Maximum precision closed-form solution for localizing diffraction-limited spots in noisy images

Joshua D. Larkin and Peter R. Cook[*]

Sir William Dunn School of Pathology, University of Oxford, South Parks Road, Oxford, OX1 3RE, UK
[*]peter.cook@path.ox.ac.uk

Abstract: Super-resolution techniques like PALM and STORM require accurate localization of single fluorophores detected using a CCD. Popular localization algorithms inefficiently assume each photon registered by a pixel can only come from an area in the specimen corresponding to that pixel (not from neighboring areas), before iteratively (slowly) fitting a Gaussian to pixel intensity; they fail with noisy images. We present an alternative; a probability distribution extending over many pixels is assigned to each photon, and independent distributions are joined to describe emitter location. We compare algorithms, and recommend which serves best under different conditions. At low signal-to-noise ratios, ours is 2-fold more precise than others, and 2 orders of magnitude faster; at high ratios, it closely approximates the maximum likelihood estimate.

OCIS codes: (180.2520) Fluorescence microscopy; (100.6640) Superresolution; (100.4999) Pattern recognition, target tracking.

References and links

1. E. Betzig, G. H. Patterson, R. Sougrat, O. W. Lindwasser, S. Olenych, J. S. Bonifacino, M. W. Davidson, J. Lippincott-Schwartz, and H. F. Hess, "Imaging intracellular fluorescent proteins at nanometer resolution," Science **313**(5793), 1642–1645 (2006).
2. M. J. Rust, M. Bates, and X. Zhuang, "Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (STORM)," Nat. Methods **3**(10), 793–796 (2006).
3. R. J. Ober, S. Ram, and E. S. Ward, "Localization accuracy in single-molecule microscopy," Biophys. J. **86**(2), 1185–1200 (2004).
4. A. V. Abraham, S. Ram, J. Chao, E. S. Ward, and R. J. Ober, "Quantitative study of single molecule location estimation techniques," Opt. Express **17**(26), 23352–23373 (2009).
5. K. I. Mortensen, L. S. Churchman, J. A. Spudich, and H. Flyvbjerg, "Optimized localization analysis for single-molecule tracking and super-resolution microscopy," Nat. Methods **7**(5), 377–381 (2010).
6. P. N. Hedde, J. Fuchs, F. Oswald, J. Wiedenmann, and G. U. Nienhaus, "Online image analysis software for photoactivation localization microscopy," Nat. Methods **6**(10), 689–690 (2009).
7. C. S. Smith, N. Joseph, B. Rieger, and K. A. Lidke, "Fast, single-molecule localization that achieves theoretically minimum uncertainty," Nat. Methods **7**(5), 373–375 (2010).
8. S. Wolter, M. SchãœTtpelz, M. Tscherepanow, S. Van De Linde, M. Heilemann, and M. Sauer, "Real-time computation of subdiffraction-resolution fluorescence images," J. Microsc. **237**(1), 12–22 (2010).
9. S. J. Holden, S. Uphoff, and A. N. Kapanidis, "DAOSTORM: an algorithm for high- density super-resolution microscopy," Nat. Methods **8**(4), 279–280 (2011).
10. M. K. Cheezum, W. F. Walker, and W. H. Guilford, "Quantitative comparison of algorithms for tracking single fluorescent particles," Biophys. J. **81**(4), 2378–2388 (2001).
11. J. D. Larkin, N. G. Publicover, and J. L. Sutko, "Photon event distribution sampling: an image formation technique for scanning microscopes that permits tracking of sub-diffraction particles with high spatial and temporal resolutions," J. Microsc. **241**(1), 54–68 (2011).
12. K. A. Winick, "Cramer-Rao lower bounds on the performance of charge-coupled-device optical position estimators," J. Opt. Soc. Am. A **3**(11), 1809–1815 (1986).
13. Derek Elsaesser, "The discrete probability density method for emitter geolocation," in Canadian Conference on Electrical and Computer Engineering, 2006. CCECE '06 (IEEE, 2006), 25–30.
14. R. E. Thompson, D. R. Larson, and W. W. Webb, "Precise nanometer localization analysis for individual fluorescent probes," Biophys. J. **82**(5), 2775–2783 (2002).

#169676 - $15.00 USD Received 8 Jun 2012; revised 19 Jul 2012; accepted 22 Jul 2012; published 27 Jul 2012

15. T. A. Laurence and B. A. Chromy, "Efficient maximum likelihood estimator fitting of histograms," Nat. Methods **7**(5), 338–339 (2010).
16. A. Löschberger, S. van de Linde, M.-C. Dabauvalle, B. Rieger, M. Heilemann, G. Krohne, and M. Sauer, "Super-resolution imaging visualizes the eightfold symmetry of gp210 proteins around the nuclear pore complex and resolves the central channel with nanometer resolution," J. Cell Sci. **125**(3), 570–575 (2012).
17. M. Heilemann, S. van de Linde, M. Schüttpelz, R. Kasper, B. Seefeldt, A. Mukherjee, P. Tinnefeld, and M. Sauer, "Subdiffraction-resolution fluorescence imaging with conventional fluorescent probes," Angew. Chem. Int. Ed. Engl. **47**(33), 6172–6176 (2008).
18. J. B. Pawley, "Points, pixels, and gray levels: digitizing image data," in *Handbook of Biological Confocal Microscopy*, 3rd ed. (Springer, 2006).
19. D. R. Larson, "The economy of photons," Nat. Methods **7**(5), 357–359 (2010).
20. A. J. Berglund, M. D. McMahon, J. J. McClelland, and J. A. Liddle, "Fast, bias-free algorithm for tracking single particles with variable size and shape," Opt. Express **16**(18), 14064–14075 (2008).
21. S. B. Andersson, "Precise localization of fluorescent probes without numerical fitting," 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2007. ISBI 2007 252–255 (2007).
22. P. G. Hoel, *Introduction to Mathematical Statistics*, 5th ed. (Wiley, 1984).
23. M. Xu and P. R. Cook, "Similar active genes cluster in specialized transcription factories," J. Cell Biol. **181**(4), 615–623 (2008).
24. R. Tam, L. S. Shopland, C. V. Johnson, J. A. McNeil, and J. B. Lawrence, "Applications of RNA FISH for visualizing gene expression and nuclear architecture," in *Fish*, 1st ed. (Oxford University Press, USA, 2002), 93–118.
25. A. Papantonis, J. D. Larkin, Y. Wada, Y. Ohta, S. Ihara, T. Kodama, and P. R. Cook, "Active RNA polymerases: mobile or immobile molecular machines?" PLoS Biol. **8**(7), e1000419 (2010).

Introduction

Techniques for 'super-resolution' fluorescence microscopy like PALM (photo-activation localization microscopy) [1] and STORM (stochastic optical reconstruction microscopy) [2] depend upon precise localization of single fluorophores. Such localization represents a challenge, as photons emitted from a point source are detected by a CCD to yield a pixelated image; then, relevant information in the pixels must be used to deduce the true location of the point source. The various localization methods currently in use differ in precision and speed. For example, minimizing least-square distances (MLS) and maximum likelihood estimation (MLE) fit a Gaussian distribution to pixel intensities before estimating a fluor's location; MLS is the most popular but less precise, while MLE is more involved but can achieve the theoretical minimum uncertainty [3–5]. Both are iterative and so computationally intensive; consequently, attempts have been made to maximize accuracy and minimize computation time [6–9]. More problematic, fitting implies an underlying model, which can introduce errors, especially at low signal-to-noise ratios (S:N). The straightforward center-of-mass (CM) estimate [10] has the advantages of simplicity and speed, but is considered less accurate than the iterative methods (mistakenly, as we shall see); as a result, it is not being used for PALM/STORM.

Borrowing principles from 'pixel-less' imaging – a technique that uses a photomultiplier as a detector [11] – we present a non-iterative (and so rapid) way of localizing fluors imaged with a CCD. Each photon registered in the image carries spatial information about the location of its source. As this information is blurred by the point-spread function (PSF) of the microscope, we use the PSF to define many independent probability distributions that describe the emitter's possible locations – one for each photon in the population (Fig. 1(a)). We then assume that all photons came from the same emitter (the usual and fundamental basis of localization), and aggregate probability distributions; the result is a joint distribution (JD) of the probability of the emitter's location (Figs. 1(b) and 1(c)). Localization by JD is similar to a weighted form of CM, offering advantages in simplicity and speed, and – for the curious practitioner – we detail the differences between the two. We also compare the performance of the various methods both quantitatively (using computer-generated images) and qualitatively (using 'real' images). Our results enable us to recommend which approach to use with images containing different degrees of noise, depending on whether precision or speed is the priority. We find that the most popular – MLS – is never the algorithm of choice. At high signal-to-noise ratios, MLE yields the highest precision, while JD offers a quick, closed-form alternative; with very noisy images (where both MLS and MLE fail) JD proves the most accurate.

Results

Theory

In a typical single-molecule or PALM/STORM experiment, an image is acquired by collecting photons from temporally- and spatially-isolated emitters using a CCD. As many photons fall on one pixel, this is analogous to binning data into a histogram, with loss of sub-pixel (sub-bin) spatial information. We will think of individual photons as independent carriers of spatial information. Then, given a pixel that has registered one photon, conventional localization methods (such as MLE, MLS, and CM) would treat a photon as having a spatial distribution represented by the blue line in Fig. 1(a). The photon has a probability density function (i.e., the probability of the location of the source of that photon) that is uniformly flat over the whole area of the pixel, giving a 2-D rectangular or 'top-hat' distribution, with zero probability in neighboring pixels. [Note that this probability density function refers to one photon and not to many.] In other words, uncertainty is inaccurately recorded as a uniform distribution over just one pixel. In contrast, JD localization represents this uncertainty as a normal distribution that spreads over several pixels (see the one red curve in Fig. 1(a), and the many red curves in Fig. 1(b)).

The PSF serves as an initial estimate of the uncertainty imparted on the position of every photon by the microscope, and we initially use a normal distribution to approximate it [12] (as is common in the field). Such a distribution is uniquely described by center location (μ_i) and width (σ_i). Our default is to place μ_i at the center of a pixel and use σ_i equivalent to that of the PSF (Fig. 1(c), left); alternatively, μ_i and/or σ_i can be varied to suit the needs of a particular experiment (Fig. 1(c), *right*). After applying a distribution to each photon, distributions are aggregated to infer the probability of the location of the emitter (Figs. 1(b) and 1(c)). [Similar joining of independent probability distributions has been proposed for geolocation [13].] We have derived a simple equation to facilitate closed-form (non-iterative) – and so rapid – calculation.

$$\mu_o = \sum_i^N \left(\mu_i \sigma_i^{-2}\right) \cdot \left(\sum_i^N \sigma_i^{-2}\right)^{-1}. \quad (1)$$

Here, μ_o is the best estimate of the location of the emitter and N is the number of photons. [See Methods at the end of the manuscript for derivation.]

In contrast, methods such as MLS and MLE describe the probability of emitter location by fitting a curve to pixel intensities, which involves many sequential calculations, then deducing location information from that curve. We now benchmark test the different methods, first for precision and then for speed.

Quantitative comparison of precision

To assess precision, we use computer-generated images of a point source whose location is known. In each simulation, a 'point source' emits a known number of 'photons' that 'pass' through a 'microscope' (to be blurred by the PSF) to yield an image (initially 15x15 pixels) on a 'CCD'; then, a specified number of 'background photons' are added. Using 10,000 such images for each condition analyzed, we go on to compute the 1-D root-mean-squared error (RMSE) between the true location of the emitter and the location estimated using each of the four methods.

