**sDaDa (Shape Dynamics Automated Data Analysis)**

**About sDaDa**

sDaDa is an open source program written in the MATLAB language, that performs shape dynamics analysis of images of membrane-labeled bacteria. The program segments the images and allows for quality control by the user and creates a rough shape model based on the microbeTracker software package. It provides accurate measurement of the width at many points the central line of the contour to achieve a shape parametrization of the bacterium. sDaDa provides analysis of labeled divisome proteins in the second channel, measuring their position, assembly and diameter. The pipeline allows to dissect cell shape parameters as a function of time (length at birth, constriction onset time, end time of the cell cycle, and constriction rate, etc.). The script was designed with a module to measure the asymmetric placement of the division site and the curvature of the long axis of the cell.

See also the description in the supplementary information of the related manuscript. sDaDa is licensed under the GPLv3. For information or support, please contact [ambroise.lambert@epfl.ch](mailto:ambroise.lambert@epfl.ch) and/or [aster.vanhecke@epfl.ch](mailto:aster.vanhecke@epfl.ch). sDaDa was written by Anna Archetti and Aster Vanhecke, based on software by Seamus Holden and MicrobeTracker (<http://microbetracker.org/>).

**Installation**

For installation, copy the files in the desired folder, and add this folder and its subfolders to the MATLAB Path (MATLAB>HOME>Set Path>Add with Subfolders…). A test dataset is provided in “sDaDa\Example dataset\”, on which the analysis can be run without changing sDaDa\_script by setting the sDaDa folder as Matlab’s current folder. To run it on your own datasets, the file names and paths need to be specified within the main script, sDaDa\_script.

**Tutorial**

**sDaDa\_script**

To run an analysis cycle, you need to set-up and run “*sDaDa\_script*”, which proceeds through all the steps of the analysis. The steps are described below and in the comments of the code, briefly: first, the experimental parameters and file- and folder names are set. Then the first part of the analysis starts: “*measureShapeDynamics*”, where segmentation and measurement of the images happens. The next part called “*analyzeShapeDynamics*”, analyzes and plots the constriction dynamics. Finally, the output is saved.

**Setting up sDaDa\_script**

Within the code in “*sDaDa\_script.m*”, change the parameters in Table 1 if necessary.

Table 1: Parameters to be edited within sDaDa\_script.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Description** | **Example (corresponding to example dataset)** |
| *tInt* | Time between consecutive frames [min] | 5 |
| *T0* | Starting time with respect to synchrony [min] | 30 |
| *pixSize* | Pixel size [nm] | 30 |
| *dataset* | Name of the dataset | ‘160713 DC WT t0 30\_’ |
| *FOVnr* | Number of the Field of View, for the filename | ‘03’ |
| *PathName* | Name of the folder holding the raw data. | ‘*C:\Documents\MATLAB\sDaDa\Example dataset\*’ |

The strings *PathName*, *dataset* and *FOVnr* are merged to form the file and folder names for both channels (default “*MTS*” and “*FtsZ*”, which can be modified in the script). The names of the input image stacks should be in the following format: *Dataset\_FOVnr\_MTS.tif* and *Dataset\_FOVnr\_FtsZ.tif*

For example: “*160713 DC WT t0 30\_01\_MTS.tif*”. The results folder is named: “*PathName\Results\FileNameFOVnr\*” and is stored in the variable “*PathNameResults*”. When these parameters are set, you can run sDaDa\_script to analyze your data.

**Segmentation and measurements: the measureShapeDynamics function**

INPUT: *FileName*, *PathName* and *PathNameResults* (described in Table 1).

This function loads and segments the membrane label images and measures the cell shape. The segmentation happens in a semi-automated fashion: cell images are shown to the user that interactively selects or corrects the cells. However, when the cell image has a high similarity to the image in the previous frame, sDaDa accepts the segmentation automatically without user intervention.

**User Interaction**

During cell segmentation, the user performs a quality control. For the first frame, this also involves choosing which cells to follow. Here the user can also decide to delete cells showing significant defect for analysis like confluent cells, bacteria not growing in the original video. For the first frame, a segmented image like Fig. 1 is shown. In the command line, the options are listed as follow:   
Cell segmentation options:

(1) Split cells

(2) Join cells

(3) Delete cells

(4) Select good cells (deletes other cells)

(5) Smooth cells

(6) Delete edge cells

(7) Undo last change

(8) Discard all changes (reset to original image

(9) Finished segmentation

Choose 1-9:

>>

Select an option by typing its corresponding number and press ‘enter’. Certain options require the user to click on the cells in the figure window displayed in Fig. 1. When finished selecting the desired cells, press ‘enter’. Use options (7) and (8) to undo the last or all changes. When ready, use option (9). There is a similar interaction step for each frame for each cell. However, when the segmented image is very similar to the previous image, manual interaction is skipped to speed up the analysis. When interacting with the single cells, there are to additional options: (0) Expand cell: when the segmenting threshold is too high and the segmented area does not include the whole cell, this option can be used to increase the segmented region. (10) Cell divided: Use this option to indicate the cell finished division and its analysis should stop before this frame, afterwards confirm with option (9). This gives the opportunity to use (7) or (8) first when option (10) is used accidentally. It often helps to look at the two-color video beforehand to determine at which frame the cell finished dividing. When a divided cell is accepted for analysis by accident (when using option (9) instead of (10)), you can no longer use options (7) and (8). However, when the segmentation/measurement by *measureShapeDynamics* is finished, the function *tooLateCorrect* can correct for this issue, provide it the numbers of the cells that need correction and the frames at which they actually divided. Use square brackets to enter multiple cells, e.g. [1 2 5].

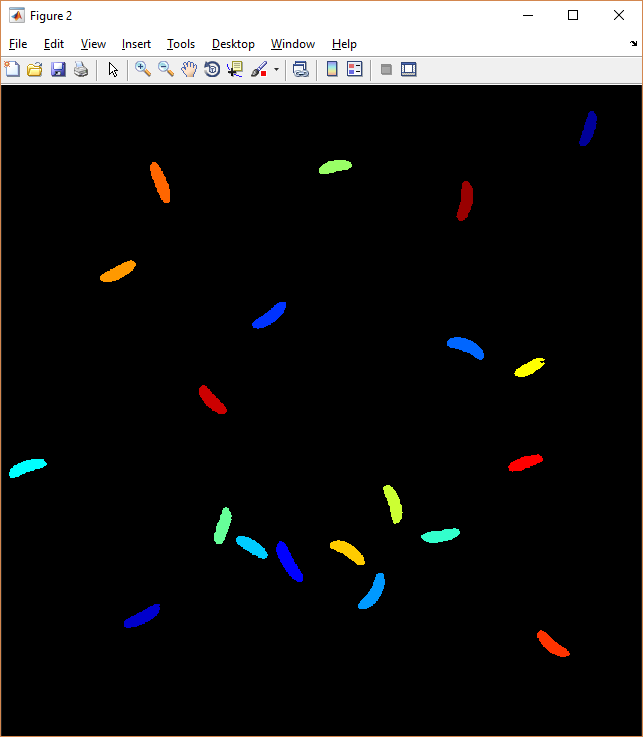


Figure 1: segmented first frame.

After the initial segmentation, the cell width is measured at regularly spaced points along the long axis of the cell, to obtain the width profile, which is plotted and saved. For two-color data with a divisome protein channel, the intensity in this channel is measured as well at equally spaced points along the long axis of the cell. The program finds the intensity maximum and measures its width. If the intensity is close to midcell and significantly brighter than the rest of the cell, in our case FtsZ or labeled divisome protein is considered to be assembled, and its width corresponds to the width of the divisome ring. These results are plotted and saved as well. Note that for the paper we manually corrected the FtsW arrival times, since these could not be robustly determined this way. The figures showing the results of the measurements are saved both in .fig and .png format in “Results\contour\” in a folder per bacterium, named *bactNframeM*, *Z\_bactNframeM*, where N is the number of the bacterium and M is the frame number. In the folder “bact0\” the first image of each bacterium is kept, to help their later identification.

Table 2 describes the output of *measureShapeDynamics*, most of which is used as the input for the next step.

Table 2: Outputs of measureShapeDynamics. Dimensions: X: image width, Y: image height, F: number of frames, C: number of cells, L: steps along length of the cell.

|  |  |  |  |
| --- | --- | --- | --- |
| Variable name | Type | Dimensions | Description |
| *ImageStack* | Matrix | X-by-Y-by-T | The raw image stack, for debugging purposes |
| *num*\_frames | Integer | 1-by-1 | number of frames, for debugging purposes |
| *divededVarEachFrm* |  |  | Array that records for each frame for each cell whether or not is has divided |
| *cellInfo* | Cell array containing cell array of structures | F-by-1,  Each frame is 1-by-C. | Based on the MicrobeTracker “cellList” format. Contains the measured parameters for each cell for each frame. Many of these parameters are self-explanatory or not used in the analysis for the manuscript. The important ones are explained below. |
| *cellInfo.bactLength* and  *cellInfo.length* | Scalar |  | Both refer to the cell length, *bactLength* was obtained via a more precise method, using the arclength by spline interpolation of the bacterial centerline, and is used in the further analysis. |
| *cellInfo.diamFWHM* | Matrix |  | The diameter profile, as measured by the Full Width at Half Max. The first row holds the position along the long axis; the second row holds the measured diameter. Units: Pixels. |
| *cellInfo.maxD* | Scalar | 1-by-1 | Maximal diameter along smoothed diameter profile (pixels). |
| *cellInfo.minD* | Scalar | 1-by-1 | Minimal diameter along diameter profile around midcell, technically between two local maxima. (Pixels.) |
| *cellInfo.waistD* | Scalar (0-1) |  | Normalized waist width, *minD*/*maxD* |
| *cellInfo.Box* | Matrix | 1-by-4 | Coordinates of region of interest: [x0, y0, width, height]. |
| *cellInfo.c0* | Scalar (0-1) |  | Coordinates of midcell, the point on the centerline corresponding to *minD*. |
| *cellInfo.ZintLength* | Matrix | 2-by-L | Intensity in the Divisome channel along the long axis. The first row holds the length along the long axis (nm), the second row holds the integrated intensity of the segment at this length. |
| *cellInfo.midcellInt* | Scalar (0-1) | 1-by-1 | Fraction of total intensity in the FtsZ channel that lies between the one- and three-quarter position along the cell long axis. |
| *cellInfo.Zprofile* | Matrix | D-by-2 | The intensity profile perpendicular to the long axis of the cell. The first column corresponds to the position along the profile (pixels), the second column corresponds to the pixel intensities (AU). |
| *cellInfo.diamZ* | Scalar | 1-by-1 | Diameter of the divisome, measured by the FWHM of *Zprofile* (nm). |

**analyzeShapeDynamics**

INPUT: *tInt, T0, pixSize, cellInfo, divededVarEachFrm, PathNameResults, diamThresh, smthThresh*

*diamThresh* and *smthThresh* are two thresholds for determining the constriction onset using the measured diameter profile. *diamThresh*, in nm, is the amount of invagination, or the difference between the maximal dimeter and minimal diameter at midcell (*maxD-minD*) at which the cell is considered as starting to constrict. For our data, a value of 90 nm was optimal to balance between sensitivity and robustness to noise. However, the method using *smthThresh* performed slightly better for our data. Here the noise is smoothed out by performing a spline fit to the normalized waist width versus time. Constriction is considered to be started when the smoothed normalized waist width falls below the value of *smthThresh*, with the optimal value for our data being 0.92.

This function analyzes and plots the measurements from the previous step. Its most important role is to find the times and corresponding lengths of the main shape dynamics events: cell birth, constriction onset and cell splitting. It also measures the curvature of the cell and the relative position of the constriction site (related to the asymmetric division). The different model fits and plots can be toggled on/off in the file “*togglePlots.m”*, a script which initializes some parameters to false or true, controlling which models to fit, what to plot etc. (described in the comments). The outputs are described in Table 3.

Table 3: Outputs of analyzeShapeDynamics. For a more detailed description, see the comments in analyzeShapeDynamics.m

|  |  |
| --- | --- |
| Variable name | Description |
| twMatrix | Matrix containing width versus time info for all cells |
| tlMatrix | Matrix containing length versus time info for all cells |
| Tvar | Matrix with Tc and Tg for each cell (measured by different models) |
| Lvar | Matrix with Lb, Lc, LG for each cell (measured by different models) |
| BIC | Matrix with Bayesian information criterion, a statistical metric to assess overfitting. |
| p1C0 | Matrix containing the position of the constriction site, measured by the MTS channel, for each time point for each cell. |
| meanR | Matrix of mean radius of curvature for each time point for each cell. |
| twZMat | Matrix containing the diameter at the constriction site measured by MTS, the width of the divisome or spot and the time, for each time point for each cell. |
| tZMat | Matrix containing information on the state of the divisome for each time point for each cell |
| TZvar | Similar to Tvar, but with Tc fixed in the model fits. |
| LZvar | Similar to Lvar, but with Tc fixed in the model fits. |
| fitResult\* | Matrices containing the fit results (fitted parameters and summed normalized residuals) for each cell (corresponding to the rows). |
| tDMatrix | Matrix containing waist diameter and maximal diameter vs time for each cell |
| alphas\_a | Fitted alpha's form Xiao model with onset time fixed to Tc. |
| tg\_frame | First frame in which the cell is divided. |

**Dependencies on MATLAB (Mathworks) toolboxes**

Table 4. lists the MATLAB toolboxes used by sDaDa. The second column lists the versions that were used for the analysis for the manuscript. Other versions have not been tested and might not give the same results or produce syntax errors.

Table : MATLAB toolboxes used by sDaDa

|  |  |
| --- | --- |
| Required Mathworks products | Version used (not necessarily required) |
| 'MATLAB' | '9.3' |
| 'Optimization Toolbox' | '8.0' |
| 'Signal Processing Toolbox' | '7.5' |
| 'Mapping Toolbox' | '4.5.1' |
| 'Image Processing Toolbox' | '10.1' |
| 'Statistics and Machine Learning Toolbox' | '11.2' |
| 'Financial Toolbox' | '5.10' |
| 'Curve Fitting Toolbox' | '3.5.6' |