Chapter 20

Identifying Fluorescently Labeled Single Molecules in Image Stacks Using Machine Learning

Scott A. Rifkin

Abstract

In the past several years, a host of new technologies have made it possible to visualize single molecules within cells and organisms (Raj et al., Nat Methods 5:877–879, 2008; Paré et al., Curr Biol 19:2037–2042, 2009; Lu and Tsourkas, Nucleic Acids Res 37:e100, 2009; Femino et al., Science 280:585–590, 1998; Rodriguez et al., Semin Cell Dev Biol 18:202–208, 2007; Betzig et al., Science 313:1642–1645, 2006; Rust et al., Nat Methods 3:793–796, 2006; Fusco et al., Curr Biol 13:161–167, 2003). Many of these are based on fluorescence, either fluorescent proteins or fluorescent dyes coupled to a molecule of interest. In many applications, the fluorescent signal is limited to a few pixels, which poses a classic signal processing problem: how can actual signal be distinguished from background noise?

In this chapter, I present a MATLAB (MathWorks (2010) MATLAB. Retrieved from http://www.mathworks.com) software suite designed to work with these single-molecule visualization technologies (Rifkin (2010) *spotFinding Suite*. http://www.biology.ucsd.edu/labs/rifkin/software.html). It takes images or image stacks from a fluorescence microscope as input and outputs locations of the molecules. Although the software was developed for the specific application of identifying single mRNA transcripts in fixed specimens, it is more general than this and can be used and/or customized for other applications that produce localized signals embedded in a potentially noisy background. The analysis pipeline consists of the following steps: (a) create a gold-standard dataset, (b) train a machine-learning algorithm to classify image features as signal or noise depending upon user defined statistics, (c) run the machine-learning algorithm on a new dataset to identify mRNA locations, and (d) visually inspect and correct the results.

Key words: Single molecule, FISH, Machine learning, MATLAB, mRNA, Microscopy, Biological image informatics

1. Introduction

*S*ingle *m*olecule RNA *f*luorescence *i*n *s*itu *h*ybridization (smFISH) is a relatively new technique that enables visualization of transcripts in fixed cells or organisms. Several varieties of the technique have been demonstrated in the literature, which all yield the same general kind of data: groups of pixels (*spots*) of relatively high intensity against a background field of lower intensity (1–5).

Measurements of variation in gene expression have become quite common in the literature of comparative biology. In evolutionary-developmental biology, they often take the form of traditional in situ assays for a set of homologous genes in different species, which give qualitative information about spatial expression patterns but are not quantitative. Evolutionary genetic techniques include, for example, microarrays, RNA-Seq, and pyrosequencing, which are usually relative measures of expression and have no spatial information because they destroy the sample. smFISH yields absolute counts of mRNA abundances with high spatial resolution. The trade-offs are that it is relatively low throughput compared to genomic techniques, and the signals are weaker than those in traditional in situ assays. smFISH is appropriate for questions that require highly resolved measures of gene expression and/or an explicit spatial context. Furthermore, because it only requires sequence information, it can be used on nonmodel organisms as long as sequences are available.

The software (10) discussed in this chapter was developed using data from the technique of Raj et al. 2008 (1, 11), and so, I briefly describe the experimental technique below. A comprehensive description of the experimental protocol can be found in ref. 11 or online at http://www.singlemoleculefish.com. A set of thirty to fifty 20-mer oligo probes is designed to be complementary to a target transcript. These oligos are labeled at the 3′ end with a fluorophore. Probes are incubated with fixed samples (e.g., cells, tissue sections, organisms) in a formamide-based hybridization buffer, and then unbound probe is removed in a series of washes. Labeled samples can then be imaged under a fluorescence microscope. When bound to an mRNA in a fixed specimen, this set of oligos appears under the microscope as a diffraction-limited fluorescent spot (Fig. 1).

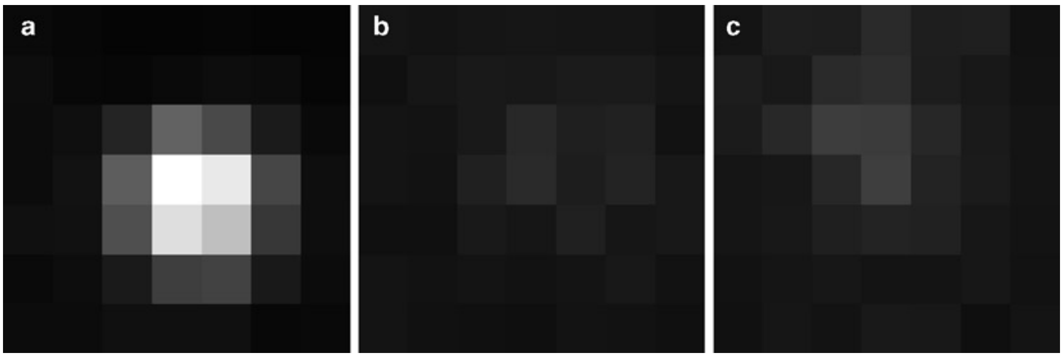


Fig. 1. Example of smFISH spots. (**a**) A 7 × 7 pixel box centered around a clear mRNA spot. (**b**) A local intensity maximum that is not an mRNA spot. (**c**) A marginal mRNA spot.

In my experience, using a wide-field epifluorescence microscope, a probe set of at least 21 oligos is necessary to be able to identify a localized signal above background. As a consequence of this requirement for multiple oligos to bind and the relative uniqueness of 20-mers in a genome, the technique is highly specific, although like many sequence-based techniques, extremely similar paralogs cannot be distinguished. Background fluorescence can come from a few oligos binding to other molecules, unbound fluorophores not being completely washed away, autofluorescence of the sample, or out-of-focus light. These factors can be controlled by checks during probe design, adjustments to the stringency of the washes, judicious use of fluorophores, or modifications to the microscopy setup. However, these modifications may involve tradeoffs that weaken the signal, and so, in many cases an appreciable degree of background noise will have to be tolerated.

