
    grid_end_y = grid_start_y + (grid_step) * rows-1;

    % create x coordinates for one row
    grid_x = grid_start_x:grid_step:grid_end_x;
    len_x = size(grid_x, 2);

    % create y coordinates for one column
    grid_y = grid_start_y:grid_step:grid_end_y;
    len_y = size(grid_y, 2);

    % repeat rows and columns
    grid_x = repmat(grid_x, 1, len_y);
    grid_y = repmat(grid_y', 1, len_x)';
    grid_y = grid_y(:)';

    grid = [grid_x; grid_y];
```

draw_colored_strain_grid.m

```
% DRAW_COLORED_STRAIN_GRID - show changes between distances of adjacent grid
% points using color map
%
% Parameters:
%   base_grid (2xN)- original coordinates for grid points
%   grid (2xN)- new coordinates for grid points
%   points_in_row - how many grid points are in a row
function draw_colored_strain_grid(base_grid, grid, points_in_row)
    connections = collect_grid_connections(grid, points_in_row, base_grid);

    % flip y axis and fix aspect ratio for figure
    figure('Name', 'Stress grid');
    set(gca, 'Ydir', 'reverse');
    daspect([1,1,1]);
    hold on;

    % draw colored stress-grid
    max_change = max(abs([connections.change]));
    c=colorbar;
    ylabel(c,'Change in length (%)') ;

    colorbar_range_percent = ceil(max_change * 100);
    caxis([-colorbar_range_percent colorbar_range_percent]);
    map = colormap;
    % number of distict color values on colorbar halfed
    halfMapTicks = (size(map, 1)-1)/2;
    for idx=1:size(connections, 2)
        % find color value for each grid connection
        col = round(halfMapTicks + (connections(idx).change/max_change *
halfMapTicks)) + 1;
        color = map(col,:);
        plot([connections(idx).start_p(1) connections(idx).end_p(1)],
[connections(idx).start_p(2) connections(idx).end_p(2)], 'Color', color, 'LineWidth', 3);
    end

    % draw grid points
    plot(grid(1,:),grid(2,:),'s','markerfacecolor',[0 0
0],'markeredgecolor','none','markersize',3);

% COLLECT_GRID_CONNECTIONS - find adjacent grid point pairs
%
% Parameters:
%   base_grid (2xN)- original coordinates for grid points
%   grid (2xN)- new coordinates for grid points
%   points_in_row - how many grid points are in a row
%
% Returns:
%   connections - array of structs of grid point coordinates and lenght
%   changes (see create_grid_connection)
function connections = collect_grid_connections(grid, points_in_row, base_grid)
    points_in_col = size(grid,2)/points_in_row;
    % shape grids into 2D matrices of rows and columns
    grid_mat = reshape([grid(1,:) grid(2,:)],[points_in_row,points_in_col,2]);
    base_grid_mat = reshape([base_grid(1,:) base_grid(2,:)],
[points_in_row,points_in_col,2]);

    connections = struct('start_p',{},'end_p',{},'change', {});

    for idx=1:points_in_row
       for idy=1:points_in_col
           if idx > 1
               % horizontal
               con = create_grid_connection(grid_mat(idx-1,idy,:), grid_mat(idx,idy,:),...
                       base_grid_mat(idx-1,idy,:), base_grid_mat(idx,idy,:));
               if isstruct(con)
                   connections(end+1) = con;
               end
           end
           if idy > 1
               % vertical
               con = create_grid_connection(grid_mat(idx,idy-1,:), grid_mat(idx,idy,:),...
                       base_grid_mat(idx,idy-1,:), base_grid_mat(idx,idy,:));
               if isstruct(con)
                   connections(end+1) = con;
               end
           end
       end
    end

% CREATE_GRID_CONNECTION - find adjacent grid point pairs
%
% Parameters:
%   start_p (x,y)- first point on new grid
%   end_p (x,y)- second point on new grid
%   orig_start_p (x,y)- first point on original grid
%   orig_end_p (x,y) - second point on original grid
%
% Returns:
%   connection - struct if points were valid coordinates, otherwise false
%      start_p - (see parameters)
%      end_p - (see parameters)
%      change - how much distance changed (-0.1 means -10%)
function connection = create_grid_connection(start_p, end_p, orig_start_p,
orig_end_p)
    dist = pdist([start_p(1) start_p(2); end_p(1) end_p(2)],'euclidean');
    original_dist = pdist([orig_start_p(1) orig_start_p(2); orig_end_p(1)
orig_end_p(2)],'euclidean');
    if isnan(dist) || isnan(original_dist)
       connection = false;
    else
       change = 1 - original_dist/dist;
       connection = struct('start_p', start_p, 'end_p', end_p, 'change', change);
    end
```

get_path_length.m