#169676 - $15.00 USD Received 8 Jun 2012; revised 19 Jul 2012; accepted 22 Jul 2012; published 27 Jul 2012

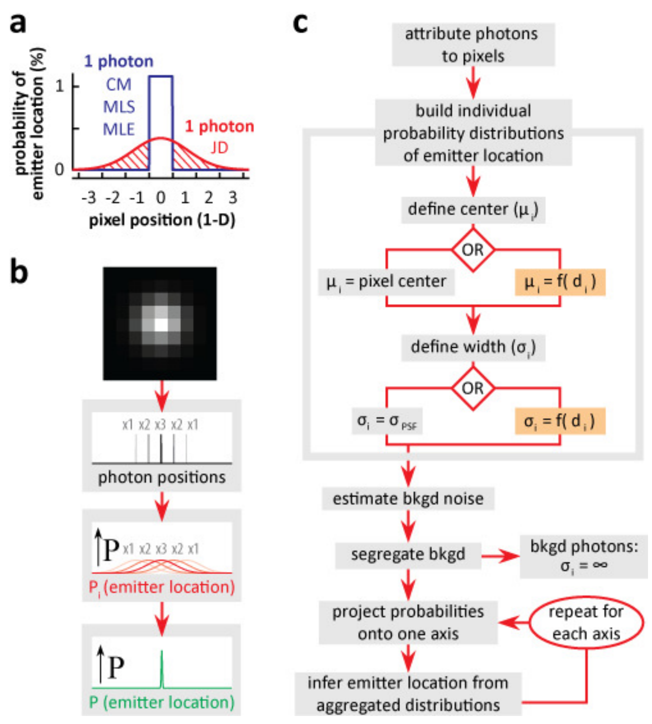


Fig. 1. JD localization. (a) An individual photon carries information about the probability of an emitter's location. CM/MLS/MLE (*blue*) assume equal probability throughout the pixel, with zero probability elsewhere; JD localization (*red*) uses a normal distribution to emulate the microscope PSF (FWHM = 2.8x pixel width) which spreads beyond the pixel (*shaded area*). The area under both curves is equal. (b) Principles behind JD localization. Each photon represented in a pixel is treated individually (1, 2, and 3, photons indicated by *x1*, *x2*, *x3*), and individual probabilities of the location of the emitter, P_i (*red curves*) are aggregated to yield the joint probability, P, of emitter location (*green*). (c) Flow diagram for JD localization. Photons are attributed to each pixel dependent on intensity, and a probability distribution of the source of each photon is built using peak location (μ_i) and width (σ_i). The peak can be located at the pixel centre (*left*) or anywhere within or outside the pixel (*right*); the width can be that of the PSF (*left*) or any arbitrary value (e.g., as a function of distance, d_i, from the most intense pixel; *right*). Background (bkgd) noise is estimated and the influence of background photons nullified by setting $\sigma_i = \infty$. Then, probabilities of individual photons are projected onto a single axis, combined to infer the probability of emitter location for that axis, and the projection and inference repeated for each axis.

In the first analyses (Figs. 2(a) and 2(b)), we apply JD using default settings (i.e., with μ_i set at the pixel center, and σ_i equivalent to that of the PSF); then, the JD equation simplifies to that used in CM (Methods). To aid comparison, we also plot the theoretical minimum uncertainty that is attainable under the particular conditions used – a lower bound (LB) computed using Eq. (6) of Thompson *et al*. [14]. This LB excludes effects of background noise, but includes those due to pixel size and PSF, and so differences from the LB reflect the influence of background noise on a method.

We first consider the case where background is absent (b = 0; Fig. 2(a)). As expected, errors in localization given by all four methods decrease as the number of photons increases. Those given by CM and JD lie on the LB at all photon counts tested. Below ~30 photons, MLS and MLE 'fail'; they either do not converge to a solution during the 200 iterations allowed, or yield a 1-D RMSE > 1 pixel (so values are not shown here) – and they sometimes even return a location outside the image (presumably because spot shape diverges significantly from a Gaussian; Supplemental Fig. 1, Media 1). As photon count increases, MLE is

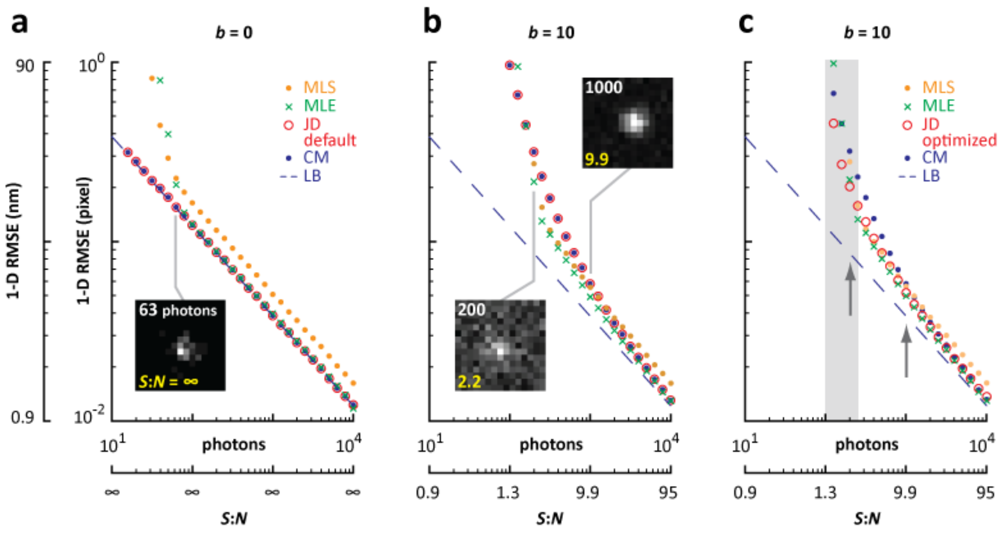


Fig. 2. Qualitative comparison of methods. Computer-generated images (15x15 pixels) like those illustrated were generated using 10-10^4 emitter photons and different numbers of background photons (i.e., b = 0 or 10 photons/pixel) to give different signal-to-noise ratios (*S*:*N*); then, the root-mean-square error of localization in one dimension (1-D RMSE in nm or pixel units) was calculated (10^4 localizations per data point) using the methods indicated. Photon counts (*top*) and *S*:*N* (*bottom*) are indicated in some typical images. The lower bound, LB (*blue dashed line*), is computed using Eq. (6) of Thompson *et al*. [14] and plotted here and in subsequent Figures as a reference. (a) With no background (b = 0), the 'default' version of JD returns the same results as CM, and both track the LB; MLS and MLE fail at low photon counts. The failure of MLS without background is examined more in Supplemental Fig. 1(b). In the presence of background (b = 10), all methods fail at low photon counts; at moderate counts, MLE performs best, and at high counts MLS is the worst as the others converge to the minimum error. (c) An 'optimized' version of JD increases precision at low *S*:*N* while retaining precision at high *S*:*N*. The grey region is analyzed further in Fig. 3. Arrows: conditions used in Fig. 4.

initially less accurate than MLS, but then errors fall progressively to reach the LB above ~100 photons. Errors given by MLS converge to a level 30% greater than the LB, as is well documented [4, 5, 15].

We now randomly add an average of 10 background photons per pixel (i.e., b = 10; Fig. 2(b)). At the very lowest signal-to-noise ratio, all methods fail (in the case of JD and CM, only because 1-D RMSE > 1 pixel). As the ratio progressively increases, JD and CM (when corrected for background; Methods) are the first to return a 1-D RMSE of less than 1 pixel, and then MLE and JD/CM (in that order) converge to the LB. Most PALM/STORM images are formed from data with *S*:*N* >5 (e.g., Löschberger *et al.* [16]), where MLE returns between 8 and 27% less RMSE than MLS.

As JD treats each photon separately, individual distributions can be tuned independently to optimize the precision and/or speed achieved at a given signal-to-noise ratio. As a first example, we eliminate the effects of outlying bright pixels that are likely to result from noise. As the PSF falls off precipitously from the central peak, few photons emitted by a point source will be detected in the image plane > 3σ distant from the true location. Then, we consider all signal detected > 3σ from the center of the brightest pixel to be noise (i.e., > 3.5 pixels away), and nullify its effects on the JD by ascribing $\sigma_i = \infty$ to each of its constituent distributions. This simple 'optimized' version of JD improves accuracy (compared with CM) over a wide range of *S*:*N* (Fig. 2(c)). It is also more precise than MLE at *S*:*N* < 2.7, than MLS at *S*:*N* < 3.0 and > 4.5, and it returns results within 5% of MLE at *S*:*N* > 7.

Tuning JD variables

When applying JD, we hitherto set μ_i = pixel center and $\sigma_i = \sigma_{PSF}$; we now tune each to maximize localization precision in noisy images (*grey region* in Fig. 2(c)) where MLS and MLE fail – first varying each one alone, and then both together. [Just as the default version of JD and CM produce the same results, we expect a tuned version of JD and an equivalent weighted CM variant (if developed) to do so too. However, we differentiate between JD and CM for several inter-related reasons: (i) By the strictest definition, CM weights each pixel position solely by a 'mass' equivalent to intensity; in contrast, in JD, μ_i and σ_i can be varied depending on distance from (and position relative to) the brightest pixel (with intensity determining the number of distributions to be joined together). (ii) Conceptually, CM applies statistics to a population of photons, whilst JD disaggregates the population into individual photons and then combines individual probabilities (with a consequential reduction in speed; below). (iii) In principle, it should be possible to derive a general form of CM that would allow tuning of the piecewise weightings of pixel positions to yield the same precision as the JD variants (below), but such a generalization would inevitably mean that the CM equation loses its characteristic simplicity.]

Consider Fig. 3(a), and the selected photon distributions (*blue curves*) in the cartoon on the left. By default, μ_i is placed at the center of the CCD pixel registering the photon (*blue dots*), even though that photon was probably emitted by a fluor in the central (brightest) pixel in the specimen plane. Therefore, the *x*- and *y*-coordinates of μ_i associated with all distributions – except those derived from the brightest pixel – are shifted between $^1/_5$ – 1 pixel towards the brightest pixel (*red dots* mark new positions for a ½-pixel shift). Distributions from the brightest pixel are also shifted from the central default location by a distance proportional to the intensities of adjacent pixels (Methods, Eqs. (2) and (3)). [In all cases, σ_i remains constant and equal to σ_{PSF}.] A shift of ½-pixel width yields the least error (not shown), giving a ~5% reduction at $S{:}N < 3$ (Fig. 3(a), *right*).

Now consider Fig. 3(b). By default, σ_i is the width of the Gaussian that emulates the microscope's PSF. As an emitter is most likely to lie in the brightest pixel, we expand distributions from other pixels (in the cartoon, the outer *blue halo* expands to give the outer dilated *red* one); distributions from pixels lying progressively further away from the brightest are expanded progressively more (Methods, Eqs. (4) and (5)). Distributions from the brightest pixel and its immediate neighbors remain unchanged (in the cartoon, the central *blue halo* gives an unchanged *red halo*). JD now yields up to 36% less error than CM (Fig. 3(b)); however, this comes at the price of higher error at higher signal-to-noise ratios (Supplemental Fig. 2, *left*).

We now combine both strategies. It turns out that an *x*-*y* shift in μ_i of ¼ pixel (not ½ pixel as in Fig. 3(a)) coupled with σ_i broadening (as in Fig. 3(b)) realizes up to 42% less error than CM – and 51% less than MLE – at $S{:}N = 1.6$ (Fig. 3(c)). Note that MLS begins to break down at $S{:}N = 2.4$ and fails completely below $S{:}N = 1.9$, while MLE never performs the best in this noisy region. In conclusion, this 'tuned' version of JD exhibits less error than (i) MLS at $S{:}N < 2.4$ and $5 < S{:}N < 38$, (ii) MLE at $S{:}N < 2.7$, and (iii) CM at $1.3 < S{:}N < 24$ (see also Supplemental Fig. 2, *right* (Media 1)).

Efficacy of localization algorithms is known to vary with image size [7] and the position of the spot within the image [10]; for example, at low *S*:*N*, CM favors the geometric center of the image. Therefore, we assessed the effects of reducing the size of the image window (from 15x15 to 7x7 pixels) and the position of the emitter relative to the center of the window (by up to 4 pixels), and found that the tuned version of JD still performs better and more robustly than the others under noisy conditions (Supplemental Fig. 3).

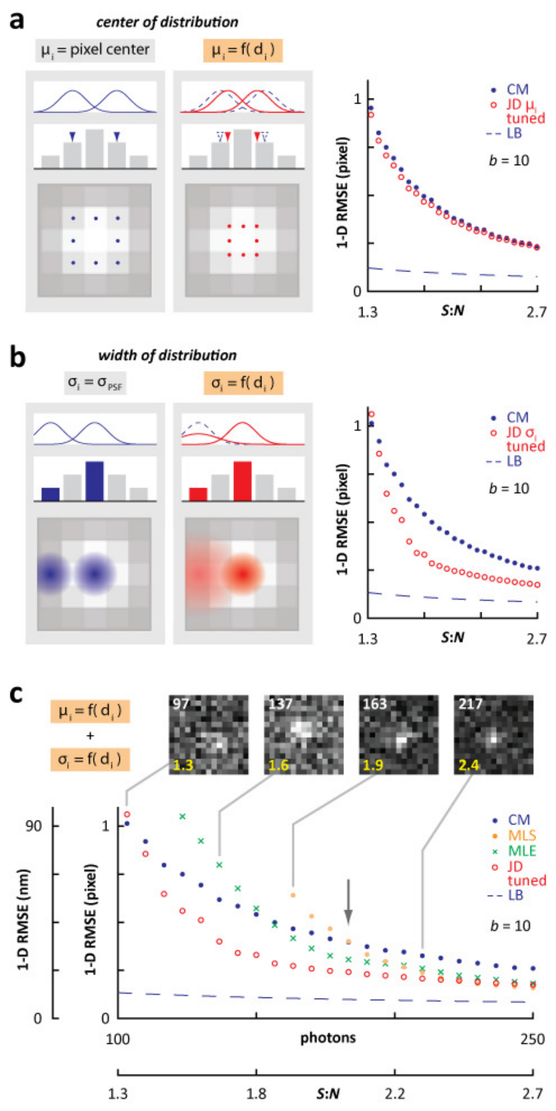


Fig. 3. Tuning μ_i and σ_i to maximize localization precision near the detection limit. The 1-D RMSE in location of a point source was determined using 10^4 computer-generated images (15x15 pixels, 50-250 emitter photons, $b = 10$) per data point. JD localizations were determined by varying peak position (μ_i) and/or width (σ_i). Errors obtained using CM, MLS, and MLE, plus the lower bound (LB), are included for comparison (MLS/MLE plots smoothed by linear regression). (a) Varying μ_i depending on distance, d_i, from the most intense pixel ($\sigma_i = \sigma_{PSF}$ for all distributions). Peaks of distributions are shifted towards the brightest pixel (in this example the central one) by ½ a pixel in both *x*- and *y*-dimensions. *Left*: cartoon illustrating how this shift applies to selected distributions (from *blue curves/dots* to *red curves/dots*). *Right*: JD localization yields 5% less error than CM for all *S*:*N* shown. (b) Varying σ_i as a function of distance, d_i, from the most intense pixel (as μ_i = pixel center for all distributions). Widths of distributions from the brightest pixel remain equal to σ_{PSF}, as those from surrounding ones expand. *Left*: cartoon illustrating these changes for selected distributions (from *blue curves/halos* to *red curves/halos*). Localization using JD yields up to 36% less error than CM. (c) Varying both peak position and width (as in (a) and (b), but using an *x* and *y* shift of ¼ pixel for μ_i). Representative images are shown (with photon counts given in *white*, and *S*:*N* in *yellow*). At this low *S*:*N*, JD localization yields less error than other methods. Arrow: condition used in Fig. 4. Error over the full range of *S*:*N* is shown in Supplemental Fig. 2. The effects of window size and offset of emitter from window center are shown in Supplemental Fig. 3.

Received 8 Jun 2012; revised 19 Jul 2012; accepted 22 Jul 2012; published 27 Jul 2012

Computation speed

To assess computation speed, we compared (using windows with 15x15, 13x13, and 10x10 pixels) the number of 2-D localizations per second using images with two *S*:*N* ratios (indicated by arrows in Figs. 2(c) and 3(c)). As expected, higher *S*:*N* inevitably favors fast solution by the two iterative approaches (MLE and MLS), but both were slower than the 'optimized' and 'tuned' versions of JD, and much slower than CM (Fig. 4).

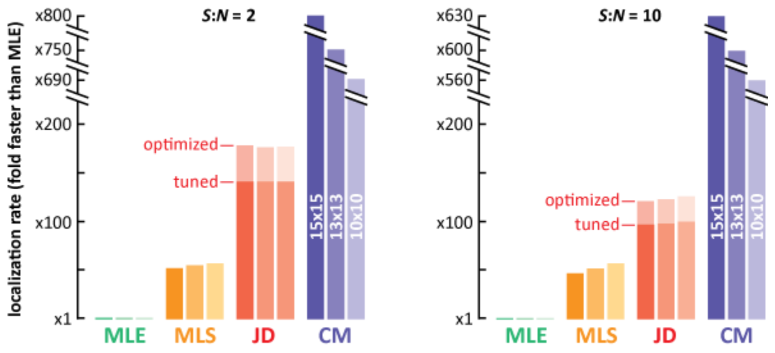


Fig. 4. Computation speeds of the different methods (expressed relative to that of MLE). The times taken by the different methods to compute 2-D localizations were determined using 10^4 computer-generated images (15x15, 13x13, or 10x10 pixels) using conditions at the points indicated by the arrows in Figs. 2(c) and 3(c) (either 183 photons, b = 10, and S:N = 2; or 1,000 photons, b = 10, and S:N = 10). JD was tested using both 'optimized' and 'tuned' versions. Our MLS script gave 390 localizations per second with 15x15-pixel images, which is even faster than other reports on comparable computers [7]; it was also 50-times faster than an MLE script written by others [5] but implemented by us. JD applied using the 'optimized' conditions was 120- to 180-times faster than MLE, and the 'tuned' version was 100- to 140-times faster than MLE. As expected, CM (applied with background correction) proved the fastest, but both JD versions were faster than the two iterative techniques.

Localization using images of biological samples

We next compared performance of the four approaches using two kinds of images of biological samples; unfortunately, the true location of fluors in both samples cannot be known, so only qualitative comparisons can be made.

In the first example, RNA fluorescence *in situ* hybridization (RNA FISH) was used to tag, with Alexa 467, a nascent RNA molecule at a transcription site in a nucleus; then, images of the resulting foci were collected using a wide-field microscope. One-hundred images with a *S*:*N* < 3 were chosen manually, passed to the four algorithms, and the resulting localizations superimposed on each image; typical results are illustrated (Fig. 5(a); Media 2 gives results for all 100 spots). Visual inspection suggests that the tuned version of JD performs at least as well as, if not better than, the other methods.

Microtubules imaged using direct STORM (dSTORM) [17] provide the second example. Tubulin in fixed cells was indirectly immuno-labeled with Alexa 647, $3x10^4$ images of temporally- and spatially-separated single flours in the same field collected, and $1.5x10^5$ windows (11x11 pixels) containing 1 centrally-located spot selected for analysis using a Gaussian spot-finding algorithm (Fig. 5(b)i illustrates a mean projection of all windows). A typical window contained one spot with *S*:*N* > 9 (Fig. 5(b)ii). Individual windows were then

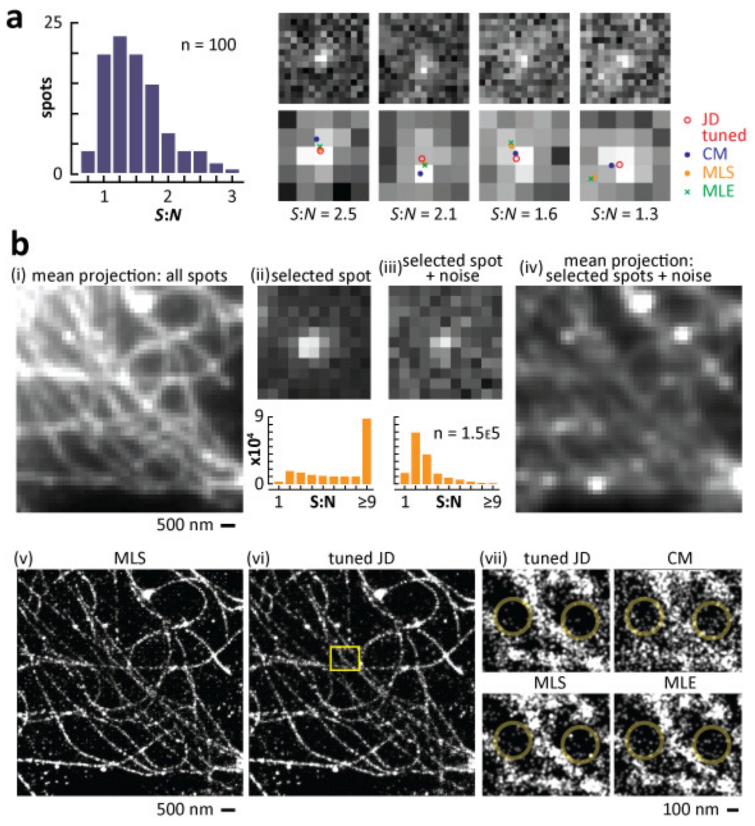


Fig. 5. Localization using ‘real’ images of transcription sites and microtubules in monkey cells (cos-7). (a) Nascent RNA at transcription sites. Cells expressing an *EGFP* gene containing an intron were fixed, and (nascent) EGFP transcripts detected using RNA FISH with probes targeting a short (sub-resolution) segment of the intron; images were collected using a wide-field microscope and CCD (90-nm pixels). One-hundred spots with $S{:}N < 3$ (*histogram*) were chosen manually, and four examples are shown at the top; the panels below illustrate the central 5x5 pixels in the upper panels, with 2-D localizations obtained by the different methods. As $S{:}N$ decreases (*left-to-right*), localizations become more scattered (see Media 2 for results with all 100 spots). (b) Microtubules. Cells were fixed, microtubules indirectly immuno-labeled with Alexa 647, and a series of 30,000 images of temporally- and spatially-separated spots of one field collected using inclined illumination and an EM-CCD (155-nm pixels); 154,040 windows (11x11 pixels) containing 1 centrally-located spot were selected for analysis (using a Gaussian spot-finding algorithm). (i) Mean projection of all windows. (ii) One representative window (the *histogram* below illustrates the number of windows with different $S{:}N$). (iii) Individual windows were deliberately corrupted with noise (typical example and histogram shown). (iv) Mean projection of all resulting windows. Individual windows were now passed to each of the four methods, and localizations convolved with a 20-nm Gaussian intensity profile to aid visualization. (v, vi) Localizations obtained by MLS and the tuned version of JD yield roughly equivalent images. (vii) Magnified areas of the inset in (vi). Large circles in JD images contain fewer isolated results than the others, consistent with fewer mis-localizations (see also Supplemental Fig. 4(d)).

deliberately corrupted with a known level of noise (Fig. 5(b)iii and iv) – in this case to reduce $S{:}N$ to less than 3 (after noise was added, spots had a mean $S{:}N$ of 2.8 and a 71% had a $S{:}N < 3$). [See Supplemental Fig. 4 (Media 1) for a comparison of results obtained using uncorrupted and corrupted windows.] Despite the noisy images, all but 14% of spots are still detected by our simple spot-finding algorithm (not shown; spots found by the algorithm had a mean $S{:}N$ of 2.8, and 71% had a $S{:}N < 3$, so the 14% were missed as a result of random chance and not low $S{:}N$). All windows were passed to the four algorithms, and localizations convolved with a 20-

nm Gaussian intensity profile to aid visualization. MLS (chosen as an example because it is used most-often during the formation of STORM images) and the tuned version of JD yield roughly equivalent images (Fig. 5(b)v and vi), although analysis of nearest-neighbor distances indicates JD returns the most highly-structured images (Supplemental Fig. 4(g)). It also yields fewer isolated results than the others (*yellow circles* in Fig. 5(b)vi), which we assume are mis-localizations resulting from poor performance. We again conclude that JD performs better with noisy images than methods used traditionally.