In the following discussion, I assume that the microscopist has taken z-stacks through a sample, producing a series of two-dimensional slices that can be concatenated into a three-dimensional image. Furthermore, I leave it to the reader to segment the image stacks. The details of how to do this will depend upon the particular type of specimen – in particular on the contrast between the object boundary and everything outside, which may include other confluent objects. There is not a single solution that will work for everything, and reviewing various approaches is beyond the scope of this chapter. The starting data for the single molecule identification is an image stack and a segmentation mask that distinguishes pixels that belong to an object from those that do not (Fig. 2). For convenience in the discussion below, I refer to these objects as

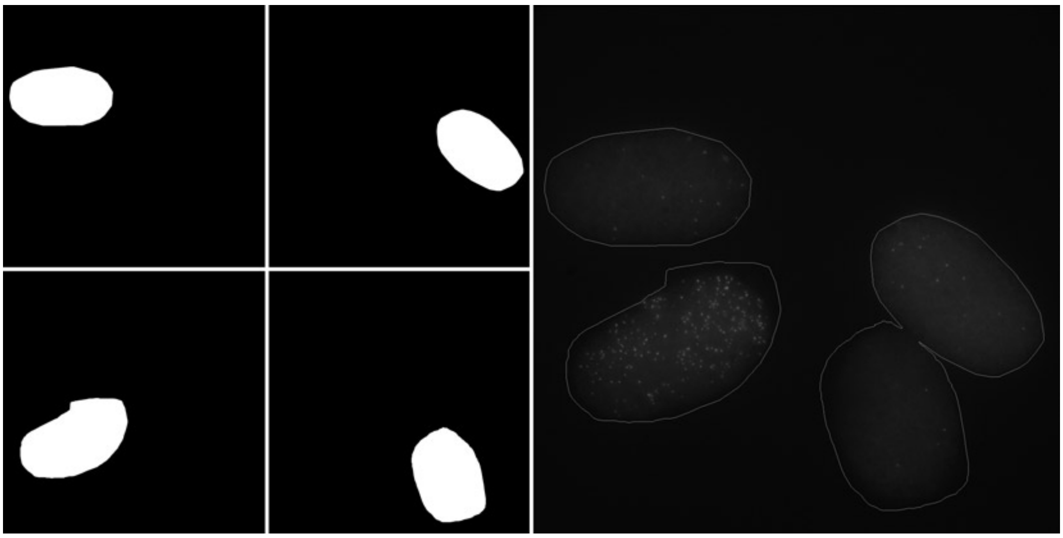


Fig. 2. An example of a series of segmentation masks for the image on the *far right*.

"*specimens*". They could be cells, small animals, anything. *Cell* is unfortunately a technical term in MATLAB and *objects* can be ambiguous, so I use *specimen* simply as a convenient specific identifier for these segmented objects. The intensity pattern from a single fluorescent molecule is called a *spot*.

The spot finding software is built around a machine learning core and consists of four parts which are covered in detail below: Subheading 3.2 – Manually annotate an image to create a gold-standard dataset; Subheading 3.3 – Use the gold-standard dataset to train a classifier to distinguish between spots representing mRNA signals and noise; Subheading 3.4 – Apply the classifier to a new image; Subheading 3.5 – Manually review and curate the results which may include re-rerunning the training and subsequent classification. This software is modular and flexible. The annotation component (Subheading 3.2) could be used on its own as a way to manually identify spots (e.g., if only a few images need to be processed). The machine learning component (Subheadings 3.3 and 3.4) uses the random forest algorithm (12–14), but this could straightforwardly be modified for the user's preferred classifier. The reviewing software (Subheading 3.5) relies only upon a particular data and file structure. As long as the output from a classification algorithm can be translated into this data format, the reviewing software can be used to manually review and correct the results from any spot-classification program.

2. Materials

1. Computer with sufficient RAM to process the image files.
2. MATLAB (http://www.mathworks.com).
3. spotFinding Suite (http://www.biology.ucsd.edu/labs/rifkin/software.html).
4. R (http://r-project.org).
5. randomForest R package (http://cran.r-project.org/web/packages/randomForest/).

   The following MATLAB packages can be downloaded individually but are also included with spotFinding Suite under the BSD license.

6. SC(http://www.mathworks.com/MATLABcentral/fileexchange/16233).
7. gfit2(http://www.mathworks.co.uk/MATLABcentral/fileexchange/22020).

The following MATLAB package can be downloaded individually but is also included with spotFinding Suite under the GNU GPL3 license.

8. tiffread (http://www.embl.de/~nedelec/misc/).

3. Methods

The software is designed to work in a single directory. All data and segmentation files, described in Subheading 3.1 below, should in the same directory, and all files that the software generates will be saved in this directory. The default file name for an image stack is (dye) (UniqueStackidentifier).tiff. For example, the third image stack labeled by the fluorescent dye TMR would be named tmr003.tiff where 003 is the unique stack identifier. The default for the segmentation file is segmenttrans(UniqueStackIdentifier).mat, e.g., segmenttrans003.mat. The software allows a user to enter image file names, segmentation file names, fluorophores, and unique stack identifiers at the command line, but it is much quicker and simpler (especially when processing many image stack files) to ensure that files are in the default format before starting.

On Unix-like systems (e.g., Linux, Mac OS X), no modifications are needed to call R. On Windows, the software by default assumes that the R executable is located at:

C:\\"Program Files"\\R\\R-2.9.0\\bin\\Rterm.exe.

If this is not correct, the user will need to locate this line in trainFISHClassifier.m and classifyFISHSpots.m and replace it with the correct location.