```
% GET_PATH - gets distance on path between first and last path point
%
% Parameters:
%   path_len (1ximage width)- current path length at any x
%   points - x,y coordinate pairs on path
%
% Returns:
%   length - path length in pixels
function length = get_path_length(path_len, points)
    start_x = round(min(points(1,:)));
    end_x = round(max(points(1,:)));
    length = abs(path_len(end_x) – path_len(start_x));
```

iterate_path.m

```
% ITERATE_PATH - finds path length and differencial for each x coordinate
%
% Parameters:
%   input_path (2ximage width)- path that describes object deflection
%       on input image
%
% Returns:
%   path_diff (1ximage width)- current path length for each x coordinate
%   path_len (1ximage width)- current path differencial for each x
%       coordinate
function [path_diff, path_length] = iterate_path(input_path)
    m=length(input_path(1,:));
    path_diff = zeros(1, m);
    path_length = zeros(1, m);

    xx=input_path(1,:);
    yy=input_path(2,:);

    length_total = 0;

    for i=1:1:m-1
        dx = xx(i+1) - xx(i);
        dy = yy(i+1) - yy(i);
        length_total = length_total + sqrt(dx^2+dy^2);
        path_diff(i) = dy;
        path_length(i+1) = length_total;
    end
```

rotated_dic.m

```
% ROTATED_DIC - Digital image correlation for deflection measurement UI
%
% Description:
%   Interactive utility for searching correlations between two images.
%   Uses two paths defined by user to transform rectangular reference
%   grid to match object deflection. Individual grid point location and
%   rotation is used to rotate image samples before correlation so that
%   same areas that have different orientation on images could be
%   correlated using conventional DIC.
%
%   Correlation can be rerun using rotated_dic_correlation function.
%
% Parameters:
%   grid_step (int) - distance in pixels between grid points (if base path
%       is nonlinear actual grid_step for a pair of grid points will
%       depend on path geometry)
%   search_window (int) - how many times the correlated sample width from
%       base image exceeds subset width
%   subset - optional, width of the sample from input image used to search correlations
%       with a sample from base image, defaults to grid step size in pixels
%   tune_rotation - optional, if 1 algorithm will try to find a rotation
%       that gives highest correlation. Improves subpixel accuracy when
%       rotations calculated from path have errors, but slows down
%       execution speed considerably. defaults to 0
%
% Output:
%   result-transform.mat
%       - base_grid (2xN)- points to be correlated on base image
%       - input_grid (2xN)- points to be correlated on input image
%       - base_path (2ximage width)- path that describes object deflection
%           on base image
%       - input_path (2ximage width)- path that describes object deflection
%           on input image
%       - base_rotations (1xN) - rotations in radians for each base grid
%           point
%       - rotations (1xN) - rotations in radians for each input grid
%           point
%       - base_filename - absolute path to base image file
%       - input_filename - absolute path to input image file
%       - points_in_row - how many points in a grid row
%       - grid_step - (see input parameters)
%       - base_p - x,y coordinate pairs used to create base_path as chosen by user
%       - input_p - x,y coordinate pairs used to create input_path as defined by user
function rotated_dic(grid_step, search_window, subset, tune_rotation)

    % turn off image too big to fit screen warning
    warning('off', 'Images:initSize:adjustingMag');

    % default subset size is the grid_step
    if not(exist('subset', 'var'))
        subset = grid_step;
    end

    % by default do not try to find better rotations
    if not(exist('tune_rotation', 'var'))
        tune_rotation = 0;
    end

    % USER: get base image
    [im_base, base_filename] = load_image();
    figure('Name', 'Base image');
    imshow(im_base,'InitialMagnification',100,'displayrange',[]);

    image_width = size(im_base,2);

    % USER: create path profile for base image
    [base_path, base_p] = create_curve(image_width);
    plot(base_path(1,:), base_path(2,:));

    % create grid for base image under defined path
    while true
        rows_answer = inputdlg('How many grid rows?');
        rows = str2double(rows_answer(1));

        [base_grid,base_rotations,points_in_row]=construct_base_grid(base_path,
base_p, grid_step, rows);
        hold on;
        grid_plot = plot(base_grid(1,:),base_grid(2,:),'s','markerfacecolor',...
            [255 255 0]/255,'markeredgecolor','none','markersize',3);

        choice = questdlg('Adjust rows?','', 'Yes', 'No', 'No');
        if strcmp(choice,'No') || isempty(choice)
            break;
        end
        delete(grid_plot);
    end

    % USER: get input image
    [im_input, input_filename] = load_image();
    figure('Name', 'Input image');
    imshow(im_input,'InitialMagnification',100,'displayrange',[]);

    % USER: create path profile for input image
    [input_path, input_p] = create_curve(size(im_input,2));

