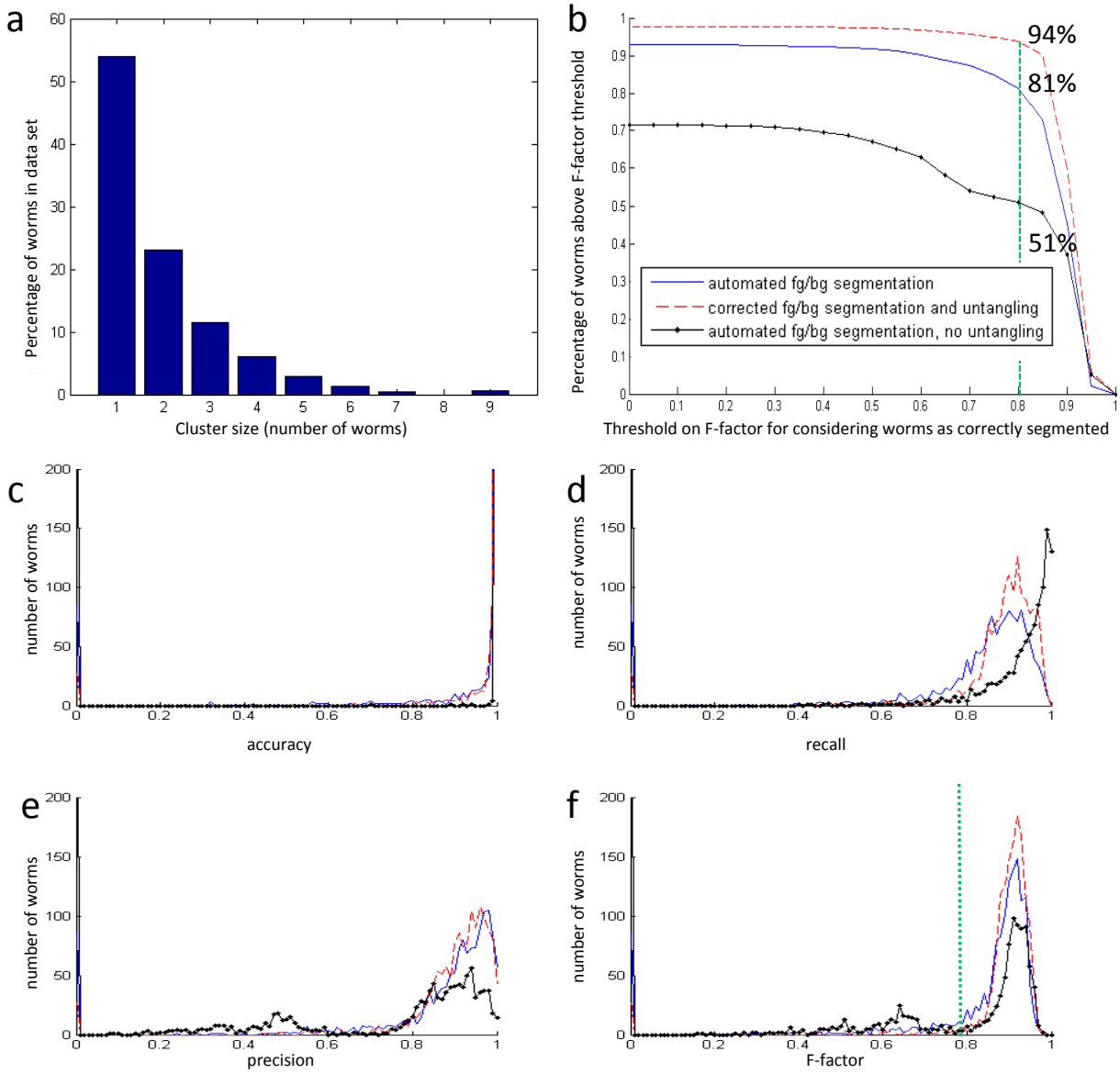
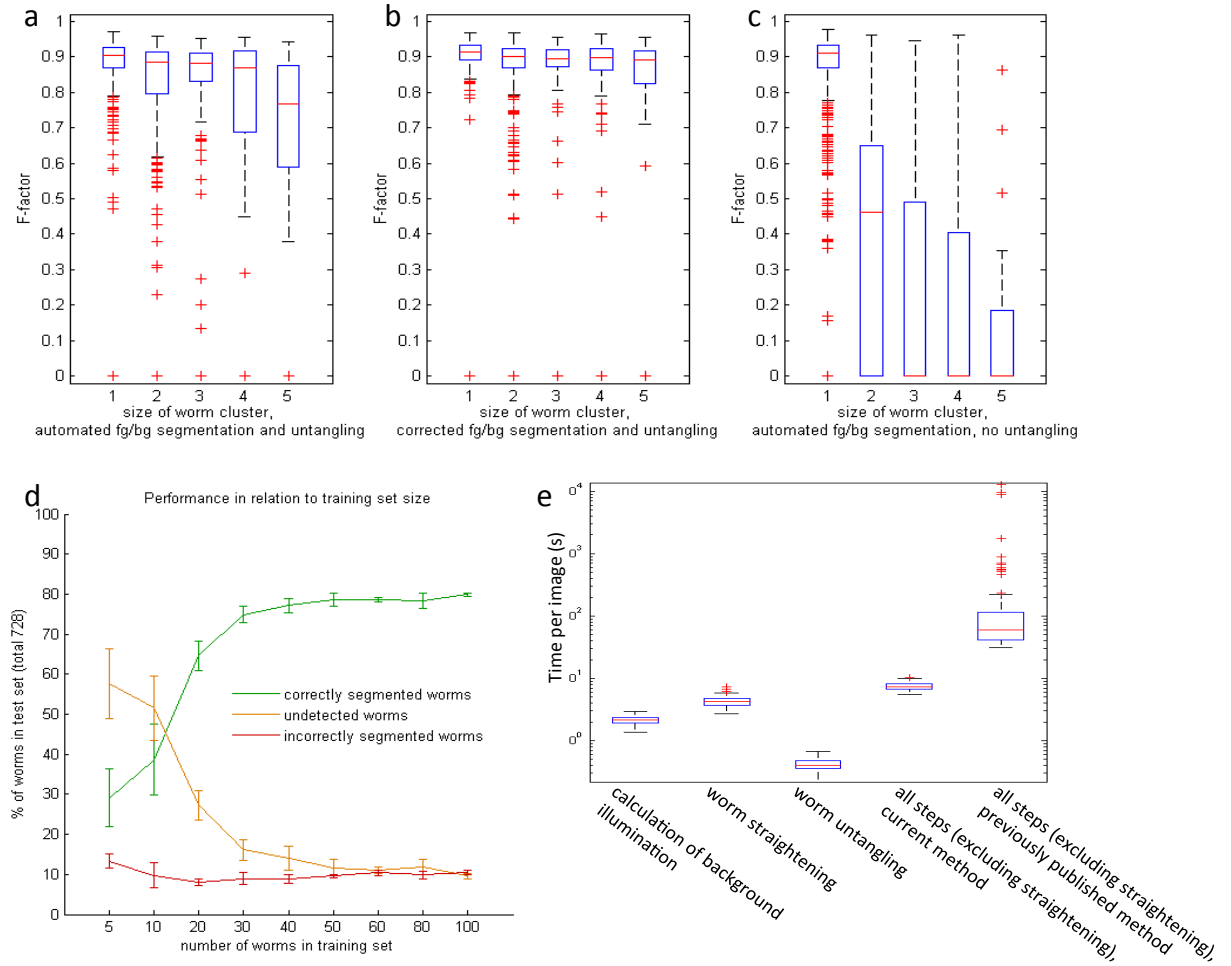
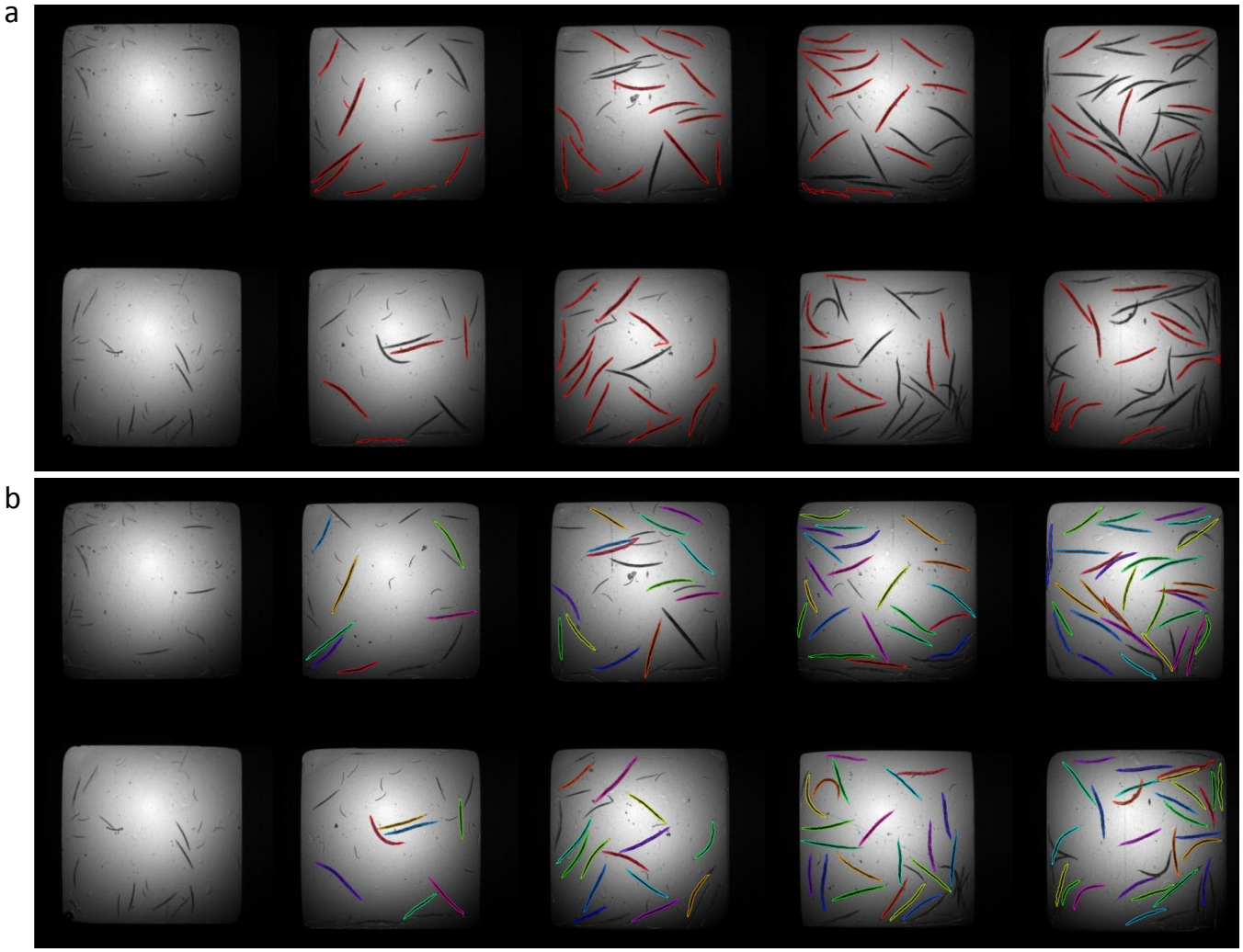