Matlab specific terms used in this chapter are defined in Table 1. More detailed information can be found in the Matlab help documents.

3.1. Correctly Format the Image and Segmentation Files

The software assumes that the segmentation and image files are in a specific format. Assume that the images are z-stacks of two-dimensional slices of size A × B pixels. The segmentation file is a MAT file consisting of a cell array that the program assumes is called currpolys. An entry in the cell array is an A × B pixel binary image with 1 s denoting a specimen of interest and 0 s denoting area outside the specimen. Often, a single image contains several specimens to segment. It is convenient to store the segmentation masks for each specimen together in a single file, each one being stored in a separate cell in the cell array.

The software comes configured to read multi-image TIFF files using the MATLAB function tiffread. An image stack consisting of MATLAB doubles is created with the following commands:

Table 1
Matlab terms used in this chapter

MAT file	A binary, Matlab-specific data file format with the extension.mat
Cell array	A data structure that can hold an assorted collection of objects. Think of it as an integer-indexed set of bins (called cells) where any Matlab data structure can be placed in a cell
Double	A double precision floating point number
Struct	Stuctured array. A data structure where each entry (value) is indexed by text (field) and can be a cell array or number. These are especially useful for collecting assorted information about a particular object (see Tables 3 and 4)
Function calls	Matlab functions are called using the following format: outputArguments = functionName(inputArgument1, inputArgument2, …). A semicolon at the end of a statement blocks output from being displayed on the screen

```
stack = tiffread(stackFileName);
stack = double(stack.data);
```

If images cannot be saved in a tiffread compatible format, the user will need to find these lines in the code and modify them to read his or her particular image format. The subsequent program represents the data stack as a three-dimensional array of doubles.

During the process of finding spots, several files are created. The names of the training set files are based on the dye and the transcript. For example, if TMR is used to label the *C. elegans elt-2* transcript, the user might designate the dye as “tmr” and the gene as “C_el_elt2”. The training set would then be stored in trainingSet_tmr_C_el_elt2.mat. Other files are associated with particular stacks via the unique stack identifier. For example, tmr003_wormGaussianFit.mat contains the results of the spot finding program run on the multi-image tiff file tmr003.tiff.

3.2. Creating a Training Set

Command:

```
trainingSet = createFISHTrainingSet(stackName, probeName);
```

Once the appropriate files are in place and properly formatted, the first step in finding spots is to annotate an smFISH image by identifying examples of true spots and examples of sets of pixels that are not spots. After the user identifies particular and segmentation files to use, a GUI window pops up, which allows the user to do this manual classification. However, some preprocessing must be done on the image to make this a manageable task. The size of spot will depend upon the microscope magnification and resolution; by default, the software assumes a spot in two-dimensions fits well within a 7×7 pixel square. A specimen of approximately 400×250 pixels will contain close to 100,000 of these 7×7 squares in each slice. It would be impossible to go through all of these manually, and so, preliminary ranking of the pixels is essential.

The software first identifies the local 3D intensity maxima in the image stack. The idea is that a single fluorescent molecule will appear as a diffraction-limited spot covering several pixels and that the intensities of these pixels will decay at a roughly Gaussian rate from the center. As a result, true spots will contain a central local maximum, and so, only pixels that are local maxima need to be considered. However, thousands of pixels will be local maxima. True spots should be more fluorescent than nonspots, and so, ranking by intensity is a natural way to order these maxima. However, some samples may have nonuniform background across the sample and may have more out-of-focus light in some slices. Instead of a straight ranking by intensity, the software corrects for local background by performing a morphological opening (see Note 1) and subtracting this opened image from the raw image. The size of the structuring element used for the opening is based upon the size of a spot in the user's microscopy setup; by default it is a disk of radius 7. Other local background correction procedures could be substituted if desired. The local maxima are then ranked by their background-subtracted intensities and presented to the user for evaluation. Although the pixel intensities in the image are manipulated in this step, all manipulations, whether local background subtraction or intensity scaling, are for the user interface only. All computational evaluations of the spots by the machine learning algorithm are performed on the unadulterated raw image. If there were a reason to adjust the images – for example to remove a systematic increase in average intensity with z-position of the slice – this could certainly be implemented (see Note 2).

The annotation GUI window called identifySpots3 (Fig. 3) has several components. On the left is a 16 × 16 pixel image with blue pixels (marked with "B" in the figure) against a background of gray pixels. For display, each slice of the stack is scaled such that the minimum pixel intensity is 0 and the maximum is 1, and the pixel intensities on the display image are the scaled intensities. As mentioned above, this scaling is for display only. The blue pixel in the ninth row, ninth column from the top is the local maximum at the rank indicated by the "Spot Rank" slider. Other blue pixels in the 16 × 16 field indicate other local maxima in the vicinity.

On the right-hand side of the window is a zoomed portion of the entire slice centered on the pixel of interest. By default, this is set to 512 × 512 pixels and is normalized for viewing such that the minimal pixel value of the slice is 0 and the maximal value is 1. It is sometimes useful to be able to see the pixel highlighted in blue on the left against a background of more of the specimen. The segmentation outline of the specimen is highlighted in yellow, and the 16 × 16 square on the left is outlined green.

Below this right-hand side image is a smaller image displaying the entire slice with segmented specimens highlighted in red and outlines of the areas depicted in the left and right larger images.

In the middle of the window is a stack of five 16 × 16 pixel surface plots. The height and color of a pixel reflects its intensity.