    % calculate grid for input image based on two paths
    [input_grid, rotations] = transform_grid(base_grid, base_path, input_path, base_p,
input_p, base_rotations);
    hold on;
    plot(input_grid(1,:),input_grid(2,:),'s','markerfacecolor',...
        [255 255 0]/255,'markeredgecolor','none','markersize',3);

    % save results before correlation
    save('result-transform.mat', 'base_grid', 'input_grid', 'base_path', 'input_path',
'base_rotations',...
         'base_filename', 'input_filename', 'points_in_row', 'grid_step', 'rotations', 'base_p',
'input_p');

    % call actual correlation (data loaded from result-transform.mat)
    display_images=1;
    rotated_dic_correlate(search_window, subset, display_images, tune_rotation);
```

```
% LOAD_IMAGE - loads an image from disk
%
% Returns:
%   im_grid - image data
%   filename
function [im_grid, filename] = load_image()
    [file_name,path_name] = uigetfile( ...
        {'*.bmp;*.tif;*.jpg;*.tiff;*.TIF;*.BMP;*.JPG;*.TIFF;*.png',...
        'Image files (*.bmp,*.tif,*.jpg,*.tiff,*.png)';'*.*',  'All Files (*.*)'}, ...
        'Open base image');
    filename = strcat(path_name, file_name);
    im_grid = imread(filename);

% CREATE_CURVE - Create curve through coordinates user clicks on
%
% Parameters:
%   width - curve will be calculated for x values 1 to width
%
% Returns:
%   curve
%   points - x and y coordinate pairs user clicked on
function [curve, points] = create_curve(width)
    % allocate memory for two points
    x = zeros(1, 2);
    y = zeros(1, 2);

    % USER: click for path start point
    [x(1), y(1)] = ginput(1);
    hold on;
       plot(x(1),y(1),'s','markerfacecolor',[255 177
100]/255,'markeredgecolor','none','markersize',3);

    idx = 2;
    while(1)
        % USER: add points one-by-one
        [x(idx), y(idx)] = ginput(1);
        hold on
        plot(x(idx),y(idx),'s','markerfacecolor',[255 177
100]/255,'markeredgecolor','none','markersize',3);

        xx = 1:width;
        % fit points on a curve
        yy = csapi(x, y, xx);
        curve = [xx; yy];

        curve_plot = plot(xx,yy);

        choice = questdlg('Add more points to path?','', 'Yes', 'No', 'No');
        if strcmp(choice,'No') || isempty(choice)
            break;
        end

        delete(curve_plot);
        idx=idx+1;
    end
    points = [x; y];
```

rotated_dic_correlate.m

```
% ROTATED_DIC_CORRELATE - Digital image correlation for deflection measurement
%
% Description:
%   For rerunning correlation with different input parameters after
%   rotated_dic has run once
%
% Parameters:
%   search_window (int) - how many times the correlated sample width from
%       base image exceeds subset width
%   subset - optional, width of the sample from input image used to search correlations
%       with a sample from base image, defaults to grid step size in pixels
%   display - optional, 1-show images and strain plot, 0-dont, defaults to 1
%   tune_rotation - optional, if 1 algorithm will try to find a rotation
%       that gives highest correlation. Improves subpixel accuracy when
%       rotations calculated from path have errors, but slows down
%       execution speed considerably. defaults to 0
%
% Input:
%   result-transform.mat (see rotated_dic.m)
%
% Output:
%   result-correlation.mat
%       - result (2xN)- adjusted grid points after correlation
%       - corr_coeff - average correlation coefficient (0-1)
function rotated_dic_correlate(search_window, subset, display, tune_rotation)

    if exist('result-transform.mat', 'file') == 0
        error('Please run rotated_dic first to generate input');
    end

    % by default show images
    if not(exist('display', 'var'))
        display = 1;
    end

    % by default do not try to find better rotations
    if not(exist('tune_rotation', 'var'))
        tune_rotation = 0;
    end

    % load saved data
    data = load('result-transform.mat');

    if not(exist('subset', 'var'))
        subset = data.grid_step;
    end

    im_base = imread(data.base_filename);
    im_input = imread(data.input_filename);

    if display
        % display base and input images when none was found open
        plot_image('Base image', im_base, data.base_grid, data.base_path);
        plot_image('Input image', im_input, data.input_grid, data.input_path);
    end

    % tune grid location with correlation
    search_zone(1:size(data.input_grid,2)) = search_window;
    correlation_threshold = 0.5;
    tic;
    [xyinput, corr_coeff] = cpcorr_mod(data.input_grid', data.base_grid', im_input,...
        im_base, subset, search_zone', correlation_threshold, data.rotations,
tune_rotation);
    fprintf('Correlation done in %f seconds.\n', toc);
    result = xyinput';

    if display
        % plot correlated grid
        figure('Name', 'Correlatation result');
        imshow(im_input,'InitialMagnification',100,'displayrange',[]);
        hold on;
        plot(result(1,:),result(2,:),'s','markerfacecolor',[0 255
255]/255,'markeredgecolor','none','markersize',3);
    end

    fprintf('Average correlation coefficient %f\n', mean(corr_coeff));