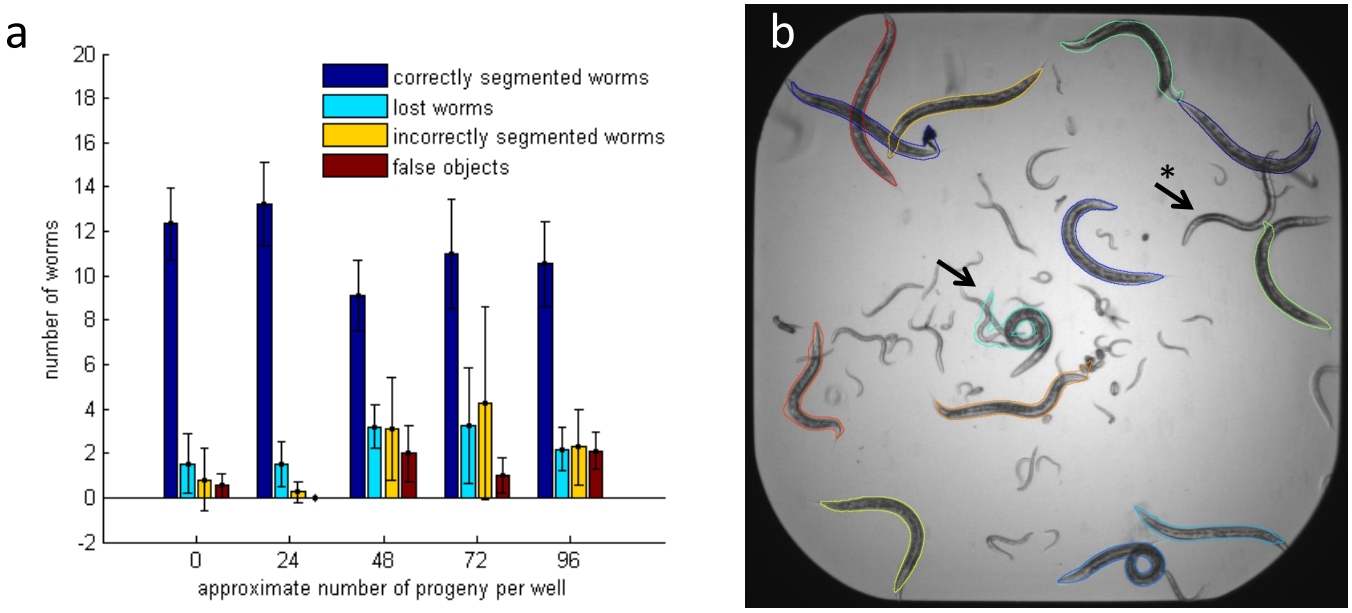
Supplementary Fig. 1: **Distribution of cluster sizes and performance of untangling.** (**a**) Based on manual delineation of 1411 worms in 100 images from a real HTS experiment, 46% of the worms are clustered. (**b**) We plotted increasing F-factor thresholds against the percentage of worms above the F-factor threshold, and defined the percentage of correctly segmented worms by an F-factor threshold of 0.8 (vertical green line), resulting in 81% correct segmentation with automated foreground/background segmentation followed by untangling, increasing to 94% with corrected foreground background segmentation followed by untangling, while only 51% of the worms are correctly segmented without untangling. (**c**) Accuracy of the initial foreground-background segmentation is crucial for the performance of the untangling, and in order to decouple errors caused by untangling from errors caused by foreground-background segmentation we evaluated the untangling twice: on binary images created by automated foreground-background segmentation (blue lines) and on images with manually corrected foreground-background segmentation (red dashed lines). We also compare to worm detection without untangling (black dash-dotted lines). We compare each ground truth worm to the best-fitting detected worm, and define accuracy as true positive area (TP) divided by false positive area (excluding overlapping regions from the false positive count). (**d**) Recall is defined as TP/total area of ground-truth worm, and (**e**) precision as TP/total area of detected worm. (**f**) The F-factor=2×(precision×recall)/(precision+recall), green vertical line represents the cutoff for correctly segmented worms, as shown in (a).



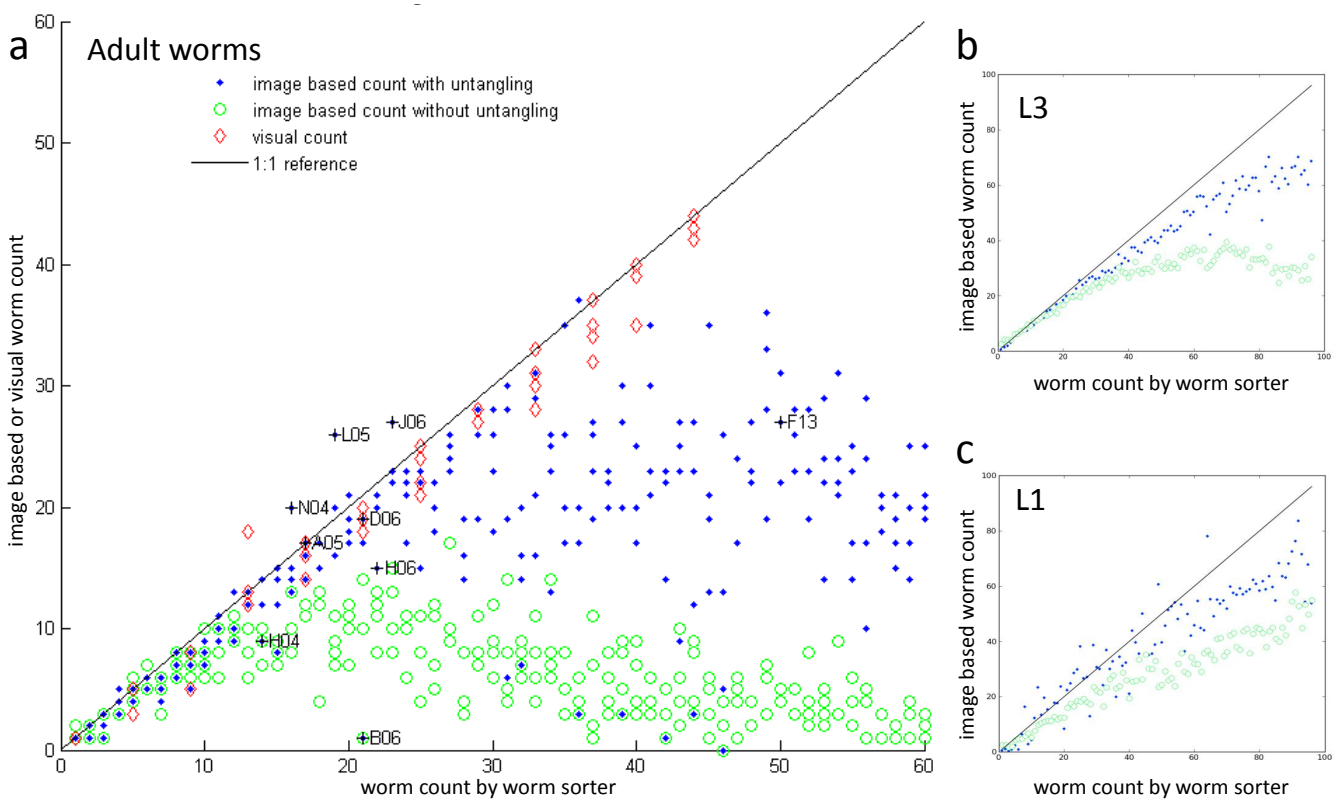
Supplementary Fig. 2: **Performance of untangling in relation to clusters size, size of training set, and speed.** (**a**) With automated foreground/background segmentation followed by untangling, the performance is reduced when clusters reach a size of about four worms. (**b**) The performance is more stable if the foreground-background segmentation is corrected. This may be due to the fact that large clusters often touch the edges of the wells where contrast is poor. (**c**) Without untangling, only non-clustered worms are correctly segmented. (**d**) Five to 100 training worms were randomly selected from a training set of 50 images, and the constructed worm models were applied to untangle a test set of 728 worms from 50 (disjoint) images. For each worm model, we counted correctly segmented worms (i.e., F-factor>0.8), undetected worms (F-factor=0), and incorrectly segmented worms (0<F-factor<=0.8). The number of correctly segmented worms (green line) plateaus at about 50 training worms. It is interesting to note that a worm model constructed from a too small training set leads to loss of worms (yellow line) rather than errors (red line) in the segmentation; if the training set is not sufficiently comprehensive true worms may be mistakenly excluded. **e** Speed of background illumination compensation, untangling, straightening and all steps (including image loading, preprocessing and untangling) are compared to all steps in our previously published approach. Median (min-max) times: calculation of background illumination: 2.2 (1.4-3)s, worm untangling: 4.2 (2.7-7)s, worm straightening: 0.4 (0.22-1)s, all steps (including image loading and thresholding but excluding straightening): 7.4 (5.5-10)s. Previous method: 59 (31.3-12773)s.



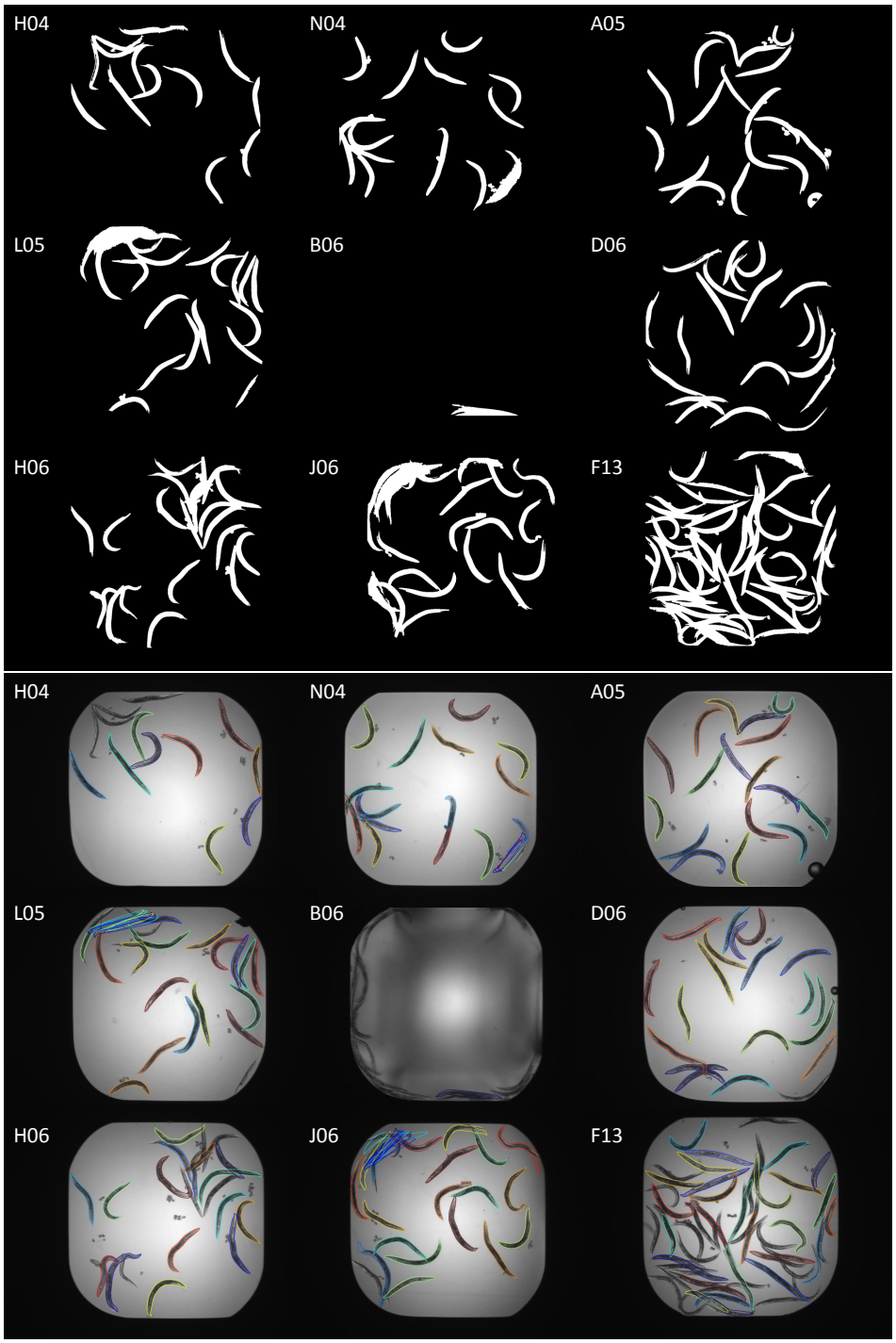
Supplementary Fig. 3: **Assay 1, part 1: Detection of individual adult worms, in the presence of clustering, eggs and progeny.** (**a**) The worm model was automatically trained on all non-touching single adult worms as filtered out based on user-specified size and shape (outlined in red). (**b**) The model was thereafter applied to the full dataset (a total of 80 images), and most worms that fall within the size/shape defined by the model are correctly untangled. Different colors are used to discriminate between outlines of different animals. Problems appear in regions with poor contrast close to the well edges. Note that small animals touching larger ones are successfully excluded by the model-based untangling.



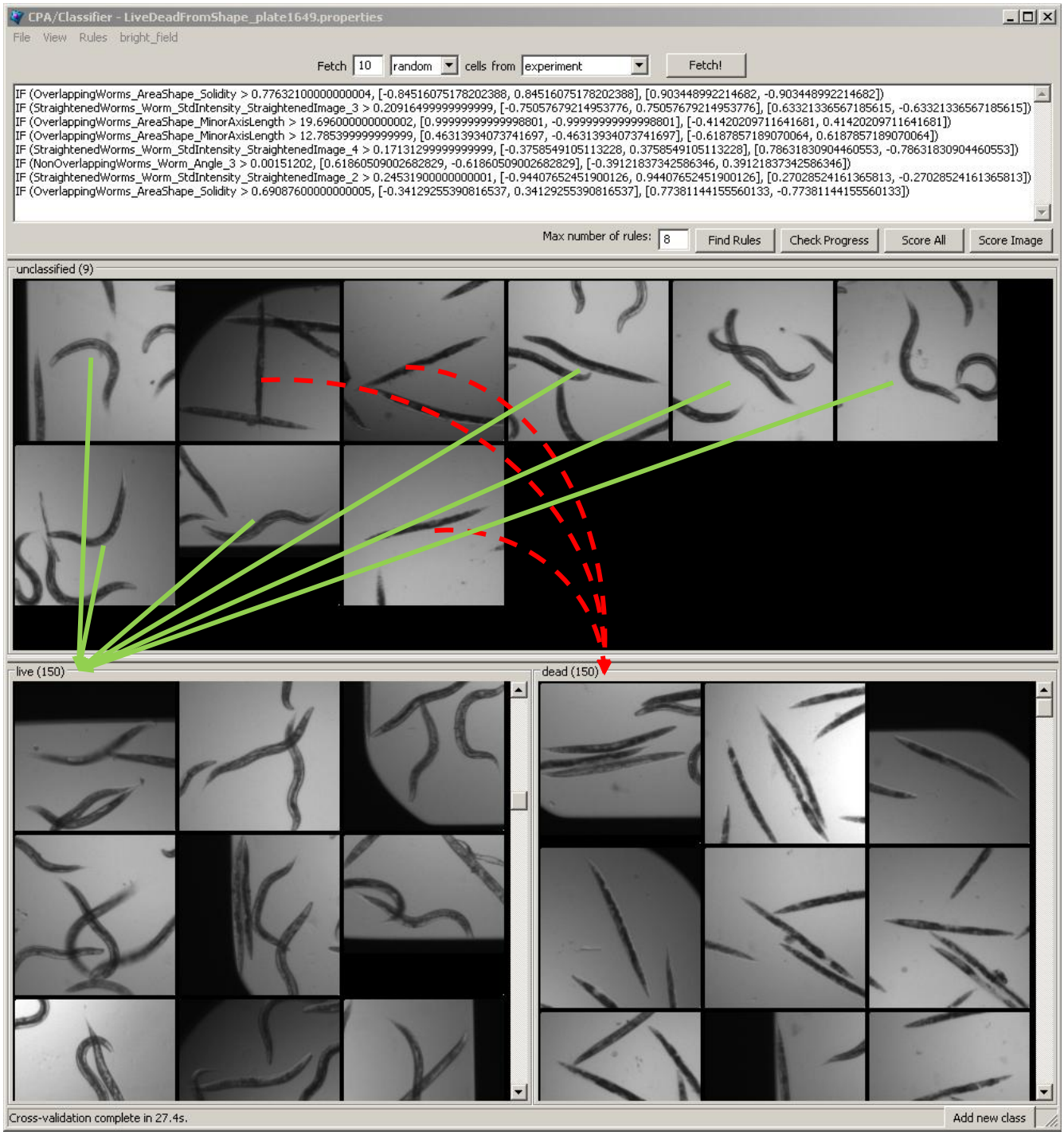
Supplementary Fig. 4: **Assay 1, part 2: Detection of adult worms at increasing concentration of progeny.** A COPAS worm sorter was set up to seed 15 adult worms per well in a 384 well plate, and add an increasing number of progeny, from 1-96, in four replicates, to each well. The worm model was trained on a subset of non-touching adult worms manually selected from the first 20 images of the experiment. The model was thereafter applied to the full dataset. (**a**) Three types of errors occur: worms are lost, worms are incorrectly segmented, or false objects, such as clusters of progeny, are detected as worms. Mean and standard deviation (n=4) in the number of adult worms found per well based on image analysis. More false objects are detected at higher concentration of progeny. The worm untangling is relatively robust in the prescence of up to 6-fold more progeny than adults, at least for detecting morphological phenotypes that do not rely on precise counting. (**b**) A well with 15 adults and 72 progeny as sorted by COPAS. Example of incorrect segmentation due to clusters of progeny (black arrow), and a lost worm (black arrow with star), likely due to small worm size as compared to those worms used for model training. Clumped worms should easily be removed at data analysis by machine learning.



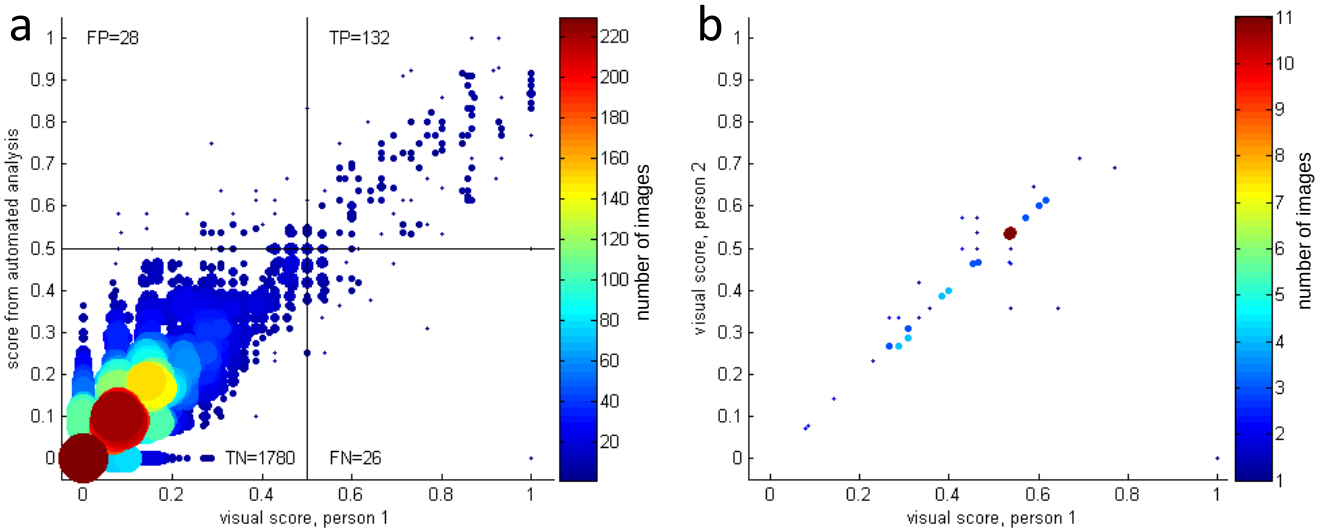
Supplementary Fig. 5: **Assay 1, part 3: Detection of individual worms in either adult or larval stage L1 or L3.** A COPAS worm sorter was set up to seed 1-96 L1, L3, or adult worms per well in a 384 well plate, in four replicates. The worm model was trained on a subset of non-touching worms (larval stage L1, L3 or adult respectively) manually selected from the first 20 images of each experiment. The model was thereafter applied to the full dataset (a total of 384 images for each stage). Each dot in the scatter plot represents the number of worms found per well based on image analysis as compared to the number of worms seeded by the worm sorter. When worm untangling fails, worms are missed and the worm count decreases. (**a**) Worm counts from wells containing 1-60 adult worms show that the untangling (blue dots) works well until about 20 worms per well. Without untangling (green circles), the worm count is under-estimated already at 10 worms. Note that variation in COPAS worm counting accuracy as compared to visual counting (red diamonds) is observed. In order to explore the modes of failure, we examined a subset of segmentation results visually (marked with black + and well name), as shown in **Supplementary Fig. 6**. (**b**) For L3 worms, the untangling works well until reaching about 30 worms per well, which is to be expected as there is more space per worm in the well. (**c**) For the small L1 worms, the image resolution only allows a few pixels per worm, making the model-based worm untangling unstable, particularly in the presence of of small bubbles which can be confused with small worms at this resolution.



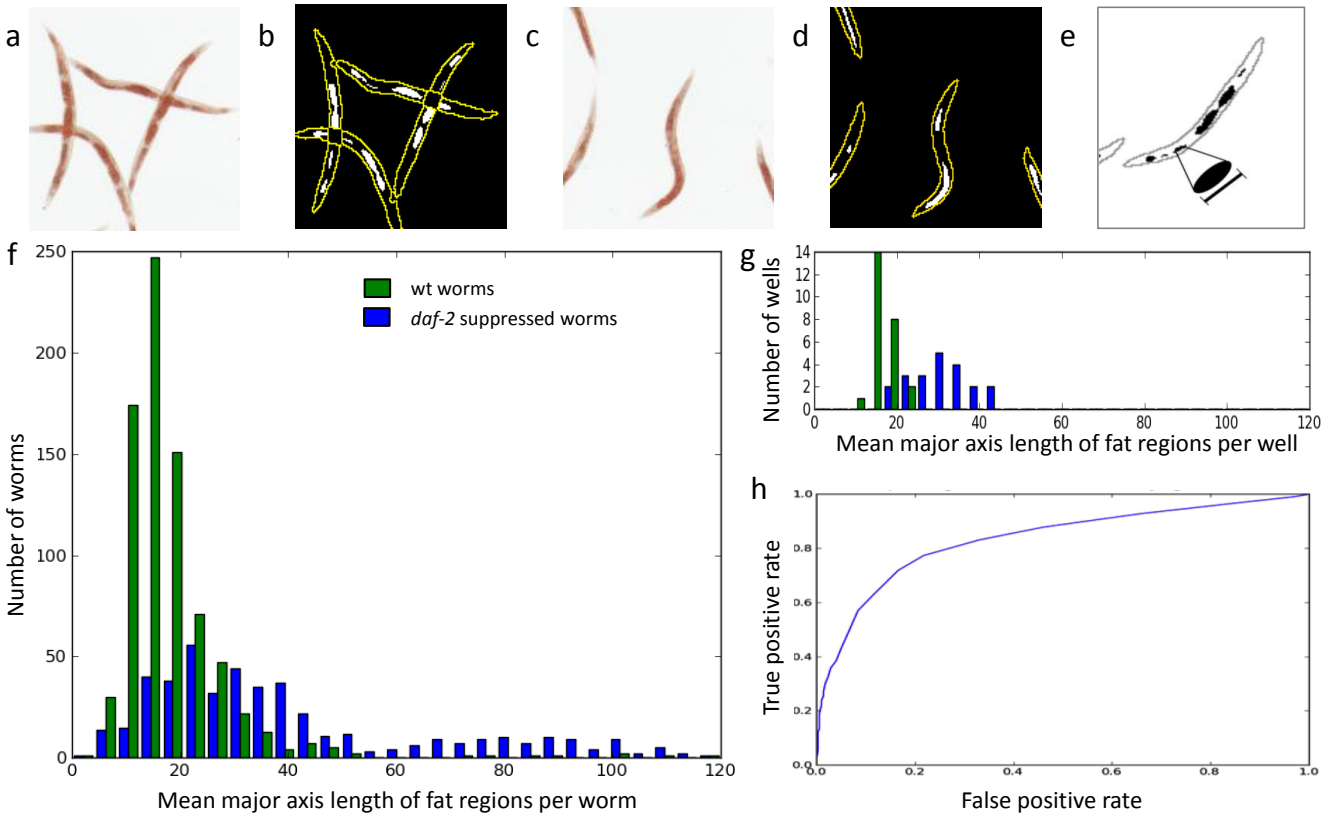
Supplementary Fig. 6: **Assay 1, part 4: Examples of modes of failure.** A subset of images from the experiment described in **Supplementary Fig. 5** were selected to illustrate the modes of failure of worm untangling. Image labels correspond to well names, as also shown in the scatter plot of **Supplementary Fig. 5a**. **(top panel)** Binary images used as input for worm untangling, and **(bottom panel)** results of worm untangling overlaid on original bright field images. Images sorted based on number of dispensed worms. The number of worms have been overestimated in wells N04, L05, and J06 as due to false clumps of worms in the binary image originating from sub-optimal background illumination correction. The worm segmentation in wells A05 and D06 was successful, while the number of worms were underestimated in well H04 due to poor contrast, B06 due to a fault in the plate, and F13 due to high worm density.



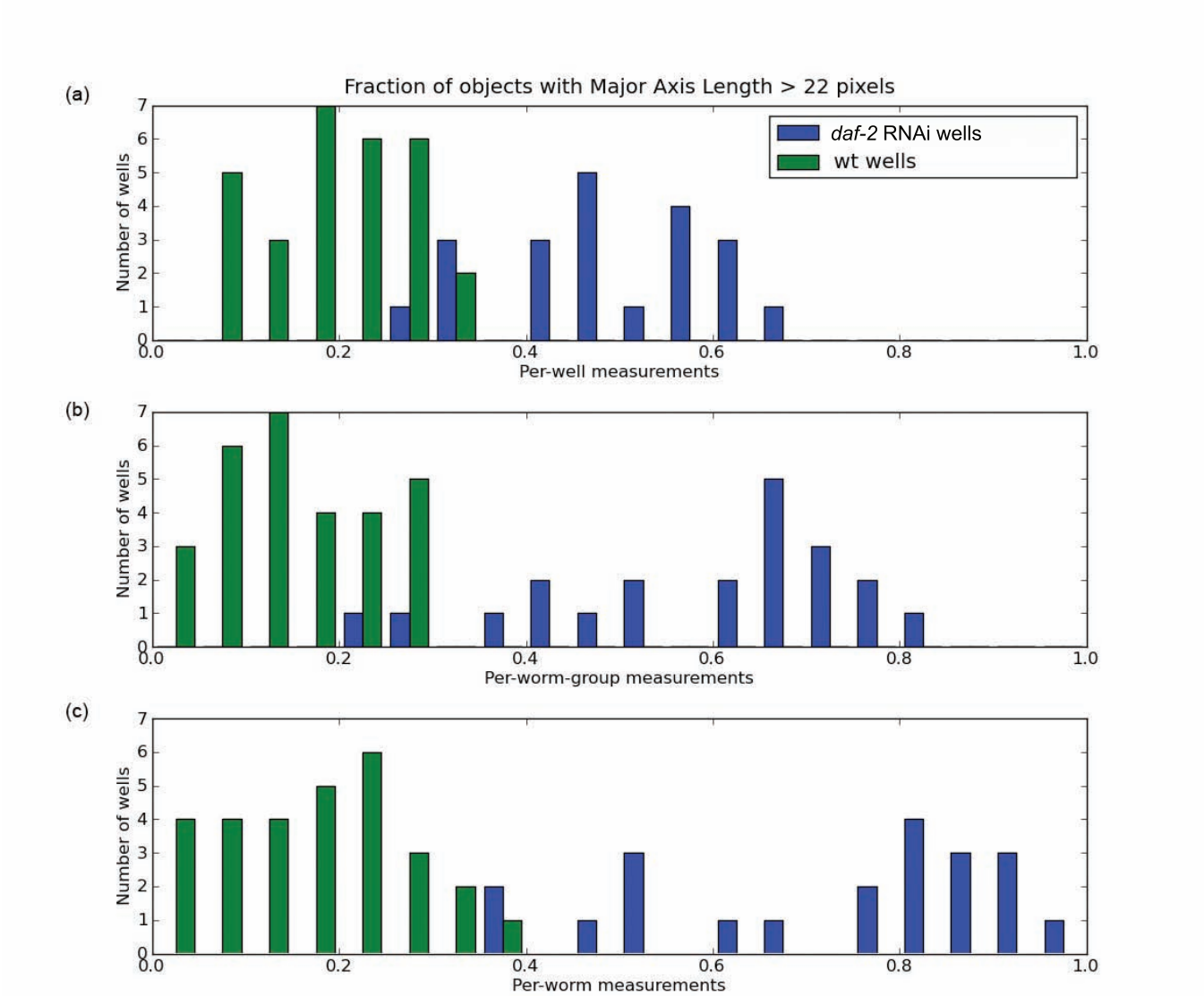
Supplementary Fig. 7: **Assay 2, part 1: Live/dead scoring based on bright field morphology.** A large number of features, such as intensity, texture and shape, were extracted from each delineated worm from bright field images. The machine learning capabilities of CellProfiler Analyst were thereafter applied to define rules separating the live and the dead phenotype. Live (green, straight arrows) and dead samples (red curved dashed arrows) are manually dragged and dropped into corresponding bins. The worm crossing the center of each image square is the one being selected for training. A limited number of rules (top panel) differentiating between the two bins are defined. The features used by the classifier include shape solidity, standard deviation of intensity, and minor axis length. All training samples (a total of 150 worms from each of live and dead samples) were disjoint from the final test set.