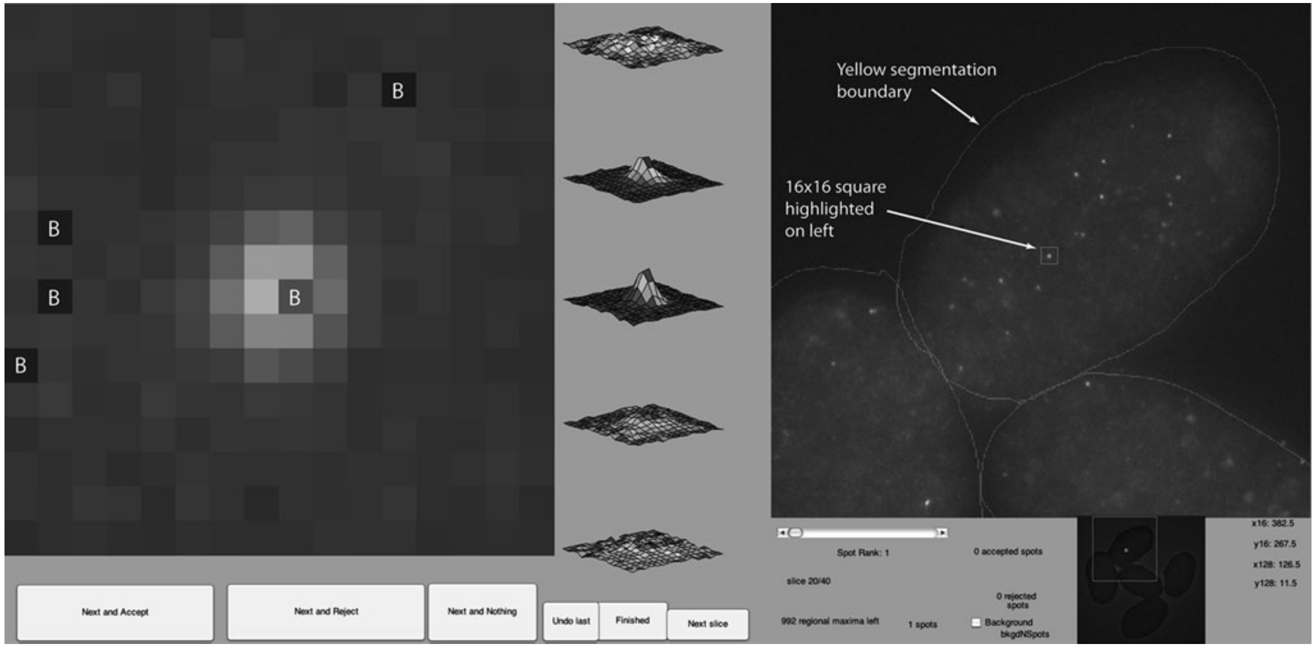


Fig. 3. The annotation GUI. For the purposes of this chapter, *blue* pixels in the *left panel* are marked with B. Other colors are as noted.

If the current slice is *n*, the surface plots are the 16 × 16 pixel area of the image on the left in (from bottom to top) slices $n-2$, $n-1$, *n*, $n+1$, $n+2$. It is good practice to take z-slices close enough together such that a spot will appear in at least two. This representation of five consecutive slices around a local maximum gives information about the three-dimensional context of the putative spot.

Along the bottom of the window are a series of push buttons that tell the software what should be done with the local maxima highlighted in the left image. The software will process local maxima in order of rank. The slider below the right-hand side image allows the user to jump to a particular rank. Finally, a series of text fields below the images give information on the current slice, the number of regional maxima (up to 1,000) that are left to evaluate, the number of potential spots (local maxima) in the current 16 × 16 field of view, and the number of local maxima that have been rejected or accepted so far. On the far right, the current coordinates for the upper left corners of the left and right images are listed for reference.

Collecting a training set entails assessing some number of these local maxima and either accepting them as spots or rejecting them. This consists of the following steps:

1. Evaluate the slice. Look at the right, large image. Are spots visible or does the entire image look fairly uniform? Slices at the top or bottom of the stack may be outside the boundary of the specimen and so may look rather uniform without clear spots or definition (see Note 3). In this case, it is worthwhile to press the *Next Slice* button. This will simply advance to the next slice up and will neither accept nor reject any of the blue local maxima in the current 16 × 16 field on the left. If spots are visible, implying that the slice is within the specimen, it is worthwhile to evaluate the local maxima.
2. Decide on an appropriate local maximum rank to look at. By default, the software starts with the top-ranked local maximum in the initial slice. If there are spots in a slice, this should be an unambiguous example of a true spot. After the user has accepted a good number of solid spots and rejected many non-spots, it might be worthwhile to include some more borderline ones. Adjusting the *Spot Rank* slider is a quick way to jump around the rankings to find intermediate cases. Jumping to a new rank is done by moving the slider and then pressing one of the *Next* buttons. This ranking is carried over between slices: if *Next Slice* is pressed with *Spot Rank* slider on 75, the focal local maximum in the next slice will be the one with rank 75.
3. Evaluate the local maxima in the 16 × 16 image. While the focal local maximum will be in the center, there may be other local maxima in the 16 × 16 image. True spots could be close together, in which case the 16 × 16 image would contain more than one true spot. Usually, however, the nonfocal blue pixels

are just noise – random local maxima in the background. Good spots generally show up in more than one slice (as long as the slices are close enough together, ~0.4 μm), are roughly Gaussian and regular in shape, and have a higher intensity than the local background. A candidate spot can be rejected by clicking on it. This turns it from blue to gray and also increases the rejected spot count by 1. If the user makes a mistake, the last action can be undone with the button *Undo last*. Clicking on a gray pixel will change it to blue, thereby designating it as a potential spot (see Note 4). When the user is satisfied that the blue pixels that remain mark true spots, clicking *Next and Accept* will store them, change their color to red, and increase the "accepted spot" count accordingly. In some cases, none of the blue pixels will mark real spots. Instead of clicking on each one, the whole lot can be rejected by clicking *Next and Reject*. Rejected pixels are changed to gray. If the remaining blue pixels are ambiguous, the user can remain agnostic so as not to contaminate the training set by clicking *Next and Nothing*. When a *Next* button is pressed, the software moves on to either the next ranked local maximum or the local maximum with rank indicated by the *Spot Rank* slider. Alternatively, the user can move the focus to a particular location within the specimen by clicking on it in the right-hand side image. The upper left corner of the 16 × 16 pixel square is moved to the clicked location.