    % save final results
    save('result-correlation.mat', 'result', 'corr_coeff');

    % plot strain grid
    if display && isfield(data,'points_in_row') && isfield(data,'grid_step')
        draw_colored_strain_grid(data.base_grid, result, data.points_in_row);
    end

% PLOT_IMAGE - displays an image with grid and path if figure with same name is not open
%
% Parameters:
%   name - figure name
%   img - image data
%   grid (2xN)- point to correlate
%   path (2ximage width)- path that describes object deflection
function plot_image(name, img, grid, path)
    if isempty(findobj('type','figure','name',name))
        figure('Name', name);
        imshow(img,'InitialMagnification',100,'displayrange',[]);
        hold on;
        plot(grid(1,:), grid(2,:),'s','markerfacecolor',[255 255
0]/255,'markeredgecolor','none','markersize',3);
        plot(path(1,:), path(2,:));
    end
```

transform_grid.m

```
% TRANSFORM_GRID - creates new grid from base_grid and given paths
%
% Description:
%   Finds new location for each base_grid point by applying geometrical
%   transformation that takes account general changes in grid displacement,
%   path length and individual path point tangent changes that given gridpoint
%   relates to. In general paths define a curve above grid that show where
%   the grid should start and end (base_p, input_p) and how the it should
%   be 'bent' to match object deflection.
%   This function is used to provide initial guesses and rotations for
%   the correlation algorithm.
%
% Parameters:
%   base_grid (2xN)- points to be correlated on base image
%   base_path (2ximage width)- path that describes object deflection
%       on base image
%   input_path (2ximage width)- path that describes object deflection
%       on input image
%   base_p - x,y coordinate pairs used to create base_path as chosen by user
%   input_p - x,y coordinate pairs used to create input_path as defined by user
%   base_rotations (1xN) - rotations in radians for each base grid
%       point
%
% Returns:
%   grid (2xN)- base_grid transformed according given paths
%   rotations (1xN) - rotations in radians for each grid point (how much
%       samples have to be rotated before correlation)
function [grid, rotations] = transform_grid(base_grid, base_path, input_path,...
                                            base_p, input_p, base_rotations)

    % preallocate memory
    len = size(base_grid, 2);
    grid_x = zeros(1, len);
    grid_y = zeros(1, len);
    rotations = zeros(1, len);

    % find path differential and current length at any given x
    [in_path_diff, in_path_length] = iterate_path(input_path);
    [~, base_path_length] = iterate_path(base_path);

    if size(input_p)>0
        % find general strain factor (how many times path lengths differ)
        len_strain = find_length_strain(base_path_length, in_path_length, base_p,
input_p);
        % correct grid startpoint when object was shifted
        shift = find_shift(base_path_length, in_path_length, base_p, input_p);
    else
        len_strain = 1;
        shift = 0;
    end

    % No rotation, base_grid is rectangular
    if size(base_rotations)==0
       base_rotations=zeros(1, len);
    end

    for idx=1:len
        % find closest point on base path for this gridpoint
        [dist_to_base_path,base_path_point_idx] = find_closest_on_path(base_path,...
            base_grid(1,idx), base_grid(2,idx));

        % how far down the path this path point occurs on both images
        len_on_base = base_path_length(base_path_point_idx) ;
        len_on_input = len_on_base*len_strain - shift;

        % find corresponding coordinate on input path
        [~, x_coord_input_path] = min(abs(len_on_input - in_path_length));
        y_coord_input_path = input_path(2, x_coord_input_path);

        % input path tangent at given coord
        diff = in_path_diff(x_coord_input_path);
        % input path rotation at given coord
        theta = atan2(diff, 1);
        % save rotation needed to be applied before correlation
        rotations(idx) = theta - base_rotations(idx);

        % final location of transformed grid point
        dy = cos(theta) * dist_to_base_path;
        dx = sin(theta) * dist_to_base_path;
        grid_y(idx) = y_coord_input_path + dy;
        grid_x(idx) = x_coord_input_path - dx;
    end

    grid = [grid_x; grid_y];

% FIND_CLOSEST_ON_PATH - finds closest point on path for given coordinates
%
% Parameters:
%   path (2ximage width) - path that describes object deflection
%   point_x - x coordinate of point
%   point_y - y coordinate of point
%
% Returns:
%   distance - distance to the closest point on path
%   idx - closest path point index in path array
function [distance, idx] = find_closest_on_path(path, point_x, point_y)
    points = [repmat(point_x, 1, size(path, 2));repmat(point_y, 1, size(path, 2))];
    euclidean_dist = sqrt(sum((path - points).^2));
    [distance, idx] = min(euclidean_dist);