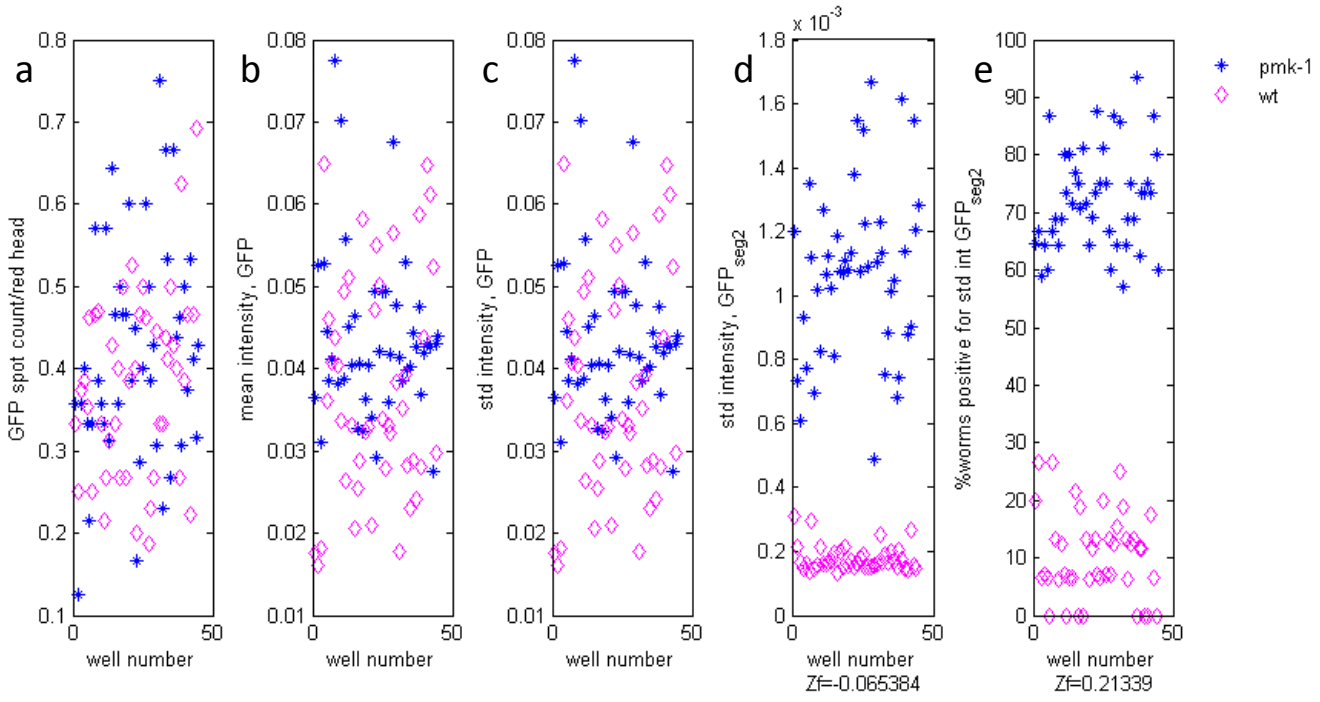
Supplementary Fig. 8: **Assay 2, part 2: Live/dead scoring of images from a large HTS experiment.** In order to test the performance of the untangling in a large scale experiment we selected ˜2000 images from the previously published screen by Moy et al[12]. The total experiment included 37,200 chemicals and ˜130,000 images, and in order to get a representative selection of images we randomly selected 1766 images from the complete experiment, and added 200 images scored as hits in the original screen. As images were randomly selected from the full screen, we believe they represent the variability and heterogeneity of a full scale screen. The same worm model and pipeline parameters were used for the full experiment, and using a computer cluster the analysis was completed in less than 15 minutes, as compared to the 20 hours needed for visual scoring, spread out over 2 weeks as visual scoring is very exhausting. A total of 25 dead and 25 live worms were visually selected from a random subset of the 2000 images (a total of 30 000 worms) to train the classifier of CellProfiler Analyst for the live and dead phenotypes. (**a**) The fraction of live worms as compared to total number of worms found by visual and automated scoring. TP=true positives, TN=true negative, FP=false positives, and FN=false negatives. The accuracy of the assay, (TP+TN)/total, is 97%, and the precision, TP/(TP+FP), is 83%. (**b**) A subset of the images (all FP and FN) were visually scored by a second person to access the inter-observer variability.



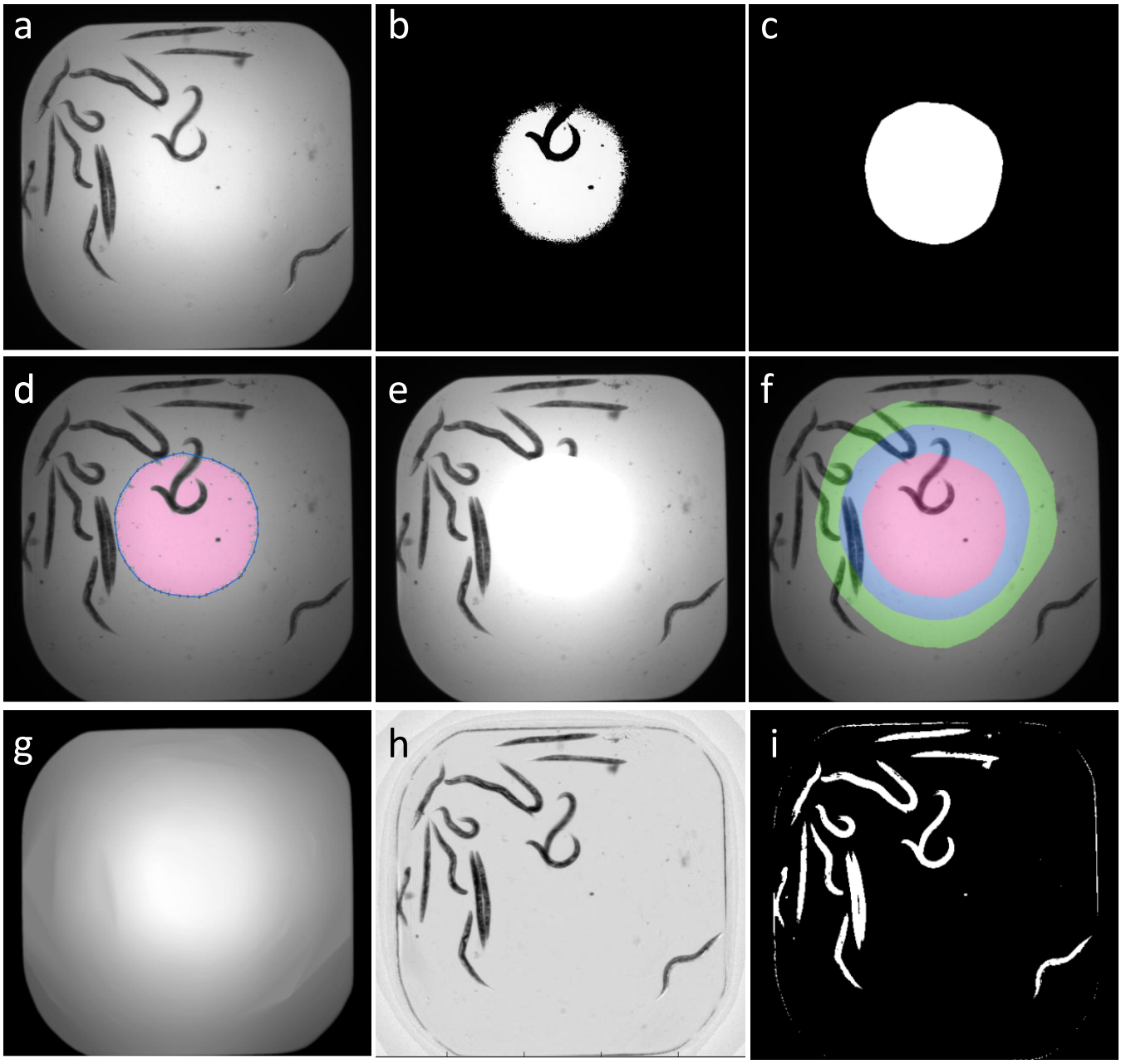
Supplementary Fig. 9: **Assay 3, part 1:Fat accumulation scoring based on oil red O staining pattern.** (**a**) Oil red O stained mock-treated worms are imaged with bright field microscopy and (**b**) untangled, worm outlines in yellow. Fat regions are defined by automated intensity thresholding of the blue image channel, resulting regions shown in white. (**c**) Corresponding images for the fat *daf-2(RNAi)* phenotype, where (**d**) fat regions are typically longer. (**e**) The assay is scored using the pattern of the fatty regions, and fatness of the worm is determined by measuring the mean major axis length of fat regions within each worm. Major axis length is defined as the length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region. (**f**) When measurements are made on a per-worm basis, the "fat" *daf-2(RNAi)* worms (blue) show a bimodal measurement distribution, with one of the modes overlapping that of the mock-treated wild-type worms (green). (**g**) This difference is largely hidden when observing population averages. (**h**) Receiver operating characteristic for classifying individual worms based on varying the threshold for the mean major axis length measurement. We chose the point on the curve that maximizes the true positive rate (that is, the number of fat worms correctly defined as fat) while minimizing the false positive rate (wild type worms incorrectly identified as fat). This point occurs at the left-most corner of the curve and corresponds to a mean major axis length of 22 pixels. (**b**)