Discussion

During the application of 'super-resolution' techniques like PALM and STORM, photons emitted from a point source pass through a microscope to yield an image on a CCD where they are registered by many pixels. Successful localization of the point source then depends on two critical steps. First, the pixelated 'spot' must be distinguished from others and the inevitable background; we have not studied this step (we apply it only in Fig. 5(b) where we rely on a cross-correlation-based 'spot-finding' algorithm to identify spots with S:N < 3). Second, the position of the point-source must be deduced using the relevant information in the isolated pixels. We introduce a method for performing this second step. Existing methods (e.g., MLE, MLS, and CM) inaccurately assume the probability of the location of each emitted photon is uniformly distributed over just one pixel; in contrast, our method represents this uncertainty as a normal distribution that spreads over several pixels (Fig. 1(a)). We then aggregate many probability distributions to yield a joint distribution (JD) of the probability of the location of the emitter (Figs. 1(b) and 1(c)).

Localization by JD has the advantage of flexibility; each individual probability distribution is defined solely by peak center (μ_i) and width (σ_i), and both can be tuned to improve precision to meet the needs of a particular experiment (Fig. 3). We anticipate that additional tuning of μ_i and σ_i (e.g., as functions of pixel intensity), and further optimization (e.g., of the rate at which σ_i increases as a function of distance) – will improve precision even further. In images where the PSF deviates from the ideal, different tuning parameters might maximize precision. Moreover, the use of smaller pixels should also increase precision, as μ_i could then be assigned more precisely. This can be accomplished, in spite of traditional knowledge that reducing pixel size decreases precision [18], by applying distributions that represent the PSF to each detected photon and summing overlapping regions to form complex images [11]. [Here, images have 90-nm pixels so as to meet the Nyquist criterion for a PSF with a 250-nm full width half maximum. Preliminary simulations indicate that a reduction in pixel width to 1 nm reduces the 1-D RMSE in localization by an additional 3%.]

All versions of JD provide computational simplicity and speed because emitter location is not calculated iteratively. Furthermore, all adeptly localize in windows with non-uniform background, as broadening individual distributions negates the influence of bright pixels distant from the brightest. They are also readily extended to both 3D localization (given a Gaussian-like PSF in the axial dimension, computation of a third dimension is straightforward because each axis is treated independently) and more than one color – and so to real-time imaging deep within living specimens. Nevertheless, they have several disadvantages. First, unlike the two fitting algorithms that 're-check' spots selected by a spot-finding algorithm for an appropriate Gaussian intensity profile, JD (and CM) provide no such back-up. [Tests of various spot-finding algorithms suggest that local-maxima techniques are liable to return multiple spots in one window, but 2-D normalized cross-correlation with a Gaussian kernel robustly selected single spots from dSTORM data (not shown).] Second, the initial disaggregation of pixel intensity into individual photons followed by the aggregation of individual probabilities into a joint distribution inevitably makes JD slower than CM. Third, the greatest gains at low S:N (from the 'tuned' version) come at the cost of precision at high S:N. Fourth, the JD scheme fails completely when the brightest pixel in a window does not contain the emitter.

We compared accuracy and speed of localization achieved by various methods using images with a wide range of noise, and find that each has its own advantages and disadvantages (Figs. 2–4). Although widely used [19], we suggest MLS should rarely, if ever, be the algorithm of choice. At high signal-to-noise ratios, MLE – though the slowest – is the most accurate (as reported by others [3–5, 7, 15]). At the highest signal-to-noise ratios, CM is only marginally less accurate than MLE; at signal-to-noise ratios > 10, CM offers greater precision than MLS. [A variant of CM involving a limited number of iterative computations is even more accurate than the basic version [20].] If temporal resolution is of the greatest concern (e.g., during real-time computation), CM is by far the fastest (Fig. 4), and its simplicity makes it attractive to groups lacking sophisticated analysis software. [Other closed-form solutions also produce fast results, but at the cost of precision [6, 21]]. Most PALM/STORM images currently being analyzed have a $S{:}N > 9$ (as in the uncorrupted spot in Fig. 5(b)ii), where MLE yields the highest precision (13-27% and 16-0.5% less 1-D RMSE per pixel than MLS and CM, respectively). However, as the signal-to-noise ratio falls, both MLE and MLS fail to converge to a solution during the 200 iterations used, or yield an error > 1 pixel; then, the tuned version of JD becomes the most accurate. For example, when $S{:}N = 1.6$, the tuned version returns 42% less 1-D RMSE per pixel than CM, and offers a 2-fold improvement over MLE (Fig. 3(c)) – both significant increases in precision. Both versions of JD are also two orders of magnitude faster than MLE – again a significant increase (Fig. (4)).

In conclusion, we see no obstacles that might hinder the immediate adoption of JD for 'super-resolution' localization at low *S*:*N*; it allows use of spots in the noisier parts of the image that are now being discarded from data sets used to form PALM/STORM images, and will permit super-resolution imaging at the noisier depths of cells and tissues. We suggest that the signal-to-noise ratio be measured prior to localization to determine the best method to use. Then, if precision in location is the goal, MLE should be used at high ratios, and the tuned version of JD a low ratios. As the signal-to-noise ratio in any PALM/STORM image stack varies within one frame, and from frame to frame, the very highest precision can only be achieved by applying MLE and/or an appropriately-tuned version of JD to each spot depending on the immediate surroundings. Alternatively, if computation speed is paramount, we suggest CM be used because the gains realized by MLE over CM at high *S*:*N* are small, and the resulting STORM images are reasonably accurate (see Supplemental Fig. 4(b), (Media 1)). Finally, the 'optimized' version of JD provides a 'one-size-fits-all' compromise between simplicity, precision, and speed, which is more precise and faster than existing methods.

Methods

Computer and software specifications

Computations were conducted on a standard desktop PC (2.83 GHz 'Core2 Quad' CPU, Intel; 8 GB RAM; 64-bit Windows 7) using software written, compiled, and executed in MATLAB (Mathworks version 7.9.0.529; R2009b) without parallel computing. Software for implementing both 'tuned' and 'optimized' versions of JD is provided in Supplemental Material.

Image generation and analysis

To permit accurate measurements of precision, simulations were run on computer-generated images with known emitter locations. Except where specified otherwise, images contained 15x15 90-nm pixels (PSF FWHM = 250 nm, oversampled by 2.78, resulting pixel width = 90 nm). An 'emitter' was placed randomly (with sub-nanometer precision) anywhere in the central $^1/_9$th of an image (i.e., in the 5x5 central pixels in a 15x15 image). Coordinates of 'emitted photons' were then randomly generated (again with sub-nanometer precision) using a 250-nm FWHM Gaussian distribution (the commonly-accepted representation of a PSF at the resolution limit of a microscope [1, 2, 4, 12, 14]), and photons binned into pixels to produce the final image. Where background noise was added, additional photon coordinates were randomly (uniformly) distributed over the entire image to obtain the average level indicated. Ten-thousand images were generated and analyzed for each data-point shown, except for those in Supplemental Fig. 3(b) where data from 10^4 images were sorted by distance into 0.1-pixel bins. Images were passed 'blindly' to localization algorithms, and the same image sets were analyzed by all methods. Images were also generated using an algorithm that first distributes photons normally in an image space, and then corrupts the image space with Poisson noise [7]. Both algorithms yield images that appear similar to the eye and result in identical localization error (not shown).

JD localization

JD begins by attributing different numbers of photon-events to each pixel using CCD intensity, and ascribing an individual (normal) probability distribution of emitter location to each photon (Fig. 1(c)). Such a distribution is uniquely described by center location (μ_i) and width (σ_i), and the default is to place μ_i at the pixel center and use σ_i equivalent to that of the PSF; alternatively, μ_i and/or σ_i can be varied. Here, we tune μ_i by shifting photons in all pixels (other than the brightest) towards the brightest one. Thus, in one dimension:

$$\mu_i = f\left(d_i\right) = \begin{cases} \mu_c + S_o / C & d_i < 0 \\ \mu_c - S_o / C & d_i > 0 \end{cases}, \tag{2}$$

where μ_c is the location of the pixel center, d_i is the distance between the pixel and brightest pixel, S_o is pixel width, and C is an arbitrary scaling constant (in Fig. 3(a) $C = 2$, and in all other cases $C = 4$; 12 values of C between 1 and 5 were tested, and $C = 4$ yielded the highest precision under the conditions described in Fig. 3(c)). Distributions from the brightest pixel are also shifted from the central default location by a distance proportional to the intensities of adjacent pixels. In the x-dimension:

$$\mu_i = \mu_c - \left(\left(I_o - I_R\right) - \left(I_o - I_L\right)\right) I_o^{-1}, \tag{3}$$

where I_o is the intensity of the brightest pixel and I_R and I_L are the intensities of the adjacent pixels to its right and left, respectively (a similar shift is applied in the y-dimension relative to the pixel intensities above and below). We also tune σ_i as a function of distance from the brightest pixel. Thus, in one dimension, x:

$$\sigma_i = f\left(d_i\right) = \begin{cases} 1/2.5 g(x) & d_i \le \mu_{max} + 3\sigma_{psf} \\ \infty & d_i > \mu_{max} + 3\sigma_{psf} \end{cases}. \tag{4}$$

Here, σ_{PSF} is the width of the PSF in terms of sigma, μ_{max} is the center location of the brightest pixel, 2.5 is scaling factor chosen such that $\sigma_i = \sigma_{PSF}$ for the maximum pixel, and $g(x)$ is a piecewise Gaussian distribution function with a flat top:

$$g\left(x, I_i, d_i\right) = \begin{cases} \left(2\pi\sigma_{psf}{}^2\right)^{-1/2} e^{-\frac{(x-(\mu_{max}-S_o))^2}{2\sigma_{psf}{}^2}} & x < \mu_{max} - 2S_o \\ \left(2\pi\sigma_{psf}{}^2\right)^{-1/2} & \left(2\pi\sigma_{psf}{}^2\right)^{-1/2} \\ \left(2\pi\sigma_{psf}{}^2\right)^{-1/2} e^{-\frac{(x-(\mu_{max}-S_o))^2}{2\sigma_{psf}{}^2}} & x > \mu_{max} + 2S_o \end{cases}. \tag{5}$$

Together, Eqs. (4) and (5) ensure that the distributions of photons from, and adjacent to, the brightest pixel are equal to the PSF, those from surrounding pixels become progressively wider the further away the pixels are, and those from distant pixels (i.e., > 3 σ_{PSF}) become infinitely wide (see Supplemental Fig. 5 for plots of these functions). To increase computation speed in the 'optimized' version of JD, the term $1/2.5g(x)$ in Eq. (4) is replaced with σ_{PSF}, and $\mu_i = \mu_c$.

Probabilistically speaking, individual probability distributions are random variables, independent and normally distributed. To infer the location of the emitter, individual probabilities are aggregated as a joint density, which is also normally distributed [22]. Given N variables, the joint density function in one dimension is:

$$P(x) = \left(\prod_i^N \sigma_i\right)^{-1} e^{-\frac{1}{2}\sum_i^N \left(\frac{x-\mu_i}{\sigma_i}\right)^2}. \tag{6}$$

Ignoring constant terms for simplicity (because they only affect the amplitude of the function, which is not of immediate interest here), we get:

$$P(x) \propto e^{-\frac{1}{2}\sum_i^N \left(\frac{x^2}{\sigma_i^2} - 2\frac{\mu_i x}{\sigma_i^2} + \frac{\mu_i^2}{\sigma_i^2}\right)} \propto e^{-\frac{1}{2}\left[\sum_i^N \left(\frac{1}{\sigma_i^2}\right)x^2 - 2\sum_i^N \left(\frac{\mu_i}{\sigma_i^2}\right)x\right]}. \tag{7}$$

Next we define:

$$k_1 = \sum_i^N \sigma_i^{-2},\ k_2 = 2\sum_i^N \mu_i \sigma_i^{-2}, \tag{8}$$

so that

$$P(x) \propto e^{\frac{1}{2}\left(-k_1 x^2 + k_2 x\right)}. \tag{9}$$

Factoring and, once again, ignoring constant terms:

$$P(x) \propto e^{-\frac{1}{2}k_1\left(x - \frac{k_2}{2k_1}\right)^2}. \tag{10}$$

After replacing k_1 and k_2, the joint distribution takes the form

$$P(x) \propto e^{-\frac{1}{2}\sum_i^N \left(\frac{1}{\sigma_i^2}\right)\left[x - \sum_i^N \left(\frac{\mu_i}{\sigma_i^2}\right) / \sum_i^N \left(\frac{1}{\sigma_i^2}\right)\right]^2}, \tag{11}$$

which resembles a simple Gaussian distribution:

$$G\left(x \mid \mu_o\right) = e^{-\frac{1}{2}\frac{1}{\sigma_i^2}(x-\mu_o)^2}, \tag{12}$$

whose center is equal to the center of the joint distribution:

$$\mu_o = \sum_i^N \left(\mu_i \sigma_i^{-2}\right) \cdot \left(\sum_i^N \sigma_i^{-2}\right)^{-1}. \tag{13}$$

(13; same as Eq. (1))

Projection and inference are repeated for each orthogonal axis. The width of the joint distribution does not provide a reliable estimate of localization precision, presumably because it does not account for effects of background noise.

To get to the CM equation from Eq. (13), first we must set σ_i to a constant for all photons, that is

$$\text{if } \sigma_i = \sigma, \text{then } \mu_o = \frac{1}{\sigma^2}\sum_i^N \mu_i / \left(N\sigma^{-2}\right) = \sum_i^N \left(\mu_i\right) N^{-1}. \tag{14}$$

The total number of photons in the image, N, can be rewritten for an m x n matrix as the sum of pixel intensities, I,

$$N = \sum_i^n \sum_j^m I_{ij} . \tag{15}$$

We then set μ_i equal to pixel center positions, x_i:

$$\text{if } \mu_i = x_i \text{ then } \mu_o = \sum_i^N x_i \cdot \left(\sum_i^n \sum_j^m I_{ij} \right)^{-1} . \tag{16}$$

Finally, the sum of all photons is equal to the sum of pixel intensities. This yields the estimated emitter location in one dimension, C_x:

$$C_x = \sum_i^n \sum_j^m x_i I_{ij} \cdot \left(\sum_i^n \sum_j^m I_{ij} \right)^{-1} , \tag{17}$$

which is equal to Eq. (3) in Cheezum *et al.* [10]. Another, more general, form of this equation would be required to incorporate weighting values equivalent to those implemented in the 'tuned' version of JD localization.

Background correction

MLE and MLS inherently correct for background, as background level is an intrinsic fitting parameter. In the presence of increasing background, emitter location estimated by CM and JD progressively diverge from the true location towards the geometric center of the image; therefore, high precision can only be achieved using these methods if background correction is included. For CM, a standard background correction [10] is used prior to localization: a noise threshold is defined (as the mean intensity plus two standard deviations in the two peripheral pixels around the circumference, which includes the 104 peripheral pixels in a 15x15 image) and subtracted from the intensity of every pixel in the image. Another background-correction algorithm tested (i.e., setting all pixels with intensity less than the threshold to zero and leaving the remaining pixels unaltered) did not perform as well (not shown). For the default version of JD, we first consider those pixels at or below the threshold (estimated as for CM); σ_i of their distributions is set to infinity, reducing amplitude to zero and negating any effect on localization. Then we consider pixels with intensity above the threshold; σ_i is set to infinity for the proportion of distributions corresponding to the fraction of intensity below the threshold. For the tuned and optimized versions of JD, background is removed similarly (note that distributions coming from 'non-spot' pixels with intensities between the noise ceiling and the brightest also have σ_i set to infinity in Eqs. (4) and (5).

Localization by CM, MLS, & MLE

CM was computed with an in-house program (described by Cheezum *et al.* [10]) as the mean of the locations of all pixel centers in the window weighted by their respective intensities. MLS fitting to a 2-D Gaussian intensity profile was also computed with an in-house program. Peak amplitude, background level, x- and y-width, plus x- and y-location were set as fitting parameters. Regression continued until changes fluctuated < 0.01% or until 200 iterations elapsed; when a solution was found, in most cases it was found within 10 iterations. MLE of a 2-D Gaussian intensity profile was implemented directly, as provided by others [5]. Neither fitting algorithm yielded a smooth line at low S:N in Fig. 3 even though 10^4 measurements were made for each data point; therefore, plots were smoothed by linear regression.

Precision measurements

Post-localization, estimates were compared with true locations of emitters and root-mean-squared errors (RMSE) computed. Signal-to-noise ratio is computed in different ways throughout the literature. Here,

$$S:N = (I_o - b)/N_b\,, \tag{18}$$

where I_o is the maximum pixel intensity, b is the background – the mean intensity of the two concentric sets of peripheral pixels in the image (i.e., the 104 peripheral pixels in a 15x15 image) – and N_b is the RMS intensity of the same peripheral pixels. This computation is as in Cheezum *et al.* [10], with two differences: (i) signal was measured as the maximum pixel intensity (instead of mean spot intensity) because images were generally so noisy, and (ii) noise was sampled from peripheral pixels (not across the whole image) to assess better the degree to which signal stands above fluctuations in background.

Speed test

All computations to assess speed were conducted serially on the same set of 1,000 images in this order: ‘optimized’ version of JD, ‘tuned’ version of JD, CM, MLS, and MLE. To test the possibility that residual computer memory loss retarded sequential computations, computations were repeated in reverse order and yielded identical results. The derivatives for MLS were computed by hand and implemented as linear equations to avoid built-in MATLAB functions known to be slow. The numbers of localizations/sec from our routine were compared with those reported by Smith *et al*. [7] (obtained using least-squares fitting on a single processor), and are similar (not shown). The mean computation rate of three independent trials is reported (in Fig. 4, standard deviations were < 1% in all cases).

RNA FISH images

Nuclear transcription sites containing nascent (intronic) RNA were detected using RNA FISH. Monkey kidney cells (cos-7) were transiently transfected with a plasmid encoding an *EGFP* gene (as in Xu and Cook [23]) with an intron containing sequences derived from intron 1 of human *SAMD4A*. One day after transfection, cells were seeded on to a coverslip etched with 0.1% hydrofluoric acid, and re-grown; 40 h post-transfection, cells were transferred to ‘CSK buffer’ for 10 min, and fixed (4% paraformaldehyde; 20 min; 20°C) [24]. Nascent (intronic) *SAMD4A* RNA was then detected by RNA-FISH using 50-nucleotide probes each tagged with ~5 Alexa 647 fluors (as in Papantonis *et al.* [25]). After hybridization, cells were mounted in Vectashield (Vector Laboratories) containing 1 μg/ml DAPI (4,6-diamidino-2-phenylindole; Sigma), and imaged using a Zeiss Axiovert microscope (63x/1.43 numerical aperture objective) equipped with a CCD camera (CoolSNAP$_{HQ}$, Photometrics). Sub-diffraction spots marking nuclear transcription sites with a S:N < 3 were selected manually for analysis.

dSTORM Images

Direct STORM (dSTORM) images were kindly provided by S. Van De Linde [8]. Microtubules in fixed cos-7 cells were indirectly immuno-labeled with Alexa 647, and 30,000 images (excitation at 641 nm under inclined illumination, emission recorded between 665 and 735 nm) of spatially-separated sub-diffraction sized spots in one field collected (image acquisition rate 885 s^{-1}) using an EM-CCD camera (Andor; EM-gain = 200; pre-amp-gain = 1). Spots were identified by 2-D cross-correlation with a randomly-generated 2-D Gaussian intensity pattern, and candidates for fitting selected by a minimum cross-correlation value. 154,040 windows (11x11 pixels) containing 1 spot were selected, and independently corrupted with noise until S:N measured < 3; then each window was passed to each of the four localization algorithms (Supplemental Fig. 4(a), (Media 1)). Localization results were rounded to the nearest nanometer, and used to reconstruct an image of the whole field using 1-nm pixels. To aid visualization, each of the resulting images was convolved with a 2-D Gaussian intensity profile with a 20 nm FWHM. Contrast and brightness of all images displayed are equal between methods.

Acknowledgments

We thank N. Publicover, J. Sutko, A. Papantonis, S. Baboo, B. Deng, J. Bartlett, S. Wolter, and M. Heilemann for discussions, and S. Van De Linde for dSTORM data. This work was supported by the Wellcome Trust (grant 086017).

Supplemental Material

Supplemental Movie 1. Nascent RNA at transcription sites in monkey nuclei, imaged using a wide-field microscope and CCD (90-nm pixels) as in Fig. 5(b). Each frame in the movie shows one of 100 windows (15x15 pixels) with a S:N < 3 (value indicated in the lower-left corner), and the 2-D localizations obtained by the different methods (tuned version of JD – *red circle*; CM – *blue dot*; MLS – *orange dot*; MLE – *green star*). Each window is shown at low magnification in the lower-right corner to give an impression of its appearance at a typical scale. The tuned version of JD performs at least as well as the others.

Supplemental Figure 1. The failure of MLS at low photon counts.
Using computer-generated images (15x15 pixels; 10-250 photons; $b = 0$; inset shows a typical image generated using 63 photons to give a S:$N = \infty$), the 1-D RMSE was calculated (10^3 localizations per data point) using MLS, and the fraction of times that the algorithm converged to a solution during the 200 iterations allowed determined. As the number of photons declines, MLS increasingly fails (as it becomes more susceptible to shot-noise that causes images to diverge from a Gaussian pattern). The onset of rapid decline coincides with the rapid increase in RMSE seen in Fig. 2.

Supplemental Figure 2. Tuning JD reduces localization error at low signal-to-noise ratios, but increases error at high ratios.
Using computer-generated images (15x15 pixels; 10-10,000 photons; $b = 10$), the 1-D RMSE (in nm or pixel units) was calculated (10^4 localizations per data point) using JD tuned by varying either μ_i alone, σ_i alone, or both together (as in Figs. 3(a), 3(b), and 3(c)). Errors given by CM are included for comparison, as they are identical to those given by the default version of JD. *Left*: At high S:N, tuning σ_i alone yields increased error, while tuning μ_i alone does not. *Right*: At high S:N, tuning both μ_i and σ_i yields high error that converges asymptotically to ~0.02 pixel.

Supplemental Figure 3. The effects of window size and the distance of a spot from the center of the window.
Using computer-generated images (183 photons; $b = 10$; S:$N = 2$), the 1-D RMSE (in nm or pixel units) was calculated using the tuned version of JD, CM, MLE, and MLS. (**a**) Varying window size. When the window chosen for analysis decreases from 15x15 to 7x7 pixels, errors given by the tuned version of JD are essentially invariant (i.e., the method performs robustly), but those given by CM and MLE fall. As errors given by MLE are known to increase as window size falls in the absence of background noise and at higher a higher S:N [7], the fall seen here with both MLE and MLS is probably due to an increased likelihood of finding the true location purely by chance. Lines are data points fit to exponential functions. (**b**) Varying the distance of a spot from the center of the window (*grey lines* indicate error at farthest point). As the true location relative to the center of a window varies, the variance given by JD is negligible, but MLE and CM are most accurate when the spot is at the image center (a known behavior of CM in noisy images [10]). Curiously, MLS seems most accurate at this low S:N when the spot is away from the image center.

Supplemental Figure 4. dSTORM images of microtubules in monkey cells (cos-7) processed using the different approaches.

