Particle Tracker DLL – 21.11.2012

LibraryRunner.h – particle tracker algorithm

`void* initRunner()`: initializes a new runner and returns it. This function has to be called before executing the particle tracker algorithm.

`void disposeRunner(void *runner)`: disposes the given runner. This function has to be called before exiting the application.

`void startRunner(void *runner)`: executes the particle tracker algorithm.

`void stopRunner(void *runner)`: stops the particle tracker algorithm.

`int isExecuting (void *runner)`: returns one if the particle tracker algorithm is in execution

`void setParameter(void *runner, char *opt, char *value)`: sets the given value to the input parameter specified (e.g.: `setParameter(runner, "p5", "200")` sets the number of images to be processed to 200).
For more parameters info see table below.

`void getParameter(void *runner, char *opt, char *value)`: copy within value the given parameter value (for input or output parameters) (e.g.: `getParameter(runner, "p11", buffer)` returns within buffer the type of convolution applied). **REMARK: make sure that `value` array size is big enough to contain parameter information.**
For more parameters info see table below.

REMARK: Parameter values can be modified / requested only when the algorithm is not running (i.e. before calling startRunner, after calling stopRunner or when isExecuting returns false).

LibraryDataReaderAPI.h – input data

Input data structure:

```
typedef struct image image;
struct image {
     int* imgData;
     // Needed to create a linked list
     image* next;
};
```

`image* getInputDataPtr(void* runner)`: called each time a new image is ready to be processed. Returns a pointer to a structure where image data have to be copied (array: `imgData`, size: imgWidth * imgHeight specified with parameters p6 and p7 before algorithm execution).

`void setInputData(void* runner, image* inputData)`: places the given image within input data buffer. This image will then be processed using the particle tracker algorithm.

`void copyInputData(image* dst, image* src)`: copy data between src and dst.

LibraryDataWriterAPI.h – ouput data

Output data structure:

```
typedef struct track track;
struct track {
     char *filename;
     int npoints;
     int *frameNo;
     double *x;
     double *y;
     double *totalSignal;
     int *cntSignal;
     double *peakSignal;
     double *meanSignal;
     double *meanBackground;
     double *meanNoise;
     double *snr;
     double *m0;
     double *m2;
     int *labels;
     // Needed to create a linked list
     track* next;
     // Maximum array size
     int maxSize;
};
```

`int hasOutputData(void* runner)`: returns one if some trajectories are available within the output data buffer

`track* getOutputData(void* runner)`: returns a pointer to a structure containing trajectories information. Info available are:

- `npoints`: number of points within the trajectory
- `frameNo`: array containing all frame ids where the particle can be found
- `x`: array containing particle’s x positions within each frame
- `y`: array containing particle’s y positions within each frame
- `totalSignal`: array containing particle’s total signal values
- `cntSignal`: array containing particle’s signal values count
- `peakSignal`: array containing particle’s peak signal values
- `meanSignal`: array containing particle’s mean signal values
- `meanBackground`: array containing particle’s mean background values
- `meanNoise`: array containing particle’s mean noise values
- `snr`: array containing particle’s SNR values
- `m0`: array containing particle’s intensity moment of order 0
- `m2`: array containing particle’s intensity moment of order 2

`void releaseOutputData(void* runner, track* outputData)`: called each time a structure containing trajectory informations can be released and overridden with new trajectory informations. **IMPORTANT: if this function is not called, each structure containing trajectory info returned by `getOutputData` function has to be manually disposed!**

`copyOutputData(track* dst, track* src)`: copy data between src and dst.

Input Parameters

- p0: Approximate radius of the particles in the images in units of pixels. The value should be slightly larger than the visible particle radius, but smaller than the smallest inter-particle separation. Must be greater than 0
- p1: The score cut-off for the non-particle discrimination. Must be non negative
- p2: The percentile (r) that determines which bright pixels are accepted as particles. All local maxima in the upper rth percentile of the image intensity distribution are considered candidate particles. Unit: percent (%). Must be greater than 0.
- p3: The maximum number of pixels a particle is allowed to move between two succeeding frames. Must be greater than 0.
- p4: The number of subsequent frames that is taken into account to determine the optimal correspondence matching. Must be greater than 1.
- p5: Number of images to be processed
- p6: Image width (must be the same for all images within the currently processed sequence)
- p7: Image height (must be the same for all images within the currently processed sequence)
- p8: Minimum value within the currently processed image sequence
- p9: Maximum value within the currently processed image sequence
- p10: (statistics) Maximum algorithm expansion radius relative to the center of the particle
- p11: (statistics) Maximum ratio between peak signal and currently analyzed point
- p12: (statistics) Signal tolerance applied to the signal value (initially the peak signal value). Sets a threshold between signal and noise/background values
- p13: (statistics) Background tolerance applied to the background value. Sets a threshold between background and noise/signal values

OutputParameters

- p14: Shows progression during input data loading and error messages when errors occur
- p15: Shows the type of convolution applied or error messages when errors occur
- p16: Shows the type of mask applied or error messages when errors occur
- p17: Shows an error message when the threshold cannot be found
- p18: Shows an error message when particles cannot be found
- p19: Shows errors occurred during position refinement
- p20: Shows errors occurred during particle discrimination
- p21: Shows an error message when particles cannot be linked
- p22: Shows a message when all trajectories has been found or error messages when errors occur
- p23: Shows messages during statistical analysis