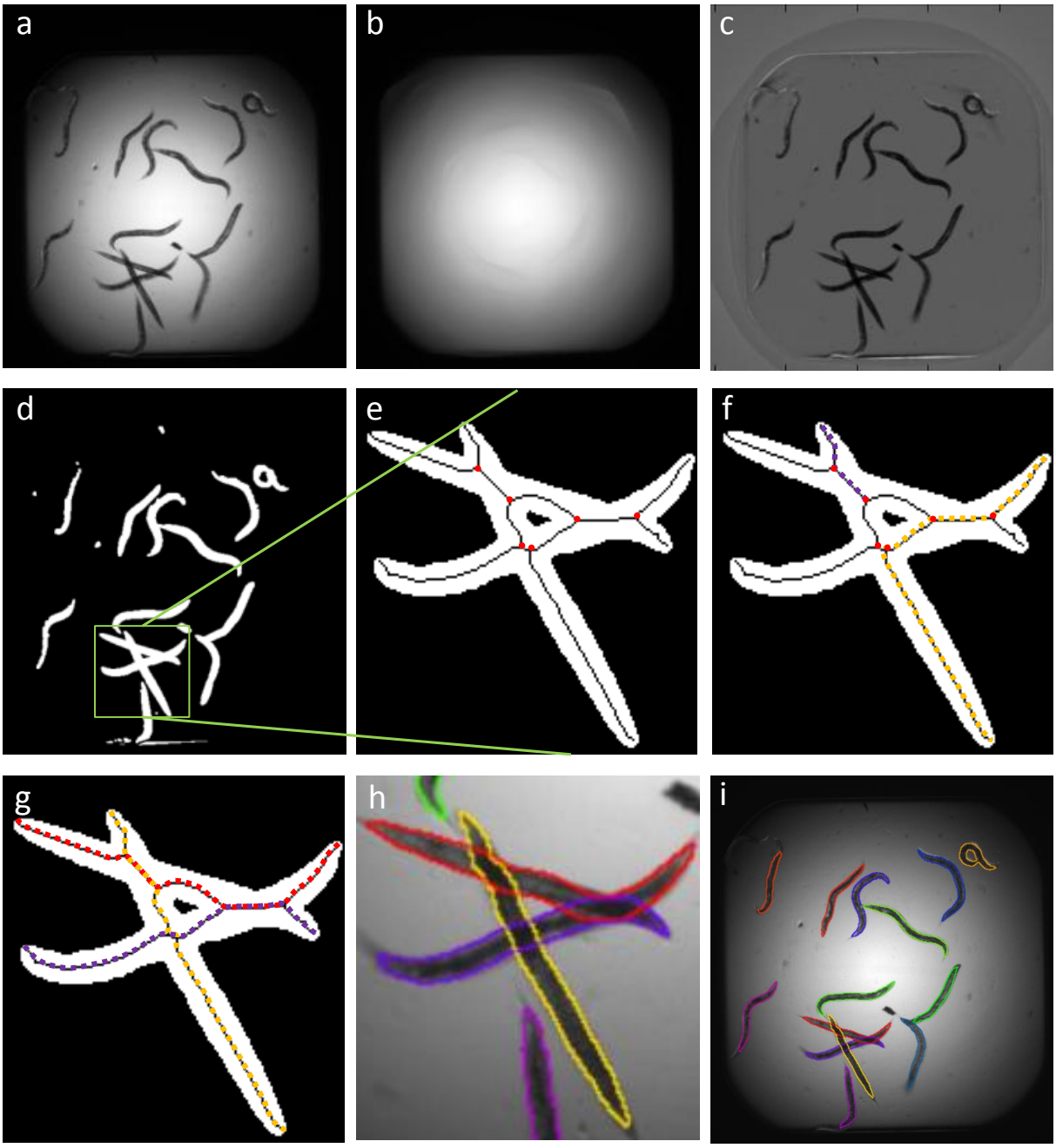
Supplementary Fig. 10: **Assay 3, part 2: Scoring of wells and population heterogeneity.** We give each well a score based on the fraction of objects with mean major axis length >22 pixels (the cutoff for fatness as computed using the ROC in **Supplementary Fig. 9**). (**a**) For per-well and (**b**) per-worm group measurements, the object is not a single worm but the entire cohort of worms and a group of worms in the well respectively. Both methods of measurement hide the true population heterogeneity visible in (**c**) per-worm measurements. In contrast, wells scored using per-worm measurements uncover that daf-2-treated worms (fat) display a range of phenotypes (due to RNAi penetrance) from wt-like to entirely fat. Due to averaging, the breadth of the phenotype is hidden in the per-well measurements: nearly all the fat wells cluster around 50% fat.



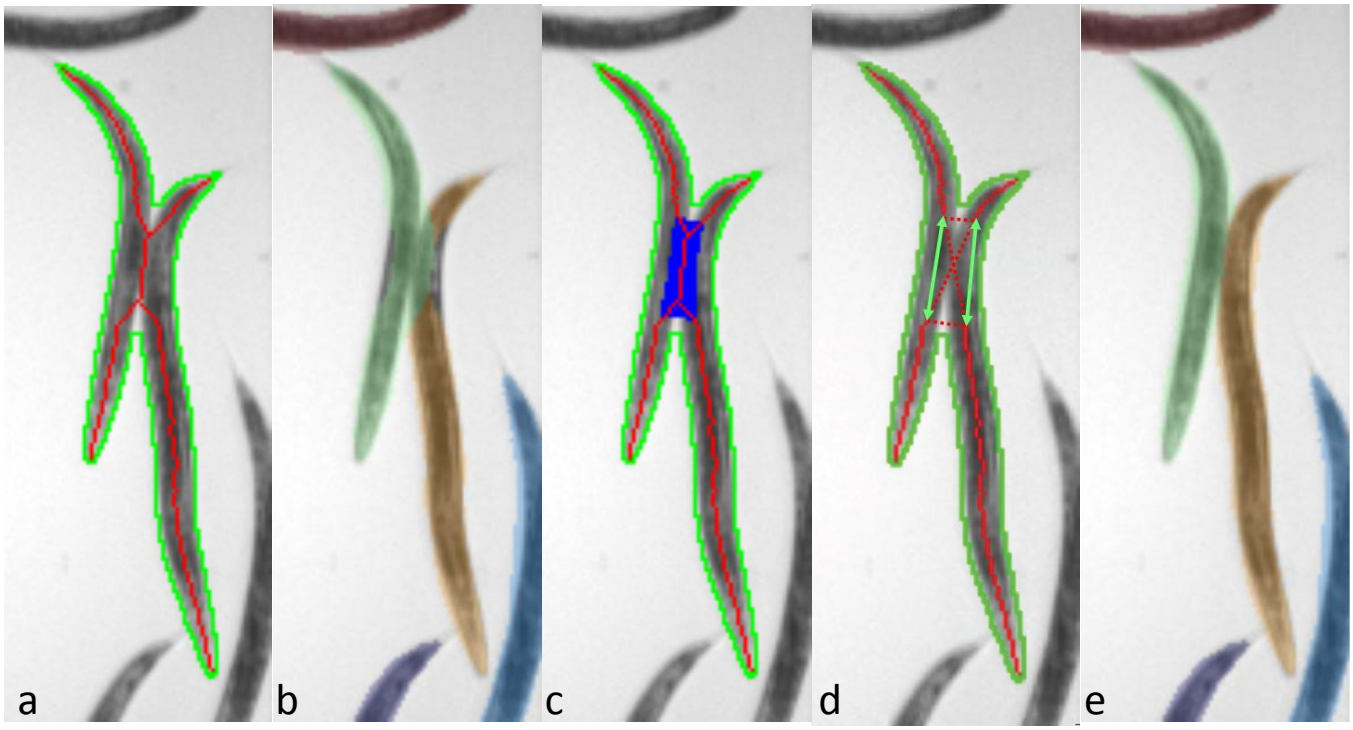
Supplementary Fig. 11: **Assay 4: Reporter pattern detection by worm straightening and atlas mapping.** Searching for a quantitative phenotype description for reporter pattern changes, we tried several different approaches. Separation of wild type animals and *pmk-1* mutants (blue stars and magenta diamonds respectively) was not feasible using (**a**) spot count or (**b**) mean or (**c**) standard deviation of intensity within worm mask. Using worm straightening and our low-resolution worm atlas to measure the standard deviation of GFP expression in segment T2of6 yielded better results (**d**), particularly when counting the percentage of worms above a threshold for that metric (**e**).



Supplementary Fig. 12: **Removing well edges and compensating illumination by a convex-hull approach.** (**a**) Input image. (**b**) Starting from the minimum intensity of the image, all pixels above a given intensity level are selected, here represented as white pixels on a black background. (**c**) The convex hull of the binary image in (b) is calculated, (**d**) here overlayed on the original image, and (**e**) all pixels in the output image that are within the convex hull are set to the current intensity level. (**f**) This procedure is iterated, here illustrated by differently colored overlays, reaching the final approximation of both illumination and well edge position shown in (**g**). (**h**) Result after dividing the input image with the approximation of the background, and (**i**) result after Otsu's thresholding of (h).



Supplementary Fig. 13: **Compensating illumination and graph-based worm cluster untangling.** (**a**) Input image. (**b**) Result of background illumination approximation using our convex-hull approach. (**c**) Result of compensating for variations in illumination by dividing the original image by the approximation of the image background illumination. (**d**) Result of automated intensity thresholding of c. (**e**) The skeleton overlaid on a worm cluster. Black skeleton segments form the edges of the graph, while the red branch points form nodes. (**f**) Dashed purple and yellow lines represent paths either too short, too long or of too costly shape to represent true worms, and are therefore eliminated. (**g**) The final ensemble of paths that minimize the ensemble cost by good cluster coverage, minimal worm overlap and low worm shape cost. (**h**) Worms reconstructed from each path of the ensemble. (**i**) Final segmentation result of full image.



Supplementary Fig. 14: **Preprocessing of cluster skeletons.** (**a**) Touching worms and their morphological skeletons in red. (**b**) Untangling without skeleton preprocessing results in worm outlines that do not follow the shapes of the worms properly as it centers the worms on the skeleton. (**c**) Highlighted in blue is the area of the binary object where the worms touch. The touching area consists of all pixels that are more than half a worm width away from the image background. (**d**) Branch points in the touching area are removed and all possible paths are created between the entering skeleton pieces, here represented as dotted lines, and the two reconnecting paths selected by the graph search are shown as green arrows. (**e**) Untangling result after skeleton preprocessing.

Supplementary Table 1: Assay design and error handling.

Design issue:	Proposed solution
Excessive amounts of bacteria or debris lead to poor foreground/background segmentation, yielding poor worm untangling performance.	Add a washing step to clean the samples before imaging.
Shadows from well walls can cause problems with foreground/background segmentation.	In cases where it is feasible, transfer worms to slides for analysis (this is a step that is sometimes necessary for particular staining protocols anyway, as in the case of the OilRedO assay in the paper).
Unusual phenotypes do not fit the worm model and are discarded as debris.	High-throughput experiments are often designed to consist of a fixed number of worms per well. It is therefore feasible to flag wells with few detected worms so they can be screened visually (or with an improved worm model), to detect unexpected phenotypes (or failures in worm detection due to large amounts of debris or other problems).
Dying worms rupture through the vulva or have blistered cuticles	During the training mode, we assume that the worms are not branched (a rupture often looks like a blob, or small branch), and train the model on the longest path through the worm. The same is true when we fit the model during untangling. The ruptured vulva will be seen as a variation in intensity at the edge of the outlined worm.
Live worms may move slightly in the time that elapses between bright field and fluorescence image capture. This may cause mis-alignment between worm segmentation (based on bright field images) and fluorescence, causing false shifts in expression patterns.	Motion can be reduced by the addition of a paralytic drug, in cases where the drug does not impact the phenotype of interest.

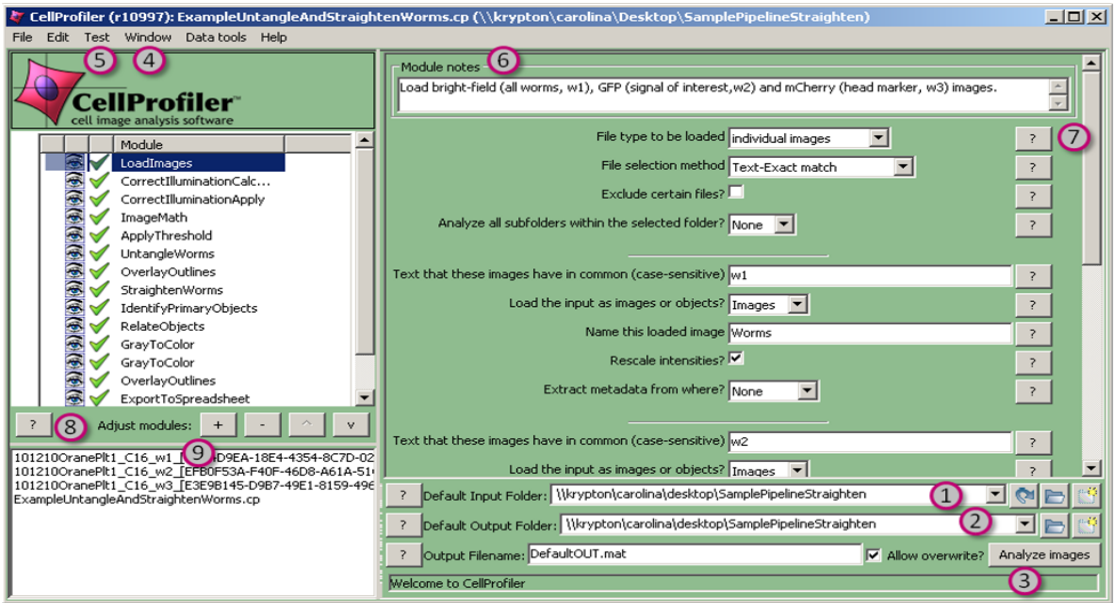
Supplementary Methods 1:

How to get started with the WormToolbox

The WormToolbox is a set of CellProfiler modules specific for worm analysis that are used in conjunction with standard modules of CellProfiler to analyze a set of images. A series of modules makes up a "pipeline". In order to get you started, we provide a few example pipelines that use modules from the WormToolbox. After trying the example pipelines as described below you can adjust the modules within them to fit your own image data and experimental goals.

Image analysis

1. Download and install CellProfiler 2.0 (version r11710 or later) from Supplementary Software or www.cellprofiler.org.
2. Go to Supplementary Software or http://www.cellprofiler.org/examples.shtml and download **one** of the example pipelines for worm analysis. The instructions below are general, but the illustration refers to the example called 'Straighten worms and extract intensity measurements using a low-resolution atlas'.
3. Start CellProfiler and set Default Input Folder (1) and Default Output Folder (2) to the folder where you have downloaded the sample pipeline (in the illustration below, '\\krypton\carolina\Desktop\SamplePipelineStraighten').
4. Doubleclick the pipeline (.cp) file in the file display window (9) to load the pipeline. The modules of the pipeline now appear in the pipeline window.
5. You can now click Analyze images (3) to run the full pipeline on the example images contained in the folder, and a result-window will pop open for each module of the pipeline. Two csv files with extracted measurements will also appear in your output folder.
6. To get a better feeling for what each step in the pipeline does, first click on 'Window' (4) and 'Close all open windows'. Then click on 'Test' (5) and 'Start test run'. This allows you to step through the different modules by clicking the 'Step' button that appears below the pipeline. You can also choose to show or not show the result of a given module by clicking the 'eye' next to the module. The 'Module notes' (6) briefly describes what each module does.
7. If you want to run the pipeline on a larger example data set, go to http://www.broadinstitute.org/bbbc/ to download more example worm images and place them in your local folder. The full data set will be analyzed if you exit the test mode and click 'Analyze images'.



8. To adapt a pipeline to your own images, you can change the 'LoadImages' module to load your own data, and adjust the modules' settings guided by the help that will appear when you click '?' next to the different parameter settings (7) or below the pipeline (8). Click the '+' to add new modules to your pipeline if needed. The worm-specific modules are listed under 'WormToolbox'.
9. Large-scale, high-throughput experiments will likely require the use of CellProfiler's batch processing tools, which use a computing cluster to speed analysis. A section of the manual describes how to accomplish this (http://cellprofiler.org/CPmanual/Help_Using%20CellProfiler_Other%20Features_Batch%20Processing.html). CellProfiler can also be run from the command prompt (without the user interface) if needed. For large datasets, we recommend using the Export To Database module in CellProfiler, which produces .csv files and export of extracted data to a database.
10. For further questions, please consult the CellProfiler manual and the user forum at http://cellprofiler.org/forum/.

Data analysis

Within CellProfiler: Once a pipeline has completed processing a set of images, some simple data analysis and visualization can be done within CellProfiler by adding modules in the "Data Tools" category to the pipeline (e.g., Display Scatterplot, Display Histogram, etc). You can also perform the same functions by using the Data Tools menu after a pipeline has completed running. One particularly useful module is Calculate Statistics, which can be added to a pipeline to calculate the Z' factor and other image assay quality statistics.

In a spreadsheet program (e. g. Excel): Measurements extracted by CellProfiler using example pipelines are saved in spreadsheet format by the Export to Spreadsheet module. The data may be explored using

any software for handling of spreadsheets. Such software can be useful to rank-order samples based on the measurements of the phenotype of interest.

Using CellProfiler Analyst's visualization/analysis/exploration tools: This companion software for CellProfiler is also freely available at www.cellprofiler.org and can be used for the visualization and exploration of high-throughput data. One advantage compared to spreadsheet programs is that visualizations in CellProfiler Analyst are interactive and linked to the image data to allow visual confirmation of phenotypes. It also enables the visualization of quality-control results from a screen, and normalization of results to correct systematic errors (REF: http://JBX.sagepub.com/content/early/2011/09/27/1087057111420292.abstract).

Using CellProfiler Analyst's machine learning tools: The Classifier tool in CellProfiler Analyst can be used to train a classifier to recognize a phenotype of interest by dragging and dropping example worms. See CellProfiler Analyst's manual for a complete description of its usage.

Using database query tools: If exported to a database using the Export To Database module in CellProfiler, data can also be explored and analyzed with database query tools.

Supplementary Methods 2:

Details of worm untangling by model guided graph search.

Graph traversal

The graph traversal algorithm selects only the paths that have an acceptable cost and length from among all possible paths. The number of possible paths increases exponentially as segments and branch points are added which means that the running time of the algorithm also increases exponentially in the worst case. There are graph traversal algorithms that find minimum cost paths through a graph with a polynomial run-time and memory usage[i] but these assume that the problem can be formulated as a graph with positive, incremental costs along each edge of the graph. In contrast, our method requires the evaluation of a cost function applied to the whole path. The cost function cannot be reduced to the sum of costs along the path and the cost of a path with an additional edge may be less than the cost of a path without that edge. We therefore use graph pruning techniques and features of the Python language to limit the run time and memory consumption of the algorithm. We also limit analysis to tractable graphs by reporting an error for graphs of large complexity.

Cost function

The cost function evaluates a path through the graph. The path derived from the graph is composed of segments and branch points linking the segments. The cost function algorithm creates a list of points that connect the pixels in the segment skeletons. It concatenates these lists, joining one segment's entry point into the branch point with the following segment's exit point. The cost function calculates the length of the path as the Euclidean distance between each point and its successor; if the path length is shorter than the acceptable minimum or longer than the acceptable maximum, the path is discarded.

The cost function then selects equidistant control points along the path using cubic spline interpolation. Each of the control points other than the first and last is at the vertex of an angle formed by the lines from its predecessor and successor. These angles and the path's length form a feature vector. This feature vector is reduced to a single cost by taking the dot product of the feature vector and cross-correlation matrix derived from the training data.

The graph traversal algorithm considers the cost of ensembles of paths. Two paths may share a segment, for instance, when worms overlap. The algorithm assigns a penalty to an ensemble proportional to the length of these overlapping segments.

Algorithm steps

Conceptually, the algorithm is composed of three steps: enumeration of paths, calculation of costs of individual paths, and calculation of costs of ensembles of paths. The first step of the algorithm generates a large number of possible paths, almost all of which are discarded by the cost function. Our implementation encapsulates path enumeration in a Python generator[ii] in order to save memory; the generator returns a path data which is immediately consumed and in most cases destroyed by the cost evaluation step, resulting in lower memory consumption and more efficient cache utilization.

Path enumeration

A worm cluster graph consists of skeleton segments which form the graph's edges and branch points which form the graph's nodes. The path enumeration algorithm generates all paths whose lengths are between the minimum and maximum acceptable length, given this graph. The following are the initialization steps for the recursive algorithm:

- The algorithm calculates the length of each segment.
- For each segment whose length is between the minimum and the maximum acceptable length, generate a path containing that single segment.
- Create a partial path, for each segment and branch point incident on that segment and use that partial path as the starting point for the recursive part of the algorithm.

The algorithm then recursively evaluates the list as follows:

- For each segment, if the segment is incident on the last branch point in the path and the segment is not yet in the path, add it to the path.
- If the length of the path is greater than the maximum acceptable length, terminate the recursion along this path since all subsequent paths will be longer than the maximum.
- If the length of the path is greater than the minimum acceptable length, generate that path.
- Find the branch area at the opposite end of the added segment. Terminate if the branch area is already part of the path.
- Add the branch area to the path and call the algorithm recursively.

The worm untangling code computes the cost function for each generated path. It discards a path if the cost function exceeds the maximum acceptable cost.

Ensemble cost calculation

The last step in worm untangling is the selection of an ensemble of paths from among all acceptable single paths. Some paths may be slight variations on others, sharing most of the same segments, others may share short segments where worms cross or overlap and others may be wholly disjoint. There are three parts to the cost of a particular ensemble: the sum of costs of the individual paths in the ensemble, a penalty cost that is proportional to the length of all segments that are shared by paths in the ensemble and a penalty cost that is proportional to the length of all segments that do not appear in any path in the ensemble. The goal of the ensemble cost calculation algorithm is to find the ensemble with the lowest cost.

Given **P** paths indexed from 1 to **P**, the algorithm begins by creating a **P**x**P** Boolean matrix where each row of the matrix represents an ensemble being considered and each column represents a path. The value in a particular cell in the matrix is true if the path represented by the column is in the ensemble represented by the row. The algorithm initializes the diagonal of the matrix to True, representing all ensembles with a single path. It then proceeds to create new ensembles from the old by adding paths where the path index is greater than any in the parent ensemble. The algorithm is performed **P** times at most, corresponding to all **P** paths being added to the graph.

The steps of the algorithm initialization are:

- The current best ensemble is set to the empty set.
- The current best cost is set to the sum of segment lengths times the penalty per unit length for an unused segment.
- The ensemble matrix is initialized to the diagonal so that single path ensembles will be considered first.

The following steps are then run at most **P** times:

- The cost (*partial* cost) of each row, not counting the cost assigned to missing segments is computed for each ensemble.
- The cost of the missing segments for each ensemble is added to the partial cost to get the *total cost*.
- The best total cost is found. If this cost is less than the current best cost, the ensemble with that cost becomes the current best ensemble and its cost becomes the current best cost.
- If the partial cost for a row is greater than the best total cost, then that row cannot have a better cost than the current best total cost and the row is excluded from consideration. If all rows have higher partial costs, the algorithm terminates, choosing the current best ensemble. The most costly ensembles are weeded out at this stage if the number of ensembles to be computed at the next stage would be too computationally expensive.
- A new set of ensembles is generated with each ensemble containing one more path than the parent ensemble. The added path must have a higher index than any in the parent ensemble; this guarantees that the generated ensembles will be unique.

---

[i] Dijkstra, E. W. (1959) "A note on two problems in connexion with graphs", Numerische Mathematik Vol 1 p 269-271

[ii] PEP 255: Simple Generators www.**python**.org/dev/**pep**s/**pep**-0**255**/

Supplementary Software

Contents:

1. Compiled version of CellProfiler 2.0 revision 11710 (including WormToolbox) for

- Mac (open the DMG and copy CellProfiler to the Applications folder): **CellProfiler_2.0_11710.dmg**
- Win32 (run the EXE and follow the prompts): **CellProfiler_2.0_win32_11710.exe**
- Win64 (run the EXE and follow the prompts): **CellProfiler_2.0_win64_11710.exe**

2. Source code (in Python) for CellProfiler revision 11710: **cellprofiler-11710.zip** Please refer to the developers pages at www.cellprofiler.org for more information. Visit the CellProfiler forum (www.cellprofiler.org/forum) for advice on running CellProfiler on Linux or on a cluster, or consult the developers' wiki (www.cellprofiler.org/wiki).

3. Pipelines using the WormToolbox

These pipelines have been developed for high-throughput screens on C. elegans and extract measurements on a per-worm basis. They may however be useful also for smaller experiments.

- Untangle worms: In this pipeline we identify individual worms and extract shape and intensity measurements. Each step of the pipeline is annotated to describe function and settings. Worm untangling requires a worm model, and such a model is provided together with the pipeline. If adjusting the pipeline to fit your own data, worm detection will likely improve by creating a new worm model based on your own image data and following the steps of the sample pipeline below. More sample images and information on the image data can be found on http://www.broadinstitute.org/bbbc/. **ExamplePipelineUntangle.zip** (1.3 MB)

- Straighten worms and extract intensity measurements using a low-resolution atlas: Once worms are untangled, they can be straightened and aligned with a low-resolution worm atlas to extract localized intensity measurements and compare patterns of reporter signals. This pipeline describes the different steps and provides a sample image. It also includes steps for identifying secondary objects (fluorescent marker signals) and relating these objects to individual worms, enabling count of signals on a per worm basis. More data can be found on http://www.broadinstitute.org/bbbc/. **ExamplePipelineStraighten.zip** (968 KB)

- Create your own worm model: The UntangleWorms module has an "Untangle" mode and a "Train" mode. This pipeline describes how the "Train" mode is used to create worm model. Training consists of providing a large number of images of worms that are representative of the worm variation within the population, and that do not touch or overlap. The selection of worms can either be semi-automatic (visually verifying each worm added to the training set) or fully automated (selecting single worms based on size). This example pipeline includes only two images and will not result in a good model, as it will not be representative of all possible variations of the worm shapes. We recommend using at least 60 worms to create a model. More

data, for creating a better model, can be found on http://www.broadinstitute.org/bbbc/. **ExamplePipelineCreateModel.zip** (1.6 MB)

- Untangle worms and make measurements bright-field staining pattern phenotype: This pipeline detects individual worms by worm untangling and finds sub-objects (fatty regions stained with oil red O) within the worms. Using bright-field data only, it detects fatty regions by intensity thresholding in a single image channel and relates the fatty regions to individual worms. This enables detection of rare phenotypes in heterogeneous populations, phenotypes that would be missed if population averages were observed. More data can be found on http://www.broadinstitute.org/bbbc/. **ExamplePipelineOilRedO.zip** (852 KB)

Supplementary Note 1: Imaging equipment and settings.

Figures 1a and f:

- Equipment: Molecular Devices Discovery-1 microscope with a 2 low magnification objective running MetaExpress software
- image size:696x520 pixels
- pixel size: approx. 8.6μm
- bit depth: 12 bit gray scale
- temperature: room temperature
- medium: worms in 384-well plate, liquid media (DMSO)
- filters: none
- imaging modality: bright field transmitted light
- processing: global adjustment of contrast and brightness to facilitate visualization using CellProfiler
- further manipulation: none

Figures 2a and b:

- Equipment: Molecular Devices Discovery-1 microscope with a 2 low magnification objective running MetaExpress software
- image size:696x520 pixels
- pixel size: approx. 8.6μm
- bit depth: 12 bit gray scale (for each channel)
- temperature: room temperature
- medium: worms in 384-well plate, liquid media (DMSO)
- filters: filter cube optimized for GFP imaging and mCherry imaging.
- imaging modality: bright field transmitted light and epifluorescence. Exposure times were set for the most highly expressed condition and kept constant throughout each experiment.
- processing: global adjustment of contrast and brightness to facilitate visualization using CellProfiler: the same adjustments were made to all images displayed.
- further manipulation: none