% FIND_LENGTH_STRAIN - find how many times path lengths differ
%
% Parameters:
%   base_path_len (1ximage width)- current path legths for each path
%      coordinate on base path
%   input_path_len (1ximage width)- current path legths for each path
%      coordinate on input path
%   base_p - x,y coordinate pairs used to create base_path as chosen by user
%   input_p - x,y coordinate pairs used to create input_path as defined by user
%
```

```
% Returns:
%   strain - ratio between input and base path length
function strain = find_length_strain(base_path_len, input_path_len, base_p, input_p)
    base_len = get_path_length(base_path_len, base_p);
    input_len = get_path_length(input_path_len, input_p);
    strain = input_len/base_len;

% FIND_SHIFT - find difference on path length where grid starts on both paths
%
% Parameters:
%   base_path_len (1ximage width)- current path legths for each path
%      coordinate on base path
%   input_path_len (1ximage width)- current path legths for each path
%      coordinate on input path
%   base_p - x,y coordinate pairs used to create base_path as chosen by user
%   input_p - x,y coordinate pairs used to create input_path as defined by user
%
% Returns:
%   shift - length on path between base and input grid start
function shift = find_shift(base_path_len, input_path_len, base_p, input_p)
    % find current path length where base grid starts
    [~,idx] = min(base_p(1,:));
    first_base_point = base_p(:, idx);
    base_start_len = base_path_len(round(first_base_point(1,1)));

    % find current path length where input grid starts
    [~,idx] = min(input_p(1,:));
    first_input_point = input_p(:, idx);
    input_start_len = input_path_len(round(first_input_point(1,1)));

    shift = base_start_len - input_start_len;
```

findpeak.m

```
function [xpeak, ypeak, max_f] = findpeak(f,subpixel)
%FINDPEAK Find extremum of matrix.
%   [XPEAK,YPEAK,MAX_F] = FINDPEAK(F,SUBPIXEL) finds the extremum of F,
%   MAX_F, and its location (XPEAK, YPEAK). F is a matrix. MAX_F is the maximum
%   absolute value of F, or an estimate of the extremum if a subpixel
%   extremum is requested.
%
%   SUBPIXEL is a boolean that controls if FINDPEAK attempts to estimate the
%   extremum location to subpixel precision. If SUBPIXEL is false, FINDPEAK
%   returns the coordinates of the maximum absolute value of F and MAX_F is
%   max(abs(F(:))). If SUBPIXEL is true, FINDPEAK fits a 2nd order
%   polynomial to the 9 points surrounding the maximum absolute value of
%   F. In this case, MAX_F is the absolute value of the polynomial evaluated
%   at its extremum.
%
%   Note: Even if SUBPIXEL is true, there are some cases that result
%   in FINDPEAK returning the coordinates of the maximum absolute value
%   of F:
%   * When the maximum absolute value of F is on the edge of matrix F.
%   * When the coordinates of the estimated polynomial extremum would fall
%     outside the coordinates of the points used to constrain the estimate.

%   Copyright 1993-2004 The MathWorks, Inc.
%   $Revision $  $Date: 2004/10/20 17:54:47 $

% get absolute peak pixel
[max_f, imax] = max(abs(f(:)));
[ypeak, xpeak] = ind2sub(size(f),imax(1));

if ~subpixel || ...
    xpeak==1 || xpeak==size(f,2) || ypeak==1 || ypeak==size(f,1) % on edge
    return % return absolute peak

else
    % fit a 2nd order polynomial to 9 points
    % using 9 pixels centered on irow,jcol
    u = f(ypeak-1:ypeak+1, xpeak-1:xpeak+1);
    u = u(:);
    x = [-1 -1 -1  0  0  0  1  1  1]';
    y = [-1  0  1 -1  0  1 -1  0  1]';

    % u(x,y) = A(1) + A(2)*x + A(3)*y + A(4)*x*y + A(5)*x^2 + A(6)*y^2
    X = [ones(9,1),  x,  y,  x.*y,  x.^2,  y.^2];

    % u = X*A
    A = X\u;

    % get absolute maximum, where du/dx = du/dy = 0
    x_offset = (-A(3)*A(4)+2*A(6)*A(2)) / (A(4)^2-4*A(5)*A(6));
    y_offset = -1 / ( A(4)^2-4*A(5)*A(6))*(A(4)*A(2)-2*A(5)*A(3));

    if abs(x_offset)>1 || abs(y_offset)>1
        % adjusted peak falls outside set of 9 points fit,
        return % return absolute peak
    end

    % return only one-thousandth of a pixel precision
    x_offset = round(1000*x_offset)/1000;
    y_offset = round(1000*y_offset)/1000;

    xpeak = xpeak + x_offset;
    ypeak = ypeak + y_offset;

    % Calculate extremum of fitted function
    max_f = [1 x_offset y_offset x_offset*y_offset x_offset^2 y_offset^2] * A;
    max_f = abs(max_f);

end
```

cpcorr_mod.m

```
function [xyinput,corr_coeff] = cpcorr_mod(varargin)
%CPCORR Tune control point locations using cross-correlation.
%   INPUT_POINTS = CPCORR(INPUT_POINTS_IN,BASE_POINTS_IN,INPUT,BASE) uses
%   normalized cross-correlation to adjust each pair of control points
%   specified in INPUT_POINTS_IN and BASE_POINTS_IN.
%
%   INPUT_POINTS_IN must be an M-by-2 double matrix containing the
%   coordinates of control points in the input image.  BASE_POINTS_IN is
%   an M-by-2 double matrix containing the coordinates of control points
%   in the base image.
%
%   CPCORR returns the adjusted control points in INPUT_POINTS, a double
%   matrix the same size as INPUT_POINTS_IN.  If CPCORR cannot correlate a
%   pairs of control points, INPUT_POINTS will contain the same coordinates
%   as INPUT_POINTS_IN for that pair.
%
%   CPCORR will only move the position of a control point by up to 4
%   pixels.  Adjusted coordinates are accurate up to one tenth of a
%   pixel.  CPCORR is designed to get subpixel accuracy from the image
%   content and coarse control point selection.
%   NOTE:  EJ modification:  CPCORR_MOD will adjust the control point by
%   more than 4 pixels, depending on the subset size!!
%
%   Note that the INPUT and BASE images must have the same scale for
%   CPCORR to be effective.
%
%   CPCORR cannot adjust a point if any of the following occur:
%     - points are too near the edge of either image
%     - regions of images around points contain Inf or NaN
%     - region around a point in input image has zero standard deviation
%     - regions of images around points are poorly correlated
%
%   Class Support
%   -------------
%   The images can be numeric and must contain finite values. The input
%   control point pairs are double.
%
%   Example
%   --------
%   This example uses CPCORR to fine-tune control points selected in an
%   image.  Note the difference in the values of the INPUT_POINTS matrix
%   and the INPUT_POINTS_ADJ matrix.
%
%       input = imread('onion.png');
%       base = imread('peppers.png');
%       input_points = [127 93; 74 59];
%       base_points = [323 195; 269 161];
%       input_points_adj = cpcorr(input_points,base_points,...
%                                 input(:,:,1),base(:,:,1))
%
%   See also CP2TFORM, CPSELECT, NORMXCORR2, IMTRANSFORM.

%   Copyright 1993-2011 The MathWorks, Inc.
%   $Revision: 1.16.4.10 $  $Date: 2011/08/09 17:49:27 $

%   Input-output specs
%   ------------------
%   INPUT_POINTS_IN: M-by-2 double matrix
%             INPUT_POINTS_IN(:)>=0.5
%             INPUT_POINTS_IN(:,1)<=size(INPUT,2)+0.5
%             INPUT_POINTS_IN(:,2)<=size(INPUT,1)+0.5
%
%   BASE_POINTS_IN: M-by-2 double matrix
%             BASE_POINTS_IN(:)>=0.5
%             BASE_POINTS_IN(:,1)<=size(BASE,2)+0.5
%             BASE_POINTS_IN(:,2)<=size(BASE,1)+0.5
%
%   INPUT:   2-D, real, full matrix
%            logical, uint8, uint16, or double
%            must be finite (no NaNs, no Infs inside regions being correlated)
%
%   BASE:    2-D, real, full matrix
%            logical, uint8, uint16, or double
%            must be finite (no NaNs, no Infs inside regions being correlated)


[xyinput_in,xybase_in,input,base,subset,search_zone,thresh,rotations,tune_rotation] =
ParseInputs(varargin{:});

CORRSIZE = subset/2;
ncp = size(xyinput_in,1);

% sample under 45deg rotation - SS
INPUT_CORRSIZE = ceil(sqrt(CORRSIZE^2 + CORRSIZE^2));

% get all rectangle coordinates
% Reveresed input and base templates - SS
rects_input = calc_rects(xyinput_in,search_zone*CORRSIZE,input);
rects_base = calc_rects(xybase_in,ones(ncp,1)*INPUT_CORRSIZE,base);

% for croping rotated sample back to original width - SS
crop_coord = INPUT_CORRSIZE-CORRSIZE+1;
crop_rect = [crop_coord, crop_coord, subset, subset];

xyinput = xyinput_in; % initialize adjusted control points matrix
corr_coeff = zeros(size(xyinput,1),1);

for icp = 1:ncp


    %Check to see if the current point is a NaN pt
    if isnan(xybase_in(icp,1)) || isnan(xyinput_in(icp,1))
        xyinput(icp,:) = NaN;
        continue
    end

    if isequal(rects_input(icp,3:4),[0 0]) || ...
       isequal(rects_base(icp,3:4),[0 0])
        % near edge, unable to adjust
        xyinput(icp,:) = NaN;
        continue
    end

    %EJ:  New check:  Moved this check from the ParseInputs function
    if xyinput_in(icp,1)<0.5 || xyinput_in(icp,2)<0.5 || ...
       xyinput_in(icp,1)>size(input,2)+0.5 || xyinput_in(icp,2)>size(input,1)+0.5
        %Control point is outside of the image
        xyinput(icp,:) = NaN;
        continue
    end

    if xybase_in(icp,1)<0.5 || xybase_in(icp,2)<0.5 || ...
       xybase_in(icp,1)>size(input,2)+0.5 || xybase_in(icp,2)>size(input,1)+0.5
        %Control point is outside of the image
        xyinput(icp,:) = NaN;
        continue
    end

    sub_input = imcrop(input,rects_input(icp,:));

    % try to find a rotation with highest correlation
    if tune_rotation
        rotation_adjustment_rad =
adjust_rotation(rotations(icp),base,rects_base(icp,:),crop_rect,sub_input);
        rotations(icp) = rotations(icp) + rotation_adjustment_rad;
    end

    % correlate
    [amplitude,xpeak,ypeak,norm_cross_corr] =
calc_corr(rotations(icp),base,rects_base(icp,:),crop_rect,sub_input);

    if amplitude==0
        % NaN or Inf, unable to adjust
        xyinput(icp,:) = NaN;
        continue
    end

    %save the correlation coefficient:
    corr_coeff(icp) = amplitude; %EJ modification 140610

    % eliminate any poor correlations
    THRESHOLD = thresh; %EJ modification 140610

    if (amplitude < THRESHOLD)
        % low correlation, unable to adjust
        xyinput(icp,:) = NaN;
        continue
    end

    % offset found by cross correlation
    zero_disp = ceil(size(norm_cross_corr)/2); %location in the normcrosscorr that corresponds
to zero displacement
    corr_offset = [xpeak,ypeak] - zero_disp;

    % eliminate any big changes in control points
    max_disp = search_zone(icp)*CORRSIZE - CORRSIZE - 1; %EJ: use when undeformed
subset size is different from 2X deformed subset size
    ind = find(abs(corr_offset) > max_disp, 1);
    if ~isempty(ind)
        % peak of norxcorr2 not well constrained, unable to adjust
        xyinput(icp,:) = NaN;
        corr_coeff(icp) = -1;
        continue
    end

    % Compensate base grid fractional locations
    correction_rad = -rotations(icp);
    rotation_matrix = [cos(correction_rad) -sin(correction_rad); sin(correction_rad)
cos(correction_rad)];
    base_fractional_offset = round(xybase_in(icp,:)) - xybase_in(icp,:);
    base_fractional_offset = base_fractional_offset*rotation_matrix;

    % adjust control point
    xyinput(icp,:) = round(xyinput(icp,:)) + corr_offset - base_fractional_offset;

end


%-------------------------------
%
function rect = calc_rects(xy,halfwidth,img)

% Calculate rectangles so imcrop will return image with xy coordinate inside center pixel

default_width = 2*halfwidth;
default_height = default_width;

% xy specifies center of rectangle, need upper left
% upperleft = round(xy) - halfwidth; %Original line of code
upperleft = round(xy) - [halfwidth,halfwidth]; %EJ modification

% need to modify for pixels near edge of images
upper = upperleft(:,2);
left = upperleft(:,1);
lower = upper + default_height;
right = left + default_width;
% width = default_width * ones(size(upper)); %Original line of code
% height = default_height * ones(size(upper)); %Original line of code
width = default_width; %EJ modification
height = default_height; %EJ modification

% check edges for coordinates outside image
[upper,height] = adjust_lo_edge(upper,1,height);
[~,height] = adjust_hi_edge(lower,size(img,1),height);
[left,width] = adjust_lo_edge(left,1,width);
[~,width] = adjust_hi_edge(right,size(img,2),width);

% set width and height to zero when less than default size
iw = find(width<default_width);
ih = find(height<default_height);
idx = unique([iw; ih]);
```

```
width(idx) = 0;
height(idx) = 0;

rect = [left upper width height];

%-------------------------------
%
function [coordinates, breadth] = adjust_lo_edge(coordinates,edge,breadth)

indx = find( coordinates<edge );
if ~isempty(indx)
    breadth(indx) = breadth(indx) - abs(coordinates(indx)-edge);
    coordinates(indx) = edge;
end

%-------------------------------
%
function [coordinates, breadth] = adjust_hi_edge(coordinates,edge,breadth)

indx = find( coordinates>edge );
if ~isempty(indx)
    breadth(indx) = breadth(indx) - abs(coordinates(indx)-edge);
    coordinates(indx) = edge;
end

%-------------------------------
%
function
[xyinput_in,xybase_in,input,base,subset,search_zone,thresh,rotations,tune_rotation] =
ParseInputs(varargin)

% narginchk(4,5);

xyinput_in = varargin{1};
xybase_in = varargin{2};
if size(xyinput_in,2) ~= 2 || size(xybase_in,2) ~= 2
    error(message('images:cpcorr:cpMatrixMustBeMby2'))
end

if size(xyinput_in,1) ~= size(xybase_in,1)
    error(message('images:cpcorr:needSameNumOfControlPoints'))
end

input = varargin{3};
base = varargin{4};
if ndims(input) ~= 2 || ndims(base) ~= 2
    error(message('images:cpcorr:intensityImagesReq'))
end

input = double(input);
base = double(base);

%EJ New Check:
%Eliminate the check on the base and input points; instead, move this check to
%within the loop over the control points.  If a base or input point is out of the
%image, make that point not correlate.  (Note that originally, I only moved
%the check on the input points into the loop; this works if you are using
```

```
%image 1 as the reference image, and so the xybase_in are the grid points.
%But if you use the preceding image as the reference image, then the
%xybase_in points are the valid_points from the previous correlation, and
%so they have the possibility to be out of the image

subset = varargin{5};
search_zone = varargin{6};
thresh = varargin{7};

% SS - rotation data
rotations = varargin{8};
tune_rotation = varargin{9};

function [rotation_adjustment_rad] =
adjust_rotation(rotation,base,rect,crop_rect,sub_input)

    % rotation increment step in radians
    increment_rad = deg2rad(1);

    % calculate correlation coefficients for adjacent rotations around
    % expected one
    left=calc_corr(rotation - increment_rad,base,rect,crop_rect,sub_input);
    centre=calc_corr(rotation,base,rect,crop_rect,sub_input);
    right=calc_corr(rotation + increment_rad,base,rect,crop_rect,sub_input);

    % unable to correlate
    if left==0 || centre==0 || right==0
       rotation_adjustment_rad=0;
       return;
    end

    % how many rotation increments should rotation be changed
    offset = 0;
    idx = 2;

    if left>right
        % left value has higher correlation, assume peak is on left
        % rotate left by increments until peak found
        while left>centre
            right = centre;
            centre = left;
            left = calc_corr(rotation - idx*increment_rad,base,rect,crop_rect,sub_input);
            idx = idx + 1;
            offset = offset - 1;
        end
    else
        % right value has higher correlation, assume peak is on right
        % rotate right by increments until peak found
        while right>centre
            left = centre;
            centre = right;
            right = calc_corr(rotation + idx*increment_rad,base,rect,crop_rect,sub_input);
            idx = idx + 1;
            offset = offset + 1;
        end
    end
    % interpolate 3 consecutive values using second order polynomial
    x=-1:1;
    x=x';
    % y=a+bx+cx^2
    Y=[ones(3,1),x,x.^2];
    % coefficients [a,b,c]
    A=Y\[left,centre,right]';

    % find extremum
    extremum = - A(2)/(2*A(3));
    % off the 3 value bounds
    if abs(extremum)>1
        rotation_adjustment_rad=offset*increment_rad;
    else
        rotation_adjustment_rad=(offset + extremum)*increment_rad;
    end

function [amplitude,xpeak,ypeak,norm_cross_corr] =
calc_corr(rotation,base,rect,crop_rect,sub_input)
        amplitude = 0;

        % Rotate sample - SS
        sub_base = imcrop(base,rect);
        sub_base = imrotate(sub_base, -rad2deg(rotation), 'bilinear', 'crop');
        sub_base = imcrop(sub_base,crop_rect);

        % make sure finite
        if any(~isfinite(sub_input(:))) || any(~isfinite(sub_base(:)))
            % NaN or Inf, unable to adjust
            return;
        end

        % check that template rectangle sub_base has nonzero std
        if std(sub_base(:))==0
            % zero standard deviation of template image, unable to adjust
            return;
        end

        norm_cross_corr = normxcorr2(sub_base,sub_input);

        % get subpixel resolution from cross correlation
        subpixel = true;
        [xpeak, ypeak, amplitude] = findpeak(norm_cross_corr,subpixel);
```