# Summary of Algorithms used by PlanetarySystemStacker

This document is a algorithmic guide through the open-source software “PlanetarySystemStacker”. It explains all the steps performed in stacking a video file or batch of single images. For every step it lists the data structures and configuration parameters (*cursive*) involved in the operation and states where in the code the step is performed.

|  |  |  |
| --- | --- | --- |
| **Description of algorithmic step** | **Data strcuture(s) involved** | **Module.Class.Method or Module.Function**  **in Python source code** |
| All parameters controlling the program are set in the configuration object. Eventually these values are to be set by the GUI and maintained between executions in a configuration file. |  | configuration.Configuration |
| Set the names of files to process. This can either be the names of video files (type .avi) or names of directories containing image files (type .tiff). In the latter case all files of one directory are taken as input for a single stacking operation. If those images differ in shape, an error is raised.  If multiple video files or multiple directories are specified, multiple stacking operations are performed in batch mode in succession.  The workflow for the entire stacking process for a video file or image directory is controlled by the “workflow” function. Eventually this is to be replaced by GUI control. | input\_names  input\_type = “video” or “image” | main\_program.\_\_main\_\_  main\_program.workflow |
| Optional: Select a “region of interest” (ROI) in the form (y\_low, y\_high, x\_low, x\_high). If set to “None”, the full frames are used | roi | main\_program.\_\_main\_\_ |
| Optional: Select that input images should be converted to grayscale before processing. If set to “None”, color images are processed as three-channel RGB. | convert\_to\_grayscale = True or False | main\_program.\_\_main\_\_ |
| For performance measurements, code sections can be timed. If the same section is executed several times, counters can be incremented. A timer object is created at start of execution. Individual counters can be added later. In the end, a table with the accumulated times of all counters can be printed. | my\_timer | timer.timer |
| Read all frames, either from a single .avi file, or from a directory into a list structure. The shape of a single frame is (pixels in y, pixels in x [, 3 in case of color]). | frames, shape | frames.Frames.init |
| Parallel to the list of frames, three lists with different versions of the frames are computed:   * 2D monochrome image (if the original frames were monochrome, this list points to the original frame list) * “Blurred” version of the monochrome image. It is computed by applying a Gaussian filter to the monochrome image. The width of the Gaussian is an input parameter. * Laplacian of the Gaussian   The blurred image versions are used later in shift computations. This helps avoiding spurious local minima caused by pixel noise.  The Laplacians are used for ranking image quality. This happens in two locations: First in ranking the overall frame quality for constructing the mean frame, and then when the frame quality in local areas around alignment points is computed in the stacking process. | frames\_mono,  frames\_mono\_blurred  frames\_mono\_blurred\_laplacian  parameter: *frames\_gauss\_width* | frames.Frames.init  frames.Frames.add\_monochrome |
| Next all frames are ordered by their overall image quality. This is done by computing the amount of structure in the (monochrome) images. Three methods can be selected for ranking, using one of the following expressions (greater value is better) on the local luminance :   * “xy gradient”: * “Laplace”: ) * “Sobel”:   with and being the horizontal / vertical Sobel operators.   The values are normalized, so that the value for the best frame is 1.0. The index of the frame with the highest rank is computed. This frame is used as the reference for the computation of global frame shifts.  A stride value can be specified. If set to a value > 1, the images are downsampled by this value before computing the score. In typical video files setting a value of 2 usually gives a good ranking and saves compute time.  Usually the method “Laplace” is to be preferred. | frame\_ranks  frame\_ranks\_max\_index  parameters:  *rank\_frames\_method*  *rank\_frames\_pixel\_stride* | rank\_frames.frame\_score  methods:   * local\_contrast * local\_contrast\_laplace * local\_contrast\_sobel   in class  miscellaneous.Miscellaneous |
| Next all frames are aligned with each other. The frame with the highest rank is used as reference (see above). It is planned to implement two alignment algorithms: “Surface” and “Planet”.   * “Surface”: First find a rectangular patch with good structure as “alignment window”. The size of the patch is a fraction of the frame size, the scale factor being a configuration parameter. By setting a parameter “align\_frames\_border\_width” the window can be kept away from the frame borders. This way, the window does not have to be moved as often when the object is drifting between frames.  For all potential alignment patches in the frame the merit function  min(,  is executed. This way the patch is found where good vertical and horizontal structures are present. In order to ignore contributions by noise in dark areas, a brightness threshold is included.  Next all frames are compared with the reference frame at this patch, and the relative shift is computed. Three methods are available for this search: - “Translation” (not recommended): Cross-correlation  - “RadialSearch” (stable but expensive): Search all positions for a local minimum of the expression given below, spiraling out from zero shift. - “SteepestDescent” (recommended): Same as “RadialSearch”, but not all positions around zero shift are evaluated, but only those in the direction of the steepest descent of the evaluation function. More precisely:  Starting with shift values [dy\_min, dx\_min] = [0, 0], the shifted alignment window in the current frame is compared with the reference frame window. For every search position, the match quality is computed with   A “sampling stride” parameter can be selected. In this case the summation indices are incremented using this stride (to save compute time).  First, for all positions with distance 1 in y or x the match quality is computed. If the minimal value is smaller than the best value so far, the corresponding position is taken as a new start point.   For this new start point, again all positions with distance 1 are tested in search for a better minimum. The search ends when no improvement is found, or the maximum search width (parameter) is reached. In the latter case, the search is regarded as unsuccessful.  Consecutive frames tend to have similar shifts. Therefore, the optimum found for one frame is taken as start value for the next one. The current shift is kept in the cumulative shifts [dy\_min\_cum, dx\_min\_cum].  If the object drift is too large, the alignment window can hit the frame border. Before that happens, the window is shifted by half the border width away from the border. * “Planet”: Not implemented yet | Parameters:  *align\_frames\_rectangle\_ scale\_factor*  *align\_frames\_rectangle\_ black\_threshold*  *align\_frames\_search\_width*  *align\_frames\_border\_width*  *align\_frames\_sampling\_stride*  *align\_frames\_ average\_frame\_percent*  align\_frames.frame\_shifts    [dy\_min\_cum, dx\_min\_cum] | align\_frames. select\_alignment\_rect  miscellaneous.Miscellaneous. quality\_measure\_alternative  align\_frames.align\_frames  miscellaneous.Miscellaneous. translation  miscellaneous.Miscellaneous. search\_local\_match  miscellaneous.Miscellaneous. search\_local\_match\_gradient |
| After aligning all frames, the pixel bounds of the intersection of all frames are computed: intersection\_shape[y, x][low, high] | align\_frames. intersection\_shape |  |
| Next, the average frame is computed by averaging the best frames, taking into account their relative global shifts. At this point no local warp effects can be corrected for. The percentage of the total number of frames is chosen via a parameter.  From now on, pixel indexing of new image objects uses the shape of the average frame, given by the index bounds in the structure “intersection\_shape”. The original frames, however, are not copied, so they keep their original indexing. The global shifts between the image frames and the new reduced-size images are stored in lists “dy” and “dx” in the “align\_frames” object. | align\_frames.average\_frame  Parameter: *align\_frames\_average\_frame\_ percent*  align\_frames.dy / align\_frames.dx | align\_frames.average\_frame |
| If a “region of interest” was selected, the intersection is reduced to this size and position in the frame. A new mean frame is computed with the new intersection shape. |  | align\_frames.set\_roi |
| Next, the alignment points (APs) for the “multi-point alignment” are defined. The methods are contained in class “alignment\_points”. All alignment points are organized in a linear list. Each entry is a dictionary containing all information on a single point. The main variables describing an AP are:   * y, x pixel coordinates of center * Lower and upper index bounds in y and x of the so-called “alignment box”. This is the area used for measuring the local (warp) shift against the mean frame. * Lower and upper index bounds in y and x of the so-called “alignment patch”. The patch is somewhat larger than the box. It is the area used for stacking around this AP. * The “reference box” with the section of the average frame at the location of the alignment box. * The stacking buffer where frame contributions are accumulated during stacking for this alignment patch.   Two lists, included with each alignment point, deserve particular attention:   * The list “low\_structure\_neighbors” contains alignment point objects for points which during AP construction did not satisfy the structure condition (i.e. they do not show enough structure to be used in shift computations, and therefore are not included in the alignment point list). Using this list, those “failed” points are registered at the closest “real” alignment point. The idea is that in stacking a real AP, the alignment patches of the failed neighbors are stacked using the shifts determined at the “real” point. This way, holes in the stacked frame are avoided. * In analogy to the above list, the list “dim\_neighbors” contains failed AP objects which did not satisfy the brightness or contrast condition. Again, the reason is that at these points a shift cannot be computed with reasonable accuracy.   Usually, first a grid of alignment points is created automatically, using the method “create\_ap\_grid”. Later, additional points can be added or removed individually (“new\_alignment\_point” , “remove\_alignment\_point”). The following example shows the result of an automatic AP creation. The picture is created by method “show\_alignment\_points”. Red crosses show the (real) alignment points. White and green quadrats are the alignment boxes and patches around the APs, respectively.  D:\SW-Development\Python\PlanetarySystemStacker\Examples\Saturn\sat_c11_ser_F0001-1731.avi.aps_copy.jpg  The automatic AP creation produces a staggered grid of points. In rows where the first and last points are farther away from the frame boundary, alignment patches are extended up to the frame boundary (in order to avoid holes). | alignment\_points. alignment\_points | alignment\_points. create\_ap\_grid  alignment\_points. new\_alignment\_point  alignment\_points. remove\_alignment\_point  alignment\_points. find\_alignment\_points  alignment\_points. find\_alignment\_point\_neighbors  alignment\_points. show\_alignment\_points |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |