New Methods for Measuring Transient Interactions Between Proteins and Curved Membranes

Philip P. Cheney
University of Denver

Follow this and additional works at: https://digitalcommons.du.edu/etd

Part of the Biological and Chemical Physics Commons, and the Physical Chemistry Commons

Recommended Citation
https://digitalcommons.du.edu/etd/1017

This Dissertation is brought to you for free and open access by the Graduate Studies at Digital Commons @ DU. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ DU. For more information, please contact jennifer.cox@du.edu,dig-commons@du.edu.
New Methods for Measuring Transient Interactions Between Proteins and Curved Membranes

A Dissertation

Presented to

The Faculty of Natural Sciences and Mathematics

University of Denver

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Philip P Cheney

August 2015

Advisor: Michelle Knowles
Abstract

Variations in the physical deformation of the plasma membrane play a significant role in the sorting and behavior of the proteins that occupy it. Determining the interplay between membrane curvature and protein behavior required the development and thorough characterization of a model plasma membrane with well defined and localized regions of curvature. This model system consists of a fluid lipid bilayer that is supported by a dye-loaded polystyrene nanoparticle patterned glass substrate. As the physical deformation of the supported lipid bilayer is essential to our understanding of the behavior of the protein occupying the bilayer, extensive characterization of the structure of the model plasma membrane was conducted. Neither the regions of curvature in the vicinity of the polystyrene nanoparticles or the interaction between a lipid bilayer and small patches of curved polystyrene are well understood, so the results of experiments to determine these properties are described. To do so, individual fluorescently labeled proteins and lipids are tracked on this model system and in live cells. New methods for analyzing the resulting tracks and ensemble data are presented and discussed. To validate the model system and analytical methods, fluorescence microscopy was used to image a peripheral membrane protein, cholera toxin subunit B (CTB). These results are compared to results obtained from membrane components that were not expected to show an preference for membrane curvature: an individual fluorescently-labeled lipid, lissamine rhodamine B DHPE, and another protein, streptavidin associated with biotin-labeled DHPE. The observed tendency for cholera toxin subunit B to avoid curved regions of
curvature, as determined by new and established analytical methods, is presented and discussed.
Acknowledgements

For my wife, Elizabeth.

You, like the stars in the sky, are evidence that this world is good and wonderful.
Introduction.................................................................................................................. 103

Methods.......................................................................................................................... 105
  Cell culture and transfection...................................................................................... 105
  Total Internal Reflection Fluorescence Microscopy (TIRFM)................................. 106
  Single Particle Tracking......................................................................................... 107
  Rhodamine Efflux Studies........................................................................................ 108
  Confocal Imaging....................................................................................................... 109

Results............................................................................................................................. 109
  PGP-EGFP removes rhodamine from the plasma membrane of live MES-SA cells. ............................................................................................................................ 109
  PGP-EGFP is mobile at short times and confined at long times (>0.7 s)........... 110

Discussion....................................................................................................................... 110

Conclusions...................................................................................................................... 111

Chapter 6 : Future Work................................................................................................. 112

Bibliography..................................................................................................................... 113

Appendix A: Tracking and analysis software source code ...................................... 119
List of Figures

Figure 1.1 The nanoparticle patterned supported lipid bilayer used in this work 3
Figure 1.2 The composition of the plasma membrane 5
Figure 1.3 The diffusion of objects in a colloid 7
Figure 2.1 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphocholine (POPC) 10
Figure 2.2 The procedure for making lipid films 13
Figure 2.3 Varying the TIRF angle 17
Figure 2.4 The size of the pixels in TIRF images 18
Figure 2.5 The mean deviation of a collection of stationary particles 19
Figure 2.6 The Bruker MSNL-10 AFM probe used for AFM experiments 21
Figure 2.7 Operation of the AFM 22
Figure 2.8 A sample well for imaging supported lipid bilayers in the AFM 23
Figure 2.9 Gold nanoparticles imaged by FM mode AFM in air 25
Figure 2.10 Gold nanoparticles imaged by AFM in aqueous buffer 26
Figure 2.11 The Rayleigh distribution 29
Figure 2.12 A series of sample displacement histograms 30
Figure 2.13 The turning angle 31
Figure 2.14 Representative frames from a typical FRAP image series 33
Figure 2.15 Vector operations 35
Figure 2.16 A very large increase in the processing rate of image series 37
Figure 3.1 Marina Blue-DHPE 42
Figure 3.2 FRAP of Marina Blue DHPE in fluid lipid bilayer 45
Figure 3.3 A 16 µm polystyrene microsphere 46
Figure 3.4 FRAP on 16 µm particles 48
Figure 3.5 Lipids surrounding 1 µm particles recover 59
Figure 3.6 The mobile fraction and the diffusion coefficient on 1 µm spheres 50
Figure 3.7 Lipid fluidity is measured by FRAP recovery 50
Figure 3.8 Supported lipid bilayers are only one bilayer thick 52
Figure 3.9 AFM of the lipid bilayer in the vicinity of a 100 nm nanoparticle 53
Figure 3.10 An AFM topograph of an individual nanoparticle and lipid bilayer 54
Figure 3.11 The lipid bilayer arrangement around polystyrene nanoparticles 55
Figure 4.1 The ganglioside GM1 60
Figure 4.2 The crystal structure of cholera toxin subunit B 62
Figure 4.3 An illustration of cholera toxin subunit B bound to GM1 62
Figure 4.4 Examples of mean squared displacement plots 67
Figure 4.5 The addition of multiple distributions 69
Figure 4.6 The displacement histograms of a single diffusing population 71
Figure 4.7 The turning angle is calculated as the deviation from a straight line 73
Figure 4.8 An illustration of turning angle histograms 73
Figure 4.9 Examples of diffusing lipids 74
Figure 4.10 The quantification of turning angle histograms 75
Figure 4.11 Residence time 76
Figure 4.12 Several fluorophores were tested 79
Figure 4.13 Location guided averaging 82
Figure 4.14 Displacement histograms of LRB-DHPE on 40 nm particles 83
Chapter 1 : Introduction

The motion of an individual protein that is associated with the plasma membrane of the cell can give great insight into the details of the environment that the protein is experiencing. The specifics of this environment include the viscosity of the lipid bilayer through which it moves, the presence of any other proteins with which it may interact, and the physical shape of the bilayer itself. Each of these different components can influence the motion of a protein in a variety of ways and the combination of all of these influences can lead to a situation where it is difficult to attribute a specific change in the motion of a protein to a specific influence. In a living cell, the plasma membrane is densely packed with proteins that interact with our moving protein in a variety of ways and also influence the lipid composition and physical deformation of the lipid bilayer that surrounds them. Likewise, the lipid composition in a region of the plasma membrane influences the physical deformation of the bilayer and dictates which proteins can be present in the region. With physical deformation also having effects on the lipid composition and the proteins that can inhabit a region, finding and attributing the interactions between a single moving protein and any one component is nearly impossible on the plasma membrane of a living cell.

The purpose of this work is the exploration of transient interactions between proteins moving across a lipid bilayer and regions of curvature in that bilayer, controlling all three of the previously mentioned parameters in order to be able to attribute any
observed effects to only the change in membrane curvature. We isolate any observed effects through the use of a model system of a cellular plasma membrane that was developed in this lab, in the form of a nanoparticle patterned supported lipid bilayer with tightly controlled lipid composition, protein occupation, and physical deformation.

In this dissertation, the model plasma membrane is characterized to determine its physical characteristics and used to investigate the behavior of individual proteins as they move across it. To accomplish the latter task, several new methods for interpreting the motion of individual proteins are introduced and discussed.

**Membrane curvature**

The curvature of the plasma membrane plays an important role in cellular function. Many biological processes involve actively deforming the plasma membrane and many processes are affected by changes in the shape of the membrane.\(^1\) Large scale cellular events such as mitosis involve physical deformation of the cellular membrane by a restructuring of the actin and microtubule cytoskeleton. Smaller scale events like the docking of secretory vesicles are facilitated by a series of proteins that capture the vesicle and mediate its fusion with another membrane. While the biochemical mechanisms involved in many of these cellular events have been studied, the interactions between membrane curvature and the function of proteins involved in this process are more difficult to determine.
The deformation of the plasma membrane can be caused by a number of factors including changes to its lipid composition, binding by curvature inducing proteins, and physical interactions with the cytoskeleton. The interplay between these factors does influence the manner in which membrane associated proteins move across the plasma membrane in ways not necessarily associated with the curvature of the membrane alone.

Throughout this work, membrane deformation will be discussed in terms of positive and negative curvature. Positive curvature is described as a convex deformation of the lipid bilayer, or a budding away from the solid support. Negative curvature is described as a concave deformation of the lipid bilayer with respect to the solid support. An illustration of our nanoparticle patterned supported lipid bilayer (not to scale) is shown in figure 1.1, which shows examples of positive and negative curvature of a lipid bilayer.

![Figure 1.1](image)

Figure 1.1 The nanoparticle patterned supported lipid bilayer used in this work. This model plasma membrane has regions of positive and negative curvature, which are separated by regions of no curvature. Note that this illustration is not to scale, as the lipid bilayer is considerably smaller than the nanoparticles.
Deformations in the lipid bilayer come at a great cost in energy. Because the lipid bilayer has thickness, deforming it will cause a change in the surface area of both leaflets. The positively curved leaflet will experience an increase in surface area and the distance between adjacent lipid headgroups will expand, exposing the hydrophobic tails to the aqueous environment outside of the bilayer, which is an entropically unfavorable situation. Filling these gaps would require the recruitment of more lipids to the area, which would increase the density of lipid tails in the hydrophobic core of the bilayer, but is less energetically unfavorable than solvating the hydrophobic center of the bilayer. At the same time, the negatively curved leaflet forces the often large headgroups together. In biological systems, this energetically unfavorable situation is stabilized by certain proteins, a change in the distribution of lipids on either leaflet, and by changes in the lipid composition in the deformed region. On our model system, where curvature is physically induced and not changing with time, the distribution of lipids among the leaflets will remain relatively constant.

Changes in lipid composition in particular regions are very common in biological systems and facilitates the binding of proteins and the accommodation of membrane deformation. The variety of lipids present in biological systems present a variety of physical properties that include different numbers of hydrophobic tails, hydrophobic tails of different lengths and degrees of unsaturation (and therefore different melting temperatures), and headgroups of different size and charge. The mixture of these lipids can allow for regions of bilayer to be formed that have different thickness and flexibility, which can significantly reduce the energetic cost of deformation. The formation of these different lipid regions can also significantly affect the behavior of resident
proteins. Because of the complex interactions between lipid composition and protein function, experiments on supported lipid bilayers are often conducted with an artificially limited selection of lipid components.

The membranes of a cell have varying composition, depending on the location and function of the membrane. Because they have different functions, the inner and outer leaflets of a membrane may also have very different composition. As shown in figure 1.2, the plasma membranes in a eukaryotic cell are made up of over half protein, by mass. The other half is lipid and carbohydrate. Of the lipid fraction of the membranes, there are a variety of headgroups (figure 1.2), fatty acid tail lengths, and degrees of unsaturation of the fatty acid tails. The outer leaflets of cellular membranes are especially enriched in phosphatidylcholine (PC) lipids. To mimic the outer leaflet of the plasma membrane in our experiments, we used bilayers primarily composed of 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphocholine (POPC) (figure 2.1).

Figure 1.2 The total composition of the cellular plasma membrane. Nearly half of the membrane, by mass, is lipid (left). The lipid fraction itself is made of many different lipids (right).
Diffusion on the plasma membrane

The plasma membrane can be understood as a fluid expanse on which its constituents may move only in two dimensions. Indeed, the fluid mosaic model proposed by Singer and Nicolson describes the cell membrane as a two-dimensional liquid.\textsuperscript{11} Though their model has seen several refinements\textsuperscript{12-14}, the basic premise of membrane components easily diffusing laterally along the membrane is still validated by experiment.

The rate of lateral diffusion of a component of the membrane is largely determined by the viscosity of the bilayer in which the component resides. As with diffusion in other media, the rate of diffusion will also depend on interactions with heterogeneity in the medium. As a result of the sorting of membrane lipids into different patches with different fluid properties, often referred to as microdomains\textsuperscript{15,16} or rafts, the diffusion rate of components may vary drastically over relatively short distances.

A model for predicting the diffusion rate of a cylindrical membrane component by consideration of the thickness and viscosity of the membrane and the viscosity of the surrounding fluid was proposed by Saffman and Delbrück shortly after the proposal of the fluid mosaic model. Their model is based on the Stokes-Einstein relation that predicts the diffusion coefficient of a sphere through a medium by the radius of the sphere and the viscosity of the fluid. The Saffman-Delbrück model describes the lateral diffusion of membrane components well at the scale of diffusing lipid microdomains, but becomes much less effective at describing smaller scale objects like small proteins or individual lipids.\textsuperscript{17} Also, large proteins with multiple membrane anchors do not appear to diffuse at
the rate predicted by Saffman-Delbrück.\textsuperscript{19}

Another drag component that influences diffusion, the intrusion of portions of a diffusing component into the lipid bilayer, has also been recently described.\textsuperscript{20} Interestingly, the viscosity of the aqueous fluid surrounding the lipid bilayer is two orders of magnitude lower, so the drag produced by protruding portions of membrane components do not contribute appreciably to their rate of diffusion.

Aside from the above interactions with the lipids of the bilayer itself, interactions with embedded proteins contribute to the observed rate of diffusion. These interactions can be chemical or also simply collisions between a diffusing component and a fixed or mobile protein obstruction.

Curvature in the membrane presents possibilities for interactions with diffusing proteins. Regions of positive curvature offer an increase in viscosity due to the crowding of the fatty acid. Also, defect sites and hydrophobic pockets exposed to the aqueous solution offer possible interactions. In the absence of defect sites or physical crowding of tails or headgroups, curvature-based enrichment of certain lipids offer a change in both

\textit{Figure 1.3 The diffusion of objects in a colloid and an early example of particle tracking.\textsuperscript{18}}
the viscosity of the bilayer and the potential for different chemical or electrostatic interactions between the lipids and a diffusing protein.

The interaction between mobile membrane components and regions of membrane curvature have been studied and modeled as an aggregate.\(^5,21-25\) The interaction between individual diffusing membrane components and membrane curvature has seen some interest recently, but it is not a well understood phenomenon.\(^26\) The work presented here describes methods that can be used to measure transient interactions between diffusing proteins and regions of curvature in the supported lipid bilayer.

This dissertation is divided into six chapters. The next chapter describes the general experimental and analytical methods used in this work. The third chapter discusses the current work in characterization of the nanoparticle patterned supported lipid bilayer. In the fourth chapter, ensemble and single molecule measurements of the diffusion of protein on the nanoparticle patterned supported lipid bilayer are described, as well as new methods employed for that purpose. The fifth chapter describes the use of single particle tracking techniques to quantify anomalous protein diffusion in live cells. Finally, Chapter 6 discusses future work that is motivated by this work.
Chapter 2 : Methods

Two different systems were used to determine the feasibility of identifying a particle’s interactions with the environment that it is exploring. These systems, live cells and our synthetic biosensor, required different approaches in preparation, imaging, and analysis.

In this chapter is described the criteria used for selecting the composition of both of the systems and the methods used to prepare the cells and the supported lipid bilayer for imaging. Also described are the methods for imaging the systems using fluorescence microscopy (total internal reflectance and confocal) and atomic force microscopy. Relevant biochemical assays are described to the extent that they were used. Also, while the theory behind relevant data analysis was described in the previous chapter, the stepwise protocol and a description of the tools used will be outlined in this chapter.

Preparation of curved biosensor

In the experiments carried out in this work, the primary component of all prepared lipid films was 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphocholine (POPC) (Avanti Polar Lipids, 850457C). As seen in figure 2.1, POPC is a singly unsaturated lipid with two fatty acid chains and a choline group incorporated onto the phosphate in the headgroup. At
biologically relevant pH, POPC is zwitterionic with a positive charge on the primary amine in the choline component and a negative charge on the phosphate group. Typical of phospholipids, the headgroup is hydrophilic and the fatty acid tails are hydrophobic. POPC has a melting temperature of -2 °C and no observed preference for curvature.\textsuperscript{5,27} In aqueous solution, POPC will form liposomes (spherical bilayers) instead of micelles (spherical monolayers), aiding in the deposition of supported lipid bilayers. While many previous studies involving supported lipid bilayers use primarily 1,2-dioleoyl-sn-glycero-3-phosphocholine (DOPC), DOPC will spontaneously form regions of negative curvature and we wanted all regions of curvature to be induced by the physical presence of the deposited nanoparticles.\textsuperscript{5}

Unless otherwise specified, all lipid mixtures used in the work contained a 2% mole ratio of Marina Blue labeled 1,2-dihexadecanoyl-sn-glycero-3-phospho-ethanolamine (DHPE) (Life Technologies, M-12652) for assessing the fluidity of the bilayer by fluorescence recovery after photobleaching (FRAP). Lipids intended for protein binding included a 1-2% mole ratio of the ganglioside GM1 (monosialotetrahexosylganglioside) (Avanti Polar Lipids, 860065P) or DHPE functionalized with a linker extended biotin (Biotin-X-DHPE, Life Technologies, B-1616). Single particle tracking experiments of individual lipids used a trace component of DHPE labeled with lissamine rhodamine B (LRB-DHPE) (Life Technologies, L-1392).
Typical observed density of LRB-DHPE on deposited bilayers is less than 1 molecule/µm².

Lipid films were made by mixing different species of pure lipids at the desired mole ratio. Lipids were mixed by first dissolving the individual lipid species in filtered chloroform and then adding the appropriate volume to a glass vial using glass syringes. The lipid/chloroform mixture containing vials were then wrapped in aluminum foil and placed under vacuum for 16 h to completely remove the chloroform. Following this, the vials were purged with nitrogen or argon gas, sealed with paraffin film, and stored at -20 ºC. All chloroform used in this work was filtered over basic alumina oxide and stored in the dark at -20 ºC.

All lipid films were made in batches of 250 nmol to keep the volumes of stock lipids in chloroform reasonable. As an example, a 250 nmol lipid mixture containing 98% POPC and 2% DHPE would contain 245 nmol POPC and 5 nmol DHPE. If chloroform stock solutions of 10 mM POPC and 1 mM DHPE were used, the mixture would be made by mixing 24.5 µL POPC solution and 5 µL DHPE solution.

The nanoparticle-patterned substrates were created in 8-well dishes (Lab-Tek® Chambered Borosilicate Coverglass System, Thermo) that contain 1.5 thickness glass coverslips on bottom. Glass surfaces were cleaned extensively by soaking in sodium dodecyl sulfate (SDS) (0.1% w/v) for 1 h, rinsing with deionized water then soaking in a 1% v/v bleach solution overnight. Cleaned coverslips were stored in ultrapure water (>18 megaohm*cm resistivity) containing 0.05% w/v sodium azide until use. Prior to use, the coverslip wells were filled with 100 µL of 2% Hellmanex (Hellma, Mullheim, Germany)
and allowed to sit for 1 h, then rinsed thoroughly with HEPES buffer (30 mM HEPES, 2 mM CaCl₂, 140 mM NaCl, pH 6.4).

Fluorescent, carboxylate-modified polystyrene nanoparticles (FluoSpheres, Life Technologies) were diluted in deposition buffer, sonicated, and deposited on the cover glass. For confocal imaging experiments, red fluorescent (580 nm excitation/605 nm emission) polystyrene nanoparticles (Life Technologies, F-8793, F-8801) were used. For TIRF microscopy, yellow fluorescent (505 nm excitation/515 nm emission) polystyrene nanoparticles (Life Technologies, F-8795, F-8803) were used. Nanoparticles were prepared for deposition by diluting the purchased 2% or 5% w/v nanoparticle suspensions at a ratio of 1:1000 in HEPES buffer in a 1.5 mL centrifuge tube and suspending the tube in a bath sonicator for 15 min to break up aggregates of particles. The sonicated suspension was then further diluted by 1:300 to 1:1000 depending on the density of deposited nanoparticles desired. The nanoparticles in this final diluted suspension were added to the 8-well dish in 100 µL and allowed to settle. After 30 min, the solution in the wells was removed.

Nanoparticle surfaces were coated with lipids using standard liposome deposition techniques.²⁸ Previously prepared lipid films were readied for use by adding 2 mL HEPES buffer and agitating with a probe sonicator (Fisher, Model 100, large probe size, power setting 2) for 5 min to create suspended liposomes. The nanoparticle patterned substrates were created by depositing 100 µL of liposome containing solution to each well and placing the covered dish in a 37 ºC incubator for 1 h. Following this, the dish was removed from the incubator and each well was washed by incremental addition and
removal of 100 µL of HEPES buffer three times to remove any liposomes that did not fuse with the bilayer on the surface of the well. The entire procedure is outlined in figure 2.2.

Figure 2.2 The procedure for making lipid films of the desired composition, creating liposomes, and depositing them on a solid substrate. (A) Lipids dissolved in chloroform are added to a glass vial in the appropriate mole ratio. (B) Chloroform is evaporated under vacuum and the resulting lipid film is stored under inert gas at -20 °C. (C) On the day of an experiment, the lipid film is suspended in aqueous buffer and sonicated to produce liposomes. (D) The suspended liposomes are deposited on the glass substrate where they fuse and form the supported lipid bilayer (SLB).

Cell culture

The MES-SA cell line (ATCC, CRL-1976) was used for all live cell experiments described in Chapter 5. MES-SA cells were chosen because they are adherent and maintain a wide, flat contact with the surface to which they are adhered. These characteristics make them ideal for use in total internal reflection fluorescence (TIRF) and confocal microscopy. These cells are also readily transfected with commercially available reagents (Lipofectamine 2000, Life Technologies, 11668027).
The MES-SA cell line was isolated from a human uterine sarcoma and has a fibroblast morphology. These cells do not natively express any appreciable level of P-glycoprotein and are sensitive to a number of chemotherapeutic agents (doxorubicin, dactinomycin, mitomycin C, and taxol). On exposure to these agents, expression of P-glycoprotein will be upregulated and the functional protein will be present on the cellular membrane.\textsuperscript{29}

A daughter cell line, MES-SA/Dx5 (CRL-1977, ATCC) was used in experiments as a positive control for P-glycoprotein expression in the absence of transfection. The MES-SA/Dx5 line was established from the MES-SA line by growing the cells in increasing concentrations of doxorubicin (from 10 nM, the IC50 of MES-SA, to 500 nM). The MES-SA/Dx5 cell line exhibits a marked cross resistance to many chemotherapeutic agents (doxorubicin, vinblastine, taxol, colchicine, dactinomycin, and mitomycin C) due to high expression of P-glycoprotein.\textsuperscript{30}

MES-SA has a doubling time of 22 to 24 hours and MES-SA/Dx5 has a doubling time of 30 hours. Cells were subcultivated at 1:6 to 1:10 every two to three days, depending on confluency and media acidification level. The target confluence for splitting a culture was 70\% and cell cultures were typically discarded if the media began to indicate high acidification (via phenol red indicator). Cells were exposed to a chelating solution, 0.02\% EDTA (Versene, Gibco), to encourage dissociation from the flask surface.

Cells were inspected at every passage to determine general health and phenotype. Passaged cultures that began to show a shift in phenotype were discarded. This was
typically observed around forty passages in MES-SA/Dx5 cultures, so all cultures were
dispersed and reestablished after thirty passages.

On receipt of the cell lines, a large culture was grown in a T-75 flask to 70%
confluence, dissociated from the surface with EDTA, and suspended in a freezing
medium containing DMSO (Recovery, Gibco). Aliquots of this suspension were stored
submerged in liquid nitrogen.

During culture, all cells were maintained at 37 °C in a humidified atmosphere
containing 5% CO₂ in McCoy’s 5A medium containing 1.5 mM L-glutamine, 100 U/mL
penicillin, 100 µg/mL streptomycin, and 10% fetal bovine serum. Cells were kept in
filter-capped T-25 sterile cell culture flasks. Cell culture flasks were reused at most one
time. After coverslip plating and/or transfection, cells were kept in appropriately sized
petri dishes with lids.

**Total Internal Reflectance Fluorescence Microscopy**

A total internal reflectance fluorescence (TIRF) microscope was used for all
single particle tracking experiments and for several of the bilayer characterization
experiments. TIRF is a method of microscopy that uses an evanescent wave to selectively
illuminate a small region of the sample directly above the glass coverslip. This region can
be as small as 100 nm and the illumination intensity decays exponentially as the distance
from the surface of the glass increases, which makes this method ideal for limiting the
field of observation to the membrane of a cell that is adhered to the glass or a supported
lipid bilayer that has been deposited on the glass. In a typical epifluorescent microscopy setup, any fluorophores that exist in the solution above the sample will still be excited by the illumination source and add out-of-focus emission to the captured images. This extraneous background light can obscure signals of interest and is especially problematic when the subject of interest is a single particle with very limited signal to begin with. In the case of the protein-binding experiments conducted in this work, the population of non-bound fluorescent protein molecules in solution is orders of magnitude larger than proteins bound to the membrane and reliably discerning bound proteins is extremely difficult.

The evanescent illumination field in TIRF microscopy is constructed by sending the excitation laser into the objective lens at an angle such that it will enter the glass sample slide and strike the upper boundary at an angle larger than the critical angle for the glass-sample interface. The critical angle for an interface is defined as the angle above which total internal reflection occurs and is determined by applying Snell’s law with the refractive indices of the glass slide and sample,

\[ n_g \sin \theta_i = n_s \sin \theta_t \]  \hspace{1cm} (2.1)

where \( n_g \) and \( n_s \) are the refractive indices of the glass and the sample, respectively, and \( \theta_i \) and \( \theta_t \) are the angles of the incident and transmitted light with respect to the normal of the interface. The critical angle \( \theta_c \), where \( \theta_t = 90^\circ \) (and so \( \sin \theta_t = 1 \)) and total internal reflectance occurs, can be found by rearranging the equation above to

\[ \theta_i = \theta_c = \arcsin \left( \frac{n_s}{n_g} \right). \]  \hspace{1cm} (2.2)
The borosilicate glass used in our experiments has an index of refraction of 1.525 and a POPC bilayer has a nominal index of refraction of 1.47,\textsuperscript{31,32} which would give a critical angle of 74.6° from normal. This is unfortunately complicated by the presence of an aqueous layer between the glass and the bilayer and will also depend on any local defects in the regularity of either component or proteins associated with the bilayer.\textsuperscript{32} These complications provide an excellent means of quantifying such properties as the thickness of the aqueous layer between the bilayer\textsuperscript{33} or the thickness of the bilayer and any attached proteins\textsuperscript{32}, but that is beyond the scope of this work. In the context of this

Figure 2.3 Varying the TIRF angle changes the extent of the evanescent field beyond the glass. At a very shallow angle with regard to the normal, the evanescent field will cease to be created and the illumination is referred to as shallow angle excitation. The micrometer reading referenced in the figure is related to the angle of the laser entering the objective lens of the microscope. The intensity of fluorescence emission from a fluorescent 200 nm nanoparticle is shown as the evanescent field is varied in penetration depth. At high micrometer readings, the excitation transitions from total internal reflectance to shallow angle excitation. At this transition point, the emission from the fluorescent nanoparticle begins to decline.
work, the complications (and others, such as the thickness of the oil layer between the objective and the glass) only mean that the critical angle must be determined empirically. Varying the incident angle around the critical angle will change the extent of the evanescent excitation field as seen in figure 2.3, where the fluorescence emission of a dye-labeled 200 nm nanoparticle is measured as the evanescent field extent is varied.

In this work, we used a Nikon TIRF microscope equipped with a 491 nm laser for excitation of the yellow-green emitting fluorophores, a 561 nm laser for excitation of red emitting fluorophores, and an EMCCD camera (Andor iXon 897+) for detection. A dual-color, TIRF filter set was used (Chroma) with emission filters at 525/50 nm and 595/60 nm. Between the microscope and the camera was a 1.5x optical magnifier. An etched micrometer calibration slide was used to determine the size of each pixel’s imaging field (figure 2.4), which was determined to be 107 nm per pixel.

In the experiments conducted in this work, the nanoparticles were stationary and so an image of the nanoparticles was taken prior to dynamic measurements of the fluorescently tagged lipids and proteins. Image series were typically captured at 22-33 Hz using MicroManager.³⁴ For the experiments described in Chapter 5, which involved imaging live transfected cells, the
cells were imaged on circular #1.0 thickness coverslips in a liquid containing imaging cell.

Figure 2.5 The mean deviation of a collection of stationary particles localized with both the new software described in this work (black) the MOSAIC particle tracking plugin in ImageJ (gray). Particles were reliably detected with a precision of 6 nm for the new software, as compared to 12 nm with MOSAIC. Also note that the distribution of deviations returned by the new software is much narrower.

Our imaging setup was tested with poly-L-lysine immobilized streptavidin labeled with four Alexa Fluor 546 dyes to determine the precision with which we could localize individual particles. As shown in figure 2.5, this was determined to be 6 nm.
Confocal Microscopy

Laser scanning confocal microscopy offers a method of imaging samples at a relatively high spatial resolution by using a pinhole to focus excitation light on a very small region of the sample and then imaging that same region though a pinhole. The use of the small pinhole apertures in this case allows excellent rejection of out-of-focus light as the only that can pass through the pinhole in any appreciable quantity is light that corresponds to the desired point of focus. In order to make images from this technique, it is necessary to scan the point of focus through the use of a galvomirror. Thus, this method is by necessity a raster scanning imaging mode and is incapable of taking full-frame images. There are some confocal microscopy designs that allow for very fast imaging by using a spinning disk containing pinholes instead of a scanning mirror, and these have been use for single particle tracking\textsuperscript{35}. However, the need to build an image by scanning multiple points limits the amount of light that can be collected from any single point and makes confocal microscopy inherently less capable of detecting fast-moving and dim single fluorescent particles.

In this work, images were taken using an Olympus FV1000 Laser Scanning Confocal Microscope. Nanoparticles were excited using a 488 nm diode laser; MB-DHPE, with a 405 nm laser; and labelled proteins with a 561 nm or 594 nm laser. FRAP images of Marina Blue labeled DHPE were acquired at a rate of 2 µs per pixel with an image size of 512 by 512 pixels. FRAP images more slowly moving proteins were acquired at a rate of 12.5 µs per pixel to 40 µs per pixel with an image size of 512 by 512
pixels. For all FRAP experiments, samples were bleached for 0.8 seconds in a 5.25 µm diameter circle.

**Atomic Force Microscopy**

Atomic force microscopy (AFM) experiments were conducted on an Asylum Research MFP-3D AFM. For all experiments, unless otherwise specified, Bruker MSNL-10 probes were used. This probe was chosen because it is very sharp: it has a 2 nm nominal tip radius and a front angle of 15 ± 2.5º (figure 2.6). This geometry makes it preferable for measuring the poorly accessible regions directly adjacent to the relatively tall nanoparticles. The probes are silicon nitride, with a gold coating on the back side of the cantilever for the readout laser. A basic diagram of AFM operation is shown in figure 2.6. Note that for the imaging modes used in this work, monitoring the state and behavior of the AFM probe is entirely dependent on monitoring the laser light being reflected from the back of the cantilever.

*Figure 2.6 The Bruker MSNL-10 AFM probe used for AFM experiments. (A) The sharp tip, with its small tip radius and steep angle, make it nearly ideal for imaging the small features adjacent to the nanoparticles. (B) Each probe contains six cantilevers for tuning the imaging setup to different sample properties and continuing to image after breaking off a cantilever.*
The gold coating on the AFM probe’s cantilever is responsible for a commonly encountered issue in which the tip continuously drifts toward the sample surface while imaging liquid samples. To attempt to avoid this phenomenon, we also imaged the bilayer with uncoated tips as other groups have successfully done. The sensitivity of the MFP-3D ultimately proved inadequate for this, however. Therefore, the need to monitor, and occasionally correct, the tip as it continually drifted toward the sample led to the production of artifacts in some of the resulting images.

Initial AFM experiments were performed on glass slides that were cleaned as usual for bilayer deposition. After cleaning, circles were drawn with a hydrophobic ink marker and a drop of HEPES buffer was placed within this circle. The drop used was essentially a volume of buffer just smaller than the hydrophobic circle could accommodate without flowing over. This volume was determined empirically, with practice, and differed based on the size of the hand-drawn hydrophobic circle. The deposition of the nanoparticles and the lipid bilayer, as well as all imaging, was carried out in this drop and it was replenished with water as it evaporated.
The method of imaging in a drop was not ideal. Evaporation was difficult to counter during long imaging sessions and any time the drop diameter decreased, the bilayer was irreversibly disrupted. To address this problem, we made a more stable imaging platform by gluing coverslips to a cleaned slide (figure 2.8). This produced an imaging chamber with shallow walls that were tall enough to contain a liquid drop, but small enough to allow the sample to be imaged without interference between the chamber and the AFM head. As the drop evaporates in this chamber, the liquid-air interface travels down the walls of the chamber a considerable distance before reaching (and disrupting) the lipid bilayer.

Figure 2.8 A sample well for imaging supported lipid bilayers in the AFM. The well is constructed by attaching coverslips to the top of a cleaned borosilicate glass slide. The height of the well walls is 170 µm, which minimizes the chance of impacting the sample well with the AFM head or probe holder. On the right is an illustration of the coverslip arrangement. After the coverslips are affixed to the slide, the extra glass is snapped off. The X denotes the sample well.

Imaging of the bilayers was carried out using FM mode AFM, which does not involve contact between the AFM probe and the sample. Contact mode AFM involves
measuring the deflection of the cantilever induced by contact between the AFM probe and the sample. The contact mode of operation is very sensitive, but leads to wear of the probe tip and the destruction of soft and sensitive samples. A different mode of operation, which does not involve continuous direct contact between the probe and the sample, is referred to as “tapping mode” and uses intermittent contact to make a topological map of the sample. This mode is also known as amplitude modulation mode, since data collection uses the blunting of a “tap” as evidence of contact between the probe and the sample. Though this mode is less damaging to the sample than contact mode, we found that imaging lipid bilayers with tapping mode led to the collection of lipids on the probe tip and a great decrease in the resolution offered by the probe as the tip became fouled with lipid material.

To avoid these problems, we moved to using a non-contact frequency modulation (FM) mode. In FM mode the cantilever is driven to oscillate just below its resonant frequency, which is determined by monitoring the thermally induced vibrations of cantilever in the imaging medium. Instead of sensing direct contact between the probe and the sample, the resonating probe detects the damping of the cantilever’s oscillation by interactions between the van der Waals forces exerted by the surface of the sample and probe. Note that this method will also be sensitive to any other long range force extending above the surface of the sample, such as electrostatic potentials. In FM mode, the oscillation of the cantilever is monitored and a feedback loop is employed to maintain a constant amplitude of oscillation. The frequency and phase of the oscillating cantilever is collected and this readout is what gives information about the sample being imaged.
In order to ensure that the relatively minute features surrounding nanoparticles could be imaged with adequate resolution, gold nanoprisms were selected to image sharp edges of a similar size. As seen in figure 2.9, nanoprisms and their dissociated lipid coating could be imaged in air as well as nanoprisms without a lipid coating on a patchy poly-L-lysine coated surface. In liquid, the sharp edges of the nanoprisms could still be adequately discerned, as shown in figure 2.10.

Figure 2.9 Gold nanoprisms imaged by FM mode AFM in air. The dissociated lipid coating can be seen in the image on the left. On the right, a patchy layer of poly-L-lysine can be seen coating the surface of a slide holding uncoated gold nanoprisms.
Single Particle Tracking

By photobleaching the fluorophores on proteins in a small region on the lipid bilayer and watching them be replaced with unbleached fluorophores from outside of the bleached region, we can measure the rate of diffusion of the labeled proteins and the fraction of protein that is immobile and unable to be replaced. If there are several populations of diffusing protein or the protein exhibits several different diffusion rates, which can also be discerned from ensemble recovery curves. However, the detection and measurement of multiple populations or a combination of rates is only possible if the ensemble is large enough and the temporal resolution of the measurements is high enough to accurately recover several components from the curve. This last requirement
may be met by increasing the rate of the measurements or modifying the membrane composition or temperature to slow diffusion.

Since effectively meeting both of these criteria may impose conditions that change the behavior of the proteins or membrane in undesirable ways, techniques like single particle tracking become very attractive, as they do not require large concentrations of protein and are inherently compatible with many different rapid acquisition imaging techniques like TIRF microscopy.

The localization and tracking of diffusing particles collected by TIRF microscopy was conducted in ImageJ with the particle tracking plugin from MOSAIC for the live cell imaging in Chapter 5 and in MATLAB with algorithms developed for this work and with tracking algorithms based on those published by Crocker, Grier, and Weeks\textsuperscript{38} and adapted by Blair and Dufresne (http://site.physics.georgetown.edu/matlab/). The basic workflow for analyzing single particle data is described in this section and can be followed in the comments of the appropriate programs in Appendix A, especially loadAndTrack2 and loadAndTrack3parallel, which are the main entry points into the analysis software.

loadAndTrack2 and loadAndTrack3parallel are largely the same, with the latter being a partial redesign with parallel computing on the high performance computing cluster in mind. Within several of the functions called by this main program, a check is made to see if a cluster is accessible and, if so, functions optimized for parallel operation are used instead of serially executed loops. A sample script for batch submission of jobs to the HPC cluster is provided in SampleBatchSubmitScript.m.
Images of single particles moving on the supported lipid bilayer were captured at as high a rate as possible, which decreases the signal to noise ratio of the individual frames. To limit the impact of the single pixel noise on particle localization and to provide a uniform background, each frame of the image series was bandpass filtered. The original bandpass filter used by Crocker and Grier did two 2D convolutions in real space on each frame with a Gaussian kernel and a boxcar kernel and then subtracted the boxcar convolved image from the Gaussian convolved image to lowpass and highpass filter each frame. This method was extremely slow and did not produce any better results than directly filtering the images in Fourier space with a finite impulse response filter, so the latter approach was adopted instead.

After each frame of an image series was bandpass filtered, a mean intensity projection (MIP) of the entire series was produced and bandpass filtered in the same manner as the individual frames. This effectively resulted in an image of any particles that remained static throughout the entire series and could be regarded as immobile. This bandpassed MIP image was subtracted from every frame of the image series, which served to remove stuck particles from the series. Further processing later in the workflow was used to find immobile particles by examining the track of each individual particle.

After filtering the images, particles were detected by first finding peaks of high intensity in each frame and then finding the centroid of each peak to sub-pixel accuracy using the cntrd function from Dufresne. The resulting particle positions were passed to a modified version of Blair’s track function to produce a series of trajectories for each
particle found in the image series. For the experiments in this work, tracks shorter than five frames were discarded and gaps of one frame were allowed.

The trajectories of each particle were then compared to a map of the static nanoparticles in separate tracks to return an indication of any colocalization events during a particle’s track.

After colocalization events were determined, the displacements of each particle were inspected to find particles that occupied the same approximate location in both the first and last frame of the series in separateStuck, as these were assumed to be immobile. It is possible that a mobile particle could coincidentally move for a long time and end up in the same place, but this seemed very unlikely to apply in many cases. Other methods of finding “stuck” particles include inspection of the mean squared displacements of individual particles and this method could also be applied in this workflow if so desired.

![Figure 2.11](image)

*Figure 2.11 The probability distribution function, $P(r)$, and cumulative distribution function, $D(r)$, of the Rayleigh distribution.*

After separating tracks into mobile and stuck subsets, trajectories of the particles were then analyzed to produce displacement histograms over several different lag times.
using *getdx5*. Displacement histograms were plotted as probability distributions, as they provide a more intuitive plot than cumulative distributions (figure 2.11). A probability distribution allows the identification of different populations as different peaks. A series of sample displacement histograms over various lag times are shown in figure 2.12.

![Displacement Histograms](image)

*Figure 2.12 A series of sample displacement histograms over 66 ms, 132 ms, 330 ms, and 660 ms. Note that the peak broadens as the lag time increases.*

The mean squared displacement of individual particles were calculated from the individual particle tracks using *msdInidividual* and the excellent *msdanalyzer* function from Jean-Yves Tinevez. An average mean squared displacement was calculated from these individual mean squared displacements using *msdanalyzer* or directly using *msd*.
The angle between adjacent steps of a tracked particle was calculated in the
*turnangle* function. This was reported as an unsigned angle from 0 to $\pi$, with 0
representing no change in the direction of motion and $\pi$ representing a complete
reflection in the direction of motion. This is depicted graphically in figure 2.13. The
evaluation of turning angles of diffusing particles represents a relatively underutilized
technique, with limited use in fields such as animal dispersal.\(^{40}\) Recently, it has begun to
see some utility as “track straightness” in particle tracking.\(^{41}\) As such, we were excited to
determine the utility of the turning angle on the elucidation of transient interactions
between diffusing particles and obstacles, such as protein aggregates or membrane
curvature.

Various statistics were also calculated and saved in a report. These included results such as the
number of nanoparticles located, the number of tracks found, the number of colocalization events, the
number of stuck particles, and the number of mobile particles. The time that particles remained
colocalized, the residence time, was calculated in
*getresidtime* as well. Figures and tables for many of the previously calculated results were created and
saved in the various *plot*... functions.

Plots of the particle tracks, with the stationary nanoparticles indicated, and the
individual displacements of each track colored by mean squared displacement,
displacement length, colocalization with nanoparticles, or useful diagnostic schemes such as track-beginning to track-end, were made with plotTracks. For diagnosing potential problems with sample contamination, dx1Int was used to assess the intensity of particles as a function of their displacement length and the results were displayed with plotScatter. The results of these different representations are presented in Chapter 4 and Appendix B.

For the purposes of combining the results of datasets from multiple individual image series from multiple days, combineResults, combineDxInt, and combineTrajAndMSD were used. All of the results discussed in Chapter 4 have been compiled from individual experiments in this manner.

**FRAP Analysis**

The data gathered from FRAP experiments are not directly comparable to each other, as the fluorescence intensity collected by the confocal microscope can be quite variable. To allow FRAP data from different experiments to be compared, the movies are analyzed as described here.

All FRAP images were corrected for imaging induced photobleaching by correcting the intensity to the intensity of a region distant from the FRAP region of interest (ROI). The mean intensity measured in each experiment’s FRAP ROI was normalized to an interval of 0 to 1. The highest value in each series of images was set to 1, which typically corresponded to the first images taken of the ROI. The lowest value in each series of images was set to 0, which typically corresponded to the frame
immediately following the bleach step. This can be seen from the typical FRAP series shown in figure 2.14.

![Figure 2.14](image)

*Figure 2.14 Representative frames from a typical FRAP image series. Note the substantial drop in fluorescence and the subsequent recovery. As the photobleached fluorophores mix with the unbleached fluorophores in the rest of the sample, the overall fluorescence intensity may decrease. This image series shows MDR-EGFP on a transfected cell’s plasma membrane. The frames shown are -16, 0, 16, 32, and 140 s after bleaching.*

The resulting curves were fit equation 2.3 in Prism 6, where $F_0$ is the intensity immediately post-bleach (typically 0), $F_\infty$ is the asymptote that the increasing curve approaches, and $k$ is the rate constant of the recovery.

$$F_t = F_0 + (F_\infty - F_0) \times (1 - e^{-kt}) \quad (2.3)$$

If more than one diffusing component was expected, extra components were added to the fit, as shown in equation 2.4, where $P_f$ is the fractional contribution of the first component and $(1 - P_f)$ is the fractional contribution of the second component. In this case, $k_1$ is the rate constant of the first component and $k_2$ is the rate constant of the second component.

$$F_t = F_0 + (F_\infty - F_0) \times P_f \times (1 - e^{-k_1t}) + (F_\infty - F_0) \times (1 - P_f) \times (1 - e^{-k_2t}) \quad (2.4)$$
The diffusion coefficient, $D$, was extracted from the resulting fit using equation 2.5, according to established methods, where $r$ is the radius of the bleached region, $\tau_{1/2}$ is the half-time, and $\gamma_D$ is a constant related to the shape of the bleached spot\textsuperscript{42}:

$$D = \left(\frac{r^2}{4\tau_{1/2}}\right)\gamma_D$$

(2.5)

For a circular bleach spot, $\gamma_D$ will be 0.88. The half-time, $\tau_{1/2}$, is directly calculated from the rate constant of the recovery

$$\tau_{1/2} = \frac{\ln(2)}{k}$$

(2.6)

The mobile fraction was then determined for each component by $F_\infty \cdot P_f$ and $F_\infty \cdot (1 - P_f)$.

**Software development**

The analysis software used for localizing and tracking particles in Chapter 5 was from MOSAIC and used as a plugin to ImageJ. While it worked well for tracking, we found it unsuitable for tracking a very large number of image series with minimal user interaction. For this reason, we developed a suite of software around the tracking algorithms provided by Crocker, Grier, and Weeks.\textsuperscript{38} Their software was originally written in IDL, but has been adapted to MATLAB by Blair and Dufresne. Much of this adapted code was used in our work, with some functions rewritten for speed or compatibility with other functions.
The MATLAB environment was chosen because we already had licenses for it and there are many algorithms for particle localization, tracking, and analysis already written and freely available. Further, modules exist to allow simplified and powerful interaction with ImageJ. The University’s high performance computing (HPC) cluster has MATLAB installed on it, which allowed us to analyze our large datasets very quickly in a parallel manner. Finally, MATLAB is well suited to quickly developing and implementing new algorithms and functions, which allowed new analytical methods to be developed relatively quickly.

Throughout the development of the tracking software in Appendix A, efforts were made to keep all functions and operations as generalized as possible to allow for reuse in other contexts or to facilitate the addition of additional steps or dimensions to the analysis. As programming these functions was a learning experience on my part and my intentions are to leave this code to be used and improved by others in the lab, every attempt was made to keep the code readable by logically implementing functions, thoroughly commenting the code, and using descriptive variable names instead of overly short and cryptic ones. All of the functions were designed to operate on two color image series of

\[ S_1 + S_2 \rightarrow S_3 \]

\[ V_1 + V_2 \rightarrow V_2 \]

*Figure 2.15 Scalar operations act on data a single step at a time, while vector operations act on multiple sets of data simultaneously.*
two dimensional dynamics, but care was taken to allow for the use of more colors or dimensions where possible.

Efforts were also taken to avoid the unnecessary use of loops and to instead use vectorized operations instead (figure 2.15), which are much faster in MATLAB. Briefly, vectorized operations are carried out on many data elements simultaneously instead of individually as in a typical loop. Efforts were made throughout development to use data formats that lent themselves to vectorized operations.

A great deal of the analysis lends itself to parallel execution, so later efforts were made to adapt the written code for parallel execution. The analysis time was reduced by these efforts on the computer used for analysis and was vastly reduced by these efforts when analysis was regularly run on the HPC cluster. Through several iterations of optimization, the time required to analyze an image series was reduced by a factor of four and execution on the HPC reduced analysis time by over an order of magnitude. Figure 2.16 shows the reduction in analysis time for a typical image series as averaged over several hundred experiments. Much of the variation in analysis time is due to a different number of particles and tracks being present in different image series.
A series of optimizations in the programs used in this work led to a very large increase in the processing rate of image series. Moving the processing to the University’s high performance computing cluster further increased the rate.

Care was taken to avoid the unnecessary use of additional MATLAB toolboxes and to use freely available implementations of these toolbox functions if possible. The only additional toolboxes required are “Statistics” and “Image Processing”, but “Distributed Computing”, “Bioinformatics”, and “Global Optimization” will be detected and used if available.
Chapter 3: Characterization of patterned lipid bilayer biosensor

Introduction

There are a variety of in vitro assays and sensors that have been developed to study protein and lipid dynamics on the curved lipid bilayer. Many of these tools, such as liposomes, lipid coated nanoparticles in solution, lipid tubes, and patterned surfaces, allow the measurement of interactions between curvature and protein and lipid sorting. Each assay has limitations, many of which are overcome with the nanoparticle patterned substrate described here. For example, in liposome based assays, polydispersity in size is an issue but overcome with single liposome based assays. However, liposomes lack a continuous membrane connecting flat and curved regions and measurement of dynamics is not possible. In other assays, ensemble measurements are made. Finally, in assays that have connected regions of positive and negative membrane curvature, single molecule imaging has been demonstrated, but the high extent of curvature, meant to mimic exocytic and endocytic processes in the substrate described here, is novel. Although all of these assays have their strengths, the nanoparticle patterned substrates characterized here add a missing tool to the toolbox of biochemically based, membrane curvature assays.
The biosensor presented in our group’s work addresses many of the shortcomings of previous biosensors. It does this by allowing the separate tuning of curvature and lipid composition, providing a continuous membrane with flat and curved regions in dynamic equilibrium, and allowing for the detection and localization of regions of curvature and individual membrane-associated lipids or proteins.

The work described in Chapter four of this dissertation involves investigating the feasibility of detecting transient interactions by tracking individual proteins, which places several more constraints upon the system that make most other membrane curvature assays unsuitable for the task. In a model that mimics a relevant biological system, the only sensing modality that offers high resolution, high rate of capture, and environmental compatibility with the sample is optical imaging. As the interactions may be very short lived, it is necessary to capture images of the system at as high a rate as possible. This constraint limits the microscope to a full-frame microscopy method.

Since both the diffusing proteins and many of the biologically relevant regions of curvature are considerably smaller than the diffraction limit of light, it is necessary to have a reporter of membrane curvature that provides excellent spatial resolution. To accomplish this, the nanoparticle patterned substrate uses fluorescent dye loaded carboxylate-modified polystyrene nanoparticles to both induce curvature in a supported lipid bilayer and provide an easily localized indicator of their location. The need to identify regions of curvature as well as moving particles sets up limitations of its own. Labels other than fluorescent dyes have been used to track diffusing proteins, such as gold nanoparticles imaged through interferometric detection of scattering. This method
has immense benefits in particle localization and tracking speed, but suffers from the need for equipment that is not commercially available and the current inability to localize regions of curvature and diffusing particles simultaneously. Using nanoparticles that are labeled with a different fluorescent dye than the protein or lipid of interest allows our sensor to monitor diffusing components and the stationary curvature-inducing nanoparticles simultaneously on commonly available microscopes.

The surface on which a bilayer rests, as in the supported lipid bilayer model used in this work, influences the mobility of membrane components in both leaflets of the bilayer.\textsuperscript{52-56} This is observed even though there is often a water layer separating the lower leaflet from the substrate itself.\textsuperscript{33} Also, even though interactions between the supporting substrate and the lower leaflet have a greater impact on the mobility of components in the lower leaflet, coupling between the two leaflets leads to observed changes in mobility in the upper leaflet.\textsuperscript{56}

Supported lipid bilayers deposited on clean glass substrates have no observed differences in bulk mobility between bilayers.\textsuperscript{52} As the hydrophilic glass surface allows for a water layer to exist between the substrate and the lower leaflet headgroups\textsuperscript{57,58}, the lipids in the lower leaflet are thought to be somewhat cushioned from all but large defects in the cleaned glass surface.\textsuperscript{59,60} Recent work has shown that there may even be some influence on the mobility of individual membrane components from extremely small surface defects.\textsuperscript{53,54}

Substrates other than glass, such as mica\textsuperscript{52} or polymer\textsuperscript{61}, have very different effects on the mobility of supported lipid bilayer components. The polystyrene
nanoparticles used as a source of curvature in this work have a different composition than the glass substrate surrounding them, so the interaction between the lipid bilayer and the nanoparticles should also differ. The polystyrene nanoparticles used are carboxylate-modified, which is achieved by grafting a carboxylic acid-rich polymer to the surface of the polystyrene sphere. The result is described as having "a highly charged, relatively hydrophilic and somewhat porous surface layer." The interactions between such a surface and a supported lipid bilayer are not well characterized, but are discussed in this work.

Expanding on the previous characterizations of the biosensor in this lab, this work addresses some critical assumptions and questions about its structure. In this chapter, a test for the fluidity of the bilayer is described along with the associated analysis and results. Also discussed is the fluidity of the bilayer where it contacts polystyrene, which is compared to contact with glass and poly-L-lysine. The impermeability of the bilayer and presence of a single, intact bilayer in the assembled biosensor is demonstrated through a quenching assay. Finally, the results of characterization of the bilayer by atomic force microscopy are presented.
Methods

FRAP of fluid bilayers

The biosensor was prepared as described in the second chapter, using previously prepared lipid films containing 1-2% MB-DHPE (figure 3.1). Marina Blue photobleaches rapidly, so the higher 2% content was preferred to maximize the fluorescent intensity during the FRAP experiments. Bilayers intended exclusively for MB-DHPE FRAP experiments were 2% MB-DHPE and 98% POPC. Bilayers intended for other experiments contained 1-2% of other components, such as GM1 or biotin-X-DHPE, as well as MB-DHPE and were tested for consistent diffusion and recovery of the MB-DHPE component.

All experiments were conducted in HEPES buffer (30 mM HEPES, 2 mM CaCl₂, 140 mM NaCl, pH 6.4). Lipid bilayers with deposited nanoparticles used 40 nm and 100
nm green fluorescent (505 nm excitation, 515 nm emission) carboxy-modified polystyrene nanoparticles (Life Technologies FluoSpheres).

Images were taken on an Olympus FV1000 Laser Scanning Confocal Microscope, as described in detail in Chapter 2. The fluorescent nanoparticles were excited with a 488 nm laser and the Marina Blue was excited with a 405 nm laser. Images of the MB-DHPE component were taken at 2 µs per pixel and an image size of 512 by 512 pixels. The bleaching step was conducted for 0.8 s on a circular region of 5.25 µm in diameter. Bleaching was carried out using the “tornado” tool, which bleaches the region of interest (ROI) in an expanding spiral from the center of the ROI. Bleaching Marina Blue was done with the 405 nm laser at 100%. Imaging was conducted with both lasers at a minimal power, typically 1-5% total power.

FRAP images were analyzed using ImageJ and GraphPad Prism 6. All FRAP images were corrected for imaging induced photobleaching by correcting the intensity to the intensity of a region distant from the FRAP ROI. The mean intensity measured in each experiment’s FRAP ROI was normalized to an interval of 0 to 1. The highest value in each series of images was set to 1, which typically corresponded to the first images taken of the ROI. The lowest value in each series of images was set to 0, which typically corresponded to the frame immediately following the bleach step. The resulting curves were analyzed as described in Chapter 2.
**NBD quenching assay**

The fluorescent dye NBD (2-(4-nitro-2,1,3-benzoxadiazol-7-yl)aminoethyl) (excitation 458 nm, emission 530 nm) is reduced by the dithionite anion (S\(_2\)O\(_4^-\)) to form a nonfluorescent dye. As dithionite is not able to pass through an intact lipid bilayer, this reaction is used to demonstrate access to a leaflet of the bilayer by dithionite. For our experiments, a stock of sodium dithionite was made at 10 mM in HEPES buffer. When used, 100 µL of stock was added to the 100 µL of HEPES already in the well. The pore forming protein, melittin was prepared at 1 mg/mL (350 µM) in HEPES buffer. Melittin was added to the well (now 200 µL) in a 1 µL volume for a final concentration of 1.76 µM. Images were taken at a rate of 10 Hz on the TIRF microscope with excitation from the 491 nm laser.

**Atomic Force Microscopy**

Atomic force microscopy experiments were conducted as described in Chapter 2.

**Results**

**Fluidity of deposited bilayers**

To ensure that the lipid bilayers used in our experiments were correctly formed and intact, the fluidity of the supported lipid bilayer was regularly tested by fluorescence recovery after photobleaching (FRAP) experiments. As mentioned in the description of the choices regarding phospholipid constituents in the previous methods chapter, the
majority of the bilayers prepared were composed of 1-2% Marina Blue DHPE (MB-DHPE) (Figure 3.1). The ultraviolet (365 nm max) absorption and blue (460 nm max) fluorescent emission of Marina Blue made that constituent essentially invisible on the TIRF microscope during single particle imaging in the green/red emission channels, but allowed for fluidity investigation of every sample on the confocal microscope.

To ensure that the membranes were fluid, FRAP assays of the MB-DHPE component were regularly carried out during the course of experimentation. On any occasion in which a potential problem was suspected with the fluidity of the bilayer, FRAP of the MB-DHPE component offered an available internal standard for fluidity. As MB-DHPE was previously established to not sense curvature\textsuperscript{25}, this method of measuring bilayer fluidity was applicable in all of the experiments performed in this work.

![Figure 3.2 FRAP of Marina Blue DHPE in fluid lipid bilayer. The bilayer is composed of 98% POPC, 2% MB-DHPE. The diffusion coefficient, D, is in units of $\mu m^2 s^{-1}$ and the plateau is expressed as the fraction of complete recovery. The error is shown as SEM. n=3.](image)

The diffusion coefficient for MB-DHPE on SoyPC bilayers was previously established by our group to be $3.70 \pm 0.41 \mu m^2 s^{-1}$\textsuperscript{25}. Diffusion on a primarily POPC bilayer is
expected to be slower by a significant amount due to the increased saturation of its tails. From FRAP experiments, it was determined to be $0.35 \pm 0.02 \ \mu m^2 s^{-1}$ (Figure 3.2).

The plateau of the recovering fluorescence of MB-DHPE, indicative of the fraction of the component’s population that is mobile, is in excess of 80% of the initial fluorescence (prior to photobleaching the sample). This is typical for a fluid bilayer without any anchored components.  

*Fluidity of bilayer over large polystyrene microspheres*

There has been some concern about the fluidity of bilayers deposited on polystyrene surfaces. To examine the extent to which this interaction affects our biosensor, we tested the fluidity of the bilayer on particles of diameters ranging from 40 nm to 16 µm.
We found that there was a substantial loss of fluidity on extended surfaces, as seen on the bilayer deposited on the 16 µm particles (Figure 3.3). The 16 µm polystyrene microsphere gives an opportunity to study the behavior of the supported lipid bilayer on a non-diffraction limited surface, while also having access to the well characterized glass surface below it. A 2% NBD-DHPE, 98% POPC bilayer was used to coat the polystyrene particle and the cleaned glass surface below it. As seen in Figure 3.3, the fluorescent bilayer can be seen coating the surface of the particle. The top of a particle and a slice through the center of it was examined. A 20 µm² circular region was bleached in both cases and the sample was observed for fluorescence recovery. There is no recovery seen over 46 s of continuous imaging (Figure 3.4). Even after 2 min and 10 min, there is still no recovery of the photobleached region (Figure 3.4). The flat surface of the slide below the particle, however, is fluid and recovers normally (not shown).

Figure 3.3 (A) A differential interference contrast image of a 16 µm polystyrene. (B) A partial confocal depth scan of the microsphere shows evidence of a fluorescent lipid bilayer coating, which is also seen in subsequent figures.
Figure 3.4 Fluorescence recovery after photobleaching does not occur on 16 µm particles that have extended bilayer contact with carboxy-functionalized polystyrene. (A) Shown is a slice of an NBD-DHPE bilayer coated 16 µm diameter particle that fails to recover from a 2 µm diameter bleach spot. Before bleaching, immediately, 20 s, and 40 s after bleaching. (B) A similar experiment on the top of the microsphere does not show recovery after 10 min. Before bleaching, immediately, 20 s, and 40 s after bleaching. Scale bar is 10 µm. (C) Quantification of the fluorescence in a-d shows no recovery occurs.
As the diameter of the particles decreases, we see an increase in the fluidity of the deposited bilayer. We have demonstrated in previous experiments that photobleached lipid bilayers on 1 µm polystyrene particles recover (Figure 3.5). At smaller diameters, between 40 nm and 200 nm, we see no significant difference between the fluidity of the bilayer above a surface decorated with nanoparticles and a flat surface (Figure 3.7). That the recovery of photobleached regions is greater than 50% makes a compelling argument for both leaflets of the bilayer being fluid. Fixing a single leaflet of the bilayer to the surface of the slide with poly-L-lysine significantly impacted the recovery of the photobleached region (Figure 3.6).
In this and previous work\textsuperscript{25}, we established that neither the diffusion rate, nor the total recovery, is significantly affected by the 40-200 nm polystyrene particles that are used in this work (Figure 3.7). Fluidity is also retained to some extent by bilayers deposited on particles as large as 1 μm in diameter (Figure 3.5). While the experiments shown in figures 3.5-3.7 were conducted with SoyPC, which is primarily composed of 1,2-dilauroyl-sn-glycero-3-phosphocholine (DLPC) with two unsaturated tails, it is expected that lack of interaction with small polystyrene nanoparticles translate to bilayers solely composed of POPC, or many other phosphocholine lipids, as the headgroup.

Figure 3.6 (A) The mobile fraction and (B) the diffusion coefficient are unchanged relative to flat surfaces. Surfaces coated with poly-L-lysine, which causes lipids to stick to the glass, limit lipid motion. This is shown here with SoyPC bilayers.

Figure 3.7 Lipid fluidity is measured by FRAP recovery. Lipid bilayers containing SoyPC and MB-DHPE are bleached at 0.0 s. (A) Average recoveries of 15 FRAP experiments are shown for 200 nm (green), 100 nm (blue), and (B) mobile fraction and (C) diffusion coefficient are unchanged relative to samples containing no nanoparticles.
Deposited bilayers are single and continuous

To demonstrate that deposited lipids formed a single, continuous bilayer on the surface of the slide, a fluorescence assay using NBD quenched by sodium dithionite was used. Melittin, a protein complex that embeds itself in a lipid bilayer and forms a pore, is added to allow dithionite to penetrate the bilayer and reach the NBD-DHPE in the lower leaflet.

This assay has been previously used to assess the continuity of lipid bilayers on small particles. While it is very reliable in that situation, applying it to the extremely large extended bilayers employed in our biosensor is much more challenging. The bilayer in our biosensor is remarkably uniform over large areas, but any small defects anywhere on that extended surface will allow dithionite to reach the lower leaflet, even without the addition of melittin. In the case of a leaky or imperfect bilayer, the NBD quenching assay can still be used to demonstrate that the sensor is composed of a single bilayer and not multiple layers by comparing the initial and later rates of quenching. While this situation was observed in most of the NBD quenching assays, an example of a defect-free and continuous bilayer is presented here.

For this assay, a lipid mixture containing 2% NBD-DHPE was deposited on a surface along with 100 nm red fluorescent polystyrene nanoparticles. The quenching reaction was observed on the TIRF microscope using the 491 nm laser and an exposure time of 100 ms. After imaging the bilayer fluorescence for 30 s, sodium dithionite was added for a 5 mM final concentration. After 120 s, melittin was added for a ~2 µM final concentration.
As seen in Figure 3.8, the initial addition of sodium dithionite results in a rapid decrease in fluorescence to around 50% of the initial intensity. After the addition of melittin, the fluorescence drops at a much slower rate to 0%. In this assay 0% intensity is defined as the lowest intensity seen in any image and appears to be the noise floor.

**Atomic Force Microscopy characterization**

In order to understand the behavior of proteins and lipids as they move across the lipid bilayer in the vicinity of nanoparticles, it is important to know how the lipid bilayer is shaped in these areas. The shape of bilayers around silica nanoparticles has been characterized by AFM. 66,67 This work shows that pores in a 1,2-dimyristoyl-sn-glycero-3-phosphocholine (DMPC) lipid bilayer are formed around nanoparticles that are ~1.2 nm to 22 nm in diameter and that nanoparticles larger than 22 nm are closely wrapped in the bilayer.

*Figure 3.8 Supported lipid bilayers are only one bilayer thick. The bilayer is composed of 2% NBD-DHPE and 98% POPC and imaged on the TIRF microscope. At ~30 s, sodium dithionite is added, leading to a rapid drop to near 50% of the initial fluorescence intensity. After ~120 s, melittin is added, leading to a more gradual drop to background fluorescence intensity.*
Because our previous work has shown a difference in bilayer fluidity when the lipids are in contact with polystyrene and borosilicate glass, it is important to not assume that the bilayer will conform to polystyrene nanoparticles in the same manner that it conforms to silica nanoparticles. Also, our use of the more biologically relevant POPC, with its drastically different phase transition temperature ($T_m(POPC) = -2 ^\circ C$, $T_m(DMPC) = 24 ^\circ C$), would lead us to expect different susceptibility to physically induced curvature.

Atomic force microscopy was used to probe the lipid bilayer in our biosensor to determine its topology in the vicinity of polystyrene nanoparticles. Characterization was conducted in HEPES buffer, as the lipid bilayer is not stable in the absence of an aqueous layer around it. The lipid bilayer itself is relatively soft and easily deformed, so we used frequency modulation non-contact mode AFM and a tip with a very low spring constant to avoid contact between the AFM probe and the bilayer.

![Figure 3.9 A height profile of the lipid bilayer in the vicinity of a 100 nm nanoparticle. Note that the profile closely resembles a tent. A nanoparticle illustration shows the scale (blue).](image)

53
Our location guided averaging experiments, discussed in chapter 4, and FRAP experiments have demonstrated that the supported lipid bilayer is continuous above the polystyrene nanoparticles of sizes from 40 nm to 1 μm. This appears consistent with what has been reported for silica nanoparticles, where pores in the lipid bilayer only form around nanoparticles smaller than ~22 nm. This property is also validated by our AFM experiments, which show the presence of a bilayer on the top of all of the investigated nanoparticles (figure 3.9). The measured height of nanoparticles above the surrounding bilayer is consistently within the size range of the provided nanoparticles. The presence, and height, of the bilayer surrounding the nanoparticles is confirmed by measuring the depth of defects in the bilayer where the glass is exposed. These defect sites are typically 4-5 nm deep, which is consistent with the thickness of a POPC bilayer.

To manipulate the lipid bilayer without approaching the hard glass surface with the AFM probe tip, we contacted one of the nanoparticles with the AFM probe and repeatedly pushed it across the imaged area, leaving a trench in the bilayer behind it. This
contact provided an opportunity to see the deformed bilayer next to the undisturbed bilayer and the presumably bare glass (figure 3.10).

The two most compelling ways in which lipid bilayer may lie over the nanoparticles is in a tightly wrapped manner (A) or a tent-like manner (B) (figure 3.11). The driving forces behind the bilayer conformation around nanoparticles are primarily membrane tension and any attraction/repulsion between the lipid bilayer and the carboxyl-modified polystyrene. A potential source of attraction or repulsion includes slight electrostatic interactions between the phosphocholine headgroups of the lipids and the carboxyl groups coating the nanoparticles, which are considerably more basic as a result of being attached to polystyrene. Though the effect is significantly more pronounced with amidine-functionalized nanoparticles, there has been observed membrane association and endocytosis of polystyrene nanoparticles. As a result of the unusually basic carboxyl group, the nanoparticles are quite hydrophobic in solution, which presents a source of repulsion between the polar headgroups of the lipids and the nanoparticle surface.

Figure 3.11 The lipid bilayer arrangement in the vicinity of polystyrene nanoparticles could present in one of several ways. (A) A tightly wrapped lipid bilayer and (B) a tent shaped lipid bilayer.
Investigating the biosensor with AFM leads us to conclude that the lipid bilayer in the vicinity of the nanoparticles is tent-shaped instead of tightly wrapped. This means that the annulus of negative curvature surrounding the nanoparticles is extended over a larger area than it would be if it was tightly wrapped, and so is less curved than the tightly wrapped interpretation.

**Discussion**

Our understanding of the physical characteristics of the biosensor is improved as a result of this work, which should help greatly when predicting the behavior of proteins interacting with it. We have shown two experiments that demonstrate that a single fluid lipid bilayer is present and that it is continuous across the top of the nanoparticles. The AFM, FRAP and NBD quenching experiments demonstrated this. We have also shown that the bilayer is fluid where it contacts the polystyrene nanoparticles, at the nanoparticle sizes that we are using. Also, we are beginning to get a better idea of the physical contour of the lipid bilayer in the vicinity of the nanoparticles. This was previously not well known for systems using polystyrene, so this is a very important observation.

There is still a considerably amount of characterization that can be done of the biosensor, though, as well as improvements that can be made. Further study of the biosensor by AFM would be desirable to continue the work started here. A great deal of time was spent trying different methods of preparing the sample, imaging the biosensor with our equipment, and optimizing sample longevity and integrity while imaging. Due to this, it is felt that more numerous and higher resolution images of the biosensor could still
be captured. Higher resolution scanning of the biosensor would provide even more information about the shape of the bilayer tent, which could help with modeling and predicting interactions with it.

Further AFM studies could also include poking the biosensor in different regions to record the force required to deform or penetrate the bilayer. This would provide information on the thickness and tension of the bilayer in different regions. Also, AFM topology measurements of various levels of protein saturation of the biosensor could be used to demonstrate the preference of proteins for different regions of curvature. This could be augmented with super-resolution optical microscopic methods.
Chapter 4 : Single particle tracking on curved lipid bilayers

Introduction

The lateral diffusion of protein on a cellular membrane is affected by many different properties of the membrane, including lipid composition, interactions with membrane-associated proteins, membrane curvature, and even the arrangement of nearby structures like the actin cytoskeleton. Because of this complicated environment of tightly interconnected interactions, attributing changes in lateral diffusion to the modification of any single component can be difficult.73

In order to measure the changes in the lateral diffusion of this particular membrane associated protein complex as it encounters regions of curvature, we used a model system of the plasma membrane with very tightly controlled chemical composition and physical structure. This model system, as described in the previous chapter, employs fluorescent polystyrene nanoparticles to induce regions of membrane curvature and provide the means to accurately localize them. The chemical composition is also controlled and extraneous protein is excluded, so that it may not influence the dynamics of the protein of interest.
The protein of interest in this work is the membrane associated protein complex, cholera toxin subunit B (CTB), which is widely used by researchers as a marker for lipid microdomains in the plasma membrane. However, it is not clearly understood how CTB interacts with the regions of extreme curvature that are commonly found in cells. This work aims to characterize the interactions between CTB and membrane curvature, which will aid in the interpretation of biological assays as well as provide a better fundamental understanding of protein-curvature interactions.

Explored in this chapter are the observed interactions between cholera toxin subunit B and the nanoparticle patterned supported lipid bilayer. These interactions were investigated with bulk colocalization measurements, as well as single particle tracking methods. New methods of analyzing transient interactions from single particle tracking data are also described in this chapter, as well as the results that they provide. The dynamics of cholera toxin subunit B is also compared to a single lipid and another protein, streptavidin, to provide context for the results. The motion of CTB bound to GM1 on the supported lipid bilayer displays behavior consistent with obstructed diffusion.

**Ganglioside GM1**

The ganglioside GM1 (monosialotetrahexosylganglioside) is an oligosaccharide that is present on the surface of some cells, most notably neuronal cells and the epithelia of mucosal tissues. In neuronal cells, it is involved in development and repair mechanisms. In the epithelia of mucosal tissues, especially in the intestinal lumen, it is
known as the binding site used by the entry mechanisms of protein enterotoxins of Vibrio cholera and Escherichia coli. All of the known gangliosides account for 10-12% of the lipid content of neuronal membranes and 1-2% of the lipid content of other tissues.\footnote{74}

As seen in figure 4.1, GM1 has a polysaccharide headgroup consisting of Neu5Acα2-3(Galβ1-3GalNAcβ1-4)Galβ1-4Glc which is β1 linked to a ceramide that is composed of sphingosine and stearic acid. The lipophilic fatty acid tails are embedded in the lipid bilayer and the hydrophilic polysaccharide headgroup resides in the aqueous layer surrounding the bilayer.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{ganglioside_GM1_diagram.png}
\caption{The ganglioside GM1 is a lipid membrane component that consists of a sialic acid containing polysaccharide attached to the amino alcohol sphingosine and a single stearic acid. This lipid has two fatty acid tails that are embedded in the lipid bilayer, with the soluble carbohydrate headgroup protruding into the aqueous layer above the bilayer.\footnote{76}}
\end{figure}
Cholera toxin subunit B

The cholera toxin is a protein toxin secreted by bacteria of the species Vibrio cholerae. This toxin is responsible for the symptoms characteristic of cholera infection and acts by triggering a sequence of events that locks adenylate cyclase in an active state and results in a massive efflux of chloride ions from infected cells. This efflux of chloride ions is accompanied by the secretion of water and other ions into the intestinal lumen and leads to severe diarrhea and dehydration.

The assembled cholera toxin protein complex is composed of six subunits: one A subunit and five B subunits. The A subunit is 28 kDa and is responsible for the ADP-ribosylation of the Gs alpha subunit proteins, which begins the pathway that leads to the symptoms described above. The A subunit has two notable chains: the A1 chain which is the enzyme payload of the toxin and the A2 chain, which binds to the assembled subunit B pentamer.
Five B subunits, each 11 kDa, assemble to form a ring-shaped pentamer. The assembled pentamer, CTB, binds the ganglioside GM1 and serves to deliver the A subunit payload to the inside of the cell. CTB is capable of binding five GM1 molecules and the binding pocket for GM1 is contained largely with a single subunit B monomer, as seen in the crystal structure. Cholera toxin subunit B is an extended planar structure with five points of membrane association. Cholera toxin has 3.5 nm between binding pockets and ring diameter of 7 nm.

Assembly of CTB from its constituent monomers occurs spontaneously in neutral pH via six hydrogen bonds at each interface. In the absence of the A subunit, assembly of the pentamer can occur through a...
number of combinations of monomers and dimers and GM1 binding is possible by any construct larger than a monomer. The dissociation constant of a subunit B multimer to monomers is $K_d = 540 \text{ nM}$.

As all of the five binding pockets are located on the same face, CTB is capable of securing itself to the GM1-laden membrane through five bound anchors. The dissociation constant of CTB for the oligosaccharide portion of GM1 is in the low micromolar range, giving it one of the highest affinities known between a protein and a carbohydrate. CTB shows selective affinity for GM1 and an interesting cooperative binding effect between adjacent and nonadjacent binding sites.

At low pH, around 3.7, CTB disassembles into individual subunits. Reassembly of the subunits to a pentamer is concentration dependent, but also has a dependence on pH. At the CTB concentration of 8.6 $\mu$M and between pH 6.5 and 6.0, there is a loss of pentameric CTB and the dominant species found in solution are monomers and dimers. Above pH 7.0, the reassembly of subunit B monomers to pentamers is not significantly inhibited.

*Cholera toxin subunit B senses curvature*

Some evidence has been found that CTB has a preference for residence in regions of negative curvature on supported lipid bilayers. Recent work used a wavy glass substrate to show that dye-labeled CTB partitions into regions of negative curvature. Using concentrations of CTB high enough to image with confocal microscopy (150 – 500 $\mu$M), this curvature based partitioning was apparent, however at concentrations suitable
for single molecule tracking (0.05 nM), there was no evidence of any preference for negative curvature. They concluded that the curvature preference was dependent on some sort of cooperative effect between multiple CTB pentamers.

In their work, they used a DOPC bilayer on a wavy surface that had smoothly continuous waves of 110 nm (peak to trough) over 1 µm (peak to peak). They did not find any preference for curvature from streptavidin bound to cap-biotin DPPE or dye labeled DHPE in ensemble or SPT experiments. As the Kₐ for CTB subunit disassociation is 540 nm, as stated above, and the binding pocket for GM1 lies almost entirely within the subunit, it is reasonable to conclude that the CTB populations observed in their experiments were not entirely pentamers but contained many monomers, dimers, trimers, and tetramers in the single molecule tracking experiments. This situation is also true for the single particle experiments in this work because all tracking work was done below the subunit-pentamer Kₐ and slightly acidic conditions.

**Methods**

*Choice of dye for single particle tracking*

When selecting between fluorophores for single particle tracking, several different dyes were tested. These fluorophores were tested as dye-protein conjugates to allow for multiple fluorophores in a single diffraction-limited spot, as would be the case in single particle tracking experiments. The following fluorophores were used: Qdot605, Alexa Fluor 546, Alexa Fluor 568, and Alexa Fluor 594 (Life Technologies). Potential
fluorophores were tested for overall intensity and photostability under normal single molecule TIRF imaging conditions.

**Supported lipid bilayer composition**

As discussed in Chapter 1, the major lipid component of the outer leaflet of eukaryotic cells is PC. All of the bilayers in this chapter were composed of primarily POPC, with 2% MB-DHPE, and the balance made up of the lipids of interest: 2% GM1 or 1% biotin-X-DHPE. Lipid bilayers with deposited nanoparticles used 40 nm and 100 nm green fluorescent (505 nm excitation, 515 nm emission) or red fluorescent (580 nm excitation, 605 nm emission) carboxy-modified polystyrene nanoparticles. All experiments were conducted in HEPES buffer (30 mM HEPES, 2 mM CaCl$_2$, 140 mM NaCl, pH 6.4).

**Single particle experiments**

Single particle experiments were conducted using lissamine rhodamine B DHPE (LRB-DHPE) (Life Technologies, L-1392), Alexa Fluor 594 Cholera Toxin Subunit B (Life Technologies, C-22842), and Alexa Fluor 546 Streptavidin (Life Technologies, S-11225). The concentration of LRB-DHPE was estimated by its presence on the SLB and was typically near 1 molecule per µm$^2$. The concentrations of the protein components used were determined by the absorbance of the dye at its peak absorbance and the manufacturer reported labeling density (4 mol dye/mol protein). The concentration range of labeled protein used was ~ 5 nM. As this is below the dissociation constant of the CTB
pentamer (K\textsubscript{d} = 540 nM\textsuperscript{23}), but required for SPT imaging, there are complications which will be described in the following sections.

\textit{FRAP of proteins on lipid bilayer}

Images were taken on an Olympus FV1000 Laser Scanning Confocal Microscope, as described in detail on Chapter 2. The fluorescent nanoparticles were excited with a 488 nm laser and the Alexa Fluor 546 or Alexa Fluor 594 labeled protein (Life Technologies, S11225 and V34405) was excited with a 559 nm laser. Images of the membrane-bound component were taken at 12.5 µs per pixel and an image size of 512 by 512 pixels. The bleaching step was conducted for 1.6 s on a circular region of 5.25 µm in diameter. Bleaching was carried out using the “tornado” tool, which bleaches the region of interest (ROI) in an expanding spiral from the center of the ROI. Photobleaching of Alexa Fluor 546 proved to be quite difficult, so the bleaching step used several lasers (405 nm, 488 nm, 559 nm) at 100\% power. All imaging was conducted at a minimal laser power, typically between 1-5\%. Analysis of the FRAP recovery data was conducted as described in Chapters 2 and 3.

\textit{Location guided averaging analysis of protein/curvature colocalization}
Location guided averaging of the bilayer was conducted using the `run_ministk` program in Appendix A on confocal microscopy data. This program finds the positions of nanoparticles in their channel and collects a series of 25 x 25 pixel cropped regions of the nanoparticle channel and the fluorescent protein channel as previously described. The mean of all of these cropped regions is then obtained and the mean intensity of every pixel is plotted as a function of the radial distance from the center of the cropped region to the center of each pixel. Multiple pixels with the same radial distance are averaged before they are plotted.

Analysis of Single Particle Tracking Data

Large amounts of data are produced by the single particle tracking programs and this section deals primarily with the interpretation of these data. Some of these results are common to ensemble measurement techniques as well and apply equally to those techniques.

---

![Image](image-url)  
*Figure 4.2 Examples of mean squared displacement plots and conclusions that can be drawn from anomalous diffusion. Depicted are (top) purely diffusive motion, (middle) actively directed motion, and (bottom) constrained motion.*
The mean squared displacement (MSD) of a moving particle over many lag times will give insight into the particle’s interaction with the medium through which it is moving. In a homogenous medium in which the timescale of any interaction is significantly slower than the measured lag time, the MSD will scale linearly with the lag time. This is regarded as purely diffusive motion and the magnitude of the MSD at any lag time can be used to find the diffusion coefficient, D.

Anomalous diffusion is diffusive motion in which the MSD does not scale linearly with the lag time. Anomalous diffusion can be described by a power law

\[ \langle r^2 \rangle \propto D \tau^a \]  

(4.1)

where \( \langle r^2 \rangle \) is the MSD, \( D \) is the diffusion coefficient, \( \tau \) is the lag time, and \( a \) is the factor by which the MSD deviates from normal purely diffusive motion.\(^{84}\) When \( a > 1 \), particles are moving faster than can be described by purely diffusive means and their motion can be described as actively directed motion. This type of motion is not expected in the systems studied in this work. When \( a < 1 \), particles are moving slower than would be expected from purely diffusive motion and their motion is described as obstructed or confined motion. Examples of these plots are shown in figure 4.4.

Another method of interpreting the results of particle tracking analysis is by building histograms of the distance travelled by the individual particles over different time scales. The distribution of the displacements of individual particles over a period of time are most accurately described by the Rayleigh distribution,
\[ R(r) = \frac{r}{2Dt} e^{-\left(\frac{r^2}{4D}\right)} \] (4.2)

where \( r \) is the distance travelled by a particle, \( D \) is the diffusion coefficient, and \( t \) is the time that the particle is allowed to travel. If there are multiple populations of diffusing particles, the observed distribution of displacements will be the sum of the distributions of each population, as such:

\[ R(r) = A_1 \frac{r}{2D_1t} e^{-\left(\frac{r^2}{4D_1t}\right)} + A_2 \frac{r}{2D_2t} e^{-\left(\frac{r^2}{4D_2t}\right)} \] (4.3)

where the only factors determining the different distributions is the diffusion coefficients of the different populations and the relative size of each population is represented by \( A \).\(^{19}\) The addition of multiple distributions is shown in figure 4.5.
In typical experimental conditions, it is not uncommon to find multiple populations of moving particles, some of which may have extremely disparate rates of diffusion. In figure 4.7 is shown an example of a distribution of a normally diffusing population of particles and the combined distributions of normally diffusing particles and extremely slow-diffusing, or immobile, particles. This situation is commonly encountered in single particle tracking experiments and it is important to note that over the time scales measured in this work, it is not possible to accurately extract any diffusion parameters from the immobile distribution.

Figure 4.5 The addition of multiple distributions is used to fit multicomponent data.
Figure 4.6 The displacement histogram series of a single uniformly diffusing population (top) and the addition of a diffusing population and an immobile population.
The motion of individual lipids and proteins in a lipid bilayer is dictated by their interactions with other components in the bilayer and these interactions can be almost entirely reduced to the temperature and the viscosity of the surrounding medium. Further refinement of this model by Saffman and Delbrück maintain the dependence on only the temperature, viscosity, and the shape of the diffusing particle. However, the motion of particles in a fluid is greatly dependent upon interactions between a particle and its environment. These interactions influence its rate of diffusion and the direction of motion available to it. Insight into the availability of directions in which a particle may move can be found by looking at the distributions of angles between subsequent steps in a particle’s track. In our work, we measure how far molecules travel in time (the rate of motion) as well as describe ways of measuring the direction of motion favored by mobile molecules (the “turning angle”).

If the lateral diffusion of a particle in a bilayer lacks interactions with the environment, the distribution of angles between subsequent steps (turning angles) will be uniform regardless of the step size chosen. The rise of variations in the distribution of turning angles, as the time interval is varied, serves as a means to determine the time scales at which any interaction between a particle and its environment occur.
Turning angles have not often been used in analyzing single particle tracks, but there is increasing attention being paid to their potential. Histograms of turning angles are made by plotting the frequency of the angle between adjacent displacements, as shown in figure 4.7. Turning angle histograms can show a tendency of particles to continue moving in the same direction or turning around and moving back in the direction from which they originated. The former case is consistent with the actively directed motion observed in MSD plots. The latter case is consistent with obstructed motion and can be thought of as particles being reflected off of obstructions. An ideal depiction of these two cases is shown in figure 4.8.
and a representative sample of moving particles being reflected from nanoparticles is shown in figure 4.9.

![Freely diffusing lipids and lipids in the vicinity of nanoparticles](image)

*Figure 4.3 Examples of freely diffusing lipids and lipids in the vicinity of nanoparticles.*

To quantify the results of the turning angle plots, the histograms were allocated into seven bins over the range 0 to $\pi$. The bin corresponding to the smallest angles and the bin corresponding to the largest angles were compared to the smallest of the three central bins to give a ratio of low and high angles to intermediate angles. The smallest of the three central bins is used instead of the central bin to avoid biasing the resulting ratios if the enrichment or depletion of the 0 or $\pi$ angles extends into the central bins. This quantification method is shown in figure 4.10.
Changes in turning angle distributions over longer time scales can be used to separate infrequent transient interactions from interactions that take place over longer spans in time or space. To separate transient interactions from long-term interactions, turning angle histograms were also plotted over various lag times. The results of this analytical method are discussed later in this chapter.

Another technique that provides insight into transient interactions between membrane components is the residence time of a mobile particle in the immediate vicinity of a static particle. If the residence time is longer than the temporal imaging resolution, the residence time can be measured by counting the number of frames that a mobile particle spends adjacent to the stationary nanoparticle. These events can also be located by plotting the intensity of the mobile particle’s fluorescence channel at the site.
of nanoparticles by time and finding periods in which the intensity increases, as seen in figure 4.11.

Figure 4.5 The intensity in the diffusing protein channel increases when a particle is in the vicinity of a nanoparticle. The time that the protein spends above the nanoparticle is the residence time.

Results

Choice of dye for single particle tracking

Accurately locating individual proteins and lipids can be challenging because they are typically labeled with only a single, or very few, fluorescent dyes. As described in Chapter 5, fluorescent proteins can be tracked with great success and are an ideal choice when imaging live cells. Fluorescent proteins are very large compared to organic dyes and are typically dimmer than a single organic dye, offering very few advantages on a supported lipid bilayer system. Quantum dots have also been used successfully\textsuperscript{87}, but present some unique problems that are discussed in this chapter.
Other non-fluorescent labels, such as gold nanoparticles, have been used to track lateral diffusion in supported lipid bilayers. Gold nanoparticles tracked by interferometric detection of scattering methods offer extremely high spatial and temporal resolution (20 nm, 1 ms), but at the cost of very high illumination requirements (~10 kW/cm²) and the lack of multiple imaging channels for tracking multiple distinct diffusing species. Dark field microscopy techniques have been used to track gold nanoparticles at resolutions as high as 2 nm and 9 µs, but achieving these results required the use of rather large nanoparticles (40 nm) and high illumination intensities.

As the work in this section required the use of multiple imaging channels and selective illumination of the lipid bilayer on a TIRF microscope, we explored the use of several organic dyes and quantum dots. The TIRF microscope has two lasers, at 491 nm and 561 nm, and the dye choices were limited to dyes excited at or near these wavelengths. The choice of fluorophores was optimized for single particle tracking experiments and was not necessarily ideal for confocal fluorescence microscopy experiments. This was considered a reasonable compromise.

Initially, the prospect of using quantum dots was attractive and we attempted the use of quantum dot labeled streptavidin to track biotin-DHPE. They have high quantum efficiency (> 90%), extremely high extinction coefficients (> 4 x 10⁶ cm⁻¹M⁻¹ at 350 nm and > 1 x 10⁶ cm⁻¹M⁻¹ at 488 nm for Qdot605) across a very large band, and do not suffer from photobleaching after extended excitation like organic dyes. These properties would make them ideal for single particle tracking. Compared to organic dyes, they are very large (5 x 12 nm for the rod shaped Qdot605).
correspondingly increased drag of quantum dots should not impact the diffusion rates of their lipid bilayer associated counterparts, assuming the quantum dot remains in the aqueous phase above the bilayer and does not interact with the lipid headgroups in any significant way. They are commonly used in both live cell and SLB contexts.\textsuperscript{87-89}

Two issues led us to not pursue the use of quantum dots in our single particle tracking experiments and though ultimately both of these issues are solvable, the merits of quantum dots were not great enough to pursue their use. The issues that limited the utility of quantum dots in our experiments were the common use of a single quantum dot conjugated to multiple proteins and the blinking phenomenon that quantum dots are very well known for. The use of multiple proteins, streptavidin in the case of this work, attached to each quantum dot resulted in multiple membrane associations being represented by a single fluorophore. This will lead to a slower rate of diffusion. Indeed, very low diffusion rates are seen in SPT experiments using quantum dots decorated with streptavidin to bind biotinylated lipids in the bilayer.\textsuperscript{90} If membrane penetration occurs, the resulting drag will cause the rate of diffusion to be considerably lower.\textsuperscript{20} Membrane penetration may also lead to long-term interactions with the glass substrate below the bilayer or defect sites within the bilayer. When imaging Qdot605-streptavidin, which had an average of ten streptavidin tetramers attached to it, we saw brief diffusion followed by every particle becoming stuck (data not shown). This could be solved by optimizing the labeling efficiency and seeking a 1:1 ratio of Qdot605 to streptavidin, but this was not attempted.
Another issue that the use of quantum dots presented was caused by the blinking phenomenon that they exhibit. This intermittency of fluorescence emission is a stochastic process that may lead to tracked particles “going dark” for seconds at a time, or remaining “on” for the entire experiment. As the phenomenon relates to competition between fluorescence emission and nonradiative relaxation paths, it can be addressed by bandgap engineering or the use of different solvents to suppress blinking. The use of such bandgap engineered for SPT is beginning to be used with great success, but are not yet commercially available. Blinking can also be addressed to some extent by allowing for gaps in the tracks of individual particles. The tracking code used in this work was capable of handling this, but allowing gaps in tracks can lead to misleading position.
connections being made if the gaps are large or the particle fields are dense. For these reasons, we chose not to pursue the use of quantum dots in our work.

Organic dyes are small, bright, and very well characterized fluorophores. They suffer from irreversible photobleaching from sustained or high intensity excitation, but this can be avoided by selecting photostable dyes, minimizing excitation intensity, and by many other approaches. Their relatively small size allows the use of multiple organic dye labels per protein, which mitigates photobleaching to some extent by allowing the particle to lose intensity stepwise instead of all at once. This also allows the creation of fluorescent particles that are almost as bright as a single quantum dot, yet significantly smaller and without the problems described above. This can be seen in figure 4.11, where the intensity and photostability of multiple fluorophore-labeled particles are compared.

Ultimately, we decided to use Alexa Fluor 546 dye for its high intensity, excellent photostability, and compatibility with emission filters currently in our possession. It is commercially available preconjugated to proteins of interest to us or as a labeling kit, which also made it very attractive. The high extinction coefficient near a laser in our instrument (~112,000 cm$^{-1}$M$^{-1}$ at 556 nm) and reasonable Stokes shift (17 nm) made it an ideal choice for our TIRF microscope. When this dye was not suitable for our uses, we used other dyes that compared favorably, such as Alexa Fluor 594 (excitation 590 nm, emission 617 nm, extinction coefficient 92,000 cm$^{-1}$M$^{-1}$ at 590 nm).

A red dye was chosen for the diffusing particles and a green dye was chosen for the stationary nanoparticles in SPT experiments. The motivation behind this choice was to minimize the collection of emission from the extremely bright nanoparticles while
imaging the diffusing particles. Because the light required for the excitation of the red dyes used on the diffusing particles is much lower energy than that required to excite the green dyes in the nanoparticles, excitation of the green dye was minimized.

*Ensemble behavior of bilayer participants*

Ensemble measurements of proteins bound to biological membranes offer some insight into their interactions. These bulk measurements find transient interactions by sampling large numbers of interactions and looking for variations in the spatial arrangement of particles. This works well unless the nature of the transient interactions is affected by the high concentrations needed for ensemble measurements. To determine whether we were able to detect interactions between membrane-bound proteins and regions of membrane curvature, we investigated several membrane-associated components using a bulk measurement method.

In this work, we used an analytical method developed in this lab to quantify the colocalization of different components from confocal microscopy images. This method, location guided averaging (LGA), gives insight into enrichment or depletion of fluorescent component around nanoparticles by averaging the images of many nanoparticles to reduce measurement noise. This work uses location guided averaging to show avoidance of membrane curvature by CTB. The results of this ensemble measurement method is compared to the SPT methods employed later in this chapter.
Colocalization of proteins with curvature on the supported lipid bilayer

Results from the location guided averaging of confocal microscopy images show no preference at all for DHPE or streptavidin, but an apparent avoidance of curvature by CTB (figure 4.13). This avoidance is not seen in the DHPE or streptavidin data. This result begins to paint a picture of diffusing CTB not only lacking a preference for curvature, but also avoiding curvature entirely.

Figure 4.13 Location guided averaging shows an apparent preference for the avoidance of curvature by cholera toxin subunit B. The x axis in this figure represents the distance from the center of the nanoparticle in μm. The y axis represents the normalized intensity of the fluorescence in the CTB channel. Shown are Alexa594-CTB (500 nM), Alexa546-streptavidin (500 nM), and LRB-DHPE (2% mol/mol). Error bars represent SEM, n=4.
The behavior of individual bilayer components

The analysis of individual diffusing proteins and lipids show an interesting story of their own. The results shown in this section demonstrate the advantages present in single particle tracking analytical techniques. The behavior of multiple distinct populations can be isolated from each other more readily, allowing more accurate identification and analysis of the individual populations.

To determine the diffusion coefficient of lissamine rhodamine B DHPE on 40 nm nanoparticle patterned surfaces, displacement histograms (figure 4.14) were analyzed. They show two distinct populations of diffusion: a fast moving population with a diffusion coefficient, $D_1$ of $0.637 \pm 0.009 \, \mu m^2 s^{-1}$ over a lag time of 91 ms, $D_1$ of $0.546 \pm$
0.008 \mu m^2 s^{-1} over a lag time of 228 ms, and D_1 of 0.345 \pm 0.008 \mu m^2 s^{-1} over a lag time of 456 ms. There is also a slow moving population with a diffusion coefficient, D_2 of 0.0171 \pm 0.0004 \mu m^2 s^{-1} over a lag time of 91 ms, D_2 of 0.0135 \pm 0.0003 \mu m^2 s^{-1} over a lag time of 228 ms, and D_2 of 0.0118 \pm 0.0003 \mu m^2 s^{-1} over a lag time of 456 ms.

To determine the diffusion coefficient of lissamine rhodamine B DHPE on 100 nm nanoparticle patterned surfaces, displacement histograms (figure 4.15) were analyzed. They show two distinct populations of diffusion: a fast moving population with a diffusion coefficient, D_1 of 0.582 \pm 0.008 \mu m^2 s^{-1} over a lag time of 91 ms, D_1 of 0.519 \pm 0.008 \mu m^2 s^{-1} over a lag time of 228 ms, and D_1 of 0.378 \pm 0.009 \mu m^2 s^{-1} over a lag time of 456 ms. There is also a slow moving population with a diffusion coefficient, D_2 of 0.0214...
+ 0.0008 \mu m^2 s^{-1} over a lag time of 91 ms, D_2 of 0.0111 \pm 0.0005 \mu m^2 s^{-1} over a lag time of 228 ms, and D_2 of 0.0068 \pm 0.0003 \mu m^2 s^{-1} over a lag time of 456 ms.

Figure 4.16 Displacement histograms of streptavidin bound to biotin-X-DHPE diffusing over 40 nm polystyrene nanoparticles. Shown is the distribution of displacements at three lag times. The solid line depicts a fit of the total population to a three component Rayleigh distribution.

To determine the diffusion coefficient of streptavidin bound to biotin-X-DHPE on 40 nm nanoparticle patterned surfaces, displacement histograms (figure 4.16) were analyzed. They show three distinct populations of diffusion: a fast moving population with a diffusion coefficient, D_1 of 0.471 \pm 0.009 \mu m^2 s^{-1} over a lag time of 91 ms, D_1 of 0.421 \pm 0.009 \mu m^2 s^{-1} over a lag time of 228 ms, and D_1 of 0.338 \pm 0.007 \mu m^2 s^{-1} over a lag time of 456 ms. There is also an intermediate moving population with a diffusion coefficient, D_2 of 0.062 \pm 0.002 \mu m^2 s^{-1} over a lag time of 91 ms, D_2 of 0.056 \pm 0.002 \mu m^2 s^{-1} over a lag time of 228 ms, and D_2 of 0.050 \pm 0.002 \mu m^2 s^{-1} over a lag time of 456 ms. There is also a slow moving population with a diffusion coefficient, D_3 of 0.0097 \pm
0.0004 \mu m^2 s^{-1} over a lag time of 91 ms, D_3 of 0.0088 \pm 0.0004 \mu m^2 s^{-1} over a lag time of 228 ms, and D_3 of 0.0070 \pm 0.0003 \mu m^2 s^{-1} over a lag time of 456 ms.

Figure 4.17 Displacement histograms of streptavidin bound to biotin-X-DHPE diffusing over 100 nm polystyrene nanoparticles. Shown is the distribution of displacements at three lag times. The solid line depicts a fit of the total population to a three component Rayleigh distribution.

To determine the diffusion coefficient of streptavidin bound to biotin-X-DHPE on 100 nm nanoparticle patterned surfaces, displacement histograms (figure 4.17) were analyzed. They show three distinct populations of diffusion: a fast moving population with a diffusion coefficient, D_1 of 0.563 \pm 0.006 \mu m^2 s^{-1} over a lag time of 91 ms, D_1 of 0.515 \pm 0.006 \mu m^2 s^{-1} over a lag time of 228 ms, and D_1 of 0.414 \pm 0.007 \mu m^2 s^{-1} over a lag time of 456 ms. There is also an intermediate moving population with a diffusion coefficient, D_2 of 0.023 \pm 0.002 \mu m^2 s^{-1} over a lag time of 91 ms, D_2 of 0.013 \pm 0.002 \mu m^2 s^{-1} over a lag time of 228 ms, and D_2 of 0.015 \pm 0.003 \mu m^2 s^{-1} over a lag time of 456 ms. There is also a slow moving population with a diffusion coefficient, D_3 of 0.0027 \pm
0.0006 μm²s⁻¹ over a lag time of 91 ms, D₃ of 0.0022 ± 0.0004 μm²s⁻¹ over a lag time of 228 ms, and D₃ of 0.0016 ± 0.0002 μm²s⁻¹ over a lag time of 456 ms. 

To determine the diffusion coefficient of cholera toxin subunit B bound to GM1 on 40 nm nanoparticle patterned surfaces, displacement histograms (figure 4.18) were analyzed. They show two distinct populations of diffusion: a fast moving population with a diffusion coefficient, D₁ of 0.602 ± 0.008 μm²s⁻¹ over a lag time of 91 ms, D₁ of 0.590 ± 0.007 μm²s⁻¹ over a lag time of 228 ms, and D₁ of 0.492 ± 0.010 μm²s⁻¹ over a lag time of 456 ms. There is also a slow moving population with a diffusion coefficient, D₂ of 0.018 ± 0.002 μm²s⁻¹ over a lag time of 91 ms, D₂ of 0.015 ± 0.001 μm²s⁻¹ over a lag time of 228 ms, and D₂ of 0.015 ± 0.001 μm²s⁻¹ over a lag time of 456 ms.

Figure 4.18 Displacement histograms of cholera toxin subunit B bound to GM1 diffusing over 40 nm polystyrene nanoparticles. Shown is the distribution of displacements at three lag times. The solid line depicts a fit of the total population to a two component Rayleigh distribution.
To determine the diffusion coefficient of cholera toxin subunit B bound to GM1 on 100 nm nanoparticle patterned surfaces, displacement histograms (figure 4.19) were analyzed. They show two distinct populations of diffusion: a fast moving population with a diffusion coefficient, $D_1$ of $0.481 \pm 0.006 \, \mu m^2 s^{-1}$ over a lag time of 91 ms, $D_1$ of $0.439 \pm 0.007 \, \mu m^2 s^{-1}$ over a lag time of 228 ms, and $D_1$ of $0.350 \pm 0.006 \, \mu m^2 s^{-1}$ over a lag time of 456 ms. There is also a slow moving population with a diffusion coefficient, $D_2$ of $0.105 \pm 0.006 \, \mu m^2 s^{-1}$ over a lag time of 91 ms, $D_2$ of $0.100 \pm 0.004 \, \mu m^2 s^{-1}$ over a lag time of 228 ms, and $D_2$ of $0.087 \pm 0.002 \, \mu m^2 s^{-1}$ over a lag time of 456 ms. As can be seen in the figure, there was also a fourth component that could not be fit to a Rayleigh distribution. This component was best fit by a Gamma distribution of the form

$$P = \frac{r^{k-1} e^{-r/\theta}}{\theta^k}$$  \hspace{1cm} (4.4)
where $k = 1.76 \pm 0.01$ over a lag time of 91 ms, $k = 1.63 \pm 0.01$ over 228 ms, and $k = 1.52 \pm 0.01$ over 456 ms, and $\theta = 0.081 \pm 0.001$ over a lag time of 91 ms, $\theta = 0.128 \pm 0.002$ over 228 ms, and $\theta = 0.179 \pm 0.004$ over 456 ms. The units of this fit are not reported as it is currently unclear what physical parameters are being fit by this Gamma distribution.

Figure 4.20 The diffusion coefficients and relative contributions of CTB, streptavidin, and DHPE on 40 nm and 100 nm polystyrene nanoparticles.
<table>
<thead>
<tr>
<th></th>
<th>$D_1$ (µm²·s⁻¹)</th>
<th>$D_1$ (%)</th>
<th>$D_2$ (µm²·s⁻¹)</th>
<th>$D_2$ (%)</th>
<th>$D_3$ (µm²·s⁻¹)</th>
<th>$D_3$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LRB 40</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 ms</td>
<td>0.637 ±</td>
<td>0.0171 ±</td>
<td>-</td>
<td>-</td>
<td>0.009</td>
<td>0.0004</td>
</tr>
<tr>
<td>228 ms</td>
<td>0.546 ±</td>
<td>0.0135 ±</td>
<td>-</td>
<td>-</td>
<td>0.008</td>
<td>0.0003</td>
</tr>
<tr>
<td>456 ms</td>
<td>0.345 ±</td>
<td>0.0118 ±</td>
<td>-</td>
<td>-</td>
<td>0.008</td>
<td>0.0003</td>
</tr>
<tr>
<td><strong>LRB 100</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 ms</td>
<td>0.582 ±</td>
<td>0.0214 ±</td>
<td>-</td>
<td>-</td>
<td>0.008</td>
<td>0.0008</td>
</tr>
<tr>
<td>228 ms</td>
<td>0.519 ±</td>
<td>0.0111 ±</td>
<td>-</td>
<td>-</td>
<td>0.008</td>
<td>0.0005</td>
</tr>
<tr>
<td>456 ms</td>
<td>0.378 ±</td>
<td>0.0068 ±</td>
<td>-</td>
<td>-</td>
<td>0.009</td>
<td>0.0003</td>
</tr>
<tr>
<td><strong>Strep 40</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 ms</td>
<td>0.471 ±</td>
<td>0.062 ±</td>
<td>0.0097 ±</td>
<td>-</td>
<td>0.009</td>
<td>0.002</td>
</tr>
<tr>
<td>228 ms</td>
<td>0.421 ±</td>
<td>0.056 ±</td>
<td>0.0088 ±</td>
<td>-</td>
<td>0.009</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>456 ms</td>
<td>91 ms</td>
<td>456 ms</td>
<td>91 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Strep</strong>&lt;br&gt;100 nm</td>
<td>0.338 ± 0.007</td>
<td>0.563 ± 0.006</td>
<td>0.414 ± 0.007</td>
<td>0.602 ± 0.008</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.050 ± 0.002</td>
<td>0.023 ± 0.002</td>
<td>0.015 ± 0.003</td>
<td>0.018 ± 0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0070 ± 0.002</td>
<td>0.0027 ± 0.006</td>
<td>0.0016 ± 0.003</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CTB 40</strong>&lt;br&gt;nm</td>
<td>0.492 ± 0.010</td>
<td>0.493 ± 0.007</td>
<td>0.481 ± 0.006</td>
<td>0.493 ± 0.007</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.001</td>
<td>0.004</td>
<td>0.015 ± 0.001</td>
<td>0.105 ± 0.006</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CTB 100</strong>&lt;br&gt;nm</td>
<td>0.481 ± 0.006</td>
<td>0.493 ± 0.007</td>
<td>0.481 ± 0.006</td>
<td>0.493 ± 0.007</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.105 ± 0.006</td>
<td>0.100 ± 0.004</td>
<td>0.105 ± 0.006</td>
<td>0.100 ± 0.004</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>456</td>
<td>0.350±</td>
<td>0.087±</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ms</td>
<td>0.006</td>
<td>0.002</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 4.1 The diffusion coefficients of membrane components over three lag times and two nanoparticle sizes. The error is represented as the SEM of the fit.*

The mean MSD over all three lag times are presented in figure 4.20, along with the relative contribution of each MSD to the entire population. A difference in MSD should show any long-term interactions between diffusing particles and their environments or a large number of transient interactions. Transient interactions with the sparse regions of curvature are unlikely to effect the MSD, and do not show up in this plot.

The individual MSD for the entire population of particles tracked for LRB-DHPE on 40 nm and 100 nm nanoparticle patterned surfaces are presented in figure 4.21. Those that were tracked for streptavidin bound to biotin-X-DHPE on 40 nm and 100 nm nanoparticle patterned surfaces are presented in figure 4.22. Also, the particles that were tracked for cholera toxin subunit B bound to GM1 on 40 nm and 100 nm nanoparticle patterned surfaces are presented in figure 4.23.
Figure 4.6.1 The individual MSD plots of LRB-DHPE over 40nm (left) and 100 nm (right) nanoparticles.

Figure 4.2.2 The individual MSD plots of streptavidin over 40nm (left) and 100 nm (right) nanoparticles.

Figure 4.7. The individual MSD plots of CTB over 40nm (left) and 100 nm (right) nanoparticles. Little data was collected with 40 nm particles, which resulted in fewer tracks to analyze.
The turning angle histograms for the entire population of particles tracked for LRB-DHPE on 40 nm and 100 nm nanoparticle patterned surfaces are presented in figure 4.24. The turning angle histograms for those that were tracked for streptavidin bound to biotin-X-DHPE on 40 nm and 100 nm nanoparticle patterned surfaces are presented in figure 4.25. The turning angle histograms for those that were tracked for cholera toxin subunit B bound to GM1 on 40 nm and 100 nm nanoparticle patterned surfaces are presented in figure 4.26.

As the turning angle histograms showed an unexplained trend toward a \( \theta \) of zero for many of the sampled populations, which would seem to indicate directed motion in a system which has no provision for directed motion, turning angles were also calculated over multiple displacements. These are shown in figure 4.27 for CTB diffusing over 100 nm nanoparticles. The possibility of seeing what could be interpreted as directed motion in the turning angle plots without seeing evidence of this in the MSD plots is discussed below.

Figure 4.24 The turning angle histograms for LRB-DHPE over one step for 40 nm (left) and 100 nm (right) nanoparticles.
Figure 4.25 The turning angle histograms for streptavidin over one step for 40 nm (left) and 100 nm (right) nanoparticles.

Figure 4.26 The turning angle histograms for CTB over one step for 40 nm (left) and 100 nm (right) nanoparticles.
Figure 4.27 The turning angle histograms for CTB over 100 nm nanoparticles are plotted over a varying number of frames. Note that the peak at 0 disappears almost immediately and the peak at π disappears more gradually.
Discussion

At both high (~500 nM) and low (~5 nM) concentrations, CTB demonstrates an apparent avoidance of curvature. This avoidance is seen in bulk experiments, such as colocalization, as well as in single particle tracking experiments. A new single particle tracking analytical method using turning angles also shows evidence of this curvature avoidance.

The depiction of diffusing populations by displacement histograms shows several distinct populations. For each of the sampled systems, there is a more rapidly diffusing population and one or more population that is moving more slowly, as seen in figure 4.20. The very quickly diffusing population in each of these histograms is consistent with a single membrane anchor. The diffusion coefficient of the fastest component is consistent with DHPE diffusion, which consists of a fluorophore attached to a single lipid. In the case of the membrane associated proteins, CTB and streptavidin, this fast component is consistent with a protein bound to a single membrane lipid. As the dissociation constant of CTB, at 540 nM, is two orders of magnitude above the concentration used in these experiments (~5 nM), it is expected that the protein bound to the GM1 on the bilayer surface will consist largely of monomers. These CTB monomers will bind a single GM1 and will diffuse at a rate very similar to a single diffusing RLB-DHPE lipid.

The measured length of a diffusing particle’s displacements is affected by the amount of time that it may remain immobile between steps. The distribution of these immobile waiting times is apparent in the distribution of displacement lengths measured
for moving particles. The gamma distribution seen in the displacement histograms of CTB in the presence of 100 nm nanoparticle (figure 4.19) may give some insight into the interactions between CTB and the regions of curvature. If the lateral diffusion of particles on the bilayer is described by a continuous-time random walk, interactions with the environment may shift the distribution of waiting time between jumps and change the distribution of observed displacement lengths. Unlike a Wiener process, where the waiting times are exponentially distributed, a different distribution of waiting times could contribute to the displacement distributions we observed in this work. This hypothesis has not been previously described in this context and deserves further analysis.

As seen in the mean squared displacement plots of individual diffusing particles, there is a large amount of variation in the motion of individual particles. A large amount of variation is not unusual and is a logical result of each individual particle experiencing a slightly different environment with regard to lipid bilayer defects and encounters with nanoparticles. Even particles that have multiple encounters with nanoparticles will have a distinctly different MSD plot than particles that only encounter one nanoparticle.

The evaluation of turning angles of diffusing particles represents a relatively underutilized technique, with limited use in fields such as animal dispersal. Recently, it has begun to see some utility as “track straightness” in particle tracking. As such, we were excited to determine the utility of the turning angle in the elucidation of transient interactions. The turning angle histograms do, in fact, display some interesting trends and demonstrate that this analytical method will be useful for many single particle tracking experiments.
One of the first attributes of the turning angle plots that is noticed is the trend toward a $\theta$ of zero in many of the plots. In the methods section above, this was presumed to be evidence of directed motion or actively facilitated diffusion. Since there is assumed to be no mechanism for this sort of diffusion on the nanoparticle patterned supported lipid bilayer, this trend would appear to be very troubling. Any interpretations of the trend as a reflection of particle localization or tracking errors also fall short of adequately explaining it. These sorts of localization and tracking errors would be expected to be found at $\theta = \pi$, or at both $\theta = 0$ and $\pi$, if they are the result of jitter in the localization of particles. The most likely explanation of this trend is the mobility of the lipids in the bilayer and any proteins attached to them being influenced by the structure of the glass that is supporting the lipid bilayer. This has been previously observed\textsuperscript{52,55,56}, even to the point of diffusing lipids following the atomic-scale structure of their supporting substrate.\textsuperscript{53,54} As these structures can be extremely minute, it is unlikely that interactions with such structures would last long enough to influence the MSD of any diffusing lipids significantly, but may cause a noticeable increase in the number of “forward”, or $\theta = 0$ steps. The borosilicate glass used in this work was extensively cleaned as described in Chapter 2 and assayed for smoothness by AFM as described in chapter 3, so it is known that it contains no features larger than a few nanometers in height or more than a few score nanometers in lateral extent, so the direction of these steps would be randomly distributed, leading to no noticeable drift or direction of motion being observed. Indeed, when the turning angles are calculated over longer lag times, as was shown in figure 4.25, this $\theta = 0$ trend disappears, indicating that such directed motion did not occur for many consecutive measurements.
The other prominent feature in the turning angle histograms is the peak at $\theta = \pi$, which suggests confinement or obstruction of diffusing particles. This peak is notably absent in the data of LRB-DHPE, which is consistent with the expected lack of interaction between DHPE and regions of membrane curvature. It can be inferred that because the diffusing LRB-DHPE molecules are so small relative to the induced curvature that they are not obstructed by the curvature and pass right along the curvature without being reflected off of it. There is also no $\theta = \pi$ peak in the data of streptavidin diffusing on lipid bilayers patterned with 100 nm nanoparticles. There is, however, a $\theta = \pi$ peak when streptavidin diffuses over 40 nm nanoparticles. The turning angle plots of cholera toxin subunit B bound to GM1 display a $\theta = \pi$ peak on lipid bilayers patterned with 100 nm nanoparticles, but no corresponding peak on lipid bilayers patterned with 40 nm nanoparticles. Interestingly, in both of the systems in which a $\theta = \pi$ peak is seen, the more slowly diffusing population has a diffusion coefficient that is considerably higher (figure 4.21). The $D_2$ values for streptavidin represent a difference of a factor of two, while for CTB the difference is an entire order of magnitude. This implies that an avoidance of, or reflection off of, the regions of membrane curvature is allowing these populations to avoid further interaction with the nanoparticles and thus not suffer a reduction in the rate of diffusion.

We were ultimately able to observe the sensing of curvature by both cholera toxin subunit B and streptavidin with single molecule tracking experiments, while previous studies were not able to, because of the specific properties of our nanoparticle patterned supported lipid bilayer. In our system, regions of membrane curvature are scarce enough that interactions with these regions do not occur often, while previously employed model
systems that use liposomes or wavy substrates place diffusing proteins in extended regions of curvature with no opportunity to escape. In past work where CTB preferred negative curvature\textsuperscript{95}, the reported results may have only been demonstrating CTB showing a greater avoidance for positive curvature than negative curvature. This may not be apparent from the behavior of single molecules, while cooperative effects between large numbers of CTB molecules may be more easily realized in regions of negative membrane curvature than positive membrane curvature. As the experiments used concentrations of CTB well below the subunit-pentamer $K_d$, the results may also be conflating the behavior of the pentameric CTB with that of the CTB monomers.

Secondly, the use of nanoparticles in our supported lipid bilayer to induce curvature allowed us to make regions of much higher curvature than can be realized in previously use systems. The curvature induced by the undulations in the wavy glass model is much lower than in our system, which would allow us to see interactions with levels of curvature that could not be observed in their system. It seems that the very high level of membrane curvature that we could create with 100 nm nanoparticles was enough to allow us to observe interactions between individual CTB molecules and membrane curvature. The lack of interaction between CTB and 40 nm nanoparticles is curious, but if we apply the findings from Chapter 3 that the lipid bilayer forms a tent-like structure instead of tightly wrapping around the nanoparticle, then the curvature around 40 nm nanoparticles may not be as great as we had previously assumed. A more thorough investigation of the nanoparticle patterned supported lipid bilayer by AFM would help to elucidate this.

At the low concentrations needed for single molecule experiments, CTB is very likely to exist predominantly as a monomer. As seen in the results of the single particle
tracking experiments, and corroborated by bulk colocalization experiments, CTB avoids regions of curvature in this monomeric form. Analysis of the single particle tracking experimental data through the use of turning angles proved useful in coming to these conclusions.
Chapter 5: Single molecule tracking of \textit{p}-glycoprotein in live cells reveals dynamic heterogeneity

\textbf{Introduction}

In response to chemotherapy, tumor cells often up-regulate multidrug resistance transporters to survive treatment. \textit{P}-glycoprotein (PGP) is an ATP dependent, drug transporter present in many forms of untreatable cancer.\textsuperscript{96,97} PGP contains 12 transmembrane domains and is a large (170 kDa) protein that binds lipophilic drug molecules from the plasma membrane to export them from the cell, effectively reducing the intracellular concentration of drug.\textsuperscript{96} The presence of PGP can be predictive of the outcome of chemotherapy treatment, with increased amounts correlating to poor prognosis.\textsuperscript{98-100}

PGP interacts with many proteins and lipids on the cell surface, including the actin cytoskeleton and lipid rafts.\textsuperscript{101,102} PGP has been shown to be enriched in lipid rafts and intermediate-density rafts.\textsuperscript{102-105} Specifically, cholesterol has been shown to modulate the membrane transport activity of PGP, increasing the efflux activity six-fold in a reconstituted membrane system.\textsuperscript{106,107} In a model cellular system transiently expressing PGP, the loss of cholesterol abolishes PGP efflux activity.\textsuperscript{108} Others have reported that the presence of cholesterol affects both the ability of chemotherapy drugs to bind to PGP.
and transport rate of PGP. Research suggests that the efflux activity of PGP is highly sensitive to the local environment. When multidrug resistant leukemia cells are treated with an antibody to CD19, a protein that interacts with PGP, PGP translocates from lipid rafts and its activity is halted, allowing the cells to become chemosensitive. The local environment clearly affects PGP activity.

In addition to known PGP interactions, the heterogeneous structure of the plasma membrane likely contributes to the anomalous diffusion of membrane proteins. The cytoskeleton has been shown to create corrals and lipid rafts can be related to transient confinement zones. Single molecule tracking allows for a direct visualization of these phenomena in regards to membrane proteins moving on the plasma membrane.

To better understand the transient interactions of PGP on the cell surface, the dynamics of PGP were assessed using single particle tracking (SPT) techniques. Single molecule tracking of drug transporters in the plasma membrane of live cells allows for the direct measurement of transient and heterogeneous interactions that have the potential for altering the efficiency of drug transport. Techniques that measure the heterogeneity of the system will lead to a better understanding of PGP interactions on the cell surface. In this work, we describe an assay for measuring the dynamics of single PGP molecules using SPT techniques in a model cell system, MES-SA cells. We show here that PGP-EGFP can be imaged and tracked in live cells to better understand PGP interactions on the plasma membrane. We find that when PGP-EGFP is expressed on the cell surface, rhodamine, a substrate, is effectively removed from the cell, showing that PGP-EGFP is functional in MES-SA cells. Using SPT, PGP is observed to be quite mobile for a large
transmembrane protein but exhibits confined diffusion over long time periods (> 0.7 s). At short times, PGP moves freely with a diffusion coefficient, $D$, of 0.2 $\mu$m$^2$/s. To our knowledge, this is the first report of single molecule of PGP dynamics.

**Methods**

*Cell culture and transfection*

MES-SA and MES-SA/Dx5 cells were cultured as described in Chapter 2. For imaging, cells were plated on coverslips that were pretreated for 15 m with a 0.01% poly-L-lysine (Sigma Aldrich) solution. Transfection with PGP-EGFP was carried out on coverslip-plated MES-SA cells using Lipofectamine 2000 (Invitrogen, Carlsbad, CA) and 1.5 $\mu$g of DNA plasmid per coverslip in serum-free OptiMEM (Gibco). The plasmid was obtained from Dr. Michael Gottesman $^{111}$. Cells were incubated in the transfection solution for six hours at 37 °C and imaged between 24 and 48 hours post-transfection. All cell culture reagents were purchased from Invitrogen.

![Figure 5.1](image.png)

*Figure 5.1 PGP-EGFP particles are imaged on the plasma membrane of a cell (A). The enlarged region (B) shows individual particles with more detail. The scale bar on both images represents 2 $\mu$m.*
**Total Internal Reflection Fluorescence Microscopy (TIRFM)**

Cells expressing PGP-EGFP were imaged using objective-style TIRFM. TIRFM allows only 100-200 nm of the cell in contact with the glass coverslip to be excited. A Nikon Eclipse Ti-U microscope equipped with a TIRF launch, a 60x oil immersion TIRFM objective (NA 1.49), and 2.5x magnification lens was used for a final magnification of 150x. A 491 nm laser (Cobolt Calypso DPSS) provided excitation of EGFP. Fluorescence emission was captured through a dichroic that passes 500-545 nm made specifically for TIRFM application (Chroma Technologies). Afterwards, fluorescence emission passed through a 525/50 nm filter (Chroma Technologies) onto a back-illuminated EMCCD camera (Andor iXon+ 897). Imaging was performed at 25 °C. Image series were captured as a stream of 100 ms exposures using µManager (http://www.micro-manager.org). Movies were 500 frames in length. A typical image that was used for single particle tracking is shown in figure 5.1.

**Figure 5.2** PGP actively removes rhodamine 6G (R6G) from cells. Cells expressing PGP-EGFP (green) display low levels of R6G uptake, even after 30 min in 1 µM R6G. Quantitation shows that significantly more R6G (red) accumulates in cells not expressing PGP-EGFP. Error is shown as the SEM.
Single Particle Tracking

The series of images captured using TIRFM was analyzed in ImageJ (http://imagej.nih.gov/ij/) using the particle tracking plug-in from MOSAIC. Particle detection parameters were as follows: the kernel radius was 3.0 pixels, the cutoff radius was 0.0 pixels, and the top one percentile of bright pixels was considered as particle candidates. Tracked particles were allowed to travel a maximum of six pixels (0.642 µm) between consecutive frames to be considered as linked particles.

The position matrices representing particle tracks produced in ImageJ were then transferred into IDL (Exelis) in order to use freely available software to analyze tracks (http://www.physics.emory.edu/~weeks/idl/tracking.html) as has been done previously by others to track membrane protein dynamics. The displacement of each particle at

![Figure 5.3](image)

*Figure 5.3 The displacement of particles is measured for various time intervals, revealing diverging populations of diffusing particles. Displacement histograms are fit to (1) (solid red line) for 0.2 s (A), 0.5 s (B), and 1.0 s (C). A simulated curve of (1) with a single population diffusing at 0.2 µm²/s is provided (dashed black line) for reference.*
varying time lags, $t$, were calculated. Mean square displacements (MSD) was calculated as described $^{38}$.

Displacement histograms and MSD values were plotted and fit in Prism 5 (GraphPad Software, La Jolla, California USA). Displacement histograms were fit to (5.1) to determine the diffusion coefficients, $D_1$ and $D_2$, as well as the amplitude of fast and slow molecules in the population, $A_1$ and $A_2$ as previously described $^{113,115}$.

$$y(r,t) = r \left[ A_1 \exp \left( -\frac{r^2}{4D_1t} \right) + A_2 \exp \left( -\frac{r^2}{4D_2t} \right) \right]$$  \hspace{1cm} (5.1)

Mean square displacements (5.2) were fit with a segmental linear regression to distinguish the two diffusing populations of PGP. The segmental linear regression was carried out as provided in Prism 5 and was allowed to find the transition between the two linear sections by the least squares method.

$$\langle r^2 \rangle = 4Dt$$  \hspace{1cm} (5.2)

**Rhodamine Efflux Studies**

Efflux activity of PGP-EGFP was tested in MES-SA cells as previously described $^{115}$. Transfected cells expressing PGP-EGFP were incubated in a 1 μM solution of rhodamine 6G (R6G) in phosphate buffered saline (PBS) for five minutes prior to confocal imaging. The R6G solution was left on the cells during imaging and confocal images of the cells were taken using both the EGFP and R6G fluorescent emission channels. Untransfected MES-SA cells were imaged in the same manner as negative efflux controls.
**Confocal Imaging**

Confocal imaging of R6G efflux was performed on an Olympus FluoView FV1000 confocal microscope using a 100x oil immersion objective (NA 1.40). Excitation was at 488 nm and 515 nm and fluorescence emission was collected from 495 nm to 510 nm and from 530 nm to 560 nm. The surface of live cells nearest the coverslip were imaged at 25 °C.

**Results**

*PGP-EGFP removes rhodamine from the plasma membrane of live MES-SA cells.*

In cells transfected with PGP-EGFP, R6G is effectively excluded from the plasma membrane. As shown in Fig. 5.2, cells that are expressing PGP-EGFP have very little R6G fluorescence. Cells that are not expressing PGP-EGFP, identified by the absence of EGFP fluorescence, have significantly more R6G fluorescence. This shows that PGP-EGFP is functional in MES-SA cells.

*Figure 5.4 Two distinct populations of diffusion are present. A segmented linear regression shows a transition from fast diffusion to slow, or confined, diffusion at 0.7 s. Error is shown as the SEM.*
PGP-EGFP is mobile at short times and confined at long times (>0.7 s)

Over short periods of time, it appears that PGP-EGFP on the surface of cells diffuses at a rate faster than when observed over longer period of times. Fig. 5.4 suggests that there is a transition between these rates at approximately 0.7 s. The displacement histograms in Fig. 4 show what appear to be two diverging populations of diffusing particles, one slowly diffusing population and another that is diffusing even more slowly. This suggests PGP-EGFP molecules are fully or partially confined as seen in Fig. 5.5. The dashed lines in Fig. 5.4 show how a population of particles diffusing at 0.2 \( \mu m^2/s \) would be distributed at various time points. The growing difference between these two curves further suggests a population of slowly moving or confined particles.

Discussion

Functional PGP-EGFP can be transiently expressed in cells that are devoid of PGP (MES-SA). PGP-EGFP effectively removes R6G from the cell membrane and cells that do not express PGP retain the red fluorescent rhodamine, as shown in Fig. 5.2 using transient transfection techniques. Cells express a diverse amount of PGP-EGFP, which allows us to study PGP-EGFP on a single molecule level amenable to particle tracking. Cells with low levels of PGP are imaged and molecules are tracked to reveal heterogeneous behavior; the heterogeneity is most obvious near the transition time from freely diffusing to confined motion (0.7 s). PGP freely diffuses on short time and length scales but is confined at longer times, as seen in the MSD in Fig. 5.4. When the step size of a molecule is measured at a very short time lag, most molecules are freely diffusing.
However, when the distance traveled over a long time lag is measured, most are confined. The heterogeneity of the system lies in the intermediate time scales. The histograms in Fig. 5.4 reveal a confined motion not observed in MSD at long times.

Future investigations using two-color single particle tracking techniques could reveal what PGP is interacting with. Interactions with the actin cytoskeleton and lipid microdomains are likely influencing the dynamics of PGP and could be a factor in how readily PGP transports chemotherapy drugs.

**Conclusions**

The dynamics of PGP suggest that diverse interactions are occurring on the surface of cells and single particle tracking allows for the direct observation of such interactions. Further investigations on the effects of drugs that activate or inhibit PGP could lead to a better understanding of how this complex membrane protein can be regulated to reduce the impact of multi-drug resistance in cancer treatment.
Chapter 6: Future Work

While the utility of the nanoparticle patterned supported lipid bilayer developed in this lab has been demonstrated in this and previous works, there are still opportunities to further develop the model system and there are plenty of new targets to apply the model system toward. In future experiments, it will become more important to tailor the lipid composition of the bilayer to match the conditions to which the protein of interest is natively involved. Investigating different lipid compositions will also give the opportunity to study lipid sorting at curvatures much greater than previously possible.\textsuperscript{5,23}

Further characterization of our model system is still required, as well. High resolution AFM of the bilayer in the vicinity of the nanoparticles would greatly help with our understanding of the behavior of diffusing lipids and proteins in that region.

The use of fused silica nanoparticles in place of polystyrene also presents the opportunity to make consistently formed nanoparticles without the uncertainty in fluidity associated with the use of polystyrene.\textsuperscript{116}
Bibliography


(49) Gilmore, S. F.; Nanduri, H.; Parikh, A. N. *PLOS ONE.* 2012 ed. 2011, p e28517.


(65) Cherry, R. J. *Biochimica et Biophysica Acta (BBA) - Reviews on Biomembranes.* December 1979, pp 289–327.
(66) Roiter, Y.; Ornatska, M.; Rammohan, A. R.; Balakrishnan, J.; Heine, D. R.;
(67) Roiter, Y.; Ornatska, M.; Rammohan, A. R.; Balakrishnan, J.; Heine, D. R.;
(69) Schultz, S. G.; Yu-Tu, L. *Biochimica et Biophysica Acta (BBA) -
(71) Negoda, A.; Kim, K.-J.; Crandall, E. D.; Worden, R. M. *Biochim Biophys Acta.*
September 2013, pp 2215–2222.
(73) Day, C. A.; Kenworthy, A. K. *PLOS ONE*. Public Library of Science April 12,
2012, p e34923.
(80) Turnbull, W. B.; Precious, B. L.; Homans, S. W. *Journal of the American
(87) Weigel, A. V.; Tamkun, M. M.; Krapf, D. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology*


(98) Suresh V Ambudkar; Saibal Dey; Christine A Hrycyna; Muralidhara Ramachandra; Ira Pastan; Gottesman, M. M. *dx.doi.org.* Annual Reviews 4139 El Camino Way, P.O. Box 10139, Palo Alto, CA 94303-0139, USA November 28, 2003, pp 361–398.


(107) Troost, J.; Albermann, N.; Emil Haefeli, W.; Weiss, J. *Biochem Biophys Res*


Appendix A: Tracking and analysis software source code

The source code written for this work is presented in this appendix. Instead of presenting this code as only unformatted text, the “Publish” function of MATLAB was used to create a more easily read representation of it. All of the comments and headers in this section are automatically generated from the source code itself and were not edited before preparing for publication in this work. To preserve long lines in some of the source code, the font size of the published work is scaled per function to allow all text to fit in the printed pages of this dissertation.

As modifications were made to Blair and Dufresne’s MATLAB implementation of the IDL tracking software by Grier, Crocker, and Weeks, it is included at the end of this appendix.
% A cell array of jobs to run: {{'data dir 1' 'analysis dir 1'}}
% -- These are loaded on the HPC, so ensure that the files exist on the HPC
% -- And that these paths point to them.
jobsToRun = {...
  {'/home/pcheney/Documents/Data/0bestdata/ctb/40nm/'  '/home/pcheney/Documents/Analysis/0bestdata/ctb/40nm/'},
  {'/home/pcheney/Documents/Data/0bestdata/ctb/100nm/'  '/home/pcheney/Documents/Analysis/0bestdata/ctb/100nm/'},
  {'/home/pcheney/Documents/Data/0bestdata/lissamine/40nm/'  '/home/pcheney/Documents/Analysis/0bestdata/lissamine/40nm/'},
  {'/home/pcheney/Documents/Data/0bestdata/lissamine/100nm/'  '/home/pcheney/Documents/Analysis/0bestdata/lissamine/100nm/'},
};

% If a pool of the correct size is open, use it. Otherwise close the old
% pool (if it exists) and start a new one of the right size.
currentPoolDetails = gcp('nocreate');
if ne(length(currentPoolDetails),0)
currentPoolSize = currentPoolDetails.NumWorkers;
if lt(currentPoolSize, length(jobsToRun))
delete(gcp);
  parpool(length(jobsToRun));
end
else
  parpool(length(jobsToRun));
end
spmd
  loadAndTrack3parallel(jobsToRun{1,labindex}{1,1},jobsToRun{1,labindex}{1,2},1);
end

Published with MATLAB® R2015a
function [ imageBP ] = bandpass( image, lower, upper )
%BANDPASS Applies a bandpass filter to an image
% This was inspired by ImageJ’s FFT_Filter.java – ok, actually it wasn’t.
% Consider autoscaling and saturating like their filter does. It will make
% peak detection by thresholding easier at the expense of confusing results
% on empty frames.

% The order of the finite impulse response filter
order = 6;

upperscale = upper / size(image,1);
lowerscale = lower / size(image,1);

filter1D = fir1(order, [lowerscale upperscale]);
filter2D = ftrans2(filter1D);

imageBP = imfilter(image,filter2D);
end
function bgSubMovie = bgSubByBpass( movie )

%bgSubByBpass -- subtract background by bandpassing a mean intensity
%projection of a movie and subtracting it from a bandpassed movie.
%
% INPUT VARIABLES:
%
% OUTPUT:
%
% Upper and lower limits of the bandpass filter
% Passing objects between 1 and 7 seems to work well for diffraction
% limited spots. Adjust as needed.
lower = 3;
upper = 7;

% Count the number of frames in the supplied movie
numberframes = size(movie,3);

% Make a mean intensity projection (MIP) of the movie
movieMIP = mean(movie,3);

% Bandpass filter the MIP
movieMIPbpassed = bandpass(movieMIP,lower,upper);

% Bandpass the movie
movieBpassed = bpassMovie(movie,lower,upper);

% Subtract the bandpassed MIP from the bandpassed movie
bgSubMovie = bsxfun(@minus,movieBpassed,movieMIPbpassed);

end
function bandpassedMovie = bpassMovie( movie , lower , upper )

% bpassmovie -- load a movie and bandpass it frame by frame
% %
% % INPUT VARIABLES:
% %
% % OUTPUT:
% %

numberframes = size(movie,3);
bandpassedMovie = zeros(size(movie,1),size(movie,2),size(movie,3));

if isParallelPossible
    parfor frame = 1:numberframes
        % if gt(gpuDeviceCount,0)
        %     bandpassedMovie(:,:,frame) = bpassGPU(movie(:,:,frame),lower,upper);
        % else
        %     bandpassedMovie(:,:,frame) = bandpass(movie(:,:,frame),lower,upper);
        % end
    end
else
    for frame = 1:numberframes
        % if gt(gpuDeviceCount,0)
        %     bandpassedMovie(:,:,frame) = bpassGPU(movie(:,:,frame),lower,upper);
        % else
        %     bandpassedMovie(:,:,frame) = bandpass(movie(:,:,frame),lower,upper);
        % end
    end
end

Published with MATLAB® R2015a
function [ ] = combineDxInt( pathToSearch, genFigs )
%combineDxInt makes big combined arrays with particle intensity of
%found arrays
% Finds all .mat files in all subdirectories of the supplied
% path, looks for arrays found in them and combines them into one big array
% - currently tuned for dx, msd, turnangle results.
% list of arrays of interest and their associated files... wow.
field1 = 'arrayname';
field2 = 'filename';
value1 = {...
'traj8Col','traj8ColMobile','traj8ColStuck'...
};
value2 = {...
'lipidTraj','lipidTraj','lipidTraj'...
};
arrays = struct(field1,value1,field2,value2);
% find all directories in 'pathToSearch'
dirList = dir(pathToSearch);
% trim out '.', '..', and non-directories
if gt(size(dirList,1),2)
% '.' and '..' are always the first two rows
dirList = dirList(3:size(dirList,1),1);
% only keep if isdir is 1
dirList = dirList(eq([dirList.isdir],1),1);
end
% main loop, repeat for each array of interest
for arraysIndex = 1:size(arrays,2)
fprintf('Processing array %s
',arrays(arraysIndex).arrayname);
% run through the subdirectories
for dirIndex = 1:size(dirList,1)
% does the file exist?
filename = fullfile(pathToSearch,dirList(dirIndex).name,strcat(arrays(arraysIndex).filename, '.mat'));
if eq(size(dir(filename),1),1)
load(filename,arrays(arraysIndex).arrayname);
% if there is no big array for this array started, start it now
if eq(exist('bigarray','var'),0)
bigarray = eval(arrays(arraysIndex).arrayname);
else
% if the bigarray exists, concatenate
bigarray = cat(1,bigarray,eval(arrays(arraysIndex).arrayname));
end
fprintf('+');
else
fprintf('-');
end
end
% I don't like this, but I can't find another way to preserve variable
% names.
eval(sprintf('%s = bigarray;',arrays(arraysIndex).arrayname));
% Here is where I recreate functionality that should already exist. Why
% doesn't ' -append' create a file if it doesn't already exist? Stupid.
if eq(exist(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '.mat')),'file'),0)
save(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '.mat'))),arrays(arraysIndex).arrayname);
else
save(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '.mat'))),arrays(arraysIndex).arrayname, '-append');
end
% Generate figures if genFigs is set
if eq(genFigs,1)
fprintf( ' generating figure'
);
% Calculate the displacements and retain the intensity measurements
dxInt = dxIntensity(bigarray,1,2);
dxIntStripped = stripNaN(dxInt);
plotScatter(dxIntStripped,0,struct('dirName', pathToSearch, 'prefix', arrays(arraysIndex).arrayname));

% Only histograms and turning angles at the moment
if gt(sum(ismember(arrays(arraysIndex).arrayname, 'dx')), 0)
    plotHistograms(bigarray, 0, struct('dirName', pathToSearch, 'prefix', arrays(arraysIndex).arrayname));
elseif gt(sum(ismember(arrays(arraysIndex).arrayname, 'turn')), 0)
    plotTurnAngles(bigarray, 0, struct('dirName', pathToSearch, 'prefix', arrays(arraysIndex).arrayname));
end
end

clear(strcat(arrays(arraysIndex).arrayname), 'bigarray');
fprintf('
');
end
function report = combineResults( pathToSearch, genFigs )
%combineResults makes big combined arrays of found arrays
% path, looks for arrays found in them and combines them into one big array
% currently tuned for dx, msd, turnangle results.
% list of arrays of interest and their associated files... wow.
field1 = 'arrayname';
field2 = 'filename';
value1 = {
    'dx01_all_all', 'dx02_all_all', 'dx05_all_all', 'dx10_all_all', ...
    'dx01_all_coloc', 'dx02_all_coloc', 'dx05_all_coloc', 'dx10_all_coloc', ...
    'dx01_all_noncoloc', 'dx02_all_noncoloc', 'dx05_all_noncoloc', 'dx10_all_noncoloc', ...
    'dx01_mobile_all', 'dx02_mobile_all', 'dx05_mobile_all', 'dx10_mobile_all', ...
    'dx01_mobile_coloc', 'dx02_mobile_coloc', 'dx05_mobile_coloc', 'dx10_mobile_coloc', ...
    'dx01_mobile_noncoloc', 'dx02_mobile_noncoloc', 'dx05_mobile_noncoloc', 'dx10_mobile_noncoloc', ...
    'dx01_stuck_all', 'dx02_stuck_all', 'dx05_stuck_all', 'dx10_stuck_all', ...
    'dx01_stuck_coloc', 'dx02_stuck_coloc', 'dx05_stuck_coloc', 'dx10_stuck_coloc', ...
    'dx01_stuck_noncoloc', 'dx02_stuck_noncoloc', 'dx05_stuck_noncoloc', 'dx10_stuck_noncoloc', ...
    'turn_all_all', 'turn_mobile_all', 'turn_stuck_all', ...
    'turn_all_coloc', 'turn_mobile_coloc', 'turn_stuck_coloc', ...
    'turn_all_noncoloc', 'turn_mobile_noncoloc', 'turn_stuck_noncoloc', ...
    'msd_all_all', 'msd_mobile_all', 'msd_stuck_all', ...
    'msd_all_coloc', 'msd_mobile_coloc', 'msd_stuck_coloc', ...
    'msd_all_noncoloc', 'msd_mobile_noncoloc', 'msd_stuck_noncoloc', ...
};
value2 = {
    'dx_all_all', 'dx_all_all', 'dx_all_all', 'dx_all_all', ...
    'dx_all_coloc', 'dx_all_coloc', 'dx_all_coloc', 'dx_all_coloc', ...
    'dx_all_noncoloc', 'dx_all_noncoloc', 'dx_all_noncoloc', 'dx_all_noncoloc', ...
    'dx_mobile_all', 'dx_mobile_all', 'dx_mobile_all', 'dx_mobile_all', ...
    'dx_mobile_coloc', 'dx_mobile_coloc', 'dx_mobile_coloc', 'dx_mobile_coloc', ...
    'dx_mobile_noncoloc', 'dx_mobile_noncoloc', 'dx_mobile_noncoloc', 'dx_mobile_noncoloc', ...
    'dx_stuck_all', 'dx_stuck_all', 'dx_stuck_all', 'dx_stuck_all', ...
    'dx_stuck_coloc', 'dx_stuck_coloc', 'dx_stuck_coloc', 'dx_stuck_coloc', ...
    'dx_stuck_noncoloc', 'dx_stuck_noncoloc', 'dx_stuck_noncoloc', 'dx_stuck_noncoloc', ...
    'turn_all', 'turn_all', 'turn_all', 'turn_all', ...
    'turn_coloc', 'turn_coloc', 'turn_coloc', 'turn_coloc', ...
    'turn_noncoloc', 'turn_noncoloc', 'turn_noncoloc', 'turn_noncoloc', ...
    'msd_all', 'msd_all', 'msd_all', 'msd_all', ...
    'msd_coloc', 'msd_coloc', 'msd_coloc', 'msd_coloc', ...
    'msd_noncoloc', 'msd_noncoloc', 'msd_noncoloc', 'msd_noncoloc', ...
};
arrays = struct(field1,value1,field2,value2);
% find all directories in 'pathToSearch'
dirList = dir(pathToSearch);
% trim out '.', '..', and non-directories
if gt(size(dirList,1),2)
    % '.' and '..' are always the first two rows
    dirList = dirList(3:size(dirList,1),1);
end
% main loop, repeat for each array of interest
for arraysIndex = 1:size(arrays,2)
    % run through the subdirectories
    for dirIndex = 1:size(dirList,1)
        % does the file exist?
        filename = fullfile(pathToSearch,dirList(dirIndex).name,strcat(arrays(arraysIndex).filename, '.mat'));
        % if so, load the array contained within
        if eq(size(filename,1),1)
            load(filename,arrays(arraysIndex).arrayname);
        end
        % if there is no big array for this array started, start it now
        if eqexist('bigarray', 'var', 0)
            bigarray = eval(arrays(arraysIndex).arrayname);
        else
            % if the bigarray exists, concatenate
            bigarray = cat(1,bigarray,eval(arrays(arraysIndex).arrayname));
        end
    end
end
fprintf(‘+’);
else
fprintf(‘-’);
end
end
clear(sprintf('%s',arrays(arraysIndex).arrayname));

% I don't like this, but I can't find another way to preserve variable
% names.
eval(sprintf('%s = bigarray;',arrays(arraysIndex).arrayname));

% Here is where I recreate functionality that should already exist. Why
% doesn't '-append' create a file if it doesn't already exist? Stupid.
if eq(exist(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '.mat')),'file'),0)
    save(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '.mat')),arrays(arraysIndex).arrayname);
else
    save(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '.mat')),arrays(arraysIndex).arrayname, '-append');
end

% Generate figures if genFigs is set
if eq(genFigs,1)
    fprintf(‘ generating figure’);
    if gt(sum(ismember(arrays(arraysIndex).arrayname, 'dx')),0)
        plotHistograms(bigarray,0,struct( ‘dirName’,pathToSearch,’prefix’,arrays(arraysIndex).arrayname));
    elseif gt(sum(ismember(arrays(arraysIndex).arrayname, 'turn ')),0)
        plotTurnAngles(bigarray,0,struct( ‘dirName’,pathToSearch,’prefix’,arrays(arraysIndex).arrayname));
    end
    clear(strcat(arrays(arraysIndex).arrayname), 'bigarray');
    fprintf(‘\n’);
end
end

Published with MATLAB® R2015a
function [ ] = combineTrajAndMSD( pathToSearch, genFigs )
%combineDxInt makes big combined arrays with particle intensity of
%found arrays
%
% Finds all .mat files in all subdirectories of the supplied
% path, looks for arrays found in them and combines them into one big array
% - currently tuned for dx, msd, turnangle results.

options.dTime = 0.0456;
options.dSpace = 0.107;

% list of arrays of interest and their associated files... wow.
field1 = 'arrayname';
field2 = 'filename';
value1 = {...
    'traj8ColMobile'
};
value2 = {...
    'lipidTraj'
};
arrays = struct(field1,value1,field2,value2);

% find all directories in 'pathToSearch'
dirList = dir(pathToSearch);

% trim out '.', '..', and non-directories
if gt(size(dirList,1),2)
    % '.' and '..' are always the first two rows
    dirList = dirList(3:size(dirList,1),1);
    % only keep if isdir is 1
    dirList = dirList(eq([dirList.isdir],1),1);
end

% main loop, repeat for each array of interest
for arraysIndex = 1:size(arrays,2)
    fprintf('Processing array %s
',arrays(arraysIndex).arrayname);

    % run through the subdirectories
    for dirIndex = 1:size(dirList,1)
        % does the file exist?
        filename = fullfile(pathToSearch,dirList(dirIndex).name,strcat(arrays(arraysIndex).filename, '.mat'));
        if eq(size(dir(filename),1),1)
            load(filename,arrays(arraysIndex).arrayname);
        else
            fprintf('-');
        end
        clear(sprintf('%s',arrays(arraysIndex).arrayname));
    end

    % I don't like this, but I can't find another way to preserve variable
    % names.
eval(sprintf(' %s = bigarray;',arrays(arraysIndex).arrayname));

    % Here is where I recreate functionality that should already exist. Why
    % doesn't '-append' create a file if it doesn't already exist? Stupid.
    if eq(exist(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '-cell.mat'))),0)
        save(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '-cell.mat'))),arrays(arraysIndex).arrayname);
    else
        save(fullfile(pathToSearch,strcat(arrays(arraysIndex).filename, '-cell.mat'))),arrays(arraysIndex).arrayname, '-append'
    end

    % Generate figures if genFigs is set
    if eq(genFigs,1)
        fprintf(' generating figure');
end

128
% Set up the msd object with 2 dimensions and units of µm and s.
msdObj = msdanalyzer(2, 'µm', 's');

% Add the trajectories to the msd object - Add every particle now and
% divide them up later with indices
msdObj = msdObj.addAll(bigarray);

% Calculate the drift in the images from the correlated velocities of
% all particles. It’s useful to have stuck and mobile particles here
% for better results.
msdObj = msdObj.computeDrift('velocity');

% Here is where I should correct for drift in the trajectories.
% FIXME

% Do the computations for MSD %and velocity autocorrelation
% Do this in parallel if possible FIXME: make this parfor safe
msdObj = msdObj.computeMSD; %{trajPropertyIndex(1,1)};
msdObj = msdObj.computeVCorr;

% Extract the MSDs for each index (these are cell arrays)
% Only [1,5,7] work now. No coloc/noncoloc data here...
%for i = [1,5,7]
%    msd{i,1} = msdObj.msd{trajPropertyIndex{i,1},1};
%end

% Make a plot of the MSDs
msdIndPlot = msdObj.plotMSD;

% Find the bulk MSD
msdBulk = msdObj.getMeanMSD;

% Save the various results
save(fullfile(pathToSearch,'msd.mat'),'msdObj','msdBulk');
csvwrite(fullfile(pathToSearch,'msdBulk.csv'),msdBulk);
end

clear(strcat(arrays(arraysIndex).arrayname), 'bigarray');
fprintf('
');
end
Is there a 'movies.txt' in the inputDir given?

if discoverMode not set and if 'movies.txt' exists, use discoverMode 0 (fix this if you add more discover modes)

Use the appropriate discover mode

Read in the file

```matlab
function [ channelList ] = discoverChannels2( inputDir )

%discoverChannels Discovers corresponding channels for tracking
% For every experiment, there are a number of images/movies that
% represent the same scene in different channels. This function
% determines which images/movies go together and returns a list of all
% corresponding channels.
% This is a simplified version that doesn't allow
% the use of ',' as a secondary delimiter:
% chan1:chan2.1,chan2.2
% is now written as
% chan1:chan2.1
% chan1:chan2.2
% on two lines instead of one.

% INPUT:
% inputDir is a directory containing multiple images/movies
% discoverMode is the method of detecting image/movie correlation
% 0 uses 'movies.txt' (default now), 1 uses directory names, ...

% OUTPUT:
% Returns a cell array in the form of ['chan1'] ['chan2']
% Where the array is 1 x N, where N is the number of channels found
% chan1 will be the filename/directory containing the movie of chan1
% chan2 will be the filename/directory containing the movie of chan2

% At the moment, this function only looks for a file called 'movies.txt'
% to figure out the correlation 'movies.txt' will be in the format
% chan1:chan2 ...
% Also, only two channels are handled at the moment... this isn't hard
% to fix, but I don't need it so I'm not going to fix it... nyah.
% TODO: The input files should be sanity checked before passing them on
% to the calling program: do the mentioned channels actually
% exist, strip leading and trailing spaces, etc

% Is there a 'movies.txt' in the inputDir given?
(Maybe it would be a good idea to look for variations of the filename or just .txt files that are in the correct format?)

%dirContents = dir(strcat(inputDir,'/movies.txt'));
dirContents = dir(fullfile(inputDir, 'movies.txt'));
if ne(exist(discoverMode,'var'),1)
    if eq(size(dirContents,1),1)
        discoverMode = 0;
    else
        error('No file named movies.txt was found. Make one or pick another discovery mode.');
    end
end

% Use the appropriate discover mode
switch discoverMode
    case 0
end

% Read in the file
```
moviesFileID = fopen(fullfile(inputDir, 'movies.txt'));

% How many channels are there? (Assuming that the first row is
% indicative of all of the rows. Fix this if it's not true)
numChannels = length(strfind(fgetl(moviesFileID), ':')) + 1;

% Read the fields delimited by a ":".
% Read everything as a string, since we're assuming these are
% filenames. You can do with them as you please outside of this
% function.
channelList = textscan(moviesFileID, '%s %s','delimiter',':','HeaderLines',0); % if you need more than two channel:
fclose(moviesFileID);

case 1
    error('This discover mode is not yet implemented.
');
otherwise
    error('This discover mode is not yet implemented.
');
end
function dx = dx1Intensity( traj8Col, ~, coloc )

% GETDX5 -- return the displacements of trajectories (non-colocalized, 
% colocalized, or all) with the accompanying intensity of the associated 
% particle

% This program calculates displacement vectors from a track array. The 
% output of the program is an array in one-to-one correspondence with the 
% track array, with data [dx, dy, dr]. dr is the length of the displacement 
% vector, in other words, sqrt(dx^2 + dy^2).

% INPUT:
% traj8Col is a track file [x,y,brightness,sqradgyr,blah,frame,track,coloc] ("blah" values are defined but not used by 
% dt is the number of steps to look at displacement over 
% rad is the radius to look in pixels 
% coloc determines whether to look at only non-colocalized (coloc=0), 
% colocalized (coloc=1) or all(coloc=2)

% OUTPUT:
% dx is an array of [dx, dy, (dx^2 + dy^2)^(1/2)]

% getdx5 5-22-00 Eric R. Weeks
% patched 8-14-01 to return -1 for nonvalid results
% see http://www.physics.emory.edu/~weeksidl/getdx.html for 
% more details
%
% This can be freely distributed, as long as this header is 
% left intact.

%adjusted to not count skipping in the track file. Michelle Knowles 1/5/2005
%adjusted to not overlap the steps. Michelle Knowles 5/16/05
%adjusted to look at only colocalized (num=1) or non colocalized (num=0) or all(num=2). m knowles 7/27/05
%similar to getdx3.pro
% PC ported from IDL file from Michelle Knowles 2013-10-22
% PC adjusted to strip stuck particles 2014-01-25
% PC moved stuck loop out to its own function 2014-05-07
% Maybe I should make this an argument, but for now, let’s just set it 
% here.
% Suppress informative outputs? (but not errors)
quiet = 1;
% For now, let’s keep dt as 1
dt = 1;

% Allowable values for coloc are 0,1,2. Anything else is treated as 2.
if and((ne(coloc,0)),(ne(coloc,1)))
    coloc = 2;
end

% Make index of coloc or non and NaN out the non-relevant tracks
if eq(coloc,0)
    noncolocIndex = find(eq(traj8Col(:,8),0));
    traj8Col(noncolocIndex,:) = nan(length(noncolocIndex),8);
elseif eq(coloc,1)
    colocIndex = find(eq(traj8Col(:,8),1));
    traj8Col(colocIndex,:) = nan(length(colocIndex),8);
end

% Shift the array of trajectories up dt steps
shiftTraj8Col = circshift(traj8Col,-dt);

% Find the difference between the track starting points over dt
dx = shiftTraj8Col(:,1:2) - traj8Col(:,1:2);

% Find the magnitude of the displacement
dx(:,3) = sqrt(dx(:,1).^2 + dx(:,2).^2);

% Transfer the intensities to the dx array
if eq(dt,1)
    dx(:,4) = traj8Col(:,3);
else
    error('figure out how to reppresent this. First step?');
end

% Find where in the array new tracks start
trackIndex = find(ne(shiftTraj8Col(:,7),traj8Col(:,7)));

% Remove the displacements that correspond to new tracks starting
dx(trackIndex,:) = nan(length(trackIndex),4);
function [fitresult, gof, fitinfo] = fitDxHist(centers, nelements, components, tau)

%FITDXHIST(CENTERS,NELEMENTS,COMPONENTS,TAU)
%  Fit displacement histograms to n component Rayleigh distribution and
%  solve for each component's diffusion coefficient (um^2/s) and realtive
%  contribution.
%  This version is intended to give a fine fit by running through starting
%  values until the fit converges and the R-squared (or other goodness of
%  fit value) reaches a defined threshold. With the icky data scaling bit
%  in place, this is reasonably fast and accurate.
%  INPUT:
%       centers:    histogram centers (x)
%       nelements:  histogram counts (y)
%       components: How many components to fit (1-3)
%       tau:        The time compnent, in seconds
%  OUTPUT:
%       fitresult : a fit object representing the fit.
%       gof : structure with goodness-of fit info.
%  2014-06-30 PC
%  2014-07-18 PC: since these have split up into several pieces, this one
%                   will become a fit that uses global optimization (if
%                   available)

% The curve fitting success is very dependent on the scale of the data.
% Ick.
% nelements = (nelements / sum(nelements)) * 100;
% Start here and move up... In most cases, this seems to be reasonable.
% startPoint = 0.9;
% Don't settle for anything less than this rsquare value (or pick something
% else appropriate from the gof struct).
% rsquare = 0.993;
% fitinfo.exitflag tells us what the status of the fit was. 0 is 'Maximum number of function evaluations or iterations wa
% fitinfo.exitflag = 0;
% gof.rsquare = 0;
% while or(eq(fitinfo.exitflag, 0),lt(gof.rsquare,rsquare))

{xData, yData} = prepareCurveData(centers, nelements);
% Set up fittype and options.
opts = fitoptions('Method', 'NonlinearLeastSquares');
% This is a bit clumsy, but should work considering I'll not likely fit to
% more than four components anyway.
switch components
    case 1
        equation = strcat( 'Amplitude1*(x/(2*D1',num2str(tau),'))*exp(-[(x^2)/(4*D1',num2str(tau),')]));
        % This is important! None of the fitted values should be negative. The
        % fitting breaks sometimes if the parameters are allowed to go negative.
        opts.Lower = [0 0];
        opts.StartPoint = [startPoint 1];
    case 2
        equation = strcat( '((Amplitude1*(x/(2*D1',num2str(tau),')))*exp(-[(x^2)/(4*D1',num2str(tau),')]))) + (Amplitude2
        opts.Lower = [0 0 0 0];
opts.StartPoint = [startPoint startPoint 1 1];

case 3
    equation = strcat('((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x^2)/(4*D1*',num2str(tau),')))) + (Amplitude2*(x/(2*D2' [num2str(tau),'])))) + (Amplitude3*(x/(2*D3')),num2str(tau),'));
    opts.StartPoint = [startPoint startPoint startPoint 1 1 1];
end

case 4
    equation = strcat('((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x^2)/(4*D1*',num2str(tau),')))) + (Amplitude2*(x/(2*D2' [num2str(tau),'])))) + (Amplitude3*(x/(2*D3')),num2str(tau),'));
    opts.StartPoint = [startTime startPoint startPoint startPoint 1 1 1 1];
end

% Fit the data to the appropriate equation
ft = fittype(equation, 'independent', 'x', 'dependent', 'y');

opts.DiffMaxChange = 0.1;
opts.Display = 'off';
opts.Robust = 'bisquare';
opts.MaxIter = 10000;

% Fit model to data. (remove the semicolon at the end for output)
[fitresult, gof, fitinfo] = fit(xData, yData, ft, opts);

fprintf('R squared: %d, startPoint: %d
',gof.rsquare,startPoint);

startPoint = startPoint * 1.1;

end

Plot fit with data.

figure('Name', 'Diffusion from histogram', 'Color',[1 1 1]);
h = plot(fitresult, xData, yData, 'predobs');

axis tight;
set(h(1), 'Color',[0 0 0])
set(h(2), 'LineWidth',2, 'LineStyle','-','Color',[0 0 1]);
set(h(3), 'LineWidth',0.1, 'LineStyle','-', 'Color',[0 0 1]);
set(h(4), 'LineWidth',0.1, 'LineStyle',':', 'Color',[0 0 1]);

warning('off','MATLAB:legend:IgnoringExtraEntries');
legend(h, 'Data', 'Distribution', '95% Confidence', 'Location','NorthEast');

% Label axes
xlabel('displacement (um)');
ylabel('occurrences');
grid on;
box off;
pbaspect([1.618 1 1]);

end

Published with MATLAB® R2015a
Because Matlab can't do a simple fit without choking, we try different starting parameters until the fit converges.

Fit: 'Diffusion from histogram'.

```matlab
function [fitresult, gof, fitinfo] = fitDxHist(centers, nelements, components, tau)

%FITDXHIST(CENTERS,NELEMENTS,COMPONENTS,TAU)
% Fit displacement histograms to n component Rayleigh distribution and
% solve for each component's diffusion coefficient (\mu m^2/s) and relative
% contribution.
% This version is intended to be fast and course, to feed
% starting values to a finer fit. That isn't the case, yet.
% INPUT:
% centers: histogram centers (x)
% nelements: histogram counts (y)
% components: How many components to fit (1-3)
% tau: The time component, in seconds
% OUTPUT:
% fitresult : a fit object representing the fit.
% gof : structure with goodness-of fit info.
% 2014-06-30 PC

% The curve fitting success is very dependent on the scale of the data.
% Ick.
nelements = (nelements / sum(nelements)) * 100;
% Start here and move up... In most cases, this seems to be reasonable.
startPoint = 0.9;
% Don't settle for anything less than this rsquare value (or pick something
% else appropriate from the gof struct).
r.square = 0.99;
% fitinfo.exitflag tells us what the status of the fit was. 0 is 'Maximum number of function evaluations or iterations
% while or(eq(fitinfo.exitflag, 0),lt(gof.rsquare,rsquare))
while or(eq(fitinfo.exitflag, 0),lt(gof.rsquare,rsquare))

% Setup fittype and options.
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
switch components
case 1
    equation = strcat( 'Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x^2)/(4*D1*',num2str(tau),'))))';
    opts.Lower = [0 0];
    opts.StartPoint = [startPoint 1];
    case 2
    equation = strcat( '((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x^2)/(4*D1*',num2str(tau),')))) + (Amplitude2*(x/(2*D2*'
                      opts.Lower = [0 0 0 0];
                      opts.StartPoint = [startPoint startPoint 1 1];
    case 3
    equation = strcat( '((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x^2)/(4*D1*',num2str(tau),')))) + (Amplitude2*(x/(2*D2*'
                      opts.Lower = [0 0 0 0 0 0];
                      opts.StartPoint = [startPoint startPoint startPoint 1 1 1];
    case 4
    equation = strcat( '((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x^2)/(4*D1*',num2str(tau),')))) + (Amplitude2*(x/(2*D2*'
```

[XData, yData] = prepareCurveData(centers, nelements);

% This is a bit clumsy, but should work considering I'll not likely fit to
% more than four components anyway.
```
opts.Lower = [0 0 0 0 0 0 0 0];
opts.StartPoint = [startPoint startPoint startPoint startPoint 1 1 1 1];
end

% Fit the data to the appropriate equation
ft = fittype(equation, 'independent', 'x', 'dependent', 'y');

opts.DiffMaxChange = 0.1;
opts.Display = 'Off';
%opts.Robust = 'Bisquare';
opts.MaxIter = 1000;
opts.MaxFun = 1000;
opts.TolFun = 1e-12;
opts.TolX = 1e-12;

% Fit model to data.
[fitresult, gof, fitinfo] = fit(xData, yData, ft, opts);

fprintf('R squared: %d, startPoint: %d
',gof.rsquare,startPoint);
startPoint = startPoint * 1.1;

end
%fprintf('%d
',startPoint);

Plot fit with data.

figure('Name', 'Diffusion from histogram', 'Color', [1 1 1]);
h = plot(fitresult, xData, yData, 'predobs');

axis tight;
set(h(1), 'Color', [0 0 0])
set(h(2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [0 0 1]);
set(h(3), 'LineWidth', 0.1, 'LineStyle', ':', 'Color', [0 0 1]);
set(h(4), 'LineWidth', 0.1, 'LineStyle', ':', 'Color', [0 0 1]);
set(h, 'XLim', [0 3]);
warning('off', 'MATLAB:legend:IgnoringExtraEntries');
legend(h, 'Data', 'Distribution', '95% Confidence', 'Location', 'NorthEast');

% Label axes
xlabel('displacement (um)');
ylabel('occurrences');
grid off;
box off;
pbaspect([1.618 1 1]);
Fit: ‘Diffusion from histogram’.

- The fitting options. Tweak these to fit better, faster, etc.
- Fit model to data.
- Plot fit with data.

```matlab
function [fitresult, gof, fitinfo] = fitDxHistFine(centers, nelements, components, tau, startPoint)

%FITDXHIST(centers,nelements,components,tau)
%  Fit displacement histograms to n component Rayleigh distribution and
%  solve for each component's diffusion coefficient (µm^2/s) and relative
%  contribution.

% This version is intended to give a finer fit, given the
% starting values from a coarser fit. Consider finding a coarse Jacobian
% with fitDxHistCoarse and passing that instead of startPoint...

% INPUT:
%     centers:    histogram centers (x)
%     nelements:  histogram counts (y)
%     components: How many components to fit (1-3)
%     tau:        The time component, in seconds
%     startPoint: The starting guess for the amplitudes of the
%                 components. This should be determined by fitDxHistCoarse.
%                 Ultimately, this arg should be a struct with individual
%                 values for each amplitude. As it is now, it works
%                 alright.

% OUTPUT:
%     fitresult : a fit object representing the fit.
%     gof : structure with goodness-of fit info.
%  2014-06-30 PC
%
% The curve fitting success is very dependent on the scale of the data.
% Ick. (If the scaling is changed in fitDxHistCoarse, change it here, too!)

nelements = (nelements / sum(nelements)) * 100;

% Don't settle for anything less than this rsquare value (or pick something
% else appropriate from the gof struct). This is currently not used.
rsquare = 0.993;

[xData, yData] = prepareCurveData(centers, nelements);

% Set up fittype and options.
opts = fitoptions('Method', 'NonlinearLeastSquares');

% This is a bit clumsy, but should work considering I'll not likely fit to
% more than four components anyway.
switch components
    case 1
        equation = strcat('Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x.^2)/(4*D1*',num2str(tau),')));
        % This is important! None of the fitted vaules should be negative. The
        % fitting breaks sometimes if the parameters are allowed to go negative.
        opts.Lower = [0 0];
        opts.StartPoint = [startPoint 1];
    case 2
        equation = strcat('((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x.^2)/(4*D1*',num2str(tau),')))) + (Amplitude2
        opts.Lower = [0 0 0 0];
        opts.StartPoint = [startPoint startPoint 1 1];
    case 3
        equation = strcat('((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x.^2)/(4*D1*',num2str(tau),')))) + (Amplitude2
        opts.Lower = [0 0 0 0 0 0];
        opts.StartPoint = [startPoint startPoint startPoint 1 1 1];
    case 4
        equation = strcat('((Amplitude1*(x/(2*D1*',num2str(tau),'))*exp(-((x.^2)/(4*D1*',num2str(tau),')))) + (Amplitude2
        opts.Lower = [0 0 0 0 0 0 0 0];
        opts.StartPoint = [startPoint startPoint startPoint startPoint 1 1 1 1];
end```
% Fit the data to the appropriate equation
ft = fittype(equation, 'independent', 'x', 'dependent', 'y');

The fitting options. Tweak these to fit better, faster, etc.
The current criteria are quite extreme, but still relatively fast.

% Maximum change in coefficients for finite difference gradients.
 opts.DiffMaxChange = 0.1;
% Change this to 'iter' to see the optimization at each iteration, 'final'
% to only show the last iteration, or 'off' to suppress all output.
% 'notify' only show output if the fit does not converge.
 opts.Display = 'notify';
% bisquare weights robust fit. 'LAR' is least absolute residual method, or
% 'off' to disable robust fitting
 opts.Robust = 'Bisquare';
% Maximum number of iterations allowed for the fit.
 opts.MaxIter = 10000;
% Maximum number of evaluations of the model allowed.
 opts.MaxFun = 10000;
% Termination tolerance on the model value.
 opts.Tolfun = 1e-18;
% Termination tolerance on the coefficient values.
 opts.TolX = 1e-18;
% Use fitoptions based normalization
 opts.Normalize = 'on';

Fit model to data.

[fitresult, gof, fitinfo] = fit(xData, yData, ft, opts);

Plot fit with data.

figure('Name', 'Diffusion from histogram', 'Color', [1 1 1]);
h = plot(fitresult, xData, yData, 'predobs');
axis tight;
set(h(1), 'Color', [0 0 0])
set(h(2), 'LineWidth', 2, 'LineStyle', '-', 'Color', [0 0 1])
set(h(3), 'LineWidth', 0.1, 'LineStyle', ':', 'Color', [0 0 1])
set(h(4), 'LineWidth', 0.1, 'LineStyle', ':', 'Color', [0 0 1])
% The axis limits:
xlim([0 2]);
ylim([0 3]);
warning('off', 'MATLAB:legend:IgnoringExtraEntries');
legend(h, 'Data', 'Distribution', '95% Confidence', 'Location', 'NorthEast');
% Label axes
xlabel('displacement (um)');
ylabel('occurrences');
grid off;
box off;
pbaspect([1.618 1 1]);

end

-----------------------------------------------
Published with MATLAB® R2015a
function [freemem, memsize] = freemem ()
% FREEMEM Returns the free and total memory in bytes

if isunix
    if ismac
        % implement this for mac
        [~,total] = unix(’memory_pressure | grep system’);
        [~,free] = unix(’memory_pressure | grep percentage’);
        totalstats = str2double(regexp(total, ’[0-9]*’, ’match’));
        freestats = str2double(regexp(free, ’[0-9]*’, ’match’));
        memsize = totalstats(2)*totalstats(3);
        freemem = freestats(1)/100*memsize;
    else
        % Linux implementation
        [~,w] = unix(’free | grep Mem’);
        stats = str2double(regexp(w, ’[0-9]*’, ’match’));
        memsize = stats(1)*1e3;
        freemem = (stats(3)+stats(end))*1e3;
    end
else
    % Windows implementation
    % -just use memory() and parse the output
end
function dx = getdx5( traj8Col, dt, coloc )

%GETDX5 -- find displacements of trajectories (non-colocalized, 
% colocalized, or all)
% This program calculates displacement vectors from a track array. The 
% output of the program is an array in one-to-one correspondence with the 
% track array, with data [dx, dy, dr]. dr is the length of the displacement 
% vector, in other words, sqrt(dx^2 + dy^2).
% INPUT:
% traj8Col is a track file [x,y,brightness,sqradgyr,blah,frame,track,coloc] ('blah' values are defined but not used by 
% dt is the number of steps to look at displacement over 
% rad is the radius to look in pixels 
% coloc determines whether to look at only non-colocalized (coloc=0), 
% colocalized (coloc=1) or all(coloc=2)
% OUTPUT:
% dx is an array of [dx, dy, (dx^2 + dy^2)^(1/2)]
% getdx5 5-22-00 Eric R. Weeks
% patched 8-14-01 to return -1 for nonvalid results
% see http://www.physics.emory.edu/~weeks/idl/getdx.html for 
% more details
% This can be freely distributed, as long as this header is
% left intact.
% adjusted to not count skipping in the track file. Michelle Knowles 1/5/2005
% adjusted to not overlap the steps. Michelle Knowles 5/16/05
% adjusted to look at only colocalized (num=1) or non colocalized (num=0) or all(num=2). m knowles 7/27/05
% similar to getdx3.pro
% PC ported from IDL file from Michelle Knowles 2013-10-22
% PC adjusted to strip stuck particles 2014-01-25
% PC moved stuck loop out to its own function 2014-05-07
% Maybe I should make this an argument, but for now, let's just set it here.
% Suppress informative outputs? (but not errors)
quiet = 1;
% Allowable values for coloc are 0,1,2. Anything else is treated as 2.
if and((ne(coloc,0)),(ne(coloc,1)))
    coloc = 2;
end
% Make index of coloc or non and NaN out the non-relevant tracks
if eq(coloc,0)
    noncolocIndex = find(eq(traj8Col(:,8),0));
    traj8Col(noncolocIndex,:) = nan(length(noncolocIndex),8);
elseif eq(coloc,1)
    colocIndex = find(eq(traj8Col(:,8),1));
    traj8Col(colocIndex,:) = nan(length(colocIndex),8);
end
% Shift the array of trajectories up dt steps
shiftTraj8Col = circshift(traj8Col,-dt);
% Find the difference between the track starting points over dt
dx = shiftTraj8Col(:,1:2) - traj8Col(:,1:2);
% Find the magnitude of the displacement
dx(:,3) = sqrt(dx(:,1).^2 + dx(:,2).^2);
% Find where in the array new tracks start
trackIndex = find(ne(shiftTraj8Col(:,7),traj8Col(:,7)));
% Remove the displacements that correspond to new tracks starting
dx(trackIndex,:) = nan(length(trackIndex),3);
% add code here to trim NaNs from array (do we want/need to?)
end

Published with MATLAB® R2015a
%getresidtime  Counts sequential 'colocalized' marks to determine the
%residence time of diffusing particles.
%
% INPUT     traj8Col    An 8 column list of trajectories. The 8th column
%                    should indicate colocalization with a '1'.
%           frameInterval The interval of time between frames. The units of
%                    the supplied number will be preserved in the
%                    output.
%
% OUTPUT    residtime   A structure containing the residence time stats
%                    (All numbers reported are in the units of
%                    frames, unless frameInterval is supplied)
%                      .mean   The mean residence time
%                      .stddev The standard deviation
%                      .num    The number of samples used to calculate the above
%                      .min    The smallest number in the set
%                      .max    The largest number in the set
%                      .median The median of the set
%                      .mode   The mode of the set
%                      .set    The entire set of residence times
%                      .hist   A histogram of residence times with the same number
%                             of bins as frames in the data
%
% 2014-09-03 PC
%

% Is traj8Col actually 8 columns and is the eighth column full of only
% zeroes and ones? (The third test will pass for negative numbers (and
% NaNs?) and should be fixed.)
if lt(size(traj8Col,1),1)
    error('traj8Col needs to contain some data!');
elseif lt(size(traj8Col,2),8)
    error('traj8Col needs to be at least 8 columns long');
elseif lt(size(find(traj8Col(:,8) <= 1),1),1)
    error('traj8Col needs to have zeroes and ones in the eighth column');
end
% Mask out non-colocalized steps
colocMask = eq(traj8Col(:,8),1);

% Shift the array of trajectories up one step
shiftTraj8Col = circshift(traj8Col(colocMask,:),-1);

% Find the difference between the original and shifted frame numbers. A % difference of 1 indicates sequential frames. This will drop the % first/last frames in a track. How big of a difference will this make over % a large number of particles? Add padding to each track to prevent this.
diffTraj8Col = shiftTraj8Col(:,6) - traj8Col(colocMask,6);

% Find where in the array new colocalization events start
newColocIndex = find(ne(diffTraj8Col,1));

% ...and how long these events are.
shiftNewColocIndex = circshift(newColocIndex,1);
shiftNewColocIndex(1,1) = zeros(1,1);
residtime.set = newColocIndex - shiftNewColocIndex;

Statistics on residence time

% Mean, median, mode
residtime.mean = mean(residtime.set);
residtime.median = median(residtime.set);
residtime.mode = mode(residtime.set);

% Standard deviation
residtime.stddev = std(residtime.set);

% Number of events found
residtime.num = size(residtime.set,1);

% Min and max
residtime.min = min(residtime.set);
residtime.max = max(residtime.set);

Make histogram

The histogram has the same number of bins as the largest number of frames in the input. This makes it easier to change the number of bins later without having to deal with aliasing artifacts.

% Make a little array of the bin centers
maxFrame = max(traj8Col(:,6));
residtime.hist.centers = linspace(1,maxFrame,maxFrame);

% Make the histogram itself
residtime.hist.elements = hist(residtime.set,residtime.hist.centers);
% Transpose them because this is how we deal with data in the civilized
% world (broken out because this is easy to miss otherwise)
residtime.hist.centers = residtime.hist.centers';
residtime.hist.elements = residtime.hist.elements';

Handle frameInterval if supplied.

    if exist('frameInterval', 'var')
        % Convert the units from frames to whatever was supplied
        residtime.mean = residtime.mean .* frameInterval;
        residtime.stddev = residtime.stddev .* frameInterval;
        residtime.min = residtime.min .* frameInterval;
        residtime.max = residtime.max .* frameInterval;
        residtime.median = residtime.median .* frameInterval;
        residtime.mode = residtime.mode .* frameInterval;
        residtime.set = residtime.set .* frameInterval;
        residtime.hist.centers = residtime.hist.centers .* frameInterval;
    end

end

Published with MATLAB® R2015a
function [ status ] = isParallelPossible ()
% All this function does is check whether the Parallel Computing toolbox is
% installed and configured. It can be expanded to check licenses and such.
% I got sick of maintaining duplicated code.
%
% returns status = 1 for yes and 0 for no

% Is a cluster configured
status = and(eq(exist('pctconfig','file'),2),gt(length(pctconfig),0));

%if and(eq(exist('pctconfig','file'),2),gt(length(pctconfig),0))
%    status=1;

%Is a pool configured
%if and(eq(exist('gcp','file'),2),ne(length(gcp('nocreate')),0))
%    status = 1;
%else
%    status = 0;
%end
%else
%    status=0;
%end

end
function [ ] = kymodraw( kymograph, options )

%kymograph - Shows time traces from a movie at specific locations
% This accepts a movie matrix and a position matrix and draws a series of
% traces representing the pixel intensities at the specified positions.
% If options.roiSize is defined, a circular region around the position
% is averaged instead of just the single pixel at that position. The
% option options.trace determines how the kymograph is drawn.
%  
% INPUT:
% kymograph is a three dimensional array representing a kymograph of
% the format [ position, time, intensity ]
% options is a struct containing the following options
% .trace is the type of trace generated for each of the positions
% in the kymograph, as follows:
% pixel/region intensity is represented as...
% 1 = a change in color
% 2 = a deviation in the y axis.
%

% Make an index for the input kymograph
%kymoIndex = linspace(1,size(kymograph,1),size(kymograph,1));

% Turn this into a column vector (broken out to make it easier to read)
%kymoIndex = kymoIndex';

% Process options

% Does options.trace exist? If so, is it larger than 1? If not, then it's 1.
if exist( 'options.trace', 'var' )
    if le(options.trace, 1)
        options.trace = 1;
    end
else
    options.trace = 1;
end

% Make the plot figure.
Here will go the different methods of generating traces

```matlab
% Plot our kymograph
kymoFig = imagesc(kymograph);
```

**GUI stuff**

make a way to hover or click and identify or jump to interesting parts of movies

```matlab
end
```

Published with MATLAB® R2015a
function [ kymograph ] = kymograph( movie, positions, options )

% kymograph - Shows time traces from a movie at specific locations
% This accepts a movie matrix and a position matrix and returns a series
% of traces representing the pixel intensities at the specified positions.
% If options.roiSize is defined, a circular region around the position
% is averaged instead of just the single pixel at that position.
% INPUT:
%   movie is a three dimensional array representing a movie of the
%   format [y,x,t]
%   positions is a two dimensional array of positions within the bounds
%   of movie. The kymograph will be generated from these
%   positions in the movie
%   options is a struct containing the following options
%       .roiSize is the radius of the region around a position that is
%       sampled, averaged, and used in the kymograph

% Reported positions may be given at subpixel resolution. Floor() these to
% capture the relevant pixel. (double check that floor is correct)
positionsFloored = floor(positions);

% Does options.roiSize exist? If so, is it larger than 1? If not, then it's 1.
if exist('options.roiSize', 'var')
    if le(options.roiSize, 1)
        options.roiSize = 1;
    end
else
    options.roiSize = 1;
end

% the array is everything in the t axis at position y,x
% Each position gets its own row
% Preallocate array
kymograph = zeros(size(positionsFloored,1),size(movie,3));

if eq(options.roiSize,1)
    % Fill out the array:
    % (There has got to be a way to do this without a loop. Figure it out!)
    % (meh, it's fast enough...)
    for positionIndex = 1 : size(positionsFloored,1)
        kymograph(positionIndex,:) = movie(positionsFloored(positionIndex,1),positionsFloored(positionIndex,2),:);
    end
else
    % Average the intensities in a circle of radius options.roiSize
    % Fill out the array
end
end
function [] = kymowrap( movieDir, analysisDir, options )

%kymograph - Loads movies and positions and passes them to kymograph()
%   This loads a TIFF stack from movieDir and a position matrix from
%   analysisDir and passes them, along with options, to kymograph(). This
%   wrapper is used to keep kymograph nice and generalized while allowing
%   wrappers to be tweaked to specific directory layouts and other needs.
%   INPUT:
%       movieDir is a directory containing a TIFF stack or a series of
%          individual TIFFs
%       analysisDir is a directory containing 'NPs.mat' which contains an
%          array called nps2Col
%       options is a struct containing options
%          .roisize defines the radius of the area around each
%            position to use in the graph
%          .trace defines the representation of the output. See
%            kymodraw.m for the available choices

Load our data

% The name of the file that contains the positions
analysisFile = 'NPs.mat';
positionVar = 'nps2Col';

% Load the movies from the provided directory
movie = loadTIFFs(movieDir);

% Load the desired file from the provided analysisDir
positionFile = strcat(analysisDir, '/', analysisFile);
load(positionFile, positionVar);

% Fish the position array out of the file (fixme)
positions = nps2Col;

Sanity test time...

(Write these sanity tests!)
% Is the movie actually three dimensional?
% Do all of the positions fall within the dimensions of the movie?

Process options.

% Does options.roiSize exist? If so, is it larger than 1? If not, then it's 1.
if exist('options.roiSize', 'var')
    if le(options.roiSize, 1)
        options.roiSize = 1;
    end
else
    options.roiSize = 1;
end

% Does options.trace exist? If so, is it larger than 1? If not, then it's 1.
if exist('options.trace', 'var')
    if le(options.trace, 1)
        options.trace = 1;
    end
else
    options.trace = 1;
end

Call external functions

% Call kymograph()
kymographArray = kymograph(movie, positions, options);

% Call kymodraw()
kymodraw(kymographArray);

end

Published with MATLAB® R2015a
Contents

- Constants and such
- Distributed Computing
- Condition reporting
- Make new working directory for this analysis
- Load the map of nanoparticles to diffusing particles
- The main loop
- What's going on?
- New output subdirectory for scene
- Load static image of nanoparticles
- Make a mean intensity projection if TIFF is multi-frame.
- Localize the particles
- Load movie of lipids diffusing, bandpass, and subtract the background
- Find the spots in every frame and make array of this \([x,y,\text{frame}]\)
- Track these results
- Convert these, yet again, for the various dx and MSD programs to
- Find colocalization with NPs
- Find stuck particles and make separated stuck and mobile traj8Col arrays
- Rewrite traj8Col in terms of \(\mu\text{m}\)
- Write track files
- Calculate displacement vectors
- Calculate residence times (at nanoparticles) and generate statistics and histograms (and rewrite in terms of ms)
- Calculate MSD and rewrite in terms of ms
- Calculate turning angles
- Make figures of calculated results
- Save report

```matlab
function [ traj8Col, nps2Col, report ] = loadAndTrack2( inputDir, outputDir, makePlots, constants )

%LOADANDTRACK - Loads movies of diffusing particles and does some analysis
% This is the main program that loads all movies, detects and tracks
% particles within them, then calculates MSD, turning angle, displacement
% histograms, etc for the tracks founds and makes pretty figures showing
% what was found.
% INPUT:
% inputDir is a directory containing (usually) individual directories
% with images and movies of nanoparticles, diffusing
% particles, etc.
% outputDir is the directory where the results are to be written. If
% it doesn't exist, it will be created. This directory will
% be populated with subdirectories containing the results of
% each analyzed scene (nanoparticle/diffusing particle pair);
% makePlots determines whether figures are made of the results (1) or
% not (0).
% constants is a struct containing parameters specific to the
% captured images, such as:
% constants.umPx = the size of a square pixel's side in \(\mu\text{m}\).
% constants.msInterval = the interval between frames in ms.
% PC 2013-11-12  - Started this beast
% PC 2014-02-25  - better file handling: just give a directory name and it
% finds TIFFs inside it, as a bunch of files or as a
% multiframe TIFF.
% For NPs, movies are averaged (mean intensity projection)
% PC 2014-02-06  - even better file handling. Tries to discover the map of
% nanoparticles to diffusing particles and load the
% appropriate files for each.
% PC 2014-05-06  - write stuck/colocalized/total counts to report struct
% PC 2014-05-07  - separated out the stuck particle finding loop into its
% own function and rewrote everything else to accommodate it
% TODO:
% * For thresholds (pkfind, cntrd, etc), devise a way to test
% images for appropriate values instead of having them set here.
% * Split more of the functions out into individual files.
```

152
% Keep the blah,blah,blah columns from cntrd.m and translate
% them to their appropriate places in the track arrays.
% 
% NOTES:
% * Throughout my code, I use eq(var,1) instead of just var to
test var in if loops, etc on purpose. I think it makes the
code easier to read and change, and it costs little in terms
of overhead.

### Constants and such

(I'd love to scrape this from the TIFF metadata, but it's so often inconsistent and would require lots of sanity checking...) Below are sane values.

% Build this struct before calling the function. Otherwise, use my
% defaults.
if exist('constants','var')
    % Test supplied values for reasonableness
else
    % The size of a square pixel's side in \textmu{}m. Default is 0.107
    constants.umPx = 0.107;
    % The interval between frames in ms. Note that this is NOT the exposure time
    constants.msInterval = 45.6;
end

### Distributed Computing

See if parallel computing is set up and use it if it is. This may be overly biased to the PBS pool at DU.

% Simple off switch. Set to 0 to turn off and 1 to turn on.
distcomp = 0;

if distcomp
    % Does pctconfig exist... Is the distcomp toolbox installed? Doesn't check
    % licensing bs.
    if eq(exist('pctconfig','file'),2)
        % Is a cluster configured?
        if gt(length(pctconfig),0)
            % Initialize the cluster and get the details. This needs work.
            clusterDetails = parClusterInit;
            fprintf('Parallel Computing configured. Using cluster blah with num nodes.\n');
            distcomp = 1;
        else
            fprintf('Parallel Computing not configured. Continuing locally...\n');
            distcomp = 0;
        end
    else
        fprintf('Parallel Computing Toolbox not present. Continuing locally...\n');
        distcomp = 0;
    end
else
    fprintf('Parallel Computing not configured. Continuing locally...
');
    distcomp = 0;
end

### Condition reporting

Use the blink1 to report success, errors, etc

if eq(exist('blink1','file'),2)
    blinkExists = 1;
    blink1('off');
else
    blinkExists = 0;
end

### Make new working directory for this analysis

Analyses of individual scenes will go in subdirectories of this

warning('off','MATLAB:MKDIR:DirectoryExists');
mkdirSuccess = mkdir(outputDir);
if and(mkdirSuccess,0)
    error('Failed to create working directory')
end
%cd(outputDir);

### Load the map of nanoparticles to diffusing particles
The main loop

```
for sceneNum = 1:numScenes
```

What's going on?

```
fprintf('Scanning scene %s (%d/%d):
',channelList{1,2}{sceneNum,1},sceneNum,numScenes);
```

New output subdirectory for scene

```
sceneDir = fullfile(outputDir,channelList{1,2}{sceneNum,1});
```

Load static image of nanoparticles

```
nps = loadTIFFs(fullfile(inputDir,channelList{1,1}{sceneNum,1}));
```

Make a mean intensity projection if TIFF is multi-frame.

```
if gt(size(nps,3),1)
    nps = mean(nps,3);
else
    nps = double(nps);
end
```

Localize the particles

```
bpsBP = bpass(nps,1,7);
% Find pixel-level positions with intensity above 500 and separation of 5/2
npsPosRough = pkfnd(npsBP,100,5);
% Don’t try working with empty sets of NPs...
if gt(size(npsPosFine,2),0)
    emptyNPset = 0;
else
    emptyNPset = 1;
end
```

% Save these for colocalization...
if writeFiles
    % This was originally an argument to the function, but it
    % doesn’t hurt to leave it on. Simplify, simplify, simplify.
    save(fullfile(sceneDir,'NPs.mat'),'nps2Col');
csvwrite(fullfile(sceneDir,'NPs.csv'),nps2Col);
end

% Write the number of NPs in the report, too.
report.npCount = size(nps2Col,1);
Load movie of lipids diffusing, bandpass, and subtract the background

```matlab
fprintf('Loading files... \n');
lipids = loadTIFFs(fullfile(inputDir,channelList{1,2}{sceneNum,1}));
fprintf('Subtracting background... \n');
lipidsBgSub = bgSubByBpass(lipids);

% A little memory management
clear lipids;
```

Find the spots in every frame and make array of this [x,y,frame]

```matlab
fprintf('Detecting particles... ');
lipidPosAll = [];
% GUIfic progress bar. Super annoying with huge batches, so disabled
% Here's a soothing text progress bar in its place...
reverseStr = ' ';
% = waitbar(0,'detecting particles...');
spotless = 0;
numberFrames = size(lipidsBgSub,3);
for i = 1:numberFrames
    more GUIfic progress bar. Super annoying with huge batches, so disabled
    % = waitbar(percDone/100,h,sprintf('Detecting particles... \%0.0f\%',percDone));
    percDone = i/numberFrames*100;
    progress = sprintf('%3.1f',percDone);
    fprintf('%%%s',reverseStr,progress);
    reverseStr = repmat(sprintf( ''), 1, length(progress) + 1);

    lipidFrame = lipidsBgSub(:,:,i);
    lipidFrameBP = bpass(lipidFrame,1,4);
    lipidFramePosRough = pkfnd(lipidFrame,50,3);
    lipidFramePosFine = cntrd(lipidFramePosRough,9);
    numSpotsFound = size(lipidFramePosFine,1);
    clear lipidFrame;
    % Skip this frame if no spots are found. This could be a bad sign and should
    % be dealt with better.
    if eq(numSpotsFound,0)
        spotless = 1;
        fprintf('Warning: Spotless frame found in scene %s, frame %d
',channelList{1,2}{sceneNum,1},i);
        %fprintf('Warning: Spotless frame found in scene %s, frame %d
',sceneList{1,2}{sceneNum,1},i);
        continue
    end

    index = repmat(i,numSpotsFound,1);
    lipidFramePosFineIndex = cat(2,lipidFramePosFine,index);
    if eq(index,1)
        lipidPosAll = lipidFramePosFineIndex;
    else
        lipidPosAll = cat(1,lipidPosAll,lipidFramePosFineIndex);
    end
end

% A little memory management...
clear lipidsBgSub;
if gt(spotless,0)
    % Warn if frames without spots were found
    fprintf('-- Warning: Spotless frames were found in scene %s
',channelList{1,2}{sceneNum,1});
else
    fprintf(' '); %fprintf(' '); % GUIfic progress bar. Super annoying with huge batches, so disabled
    %close(h);
    %close all hidden;
end

% Save positions with all five columns [x,y,brightness,sqRadGyr,frame]
%if writeFiles
    save(fullfile(sceneDir, 'lipidPos.mat'),'lipidPosAll');
csvwrite(fullfile(sceneDir, 'lipidPos.csv'),lipidPosAll);
%end

% Remove columns after [x,y] because track.m doesn't want them
lipidPos3Col = lipidPosAll(:,1:2);
% Add the last column [frame] because this way seems to be fastest. This
% can't be the best way to do this...
lipidPos3Col = cat(2,lipidPos3Col,lipidPosAll(:,5));
clear lipidPosAll;
% Save positions with columns suitable for tracking [x,y,frame]
%if writeFiles
```

155
Track these results

```
fprintf('Tracking particles...
');
```

```
% The maxdisp argument below (the second argument) is determined by the
% max expected diffusion of the particles. This will need to be scaled
% manually. If an ugly unhandled error crashes in track.m saying
% something about "Excessive Combinatorics", this needs to be reduced.
% I'd love to properly handle that error or detect it and reduce
% maxdisp, but I haven't yet done so. Be warned.
traj4Col = track(lipidPos3Col,5,struct('good',5,'mem',0,'dim',2,'quiet',0));
```

```
% If no tracks are found, continue to the next scene gracefully.
if eq(length(traj4Col),0)
    fprintf('No suitable tracks were found in scene %s.

    Finished with scene %s.

    continue
    else
    fprintf('%d tracks were found in scene %s

    traj7Col = traj4Col(:,1:2);
    trajLength = size(traj4Col,1);
    blah3Col = zeros(trajLength,3);
    traj7Col = cat(2,traj7Col,blah3Col,traj4Col(:,3:4));
    clear traj4Col;
```

Find colocalization with NPs

```
if ne(emptyNPset,1)
    fprintf('Finding colocalization events...
');
    [traj8Col, report.coloc] = separatetracks4(traj7Col,nps2Col,2);
else
    fprintf('No nanoparticles found, skipping colocalisation events...
');
    traj8Col = cat(2,traj7Col,zeros(size(traj7Col,1),1));
    clear traj7Col;
```

Find stuck particles and make separated stuck and mobile traj8Col arrays

```
maxStuckDisp = 6;
    [ traj8ColStuck, traj8ColMobile, report.stuck ] = separateStuck(traj8Col,maxStuckDisp);
```

```
% Rewrite report to exclude stuck particles in (non)colocalisation
% counts
    report.coloc.colocNonstuck = sum(eq(traj8ColMobile(:,8),1));
    report.coloc.noncolocNonstuck = sum(eq(traj8ColMobile(:,8),0));
```

Rewrite traj8Col in terms of \( \mu m \)

```
This uses constants.umPx, which is an argument of, or defined at the beginning of loadAndTrack2 (this function)
```

```
traj8ColUm = cat(2,traj8Col(:,1:2) .* constants.umPx,traj8Col(:,3:8));
```

Write track files

```
% If writeFiles
    save(fullfile(sceneDir, 'lipidTraj.mat'), 'traj8Col', 'traj8ColStuck', 'traj8ColMobile');
csvwrite(fullfile(sceneDir, 'lipidTraj.csv'),traj8Col);
csvwrite(fullfile(sceneDir, 'lipidTrajStuck.csv'),traj8ColStuck);
csvwrite(fullfile(sceneDir, 'lipidTrajMobile.csv'),traj8ColMobile);
```
Calculate displacement vectors
This is clumsy.

Compute displacement vectors:

```matlab
fprintf('Calculating displacement vectors...
');
% ...mobile
dx01_mobile_all = getdx5(traj8ColMobileUm,1,2);
dx02_mobile_all = getdx5(traj8ColMobileUm,2,2);
dx05_mobile_all = getdx5(traj8ColMobileUm,5,2);
dx10_mobile_all = getdx5(traj8ColMobileUm,10,2);
dx01_mobile_coloc = getdx5(traj8ColMobileUm,1,0);
dx02_mobile_coloc = getdx5(traj8ColMobileUm,2,0);
dx05_mobile_coloc = getdx5(traj8ColMobileUm,5,0);
dx10_mobile_coloc = getdx5(traj8ColMobileUm,10,0);
dx01_mobile_noncoloc = getdx5(traj8ColMobileUm,1,1);
dx02_mobile_noncoloc = getdx5(traj8ColMobileUm,2,1);
dx05_mobile_noncoloc = getdx5(traj8ColMobileUm,5,1);
dx10_mobile_noncoloc = getdx5(traj8ColMobileUm,10,1);
% ...all
dx01_all_all = getdx5(traj8ColUm,1,2);
dx02_all_all = getdx5(traj8ColUm,2,2);
dx05_all_all = getdx5(traj8ColUm,5,2);
dx10_all_all = getdx5(traj8ColUm,10,2);
dx01_all_coloc = getdx5(traj8ColUm,1,0);
dx02_all_coloc = getdx5(traj8ColUm,2,0);
dx05_all_coloc = getdx5(traj8ColUm,5,0);
dx10_all_coloc = getdx5(traj8ColUm,10,0);
dx01_all_noncoloc = getdx5(traj8ColUm,1,1);
dx02_all_noncoloc = getdx5(traj8ColUm,2,1);
dx05_all_noncoloc = getdx5(traj8ColUm,5,1);
dx10_all_noncoloc = getdx5(traj8ColUm,10,1);
% if writeFiles
% ...mobile
save(fullfile(sceneDir, 'dx_mobile_all.mat'),'dx01_mobile_all','dx02_mobile_all','dx05_mobile_all','dx10_mobile_all');
save(fullfile(sceneDir, 'dx_mobile_coloc.mat'),'dx01_mobile_coloc','dx02_mobile_coloc','dx05_mobile_coloc','dx10_mobile_coloc');
save(fullfile(sceneDir, 'dx_mobile_noncoloc.mat'),'dx01_mobile_noncoloc','dx02_mobile_noncoloc','dx05_mobile_noncoloc');
% ...stuck
save(fullfile(sceneDir, 'dx_stuck_all.mat'),'dx01_stuck_all','dx02_stuck_all','dx05_stuck_all','dx10_stuck_all');
save(fullfile(sceneDir, 'dx_stuck_coloc.mat'),'dx01_stuck_coloc','dx02_stuck_coloc','dx05_stuck_coloc','dx10_stuck_coloc');
save(fullfile(sceneDir, 'dx_stuck_noncoloc.mat'),'dx01_stuck_noncoloc','dx02_stuck_noncoloc','dx05_stuck_noncoloc');
% ...all
save(fullfile(sceneDir, 'dx_all_all.mat'),'dx01_all_all','dx02_all_all','dx05_all_all','dx10_all_all');
save(fullfile(sceneDir, 'dx_all_coloc.mat'),'dx01_all_coloc','dx02_all_coloc','dx05_all_coloc','dx10_all_coloc');
save(fullfile(sceneDir, 'dx_all_noncoloc.mat'),'dx01_all_noncoloc','dx02_all_noncoloc','dx05_all_noncoloc','dx10_all_noncoloc');
%end
```

Calculate residence times (at nanoparticles) and generate statistics and histograms (and rewrite in terms of ms)

Residence times:

```matlab
fprintf('Calculating residence times...
');
% All particles
report.residtime.all = getresidtime(traj8Col,constants.msInterval);
```
Calculate MSD and rewrite in terms of ms

This is also clumsy. This uses constants.msInterval, which is defined at the beginning of this function (Should we just pass the conversion to msd.m and do the conversion there?)

```matlab
% if writeFiles
save(fullfile(sceneDir, 'msd_all.mat'),'msd_all_all','msd_mobile_all','msd_stuck_all');
save(fullfile(sceneDir, 'msd_coloc.mat'),'msd_all_coloc','msd_mobile_coloc','msd_stuck_coloc');
save(fullfile(sceneDir, 'msd_noncoloc.mat'),'msd_all_noncoloc','msd_mobile_noncoloc','msd_stuck_noncoloc');
%end
```

Calculate turning angles

Same method of other calculation sections...

```matlab
fprintf('Calculating turning angles...\n');
turn_all_all = turnangle(traj8ColUm,2);
turn_stuck_all = turnangle(traj8ColStuckUm,2);
turn_mobile_all = turnangle(traj8ColMobileUm,2);
turn_all_coloc = turnangle(traj8ColUm,1);
turn_stuck_coloc = turnangle(traj8ColStuckUm,1);
turn_mobile_coloc = turnangle(traj8ColMobileUm,1);
turn_all_noncoloc = turnangle(traj8ColUm,0);
turn_stuck_noncoloc = turnangle(traj8ColStuckUm,0);
turn_mobile_noncoloc = turnangle(traj8ColMobileUm,0);
```
Make figures of calculated results

Once again... clumsy.

if makePlots
    fprintf('Generating figures...');
    fprintf('(displacement histograms)...');
    % The displacement histograms
    % ...mobile
    plotHistograms(dx01_mobile_all,0,struct('dirName',sceneDir,'prefix','dx01-mobile-all'));
    plotHistograms(dx02_mobile_all,0,struct('dirName',sceneDir,'prefix','dx02-mobile-all'));
    plotHistograms(dx05_mobile_all,0,struct('dirName',sceneDir,'prefix','dx05-mobile-all'));
    plotHistograms(dx10_mobile_all,0,struct('dirName',sceneDir,'prefix','dx10-mobile-all'));
    plotHistograms(dx01_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx01-mobile-coloc'));
    plotHistograms(dx02_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx02-mobile-coloc'));
    plotHistograms(dx05_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx05-mobile-coloc'));
    plotHistograms(dx10_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx10-mobile-coloc'));
    plotHistograms(dx01_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx01-mobile-noncoloc'));
    plotHistograms(dx02_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx02-mobile-noncoloc'));
    plotHistograms(dx05_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx05-mobile-noncoloc'));
    plotHistograms(dx10_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx10-mobile-noncoloc'));
    % ...stuck
    plotHistograms(dx01_stuck_all,0,struct('dirName',sceneDir,'prefix','dx01-stuck-all'));
    plotHistograms(dx02_stuck_all,0,struct('dirName',sceneDir,'prefix','dx02-stuck-all'));
    plotHistograms(dx05_stuck_all,0,struct('dirName',sceneDir,'prefix','dx05-stuck-all'));
    plotHistograms(dx10_stuck_all,0,struct('dirName',sceneDir,'prefix','dx10-stuck-all'));
    plotHistograms(dx01_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx01-stuck-coloc'));
    plotHistograms(dx02_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx02-stuck-coloc'));
    plotHistograms(dx05_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx05-stuck-coloc'));
    plotHistograms(dx10_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx10-stuck-coloc'));
    plotHistograms(dx01_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx01-stuck-noncoloc'));
    plotHistograms(dx02_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx02-stuck-noncoloc'));
    plotHistograms(dx05_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx05-stuck-noncoloc'));
    plotHistograms(dx10_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx10-stuck-noncoloc'));
    % ...all
    plotHistograms(dx01_all_all,0,struct('dirName',sceneDir,'prefix','dx01-all-all'));
    plotHistograms(dx02_all_all,0,struct('dirName',sceneDir,'prefix','dx02-all-all'));
    plotHistograms(dx05_all_all,0,struct('dirName',sceneDir,'prefix','dx05-all-all'));
    plotHistograms(dx10_all_all,0,struct('dirName',sceneDir,'prefix','dx10-all-all'));
    plotHistograms(dx01_all_coloc,0,struct('dirName',sceneDir,'prefix','dx01-all-coloc'));
    plotHistograms(dx02_all_coloc,0,struct('dirName',sceneDir,'prefix','dx02-all-coloc'));
    plotHistograms(dx05_all_coloc,0,struct('dirName',sceneDir,'prefix','dx05-all-coloc'));
    plotHistograms(dx10_all_coloc,0,struct('dirName',sceneDir,'prefix','dx10-all-coloc'));
    plotHistograms(dx01_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx01-all-noncoloc'));
    plotHistograms(dx02_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx02-all-noncoloc'));
    plotHistograms(dx05_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx05-all-noncoloc'));
    plotHistograms(dx10_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx10-all-noncoloc'));
    % The MSD plots
    % (Trying out the nifty conditional operator && to avoid making more stupid if...end loops around everything. If there is only one element in any of these, then it is almost certainly a single NaN. The plot function doesn't handle that situation yet.)
    fprintf('(MSD plots)...');
    gt(length(msd_all_all),1) && plotMSD(msd_all_all,0,struct('dirName',sceneDir,'prefix','msd-all-all'));
    gt(length(msd_mobile_all),1) && plotMSD(msd_mobile_all,0,struct('dirName',sceneDir,'prefix','msd-mobile-all'));
    gt(length(msd_stuck_all),1) && plotMSD(msd_stuck_all,0,struct('dirName',sceneDir,'prefix','msd-stuck-all'));
    gt(length(msd_all_coloc),1) && plotMSD(msd_all_coloc,0,struct('dirName',sceneDir,'prefix','msd-all-coloc'));
    gt(length(msd_mobile_coloc),1) && plotMSD(msd_mobile_coloc,0,struct('dirName',sceneDir,'prefix','msd-mobile-coloc'));
    gt(length(msd_stuck_coloc),1) && plotMSD(msd_stuck_coloc,0,struct('dirName',sceneDir,'prefix','msd-stuck-coloc'));
    gt(length(msd_all_noncoloc),1) && plotMSD(msd_all_noncoloc,0,struct('dirName',sceneDir,'prefix','msd-all-noncoloc'));
    gt(length(msd_mobile_noncoloc),1) && plotMSD(msd_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','msd-mobile-noncoloc'));
    gt(length(msd_stuck_noncoloc),1) && plotMSD(msd_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','msd-stuck-noncoloc'));
}
Save report

For now, this is just a .mat with some numbers in it. Maybe make a nice report later...

```matlab
save(fullfile(sceneDir,'/','report.mat'),'report','constants');
fprintf(['\nFinished with scene %s\n\n\n',channelList{1,2}{sceneNum,1});
end
fprintf('Done!\n');
if blinkExists
    blink1('on','green');
end
end
```

Published with MATLAB® R2015a
Define constants and such
Distributed Computing
Condition reporting
Make new working directory for this analysis
Load the map of nanoparticles to diffusing particles
The main loop
What’s going on?
New output subdirectory for scene
Load static image of nanoparticles
Make a mean intensity projection if TIFF is multi-frame.
Localize the particles
Load movie of lipids diffusing, bandpass, and subtract the background
Find the spots in every frame and make array of this [x,y,frame]
To parallelize this loop,
Track these results
Convert these, yet again, for the various dx and MSD programs to
Find colocalization with NPs
Find stuck particles and make separated stuck and mobile traj8Col arrays
Rewrite traj8Col in terms of \( \mu m \)
Write track files
To parallelize the following,
Calculate displacement vectors
Calculate residence times (of nanoparticles) and generate statistics and histograms (and rewrite in terms of ms)
Calculate MSD and rewrite in terms of ms
Calculate individual MSD using Tinevez’s functions and make the plots for them
Calculate turning angles
Make figures of calculated results
Save report

Function [ traj8Col, nps2Col ] = loadAndTrackParallel( inputDir, outputDir, makePlots, constants )

%LOADANDTRACK - Loads movies of diffusing particles and does some analysis
% This is the main program that loads all movies, detects and tracks
% particles within them, then calculates MSD, turning angle, displacement
% histograms, etc for the tracks founds and makes pretty figures showing
% what was found.
% INPUT:
% inputDir is a directory containing (usually) individual directories
% with images and movies of nanoparticles, diffusing
% particles, etc.
% outputDir is the directory where the results are to be written. If
% it doesn’t exist, it will be created. This directory will
% be populated with subdirectories containing the results of
% each analyzed scene (nanoparticle/diffusing particle pair)
% makePlots determines whether figures are made of the results (1) or
% not (0).
% constants is a struct containing parameters specific to the
% captured images, such as:
% constants.umPx = the size of a square pixel’s side in \( \mu m \).
% constants.msInterval = the interval between frames in ns.
% OUTPUT:
% traj8Col is [x position, y position, particle intensity, squared
% radius of gyration, zero?, frame number, track number,
% colocalized]
% nps2Col is [x position, y position]
%
% PC 2013-11-12 - Started this beast
% PC 2014-02-25 - better file handling: just give a directory name and it
% finds TIFFs inside it, as a bunch of files or as a
% multiframe TIFF.
% For NPs, movies are averaged (mean intensity projection)
% PC 2014-02-06 - even better file handling. tries to discover the map of
% nanoparticles to diffusing particles and load the
Define constants and such
(I’d love to scrape this from the TIFF metadata, but it’s so often inconsistent and would require lots of sanity checking...) Below are sane values.

```matlab
if exist('constants','var')
    % Test supplied values for reasonableness
else
    % The size of a square pixel’s side in µm. Default is 0.107
    constants.umPx = 0.107;
    % The interval between frames in ms. Note that this is NOT the exposure
    % time
    constants.msInterval = 45.6;
end
```

Distributed Computing
See if parallel computing is set up and use it if it is. This may be overly biased to the HPC cluster at DU.

```matlab
parallel = 1;
if parallel
    if isParallelPossible
        % Initialize the cluster and get the details. This needs work.
        clusterDetails = parClusterInit;
        fprintf('Parallel Computing configured. Using cluster blah with num nodes.
');
        parallel = 1;
    else
        fprintf('Parallel Computing not configured. Use loadAndTrack2.m...
');
        parallel = 0;
    end
end
```

Condition reporting
Use the blink1 to report success, errors, etc

```matlab
if eq(exist('blink1','file'),2)
    blinkExists = 1;
    blink1('off');
else
    blinkExists = 0;
end
```

Make new working directory for this analysis
Analyses of individual scenes will go in subdirectories of this

```matlab
warning('off','MATLAB:MKDIR:DirectoryExists');
mkdirSuccess = mkdir(outputDir);
if eq(mkdirSuccess,0)
    error('Failed to create working directory')
end
%cd(outputDir);
```
Load the map of nanoparticles to diffusing particles

```matlab
channelList = discoverChannels2(inputDir);
if and(length(channelList),0)
    error('No channel list found');
end
numChannels = size(channelList,2);
numScenes = length(channelList{1,1});
fprintf('Channel detection: %d channels detected, %d scene(s) detected
',numChannels,numScenes);
```

The main loop

This will require some restructuring to use parfor. How to load files from within the node?

```matlab
for sceneNum = 1:numScenes
```

What's going on?

```matlab
fprintf('
Scanning scene %s (%d/%d):
',channelList{1,2}{sceneNum,1},sceneNum,numScenes);
```

New output subdirectory for scene

```matlab
sceneDir = fullfile(outputDir,channelList{1,2}{sceneNum,1});
warning('off','MATLAB:MKDIR:DirectoryExists');
mkdirSuccess = mkdir(sceneDir);
if and(mkdirSuccess,0)
    error('Failed to create scene directory.');
end
```

Load static image of nanoparticles

```matlab
nps = loadTIFFs(fullfile(inputDir,channelList{1,1}{sceneNum,1}));
```

Make a mean intensity projection if TIFF is multi-frame.

Return a double either way.

```matlab
if gt(size(nps,3),1)
    nps = mean(nps,3);
else
    nps = double(nps);
end
```

Localize the particles

bandpass between 1 and 7 pixels. (This depends on the optics. Correct if needed)

```matlab
npsBP = bpass(nps,1,7);
clear nps;

% Find pixel-level positions with intensity above 500 and separation of 5/2
npsPosRough = pkfnd(npsBP,100,5);
warning off;
hold off;colormap('gray');imshow(npsBP,[]);hold on;plot(npsPosRough(:,1),npsPosRough(:,2),'r');
% Find subpixel-level positions
npsPosFine = cntrd(npsBP,npsPosRough,9);
clear npsBP;

% Don't try working with empty sets of NPs...
if gt(size(npsPosFine,2),0)
    nps2Col = npsPosFine(:,1:2);
    emptyNPset = 0;
else
    nps2Col = [];%printf('No nanoparticles were found...
    emptyNPset = 1;
end

% Save these for colocalization...
if writeFiles % This was originally an argument to the function, but it
    % doesn't hurt to leave it on. Simplify, simplify, simplify.
    save(fullfile(sceneDir,'NPs.mat'), 'nps2Col');
csvwrite(fullfile(sceneDir,'NPs.csv'),nps2Col);
end
```

% Write the number of NPs in the report, too.
Report npCount = size(nps2Col,1);

Load movie of lipids diffusing, bandpass, and subtract the background

fprintf('Loading files... \n');
liptds = loadTIFFs(fullfile(inputDir,channelList{1,2}{sceneNum,1}));
fprintf('Subtracting background... \n');
liptsBgSub = bgSubByBpass(liptds);

% A little memory management
clear liptds;

Find the spots in every frame and make array of this [x,y,frame]

fprintf('Detecting particles... ');
liptdPosAll = []; % GUI fic progress bar. Super annoying with huge batches, so disabled
% Heres a soothing text progress bar in its place...
% reverseStr = '';
% h = waitbar(0,'detecting particles...');
% spotless = 0;
% numberFrames = size(lipidsBgSub,3);

To parallelize this loop,

I need to restructure how I deal with the position lists. I cant cat() inside the loop, but I can build the lists inside and cat() once the loop is done.

for i = 1:numberFrames
% more GUI fic progress bar. Super annoying with huge batches, so disabled
% h = waitbar(percDone/100,h,sprintf('Detecting particles... %0.0f%%',percDone));
% percDone = i/numberFrames*100;
% progress = sprintf('%3.1f',percDone);
% fprintf('%s%%',[reverseStr,progress]);
% reverseStr = repmat(sprintf(''), 1, length(progress) + 1);
lipidFrame = lipidsBgSub(:,:,i);
% lipidFrameBP = bpass(lipidFrame,1,4);
lipidFramePosRough = pkfnd(lipidFrame,100,3);
lipidFramePosFine = cntrd(lipidFrame,lipidFramePosRough,9);
numSpotsFound = size(lipidFramePosFine,1);
%clear lipidFrame;

% Skip this frame if no spots are found. This could be a bad sign and should
% be dealt with better.
if eq(numSpotsFound,0)
  spotless = 1;
  fprintf('Warning: Spotless frame found in scene %s, frame %d
',channelList{1,2}{sceneNum,1},i);
  continue
end

index = repmat(i,numSpotsFound,1);
lipidFramePosFineIndex = cat(2,lipidFramePosFine,index);
if eq(i,1)
  lipidPosAll = lipidFramePosFineIndex;
else
  lipidPosAll = cat(1,lipidPosAll,lipidFramePosFineIndex);
end

% A little memory management...
clear lipidFrame;

if gt(spotless,0)
  % Warn if frames without spots were found
  fprintf('\n-- Warning: Spotless frames were found in scene %s, frame %d\n',channelList{1,2}{sceneNum,1},1);
else
  fprintf('\n');
end

% more GUI fic progress bar. Super annoying with huge batches, so disabled
%close(h);
%close all hidden;
% Save positions with all five columns {x,y,brightness,sqRadGyr,frame}
%writeFiles
  save(fullfile(sceneDir,'lipidPos.mat'),'lipidPosAll');
  savewrite(fullfile(sceneDir,'lipidPos.csv'),lipidPosAll);
%end
% Remove columns after [x,y] because track.m doesn't want them
% On second thought, let's keep them. track.m will pass them through
% untouched.
%lipidPos3Col = lipidPosAll(:,1:2);

% Add the last column (frame) because this way seems to be fastest. This
% can't be the best way to do this...
%lipidPos3Col = cat(2,lipidPos3Col,lipidPosAll(:,5));
%clear lipidPosAll;
% Save positions with columns suitable for tracking [x,y,frame]

if writeFiles
%     save('lipidPos3Col.mat','lipidPos3Col');
%end

% Track these results

fprintf('Tracking particles...
');
% The maxdisp argument below (the second argument) is determined by the
% max expected diffusion of the particles. This will need to be scaled
% manually. If an ugly unhandled error crashes in track.m saying
% something about "Excessive Combinatorics", this needs to be reduced.
% I'd love to properly handle that error or detect it and reduce
% maxdisp, but I haven't yet done so. Be warned.
traj6Col = track(lipidPosAll,5,struct('good',5,'mem',1,'dim',2,'quiet',0));
clear lipidPosAll;
% If no tracks are found, continue to the next scene gracefully.
if eq(length(traj6Col),0)
    fprintf('No suitable tracks were found in scene %s.

',channelList{1,2}{sceneNum,1});
    fprintf('Finished with scene %s.

',channelList{1,2}{sceneNum,1});
    continue
else
    fprintf('%d tracks were found in scene %s

',size(traj6Col,1),channelList{1,2}{sceneNum,1});
end

% Convert these, yet again, for the various dx and MSD programs to
% [x,y,brightness,sqRadGyr,zero (blah),frame,track]

traj7Col = traj6Col(:,1:4);
trajLength = size(traj6Col,1);
blah1Col = zeros(trajLength,1);
traj7Col = cat(2,traj7Col,blah1Col,traj6Col(:,5:6));
clear traj6Col;

% Find colocalization with NPs
looking within 2 pixels of NPs. Colocalization report includes stuck particles...FIX THIS.

if ne(emptyNPset,1)
    fprintf('Finding colocalisation events...
');
    % replace the ~ with report.coloc to reenable reports
    [traj8Col, report.coloc] = separatetracks4(traj7Col,nps2Col,2);
else
    fprintf('No nanoparticles found, skipping colocalisation events...
');
    traj8Col = cat(2,traj7Col, zeros(length(traj7Col),1));
end
clear traj7Col;

% Find stuck particles and make separated stuck and mobile traj8Col arrays
Stuck particles are defined by maximum displacement in pixels

FIXME: Use Tinevez's drift parameter to correct all trajectories before proceeding.

% Maximum distance a particle position can move to be considered stuck.
% This number should be based on the single frame localization error.
% TODO: determine this number algorithmically by investigating several
% frames of the movie.
% Alt TODO: use a particle's MSD over the whole movie to determine this.
maxStuckDisp = 5;
% replace the ~ with report.stuck to reenable reports
[ traj8ColStuck, traj8ColMobile, report.stuck ] = separateStuck(traj8Col,maxStuckDisp);

% Rewrite report to exclude stuck particles in (non)colocalization
% counts
report.coloc.colocNonstuck = sum(eq(traj8ColMobile(:,8),1));
report.coloc.noncolocNonstuck = sum(eq(traj8ColMobile(:,8),0));

% % % Instead of making different arrays, I should just make a set of
% % % indices for each condition. That will make analyzing each condition
% % much faster and easier to parallelize.
% Make the set of indices here. We need an index for
% [all,mobile,stuck] [all,coloc,noncoloc], so nine total.
% FINNME: Do this here, instead of later in the MSD block.
trajPropertyIndex = cell(9,1);

Rewrite traj8Col in terms of µm
This uses constants.umPx, which is an argument of, or defined at the beginning of loadAndTrack2 (this function)

traj8ColUm = cat(2, traj8Col(:,1:2) .* constants.umPx, traj8Col(:,3:8));
traj8ColStuckUm = cat(2, traj8ColStuck(:,1:2) .* constants.umPx, traj8ColStuck(:,3:8));
traj8ColMobileUm = cat(2, traj8ColMobile(:,1:2) .* constants.umPx, traj8ColMobile(:,3:8));

Write track files

%if writeFiles
save(fullfile(sceneDir,'lipidTraj.mat'), 'traj8Col', 'traj8ColStuck', 'traj8ColMobile' );
csvwrite(fullfile(sceneDir,'lipidTraj.csv'),traj8Col);
csvwrite(fullfile(sceneDir,'lipidTrajStuck.csv'),traj8ColStuck);
csvwrite(fullfile(sceneDir,'lipidTrajMobile.csv'),traj8ColMobile);
%end

To parallelize the following,
I need to simply set up a massive SPMD structure. Is anything dependent
on anything else?

Calculate displacement vectors
This is clumsy.

fprintf('Calculating displacement vectors...
');

% ...mobile
dx01_mobile_all = getdx5(traj8ColMobileUm,1,2);
dx02_mobile_all = getdx5(traj8ColMobileUm,2,2);
dx05_mobile_all = getdx5(traj8ColMobileUm,5,2);
dx10_mobile_all = getdx5(traj8ColMobileUm,10,2);

dx01_mobile_coloc = getdx5(traj8ColMobileUm,1,0);
dx02_mobile_coloc = getdx5(traj8ColMobileUm,2,0);
dx05_mobile_coloc = getdx5(traj8ColMobileUm,5,0);
dx10_mobile_coloc = getdx5(traj8ColMobileUm,10,0);

% ...stuck
dx01_stuck_all = getdx5(traj8ColStuckUm,1,2);
dx02_stuck_all = getdx5(traj8ColStuckUm,2,2);
dx05_stuck_all = getdx5(traj8ColStuckUm,5,2);
dx10_stuck_all = getdx5(traj8ColStuckUm,10,2);

% ...all
dx01_all_all = getdx5(traj8ColUm,1,2);
Calculate residence times (at nanoparticles) and generate statistics and histograms (and rewrite in terms of ms)

Residence times

fprintf('Calculating residence times...
');

% All particles
report.residtime.all = getresidtime(traj8Col,constants.msInterval);

% All moving particles
report.residtime.mobile = getresidtime(traj8ColMobile,constants.msInterval);

% All stuck particles
report.residtime.stuck = getresidtime(traj8ColStuck,constants.msInterval);

Calculate MSD and rewrite in terms of ms

This is also clumsy. This uses constants.msInterval, which is defined at the beginning of this function (Should we just pass the conversion to msd.m and do the conversion there?)

fprintf('Calculating mean squared displacements...
');

msd_all_all = msd(traj8ColUm,2);
if gt(length(msd_all_all),1)
   msd_all_all = cat(2,msd_all_all(:,1) .* constants.msInterval,msd_all_all(:,2:4));
end

msd_mobile_all = msd(traj8ColMobileUm,2);
if gt(length(msd_mobile_all),1)
   msd_mobile_all = cat(2,msd_mobile_all(:,1) .* constants.msInterval,msd_mobile_all(:,2:4));
end

msd_stuck_all = msd(traj8ColStuckUm,2);
if gt(length(msd_stuck_all),1)
   msd_stuck_all = cat(2,msd_stuck_all(:,1) .* constants.msInterval,msd_stuck_all(:,2:4));
end

msd_all_coloc = msd(traj8ColUm,1);
if gt(length(msd_all_coloc),1)
   msd_all_coloc = cat(2,msd_all_coloc(:,1) .* constants.msInterval,msd_all_coloc(:,2:4));
end

msd_mobile_coloc = msd(traj8ColMobileUm,1);
if gt(length(msd_mobile_coloc),1)
   msd_mobile_coloc = cat(2,msd_mobile_coloc(:,1) .* constants.msInterval,msd_mobile_coloc(:,2:4));
end

msd_stuck_coloc = msd(traj8ColStuckUm,1);
if gt(length(msd_stuck_coloc),1)
   msd_stuck_coloc = cat(2,msd_stuck_coloc(:,1) .* constants.msInterval,msd_stuck_coloc(:,2:4));
end
msd_all_noncoloc = msd(traj8ColUm,0);
if gt(length(msd_all_noncoloc),1)
    msd_all_noncoloc = cat(2,msd_all_noncoloc(:,1).*constants.msInterval,msd_all_noncoloc(:,2:4));
end

msd_mobile_noncoloc = msd(traj8ColMobileUm,0);
if gt(length(msd_mobile_noncoloc),1)
    msd_mobile_noncoloc = cat(2,msd_mobile_noncoloc(:,1).*constants.msInterval,msd_mobile_noncoloc(:,2:4));
end

msd_stuck_noncoloc = msd(traj8ColStuckUm,0);
if gt(length(msd_stuck_noncoloc),1)
    msd_stuck_noncoloc = cat(2,msd_stuck_noncoloc(:,1).*constants.msInterval,msd_stuck_noncoloc(:,2:4));
end

%if writeFiles
save(fullfile(sceneDir,'msd_all.mat'],'msd_all_all','msd_mobile_all','msd_stuck_all');
save(fullfile(sceneDir,'msd_coloc.mat'),'msd_all_coloc','msd_mobile_coloc','msd_stuck_coloc');
save(fullfile(sceneDir,'msd_noncoloc.mat'),'msd_all_noncoloc','msd_mobile_noncoloc','msd_stuck_noncoloc');
%end

Calculate individual MSD using Tinevez's functions and make the plots for them
ref here

% Set up the options for the conversion. Some of this is already done
% in traj8ColUm, but it's fast so we'll repeat it here.
options.dSpace = constants.umPx;
options.dTime = constants.msInterval / 1000;

% Make a set of indices here. We need an index for
% [all,mobile,stick][all,coloc,noncoloc], so nine total.
% FIXME: Do this above, instead of here in the MSD block.
% The indices are:
% 1: all,all  2: all,coloc  3: all,noncoloc
% 4: mobile,all 5: bobile,coloc 6: mobile,noncoloc
% 7: stuck,all  8: stuck,coloc  9: stuck,noncoloc

trajPropertyIndex = cell(9,1);

% Make the indices. I'll make them from the existing arrays, since
% that's quickest right now. Ultimately, they should be made when the
% arrays are made.
% all=~isnan, coloc=col8 is 1, noncoloc=col8 is 0
%
% FIXME - these are indices of steps, not tracks. This won't work with
% things like coloc/noncoloc, which are smaller than entire tracks.
trajPropertyIndex{1,1} = ~isnan(traj8Col(:,8));
trajPropertyIndex{2,1} = eq(traj8Col(:,8),1);
trajPropertyIndex{3,1} = eq(traj8Col(:,8),0);
trajPropertyIndex{4,1} = ~isnan(traj8ColMobile(:,8));
trajPropertyIndex{5,1} = eq(traj8Col(:,8),1);
trajPropertyIndex{6,1} = eq(traj8Col(:,8),0);
trajPropertyIndex{7,1} = ~isnan(traj8ColStuck(:,8));
trajPropertyIndex{8,1} = eq(traj8Col(:,8),1);
trajPropertyIndex{9,1} = eq(traj8Col(:,8),0);

% Convert the trajectory array into a cell array to use Tinevez's
% function. His format is nice and clean and is easier to program
% around but precludes many vectorized operations and makes processing
% slow.... Oh well.
[trajCell,cellIndex] = trajToTrajsCell(traj8ColMobile, options);

% Set up the msd object with 2 dimensions and units of µm and s.
msdObj = msdanalyzer(2,['µm','s']);

% Add the trajectories to the msd object - Add every particle now and
divide them up later with indices
msdObj = msdObj.addAll(trajCell);

% Calculate the drift in the images from the correlated velocities of
% all particles. It's useful to have stuck and mobile particles here
% for better results.
msdObj = msdObj.computeDrift('velocity');

% Here is where I should correct for drift in the trajectories.
% FIXME
% Do the computations for MSD and velocity autocorrelation
% Do this in parallel if possible FIXME: make this parfor safe
msdObj = msdObj.computeMSD;  % (trajPropertyIndex{1,1});
%msdObj = msdObj.computeVCorr;

% Extract the MSDs for each index (these are cell arrays)
% Only [1,5,7] work now. No coloc/noncoloc data here...
%for i = [1,5,7]
%    msd{i,1} = msdObj.msd{trajPropertyIndex{i,1},1};
%end
% Make a plot of the MSDs
%msdIndPlot = msdObj.plotMSD;
% Find the bulk MSD
%msdBulk = msdObj.getMeanMSD;
% Save the various results
%save(fullfile(sceneDir,'msd.mat'),'msdObj','msdBulk');
%save(fullfile(sceneDir,'msd_coloc.mat'),'msd_all_coloc','msd_mobile_coloc','msd_stuck_coloc');
%save(fullfile(sceneDir,'msd_noncoloc.mat'),'msd_all_noncoloc','msd_mobile_noncoloc','msd_stuck_noncoloc');

Calculate turning angles
Same method of other calculation sections ...

fprintf('Calculating turning angles...
');
turn_all_all = turnangle(traj8ColUm,2);
turn_stuck_all = turnangle(traj8ColStuckUm,2);
turn_mobile_all = turnangle(traj8ColMobileUm,2);
turn_all_coloc = turnangle(traj8ColUm,1);
turn_stuck_coloc = turnangle(traj8ColStuckUm,1);
turn_mobile_coloc = turnangle(traj8ColMobileUm,1);
turn_all_noncoloc = turnangle(traj8ColUm,0);
turn_stuck_noncoloc = turnangle(traj8ColStuckUm,0);
turn_mobile_noncoloc = turnangle(traj8ColMobileUm,0);
%if writeFiles
%save(fullfile(sceneDir,'turn_all.mat'),'turn_all_all','turn_mobile_all','turn_stuck_all');
%save(fullfile(sceneDir,'turn_coloc.mat'),'turn_all_coloc','turn_mobile_coloc','turn_stuck_coloc');
%save(fullfile(sceneDir,'turn_noncoloc.mat'),'turn_all_noncoloc','turn_mobile_noncoloc','turn_stuck_noncoloc');
%end

Make figures of calculated results
Once again... clumsy.

if makePlots
fprintf('Generating figures...
');
fprintf(' (displacement histograms)...');
% The displacement histograms
% ... mobile
plotHistograms(dx01_mobile_all,0,struct('dirName',sceneDir,'prefix','dx01-mobile-all'));
plotHistograms(dx02_mobile_all,0,struct('dirName',sceneDir,'prefix','dx02-mobile-all'));
plotHistograms(dx05_mobile_all,0,struct('dirName',sceneDir,'prefix','dx05-mobile-all'));
plotHistograms(dx10_mobile_all,0,struct('dirName',sceneDir,'prefix','dx10-mobile-all'));
plotHistograms(dx01_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx01-mobile-coloc'));
plotHistograms(dx02_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx02-mobile-coloc'));
plotHistograms(dx05_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx05-mobile-coloc'));
plotHistograms(dx10_mobile_coloc,0,struct('dirName',sceneDir,'prefix','dx10-mobile-coloc'));
plotHistograms(dx01_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx01-mobile-noncoloc'));
plotHistograms(dx02_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx02-mobile-noncoloc'));
plotHistograms(dx05_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx05-mobile-noncoloc'));
plotHistograms(dx10_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','dx10-mobile-noncoloc'));
% ... stuck
plotHistograms(dx01_stuck_all,0,struct('dirName',sceneDir,'prefix','dx01-stuck-all'));
plotHistograms(dx02_stuck_all,0,struct('dirName',sceneDir,'prefix','dx02-stuck-all'));
plotHistograms(dx05_stuck_all,0,struct('dirName',sceneDir,'prefix','dx05-stuck-all'));
plotHistograms(dx10_stuck_all,0,struct('dirName',sceneDir,'prefix','dx10-stuck-all'));
plotHistograms(dx01_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx01-stuck-coloc'));
plotHistograms(dx02_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx02-stuck-coloc'));
plotHistograms(dx05_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx05-stuck-coloc'));
plotHistograms(dx10_stuck_coloc,0,struct('dirName',sceneDir,'prefix','dx10-stuck-coloc'));

169
plotHistograms(dx01_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx01-stuck-noncoloc'));
plotHistograms(dx02_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx02-stuck-noncoloc'));
plotHistograms(dx05_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx05-stuck-noncoloc'));
plotHistograms(dx10_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','dx10-stuck-noncoloc'));

% ...all
plotHistograms(dx01_all_all,0,struct('dirName',sceneDir,'prefix','dx01-all-all'));
plotHistograms(dx02_all_all,0,struct('dirName',sceneDir,'prefix','dx02-all-all'));
plotHistograms(dx05_all_all,0,struct('dirName',sceneDir,'prefix','dx05-all-all'));
plotHistograms(dx10_all_all,0,struct('dirName',sceneDir,'prefix','dx10-all-all'));

plotHistograms(dx01_all_coloc,0,struct('dirName',sceneDir,'prefix','dx01-all-coloc'));
plotHistograms(dx02_all_coloc,0,struct('dirName',sceneDir,'prefix','dx02-all-coloc'));
plotHistograms(dx05_all_coloc,0,struct('dirName',sceneDir,'prefix','dx05-all-coloc'));
plotHistograms(dx10_all_coloc,0,struct('dirName',sceneDir,'prefix','dx10-all-coloc'));

plotHistograms(dx01_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx01-all-noncoloc'));
plotHistograms(dx02_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx02-all-noncoloc'));
plotHistograms(dx05_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx05-all-noncoloc'));
plotHistograms(dx10_all_noncoloc,0,struct('dirName',sceneDir,'prefix','dx10-all-noncoloc'));

% The MSD plots
% (Trying out the nifty conditional operator && to avoid making
%  more stupid if...end loops around everything. If there is only
%  one element in any of these, then it is almost certainly a single
%  NaN. The plot function doesn't handle that situation yet.)
% fprintf('(MSD plots)...');

gt(length(msd_all_all),1) && plotMSD(msd_all_all,0,struct('dirName',sceneDir,'prefix','msd-all-all'));

gt(length(msd_mobile_all),1) && plotMSD(msd_mobile_all,0,struct('dirName',sceneDir,'prefix','msd-mobile-all'));

gt(length(msd_stuck_all),1) && plotMSD(msd_stuck_all,0,struct('dirName',sceneDir,'prefix','msd-stuck-all'));

gt(length(msd_all_coloc),1) && plotMSD(msd_all_coloc,0,struct('dirName',sceneDir,'prefix','msd-all-coloc'));

gt(length(msd_mobile_coloc),1) && plotMSD(msd_mobile_coloc,0,struct('dirName',sceneDir,'prefix','msd-mobile-coloc'));

gt(length(msd_stuck_coloc),1) && plotMSD(msd_stuck_coloc,0,struct('dirName',sceneDir,'prefix','msd-stuck-coloc'));

gt(length(msd_all_noncoloc),1) && plotMSD(msd_all_noncoloc,0,struct('dirName',sceneDir,'prefix','msd-all-noncoloc'));

gt(length(msd_mobile_noncoloc),1) && plotMSD(msd_mobile_noncoloc,0,struct('dirName',sceneDir,'prefix','msd-mobile-noncoloc'));

gt(length(msd_stuck_noncoloc),1) && plotMSD(msd_stuck_noncoloc,0,struct('dirName',sceneDir,'prefix','msd-stuck-noncoloc'));

% The turning angle plots
fprintf('(turning angle plots)...');

gt(length(turn_all_all),1) && plotTurnAngles(turn_all_all,0,struct('dirName',sceneDir,'prefix','turn-all-all'));

gt(length(turn_mobile_all),1) && plotTurnAngles(turn_mobile_all,0,struct('dirName',sceneDir,'prefix','turn-mobile-all'));

gt(length(turn_stuck_all),1) && plotTurnAngles(turn_stuck_all,0,struct('dirName',sceneDir,'prefix','turn-stuck-all'));

gt(length(turn_all_coloc),1) && plotTurnAngles(turn_all_coloc,0,struct('dirName',sceneDir,'prefix','turn-all-coloc'));

gt(length(turn_mobile_coloc),1) && plotTurnAngles(turn_mobile_coloc,0,struct('dirName',sceneDir,'prefix','turn-mobile-coloc'));

gt(length(turn_stuck_coloc),1) && plotTurnAngles(turn_stuck_coloc,0,struct('dirName',sceneDir,'prefix','turn-stuck-coloc'));

% The intensity vs dx plots
fprintf('(intensity scatter plots)...');

gt(length(dxIntAll),1) && plotScatter(dxIntAll,0,struct('dirName',sceneDir,'prefix','dxIntAll'));

gt(length(dxIntMobile),1) && plotScatter(dxIntMobile,0,struct('dirName',sceneDir,'prefix','dxIntMobile'));

gt(length(dxIntStuck),1) && plotScatter(dxIntStuck,0,struct('dirName',sceneDir,'prefix','dxIntStuck'));

Save report
For now, this is just a .mat with some numbers in it. Maybe make a nice report later...

save(fullfile(sceneDir,'/','report.mat'),'report','constants');

fprintf('
Finished with scene %s.


',channelList{1,2}{sceneNum,1});
fprintf('Done\n');
if blinkExists
    blink1('on','green');
end

end
% load tiff stack and determine number of frames
stack = TIFFStack('CsA 01.stk');
num_frames = size(stack,3);
%imshow(frame001stk(:,:,1)) % show first frame of stack
particles = double([',',',',']);
for count = 1:num_frames
    % copy each frame into its own double var
    frame = double(stack(:,:,count));
    %whos frame001
    % find peaks
    pk = pkfnd(frame,20000,5);
    % add sequence number to peak list
    pks_found = size(pk,1);
    index = zeros(pks_found,1);
    index(:) = count-1;
    pk_list = cat(2,pk,index);
    % adjust peaks to centroids (disappointing, so disabled)
    %cnt = cntrd(frame,pk,3);
    particles = cat(1,particles,pk_list);
end
% tracking
param = struct('mem',10,'good',5,'dim',2,'quiet',0);
tr = track(particles,5,param);
% saving
save('analysis.mat');

% plotting
hold off;
colormap('gray'),imagesc(frame);
hold on;
plot(pk(:,1),pk(:,2),'r.');
%plot(cnt(:,1),cnt(:,2),'g.');
linenum = 1;
tracksfound = max(tr(:,4));
for tracknum = 1:tracksfound
    xtraj = double.empty;
    ytraj = double.empty;
while ~(tracknum < tr(linenum,4))
    xtraj = cat(1,xtraj,tr(linenum,1));
    ytraj = cat(1,ytraj,tr(linenum,2));
    linenum = linenum + 1;
end

plot(xtraj,ytraj,'y-')
end
function [ TIFFstack ] = loadTIFFs( directory )
%loadTIFFs Loads all TIFF files in 'directory' as a 3D matrix
% For a given directory, all TIFFs inside are loaded as a 3D matrix.
% Multiple files are handled, as are multi-frame TIFFs.
%
% TODO:
% * Multiple multi-frame TIFFs are not handled (not hard to fix, if needed).
% * Files should be named *.tif, though a general case wouldn't be hard to
%   set up either.
% * Better error handling
%
% What are the contents of the directory?
%dirContents = dir(strcat(directory,'/*.tif'));
dirContents = dir(fullfile(directory,'*.tif'));

% Turn off annoying tifflib warnings
warning('off','MATLAB:imagesci:tiffmexutils:libtiffWarning');

% How many files are there? Act appropriately...
if gt(size(dirContents,1),1)
    % There is more than one file here. Load a movie from individual files.
    numberFrames = size(dirContents,1);
    % Assume the all of the images are the same size
    % (Really should check this and bail if they're not...)
    %file = strcat(directory,'/','dirContents(1,1).name);
    file = fullfile(directory,dirContents(1,1).name);
    infoImage = imfinfo(file);
    mImage = infoImage(1).Width;
    nImage = infoImage(1).Height;
    TIFFstack = zeros(nImage,mImage,numberFrames,'uint16');
    for i = 1:numberFrames
        %file = strcat(directory,'/','dirContents(i,1).name);
        file = fullfile(directory,dirContents(i,1).name);
        TifLink = Tiff(file,'r');
        TifLink.setDirectory(1);
        TIFFstack(:,:,i) = TifLink.read();
    end
    TifLink.close();
elseif eq(size(dirContents,1),1)
    % There is one file here. Load as if a multi-frame TIFF. A single-frame
    % TIFF will load just fine in this manner, too.
    %file = strcat(directory,'/','dirContents(1,1).name);
    file = fullfile(directory,dirContents(1,1).name);
    % Responsibly load our TIFF stack
    infoImage = imfinfo(file);
    mImage = infoImage(1).Width;
    nImage = infoImage(1).Height;
    numberFrames = length(infoImage);
    TIFFstack = zeros(nImage,mImage,numberFrames,'uint16');
    TifLink = Tiff(file,'r');
    for i = 1:numberFrames
        TifLink.setDirectory(i);
        TIFFstack(:,:,i) = TifLink.read();
    end
    TifLink.close();
else
    % There is no file here. Do nothing.
end
TIFFstack(:,:,i) = TifLink.read();
end
TifLink.close();

elseif eq(size(dirContents,1),0)
    % There are no TIFF files here. Issue an error.
    fprintf('The directory supplied has no .tif images');
    return
else
    % Something went horribly wrong. Issue a more serious error.
    fprintf('There is something very wrong with the directory supplied. Panic.');
    return
end
end
function msd = msd( traj8Col, coloc )
%MSD -- find mean squared displacement of trajectories (all, colocalized, or
%noncolocalized)
%
% This program calculates msd from a track array. The
% output of the program is an array of mean squared displacements for
% each successive dt up to 1/4 the longest track length
%
% INPUT VARIABLES:
% traj8Col is a track file [x,y,blah,blah,blah,frame,track,coloc]
% coloc determines whether to look at only colocalized (coloc=1) or
% non colocaled (coloc=0) or all (coloc=2, default)
%
% OUTPUT:
% msd is a matrix containing [dt,<r^2>,SD,N]
% (BE WARNED: this function may return a NaN for some elements in
% certain situations. Be prepared to handle them!)
%
% PC ported from IDL file from Michelle Knowles 2013-10-22
% PC adjusted to strip stuck particles 2014-01-30
% PC moved stuck loop out to its own function 2014-05-07
% Maybe I should make this an argument, but for now, let’s just set it
% here. Suppress informative outputs? (but not errors)
quiet = 0;

% Allowable values for coloc are 0,1,2. Anything else is treated as 2.
if and((ne(coloc,0)),(ne(coloc,1)))
coloc = 2;
end

% Make index of coloc or non and NaN out the non-relevant tracks
if eq(coloc,0)
    noncolocIndex = find(eq(traj8Col(:,8),0));
    traj8Col(noncolocIndex,:) = nan(length(noncolocIndex),8);
elseif eq(coloc,1)
    colocIndex = find(eq(traj8Col(:,8),1));
    traj8Col(colocIndex,:) = nan(length(colocIndex),8);
end

% Cleanup time is over! Let’s find some mean square displacements!
% Find length of longest track
longestTrack = max(traj8Col(:,6));

% Can we work with the tracks given?
if isnan(longestTrack)
    if eq(quiet,0)
        fprintf('No tracks present. Cannot calculate MSD for this dt.
        end
        msd = nan(1);
        return
    elseif lt(longestTrack,4)
        if eq(quiet,0)
            fprintf('Too few tracks present. Cannot calculate MSD for this dt.
        end
        msd = nan(1);
        return
    else
        max dt is 1/4 longest track (Saxon, 1997) % Using 1/2 for now
        maxdt = longestTrack / 2;
        logMaxdt = log10(maxdt);
        % Initialize index of dts
        msd = cat(2,unique(round(logspace(0,logMaxdt,100)'),zeros(length(unique(round(logspace(0,logMaxdt,100)'),)),2)));
        % Shift through all dts and get msd for each
for dtCount = 1:length(msd)
    % Shift the array of trajectories up dt steps
    shiftTrajCol = circshift(trajCol,-msd(dtCount,1));

    % Find the difference between the track starting points over dt
    dx = shiftTrajCol(:,1:2) - trajCol(:,1:2);

    % Find the squared magnitude of the displacement
    dx(:,3) = dx(:,1).^2 + dx(:,2).^2;

    % Find where in the array new tracks start
    trackIndex = find(ne(shiftTrajCol(:,7),trajCol(:,7)));

    % Remove the displacements that correspond to new tracks starting
    dx(trackIndex,:) = nan(length(trackIndex),3);

    % add code here to trim NaNs from array (do we want/need to?)

    % Calculate the MSD, SD, and the number of samples (uses Statistics Toolbox)
    msd(dtCount,2) = nanmean(dx(:,3));
    msd(dtCount,3) = nanstd(dx(:,3));
    msd(dtCount,4) = length(dx(:,3)) - length(find(isnan(dx(:,3))));
end
end

Published with MATLAB® R2015a
function msd = msdIndividual( traj8Col, coloc )

%MSD -- find mean squared displacement of individual tracks (all, colocalized, or 
% noncolocalized) and Dsm (single molecule diffusion coefficient)
% This program calculates msd from a track array. The 
% output of the program is an array of mean squared displacements for 
% each successive dt up to 1/4 the longest track length
% INPUT VARIABLES:
% traj8Col is a track file [x,y,blah,blah,blah,frame,track,coloc]
% coloc determines whether to look at only colocalized (coloc=1) or 
% non colocalized (coloc=0) or all (coloc=2, default)
% OUTPUT:
% trajMSD is the original track file with two extra columns appended. 
% These are [ traj8Col... , dt, <r^2> ], where each individual msd 
% starts where the relevant track begins and has appx 1/4 the 
% elements as there are steps in the track.
% %msd is a matrix containing [dt,<r^2>,SD,N] 
% (BE WARNED: this function may return a NaN for some elements in 
% certain situations. Be prepared to handle them!)
% PC ported from IDL file from Michelle Knowles 2013-10-22 
% PC adjusted to strip stuck particles 2014-01-30 
% PC moved stuck loop out to its own function 2014-05-07
% This whole function needs to be rewritten from scratch.
% The plan: Walk through the track numbers. Calculate all possible MSDs 
% for each track and write them to the traj8Col matrix.
%
% Maybe I should make this an argument, but for now, let’s just set it 
% here. % Suppress informative outputs? (but not errors)
quiet = 0;
% Allowable values for coloc are 0,1,2. Anything else is treated as 2.
if and((ne(coloc,0)),(ne(coloc,1)))
   coloc = 2;
end
% Make index of coloc or non and NaN out the non-relevant tracks 
if eq(coloc,0)
   noncolocIndex = find(eq(traj8Col(:,8),0));
   traj8Col(noncolocIndex,:) = nan(length(noncolocIndex),8);
elseif eq(coloc,1)
   colocIndex = find(eq(traj8Col(:,8),1));
   traj8Col(colocIndex,:) = nan(length(colocIndex),8);
else 
   traj8Col(:,8) = nan(length(traj8Col(:,8)),1);
end
%
% Cleanup time is over! Let’s find some mean square displacements!
%
% Find length of longest track.
%longestTrack = max(traj8Col(:,6));
% % % This isn’t relevent for individual MSD calculation. (Well, the second 
% % % test isn’t...) 
% Can we work with the tracks given?
if isnan(longestTrack)
   if eq(quiet,0)
      fprintf('No tracks present. Cannot calculate MSD for this dt.\n');
   end
   msd = nan(1);
   return
elseif lt(longestTrack,4)
   if eq(quiet,0)
      fprintf('Too few tracks present. Cannot calculate MSD for this dt.\n');
   end
end
msd = nan(1);
    return
else
end

% Max dt is 1/4 longest track (Saxon, 1997)
% For indiv msd, we’ll calculate up to dt = tracklength / 4
maxdt = longestTrack / 4;
logMaxdt = log10(maxdt);

% Initialize index of dts
msd = cat(2,unique(round(logspace(0,logMaxdt,100)')),zeros(length(unique(round(logspace(0,logMaxdt,100)'))),2));

% Find tracks that are not NaN’d out in traj8Col
tracksPresent = unique(traj8Col(:,7));
tracksPresent = tracksPresent(~isnan(tracksPresent));

% Shift through all tracks and get msd for each
for trackNum = tracksPresent
    % Limit the scope to the current track
    currentTrackMask = eq(traj8Col(:,7),trackNum);
    % Find the length of the track
    trackLength = length(traj8Col(currentTrackMask,1));
    % The max dt is 1/4 longest track (Saxon, 1997)
    maxdt = trackLength / 4;
    for currentdt = 1:maxdt
        % Shift the array of trajectories up dt steps
        shiftTraj8Col = circshift(traj8Col,-msd(dtCount,1));
        % Find the difference between the track starting points over dt
        dx = shiftTraj8Col(:,1:2) - traj8Col(:,1:2);
        % Find the squared magnitude of the displacement
        dx(:,3) = dx(:,1).^2 + dx(:,2).^2;
        % Find where in the array new tracks start
        trackIndex = find(ne(shiftTraj8Col(:,7),traj8Col(:,7)));
        % Remove the displacements that correspond to new tracks starting
        dx(trackIndex,:) = nan(length(trackIndex),3);
        % Add code here to trim NaNs from array (do we want/need to?)
        % Calculate the MSD, SD, and the number of samples (uses Statistics Toolbox)
        msd(dtCount,2) = nanmean(dx(:,3));
        msd(dtCount,3) = nanstd(dx(:,3));
        msd(dtCount,4) = length(dx(:,3)) - length(find(isnan(dx(:,3))));
    end
end
end

Published with MATLAB® R2015a
function [ clusterDetails ] = parClusterInit ()

% Take the requested cluster attributes and set it up
% Return the details in a struct

% What do we have to work with?
classDetails.defaultCluster = parallel.defaultClusterProfile;
classDetails.availableClusters = parallel.clusterProfiles;

end
function parsave(fname, x,y)

save(fname, x, y)

end
function [ histogram ] = plotHistograms( dxInput, display, outputDir )

% plotHistograms takes a bunch of displacements and makes a histogram
% plotHistograms makes histograms and shows them or saves them or simply
% eats them.

% INPUT:
%   dx is a list of displacements in the form of [dx, dy, dr], where dr
%   is the magnitude of the total displacement.
%   display determines whether the histograms are shown on the screen
%   (display=1) or hidden (display=0).
%   outputDir determines where the histogram images are saved. If this
%   is not set, the histograms are not saved. This argument should
%   be a struct containing the directory name (outputDir.dirName)
%   and any prefix (string) to be appended to the filename
%   (outputDir.prefix).

% OUTPUT: This function displays or saves images of histograms.
% The generated fig will be returned

% Is an outputDir argument set? If not, skip the save step
if eq(nargin,2)
    saveFiles = 0;
    outputDir.prefix = 'histogram';
else
    saveFiles = 1;
end

% Make figure and plot it
if display
    histogramFig = figure('visible','on');
else
    histogramFig = figure('visible','off');
end

% Make the histogram and plot it. This is in two steps so that we can save
% the histogram values for further analysis.
[nelements,centers] = hist(dxInput(:,3),200);
bar(centers,nelements);

% Make a pretty array for saving the histogram data...
% two columns: centers and nelements
histogram = cat(2,centers',nelements');

% Title the figure with the provided prefix
if exist('outputDir.prefix','var')
    title(outputDir.prefix);
end

% The X-axes here are arbitrary. They could be more rationally determined.
set(gca, 'XLim', [0 3]);
ylabel('Occurrences');
xlabel('Displacement (µm)');

% If an outputDir argument was passed, save the generated file in that place
if saveFiles
    save file
    hgexport(gcf, fullfile(outputDir.dirName,[outputDir.prefix '-dx.png']), hgexport('factorystyle'), 'Format', 'png');
csvwrite(fullfile(outputDir.dirName,[outputDir.prefix '-histogram.csv']),histogram);
end

% We're done with the current figure
close;
function [ msdFig ] = plotMSD( msd, display, outputDir )
%plotMSD Takes a bunch of MSDs and makes a histogram
% plotMSD makes plots of MSD vs dt and shows them or saves them or simply
% eats them.

% INPUT:
%   msd is a matrix containing [dt,<r^2>,SD,N]
%   display determines whether the plots are shown on the screen
%   (display=1) or hidden (display=0).
%   outputDir determines where the plot images are saved. If this
%   is not set, the plots are not saved. This argument should
%   be a struct containing the directory name (outputDir.dirName)
%   and any prefix (string) to be appended to the filename
%   (outputDir.prefix).

% OUTPUT: This function displays or saves images of MSD plots.
% The generated fig will be returned

% Is an outputDir argument set? If not, skip the save step
if eq(nargin,2)
    saveFiles = 0;
else
    saveFiles = 1;
end

% Make figure and plot it
if display
    msdFig = figure('visible','on');
else
    msdFig = figure('visible','off');
end

% MSD gets weird at larger taus, both from real effects and from bias in
% the long-lived datapoints, so let's only plot a certain range of taus.
% Here, we're only plotting to 500 ms
maxTau = 500;
maxTauMask = msd(:,1)<=maxTau;

% To plot the entire set of MSDs, replace 'maxTauMask' with ':'
plot(msd(maxTauMask,1),msd(maxTauMask,2));

% Y-axes here are arbitrary. They could be more rationally determined.
set(gca,'XLim',[0 3]);
ylabel('mean squared displacement (\mu m^2/s)');
xlabel('interval (ms)');

% Title the figure with the provided prefix
if exist('outputDir.prefix','var')
    title(outputDir.prefix);
end

% If an outputDir argument was passed, save the generated file in that place
if saveFiles
    % save file
    hgexport(gcf, fullfile(outputDir.dirName,[outputDir.prefix '-msd.png'])), hgexport('factorystyle', 'Format', 'png');
end

% We're done with the current figure
if eq(display,0)
    close;
end

end
function [ ] = plotScatter( dxInput, display, outputDir )
% plotScatter Takes displacements and intensities and and makes a scatter plot
% plotScatter makes scatter plots and shows them or saves them or simply
% eats them.
%
% INPUT:
%   dx is a list of displacements in the form of [dx, dy, dr, int],
%   where dr is the magnitude of the total displacement and int is
%   intensity is the intensity of a particle
%   display determines whether the scatterplots are shown on the screen
%   (display=1) or hidden (display=0).
%   outputDir determines where the scatterplots images are saved. If this
%   is not set, the scatterplots are not saved. This argument should
%   be a struct containing the directory name (outputDir.dirName)
%   and any prefix (string) to be appended to the filename
%   (outputDir.prefix).
%
% OUTPUT: This function displays or saves images of scatterplots.
% The generated fig will be returned

% Is an outputDir argument set? If not, skip the save step
if eq(nargin,2)
    saveFiles = 0;
    outputDir.prefix = 'histogram';
else
    saveFiles = 1;
end

% Make figure and plot it
if display
    scatterFig = figure('visible','on');
else
    scatterFig = figure('visible','off');
end
plotScatter3D(dxInput(:,3),dxInput(:,4),'.');
smoothhist2D(dxInput(:,3:4),5,[200 200]);
colormap(parula);
axis xy;
% Title the figure with the provided prefix
if exist('outputDir.prefix','var')
    title(outputDir.prefix);
end

% The X-axes here are set by the max x value fed in.
set(gca,'XLim',[0 3]);
ylabel('Intensity (RFU)');
xlabel('displacement (µm)');
scatter = dxInput(:,3:4);

% If an outputDir argument was passed, save the generated file in that place
if saveFiles
    save file
    hgexport(gcf, fullfile(outputDir.dirName,[outputDir.prefix '-IntensityScatter.png'])), hgexport('factorystyle'), 'Form
    writencell(fullfile(outputDir.dirName,[outputDir.prefix '-IntensityScatter.csv'])), scatter;
end

% We’re done with the current figure
%close;
end
function plotTracks (imageDir, analysisDir, minTrackLength)

% Input should be image location, analysis results, minimum track length to display
% Ultimately wrap this whole thing in a GUI that allows redefinition of
% minimum track length, mobile/stuck/all. All = 0, mobile = 1, stuck = 2
% coloring has definitions for coloring of tracks desired
%
Definitions
Right now, we'll just define 'coloring' here

coloring = 5;

Load files

% Is an imageDir set?
if eq(nargin,2)
    plotImageToo = 0;
else
    plotImageToo = 1;
end

% Load static image, if provided
if plotImageToo
    npImage = loadTIFFs(imageDir);
else
    % Handle arguments better, please!
    minTrackLength = analysisDir;
    analysisDir = imageDir;
end

% Load np positions
load(fullfile(analysisDir,'NPs.mat'));

% Load lipid trajectories
load(fullfile(analysisDir,'lipidTraj.mat'));

% Only use mobile particles right now. Make this an argument if interested
% in stuck or all particles.
mobilestuck = 1;
switch mobilestuck
    case 0,
        clear traj8ColMobile traj8ColStuck;
    case 1,
        clear traj8Col traj8ColStuck;
        traj8Col = traj8ColMobile;
    case 2,
        clear traj8ColMobile traj8ColStuck;
        traj8Col = traj8ColStuck;
    otherwise
        error('The allowed options for mobilestuck are 0, 1, or 2');
end

Make a pretty picture

hold off;

if plotImageToo
    % If a pretty NP picture is provided, show it and scale all of the
    % intensities up.
    figure1 = imshow(npImage,(0,(max(npImage(:))*0.5)));
end
else
% If no pretty NP picture is provided, plot the NP positions.
    plot(nps2Col(:,1),nps2Col(:,2),'r.','MarkerSize',10);
end

Separate the tracks into appropriate little chunks

% Trim NaNs from the array
traj8Col = traj8Col(isfinite(traj8Col(:,1)),:);
% Make an index of the last frame of each track
shiftTraj8Col = circshift(traj8Col,-1);
trackIndex = find(ne(shiftTraj8Col(:,7),traj8Col(:,7)));
% Make an array of the length of each track
shiftTrackIndex = circshift(trackIndex,1);
trackIndex(:,2) = trackIndex(:,1) - shiftTrackIndex(:,1);
trackIndex(1,2) = trackIndex(1,1);
% Add a column to make a general index of track numbers.
trackIndex(:,3) = traj8Col(trackIndex(:,1),7);
% Make an index of tracks that are longer than minTrackLength
% This is a list of the track numbers (column 7 of traj8Col) of tracks
% that are longer than minTrackLength. This sets stuff up for a handy
% find() for plotting.
longEnoughTracks = traj8Col(trackIndex(find(trackIndex(:,2)>=minTrackLength)),7);
% Generalize this to the image size
axis([0 512 0 512]);

Handle color and draw tracks that are long enough on the pretty picture

coloring mode to use: 0 = fixed color 1 = tracks are each colored differently 2 = individual displacements are colored by length 3 = first n steps are a different color 4 = "colocalized" displacements are a different color

switch coloring
    case 0
% Every track is the same color.
% Plot the tracks that are long enough. The for loop is slower, but allows
% individual tracks to not be connected.
    for trackToPlotIndex = 1:length(longEnoughTracks)
        pointToPlotIndex = find(eq(traj8Col(:,7),longEnoughTracks(trackToPlotIndex,1)));
        plot(traj8Col(pointToPlotIndex,1),traj8Col(pointToPlotIndex,2), 'Color',[0 0 1]);
    end
    case 1
% Each track is a different color.
% Set to your favorite colormap.
    cmap = hsv(256);
    cmap = parula(256);
    for trackToPlotIndex = 1:length(longEnoughTracks)
        pointToPlotIndex = find(eq(traj8Col(:,7),longEnoughTracks(trackToPlotIndex,1)));
        % Here, we cycle through the colors track-by-track
        plot(traj8Col(pointToPlotIndex,1),traj8Col(pointToPlotIndex,2), 'Color',cmap(mod(trackToPlotIndex,255)+1,:));
    end
    case 2
% Individual displacements are colored based on length.
% I’m using the pmkmp() colormap here because it shows transitions
% smoother. Change this to hsv() or whatever if you don’t like/have
% pmkmp().
% (http://www.mathworks.com/matlabcentral/fileexchange/28982-perceptually-improved-colormaps)
    cmap = pmkmp(256);
    cmap = flip(jet(256),1);
    cmap = parula(256);
    % Here we’ll set the displacement limits and map the colors to
% these limits. This should be passed as an argument, eventually.
% [For testing purposes, they are mapped to cover the range of
% available displacements.]
    % Get our list of displacements (this should be trimmed to
% longEnoughTracks) and find the 99th percentile in length
% (Rayleigh distributions have a long tail.)
dxl = getdx5(traj8Col,1,2);
maxDxLength = prctile(dxl(:,3),99);
minDxLength = min(dxl(:,3));

% Make a fourth dx1 column that contains an integer representing
% the appropriate color (1-256) for the dx length.
%dx1(:,4) = ceil(((dx1(:,3)-minDxLength)+1)/maxDxLength*256);
dx1(:,4) = ceil(((dx1(:,3)-minDxLength)+1)/maxDxLength*256);
% And make sure it doesn't go above 256 (remember the tail).
dx1(find(dx1(:,4)>256),4) = 256;

% Whoa, this is slow and the resulting figures are a pain to work
% with! I think the problem lies with Matlab, though.
for trackToPlotIndex = 1:length(longEnoughTracks)
    pointToPlotIndex = find(eq(traj8Col(:,7),longEnoughTracks(trackToPlotIndex,1)));
    % Here, we walk through the points in the tracks and plot each
    % displacement with its own color
    for subPointToPlotIndex = 2:length(pointToPlotIndex)
        plot(traj8Col(pointToPlotIndex([subPointToPlotIndex subPointToPlotIndex-1],1),1),traj8Col(pointToPlotIndex([subPointToPlotIndex subPointToPlotIndex-1],1),2),
    %colorbar('Ylim',linspace(min(dx1(:,3)),max(dx1(:,3)),6))
end
plot(nps2Col(:,1),nps2Col(:,2),'ko','MarkerSize',6);

%case 3
% The first n steps of each track are a different color
% Let's start the tracks in red and turn blue after firstSteps steps
%firstSteps = 5;
%firstStepsColor = 'r-';
laterStepsColor = 'b-';

%case 4
% 4 = "colocalized" displacements are a different color

%case 5
% 5 = tracks are colored by msd of individual track

% Load the MSD file
load(fullfile(analysisDir,'msd.mat'));
% Are there the same number of msd elements as tracks?
if ne(length(unique(traj8Col(:,7))), length(msdObj.msd))
    % If not, bail.
    error('Nondescript error #145: Tea not warm enough.');
end
% Set your favorite colormap
cmap = parula(256);
% Make a list of MSDs that correspond to individual tracks to be
% plotted. Looping is stupid, but referring to arrays in cells
% doesn't work??
msdIndex = zeros(length(longEnoughTracks),1);%1:length(length(unique(traj8Col(:,7))), length(msdObj.msd))
for msdIndexIndex = 1:length(longEnoughTracks)
    % This finds the msd (at 0.928s) at track numbers that are longEnough
    msdIndex(msdIndexIndex,1) = msdObj.msd{eq(longEnoughTracks(msdIndexIndex),trackIndex(:,3)),1}(3,2);
end
% Make a colormap of MSDs
% Define these manually. Or add a dynamic scaling function as seen
% above. The units here are µm^2
lowestMSD = min(msdIndex(:,1));
highestMSD = max(msdIndex(:,1));
msdIndex(:,2) = ceil(((msdIndex(:,1)-lowestMSD)/(highestMSD-lowestMSD))+1);
% And make sure it doesn't go above 256 (remember the tail).
msdIndex(msdIndex(:,2)>256,2) = 256;
% Do the plotting
for trackToPlotIndex = 1:length(longEnoughTracks)
    pointToPlotIndex = find(eq(traj8Col(:,7),longEnoughTracks(trackToPlotIndex,1)));
    % Here, we cycle through the colors track-by-track
    plot(traj8Col(pointToPlotIndex,1),traj8Col(pointToPlotIndex,2), 'Color', cmap(msdIndex(trackToPlotIndex,2),:));
end
%colorbar('YTickLabel',linspace(min(msdIndex(:,1)),max(msdIndex(:,1)),6))
end

%% Invert all of the colors
whitespace(gcf);

end
function [ histogramFig ] = plotTurnAngles( turnangles, display, outputDir )
%plotTurnAngles Takes a bunch of turning angles and makes a histogram
% plotTurnAngles makes histograms and shows them or saves them or simply
% eats them.
% INPUT:
%   turnangles is a list of angles in the form of [theta], where theta
%   is the turning angle in radians.
%   display determines whether the histograms are shown on the screen
%   (display=1) or hidden (display=0).
%   outputDir determines where the histogram images are saved. If this
%   is not set, the histograms are not saved. This argument should
%   be a struct containing the directory name (outputDir.dirName)
%   and any prefix (string) to be appended to the filename
%   (outputDir.prefix).
% OUTPUT: This function displays or saves images of histograms.
%   The generated fig will be returned (will it?)

% Is an outputDir argument set? If not, skip the save step
if eq(nargin,2)
    saveFiles = 0;
else
    saveFiles = 1;
end

% Make figure and plot it
if display
    histogramFig = figure('visible','on');
else
    histogramFig = figure('visible','off');
end

% Flip the data around so that "turning angle" is more meaningful
turnangles = pi - turnangles;

hist(turnangles(:,1),180);

% Title the figure with the provided prefix
if exist('outputDir.prefix','var')
    title(outputDir.prefix);
end

% Show bins from zero to pi with ticks at 0, pi/2, and pi
set(gca, 'XLin', [0 pi], 'XTick', [0 0.78538 1.57075 2.3561 3.1415], 'XTickLabel', {'0' 'pi/4' 'pi/2' '3 pi/4' 'pi'});
xlabel('turning angle (rad)');

% If an outputDir argument was passed, save the generated file in that place
if saveFiles
    % save file
    hgexport(gcf, fullfile(outputDir.dirName,[outputDir.prefix '-turn.png']), hgexport('factorystyle'), 'Format', 'png');
end

% We've done with the current figure
if eq(display,1)
close;
end
end

Published with MATLAB® R2015a
function [ output ] = seeingSpots( movieOrImage, frameNum )

%SEEINGSPOTS -- Find the properties of spots in an image or movie
% this program is used to determine the properties of a movie or image
% containing spots.
% INPUT:
%   movieOrImage is a movie or image containing spots
%   frame is the frame of a movie to process (only one) (optional)
%   defaults to all frames of movie, only frame of image (!)
% OUTPUT:
%   output is a struct containing lots of stuff. See below for more details
%   (Note: the struct is dynamic -- see function help for details)
% PC 10 Mar 2014

% Is a frame argument set? If not, set it to zero (all frames)
if eq(nargin,1)
    frameNum = 0;
end

% Is the input a movie or a single image?
if gt(size(movieOrImage,3),1)
    isMovie = 1;
else
    isMovie = 0;
end

% What are the dimensions of the movie or image?
dimensions = size(movieOrImage);

% What are the min/mean/max values of the movie or image?
minValue = min(movieOrImage(:));
meanValue = mean(movieOrImage(:));
maxValue = max(movieOrImage(:));

% How many particles are detected?
switch isMovie
    case 0
        imageBP = bpass(movieOrImage,1,7);
        spotPosRough = pkfnd(imageBP,100,5);
        spotPosFine = cntrd(imageBP,spotPosRough,9);
        numberFrames = 1;
    case 1
        % If a single frame is requested, only process it, otherwise,
        % process the whole movie.
        if ne(frameNum,0)
            frame = movieOrImage(:,:,frameNum);
            frameBP = bpass(frame,1,4);
            framePosRough = pkfnd(frameBP,20,5);
            framePosFine = cntrd(frameBP,framePosRough,9);
            numberFrames = size(movieOrImage,3);
        else
            % movie loop here... (instead of huge sequential array, maybe build
            % a struct with numberFrames fields or a big 3D array packed with
            % NaNs ... done)
            numberFrames = size(movieOrImage,3);
            spotPosFine = nan(500,4,numberFrames);
            for i = 1:numberFrames
                frame = movieOrImage(:,:,i);
                frameBP = bpass(frame,1,4);
                framePosRough = pkfnd(frameBP,20,5);
                framePosFine = cntrd(frameBP,framePosRough,9);
                numSpotsFound = length(framePosFine);
                % Skip this frame if no spots are found. This could be a bad sign and should
                % be dealt with better.
                if numSpotsFound == 0
                    warning('Spotless frame found: %d',i);
                    continue
                end
                % Copy frame to huge array
                spotPosFine(:,i) = framePosFine;
            end
            % Trim out those pesky extra NaN rows
            spotPosFine(all(isnan(spotPosFine(:, i)), 2), i) = [];
        end
end
end
% How many fixed spots (e.g., NPs) are there?
nspts = length(spotPosFine);
output = struct('dimensions',dimensions,'minValue',minValue,'meanValue',meanValue,'maxValue',maxValue,'numberSpots',nspts);
end
function [ traj8ColStuck, traj8ColMobile, report ] = separateStuck( traj8Col, maxDisp )

% separateStuck -- find and separate out stuck and mobile particles
%
% INPUT VARIABLES:
% traj8Col is a track file [x,y,blah,blah,blah,frame,track,coloc]
% stuck is the maximum displacement of the particles allowed to be
% considered stuck
%
% OUTPUT:
% traj8ColStuck is only stuck particles, with everything else replaced by
% NaN
% traj8ColMobile is only mobile particles, with everything else replaced
% by NaN
%
% PC 2014-05-07
%
% Maybe I should make this an argument, but for now, let's just set it
% here.
% Suppress informative outputs? (but not errors)
quiet = 0;
% Find stuck particles by finding particles that begin and end in the same
% place (within 3 pixels), then average all displacements for the particle
% to make sure it never moved much more than 3 pixels.
% Should I instead look for its max displacement? What if that's a fluke??
% Make index of where particles start
sShiftTraj8Col = circshift(traj8Col,-1);
sTrackIndex = find(ne(sShiftTraj8Col(:,7),traj8Col(:,7)));
% Be informative
if eq(quiet,0)
    fprintf('Finding and removing stuck particles.\n');
end
% Make stuck and mobile arrays
traj8ColStuck = traj8Col;
traj8ColMobile = traj8Col;
% Step through particles
stuckParticleCounter = 0;
firstParticlePos = 1;
for count = 1:length(sTrackIndex(:,1))
    particleIndex = sTrackIndex(count,1);
    totalDx = traj8Col(particleIndex,1:2) - traj8Col(firstParticlePos,1:2);
    totalDx(:,3) = sqrt(totalDx(:,1).^2 + totalDx(:,2).^2);
    if lt(totalDx(1,3),maxDisp)
        meanDx = mean(dx(firstParticlePos:(particleIndex - 1),3));
        if lt(meanDx,maxDisp)
            stuckIndex = find(eq(traj8Col(particleIndex,7),traj8Col(:,7)));
            % Turn traj of stuck particles to NaN
            traj8ColMobile(stuckIndex,:) = nan(length(stuckIndex),8);
            stuckParticleCounter = stuckParticleCounter + 1;
            fprintf('.');
        end
    end
    firstParticlePos = particleIndex + 1;
end
% Make the traj8ColStuck array as the inverse of the traj8ColMobile array
mobileIndex = find(isfinite(traj8ColMobile(:,7)));
%sstuckIndex = find(isnan(traj8ColMobile(:,7)));
traj8ColStuck(mobileIndex,:) = nan(length(mobileIndex),8);
% Report here
report.stuck = length(find(isfinite(unique(trajColStuck(:,7)))));
report.mobile = length(find(isfinite(unique(trajColMobile(:,7)))));

% Be informative
if eq(quiet,0)
    fprintf('%d stuck particles found of %d total particles.
',stuckParticleCounter,length(sTrackIndex(:,1)));
end
end
function [ traj8Col, report ] = separatetracks4( traj7Col, fixedSpots, rad )

% SEPARATETRACKS5 -- find parts of trajectories colocalized within radius
% this program is used to determine which trajectories are colocalized
% within radius rad portions of trajectories are separated according to 4
% categories: both (beginning and end of step) colocalized, only beginning,
% only end, neither.
% INPUT VARIABLES:
% trajCol is a track file [x,y,blah,blah,frame,track]
% fixedSpots is the center location particles in the other color channel-
% in [x,y] format.
% rad is the radius to look in pixels
% PC ported from IDL file from Michelle Knowles 2013-10-22
% PC added report variable to be returned

% How many fixed spots (eg, NPs) are there? (not used)
% nspts = length(fixedSpots);
% How many total trajectories are there?
% ntrajsTotal = length(trajCol);
% How many tracked particles are there? (not used)
% ntracks = max(trajCol(:,7));

% Add an extra column of zeros to make
% [x,y,blah,blah,frame,track,coloc]
traj8Col = cat(2,traj7Col,zeros(ntrajsTotal,1)); % traj8Col
colocCount = 0;
noncolocCount = 0;

% This can (and should) be vectorized, but it's fast enough like this to
% make it not worth fixing (yet). If you're looking for speedups, this may
% be relevant.
for i = 1:ntrajsTotal
    % Calculate the distance between the start of a trajectory and any spot
    distanceFromSpots = min(sqrt((traj7Col(i,1) - fixedSpots(:,1)).^2 + (traj7Col(i,2) - fixedSpots(:,2)).^2));
    % If any trajectory is closer than rad, add a '1' in the last column
    if le(distanceFromSpots, rad)
        traj8Col(i,8) = 1;
        colocCount = colocCount + 1;
    else
        noncolocCount = noncolocCount + 1;
    end
end

% Should return these as variables instead of printing to stdout
% [Done. Remove the fprintf lines below to make this function quieter.]
report.coloc = colocCount;
report.noncoloc = noncolocCount;

fprintf('%d colocalization events found\n', colocCount);
fprintf('%d non-colocalization events found\n', noncolocCount);
end

Published with MATLAB® R2015a
function [ sawSpots ] = spottest( movieOrImage )

% spottest -- Load, look for spots, show a picture
%
% this program is used to determine the properties of a movie or image
% containing spots.
%
% INPUT:
% movieOrImage is a movie or image containing spots
%
% OUTPUT:
% sawSpots is a struct containing lots of stuff. See seeingSpots.m for more details
%
% PC 11 Mar 2014

figure;
movieOrImageLoaded = loadTIFFs(movieOrImage);
sawSpots = seeingSpots(movieOrImageLoaded,1);

imshow(movieOrImageLoaded(:,:,1),[min(movieOrImageLoaded(:)) max(movieOrImageLoaded(:))*0.25]);
hold on;
plot(sawSpots.spotsDetected(:,1,1),sawSpots.spotsDetected(:,2,1), 'ro','MarkerSize',10);
title('Frame 1');
if eq(sawSpots.isMovie,1)
    figure;
sawSpots = seeingSpots(movieOrImageLoaded,sawSpots.numberFrames);
    imshow(movieOrImageLoaded(:,:,sawSpots.numberFrames),[min(movieOrImageLoaded(:)) max(movieOrImageLoaded(:))*0.25]);
    hold on;
    plot(sawSpots.spotsDetected(:,1,1),sawSpots.spotsDetected(:,2,1), 'ro','MarkerSize',10);
    title([ 'Frame ',num2str(sawSpots.numberFrames)]);
end
function output = stripNaN( input, column )
%STRIPNAN removes a row if the cell in its first column is a NaN

% If no column is specified, look in the first column
if eq(nargin, 1)
    column = 1;
end

% Strip NaNs if they are present.
output = input(~isnan(input(:,column)),:);

end
function [sub,fls] = subdir(CurrPath)
% SUBDIR lists (recursive) all subfolders and files under given folder
% SUBDIR
%   returns all subfolder under current path.
% P = SUBDIR('directory_name')
%   stores all subfolders under given directory into a variable 'P'
% [P F] = SUBDIR('directory_name')
%   stores all subfolders under given directory into a
%   variable 'P' and all filenames into a variable 'F'.
%   use sort([F{:}]) to get sorted list of all filenames.
% See also DIR, CD

% author:  Elmar Tarajan [Elmar.Tarajan@Mathworks.de]
% version: 2.0
% date:    07-Dec-2004

if nargin == 0
    CurrPath = cd;
end % if
if nargout == 1
    sub = subfolder(CurrPath,'');
else
    [sub fls] = subfolder(CurrPath,'','');
end % if

function [sub,fls] = subfolder(CurrPath,sub,fls)
%------------------------------------------------
tmp = dir(CurrPath);
tmp = tmp(~ismember({tmp.name},{'.' '..'}));
for i = {tmp(~[tmp.isdir]).name}
    sub{end+1} = fullfile(CurrPath,i{:});
    if nargin==2
        sub = subfolder(sub(end),sub);
    else
        tmp = dir(sub{end});
        fls{end+1} = {tmp(~[tmp.isdir]).name};
        [sub fls] = subfolder(sub{end},sub,fls);
    end % if
end % if

Published with MATLAB® R2015a
function [ trajsCell, cellIndex ] = trajsToTrajsCell( traj8Col, options )
%TRAJSTOTRAJSCELL Takes a list of trajectories and returns a cell array of
%trajectories in Tivanex format.
%
% INPUT =
% OUTPUT = trajsCell where each cell contains [time, xpos, ypos]
% - if options.dTime and options.dSpace are supplied, then the
% output will be in the units supplied
% cellIndex whilch contains an index of which track is in which
% cell
if eq(nargin,2)
    % Convert frame and pixel values to the specified units
    traj8Col(:,1:2) = traj8Col(:,1:2) .* options.dSpace;
    traj8Col(:,6) = traj8Col(:,6) .* options.dTime;
end

% Strip NaNs if they are present.
traj8Col = traj8Col(~isnan(traj8Col(:,1)),:);

% Generate a list of the tracks that are left
cellIndex = unique(traj8Col(:,7));

% Make an empty cell array to fill with our tracks
trajsCell = cell(length(cellIndex),1);

% Step through the tracks and make the cell array from them
for i = 1:length(cellIndex)
    % transfer columns 6,1,2 to the cell array
    trajsCell{i,1} = cat(2,traj8Col(eq(cellIndex(i),traj8Col(:,7)),6),traj8Col(eq(cellIndex(i),traj8Col(:,7)),1:2));
end
end

Published with MATLAB® R2015a
function turnangle = turnangle( traj8Col, coloc, dt )

%TURNANGLE -- find the turning angles of trajectories (all, colocalized, or 
%noncolocalized)
% This program finds the turning angles in a track array. The 
% output of the program is an array of mean turning angles for 
% each successive dt up to 1/4 the longest track length 
% INPUT VARIABLES:
% traj8Col is a track file [x,y,blah,blah,blah,frame,track,coloc] 
% coloc determines whether to look at only colocalized (coloc=1) or 
% non colocalized (coloc=0) or all (coloc=2, default) 
% OUTPUT:
% turnangle is an array containing [angle], where angle is in radians 

% Maybe I should make this an argument, but for now, let's just set it
% here.
% Suppress informative outputs? (but not errors)
quiet = 1;
% Allowable values for coloc are 0,1,2. Anything else is treated as 2.
if and((ne(coloc,0)),(ne(coloc,1)))
  coloc = 2;
end
% Over how many frames should we look? If not specified, this defaults to 1
if lt(nargin,3)
  dt = 1;
end
% Make index of coloc or non and NaN out the non-relevant tracks
if eq(coloc,0)
  noncolocIndex = find(eq(traj8Col(:,8),1));
  traj8Col(noncolocIndex,:) = nan(length(noncolocIndex),8);
elseif eq(coloc,1)
  colocIndex = find(eq(traj8Col(:,8),0));
  traj8Col(colocIndex,:) = nan(length(colocIndex),8);
end
% Cleanup time is over! Let's find some turning angles!

Input sanity check

% Can we work with the tracks given?
if isnan(longestTrack)
  if eq(quiet,0)
    fprintf('No tracks present. Cannot determine turning angles for this dt.
    turnangle = nan(1);
  return
elseif lt(longestTrack,4)
% The 4 above should be adjusted to a more reasonable value.
if eq(quiet,0)
    fprintf('Too few tracks present. Cannot determine turning angles for this dt.\n');
end
turnangle = nan(1);
return
else
end

Calculate the turning angles

% Shift the array of trajectories up one and two steps
% traj8Col is point 1, shift1Traj8Col is point 2, shift2Traj8Col is point 3
shift1Traj8Col = circshift(traj8Col,-dt);
shift2Traj8Col = circshift(traj8Col,-(2*dt));

% Find the sides of the triangle formed by each three points
% The first displacement vector (point 1 to point 2)
side12 = shift1Traj8Col(:,1:2) - traj8Col(:,1:2);
side12(:,3) = sqrt(side12(:,1).^2 + side12(:,2).^2);

% The second displacement vector (point 2 to point 3)
side23 = shift2Traj8Col(:,1:2) - shift1Traj8Col(:,1:2);
side23(:,3) = sqrt(side23(:,1).^2 + side23(:,2).^2);

% The total displacement vector (point 1 to point 3)
side13 = shift2Traj8Col(:,1:2) - traj8Col(:,1:2);
side13(:,3) = sqrt(side13(:,1).^2 + side13(:,2).^2);

% Find the angle between the two actual displacement vectors, law of cosines style.
% turnangle = acos((side12(:,3).^ 2 + side23(:,3).^ 2 - side13(:,3).^ 2) ./ (2 .* side12(:,3) .* side23(:,3)));

Cleanup time

% Find where in the array new tracks start
trackIndex = find(ne(shift1Traj8Col(:,7),traj8Col(:,7)));
trackIndex2 = find(ne(shift2Traj8Col(:,7),traj8Col(:,7)));

% Remove the angles that correspond to new tracks starting
turnangle(trackIndex,:) = nan(length(trackIndex),1);
turnangle(trackIndex2,:) = nan(length(trackIndex2),1);

end

Published with MATLAB® R2015a
function res = bpass(image_array,lnoise,lobject,threshold)

% NAME:
% bpass

% PURPOSE:
% Implements a real-space bandpass filter that suppresses pixel noise and long-wavelength image variations while retaining information of a characteristic size.

% CATEGORY:
% Image Processing

% CALLING SEQUENCE:
% res = bpass( image_array, lnoise, lobject )

% INPUTS:
% image: The two-dimensional array to be filtered.
% lnoise: Characteristic lengthscale of noise in pixels. Additive noise averaged over this length should vanish. May assume any positive floating value. May be set to 0 or false, in which case only the highpass "background subtraction" operation is performed.
% lobject: (optional) Integer length in pixels somewhat larger than a typical object. Can also be set to 0 or false, in which case only the lowpass "blurring" operation defined by lnoise is done, without the background subtraction defined by lobject. Defaults to false.
% threshold: (optional) By default, after the convolution, any negative pixels are reset to 0. Threshold changes the threshold for setting pixels to 0. Positive values may be useful for removing stray noise or small particles. Alternatively, can be set to -Inf so that no threshholding is performed at all.

% OUTPUTS:
% res: filtered image.

% PROCEDURE:
% simple convolution yields spatial bandpass filtering.

% NOTES:
% Performs a bandpass by convolving with an appropriate kernel. You can think of this as a two part process. First, a lowpassed image is produced by convolving the original with a gaussian. Next, a second lowpassed image is produced by convolving the original with a boxcar function. By subtracting the boxcar version from the gaussian version, we are using the boxcar version to perform a highpass.
% original - lowpassed version of original = highpassed version of the original
% Performing a lowpass and a highpass results in a bandpassed image.
% Converts input to double. Be advised that commands like 'image' display % double precision arrays differently from UINT8 arrays.

% MODIFICATION HISTORY:
% Written by David G. Grier, The University of Chicago, 2/93.
% Greatly revised version DGG 5/95.
% Added /field keyword JCC 12/95.
% Memory optimizations and fixed normalization, DGG 8/99.
% Converted to Matlab by D.Blair 4/2004-ish
% Fixed some bugs with conv2 to make sure the edges are removed D.B. 6/05
% Removed inadvertent image shift ERD 6/05
% Added threshold to output. Now sets all pixels with negative values equal to zero. Gets rid of ringing which was destroying sub-pixel accuracy, unless window size in cntrd was picked perfectly. Now centrd gets sub-pixel accuracy much more robustly ERD 8/24/05
% Refactored for clarity and converted all convolutions to use column vector kernels for speed. Running on my macbook, the old version took ~1.3 seconds to do bpass(image_array,1,19) on a 1024 x 1024 image; this version takes roughly half that. JWM 6/07
% This code 'bpass.pro' is copyright 1997, John C. Crocker and David G. Grier. It should be considered 'freeware'- and may be distributed freely in its original form when properly attributed.

if nargin < 3, lobject = false; end
if nargin < 4, threshold = 0; end

normalize = @(x) x/sum(x);

image_array = double(image_array);

if lnoise == 0
    gaussian_kernel = 1;
else
    gaussian_kernel = normalize(...
        exp(-((-ceil(5*lnoise):ceil(5*lnoise))/(2*lnoise)).^2));
end

if lobject
    boxcar_kernel = normalize(...
        ones(1,length(-round(lobject):round(lobject))));
end
% JWM: Do a 2D convolution with the kernels in two steps each. It is
% possible to do the convolution in only one step per kernel with
% gconv = conv2(gaussian_kernel',gaussian_kernel,image_array,'same');
% bconv = conv2(boxcar_kernel', boxcar_kernel,image_array,'same');
% but for some reason, this is slow. The whole operation could be reduced
% to a single step using the associative and distributive properties of
% convolution:
% filtered = conv2(image_array,...
%   gaussian_kernel'*gaussian_kernel - boxcar_kernel'*boxcar_kernel,...
%   'same');
% But this is also comparatively slow (though inexplicably faster than the
% above). It turns out that convolving with a column vector is faster than
% convolving with a row vector, so instead of transposing the kernel, the
% image is transposed twice.

gconv = conv2(image_array',gaussian_kernel','same');
gconv = conv2(gconv',gaussian_kernel','same');

if lobject
    bconv = conv2(image_array',boxcar_kernel','same');
bconv = conv2(bconv',boxcar_kernel','same');
    filtered = gconv - bconv;
else
    filtered = gconv;
end

% Zero out the values on the edges to signal that they're not useful.
lzero = max(lobject,ceil(5*lnoise));
filtered(1:(round(lzero)),:) = 0;
filtered((end - lzero + 1):end,:) = 0;
filtered(:,1:(round(lzero))) = 0;
filtered(:,(end - lzero + 1):end) = 0;

% JWM: I question the value of zeroing out negative pixels. It’s a
% nonlinear operation which could potentially mess up our expectations
% about statistics. Is there data on ‘Now centroid gets subpixel accuracy
% much more robustly’? To choose which approach to take, uncomment one of
% the following two lines.
% ERD: The negative values shift the peak if the center of the cntrd mask
% is not centered on the particle.

% res = filtered;
filtered(filtered < threshold) = 0;
res = filtered;
function out=cntrd(im,mx,sz,interactive)
% PURPOSE: calculates the centroid of bright spots to sub-pixel accuracy.
% Inspired by Grier & Crocker's feature for IDL, but greatly simplified and optimized
% for Matlab

% INPUT:
% im: image to process, particle should be bright spots on dark background with little noise
% often an bandpass filtered brightfield image or a nice fluorescent image
% mx: locations of local maxima to pixel-level accuracy from pkfnd.m
% sz: diameter of the window over which to average to calculate the centroid.
%     should be big enough to capture the whole particle but not so big that it captures others.
%     if initial guess of center (from pkfnd) is far from the centroid, the
%     window will need to be larger than the particle size. RECOMMENDED
%     size is the long lengthscale used in bpass plus 2.
%     interactive: OPTIONAL INPUT set this variable to one and it will show you the image used to calculate
%     each centroid, the pixel-level peak and the centroid

% NOTE:
% - if pkfnd and cntrd return more than one location per particle then
% you should try to filter your input more carefully. If you still get
% more than one peak for particle, use the optional sz parameter in pkfnd
% - If you want sub-pixel accuracy, you need to have a lot of pixels in your window (sz>>1).
% To check for pixel bias, plot a histogram of the fractional parts of the resulting locations
% - It is HIGHLY recommended to run in interactive mode to adjust the parameters before you
% analyze a bunch of images.

% OUTPUT: a N x 4 array containing, x, y and brightness for each feature
% out(:,1) is the x-coordinates
% out(:,2) is the y-coordinates
% out(:,3) is the brightnesses
% out(:,4) is the square of the radius of gyration

% CREATED: Eric R. Dufresne, Yale University, Feb 4 2005
% 5/2005 inputs diameter instead of radius
% Modifications:
% D.B. (6/05) Added code from imdist/dist to make this stand alone.
% ERD (6/05) Increased frame of reject locations around edge to 1.5*sz
% ERD 6/2005 By popular demand, 1. altered input to be formatted in x,y
% space instead of row, column space 2. added forth column of output,
% rg^2
% ERD 8/05 Outputs had been shifted by [0.5,0.5] pixels. No more!
% ERD 8/24/05 Woops! That last one was a red herring. The real problem
% is the 'ringing' from the output of bpass. I fixed bpass (see note),
% and no longer need this kludge. Also, made it quite nice if mx=[];
% ERD 6/06 Added size and brightness output ot interactive mode. Also
% fixed bug in calculation of rg^2
% JWM 6/07 Small corrections to documentation

if nargin==3
    interactive=0;
end

if sz/2 == floor(sz/2)
    warning('sz must be odd, like bpass');
end

if isempty(mx)
    warning('there were no positions inputted into cntrd. check your pkfnd threshold');
    out=[];
    return;
end

if nargin==3
    interactive=0;
end

if sz/2 == floor(sz/2)
    warning('sz must be odd, like bpass');
end

if isempty(mx)
    warning('there were no positions inputted into cntrd. check your pkfnd threshold');
    out=[];
    return;
end
\[ r = \frac{sz+1}{2}; \]
% create mask - window around trial location over which to calculate the centroid
\[
m = 2r; \]
\[
x = 0:(m-1) ;\]
\[
cent = (m-1)/2; \]
\[
x2 = (x-cent)^2; \]
\[
dst=zeros(m,m); \]
\[
for i=1:m
  dst(i,:) = sqrt((i-1-cent)^2 + x2); \]
end

ind = find(dst < r);
msk = zeros([2*r,2*r]);
msk(ind) = 1.0;
msk = circshift(msk,[-r,-r]);
dst2 = msk.*(dst.^2);
ndst2 = sum(sum(dst2));

[nr,nc] = size(im);
remove all potential locations within distance sz from edges of image
ind = find(mx(:,2) > 1.5*sz & mx(:,2) < nr-1.5*sz);
mx = mx(ind,:);
ind = find(mx(:,1) > 1.5*sz & mx(:,1) < nc-1.5*sz);
mx = mx(ind,:);

[nmx, crap] = size(mx);
% inside of the window, assign an x and y coordinate for each pixel
xl = zeros(2*r,2*r);
for i = 1:2*r
  xl(i,:) = (1:2*r);
end
yl = xl';
pts = [];
% loop through all of the candidate positions
for i = 1:nmx
  % create a small working array around each candidate location, and apply the window function
tmp = msk.*im(mx(i,2)-r+1:mx(i,2)+r,mx(i,1)-r+1:mx(i,1)+r);
  % calculate the total brightness
  norm = sum(sum(tmp));
  % calculate the weighted average x location
  xavg = sum(sum(tmp.*xl))/norm;
  % calculate the weighted average y location
  yavg = sum(sum(tmp.*yl))/norm;
  % calculate the radius of gyration^2
  rg = (sum(sum(tmp.*dst2))/ndst2);
  % concatenate it up
  pts = [pts, [mx(i,1)+xavg-r, mx(i,2)+yavg-r, norm, rg]'];
end

% optional plot things up if you're in interactive mode
if interactive == 1
  imagesc(tmp)
  axis image
  hold on;
  plot(xavg, yavg, 'x')
  plot(xavg, yavg, 'o')
  plot(r, r, '.')
  hold off
  title(['brightness ', num2str(norm), ' size ', num2str(sqrt(rg))])
  pause
end
out = pts';
function trlen = lentrk(tr);

s = size(tr);

ndat = s(2)

[b,u] = unique(tr(:,ndat));
ntracks = length(u)
u = [0;u];

for i=2:ntracks
    res(i-1,1) = tr(u(i),ndat-1) - tr(u(i-1)+1, ndat-1);
    res(i-1,2) = tr(u(i),ndat);
end

trlen=res;
function out=pkfnd(im,th,sz)
% finds local maxima in an image to pixel level accuracy.
% this provides a rough guess of particle centers to be used by cntrd.m. Inspired by the lmx subroutine of Grier
% and Crocker's feature.pro
% INPUTS:
% im: image to process, particle should be bright spots on dark background with little noise
% often an bandpass filtered brightfield image (fbps.m, fftl.m or bpasa.m) or a nice
% fluorescent image
% th: the minimum brightness of a pixel that might be local maxima.
% (NOTE: Make it big and the code runs faster
% but you might miss some particles. Make it small and you'll get
% everything and it'll be slow.)
% sz: if your data's noisy, (e.g. a single particle has multiple local
% maxima), then set this optional keyword to a value slightly larger than the diameter of your blob. if
% multiple peaks are found withing a radius of sz/2 then the code will keep
% only the brightest. Also gets rid of all peaks within sz of boundary
% OUTPUT: a N x 2 array containing, [row,column] coordinates of local maxima
% out(:,1) are the x-coordinates of the maxima
% out(:,2) are the y-coordinates of the maxima
%CRED: Eric R. Dufresne, Yale University, Feb 4 2005
%MODIFIED: ERD, 5/2005, got rid of ind2rc.m to reduce overhead on tip by
% Dan Blair; added sz keyword
% ERD, 6/2005: modified to work with one and zero peaks, removed automatic
% normalization of image
% ERD, 6/2005: due to popular demand, altered output to give x and y
% instead of row and column
% ERD, 8/24/2005: pkfnd now exits politely if there's nothing above
% threshold instead of crashing rudely
% ERD, 6/14/2006: now exits politely if no maxima found
% ERD, 10/5/2006: fixed bug that threw away particles with maxima
% consisting of more than two adjacent points

find all the pixels above threshold
im=im./max(max(im));
ind=find(im > th);
[nr,nc]=size(im);
tst=zeros(nr,nc);
end
mx=[];
%convert index from find to row and column
rc=[mod(ind,nr),floor(ind/nr)+1];
for i=1:n
r=rc(i,1);c=rc(i,2);
%check each pixel above threshold to see if it's brighter than it's neighbors
% THERE'S GON'T TO BE A FASTER WAY OF DOING THIS. I'M CHECKING SOME MULTIPLE TIMES,
% BUT THIS DONGN'T SEEM THAT SLOW COMPARED TO THE OTHER ROUTINES, ANYWAY.
if r>1 & r<nr & c>1 & c<nc
if im(r,c) >=im(r-1,c-1) & im(r,c) >=im(r-1,c) & im(r,c) >=im(r-1,c+1) & ... & im(r,c) >=im(r+1,c-1) & im(r,c) >=im(r+1,c) & im(r,c) >=im(r+1,c+1)
mx=[mx(r,c)];
tst(ind(i))=im(ind(i));
end
end
end
out=tst;
mx=mx';
[npkks,crap]=size(mx);

if size is specified, then get ride of pts within size of boundary
if nargin==3 & npkks>0
throw out all pts within sz of boundary:
ind=find(mx(:,1)>sz & mx(:,1)<(nr-sz) & mx(:,2)>sz & mx(:,2)<(nc-sz));
end
%prevent from finding peaks within size of each other
[npkks,crap]=size(mx);
if npks > 1

% CREATE AN IMAGE WITH ONLY PEAKS
nmx=npks;
tmp=0.*im;
for i=1:nmx
    tmp(mx(i,1),mx(i,2))=im(mx(i,1),mx(i,2));
end

% LOOK IN NEIGHBORHOOD AROUND EACH PEAK, PICK THE BRIGHTEST
for i=1:nmx
    roi=tmp( (mx(i,1)-floor(sz/2)):(mx(i,1)+(floor(sz/2)+1)),(mx(i,2)-floor(sz/2)):(mx(i,2)+(floor(sz/2)+1))) ;
    [mv,indi]=max(roi);
    [mv,indj]=max(mv);
    tmp( (mx(i,1)-floor(sz/2)):(mx(i,1)+(floor(sz/2)+1)),(mx(i,2)-floor(sz/2)):(mx(i,2)+(floor(sz/2)+1)))=0;
    tmp(mx(i,1)-floor(sz/2)+indi(indj)-1,mx(i,2)-floor(sz/2)+indj-1)=mv;
end

ind=find(tmp>0);
mx=[mod(ind,nr),floor(ind/nr)+1];
end

if size(mx)==[0,0]
    out=[];
else
    out(1,2)=mx(:,1);
    out(1,1)=mx(:,2);
end

Published with MATLAB® R2015a
function tracks = track(.xyzs, maxdisp, param)

%;
%; see http://glinda.lrms.upenn.edu/~weeks/idl
%; for more information
%;
%; NAME:
%; track
%; PURPOSE:
%; Constructs n-dimensional trajectories from a scrambled list of
%; particle coordinates determined at discrete times (e.g. in
%; consecutive video frames).
%; CATEGORY:
%; Image Processing
%; CALLING SEQUENCE:
%; result = track(poslist, maxdisp, param )
%; set all keywords in the space below
%; INPUTS:
%; poslist: an array listing the scrambled coordinates and data
%; of the different particles at different times, such that:
%; poslist(0:d-1,*) : contains the d coordinates and
%; data for all the particles, at the different times. must be positive
%; poslist(d,*) : contains the time t that the position
%; was determined, must be integers (e.g. frame number. These values must
%; be monotonically increasing and uniformly gridded in time.
%; maxdisp: an estimate of the maximum distance that a particle
%; would move in a single time interval.(see Restrictions)
%; OPTIONAL INPUT:
%; param: a structure containing a few tracking parameters that are
%; needed for many applications. If param is not included in the
%; function call, then default values are used. If you set one value
%; make sure you set them all:
%;     param.mem: this is the number of time steps that a particle can be
%;     'lost' and then recovered again. If the particle reappears
%;     after this number of frames has elapsed, it will be
%;     tracked as a new particle. The default setting is zero.
%;     this is useful if particles occasionally 'drop out' of
%;     the data.
%;     param.dim: if the user would like to unscramble non-coordinate data
%;     for the particles (e.g. apparent radius of gyration for
%;     the particle images), then poslist should
%;     contain the position data in poslist(0:dim-1,*)
%;     and the extra data in poslist(dim:dimd-1,*). It is then
%;     necessary to set dim equal to the dimensionality of the
%;     coordinate data to so that the track knows to ignore the
%;     non-coordinate data in the construction of the
%;     trajectories. The default value is two.
%;     param.good: set this keyword to eliminate all trajectories with
%;     fewer than param.good valid positions. This is useful
%;     for eliminating very short, mostly 'lost' trajectories
%;     due to blinking 'noise' particles in the data stream.
%;     param.quiet: set this keyword to 1 if you don't want any text
%; OUTPUTS:
%; result: a list containing the original data rows sorted
%; into a series of trajectories. To the original input
%; data structure there is appended an additional column
%; containing a unique 'id number' for each identified
%; particle trajectory. The result array is sorted so
%; rows with corresponding id numbers are in contiguous
%; blocks, with the time variable a monotonically
%; increasing function inside each block. For example:
%;
% For the input data structure (positionlist):
% (x)  (y)  (t)
% pos = 3.60000 5.00000 0.00000
% 15.1000 22.6000 0.00000
% 4.10000 5.50000 1.00000
% 15.9000 20.7000 2.00000
% 6.20000 4.30000 2.00000
% IDL> res = track(pos,5,mem=2)
% track will return the result 'res'
% (x)  (y)  (t)  (id)
% res = 3.60000 5.00000 0.00000 0.00000
% 4.10000 5.50000 1.00000 0.00000
% 6.20000 4.30000 2.00000 0.00000
% 15.1000 22.6000 0.00000 1.00000
% 15.9000 20.7000 2.00000 1.00000
% NB: for t=1 in the example above, one particle temporarily
% vanished. As a result, the trajectory id=1 has one time
% missing, i.e. particle loss can cause time gaps to occur
% in the corresponding trajectory list. In contrast:
% IDL> res = track(pos,5)
% track will return the result 'res'
% (x)  (y)  (t)  (id)
% res = 15.1000 22.6000 0.00000 0.00000
% 3.60000 5.00000 0.00000 1.00000
% 4.10000 5.50000 1.00000 1.00000
% 6.20000 4.30000 2.00000 1.00000
% 15.9000 20.7000 2.00000 2.00000
% where the reappeared 'particle' will be labelled as new
% rather than as a continuation of an old particle since
% mem=0. It is up to the user to decide what setting of
% 'mem' will yield the highest fidelity.
% SIDE EFFECTS:
% Produces informational messages. Can be memory intensive for
% extremely large data sets.
% RESTRICTIONS:
% maxdisp should be set to a value somewhat less than the mean
% spacing between the particles. As maxdisp approaches the mean
% spacing the run time will increase significantly. The function
% will produce an error message: "Excessive Combinatorics!" if
% the run time would be too long, and the user should respond
% by re-executing the function with a smaller value of maxdisp.
% Obviously, if the particles being tracked are frequently moving
% as much as their mean separation in a single time step, this
% function will not return acceptable trajectories.
% PROCEDURE:
% Given the positions for n particles at time t(i), and m possible
% new positions at time t(i+1), this function considers all possible
% identifications of the n old positions with the m new positions,
% and chooses that identification which results in the minimal total
% squared displacement. Those identifications which don’t associate
% a new position within maxdisp of an old position ( particle loss )
% penalize the total squared displacement by maxdisp^2. For non-
% interacting Brownian particles with the same diffusivity, this
% algorithm will produce the most probable set of identifications
% ( provided maxdisp >> RMS displacement between frames ).
% In practice it works reasonably well for systems with oscillatory,
% ballistic, correlated and random hopping motion, so long as single
% time step displacements are reasonably small. NB: multidimensional
% functionality is intended to facilitate tracking when additional
% information regarding target identity is available (e.g. size or
% color). At present, this information should be rescaled by the
% user to have a comparable or smaller (measurement) variance than
% the spatial displacements.

% MODIFICATION HISTORY:
% 2/93 Written by John C. Crocker, University of Chicago (JFI).
% 7/93 JCC fixed bug causing particle loss and improved performance
% for large numbers of (>100) particles.
% 11/93 JCC improved speed and memory performance for large
% numbers of (>1000) particles (added subnetwork code).
% 3/94 JCC optimized run time for trivial bonds and d<7. (Added
% d-dimensional raster metric code.)
% 8/94 JCC added functionality to unscramble non-position data
% along with position data.
% 9/94 JCC rewrote subnetwork code and wrote new, more efficient
% permutation code.
% 5/95 JCC debugged subnetwork and excessive combinatorics code.
% 12/95 JCC added memory keyword, and enabled the tracking of
% newly appeared particles.
% 3/96 JCC made inipos a keyword, and disabled the adding of 'new'
% particles when inipos was set.
% 3/97 JCC added 'add' keyword, since Chicago users didn't like
% having particle addition be the default.
% 9/97 JCC added 'goodenough' keyword to improve memory efficiency
% when using the 'add' keyword and to filter out bad tracks.
% 10/97 JCC streamlined data structure to speed runtime for >200
% timesteps. Changed 'quiet' keyword to 'verbose'. Made
% time labelling more flexible (uniform and sorted is ok).
% 9/98 JCC switched trajectory data structure to a 'list' form,
% resolving memory issue for large, noisy datasets.
% 2/99 JCC added Eric Weeks's 'uberize' code to post-facto
% rationalize the particle id numbers, removed 'add' keyword.
% 1/05 Transmuted to MATLAB by D. Blair
% 5/05 ERD Added the param structure to simplify calling.
% 6/05 ERD Added quiet to param structure
% 7/05 DLB Fixed slight bug in trivial bond code
% 3/07 DLB Fixed bug with max disp pointed out by Helene Delanoe-Ayari
% 3/14 PC Started cleaning up Matlab port of this. OMG, what have I gotten myself into?!
% This code 'track.pro' is copyright 1999, by John C. Crocker.
% It should be considered 'freeware'- and may be distributed freely
% (outside of the military-industrial complex) in its original form
% when properly attributed.
% %
% dd = length(xyzs(1,:));

% use default parameters if none given
if nargin==2
    % default values
    memory_b = 0; % if mem is not needed set to zero
    goodenough = 0; % if goodenough is not wanted set to zero
    dim = dd - 1;
    quiet = 0;
else
    memory_b = param.mem;
    goodenough = param.good;
    dim = param.dim;
    quiet = param.quiet;
end
% checking the input time vector
% t = xys(:,:,dd);
st = circshift(t,1);
st = t(2:end) - st(2:end);
if sum(st(st < 0)) ~= 0
    disp('The time vectors are not in order')
    return
end
info = 1;

w = find(st > 0);
z = length(w);
if isempty(w)
    disp('All positions are at the same time... go back!')
    return
end

% partitioning the data with unique times
% res = unq(t);
% implanting unq directly
% indices = find(t == circshift(t,-1));
% count = length(indices);
% if count > 0
%    res = indices;
% else
%    res = length(t) -1;
% end

res = 1:length(t);

% garbage collection
% clear t;

ngood = res(2) - res(1) + 1;
eyes = 1:ngood;
pos = xys(eyes,1:dim);
istart = 2;
n = ngood;

zspan = 50;
if n > 200
    zspan = 20;
end
if n > 500
    zspan = 10;
end
resx = zeros(zspan,n) - 1;

bigresx = zeros(z,n) - 1;
mem = zeros(n,1);
whos resx
% whos bigresx
uniqid = 1:n;
maxid = n;
olist = [0.,0.];

if goodenough > 0
    dumphash = zeros(n,1);
    nvalid = ones(n,1);
end

% whos eyes;
resx(1,:) = eyes;
% setting up constants
maxdisq = maxdisp^2;

% John calls this the setup for "fancy code" ???
notnsqrd = (sqrt(n*ngood) > 200) & (dim < 7);
notnsqrd = notnsqrd(1);
if notnsqrd
  % construct the vertices of a 3x3x3... d-dimensional hypercube
  cube = zeros(3^dim,dim);
  for d=0:dim-1,
    numb = 0;
    for j=0:(3^d):(3^dim)-1,
      cube(j+1:j+(3^(d)),d+1) = numb;
      numb = mod(numb+1,3);
    end
  end

  % calculate a blocksize which may be greater than maxdisp, but which
  % keeps nblocks reasonably small.
  volume = 1;
  for d = 0:dim-1
    minn = min(xyzs(w,d+1));
    maxx = max(xyzs(w,d+1));
    volume = volume * (maxx-minn);
  end
  blocksize = max( [maxdisp,(volume)/(20*ngood))^(1.0/dim)] );
end

% Start the main loop over the frames.
for i=istart:z
  ispan = mod(i-1,zspan)+1;
  %disp(ispan)
  % get new particle positions
  m = res(i+1) - res(i);
  res(i);
  eyes = 1:m;
  eyes = eyes + res(i);
  if m > 0
    xyi = xyzs(eyes,1:dim);
    found = zeros(m,1);
    % THE TRIVIAL BOND CODE BEGINS
    if notnsqrd
      %Use the raster metric code to do trivial bonds
      % construct "s", a one dimensional parameterization of the space
      % which consists of the d-dimensional raster scan of the volume.)
      abi = fix(xyi./blocksize);
      abpos = fix(pos./blocksize);
      si = zeros(m,1);
      spos = zeros(n,1);
      dimm = zeros(dim,1);
      coff = 1.;
      for j=1:dim
minn = min([abi(:,j);abpos(:,j)]);
maxx = max([abi(:,j);abpos(:,j)]);
abi(:,j) = abi(:,j) - minn;
abpos(:,j) = abpos(:,j) - minn;
dimm(j) = maxx-minn + 1;
si = si + abi(:,j).*coff;
spos = spos + abpos(:,j).*coff;
coff = dimm(j).*coff;
end
nblocs = coff;
% x = find( dimm < 3);
cub = cube;
if ~isempty(x)
    for j = 0:length(x)-1
        cub = cub(cub(:,x(j+1)) < dimm(x(j+1)),:);
    end
end
% calculate the "s" coordinates of hypercube (with a corner @ the origin)
scube = zeros(length(cub(:,1)),1);
coff = 1;
for j=1:dim
    scube = scube + cub(:,j).*coff;
    coff = coff*dimm(j);
end
% shift the hypercube "s" coordinates to be centered around the origin
coff = 1;
for j=1:dim
    if dimm(j) > 3
        scube = scube - coff;
    end
    coff = dimm(j).* coff;
end
scube = mod((scube + nbloc),nbloc);
% debug: catch 'out of memory' errors before they happen
% if gt(nbloc,freemem)
  % if eq(exist('blink1','file'),2)
    % blink1('on','red');
  % end
  % fprintf('Failed at iteration %d\n',i);
  % error('Out of memory was going to occur. Make me more resilient to that!');
end
strt = zeros(nbloc,1) -1;
fnsh = zeros(nbloc,1);
h = find(si == 0);
lh = length(h);
if lh > 0
    si(h) = 1;
end
for j=1:m
    if strt(si(isort(j))) == -1
        strt(si(isort(j))) = j;
        fnsh(si(isort(j))) = j;
    else
        fnsh(si(isort(j))) = j;
    end
end
if lh > 0
    si(h) = 0;
end
coltot = zeros(m,1);
rowtot = zeros(n,1);
which1 = zeros(n,1);
for j=1:n
    map = fix(-1);
    scub_spos = scube + spos(j);
    s = mod(scub_spos,nblocks);
    whzero = find(s == 0, 1);
    if ~isempty(whzero)
        nfk = s ~= 0;
        s = s(nfk);
    end
    w = find(strt(s) == -1);
    ngood = length(w);
    %ltmax=0;
    if ngood == 0
        s = s(w);
        for k=1:ngood
            map = [map;isort( strt(s(k)):fnsh(s(k)))];
        end
        map = map(2:end);
        % if length(map) == 2
        %     if (map(1) - map(2)) == 0
        %         map = unique(map);
        %     end
        % end
        % map = map(umap);
    end
    % find those trivial bonds
    distq = zeros(length(map),1);
    for d=1:dim
        distq = distq + (xyi(map,d) - pos(j,d)).^2;
    end
    ltmax = distq < maxdisq;
    rowtot(j) = sum(ltmax);
    if rowtot(j) >= 1
        w = find(ltmax == 1);
        coltot( map(w) ) = coltot( map(w) ) +1;
        which1(j) = map( w(1) );
    end
end
ntrk = fix(n - sum(rowtot == 0));
w = find(rowtot == 1);
ngood = length(w);

if ngood ~= 0
    ww = find(coltot(which1(w)) == 1);
    ngood = length(ww);
    if ngood == 0
        disp(size(w(ww)))
        resx(ispan,w(ww)) = eyes(which1(w(ww)));
        found(which1(w(ww))) = 1;
        rowtot(w(ww)) = 0;
        coltot(which1(w(ww))) = 0;
    end
end

labely = find(rowtot > 0);
ngood = length(labely);
if ngood ~= 0
    labelx = find(coltot > 0);
else
    nontrivial = 1;
else
    nontrivial = 0;
end

else
    % or: Use simple N^2 time routine to calculate trivial bonds
    % let's try a nice, loopless way!
    % don't bother tracking perm. lost guys.
    wh = find(pos(:,1) >= 0);
ntrack = length(wh);
    if ntrack == 0
        fprintf('There are no valid particles to track idiot!');
        break
    end
    xmat = zeros(ntrack,m);
count = 0;
    for kk=1:ntrack
        for ll=1:m
            xmat(kk,ll) = count;
            count = count+1;
        end
    end
    count = 0;
    for kk=1:m
        for ll=1:ntrack
            ymat(kk,ll) = count;
            count = count+1;
        end
    end
    xmat = (mod(xmat,m) + 1);
    ymat = (mod(ymat,ntrack) +1)';
    [lenx,lenxm] = size(xmat);
    whos ymat
    whos xmat
    disp(m)
    for d=1:dim
        x = xyi(:,d);
        y = pos(wh,d);
\[ \text{xm} = x(xmat); \]
\[ \text{ym} = y(ymat(1:lenxn,1:lenxm)); \]
\[ \text{if size(xm) \neq size(ym)} \]
\[ \text{xm} = \text{xm}'; \]
\[ \text{end} \]
\[ \text{if } d == 1 \]
\[ \text{dq} = (\text{xm} - \text{ym})^2; \]
\[ \text{end} \]
\[ \text{else} \]
\[ \text{dq} = \text{dq} + (\text{xm} - \text{ym})^2; \]
\[ \text{end} \]
\[ \text{ltmax} = \text{dq} < \text{maxdisq}; \]
\[ \% \text{figure out which trivial bonds go with which} \]
\[ \text{rowtot} = \text{zeros(n,1)}; \]
\[ \text{rowtot(wh)} = \text{sum(ltmax,2)}; \]
\[ \text{if } \text{ntrack} > 1 \]
\[ \text{coltot} = \text{sum(ltmax,1)}; \]
\[ \text{else} \]
\[ \text{coltot} = \text{ltmax}; \]
\[ \text{end} \]
\[ \text{whichl} = \text{zeros(n,1)}; \]
\[ \text{for } j=1:\text{ntrack} \]
\[ \text{[~, w] = max(ltmax(j,:));} \]
\[ \text{whichl(wh(j)) = w;} \]
\[ \text{end} \]
\[ \text{ntrk} = \text{fix( n - sum(rowtot == 0));} \]
\[ \text{w} = \text{find( rowtot == 1)}; \]
\[ \text{ngood} = \text{length(w)}; \]
\[ \text{if } \text{ngood} == 0 \]
\[ \text{ww} = \text{find(coltot(whichl(w)) == 1)}; \]
\[ \text{ngood} = \text{length(ww)}; \]
\[ \text{if } \text{ngood} == 0 \]
\[ \text{resx( ispan, w(ww) ) = eyes( whichl( w(ww)));} \]
\[ \text{found(whichl( w(ww))) = 1;} \]
\[ \text{rowtot(w(ww)) = 0;} \]
\[ \text{coltot(whichl(w(ww))) = 0;} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{labely} = \text{find( rowtot > 0)}; \]
\[ \text{ngood} = \text{length(labely)}; \]
\[ \text{if } \text{ngood} == 0 \]
\[ \text{labelx} = \text{find( coltot > 0)}; \]
\[ \text{nontrivial} = 1; \]
\[ \text{else} \]
\[ \text{nontrivial} = 0; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \% \text{THE TRIVIAL BOND CODE ENDS} \]
\[ \text{if } \text{nontrivial} \]
\[ \text{xdim} = \text{length(labelx)}; \]
\[ \text{ydim} = \text{length(labely)}; \]
% make a list of the non-trivial bonds

bonds = zeros(1,2);
bondlen = 0;

for j=1:ydim
distq = zeros(xdim,1);

    for d=1:dim
        %distq
        distq = distq + (xyi(labelx,d) - pos(labely(j),d)).^2;
    %distq

    end

w= find(distq < maxdisq)' - 1;
good = length(w);
newb = [w;(zeros(1,good)+j)];

bonds = [bonds;newb'];

bondlen = [bondlen;distq( w + 1)];

end

bonds = bonds(2:end,:);
bondlen = bondlen(2:end);
numbonds = length(bonds(:,1));
mbonds = bonds;
max([xdim,ydim]);

if max([xdim,ydim]) < 4
    nclust = 1;
    maxsz = 0;
    mxsz = xdim;
    mysz = ydim;
    bmap = zeros(length(bonds(:,1)+1),1) - 1;
else

    % THE SUBNETWORK CODE BEGINS
    lista = zeros(numbonds,1);
    listb = zeros(numbonds,1);
    nclust = 0;
    maxsz = 0;
    thru = xdim;

    while thru == 0
        % the following code extracts connected
        % sub-networks of the non-trivial
        % bonds. NB: lista/b can have redundant entries due to
        % multiple-connected subnetworks

        w = find(bonds(:,2) >= 0);
        % size(w)

        lista(1) = bonds(w(1),2);
        listb(1) = bonds(w(1),1);
        bonds(w(1),:) = -(nclust+1);

    end

    adda = 1;
addb = 1;
donea = 0;
doneb = 0;
if (donea == adda) || (doneb == addb)
    true = 0;
else
    true = 1;
end

while ~true
    if (donea == adda)
        w = find(bonds(:,2) == lista(donea+1));
        ngood = length(w);
        if ngood == 0
            listb(addb+1:addb+ngood,1) = bonds(w,1);
            bonds(w,:) = -(nclust+1);
            addb = addb+ngood;
        end
        donea = donea+1;
    end
    if (doneb == addb)
        w = find(bonds(:,1) == listb(doneb+1));
        ngood = length(w);
        if ngood == 0
            lista(adda+1:adda+ngood,1) = bonds(w,2);
            bonds(w,:) = -(nclust+1);
            adda = adda+ngood;
        end
        doneb = doneb+1;
    end
    if (donea == adda) || (doneb == addb)
        true = 0;
    else
        true = 1;
    end
end

 [~,pqx] = sort(listb(1:doneb));
%unx = unq(listb(1:doneb),pqx);
%implanting unq directly
    arr = listb(1:doneb);
    q = arr(pqx);
    indices = find(q == circshift(q,-1));
    count = length(indices);
    if count > 0
        unx = pqx(indices);
    else
        unx = length(q) -1;
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xs2 = length(unx);
 [~,pyy] = sort(lista(1:donea));
%uny = unq(lista(1:donea),pyy);
%implanting unq directly
    arr = lista(1:donea);
    q = arr(pyy);
    indices = find(q == circshift(q,-1));
    count = length(indices);
    if count > 0
        uny = pyy(indices);
    else
        uny = length(q) -1;
    end
ysz = length(uny);
if xsz*ysz > maxsz
    maxsz = xsz*ysz;
    mxsz = xsz;
    mysz = ysz;
end

thru = thru -xsz;
nclust = nclust + 1;
end
bmap = bonds(:,2);

% THE SUBNETWORK CODE ENDS
% put verbose in for Jaci

% THE PERMUTATION CODE BEGINS
for nc =1:nclust
    w = find( bmap == -1*(nc));

    nbonds = length(w);
    bonds = mbonds(w,:);
    lensq = bondlen(w);
    [~,st] = sort( bonds(:,1));
    %implanting unq directly
    arr = bonds(:,1);
    q = arr(st);
    indices = find(q == circshift(q,-1));
    count = length(indices);
    if count > 0
        un = st(indices);
    else
        un = length(q) -1;
    end
    uold = bonds(un,1);
    nold = length(uold);
    %un = unq(bonds(:,2));
    %implanting unq directly
    indices = find(bonds(:,2) == circshift(bonds(:,2),-1));
    count = length(indices);
    if count > 0
        un = indices;
    else
        un = length(bonds(:,2)) -1;
    end
    unew = bonds(un,2);
    nnew = length(unew);
    if nnew > 5
        rnsteps = 1;
for ii =1:nnew
    rnsteps = rnsteps * length(find(bonds(:,2) == ... 
        unew(ii)));
    if rnsteps > 5.e+4
        disp('Warning: difficult combinatorics encountered.')
    end
    if rnsteps > 2.e+5
        disp(['Excessive Combinatorics you FOOL LOOK WHAT YOU HAVE' ... 
            ' DONE TO ME!!!'])
        % try reducing maxdisp
        return
    end
end
st = zeros(nnew,1);
fi = zeros(nnew,1);
h = zeros(nbonds,1);
ok = ones(nold,1);
nlost = (nnew - nold) > 0;

for ii=1:nold
    h(bonds(:,1) == uold(ii)) = ii;
end
st(1) = 1 ;
fi(nnew) = nbonds; % check this later
if nnew > 1
    sb = bonds(:,2);
sbr = circshift(sb,1);
sbl = circshift(sb,-1);
st(2:end) = find( sb(2:end) ~= sbr(2:end)) + 1;
    fi(1:new-1) = find( sb(1:nbonds-1) ~= sbl(1:nbonds-1));
end
%    if i-1 == 13
%        hi
%    end
checkflag = 0;
while checkflag ~= 2
    pt = st -1;
    lost = zeros(nnew,1);
    who = 0;
    losttot = 0;
    mndisq = nnew*maxdisq;
    while who ~= -1
        if pt(who+1) ~= fi(who+1)
            w = find( ok{ h( pt( who+1 )+1:fi( who+1 ) ) } );
            % check this -1
            ngood = length(w);
            if ngood > 0
                if pt(who+1) == st(who+1)-1
                    ok(h(pt(who+1))) = 1;
                end
                pt(who+1) = pt(who+1) + w(1);
                ok(h(pt(who+1))) = 0;
                if who == nnew -1
                    ww = find( lost == 0);
                    dsq = sum(lensq(pt(ww))) + losttot*maxdisq;
                    if dsq < mndisq
                        minbonds = pt(ww);
                    end
                end
            end
        end
    end
end
mndisq = dsq;
end
else
    who = who + 1;
end
else
    if ~lost(who+1) && (losttot == nlost)
        lost(who+1) = 1;
        losttot = losttot + 1;
        if pt(who+1) == st(who+1) -1;
            ok(h(pt(who+1))) = 1;
        end
    if who == nnew -1
        ww = find( lost == 0);
        dsq = sum(lensq(pt(ww))) + losttot*maxdisq;
        if dsq < mndisq
            minbonds = pt(ww);
            mndisq = dsq;
        end
    else
        who = who + 1;
    end
else
    if pt(who+1) == (st(who+1) -1)
        ok(h(pt(who+1))) = 1;
    end
    pt(who+1) = st(who+1) -1;
    if lost(who+1) == 1;
        losttot = losttot -1;
        who = who -1;
    end
end
else
    if ~lost(who+1) && (losttot == nlost)
        lost(who+1) = 1;
        losttot = losttot + 1;
        if pt(who+1) == st(who+1) -1
            ok(h(pt(who+1))) = 1;
        end
    if who == nnew -1
        ww = find( lost == 0);
        dsq = sum(lensq(pt(ww))) + losttot*maxdisq;
        if dsq < mndisq
            minbonds = pt(ww);
            mndisq = dsq;
        end
    else
        who = who + 1;
    end
else
    if pt(who+1) == st(who+1) -1
        ok(h(pt(who+1))) = 1;
    end
    pt(who+1) = st(who+1) -1;
    if lost(who+1) == 1;
        losttot = losttot -1;
        who = who -1;
    end
end
end

checkflag = checkflag + 1;
if checkflag == 1
    plost = min((fix(mndisq/maxdisq) , (nnew -1)));  % end
    if plost > nlost
        nlost = plost;
    else
        checkflag = 2;
    end
end

end
%   update resx using the minimum bond configuration
resx(ispan,labely(bonds(minbonds,2))) = eyes(labelx(bonds(minbonds,1)+1));
found(labelx(bonds(minbonds,1)+1)) = 1;

end
%   THE PERMUTATION CODE ENDS
end

w = find(resx(ispan,:) >= 0);
nww = length(w);
if nww > 0
    pos(w,:) = xyzs( resx(ispan,w) , 1:dim);
    if goodenough > 0
        nvalid(w) = nvalid(w) + 1;
    end
end  %go back and add goodenough keyword thing
newguys = find(found == 0);
nnew = length(newguys);
if (nnew > 0) % & another keyword to workout inipos
    newarr = zeros(zspan,nnew) -1;
    resx = [resx,newarr];
    % garbage collection
    clear newarr;

    resx(ispan,n+1:end) = eyes(newguys);
    pos = {{pos};[xyzs(eyes(newguys),1:dim)]};
nmem = zeros(nnew,1);
mem = [mem;nmem];
nun = 1:nnew;
uniqid = [uniqid,((nun) + maxid)];
maxid = maxid + nnew;
if goodenough > 0
    dumphash = [dumphash;zeros(1,nnew)];
    nvalid = [nvalid;zeros(1,nnew)+1];
end
% put in goodenough
n = n + nnew;
end
else
    warning('No positions found for t=%d',i);
end
w = find( resx(ispan,:) == -1);
nok = length(w);
if nok ~= 0

mem(w) = 0;
end

mem = mem + (resx(1,:)’ == -1);
wlost = find(mem == memory_b+1);
nlost = length(wlost);

if nlost > 0
    pos(wlost,:) = -maxdisp;
if goodenough > 0
    wdump = find(nvalid(wlost) < goodenough);
ndump = length(wdump);
    if ndump > 0
        dumphash(wlost(wdump)) = 1;
    end
end
% put in goodenough keyword stuff if
end
if (ispan == zspan) || (i == z)
nold = length(bigresx(1,:));
nnew = n - nold;
if nnew > 0
    newarr = zeros(z,nnew) -1;
    bigresx = [bigresx,newarr];
    % garbage collection
    clear newarr;
end
if goodenough > 0
    if (sum(dumphash)) > 0
        wkeep = find(dumphash == 0);
nkeep = length(wkeep);
        resx = resx(:,wkeep);
        bigresx = bigresx(:,wkeep);
        pos = pos(wkeep,:);
        mem = mem(wkeep);
        uniqid = uniqid(wkeep);
        nvalid = nvalid(wkeep);
        n = nkeep;
        dumphash = zeros(nkeep,1);
    end
end
% again goodenough keyword
if quiet!=1
    fprintf('%d of %d done. Tracking %d particles %d tracks total\n',i,z,ntrk,n);
end
bigresx(i-(ispan)+1:i,:) = resx(1:ispan,:);
resx = zeros(zspan,n) - 1;

wpull = find(pos(:,1) == -maxdisp);
npull = length(wpull);

if npull > 0
    lillist = zeros(1,2);
    for ipull=1:npull
        wpull2 = find(bigresx(:,wpull(ipull)) == -1);
npull2 = length(wpull2);
        thing = [bigresx(wpull2,wpull(ipull)),zeros(npull2,1)+uniqid(wpull(ipull))];
lillist = [lillist;thing];
    end
    olist = [olist;lillist(2:end,:)];
wkeep = find(pos(:,1) >= 0);
nkeep = length(wkeep);
if nkeep == 0
    warning('Were going to crash now, no particles....');
end
resx = resx(:,wkeep);
bigresx = bigresx(:,wkeep);
mem = mem(wkeep);
uniqid = uniqid(wkeep);
n = nkeep;
dumphash = zeros(nkeep,1);
if goodenough > 0
    nvalid = nvalid(wkeep);
end
if goodenough > 0
    nvalid = sum(bigresx >= 0 ,1);
wkeep = find(nvalid >= goodenough);
nkeep = length(wkeep);
if nkeep == 0
    % Suppress this output. This is handled elsewhere.
    %fprintf('No tracks were found in the input.
')
    tracks = [];
    return
end
if nkeep < n
    bigresx = bigresx(:,wkeep);
n = nkeep;
    uniqid = uniqid(wkeep);
    pos = pos(wkeep,:);
end
wpull = find( pos(:,1) ~= -2*maxdisp);
npull = length(wpull);
if npull > 0
    lillist = zeros(1,2);
    for ipull=1:npull
        wpull2 = find(bigresx(:,wpull(ipull)) ~= -1);
        npull2 = length(wpull2);
        thing = [bigresx(wpull2,wpull(ipull)),zeros(npull2,1)+uniqid(wpull(ipull))];
        lillist = [lillist;thing];
    end
    olist = [olist;lillist(2:end,:)];
end
olist = olist(2:end,:);
%bigresx = 0;
%resx = 0;
nolist = length(olist(:,1));
res = zeros(nolist,dd+1);
for j=1:dd
    res(:,j) = xyzs(olist(:,1),j);
end
res(:,dd+1) = olist(:,2);
% garbage collection
%clear xys;

% this is uberize included for simplicity of a single monolithic code
ndat=length(res(1,:));
newtracks=res;

%u=ung(newtracks(:,ndat));

% inserting ung
indices = find(newtracks(:,ndat) ~= circshift(newtracks(:,ndat),-1));
    count = length(indices);
    if count > 0
        u = indices;
    else
        u = length(newtracks(:,ndat)) -1;
    end

ntracks=length(u);
u=[0;u];
for i=2:ntracks+1
    newtracks(u(i-1)+1:u(i),ndat) = i-1;
end

% end of uberize code
tracks = newtracks;

Published with MATLAB® R2015a

226