4. Decide whether to move to a new slice. If there are just a few true spots in a slice, the user will quickly move through them and start rejecting local maxima that are not spots. While it is important to include local maxima that are not spots in the training set as negative examples, this is rather easy, and it does not make sense to process all 1,000 local maxima in a slice before moving on to a new slice. In addition, it is not necessary to include all the spots in a sample in the training set, and it makes sense to include spots from multiple slices of the sample, although whether this is strictly necessary is unclear. In practice, around 100 accepted and 100 rejected spots works well, especially because the training set can be augmented later on by corrections to a classification (Subheading 3.5). The user can switch to a new slice by clicking *Next Slice* and, if finished collecting the training set, can press *Finished* to move on to the next step.

3.3. Train the Classifier

Command:

```
trainingSet=trainFISHClassifier(trainingSet,0);
```

The output from the annotation is stored in two files. goldSpots_(dye)_(gene).mat contains an X-by-3 array of doubles where each row has the coordinates of one of the X accepted local maxima (row, column, slice) in the matrix representation of the image stack. rejectedSpots_(dye)_(gene).mat contains the rejected maxima coordinates. These coordinates, along with the raw image file are piped to a function that will calculate various statistics on the maxima and train a classifier based on these statistics.

Statistics, or observations, of the data are a crucial part of any classification scheme. Good statistics capture aspects of true spots that distinguish them from noise. Designing a statistic is something of an art, but the usefulness of a statistic can be evaluated post hoc based on how much importance the classifier assigns it. In this software, most statistics are calculated on a square of pixels centered around a local maximum. By default this is a 7×7 pixel square but the size can be adjusted depending upon the usual size of a spot in the user's particular microscopy/camera setup. A few of the default statistics use the $7 \times 7 \times 3$ pixel box around the maximum. Most of the default statistics are based on measuring how the pixel intensity drops off around the maximum under the prediction and observation that good spots have a Gaussian profile. Users can add their own statistical functions if desired. This requires writing a MATLAB function that takes a two- or three-dimensional array of doubles as input and outputs one or more statistics and corresponding names. Users can use default statistics in the folder spotFindingStatistics as templates and add the appropriate lines to the function calculateFISHStatistics.m.

The software calculates statistics on the maxima in the training set and outputs the results to a large matrix with one row for each maximum, one column for each statistic, and a final column with the classification index. This is then sent to the machine learning classifier. The software currently uses the R implementation of the random forest classifier called randomForest (12). In principle, the user could substitute any classifier with fairly minimal code modification. Random forests perform comparably to other machine learning classifiers or better, are fast, and are amenable to parallelization, which could be useful for processing large datasets. For complete details about the theory and implementation of random forests and in particular for information about how to set and interpret the parameters it takes, readers are encouraged to consult refs. 11–13. In brief, random forests are sets of decision trees that are built using subsets of the training data – about two third of the data end up being used for each tree. A tree consists of successive branch points at which the dataset is recursively partitioned based on a subset of the statistics with the goal of generating homogenous partitions. At each recursion, the random forest algorithm chooses a random subset of the statistics and finds a best partition of the remaining set of data. The algorithm stores exactly how the partition was made at each recursion in each tree. Classifying a new local maximum consists of running its statistics down each tree in the forest. Each branching point of each tree corresponds to a particular partition based on a subset of the statistics, and eventually the local maximum will reach a terminal leaf where it is labeled as either a spot or not. Each tree in the forest performs this classification with different combinations of subsets of the statistics and votes as to whether the local maximum is a spot or not. The majority decision is the final classification of the forest. Because only two third of the training data is used to generate any given tree, the remaining one third of the training data can be run down the tree to generate an independent estimate the error rate of the classification algorithm (12–14). The extent to which this error rate can be extrapolated to novel data depends upon how representative the training set is of the novel data. The initial training set is likely to be overrepresented for maxima where the classification is clear, and so, this initial error rate will underestimate the actual error rate on novel data. Subheading 3.5 describes two ways to estimate a more accurate error rate for the final dataset.

randomForest outputs several files (all prefixed by trainingSet_(dye)_(gene)) which can be used to assess the statistics and the likely classification error. Of these, the most useful are listed in Table 2.

Readers should consult refs. 12, 13 for other further randomForest options. These can be passed to the randomForest function via the tuneRF function call that can be found in trainFISHClassifier.m.

After the classifier is trained, the software stores all of the classification data including the training set and input and output to and from the random forest alogorithm in a structure called trainingSet stored in the file trainingSet_(dye)_(gene).mat.

Table 2
Output files from randomForest

.randomForest	contains the trained classifier to be applied to novel data
_votes.txt _margin.pdf	List and plot of the results of the voting across the forest for each maximum
_confusion.txt	Estimate of the error rate. The first row consists of maxima annotated as nonspots; the second row is for true spot annotations. The first column has numbers of maxima classified as nonspots; the second column numbers represent maxima classified as spots. The third column has rates of misclassification
_varImp.pdf	Plot of statistic importance. This plot portrays two measures of the importance of individual statistics to correct classification. Of the default statistics, intensity and goodness-of-fit to a Gaussian often dominate the classification
_MDS.pdf	Multidimensional scaling plot of the data classification. By default, this is a two-dimensional representation

3.4. Apply the Classifier to a New Image

Commands:

worms = evaluateFISHImageStack(stackName,1);

worms = classifyFISHSpots(dye, uniqueStackIdentifier, probeName, optional(worms));

After the classifier is trained with the initial training data, it can be applied to novel data. As for the training data, this requires segmentation and image files in the proper format (Subheading 3.1). As before, local maxima are identified and ranked in the three-dimensional image according to their local background subtracted intensities. The number of local maxima depends upon many factors including the size of the image stack, the concentration of mRNA, and the microscopy setup, but it can run upward of 1% of the pixels. This could be tens of thousands of local maxima. Computing and evaluating statistics for each of these would be computationally intensive and completely unnecessary, since the vast majority is just spurious noise. The challenge lies in determining when to stop without imposing user-dependent thresholds or cutoffs (cf. (1)). The software takes the following approach:

1. Start with the highest ranked local maximum and proceed in rank order.
2. Calculate the statistics for the local maximum. These will be the same set of statistics used for the training set.
3. Extract the statistic that measures goodness-of-fit to a Gaussian (see Note 5) and store it in an array. Along with intensity, this is usually one of the most relevant statistics for classification.
4. Go to step 1 until the first 30 local maxima have been evaluated meaning that the goodness-of-fit array is of length 30.
5. Find the 60th percentile of this array.
6. If the goodness-of-fit statistic for the 60th percentile is below 0.9, stop. If not, then continue down the ranked list as in step 1, adding the goodness-of-fit statistic to the array. Maintain an array of length 30, i.e., if the software has just processed the local maximum with rank 45, then the goodness-of-fit array would contain statistics from ranks 16 to 45. If at any point the 60th percentile drops below 0.9, then stop (see Note 6).

Once the list of candidate local maxima has been identified, the software passes the randomForest R function a matrix comprised of their statistics along with the random forest file from the training set. The machine learning program runs the data from each maximum down the decision trees in the forest. Each tree classifies the local maximum as a spot or not, and the final classification is decided by a majority vote. The classification for all of the candidate maxima is passed back to MATLAB.

MATLAB generates several files in the course of evaluating these local maxima. They are uniquely labeled by the image stack

Table 3
The fields in the struct worms{i}

mask	A binary image that is the segmentation mask for the specimen
boundingBox	A stuct containing the BoundingBox measurement from MATLAB's regionprops function
regMaxSpots	An Nx5 double array. Each row corresponds to a local maximum with the information: [row, column, slice, raw intensity, filtered intensity]
spotInfo	A cell array containing information about the local maxima evaluated
goodWorm	Indicates whether something is wrong with the specimen (0) or not (1)
spotsFixed	Indicates whether the spot classifications have been reviewed (see Subheading 3.5)
probeName	Name of the smFISH probe
RFNSpots	Number of local maxima classified as spots by the randomForest algorithm
trainingFileName	Name of the training set file used for the classification
nSpotsFinal	Final count of spots. Manual review and curation could change this

name and the number of the specimen in the segmented image. For example, if the image stack contains five specimens, the naming convention might be RFtestdatamatrix_tmr003_w3.txt where tmr003 is the unique stack identifier and this is the statistics matrix for the third specimen. The classification is stored in a cell array of structs called worms in the file tmr003_wormsGaussianFit.mat. Results from each specimen in the image stack will have its own struct in the cell array, and further functions in the software work with this data structure (Table 3). The use of "worms" and "w" in file names and in the software itself reflects its history of being developed and tested with data from nematodes.

One of the fields, spotInfo, contains information specific to a local maximum (Table 4).

3.5. Manually Review and Curate the Classification

Command:

reviewFISHSpotClassification(dye, uniqueStackIdentifier, worms);

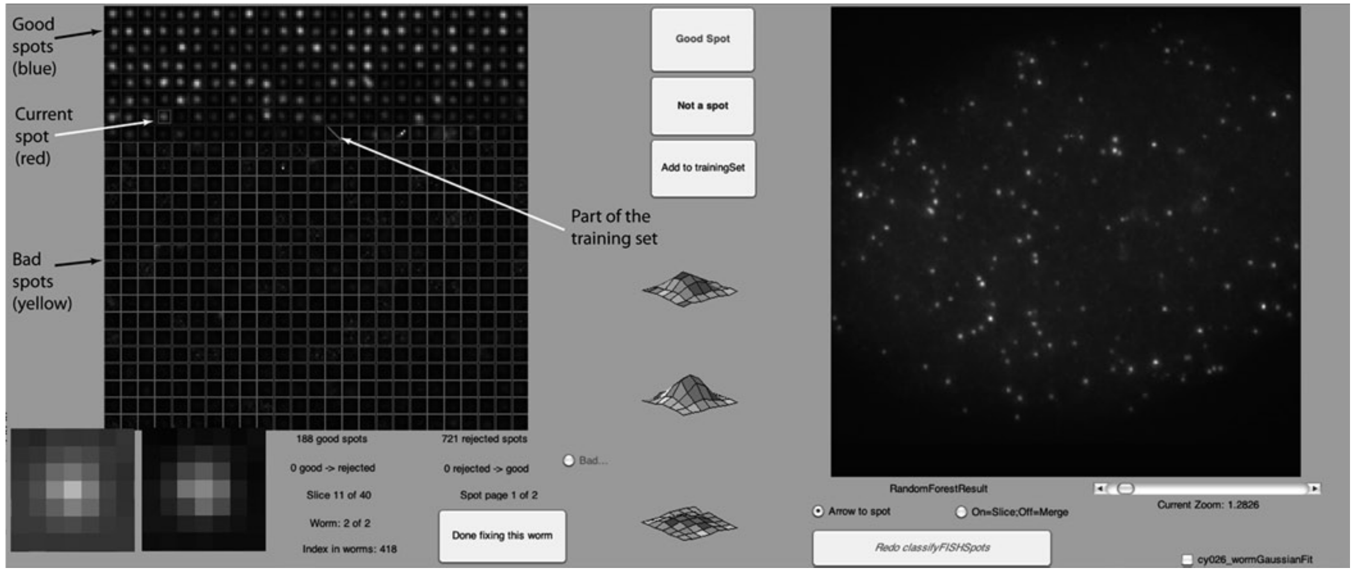
The final step in identifying mRNA spots consists of reviewing the automated classification and refining the classifier. The initial training set is likely to have an overabundance of easily classified local maxima. As a result, the classifier will not do an optimal job of classifying borderline cases. The manual review component of the software allows the user to correct misclassified local maxima and to add these to the training set. A new classifier can then be trained based on the augmented training data and applied to the image to generate a revised classification. As might be expected, adding marginal local maxima to the training set quickly improves the accuracy of the classifier. Perfect automated classification,

Table 4
Fields in worms{i}.spotInfo{j}

Locations	A struct containing [row, column, slice] location of the local maximum (a) in the entire stack and (b) relative to the BoundingBox of the specimen
rawValue	Raw intensity of the local maximum
filteredValue	Intensity of the local maximum after background subtractions
spotRank	Rank of the local maximum after background subtractions
dataMat	A square matrix of pixel intensities (default 7×7) centered on the local maximum
directory	A cell array containing the path to the working directory which contains the data and output files
dye	Fluorescent dye
stackSuffix	Unique identifier for the image stack
wormNumber	Specimen number (based on segmentation) within the image
statNames	The names of the statistics in the column order of the data
statData	The statistics calculated on the local maximum
machLearnResult	A quantitative measure of the machine learning classification. For the random forest algorithm, this is the fraction of trees that voted for spothood
classification	A struct containing the classification from the machine learning algorithm and the final classification, which could be modified after review

while theoretically possible, should not be the goal. As mentioned in Subheading 3.3, the randomForest function returns an estimate of the error rate based on the training data. The error rate for any given specimen can be calculated directly using the GUI described in this section. If the error rate is reasonably low, the user can choose to correct misclassified spots manually using the GUI (presumably achieving a perfect classification, at least in the eye of the user) or to deem the error rate acceptable and account for it in subsequent analyses. The user should refer to step three in Subheading 3.2 for criteria to use in evaluating spots. It can also be useful to switch between maximum merge and individual slice views in the reviewing GUI (see below, Fig. 4) to compare spot intensities between slices. In many samples, the distinction between spots and nonspots is stark and easy to score. In others, there will be more ambiguity. It might be useful to look at many stacks to get a feel for the variation in spot morphology and intensity, including negative controls where potential spots simply reflect noise.

The reviewing GUI window contains several images, information fields, and buttons (Fig. 4). On the left is a 25×25 grid of



188 good spots
721 rejected spots
0 good -> rejected
0 rejected -> good
Slice 11 of 40
Spot page 1 of 2
Worm: 2 of 2
Index in worms: 418

[✓ Arrow to spot] On=Slice;Off=Merge

Current Zoom: 1.2826

cy026_wormGaussianFit

Fig. 4. The reviewing GUI. Colors in the actual display are as marked.

evaluated local maxima and their (by default) 7×7 pixel surroundings, extracted from the image slice they are found in after scaling all intensities to lie between 0 and 1. For compactness, I refer to this 7×7 box as the *spot box*. Each spot box is initially edged in blue or yellow. A blue border means that the local maximum was classified as an mRNA spot. A yellow border means that it was rejected. Local maxima that are included in the training set have diagonal lines across their spot boxes. The spot box in current focus is highlighted by a red boundary (the *focal maximum*). The user can navigate around this grid using the mouse by clicking on different spot boxes or by the arrow keys. Only 625 spot boxes fit on a screen; page up/down will scroll to the next set. The local maxima are arranged in order of the fraction of spot votes they received in the random forest voting, from those that were unanimously classified as true spots to those that were unanimously rejected. This results in questionable calls being primarily grouped near each other around the place where blue boundaries yield to yellow ones. The GUI opens with the most marginal good spot in focus.

Below the spot box grid are two 7×7 pixel images. These portray the spot box in current focus. In the left one, the raw pixel intensities are tinted blue with the regional maximum tinted pink. The right one is gray scale and is equivalent to the scaled spot box in the large left image.

On the right-hand side of the GUI window is a large image that shows the specimen. The focal maximum will generally be in the middle unless its position relative to the bounding box of the specimen does not allow this. The putative spot is marked by a red arrow that can be toggled on or off. Initially, the large image contains the slice of the focal maximum. A toggle button below the image switches to a maximum merge across slices. Provided the mRNAs are not too dense, a true spot should show up in the maximum merge as well as in its own slice. A slider below the image allows the user to zoom from 1× to 6×.

Three surface plots in the middle of the window show the spot box and the 7×7 pixel squares above and below it. A toggle button allows the user to flag a specimen as bad, although bad specimens were probably flagged earlier in the pipeline. The window has text fields reporting the number of accepted and rejected spots, the number and direction of any corrections, and some identifiers of the focal maximum. A checkbox in the lower right indicates whether the image stack has already been reviewed or not.

Three buttons in the middle of the window help the user review and correct the classification. If the classification for the focal maximum is incorrect, the user can correct it from bad to good by clicking *Good Spot* and from good to bad by clicking *Not a Spot*. When the user changes a spot from bad to good, the yellow boundary changes to cyan, the numbers are updated, the local maximum is added to the training set, and the focus shifts to the next spot box. The same is true for the converse except that the boundary changes from blue to orange. A third button, *Add to trainingSet*, allows the user to add a local maximum to the training set without changing the machine classification. This option is useful for adding marginal, but correctly classified, examples to the training set.

