Tracker – Documentation

Vincent Richefeu ⟨*Vincent.Richefeu@3sr-grenoble.fr*⟩
Gael Combe ⟨*Gael.Combe@3sr-grenoble.fr*⟩
Lab. 3SR, University of Grenoble, France

June 10, 2015

Abstract

This document describes how to use the code TRACKER dedicated to digital image correlation (DIC).

Contents

1 Introduction

TRACKER is a soft dedicated to 2D digital image correlation (DIC). It is written with C++ language and should easily be compiled on any operating system (linux, Windows or Mac OS X). TRACKER requires the library `libraw` to read raw images, and the library `ImageMagick` (to read tiff, png, bmp, and so on) can be used or not. The compilation consists simply to type `make` in the source folder.

! → This code is *not* free. It is designed for academic researches only.

2 The Particle Image Tracking procedure

2.1 Input

`grains_to_follow` (string)`filename`

where `filename` is the name of the input file. The format of the input (ascii) file is simply the number of point to be tracked, followed by their respective position (`x`, `y`) and radius `r` (both in pixels) on the reference image (usually the first one). If for example we want to follow 3 rods of radius 2, 5 and 8 at the respective positions (4,7), (12,15), (32,24) on the reference image, it will give:

```
3
4   7   2
12  15  5
32  24  8
```

! → the y axis is defined from top to bottom of the image. By convention we assign to the fixed points a radius equal to 2, and the corners have a radius equal to 1. This allows for the post-processing to distinguish those particular points to the sample elements in the output file (see Section **??** for the post-processing calculations)

`restart_file` (string)`filename`

where `filename` is the name an output file used as an input file. The format of this file is given in Section **??**.

`image_name` (string)`filename-format`

It is the path to the image files. The filename format should be written as in C language. For example, we can have :
`image_name /path/to/imageName%04d.tif`
which means that the images are in `/path/to` folder and have all the name of the form:
`imageName0001.tif`, `imageName0002.tif`, `imageName0003.tif`...

2.2 Multi-threading

The program uses multi-threading to get faster. Basically, the job is shared by n threads with an optimum value that depends on the computer used.

`wanted_num_thread` (integer)`number`

Set the number of threads to be used.

2.3 Image flow

A series of 4 parameters controls how the images to be processed are read.

`iref` (integer)`number`

Number of the reference image. The reference image is the image from which the computer will calculate the displacement (and rotation). Comparing all the time to the same image allows to avoid an addition of roundoff error that would appear when changing the reference.

`ibeg` (integer)`number`

Number of the image from which we start to calculate the displacement (and rotation). This number can be different from the number of the reference image for in the case where we want to analyze the data in 2 series. after having done the half, we want to analyze only the second half and then `ibeg` would be different from `iref`.

`iinc` (integer)`number`

Increment between the numbers of images processed/analyzed. It is usually set to one, however if for any reason you don't want to analyze every image but only one over 5, setting it to 5 will analyze image number 1, 6, 11, 16 ,21 and so on.

`iraz` (integer)`number`

Do not use this feature (pending development). Juste set it to a big number.

2.4 Image generation for control

With the data from correlation, it is possible to generate pgm images. They allow to check visually if the correlation is good.

`make_images` [`1` or `0`]

Enable (1) or disable (0) the generation of check-images.

`image_div` (integer)number

The dimensions (width and height) of the images created corresponds to the original dimensions divided by this number. When you are dealing with a lot of images that are heavy, the fact to create a new image with the displacement on it at each iteration can slow down significantly the program. A way to slow it down less is to consider an image with a lower size. Example: "`image_div 2`" would give images with half the original dimensions and thus a quarter in number of pixels.

2.5 Search zone, rescue and super-rescue

The program is working as follow, we define region around the center of the particle and look for the best correlation of the region considered in the new image within a search zone taking also into account the rotation. If the correlation coefficient is not high enough (`NC_min`, define by the user), then the program (if activated) use rescue or super rescue function which extended more and more the search zone in order to find the displacement of the particle.

`rescue_level` [`0` or `1` or `2`]

It is here that we say if we want to activate no rescue (0), or rescue (1), or rescue and super rescue (2). Putting it to 2 allows to have less errors.

Search zone for the first attempt, the second (rescue) and the third (super rescue), without sub-pixel : For every attempt we define how much we vary the different parameters to look for the best correlation coefficient. This has to be done in term of pixel. To evaluate what you should put, the best is to take the first and the last picture and evaluate more or less from how many pixel it is moving from one picture to the other. For the rotation of the grain it is harder to evaluate but put a value that seems reasonable for the test. The program look for the correlation coefficient in a search zone which is define by the left, right, up and down maximal displacement in pixel we are looking for.

`search_zone.left` Length in pixel which correspond to the distance until when we are looking on the left side.

`search_zone.right` Length in pixel which correspond to the distance until when we are looking on the right side.

`search_zone.up` Length in pixel which correspond to the distance until when we are looking on the up side.

`search_zone.down` Length in pixel which correspond to the distance until when we are looking on the down side.

When we look for the best correlation coefficient in the search zone, at each step, the program is also trying to see if a rotation has occurs, for that it consider a value and a number of increment of rotation.

`search_zone.inc_rot` Value of the rotation of one increment in radians. This should be carefully defined: as the program is not at that point taking into account subpixel displacement, the minimum rotation should correspond to at least a displacement of 1 pixel of the point from the pattern define previously which is the further from the center.

`search_zone.num_rot` Number of increment of rotation we look for. This should be taken considering the rotation grains are experiencing between 2 images.

This has be defined in the case of the first attempt to look for the best correlation, it means only few pixel around. However, in some case (jump of the grain), the displacement can be superior and so we use rescue and super rescue function. This functions are activated if the maximum value of the normalized correlation coefficient (so the value is between 0 and 1) is considered as too low by the user. This is define in NCC_min :

`NCC_min` Minimum value of the maximum correlation coefficient from the first attempt, under which the rescue function is activated.

`NCC_min_super` Minimum value of the maximum correlation coefficient from the second attempt (rescue), under which the super rescue function is activated.

For each of the function (rescue and super rescue), the search zone in term of displacement and rotation has also to be defined. It is exactly the same principle as the first attempt, however, the search zone should be bigger and bigger for rescue and super rescue in order to have more chance to catch the right displacement of the grain. If the maximum correlation coefficient is still not at the value expected, the program prints "super rescue fail" but take into account the highest correlation coefficient it found, in order to calculate a value of the displacement of this particle.

2.6 Shape of the pattern to be correlated

The program use a pattern around each point we want to track to correlate 2 images. The shape of the pattern can be defined as follows :

`pattern rect` ⟨dx⟩ ⟨dy⟩ Take a rectangular pattern center around the point we want to follow. Define the size of the rectangle by writing after this instruction the half size of the rectangle in pixel (integers value) in the x and y direction respectively. For example: `pattern rect 5 10` will give a rectangular pattern of dimension 11×21, that is (2.`dx`+1)×(2.`dy`+1) because we need odd number so that it is centered around the point considered.

`pattern circ` ⟨R⟩ Take a circular pattern, i.e. a disk around the point considered of radius `R` (integer in pixel). Example: `pattern circ 20`.

`pattern ring` ⟨Rin⟩ ⟨Rout⟩ Take a ring pattern, i.e. a ring around the point considered as pattern to follow. Example: `pattern ring 15 20`, both integers in pixel, rinf and rsup are the radii which define the ring which is between radius rinf and rsup.

`pattern file` ⟨MyFile⟩ It is also possible to define a special pattern we want to follow by editing a file. To activate this feature, the writting is : pattern file directory/-nameofthefile. (what about the file, how is it constructed? voir program)

2.7 Subpixel resolution of displacements

A subpixel refinement can be applied to the displacement, and consist in interpolating the image between the pixel by fitting the grey level of the image with, in our case, a spleen bicubic function. This allow to look for the correlation coefficient at the subpixel scale and by minimizing (1 - correlation coefficient) with Powell method we get the subpixel displacement.

`subpixel` ⟨0 | 1⟩ 0 for no subpixel correction, another figure (1 for example) for activation of the subpixel refinement.

`interp_half_refsize` Zone of the image which is interpolated.

`interp_mask_size` For the subpixel refinement, we redefine a mask size, that is the size of the region around the center of the particle which we are going to look for in the next image. It is usually smaller than the normal mask size.

`subpix_niter` number of iteration for the powell minimization method.

`subpix_tol` tolerance for the powell minimization method.

`initial_direction_dx`, `initial_direction_dy`, `initial_rotation_dr` In the Powell method we need to do a first guess, This three are the guess in x, y direction and for the rotation.This guess should be small enough to find the global minimum and not the local one. Can influence the result of the subpixel refinement.

`rotation` take into account the rotation (1) or not (0) during the subpixel refinement.

2.8 Output

The files resulting from the particle image tracking procedure are named 'dic_out_xxx.txt', where xxx is the number of the ??? This so-called 'dic files' are actually ASCII files with a simple format. The first value `n` corresponds to the number of tracked points, and for each point we've got:

1. refcoord_xpix (todo: add description)
2. refcoord_ypix
3. refrot
4. radius_pix
5. dx
6. dy
7. drot
8. upix
9. vpix
10. rot_inc
11. NCC
12. NCC_rescue
13. NCC_subpix

3 Synthetic image generation

Invoque TRACKER with the '-g' option. You will then be asked to enter some requested data. A series of images is generated by means of the Perlin Noise algorithm. The parameters that control the pattern are octave (an integer value), persistence and zoom (both real values).

4 Correction of distortion

There are two methods to correct the distortion: the *grid method* and the *displacement method*. Both methods try to minimize an error function that parametrizes an undistortion. This undistortion is as follows:

$$x_d = x_u... \quad (1)$$
$$y_d = y_u... \quad (2)$$
$$(3)$$

...

With the grid method, the principle is to find the distortion parameters so that the horizontal lines are parallel (and really horizontal), and the vertical lines are parallel (and really vertical). Practically, a photograph of a regular grid is shot and the positions of the line intersections are set manually (by using imagej for example). The Figure xx shows how these intersection points must be ordered (from top-left to bottom-right).

[TO BE CONTINUED]

5 Subpixel definition of particle centers

[TODO]

6 Characterization of speckle patterns

[TODO]

Figure 1: example caption

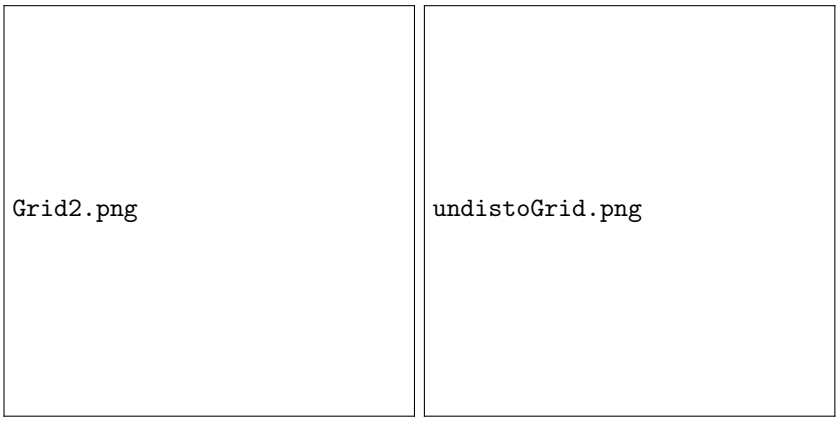

Figure 2: example caption

7 Processing tools specific to $1\gamma 2\varepsilon$ device

What we call post-processing consists basically to the following steps:

1. apply the distortion parameters to correct the pixel positions (the correction of rotations is not yet implemented),
2. set the origin of coordinates at the first fixe-point, the frame being with y-axis pointing upwards,
3. rescale the positions according to a known length in the image,
4. compute some global data such as the strain tensor, the mean value of ZNCC, etc.
5. synchronize some data from the device sensors (stress tensor...) with each configuration.