The 30,000 images ('frames') of diffraction-limited spots of one field (collected using inclined illumination and a CCD) used for Fig. 5(b) were analyzed (some images and histograms from that figure are included here for comparison). (**a**) Window selection, and corruption with noise. (i) Mean projection of all spots; this projection yields an image analogous to that obtained using a wide-field microscope. (ii) A typical frame, illustrating individual diffraction-limited spots, from which windows that contained 1 centrally-located and isolated spot were selected (using a Gaussian spot-finding algorithm) for localization. (iii) Four isolated spots (11x11 pixel windows) selected by the algorithm from frame 30 (the *histogram* illustrates the 154,040 windows with different *S*:*N* ratios; a *S*:*N* > 9 is typical of that found in current STORM data). (iv) Windows in frame 30 after corruption with noise to reduce *S*:*N* to < 3 (*histogram*). (v) Mean projection of all corrupted windows; analogous to a 'noisy' wide-field image. (**b**) Reconstructed images prepared by passing all selected windows prior to noise corruption to the different algorithms (i.e., the tuned version of JD, plus CM, MLS, and MLE), compiling localization results, and then convolving localizations with a 20-nm Gaussian intensity profile to aid visualization (higher magnifications of the yellow box are shown below). The different approaches yield roughly equivalent images. At this high *S*:*N*, the MLE image (green boxes) should contain localizations with the least RMSE (Fig. 2), so we consider it to provide the truest representation of microtubule structure. (**c**) Intensity profile (the average intensity in arbitrary units, au, of a region 35 nm orthogonal to the line indicated) across lines *1* and *2* in (b). Generally, results given by JD and MLE are similar, as those returned by CM (*profile 1*) and MLS (*profile 2*) slightly diverge. (**d**) Reconstructed images prepared as in (b), but using the windows after corruption with noise. 142,728, 142,684, 142,275, and 142,669 localizations were returned by the tuned version of JD, CM, MLS, and MLE respectively. At this low *S*:*N* ratio, images prepared using the tuned version of JD (red boxes) return the least error (Fig. 3(c)), so we now consider them to be the truest representations. CM gives the 'fuzziest' image, presumably because it performs the worst at *S*:*N* >2. At the highest magnification (bottom row) – where the tuned version of JD, CM, MLS, and MLE yield 50.5, 50.9, 50.7, and 50.6 localizations per unit area, respectively – CM, MLS, and MLE probably return many mis-localizations (compare the number of isolated spots in the dark area at bottom-center in the bottom row). In the case of MLS and MLE, a failure to converge to a solution during the 200 iterations allowed is returned as a 'zero-result' in the upper-left corner of a window, and this generates a grid pattern after reconstruction that is especially obvious in the MLS image (*orange circles*). Such localizations are usually discarded, but have been retained here to allow fair comparison between a similar numbers of localizations from the different methods. (**e**) Intensity profiles across lines *1* and *2*. Profiles are slightly wider here, compared to those in (c). (**f**) Comparison of intensity profiles of line *2*. We expect MLE applied to uncorrupted windows to yield the truest representation (above); when using corrupted windows, the tuned version of JD returns a profile that is closer to the 'true' one (compared to MLS). (**g**) The fraction of nearest-neighbor distances (3-nm bins) given by all localizations obtained with the corrupted windows shown in the middle row of (d). Data using MLE with uncorrupted windows (which we assume gives the 'truest' representation; above), and for the same number of randomly-distributed spots, are included for comparison. Again, the tuned version of JD returns a distribution closest to the 'true' one. Results using MLS have been omitted, because many 'failed' localizations heavily skew the histogram.

Supplemental Figure 5. Shapes of functions used in equations (3) and (4).

Top: in equation (3), σ_i is tuned as a function of distance from the brightest pixel, d_i; the brightest pixel is at 0, and σ_i increases rapidly at ±3 to equal ∞ at ±4. *Bottom*: in equation (4), $g(x)$ is a piecewise Gaussian distribution function with a flat top (width 5 pixels) and Gaussian slopes that match the PSF. Applications of these two equations ensures that the distributions of photons from the brightest pixels are equal to the PSF, those from surrounding pixels become progressively wider the further away the pixels are, and those from distant pixels (i.e., > 3 σ_{PSF}) become infinitely wide.

JD Localization Software.

JDLoc_Test.m
This script demonstrates the use of JD_2D_tuned and JD_2D_optimized.

genSpot.m
A subroutine to generate an array of spot images to be used for localization. (Requires MATLAB Statistics Toolbox.)

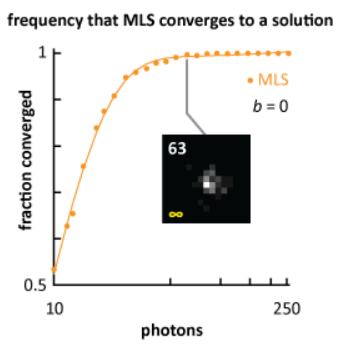
JD_2D_tuned.m
Single molecule localization using the Joint Distribution method tuned for maximum precision at low signal-to-noise ratios.

JD_2D_optimized.m
Single molecule localization using the Joint Distribution method optimized for speed and precision across a wide range of signal-to-noise ratios.

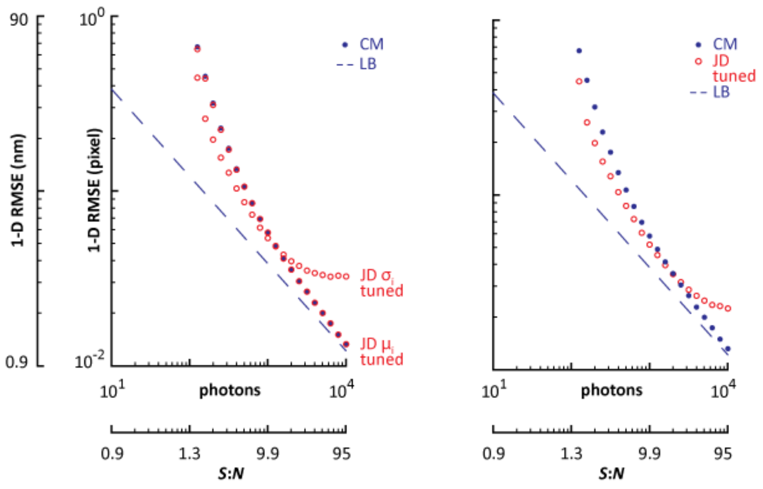
Instructions
Copy JDLoc_Test.m, genSpot.m, JD_2D_tuned.m, and JD_2D_optimized.m to your MATLAB directory. For a demonstration, run JDLoc_Test in MATLAB (MATLAB Statistics Toolbox is required). For more details type "help JDLoc_Test" in MATLAB. To localize your own image, type "help JD_2D_tuned" or "help JD_2D_optimized" in MATLAB and follow the instructions.

To incorporate Joint Distribution (JD) localization into your own script, all pertinent code is contained within JD_2D_tuned.m or JD_2D_optimized.m.

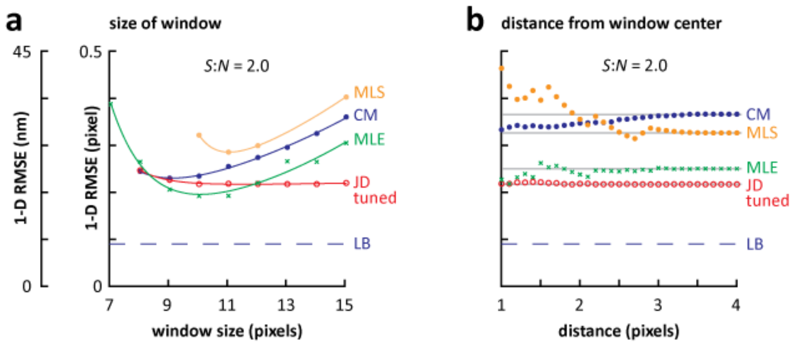
Supplemental Figure 1



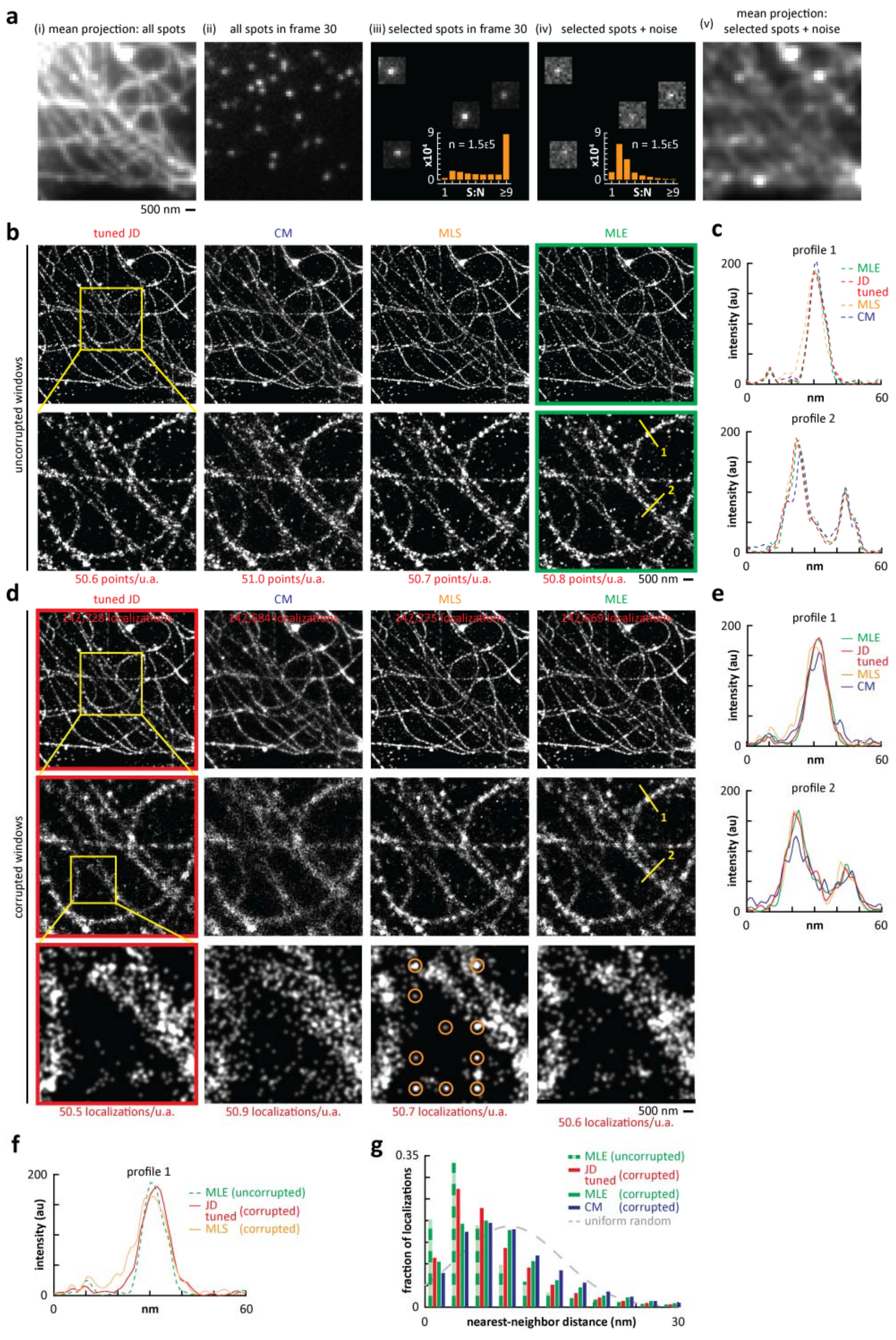
Supplemental Figure 2



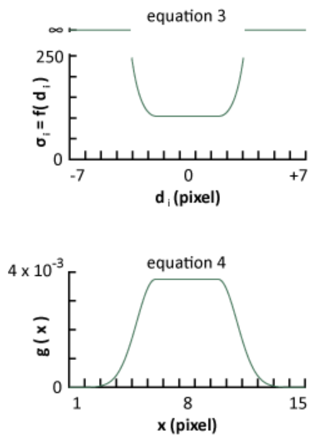
Supplemental Figure 3



Supplemental Figure 4



Supplemental Figure 5



```
%JDLoc_Test.m
%   This script demonstrates the use of JD_2D_tuned and JD_2D_optimized.
%
%   [] = JDLoc_Test ()
%
%   -----------------------------------------------------------------------
%   This code is free for academic use only. Please reference:
%       "A maximum precision closed-form solution for localizing
%       diffraction-limited spots in noisy images" Joshua D. Larkin and
%       Peter R. Cook, Optics Express (2012).
%
%   For commercial use please contact Joshua Larkin
%       email: joshlarkin at gmail.com
%
%   copyright Joshua D Larkin 2012
%   -----------------------------------------------------------------------
%
%   This code builds a stack of 2D images using subroutine genSpot, each of
%   which contains a single diffraction-limited spot. Spots in each image
%   are then localized serially using JD_2D_tuned and JD_2D_optimized. The
%   number of localizations per second is computed and reported. Derived
%   locations are compared to true locations, and the root-mean-square
%   error reported.
%
%   Note:
%       MATLAB Statistics Toolbox is required for image generation script
%
%   INPUTS:
%       None
%
%   OUTPUTS:
%       None

function JDLoc_Test



%--Generate spot images--

    %Variables
    Nspots = uint32(1000);    %number of spot images to generate in stack
    boxSize = uint8(15);      %dimension of each 2D image in pixels
    pixelSize = double(90);   %linear dimension of a square pixel in nm
    FWHMpsf = double(250);    %FWHM of microscope PSF in nm
    count = uint32(183);     %photon-event count per image
    noise = double(10);        %background noise in average photons/pixel

    [spotStack, spotLocs] = genSpot(Nspots, boxSize, pixelSize, FWHMpsf,...
                                    count, noise);
    disp(' ')
    disp('--')
    disp([int2str(Nspots) ' images generated, begining localization'])
    disp(' ')

    %Change psf FWHM to Sigma
    psf = FWHMpsf / 2.35;




%--Localize spots--
```

```
%   Tuned version of JD Localization
    mu1 = zeros(Nspots, 2); %center locations
    tic %timer start
    for i=1:Nspots
        mu1(i,:) = JD_2D_tuned(spotStack(:,:,i), pixelSize, psf);
    end
    t1=toc; %timer stop


%   Optimized version of JD Localization
    mu2 = zeros(Nspots, 2); %center locations
    tic %timer start
    for i=1:Nspots
        mu2(i,:) = JD_2D_optimized(spotStack(:,:,i), pixelSize, psf);
    end
    t2=toc; %timer stop




%--Compute Error--

    %root-mean-square error, components of each Cartesian coordinate
    rmse1Comp = sqrt(sum( (mu1 - double(spotLocs') ).^2)/double(Nspots));
    rmse2Comp = sqrt(sum( (mu2 - double(spotLocs') ).^2)/double(Nspots));
    %linear root-mean-square error, normalized to pixel size
    rmse1 = sqrt(rmse1Comp(1)^2 + rmse1Comp(2)^2) / pixelSize;
    rmse2 = sqrt(rmse2Comp(1)^2 + rmse2Comp(2)^2) / pixelSize;




%--Display Results--
    disp('Tuned JD:')
    disp(['  measured error (rmse) = ', num2str(rmse1,'%11.3g') ' pixels'])
    disp(['  computation rate = ' num2str(Nspots/t1) ' localizations/sec'])
    disp(' ')
    disp('Optimized JD:')
    disp(['  measured error (rmse) = ', num2str(rmse2,'%11.3g') ' pixels'])
    disp(['  computation rate = ' num2str(Nspots/t2) ' localizations/sec'])
    disp(' ')

end
```

```
%genSpot.m
%   A subroutine to generate an array of spot images to be used for
%   localization.
%
%   [spotStack, spotLocs] = genSpot (Nspots, boxSize, pixelSize, FWHMpsf,
%                                   count, noise)
%
%   ----------------------------------------------------------------------
%   This code is free for academic use only. Please reference:
%       "A maximum precision closed-form solution for localizing
%       diffraction-limited spots in noisy images" Joshua D. Larkin and
%       Peter R. Cook, Optics Express (2012).
%
%   For commercial use please contact Joshua Larkin
%       email: joshlarkin at gmail.com
%
%   copyright Joshua D Larkin 2012
%   ----------------------------------------------------------------------
%
%   This code builds a stack of 2D images, each of which contains a single
%   diffraction pattern. This is done by normally distributing 'photon'
%   coordinates in two dimensions about a randomly selected center, adding
%   uniformly distributed background noise, and binning photons into pixels
%   of specified size. The true center location of each spot is also
%   returned.
%
%   Note:
%       MATLAB Statistics Toolbox is required
%
%   INPUTS:
%       Nspots:     number of spot images to generate in stack (unit32)
%       boxSize:    dimension of each 2D image in pixels (uint8)
%       pixelSize:  linear dimension of a square pixel in nm (double)
%       FWHMpsf:    FWHM of microscope PSF in nm (double)
%       count:      photon-event count per image (uint32)
%       noise:      background noise in average photons/pixel (double)
%
%   OUTPUTS:
%       spotStack:  3-dimensional stack of computer generated spot images
%       spotLocs:   2xN array of true spot center locations in nm

function [spotStack, spotLocs] = genSpot (Nspots, boxSize, pixelSize,...
                                          FWHMpsf, count, noise)

%Reformat inputs
    %Change psf FWHM to Sigma
    psf = FWHMpsf / 2.35;

    %Image dimension in nm
    imDim = pixelSize * uint16(boxSize);

%Initialize Variables
    %Stack of images for output
    spotStack = zeros(boxSize, boxSize, Nspots, 'uint16');

    %array of true spot center locations [x,y]
    spotLocs = zeros(2, Nspots, 'uint16');

%Build Image Stack
```

```
for i=1:Nspots

    %--Generate photon observations--

        % randomize center location
        locMin = uint16(floor(imDim*0.33));
        locMax = uint16(ceil(imDim*0.67));
        xLoc = (locMin-1) + unidrnd(locMax-(locMin-1));
        yLoc = (locMin-1) + unidrnd(locMax-(locMin-1));

        % normal distribution of photon-events about center
        xCoords = normrnd(double(xLoc), psf, [1 count]);
        yCoords = normrnd(double(yLoc), psf, [1 count]);
            %note: it's possible for photons to be outside of image frame


    %--Generate background noise--

        % uniform distribution of background noise
        xNoise = unidrnd(double(imDim),...
                                    [1 uint16(double(boxSize)^2 * noise)]);
        yNoise = unidrnd(double(imDim),...
                                    [1 uint16(double(boxSize)^2 * noise)]);

        % add noise photon-events to spot photon-events
        d = [cat(2,xCoords,xNoise); cat(2,yCoords,yNoise)];

    %--Form histogram (pixels)--

        % create pixels by defining edges
        edges = cell(2,1);
        edges{1} = double(1:pixelSize:imDim);
        edges{2} = double(1:pixelSize:imDim);

        % populate pixels with photon-events
        n = coords2im(d, edges);

    %--Organize output--

        spotStack(:,:,i) = n;
        spotLocs(:,i) = [xLoc yLoc];

end

end

function [im] = coords2im(photonCoords, edges)
%subroutine to populate an image space, given photon coordinates
%
%Inputs:
%   photonCoords = [x vector; y vector] coordinates of photons in nm
%
%   edges = cell array of vectors containing the edge locations of pixels
%   in nm, where edges{1} = left edges of pixels in x-dimension, edges{2} =
%   top edges of pixels in y-dimension.
%
%Outputs:
%   im = output image
```

```
%

%Reformat inputs
    X = photonCoords(1,:);
    Y = photonCoords(2,:);

    Xbins = edges{1};
    Ybins = edges{2};

    XpixelSize = Xbins(2) - Xbins(1);
    YpixelSize = Ybins(2) - Ybins(1);

%Initialize variables
    im = zeros(size(Xbins,2), size(Ybins,2), 'uint16');

%Populate image with photon-events
    for i=1:size(photonCoords,2)

        %compute image coordinates from photon coordinates
        imX = ceil( X(i) / XpixelSize);
        imY = ceil( Y(i) / YpixelSize);

        %ensure photon is within image frame
        if ~any([imX imY] > size(im) | [imX imY] < [1 1])

            %place each photon in a pixel
            im(imY, imX) = im(imY, imX) + 1;

        end

    end

    %imshow(im,[]);

end
```

```
%JD_2D_tuned.m
%   Single molecule localization using the Joint Distribution method tuned
%   for maximum precision at low signal-to-noise ratios.
%
%   [mu0] = JD_2D_tuned(spotIm, pixelSize, psfSig)
%
%   -----------------------------------------------------------------------
%   This code is free for academic use only. Please reference:
%       "A maximum precision closed-form solution for localizing
%       diffraction-limited spots in noisy images" Joshua D. Larkin and
%       Peter R. Cook, Optics Express (2012).
%
%   For commercial use please contact Joshua Larkin
%       email: joshlarkin at gmail.com
%
%   copyright Joshua D Larkin 2012
%   -----------------------------------------------------------------------
%
%   This code performs the joint distribution localization of a
%   sub-diffraction sized particle imaged as a diffraction-limited spot
%   using a CCD array. If necessary, the image should be corrected for CCD
%   gain and offset such that unit pixel intensity represents a single
%   photo-electric event. The input is a single image of a single
%   diffraction-limited spot (isolated from a larger image of multiple
%   spots if necessary), assumed to contain a single fluorophore.
%   This 'tuned' version of JD localization has been optimized for spot
%   images with signal-to-noise ratios below 3, as defined in the
%   accompanying publication. Briefly, photons are shifted from pixel
%   centers by 1/4 of a pixel width, photon distributions from the
%   brightest pixel are set equal to the point-spread function while those
%   further away are progressively broadened. Photons too far from the
%   maximum intensity pixel to likely have come from the emitter of
%   interest are negated. Background noise in the image is estimated as the
%   mean intensity of peripheral pixels plus two standard deviations. The
%   influence on localization of photons thought to have come from
%   background is also negated.
%
%   INPUTS:
%       spotIm:      single spot image to localize (unit16)
%                    typical dimensions: 7x7 to 15x15 pixels (square only)
%       pixelSize:   linear dimension of square pixel in nm (double)
%       psfSigma:    sigma of a Gaussian function representing the
%                    microscope PSF in nm (double)
%
%   OUTPUTS:
%       mu:          [x,y] spot localization in nm, relative to upper-left
%                    corner of input image

function [mu0] = JD_2D_tuned(spotIm, pixelSize, psfSig)



%--Check Input Parameters--
    if size(spotIm,2) ~= size(spotIm,1)
        error('spot image must be square')
    end
    if ~isa(spotIm, 'uint16')
        spotIm = uint16(spotIm);
        warning('JDL:spotImClass',...
            'spotIm must be of class uint16; it has been converted')
    end
    if ~isa(pixelSize, 'double')
        pixelSize = double(pixelSize);
    end
    if ~isa(psfSig, 'double')
        psfSig = double(psfSigma);
    end



%--Determine Noise Level--
%   Sample outer two concentric rings of pixels (assuming these are
%   representative of uniform background noise in the image). Define
%   background as the mean value of these pixels plus two standard
%   deviations.

%   Interrogate pixel values at image perimeter, 2 pixels deep
        sizeX = size(spotIm,2);
        sizeY = size(spotIm,1);
        bottom = spotIm(1:2, 1:sizeX);
        top = spotIm(sizeY-1:sizeY, 1:sizeX);
        left = spotIm(3:sizeY-2, 1:2);
        right = spotIm(3:sizeY-2, sizeX-1:sizeX);

%   Define perimeter pixels as background
        bkgdValues = double(cat(2, bottom, top, left', right'));
        bkgdVector = bkgdValues(:);
        N = length(bkgdVector);

%   Compute background level as mean + 2 standard deviations
        meanBkgd = sum(bkgdVector) / N;
        stdBkgd = sqrt( sum( (bkgdVector-meanBkgd).^2 ) / (N-1) );
        bkgdLevel = ceil( meanBkgd + 2*stdBkgd );

%   Identify pixels with background noise
        ImBkgd0 = double(uint16(spotIm-bkgdLevel));



%--Shift photons from pixel centers--
%   Identify the brightest pixel and shift mu_i for all photons from other
%   pixels towards it by 1/4 of a pixel width in each dimension. For
%   photons from the brightest pixel, shift mu_i from the pixel center by a
%   value proportional to the intensities of adjacent pixels.

%   Determine pixel center values in nm (1 = x, 2 = y)
        centers(1,:) = pixelSize/2 : pixelSize : pixelSize*size(spotIm,2);
        centers(2,:) = pixelSize/2 : pixelSize : pixelSize*size(spotIm,1);

%   Determine location of max intensity pixel
        [~, Ix] = max(sum(ImBkgd0,1));
        [~, Iy] = max(sum(ImBkgd0,2));
        CM(1) = centers(1,Ix);
        CM(2) = centers(2,Iy);

%   Define shifted positions
        weightedCenters = zeros(size(centers));
        for i=1:2 %(1 = x, 2 = y)
```

```
            proj = sum(ImBkgd0,i); %project image onto one axis
            [~, maxI] = max(proj); %identify max intensity pixel
            j=1:length(centers); %independent variable along each axis

            %pixels left/above max pixel
            weightedCenters(i,:) = (centers(i,:) + pixelSize/4);

            %pixels right/below max pixel
            weightedCenters(i,j>maxI) = centers(i,j>maxI) - pixelSize/4;

            %max pixel
            if maxI>1 && maxI<length(centers) %ensure not at perimeter
                %scale center in max pixel by neighboring intensities
                leftInfluence = (proj(maxI) - proj(maxI-1)) / proj(maxI);
                scaledLeftInfl = (leftInfluence*pixelSize - pixelSize)/2;
                rightInfluence = (proj(maxI) - proj(maxI+1)) / proj(maxI);
                scaledRightInfl = (rightInfluence*pixelSize - pixelSize)/2;
                weightedCenters(i,maxI) = uint16(centers(i,maxI)+...
                    scaledLeftInfl-scaledRightInfl);
            else
                weightedCenters(i,maxI) = centers(i,maxI);
            end

        end



%--Assign Variable Distribution Widths-- (slow version)
%   Makes photon distributions narrower at max pixel and progressively
%   wider the further a pixels is from the max pixel. Because a finite (and
%   small for small images) number of possible solutions exist, to speed
%   things up all possible distribution/sigma functions can be defined once,
%   outside of this function, and passed to this function to be looked-up.

%   Initialize variables
        f = zeros(pixelSize*size(spotIm,2),2); %distribution function
        sf = zeros(pixelSize*size(spotIm,2),2); %sigma function
        So = 2.0 * pixelSize; %width of flat-top in distribution function
        for i=1:2 %(1 = x, 2 = y)

%   Create a piecewise Gaussian distribution function
            for x = 1:pixelSize*size(spotIm,2)
                if x < CM(i)-So
                    f(x,i) = 1/sqrt(2*pi*psfSig^2)*...
                                   exp(-(x-(CM(i)-So))^2/(2*psfSig^2));
                elseif x > CM(i)+So
                    f(x,i) = 1/sqrt(2*pi*psfSig^2)*...
                                   exp(-(x-(CM(i)+So))^2/(2*psfSig^2));
                else
                    f(x,i) = 1/sqrt(2*pi*psfSig^2);
                end
            end

%   Use distribution function to create sigma function
            for x = 1:pixelSize*size(spotIm,2)
                if x < CM(i)-3*psfSig,
                    sf(x,i)=Inf;
                elseif x > CM(i)+3*psfSig,
                    sf(x,i)=Inf;
                else
                    sf(x,i)=1/(2.5*f(x,i));
                end
            end

        end



%--Localize by Joint Distribution--
%   Involves projecting the image onto one dimension, forming a list of
%   photon coordinates, and assigning distributions for each photon-event;
%   then repeating for the second dimension and computing the center spot
%   location by as the joint of all independent distributions.

%   Initialize variables
        pe=cell(2,1); %photon-event list
        s=cell(2,1); %photon-event sigma list
        for i=1:2 %(1 = x, 2 = y)

%   Project image onto single axis
            projection = sum(spotIm,i); %(1 = x, 2 = y)
            bkgdProj = sum(ImBkgd0,i); %background pixels
            if i==2
                %rotate y-axis
                projection = projection';
                bkgdProj = bkgdProj';
            end

%   Build photon-event list
            pe{i} = zeros(1,sum(projection(:))); %photon-event list
            s{i} = Inf.*ones(1,sum(projection(:))); %set all sigma_i to Inf
            idx=1; %index of coordinates array

%   Convert pixel values into photon coordinates
            for j=1:size(projection,2)

%   Record mu_i
                nPhotons = projection(j); %number of photons to record
                if nPhotons>0
                    pe{i}(idx:idx+nPhotons-1) = weightedCenters(i,j)...
                                                    * ones(1,nPhotons);

%   Record sigma_i
                    if bkgdProj(j)>0
                        s{i}(idx:idx+bkgdProj(j)-1) = ...
                                        sf(uint16(weightedCenters(i,j)),i);
                    end

                end
                idx = idx + nPhotons; %increment index of coordinates array

            end

        end

%   Compute joint distribution
        photonEvents(1,:) = pe{1};
        photonEvents(2,:) = pe{2};
        photonDists(1,:) = s{1};
        photonDists(2,:) = s{2};
        sSquared = photonDists.^2;
        sSumInvSqr = sum(1./sSquared, 2);
        mu0 = sum(photonEvents./sSquared, 2) ./ sSumInvSqr;

end
```

```matlab
%JD_2D_optimized.m
%   Single molecule localization using the Joint Distribution method
%   optimized for speed and precision across a wide range of signal-to-
%   noise ratios.
%
%   [mu0] = JD_2D_optimized(spotIm, pixelSize, psfSig)
%
%   ---------------------------------------------------------------------
%   This code is free for academic use only. Please reference:
%       "A maximum precision closed-form solution for localizing
%       diffraction-limited spots in noisy images" Joshua D. Larkin and
%       Peter R. Cook, Optics Express (2012).
%
%   For commercial use please contact Joshua Larkin
%       email: joshlarkin at gmail.com
%
%   copyright Joshua D Larkin 2012
%   ---------------------------------------------------------------------
%
%   This code performs the joint distribution localization of a
%   sub-diffraction sized particle imaged as a diffraction-limited spot
%   using a CCD array. If necessary, the image should be corrected for CCD
%   gain and offset such that unit pixel intensity represents a single
%   photo-electric event. The input is a single image of a single
%   diffraction-limited spot (isolated from a larger image of multiple
%   spots if necessary), assumed to contain a single fluorophore. This
%   'optimized' version of JD localization is precise across a wide range
%   of signal-to-noise ratios and is faster than the 'tuned' version.
%   Briefly, photon distributions are centered at pixel centers, and photon
%   distributions from the brightest pixels are set equal to the
%   point-spread function while those further away than 3*sigma_psf are
%   negated. Background noise in the image is estimated as the mean
%   intensity of peripheral pixels plus two standard deviations. The
%   influence on localization of photons thought to have come from
%   background is also negated.
%
%   INPUTS:
%       spotIm:     single spot image to localize (unit16)
%                   typical dimensions: 7x7 to 15x15 pixels (square only)
%       pixelSize:  linear dimension of square pixel in nm (double)
%       psfSig:     sigma of a Gaussian function representing the
%                   microscope PSF in nm (double)
%
%   OUTPUTS:
%       mu:         [x,y] spot localization in nm, relative to upper-left
%                   corner of input image

function [mu0] = JD_2D_optimized(spotIm, pixelSize, psfSig)



%--Check Input Parameters--
    if size(spotIm,2) ~= size(spotIm,1)
        error('spot image must be square')
    end
    if ~isa(spotIm, 'uint16')
        spotIm = uint16(spotIm);
        warning('JDL:spotImClass',...
            'spotIm must be of class uint16; it has been converted')
    end
    if ~isa(pixelSize, 'double')
        pixelSize = double(pixelSize);
    end
    if ~isa(psfSig, 'double')
        psfSig = double(psfSigma);
    end



%--Determine Noise Level--
%   Sample outer two concentric rings of pixels (assuming these are
%   representative of uniform background noise in the image). Define
%   background as the mean value of these pixels plus two standard
%   deviations.

%   Interrogate pixel values at image perimeter, 2 pixels deep
        sizeX = size(spotIm,2);
        sizeY = size(spotIm,1);
        bottom = spotIm(1:2, 1:sizeX);
        top = spotIm(sizeY-1:sizeY, 1:sizeX);
        left = spotIm(3:sizeY-2, 1:2);
        right = spotIm(3:sizeY-2, sizeX-1:sizeX);

%   Define perimeter pixels as background
        bkgdValues = double(cat(2, bottom, top, left', right'));
        bkgdVector = bkgdValues(:);
        N = length(bkgdVector);

%   Compute background level as mean + 2 standard deviations
        meanBkgd = sum(bkgdVector) / N;
        stdBkgd = sqrt( sum( (bkgdVector-meanBkgd).^2 ) / (N-1) );
        bkgdLevel = ceil( meanBkgd + 2*stdBkgd );

%   Identify pixels with background noise
        ImBkgd0 = double(uint16(spotIm-bkgdLevel));



%--Assign photons to pixel centers--
%   Identify the brightest pixel and set mu_i for all photons equal to the
%   centers of their respective pixels.

%   Determine pixel center values in nm (1 = x, 2 = y)
        centers(1,:) = pixelSize/2 : pixelSize : pixelSize*size(spotIm,2);
        centers(2,:) = pixelSize/2 : pixelSize : pixelSize*size(spotIm,1);

%   Determine location of max intensity pixel
        [~, Ix] = max(sum(ImBkgd0,1));
        [~, Iy] = max(sum(ImBkgd0,2));
        CM(1) = centers(1,Ix);
        CM(2) = centers(2,Iy);



%--Assign Variable Distribution Widths-- (slow version)
%   Makes photon distributions equal to the PSF near the max pixel and
%   infintiy more than 3 sigma_psf from the max pixel. Because a finite
%   (and small for small images) number of possible solutions exist, to
%   speed things up all possible sigma functions can be defined once,
%   outside of this function, and passed to this function to be looked-up.

%   Initialize variables
        sf = zeros(pixelSize*size(spotIm,2),2); %sigma function
        for i=1:2 %(1 = x, 2 = y)

%   Create sigma function
            for x = 1:pixelSize*size(spotIm,2)
                if x < CM(i)-3*psfSig,
                    sf(x,i)=Inf;
                elseif x > CM(i)+3*psfSig,
                    sf(x,i)=Inf;
                else
                    sf(x,i)=psfSig;
                end
            end

        end




%--Localize by Joint Distribution--
%   Involves projecting the image onto one dimension, forming a list of
%   photon coordinates, and assigning distributions for each photon-event;
%   then repeating for the second dimension and computing the center spot
%   location by as the joint of all independent distributions.

%   Initialize variables
        pe=cell(2,1); %photon-event list
        s=cell(2,1); %photon-event sigma list
        for i=1:2 %(1 = x, 2 = y)

%   Project image onto single axis
            projection = sum(spotIm,i); %(1 = x, 2 = y)
            bkgdProj = sum(ImBkgd0,i); %background pixels
            if i==2
                %rotate y-axis
                projection = projection';
                bkgdProj = bkgdProj';
            end

%   Build photon-event list
            pe{i} = zeros(1,sum(projection(:))); %photon-event list
            s{i} = Inf.*ones(1,sum(projection(:))); %set all sigma_i to Inf
            idx=1; %index of coordinates array

%   Convert pixel values into photon coordinates
            for j=1:size(projection,2)

%   Record mu_i
                nPhotons = projection(j); %number of photons to record
                if nPhotons>0
                    pe{i}(idx:idx+nPhotons-1) = centers(i,j)...
                                                             * ones(1,nPhotons);

%   Record sigma_i
                    if bkgdProj(j)>0
```

```
                        s{i}(idx:idx+bkgdProj(j)-1) = ...
                                                sf(uint16(centers(i,j)),i);
                    end

                end
                idx = idx + nPhotons; %increment index of coordinates array

            end

        end

%   Compute joint distribution
        photonEvents(1,:) = pe{1};
        photonEvents(2,:) = pe{2};
        photonDists(1,:) = s{1};
        photonDists(2,:) = s{2};
        sSquared = photonDists.^2;
        sSumInvSqr = sum(1./sSquared, 2);
        mu0 = sum(photonEvents./sSquared, 2) ./ sSumInvSqr;

end
```