At the bottom of the right-hand side of the window lies a button called *Redo classifySpots*. Clicking this button retrains the classifier on the training set, which has been augmented by any corrections or additions, and then reruns the classification on the current image stack. The GUI images will be updated to reflect this latest classification. The user can repeat this process until satisfied with the performance of the machine learning classifier. However, unless the user manually reviews each image stack in a dataset (making the user the final classifier), it is important to use the same iteration of the classifier on all the images in the dataset to avoid artifacts introduced by improvements in the training set.

All corrections are stored in the worms data structure. Final classifications are stored in the classification field of spotInfo (Table 3). The user can write a MATLAB script to cycle through them and collect absolute counts of single molecules or use the locations field to perform a spatial analysis.

4. Notes

1. In mathematical morphology, opening is an erosion followed by a dilation of an image by a smaller set of pixels called a structuring element (for example, a square of area 25 pixels or a disk of radius 7 pixels). The process can be pictured as follows: Imagine that the image is a landscape where the height above sea level reflects the pixel intensity. Opening an image is the equivalent of taking the structuring element, holding it horizontally, and running it along the underside of the landscape and recording the maximum height achieved by the structuring element at each pixel location. A small structuring element will track the surface with good fidelity, only lowering tiny bumps that are smaller than it is. A large structuring element will be too big to be pressed up into most hills and will end up razing the landscape. A structuring element that is larger, but not too much larger, than the real features of the image will do a decent job of capturing the varied local background of the features. The program then subtracts this background estimate from the actual image and uses the pixel values of the result to rank local maxima.

2. For ranking purposes, most systematic differences in the raw intensities of the slices will be removed by the local background correction. However, because the machine learning algorithm works with the raw data, large differences in intensity between slices could reduce the usefulness of intensity as a diagnostic statistic. Normally, intensity is one of the most important statistics. For this reason, it may be a good idea to adjust the raw intensities to remove any systematic, noninformative differences between slices.
3. Because of the slice-based normalization, out-of-focus slices and slices without any spots will usually appear bright even if their average raw intensities are low. Because the intensity of these slices comes from noise, the average pixel intensity will fall around 0.5. Slices that contain true spots would ideally have bright pixels around spots and very low pixel intensities elsewhere.
4. If the ranking algorithm works as designed, it should be rarely necessary to change a gray pixel to blue, thereby designating a nonlocal maximum as a spot. This could happen if molecules are close together and their coronas overlap such that one obscures the other. Usually, though, each will be a local maximum, but they will be closer than 7 pixels from each other. The machine learning algorithm does a surprisingly good job of identifying such pairs or triplets as individual spots even if no such overlapping spots are contained in the training set. Because local maxima are identified in three dimensions, there will be bright pixels that clearly radiate intensity that are not marked with blue. In these cases, the surface plots of the neighboring slices will show that this pixel belongs to a local maximum from a neighboring slice.
5. The goodness-of-fit statistic is calculated as follows. First one-dimensional Gaussian parameters are estimated from the (by default) 7 pixel horizontal and vertical slices through the maximum intensity pixel. The mean squared error of each of these slices from an ideal Gaussian distribution is calculated, and the goodness-of-fit statistic is the square root of their average mean squared error.
6. There is a crucial distinction between the cutoffs here and the cutoffs used in threshold-based procedures for identifying spots (1). Spot classification is done by the machine learning algorithm. Evaluating all local maxima would certainly be possible, but it would be overkill. The cutoffs here are designed to preselect a set of local maxima that is empirically guaranteed to contain all true spots. There are two cutoffs in this procedure: the 60th percentile and the 0.9 threshold for the statistic. These only determine the lowest ranked local maximum that

will be evaluated by the classifier. These cutoffs were empirically chosen such that they will never exclude any potential spot. Because they are so conservative, the software will include many more maxima than will be true spots. To modify the values of these parameters, change the values for cutoffPercentile and cutoffStatisticValue in the file evaluateFISHImageStack.m.

References

1. Raj A, van den Bogaard P, Rifkin SA et al (2008) Imaging individual mRNA molecules using multiple singly labeled probes. Nature Methods **5**:877–879
2. Paré A, Lemons D, Kosman D et al (2009) Visualization of individual Scr mRNAs during drosophila embryogenesis yields evidence for transcriptional bursting. Curr Biol **19**: 2037–2042
3. Lu J, Tsourkas A (2009) Imaging individual microRNAs in single mammalian cells in situ. Nucleic Acids Res **37**:e100
4. Femino AM, Fay FS, Fogarty K et al (1998) Visualization of single RNA transcripts in situ. Science **280**:585–590
5. Rodriguez AJ, Condeelis J, Singer RH et al (2007) Imaging mRNA movement from transcription sites to translation sites. Seminars Cell Dev Biol **18**:202–208
6. Betzig E, Patterson GH, Sougrat R et al (2006) Imaging intracellular fluorescent proteins at nanometer resolution. Science **313**:1642–1645
7. Rust MJ, Bates M, Zhuang X (2006) Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (STORM). Nature Methods **3**:793–796
8. Fusco D, Accornero N, Lavoie B et al (2003) Single mRNA molecules demonstrate probabilistic movement in living mammalian cells. Curr Biol **13**:161–167
9. MathWorks (2010) MATLAB. Retrieved from http://www.mathworks.com
10. Rifkin SA (2010). *spotFinding Suite*. http://www.biology.ucsd.edu/labs/rifkin/software.html
11. Raj A, Tyagi S (2010) Detection of individual endogenous RNA transcripts in situ using multiple singly labeled probes. In: Walter NG (ed), Single Molecule Tools: Fluorescence Based Approaches, Part A. Academic Press
12. Liaw A, Wiener M (2002) Classification and regression by randomForest. R News **2**: 18–22
13. Breiman L (2001) Random forests. Machine Learning **45**:5–32
14. Breiman L, Cutler A (2001) Random Forests. http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm