Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Schuyler Arn April 10, 2023

Precision particle tracking of discretized light spots in images using machine learning

by

Schuyler Arn

Justin C. Burton, Ph.D.
Advisor

Physics

Justin C. Burton, Ph.D.
Advisor

Alissa Bans, Ph.D.
Committee Member

Jed Brody, Ph.D.
Committee Member

2023

Precision particle tracking of discretized light spots in images using machine learning

By

Schuyler Arn

Justin C. Burton, Ph.D.
Advisor

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements for the degree of
Bachelor of Science with Honors
Physics
2023

Abstract

Precision particle tracking of discretized light spots in images using machine learning
By Schuyler Arn

Particle tracking is a broad field of software with myriad applications, including use in the study of dusty plasma, a system in which colloidal "dust" particles in the μm regime are suspended in a plasma. Dusty plasma can be created experimentally in a lab or found in space and the Earth's upper atmosphere, but attempts to study the dynamics of such microscopic particles are often plagued by pixel-locking, a phenomenon under which estimations of positions for randomly positioned particles in digital images are unnaturally biased towards integer values. This bias is due to the discretization of light detected by digital camera sensors to produce pixelized images, as well as due to the feature detection method used to estimate particle positions. Accordingly, we attempted to train machine learning models on a data set containing simulated images of individual dusty plasma particles with known positions to better estimate the positions and dynamics of simulated test images. We found that the error between the predicted positions found by the rudimentary models, as well as the presence of pixel-locking, and their ability to track a moving particle was comparable to if not better than the extant standard in Python-based particle tracking with TrackPy. Our findings imply that trained machine learning models are capable of becoming more accurate at simulated particle position estimation than pixel-intensity weighted algorithms, and this performance increase possibly extends to testing on actual experimental data. So long as the relationships between the accuracy metrics we employ to quantify method performance persist on experimental data, machine learning models could provide faster and more accurate dynamical measurements for small particles with possible benefits of great biological, astronomical, and physical interest.

Precision particle tracking of discretized light spots in images using machine learning

By

Schuyler Arn

Justin C. Burton, Ph.D.
Advisor

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements for the degree of
Bachelor of Science with Honors
Physics
2023

Acknowledgments

I want to, above all else, thank my friends and family for their unwavering support and belief in my abilities, without which, I'm certain this paper would not exist. Moreover, I am incredibly grateful to Dr. Burton and the rest of the Burton Lab for their invaluable companionship, guidance, and willingness to give me an incredible research experience over the past months. Lastly, I have to thank the lovely Carolyn for always pushing me to step out of my comfort zone—I couldn't have done this without you.

Contents

1 Introduction **1**

1.1 What is Particle Tracking? 1

1.1.1 Broad Applications of Particle Tracking 4

1.1.2 Pixel-Locking and Limitations of Particle Tracking Methods 5

1.2 Dusty Plasma 10

1.3 Machine Learning: A Better Solution for Tracking Dusty Plasma Particles . 12

2 Methods **14**

2.1 Image Generation 14

2.2 Machine Learning Models 20

2.3 Particle as a Stochastic Harmonic Oscillator 23

3 Results and Discussion **26**

3.1 Parameters and Pixel-Locking 26

3.2 Machine Learning Performance and Pixel-Locking 27

3.3 Velocity Distributions and Estimated p Fit 32

4 Conclusion **38**

4.1 Future Research 39

Bibliography **40**

List of Figures

1.1 Illustration of a generalized, computer-assisted object tracking system. . . . 2
1.2 Examples of different particle tracking applications. 6
1.3 Illustration of the discretization of particles when imaged by a digital camera. 7
1.4 Examples of sub-pixel distributions with and without the presence of pixel-locking. 8
1.5 Illustration of our lab's experimental setup for studying dusty plasma. 12

2.1 Comparison between real and simulated dusty plasma particles. 15
2.2 Examples of the effects of varied parameters on image generation. 17
2.3 Process for generating Gaussian images from initial function to noised, discretized array. 19
2.4 A representation of the procedure for training and prediction with the machine learning models. 23
2.5 Translation of SHO time series positions into simulated images. 25

3.1 Pixel locking due to differences in σ. 28
3.2 Average error comparison between the feature detection models. 29
3.3 Sub-pixel distribution for predicted positions from each feature detection model. 30
3.4 Previously found velocity distribution performance. 33
3.5 Result of curve fitting procedure on the SHO time series. 34

3.6 Result of curve fitting procedure on the SHO time series with each particle tracking method. 35
3.7 Average error comparison between the feature detection models' performance on SHO particle. 36

List of Tables

2.1 Tabulation of the parameters selected when generating simulated images and the stochastic harmonic oscillator time series. 16

Chapter 1

Introduction

1.1 What is Particle Tracking?

Object tracking, at its most general, is the use of some kind of electromagnetic or acoustic sensor to determine dynamical information about a target in a number of contexts and different scales. This is a large field of interest, with applications ranging from video surveillance, radar tracking of aircraft and weather balloons, and GPS devices, all the way down to the study of individual cells like lymphocytes [8]. We are most interested in single object tracking on targets with scales comparable to the latter, which is more specifically defined as particle tracking, though the general process of object tracking from the raw signal of a detected observable to digitally treated data and estimated position, as described in Fig. 1.1, is adhered to regardless of scale. A target object must be imaged by a sensor with some detection of energy, which introduces a degree of electronic noise in the process. The data from this detection must be processed before passing this information to the tracking algorithm, at which point the measurement is made, the data is gated to treat noise and determine the valuable information contained in the measurement, and an estimate of the particle's position is produced. Particle tracking, then, as it derives from object tracking, is this process of determining the position of one or many microscopically small particles as

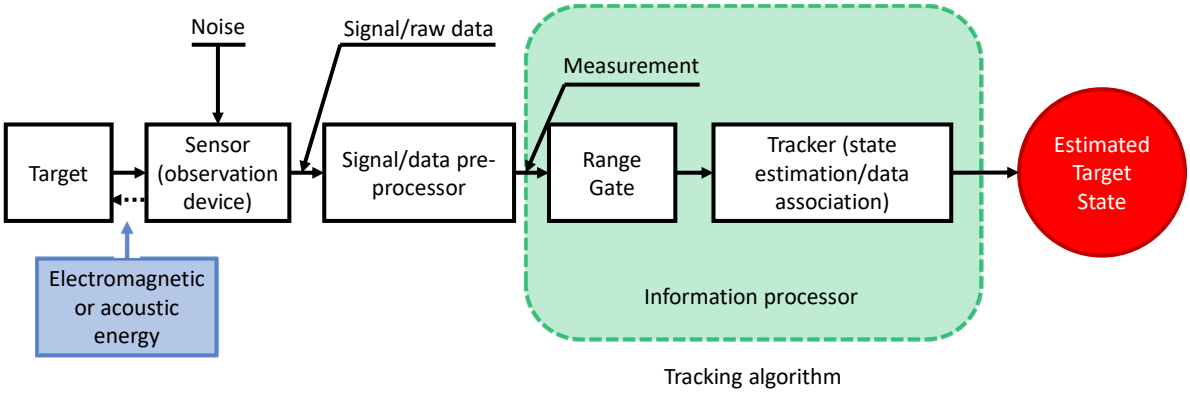

Figure 1.1: Illustration of a generalized, computer-assisted object tracking system. Reproduced from [8].

they move throughout their confinement space, though I will often use the phrase "feature detection" interchangeably when discussing the process of finding the particle's location in a single frame or at a specific moment in time.

This feature detection can be done either manually or with the assistance of a computer, using a number of image processing programs. The former introduces some degree of human error into feature detection and can be very time-consuming when large amounts of data or multiple particles are processed. However, a trailblazing paper by Crocker and Grier showed off the possibilities of computer-assisted feature detection on colloids in digital videos, and their work continues to influence the field today with the standardization of automated particle tracking [11]. Importantly, they recognized that the imaged colloids create pixel intensity patterns that can be modeled by a Gaussian surface of revolution, as generally shown in Eq. 1.1, where $\mathbf{r}_0 = (x_0, y_0)$ describes the estimated origin of the particle, A_0 is the maximum intensity of the light scattering from the particle, and s is the particle's apparent radius or the Gaussian's half width at half maximum.

$$A(x, y) = A_0 \exp\left(-\frac{2|\mathbf{r} - \mathbf{r}_0|^2}{s^2}\right) \tag{1.1}$$

Countless research efforts in this field have since relied on and built upon this model,

and by extension, the scope of possible imaging scenarios is often defined by the ability of a particle or feature to be emulated by the Gaussian model. Moreover, the procedure that the paper describes of initially assuming the brightest pixel present as containing the feature's center and then refining that guess with a weighted offset from the intensities of the surrounding pixels is still used today. This weighting methodology is described by an analog of the classical center of mass equation, where the neighboring pixels, $\mathbf{X}_k$ are the different points of measurement about the space and their intensities, I_k, are regarded as their corresponding "mass" to find the calculated centroid coordinate, $\mathbf{X}_{calc}$, as shown in Eq. 1.2 [12].

$$\bar{x} = \frac{1}{M}\sum_i x_i m_i \longrightarrow \mathbf{X}_{calc} = \frac{\sum_k \mathbf{X}_k I_k}{\sum_k I_k} \tag{1.2}$$

This methodology is still championed by the original IDL program made by Crocker and Grier, along with a number of successors developed across many other programming languages. For example, MosaicSuite and ImageJ allow for both automated or manual selection of features per frame by users in prepackaged programs, while Python's TrackPy library also bases its feature detection process on this procedure for broader code implementation [3, 28]. Many modifications to TrackPy have been made over the years, ranging from general improvements in efficiency and data processing to an update to its algorithm that allows for refined feature accuracy by using the previously estimated particle positions to inform predicted coordinates and assist in linking detected particles over time [30]. TrackPy's ability to link traveling features over time parallels another, similar category of dynamical particle tracking methods, which is known as particle image velocimetry, or PIV. PIV involves determining the local velocity in small sections of an image or video frame to produce a vector field of velocities and is often employed in fluids studies, while lower-density images enter into the regime of particle tracking velocimetry, or PTV, which is often used to examine dynamics of individual or smaller numbers of particles [2].

Alongside the Crocker and Grier tracking algorithm, there are dozens of alternative options for particle tracking programs available, with a number of data processing methods employed beyond the original Gaussian model. Chenouard et al. even produced a survey comparing of multiple methods of feature detection and trajectory linking on four simulated data sets to see if a particular method was ideal. While no silver bullet method appeared, working perfectly regardless of application, they remarked both that particle density and the signal-to-noise ratio present in the image were notable determinants of performance and that the models themselves were all largely similar [9, 29]. This suggests that the development of particle tracking models which largely adhere to previous tracking implementations while adding improved parameter selection and treatment or processing of provided data could be the key to future improvements in the field.

1.1.1 Broad Applications of Particle Tracking

There are a number of research applications where knowing the precise locations or dynamics of microscopic or distant objects are of great benefit or interest, such as when investigating biological processes, astronomical bodies, and as we will extensively discuss, dusty plasma. In the realm of biology, particle tracking has been used to study the dynamics of individual cells in developing embryos, among other hard-to-differentiate groups of small bodies [7]. Another often-used experimental procedure for biological and dynamical study involves the insertion of fluorescent beads or a polymer into a region of interest, be that in cells of human skeletal muscle fibers, a fluid, or the cytoskeleton of an individual cell's cytoplasm, so as to signal or even cause microscopic dynamics with the movement of these small markers [6, 13, 16, 31]. Expansion microscopy is name for the latter technique, and it is employed when the polymer present is used to physically expand or alter the structure of the target of study, rather than simply promoting visibility of desired features.

Particle tracking is also providing insights into the nature of comet trails now that we have the ability to capture images from probes orbiting remote comets. Pfeifer et al. developed a

new algorithm in Python specifically for tracking the dust and other material ejected from comet 67P in imaging captured by the Rosetta probe in early 2016, which works to detect individual particle positions and then uses a paired tracking between neighboring frames to calculate other dynamical properties [21].

These instances are generally colloidal, but TrackPy's documentation even includes examples of more specialized tracking applications used in published research such as imaging moving clusters of bubbles in a foam, or even non-colloidal imaging of particle rings in bright-field microscopy [25, 27]. Importantly, the reference to a specific "particle" in the more general process of particle tracking can be expanded to just about any object that occupies a small neighborhood of pixels in the captured image—allowing for the use of these same particle tracking methods in the astronomical study of stars and other bodies, so long as the similar appearance of these objects to the colloidal analog is maintained. Examples of particle tracking use in a number of fields is shown in Fig 1.2. The key operation for the use of particle tracking on objects with a range of sizes is the scaling of smaller or larger objects into meso-scale digital images for computer analysis, though this transition is often accompanied a number of issues.

1.1.2 Pixel-Locking and Limitations of Particle Tracking Methods

While TrackPy does an admirable and efficient job of locating the pixel-scale position of image features, the algorithm often leaves much to be desired where the sub-pixel resolution of particle positions are concerned. This becomes particularly nascent when attempting to take measurements of feature dynamics of microscopic particles, such as velocity, wherein errors in the found location of the particle contribute to greater inaccuracy in these further or derived measurements. These errors most commonly take the form of a systemic bias introduced with digital imaging known as pixel-locking or peak-locking.

Pixel-locking is a problem that arises due to the limited pixel resolution offered by a digital camera sensor, causing a loss of intensity information which tends to cluster the sub-pixel,

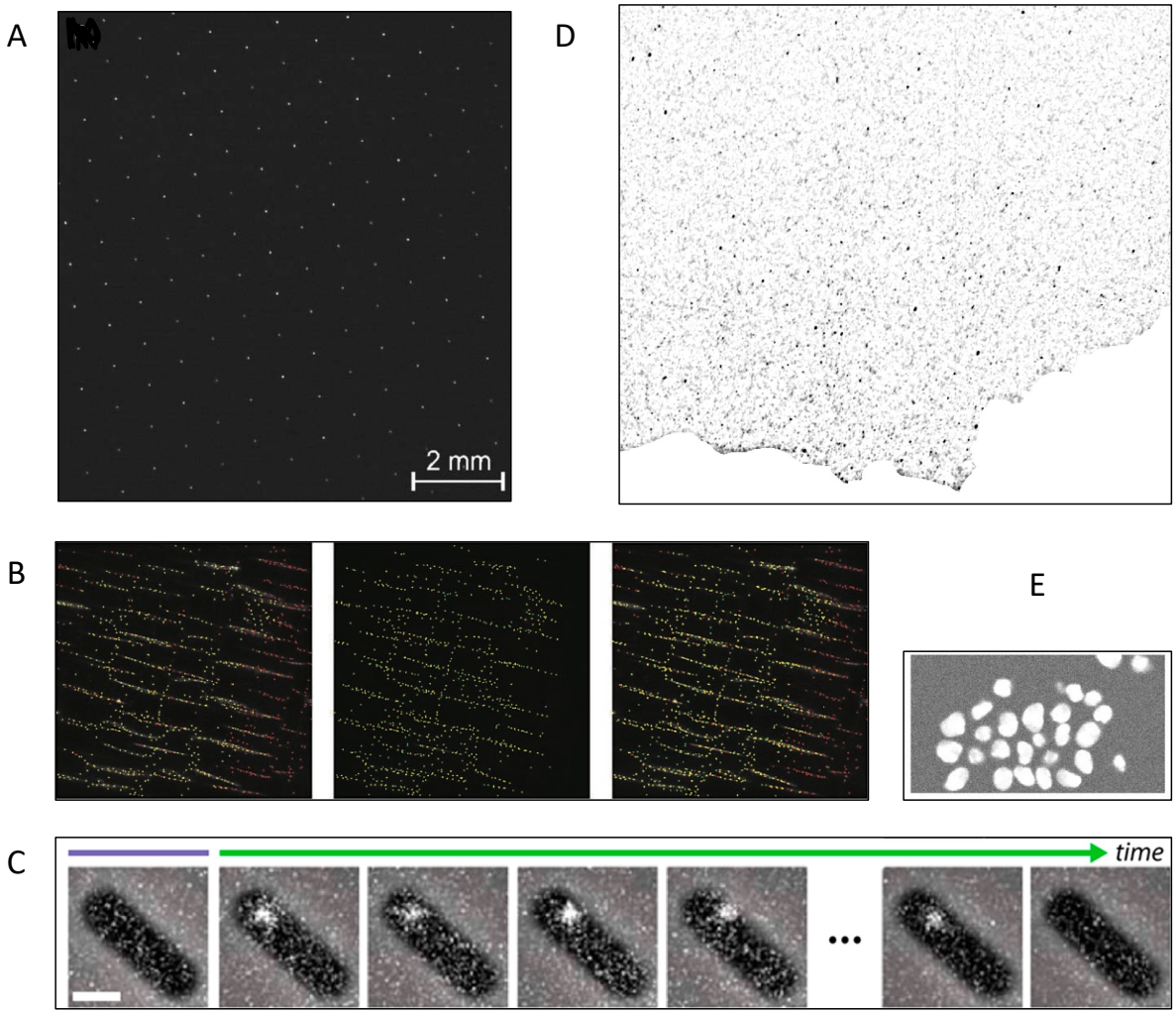

Figure 1.2: Examples of different particle tracking applications. (A) depicts an experimental bitmap image of a crystallized, monolayer suspension of micro-spheres in a dusty plasma [12]. (B) is a colorized image of NET1A-green fluorescent protein signals in plant cells from *Arabidopsis thaliana*, a small flowering plant [19]. (C) shows the photoactivation of a single copy of MutS-PAmCherry, a protein related to DNA repair, in a cell [29]. (D) is a cleaned and brightness-inverted image of the dust ejected from the surface of comet 67P as captured by the Rosetta probe [21]. (E) is a frame from a sequenced image example of cells in a developing embryo [7].

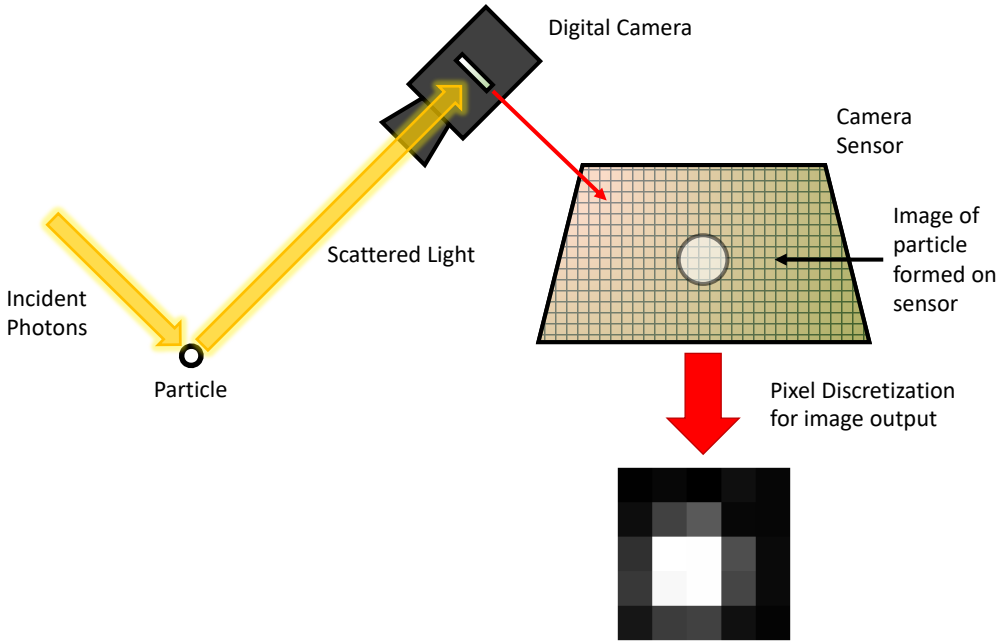

Figure 1.3: Illustration of the discretization of particles when imaged by a digital camera.

decimal component of particles' found position about integer values representing the edges of the pixels [10, 12, 22]. This ultimately makes accurate particle position detection without pixel-locking bias a kind of discretization problem, as illustrated in Fig. 1.3. Incident light that scatters from the object of interest reaches the digital camera sensor, which has both a limited resolution and the need to translate the random impact of photons on the sensor into digital values of pixel intensity. With the current inability to develop a digital camera with such high resolution as to image microscopic targets without discretization, requiring the resolution of the camera sensor to rival the size of the colloidal target, pixel-locking continues to affect digital cameras and other imaging methods. Additionally, examination of the centroid methodology for determining particle position has found it contributes to biasing towards the center of a pixel when selecting random subpixel positions for features [19]. In each instance, this bias is obviously problematic, whether due to the camera or the analysis method, because the sub-pixel coordinates of truly randomly positioned particles in space are expected to follow a uniform distribution, under which no particular sub-pixel value or position would be preferred by the particles. The presence of pixel-locking is then suggested

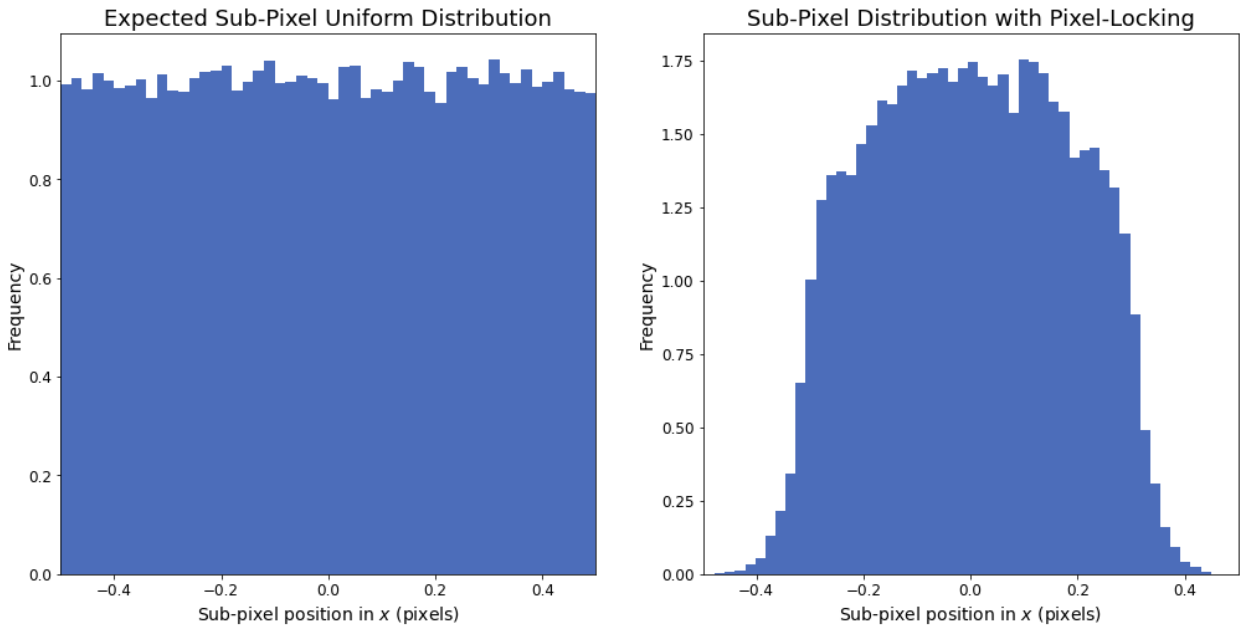

Figure 1.4: Examples of sub-pixel distributions with and without the presence of pixel-locking. The heavy pixel-locking instance comes from TrackPy performance in x on a randomly generated, 100000-element image set.

and can be estimated by viewing sub-pixel position distributions for the predicted particle positions, which feature peaks about preferred coordinates at the boundaries or center of the possible values, as shown in Fig. 1.4. Notice the flatness of the uniform distribution to the left of the image, while the image on the right exemplifies a biasing towards the whole integer values with greater density of predicted particle positions appearing about the center at 0, while less frequently appearing about the edges of the distribution approaching ± 0.5. Importantly, however, these plots are generated from the averaged predicted positions in both x and y. While we can make some initial assessments on the degree of pixel locking present with this one-dimensional representation, valuable information is gained by using a 2D plot to visualize pixel-locking effects due to x and y separately.

The camera used to image particles in tracking is also the source of additional errors and biases beyond pixel-locking, often coming from the electronic noise of the digital sensor, shot noise, and other random errors [10]. Shot noise, or photon noise, is of particular interest, coming about due to the photons reaching the camera sensor following an uneven Poisson

distribution, while other types of noise are introduced depending on the type of camera and sensor used [26, 29]. However, given their systemic and often immutable nature, as side effects of the imaging system in use, we have to work around this noise and attempt to reduce error with more direct confrontation of pixel-locking.

While some observables, such as ensemble averages of PIV-found velocity fields, can be treated or averaged to minimize or largely negate the effects of pixel-locking, the majority of other particle characteristics require alternative solutions [10]. One treatment for the problems that arise from pixel-locking in feature detection is the single-pixel interior filling function or SPIFF. SPIFF attempts to remedy pixel-locking by integrating or summing over the cumulative distribution of estimated positions in the given dimension to make the particles' found positions assume a uniform distribution as would correlate to a truly random, unbiased estimation of position [5]. SPIFF is rather easy to implement, and accordingly, does a great deal in providing performance and accuracy improvements over the initial feature detection of TrackPy. Yifat et al. studied the efficacy of SPIFF in correcting the mean error of tracked, simulated data and found a sizeable reduction of up to 0.19 pixels in the mean error and 0.04 pixels in the standard deviation between the predicted and ground truth positions [32]. However, when compared to the reduced error and faster performance of machine learning models when it comes to feature detection on individual simulated images, it seems that SPIFF is often only partially effective in its goal of increasing detected feature position accuracy by reducing the effects of pixel-locking.

One solution proposed as an alternative to more standard fixes for pixel-locking, like the general histogram-equalization process of SPIFF or defocussing/blurring the image in the context of PIV, is the treatment of each vector individually due to the non-uniform nature of pixel-locking across the imaging space. Because integer values might be often reached at boundaries, pixel-locking is a localized phenomenon, and errors can be generated unevenly across the field by the camera. Accordingly, the use of per-vector histogram-equalization produces a series of transfer functions for each location in the vector field, which tends to

be more accurate than non-specific application [15]. Alternatively, Adatrao et al. suggest a method for reducing pixel-locking bias in PIV by calculating a regression of dynamic measurements over a range of time steps, as a means of smoothing over the effect present under a constant Δt value [1].

1.2 Dusty Plasma

The interest in machine learning for increased precision in determining particle positions came about because our lab also conducts experiments to study the nature of dusty plasma, a low-pressure plasma that is found in planetary rings and comet tails, among other scenarios [22]. Wentao Yu, my mentor and a graduate student in the Burton Lab, has recently been working to understand the presence of non-reciprocal forces exerted on the dusty plasma particles. These come about due to differences in the vertical position and charge on each particle, along with ionic perturbations in the plasma due to the charge of the electrode. Achieving greater accuracy in particle position estimations is then an ongoing struggle of great importance in this space due to the spatial dependence present in force calculations and the dynamics of these particles, as examining their velocities (or acceleration) requires differentiating over their found positions with respect to time. Accordingly, any errors in found particle location, such as those arising from pixel-locking, are immediately compounded and undesirable when a greater degree of accuracy in measurement or use in testing experimental data against simulated models is needed.

Experimentally, the plasma of study is created in a stainless-steel vacuum chamber which is filled with diatomic nitrogen and kept at very low pressures, typically in the range of 0.3mTorr or less. This experimental setup is illustrated in Fig. 1.5. When subjected to a voltage, plasma is generated in the chamber, while the presence of a bias voltage, V_{bias}, on the electrode in the bottom of the chamber creates a repulsive force on the particles to oppose gravity. Also within the chamber are multiple particulate reservoirs which we can use to

manually deposit or "shake" dust particles into the plasma. We use melamine-formaldehyde particles produced by microParticles GmbH with diameters ranging from 4.90 μm to 12.8 μm. Once the particles are suspended at some height in the plasma, we use a diode laser with a wavelength of 538.2 nm and the two lenses pictured to create a laser sheet across the confinement space. With the laser sheet, we can use a high-speed camera to visualize the movement of the individual particles and the formation of crystalline structures. The incident photons in the laser sheet hit the particles and are occasionally scattered in a process of Mie scattering, the light from which can be detected by the digital camera above. It is important to note the particle dynamics resulting from modulating the V_{bias} through the radio-frequency power supply, which can excite the particles out of crystallization and into oscillatory traversal about the confinement space at speed [14, 18].

Besides oscillation or crystallization in a 2D plane, the dust particles are also able to oscillate in the z-direction, leading to the use of the galvo mirror and a connected function generator to oscillate the laser sheet up and down across the vertical space. While the thickness of the laser sheet is quite large compared to the size of the particles ($\approx 100\,\mu$m), their vertical movement often requires using the sweeping motion of the laser sheet to image the particles throughout the chamber [14]. With adequately high frame rates on the imaging camera filming from above, the vertical position of the particle can also be extrapolated as the height of the laser sheet is sliced across a series of frames. Accordingly, experimental data can effectively be translated between two or three spatial dimensions against a time dimension for analysis by simply skipping empty or 'off-height' frames to only view those in the series where the particle is at a constant height.

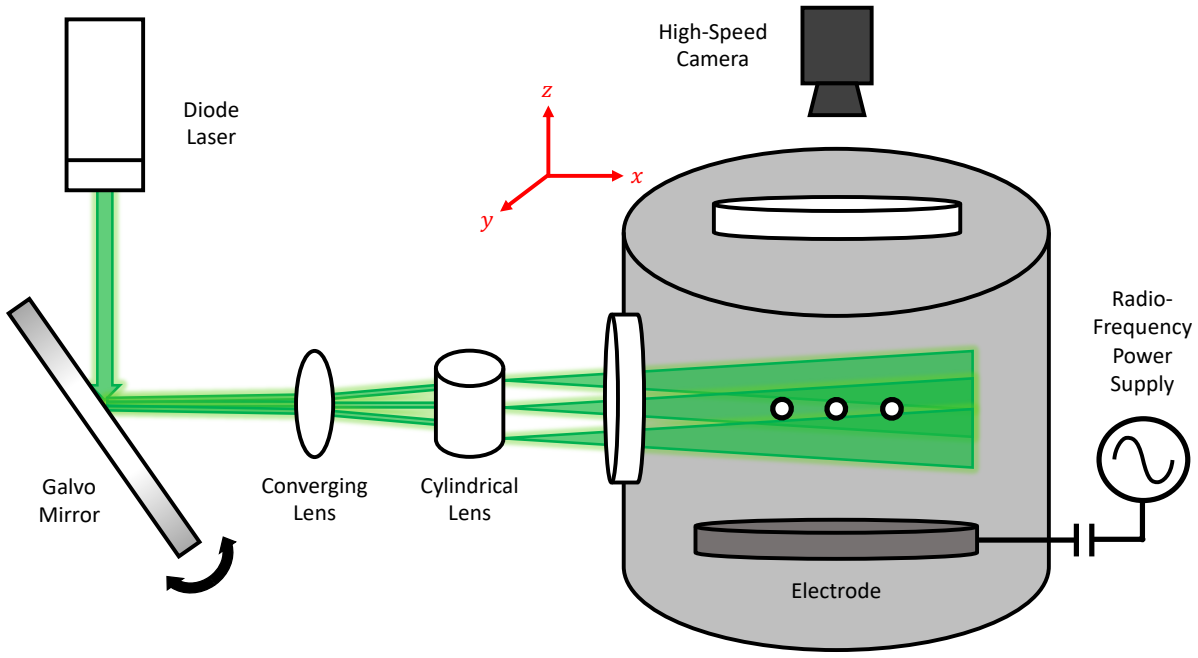

Figure 1.5: Illustration of our lab's experimental setup for studying dusty plasma. Reproduced from [14, 33].

1.3 Machine Learning: A Better Solution for Tracking Dusty Plasma Particles

Machine learning is already a transforming force across current scientific research, as neural networks are assisting in the solving of categorization and regression tasks in a number of fields. Within cellular microscopy, one extant application of neural networks is training them to detect markers that, while hard to see with the human eye, signify states of dynamic muscle fiber characteristics, among other biological processes [16]. Accordingly, the use of a machine-learning approach to particle tracking came about from the hope that we could somehow exploit the unknown, 'black-box' nature of machine-learning algorithms to find a better way of determining particle coordinates than we are currently able. This describes the often unknown nature of the means by which neural networks can make connections between data points to arrive at a desired result. By training the machine-learning models on a large

data set containing millions of simulated images of particles and the corresponding origin coordinates we used to produce said images, we hoped that the models would be able to find some sort of hidden relationship between the position of the particle and its discretized, digital representation.

The search for this relationship describes a typical regression problem for machine-learning models, as opposed to a classification problem. The former has continuous values as its output while the latter is related to more categorical data, such as the common computer vision problem of determining whether an image depicts a member of a particular group. These might describe choosing if an image contains a cat or a dog, or whether the sensors on a semi-autonomous car are detecting a pedestrian, car, or truck. Instead, for a regression model, we generally describe the system as some order of polynomial weighted by a series of parameters, as shown in Eq. 1.3, which we teach the model to optimize for through a supervised learning process [4]. Within our particular use case, the "features" or individual pixels of the images that we will pass to the machine learning models become the x^n terms described, as the models must work to discover a relationship between the desired coordinates of the given particle and intensity pattern expressed by the image.

$$y = w_0 + w_1 x + w_2 x^2 + ... + w_n x^n \tag{1.3}$$

Python libraries already allow for the easy implementation of this learning process, only requiring the selection of a model, training data set, and test data set—which we hope to employ to make a straight-forward and potentially better-performing feature detection model than already offered by TrackPy.

Chapter 2

Methods

2.1 Image Generation

To generate simulated images of dusty plasma particles for testing the feature detection capabilities of TrackPy, training new machine-learning models, or testing said models, we had to develop a pipeline for producing large batches of images in series. The product of said image simulation along with a comparison against actual experimental data of a particle suspended in dusty plasma is demonstrated in Fig. 2.1. Generating features that appear visually similar to individual experimental particles is trivial at small scales where the feature is only around 3 pixels wide. At the same time, however, this figure only includes one simulated image, the parameters of which I chose to closely approximate the actual particle. The generalizability of the data we train machine-learning models on is of great concern, as a model can only accurately predict positions for particles that are similar in nature to those it is already trained on. To then ensure that the simulated particles describe a breadth of possible experimental scenarios for accurate training of models, including programmed variation of parameters was necessary. This was done by initially simulating images within a 5×5 array, where we could corral the origin of the particle to a single central pixel about the origin, with bounds in both x and y given by the interval $-0.5 \leq (x, y) \leq 0.5$ for a Cartesian

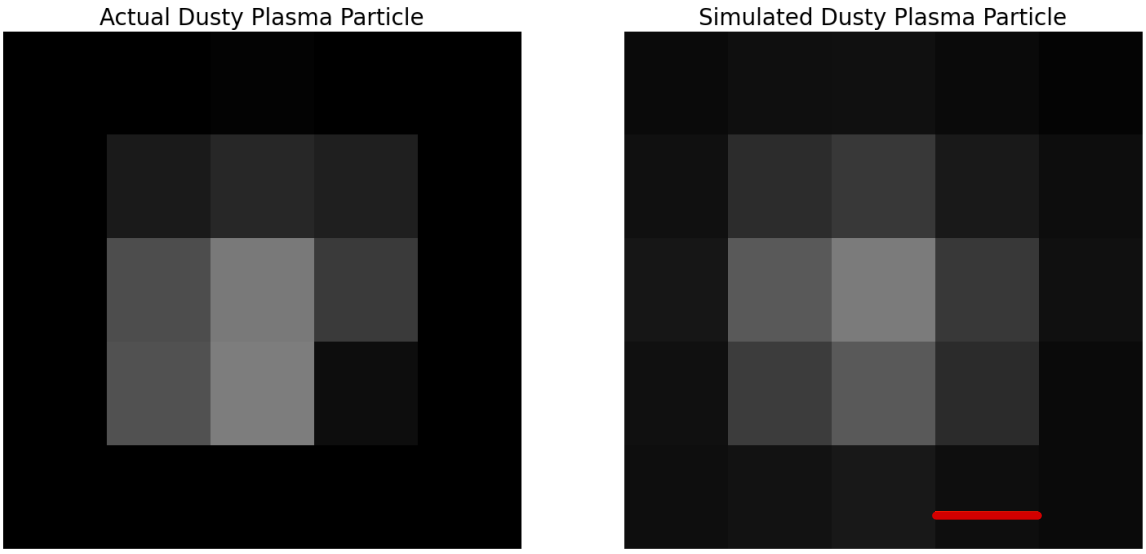

Figure 2.1: Comparison between real and simulated dusty plasma particles. Scale bar is 1 pixel $\approx 50\,\mu$m. The actual particle was imaged experimentally with a Phantom high-speed CCD camera and then cropped to fit a 5×5 pixel resolution. Simulated image parameters employed were $A_0 = 120$, $\sigma = 1.1$, $\alpha = 2$, $\mu = 8$, and $x_0 = y_0 = -0.2$.

plane. Initially, we described these bounds with reference to the indices of the generated arrays, which instead would place the particle within $1.5 \leq (x, y) \leq 2.5$ and require a translation for further analysis, but the use of Numpy's `meshgrid()` method alleviated this issue. This simulation involved using an expanded version of the Gaussian expression that Crocker and Grier relied on, as given by

$$A(x, y) = A_0 \exp\left(-\left(\frac{|x - x_0|^\alpha}{2\sigma^\alpha} + \frac{|y - y_0|^\alpha}{2\sigma^\alpha}\right)\right) + N_p(\mu). \tag{2.1}$$

There are a number of parameters contained within this expression, including the origin coordinates of the feature, which are randomly selected from uniform distributions and influence the appearance of the features in the generated data set. A_0 is the maximum amplitude or pixel intensity of the Gaussian peak, while σ determines the width of the peak and α determines its sharpness. See Table 2.1 for a list of the ranges of values these parameters pulled from, along with Fig 2.2 for a visualization of the simulated image differences due to changes in each parameter. By gradually tapering the interval for each distribution over

generations of simulated data sets and tests for the capabilities of our detection methods, we were able to settle on a set of parameter ranges that can accurately approximate actual μm-scale particles and be detected by each detection method. For example, smaller values of σ can result in the Gaussian function peaking entirely within the space of a single pixel. This minimizes the impact seen in neighboring pixel brightness and can, in turn, prevent TrackPy from effectively determining a feature's sub-pixel position when its "center of mass" detection method is only able to weigh neighboring pixel intensity comparable to noise. Crocker and Grier initially found the best outcomes from choosing camera and magnification settings so that the imaged particle radii ≈ 3 pixels, so we opted for $0.6 \leq \sigma \leq 1.2$ [11]. Similarly, smaller values for A_0 and the signal-to-noise ratio can impede efforts to distinguish particles from noise, leading to the inaccurate detection of multiple non-existent, ghost features, so A_0 was given a minimum value of 100 while N_p, the Poisson-distributed noise we add to the Gaussian, has a maximum mean value of $\mu = 20$. This ensures that each simulated image is generated with a signal-to-noise ratio ≥ 5, which is widely recognized as a kind of lower bound for acceptable performance with many feature detection methods [9].

Table 2.1: Tabulation of the parameters selected when generating simulated images and the stochastic harmonic oscillator time series.

Parameter	**Values**	**Selection**	**Description**
x_0	[-0.5, 0.5]	Uniform Distribution	Feature Origin in x
y_0	[-0.5, 0.5]	Uniform Distribution	Feature Origin in y
A_0	[100, 255]	Discrete Uniform Distribution	Amplitude of Intensity
σ	[0.6, 1.2]	Uniform Distribution	Feature Width
α	[2, 6]	Uniform Distribution	Feature Sharpness
μ	[5, 20]	Uniform Distribution	Mean of Poisson Noise
γ	0.9	Constant	SHO Damping Coefficient
ω	$1.8\,\mathrm{s}^{-1}$	Constant	SHO Frequency

Image creation occurs in three general steps, as shown in Fig. 2.3. From the employment of the first term in Eq. 2.1 with randomly selected parameters to generate the initial Gaussian on the blank, square image array with size 5×5 or 7×7. Many of the pixels, particularly those about the borders of the image, still have an intensity value which is either zero or incredibly

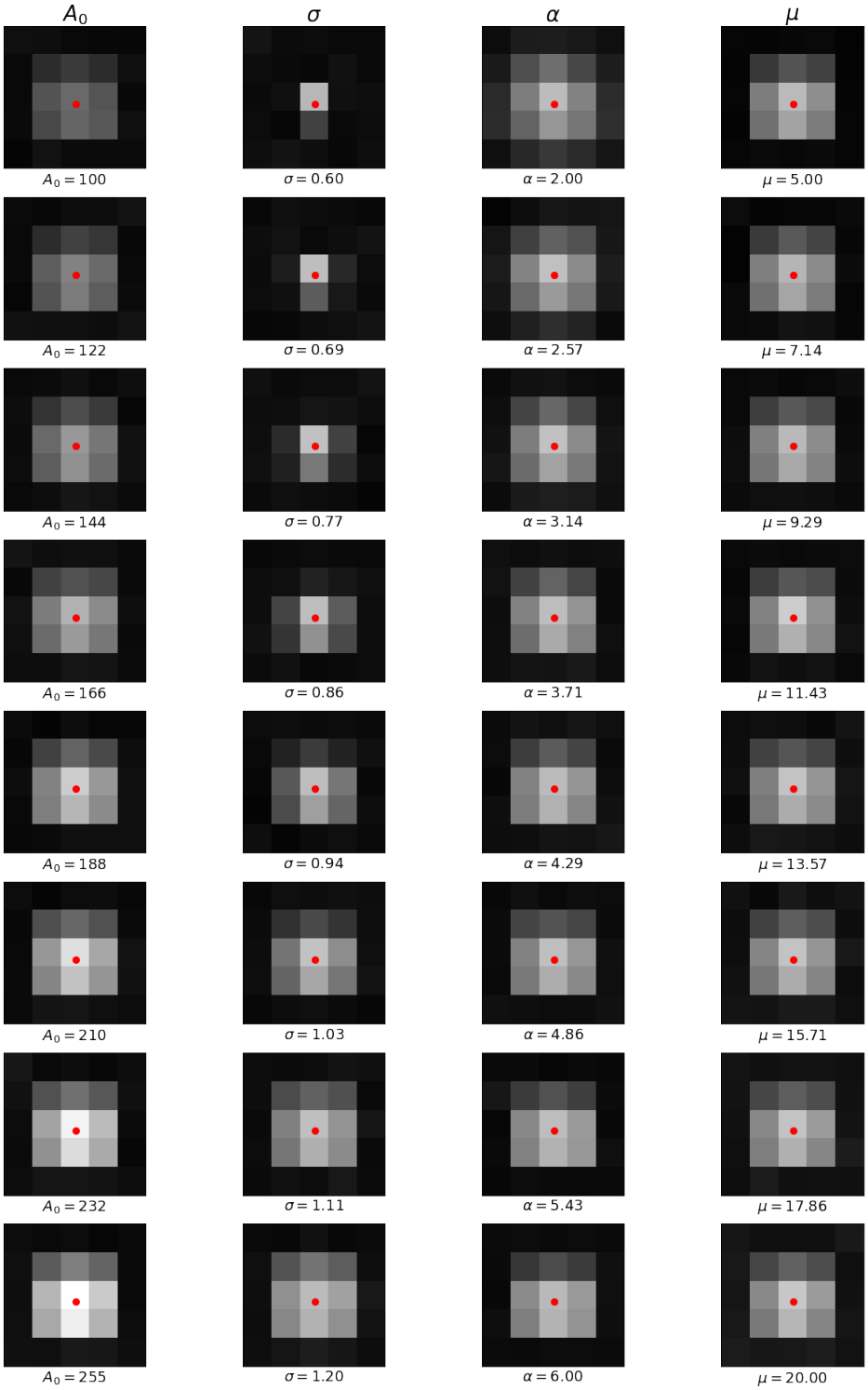

Figure 2.2: Examples of the effects of varied parameters on image generation. If not altered, the parameters held fixed in this image set were $A_0 = 140$, $\sigma = 1.4$, $\alpha = 4$, and $N = 12$. Each image generated shares an identical origin of $x_0 = -0.352$ and $y_0 = -0.314$, as marked with the red circles.

small due to the exponential decay present in the function. Because this does not reflect the degree of noise or pixel intensity values present in experimental images, Poisson-distributed noise, N_P, is then added over the array. The decision to use specifically Poisson-distributed noise in simulation came about due to the nature of the aforementioned shot noise and the sensor of a digital camera effectively functioning as a photon detector when creating digital images of real particles. There can only be an integer number of photons impacting each pixel area of the sensor, along with only one "brightest" pixel for a single particle when correctly focused. These characteristics along with the translation of the captured light to three-channel, 8-bit integer RGB color values ranging from 0-255 in the third step mean that Poisson-distributed noise makes for an accurate approximation for the noise present when filming particles moving in experimental dusty plasma. Applying the equivalent of this translation from the analog to the digital when producing the simulated images also required care for the instances that could see the peak of the Gaussian acquire a value greater than 255 once we applied noise. For Python, visually representing an array where this is the case simply subtracts 255 from that pixel's intensity value, causing what should be the brightest pixel in the image to unrealistically become the dimmest. In reality, that light would simply flood or blow out the digital camera's sensor with photons and reach the maximum brightness possible. Accordingly, we had to add conditions for the conversion of the generated array from the continuous Gaussian output to discrete integer color values without deviation from the expected behavior of the real imaging. These processes are all packaged into a single function, allowing for reproducibility and easy alteration of the generation parameters across data sets of any size, though the computational demand of enough memory to store the data of each image array is a limiting factor when making image sets. However, this is concern is reduced thanks to the small image size we used and storing only one channel of the image intensity.

In one last addition to the initial procedure employed for generating simulated images, we also increased the size of the confinement space for the particle depending on the use

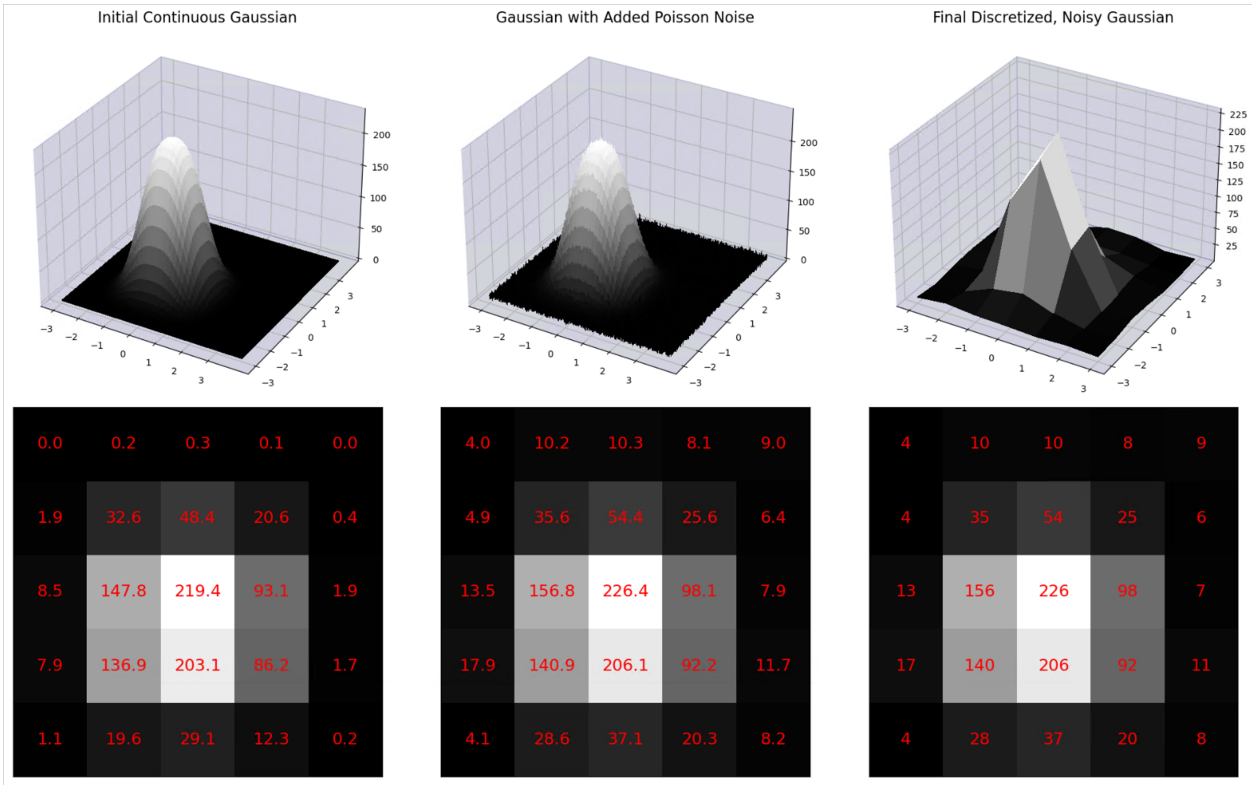

Figure 2.3: Process for generating Gaussian images from initial function to noised, discretized array. Greater resolution for the continuous representations is used for visualization, while computational efficiency requires that the actual images are generated from their final array size. This image was generated with Gaussian parameters of $A_0 = 233$, $\sigma = 0.876$, $\alpha = 2.724$, $\mu = 6.988$, $x_0 = -0.141$, and $y_0 = 0.935$.

case. This provides greater variation in particle origin, ensures that movement of a driven particle would be captured, and prevent possible "edge cases" causing issues for TrackPy, which were further reduced by framing the enlarged image in an array with an additional 2 or 3 pixels of zero intensity "matte."

2.2 Machine Learning Models

To implement machine learning into the research, we utilized the scikit-learn library for Python, which provides a number of regression models and accompanying methods for treating them that we can use to predict sub-pixel positions of features in images [20]. We wanted to use machine learning as a "black box" into which we could pass known, ground truth position vectors along with the corresponding simulated images for the methods to generate models specifically trained to predict features within our produced data sets. Recognizing that this task would require the use of a regression model to produce the desired coordinate estimations, we considered each option that scikit-learn offers for regressors. We settled on testing three different machine-learning models, namely Multi-Level Perceptrons Regression or MLP, Linear Regression or LR, and Histogram-based Gradient Boosting Regression, or HGB. Additionally, each model required a number of particular arguments to modify operating parameters and ensure that they could handle the number of training features present in the simulated images, which equaled the number of pixels present.

MLP is a neural network model which can work as a non-linear function approximator, taking the input features of the images and assigns each to a neuron in the first layer of the model, known as the input layer [20]. This process then establishes each pixel as a weighted term in the regression form of Eq. 1.3, before translating this into a chosen activation function, such as the hyperbolic tangent function, $\tanh(x)$. This supervised learning process continues through the arranged layers of neurons as the model works to minimize the square error function between the paired ground truth positions we provided and the

predicted coordinates. Because of the assignment of image pixels to features, we had to ensure that the layers of perceptrons were wide enough to capture the total number of pixels or elements present in the image arrays. This communication between layers of neurons is where the "black-box" nature of the machine learning comes in, as the predicted coordinates are produced from the connections the model forms between the ground truth points and the corresponding element of the flattened intensities row vector. As a model with native support for multi-output training and prediction, we can pass the ground truth positions as a vector of size $(n \times 2)$, instead of creating separate column vectors for x_0 and y_0 to individual MLP models for each dimension. The latter approach is capable of causing disparate performance between each model due to what we've termed the 'wraparound issue,' wherein the scikit-learn limitation of a 1D features array might give x-coordinate prediction an advantage over the same prediction in y. The flattened intensities row vector is arranged in row-major order, such that each row of pixels in the simulated image is appended to the preceding one. Neighboring pixels in each row remain neighbors, possibly providing greater context about the particle position in x, while the model is left to infer the relationship between neighboring pixels in each column, which now occupy each m^{th} element of the array if m is the array size. Nonetheless, this possibility partly informs our decision to average over both x and y predictions when engaging in error analysis for each model.

For the linear regression model, we had to treat the data and slightly alter the model to ensure that its performance was comparable to the other two models. As a single linear regression implementation, we are again referencing the general form of Eq. 1.3, where the prediction comes from minimizing an ordinary least squares difference between the given data set and an linear approximation [20]. However, we should note that there are some drawbacks for using such a simple model. It is easy for the model to be overwhelmed by the number of features contained in the simulated images passed to the model, greatly increasing the error between predicted and actual positions to be so large that the model's predictions become effectively worthless. Accordingly, we treat the data with scikit-learn's

built-in QuantileTransformer method, which processes the features of the simulated images to follow a uniform distribution, as expected from our generation methodology. Next, the complexity of the model is reduced by using the PolynomialFeatures method to limit the relationships formed between features to a second-degree polynomial or less. Finally, we need to promote the inputs and outputs of the model to handle the same paired ground truth positions array as the MLP training process, which is done by the MultiOutputRegressor method. A beneficial side effect of this promotion is the greater computational power that we can now provide the model, speeding up the training process for the linear regression to be much faster than both of the other models.

Histogram-based gradient boosting regression was selected for its increased performance benefits over the regular gradient boosting regression model when operating on a very large data set, as we intend to do. This improvement comes from the pre-binning of data into histogram bins instead of the model working through sorted continuous values, while still using squared error as its loss function. Preparing the model requires less effort than the linear regression model, but we still must promote the inputs and outputs of the model to handle the same paired ground truth positions array as the MLP training process with the MultiOutputRegressor method. Limiting the maximum depth and number of iterations to 3 and 200 respectively alters the nature of the decision trees that constitute the model, meaning that the model can produce 200 trees and each now has 3 "edges" between the root and deepest leaf [20].

In the interest of ensuring that the memory required to generate a large set of images and parameters would remain low while slightly increasing the image size to allow for more particle exploration of the space, we altered the simulated image generation to fill a 7×7 array for the final $n = 2500000$ element image set. Regardless, the process shown in Fig. 2.4 is still generally applicable to the training and prediction stages of the machine learning models beyond the selection of a particular n and array size.

To train the machine learning models on finding the particle positions, we enforced an

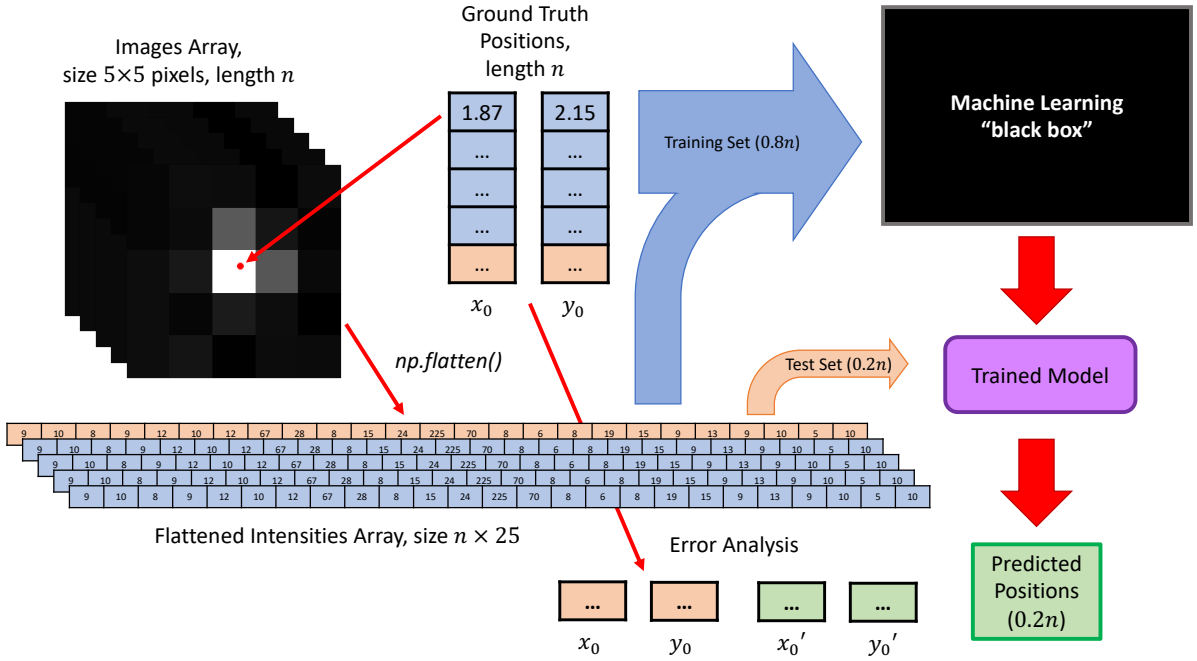

Figure 2.4: A representation of the procedure for training and prediction with the machine learning models.

80/20 training and testing split, such that 80% of our data set's generated images and known particle positions were passed into each model, while the remaining 20% of the generated images were used as the basis for position predictions by the now-trained model. Error analyses on the accuracy of the predictions were then carried out by comparing said predictions against the already-known positions of the remaining 20% test set used to initially generate the images for the predicted set. In this process, the models are never given the test set positions, so the performance on the test image set is both entirely determined by the larger training set and directly comparable to data unknown to the model to avoid overfitting or simply regurgitating provided values.

2.3 Particle as a Stochastic Harmonic Oscillator

To study the dynamics of particles that are not only simulated to appear like the actual dusty plasma particles we hope to study, but also moving as they do, we then turned to

Mathematica in using an Ito process to generate a time series solving an under-damped Stochastic Harmonic Oscillator, or SHO. This oscillator is driven by the Brownian motion of a Wiener Process as a surrogate for the traversal of particles within the dusty plasma field of vacuum chamber experiments and is described by the stochastic differential equation given in Eq. 2.2 [17, 23, 24]. This model was previously found to accurately describe the forces exerted on the μm-scale particles when suspended in a diatomic nitrogen plasma when accompanied by an accurate frequency of oscillation ($\omega = 1.8\text{s}^{-1}$) and damping coefficient ($\gamma = 0.9$) for the restorative force, so we adhered to it when attempting to more accurately detect sub-pixel features of this kind [33]. Given the independent nature of each dimension of the particle's movement along with the one-dimensional SHO implementation, separate, random-integer seeded time series for each dimension were generated to ensure that the particle is free to roam about the confinement space, as shown in Fig 2.5. We can also plot the simulated features produced at selected time steps to see the influence of origin position in image generation along with the translation due to independent changes in x_0 and y_0. This independence also eases later error analysis due to the effectively identical behavior of the SHO in both x and y, allowing for averaging over both dimensions as with the positions predicted for the randomly-generated image set.

$$\frac{d^2x(t)}{dt^2} = -\gamma\frac{dx(t)}{dt} - \omega^2 x(t) + N_W \tag{2.2}$$

Slightly modifying the initial procedure employed for generating simulated images, we created a large data set based upon a pair of time series with a length of $n = 2500000$ and time steps of $dt = 0.02$ using the Mathematica solver. We then treated the velocity distributions of each series to normalize the standard deviation of the velocity, σ_v, to have a value of 0.3 pixels/s. This limitation prevents the particle from traveling too far in the course of the time series, corralling it into an acceptable range extending approximately one pixel around the origin for generating the simulated images when translating into Python. For this transition, the series are exported as 64-bit floating point binary files, which can be

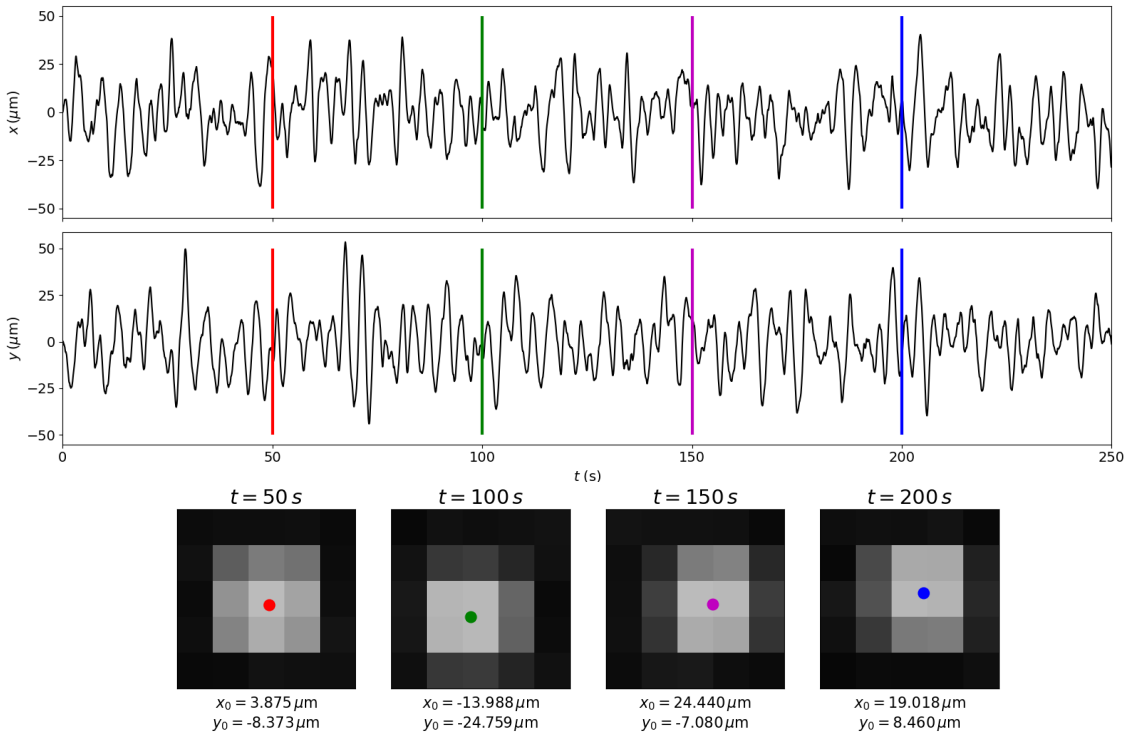

Figure 2.5: Translation of SHO time series positions into simulated images. Units of μm are reached from the approximation of 1 pixel $\approx 50\,\mu$m.

read into Python with NumPy. We can then pass the time series into our modified image creation function instead of producing uniformly-distributed x_0 and y_0 pairs to image the motion of the simulated particle for further study. With these new driven image sets, we can test the performance of TrackPy and the previously-trained machine-learning models in recognizing the dynamics of the particle over time.

Chapter 3

Results and Discussion

3.1 Parameters and Pixel-Locking

One important result of our simulation parameter study is the recognition that σ, the parameter responsible for the particle's sharpness, has an impact on the degree of pixel-locking present in generated images. The differences in sub-pixel distribution, as imaged by TrackPy, between 6 evenly-spaced values of σ chosen from the generation range used are depicted in Fig. 3.1. Note first the absence of values about the edges of each distribution, as seen in the "pillow" shaped feature characterizing the first four plots. This suggests that there is some bias away from the whole integer values of pixels. $\sigma = 0.6$ has greater concentration of predicted features at the $(0, 0)$ origin, while $\sigma = 0.72, 0.84, \text{and } 0.96$ all have similar concentrations towards the corners of the distribution with $x = y = \pm 0.5$. At the two higher values of σ, the same central clustering behavior returns but to a smaller degree. We must also recognize the often nearly-uniform distribution of sub-pixel features within the accessible regions. The "TV static" appearance of swaths of each plot corresponds to some degree of uniformity, though this is not as important as the fact that this uniformity does not extend to cover the entire distribution.

Clearly, the width of the particle generated has a notable effect on the presence of pixel

locking, at least when viewed on a large scale. At larger values of σ, the entirety of the particle's Gaussian peak can be contained within one pixel, which might promote more uniform determination of sub-pixel position with less ability to reference neighboring-pixel context. Conversely, at smaller values of σ, the peak of the Gaussian spreads across multiple pixels, providing a wealth of information with the neighboring-pixel context of 10 or 12 pixels that are differentiable from the image's background noise. Therein, there is something about the peak width with middling values of σ that leads to a pixel-locking bias towards the corners of the central $0.4 \leq (x, y) \leq 0.6$ sub-pixel ranges. Note that we do not see the expected tendency towards integer values, nor to the center of the distribution, which seem to be more likely when examining pixel-locking.

This phenomena warrants further study beyond simply determining and describing the presence of pixel-locking about preferred values due to changes in σ with only TrackPy. Differences in performance across the machine learning models might be expected, as might changes in the distribution when controlling for another parameter that could have some relationship with σ. For example, α, which instead determines the "flatness" of the Gaussian peak, also influences the number of neighbors present about the central bright pixel of a feature, as shown in Fig. 2.2. Sub-pixel distributions of flatter, wider features might then behave differently to those with sharper peaks due to the influence of neighbors, among other factors.

3.2 Machine Learning Performance and Pixel-Locking

To consider the errors between each of the feature detection models and the ground truth positions for x and y from the test set, we used a root mean squared error calculation to see how far from the actual position of the particle each predicted feature was. Again, each model is accurate enough to correctly identify the pixel or array index where the particle is located, but the sub-pixel accuracy is what we are truly concerned with, so we would expect to see

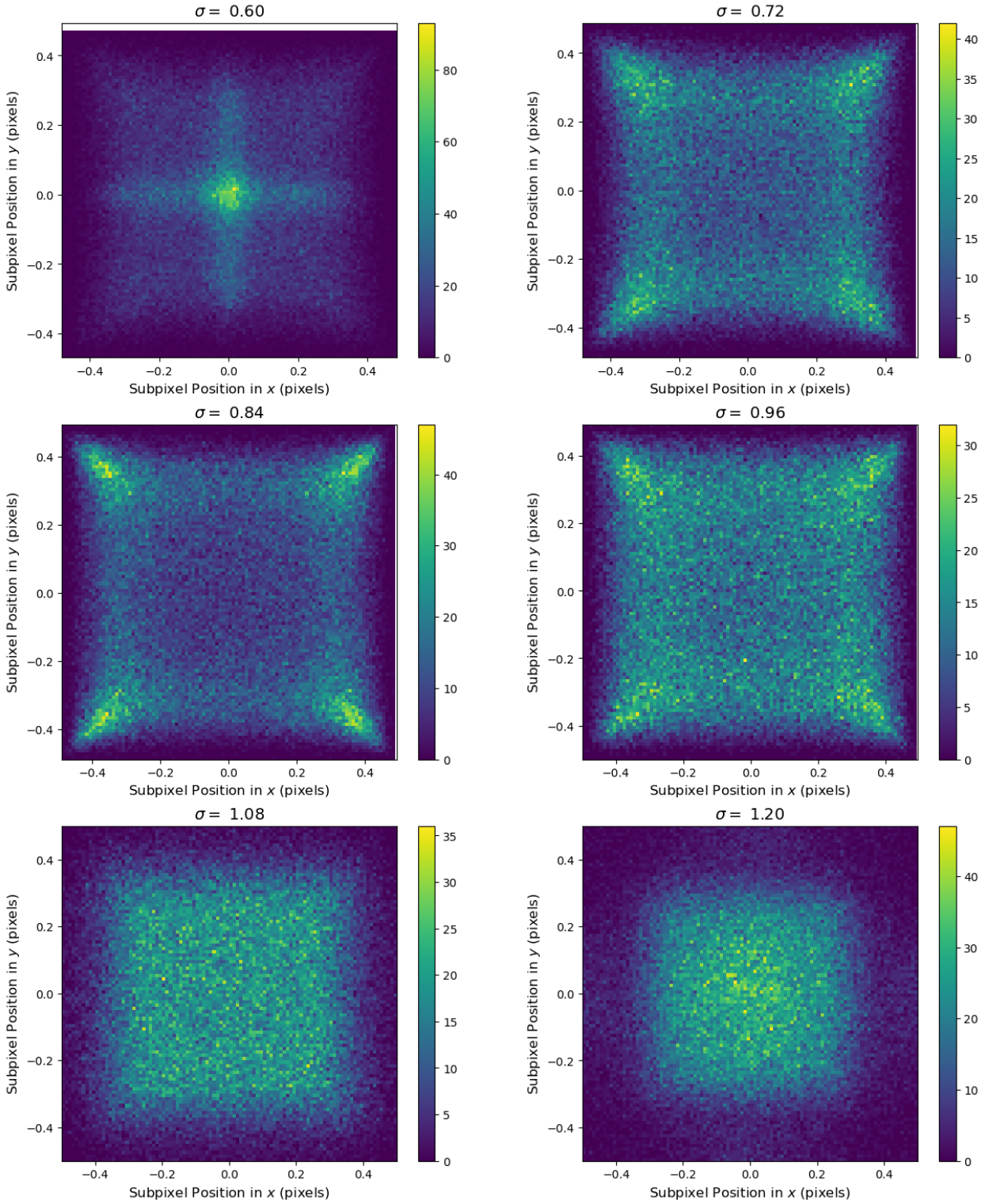

Figure 3.1: Pixel locking due to differences in σ. Distributions were produced from TrackPy feature detection on 6 sets of $n = 100000$ arrays generated from uniformly distributed parameters besides selected σ values as listed.

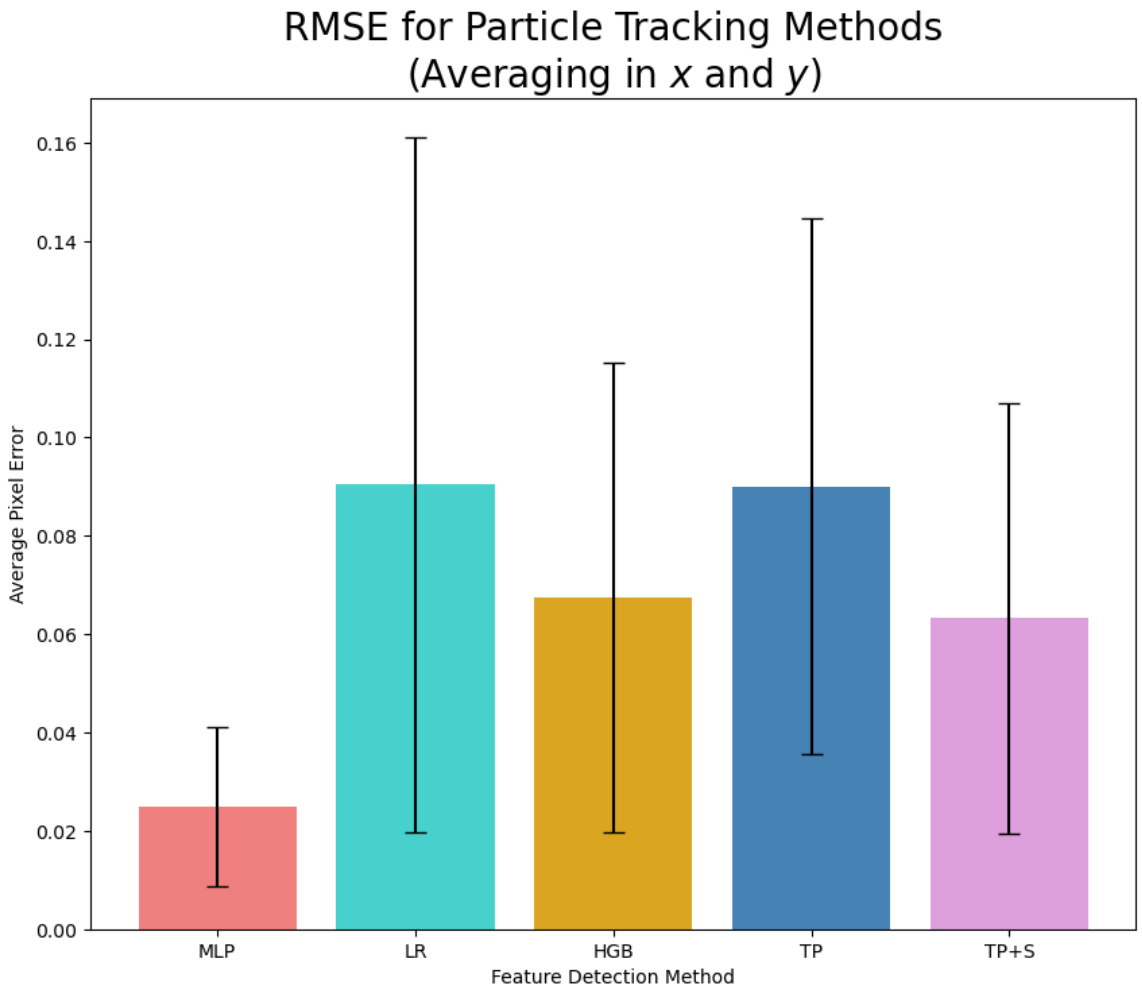

Figure 3.2: Average error comparison between the feature detection models. Whiskers depict $\pm\sigma_{error}$, the standard deviation of the error values for model performance. MLP is Multi-Level Perceptrons, LR is Linear Regression, HGB is Histogram Gradient-Boosting, TP is TrackPy, and TP+S is TrackPy with added SPIFF correction.

error values that are within pixel resolution. To compare the error values, we then plotted each of the averaged values of the calculated error on a bar plot and generated error bars based on the standard deviation about the average value for each model, or σ_{error}, as shown in Fig 3.2. We can see that the average error was lowest for MLP, even outperforming both TrackPy and TrackPy with SPIFF. Of course, linear regression has the highest error value with its σ_{error} value extending all the way to 0.30, which suggests very poor performance on this data set.

The calculated error between the predicted and actual particle positions is unable to tell

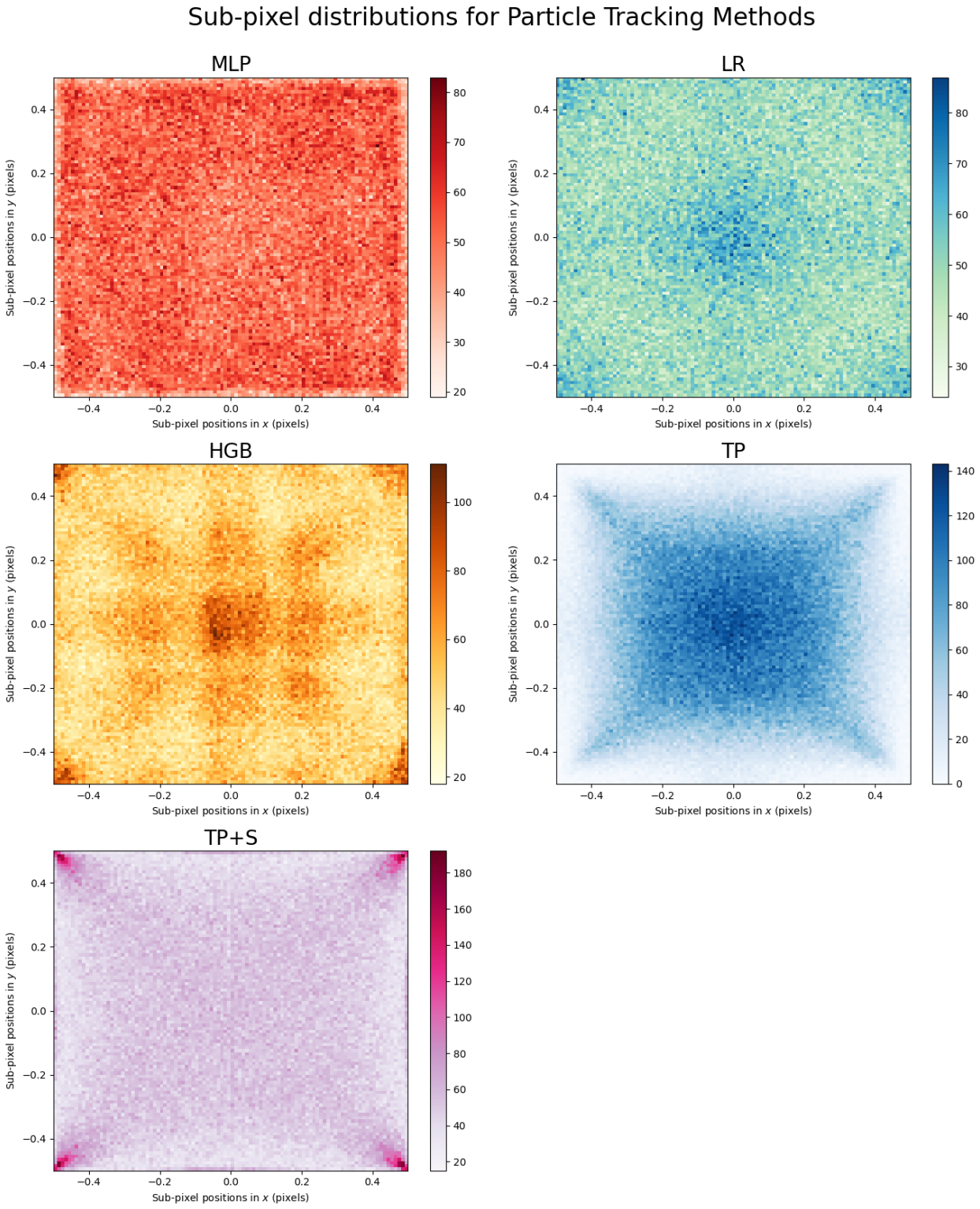

Figure 3.3: Sub-pixel distribution for predicted positions from each feature detection model. Darker colors indicate greater density of features, while lighter colors indicate less density regions. A distribution without pixel-locking would have a similar density across the entirety of the plot, as opposed to biasing towards the center, edge, or corner values.

us the full story about the quality of predictions. This is where the analysis of the sub-pixel distributions of the predicted positions comes in, as we need to also determine the degree to which pixel-locking is present in each method's estimations. The same relationship works in reverse, as if we were to look only at the pixel-locking performance of each model, we might not attain a full understanding of the model performance. While the distribution for linear regression appears to be rather flat outside of the center and corners, if not comparable to HGB and TrackPy, we know this to be less valuable due to its much higher degree of error compared to the rest of the models. Interestingly, while MLP had the best error performance, its sub-pixel distribution appears to have some biasing towards the edges of the plot, though this still handily outperforms both TrackPy and TrackPy with SPIFF. We can clearly see the benefit of applying the SPIFF correction to the predicted positions TrackPy generated, as the sub-pixel distribution of TrackPy with SPIFF set more uniformly distributes positions to fill in the missing spaces of the "pillow" shape about the edges, though this appears to make the degree of pixel-locking in the corners of the distribution more aggressive.

The spread of error values described by σ_{error} is also worth examining, as it seems that the σ_{error} value for HGB is comparable to TrackPy, even though we can see a rather extreme case of pixel-locking on the HGB predictions in its sub-pixel distribution plot. Clearly, it seems difficult for a single feature detection method to produce the trifecta of a low value for the average error, low accompanying σ_{error}, and a uniform distribution without a presence of visually obvious pixel bias. For example, if we were to test MLP on experimental data, we might produce a number of estimated positions which are on average rather close to the actual position of the experimental particles, but this is a rather shallow victory if many of the estimations are still especially affected by pixel-locking.

3.3 Velocity Distributions and Estimated p Fit

To analyze the velocity distributions generated for the prediction of the simulated SHO series and ensure that the models were capable of recognizing the noise to be Gaussian in nature we used a central difference across the data points in the time series (with time steps of $\Delta t = 0.02\,\mathrm{s}$) to differentiate the position of the particle for its velocity. The velocities were then treated with a moving average with a window size of 2000 elements, averaged across both v_x and v_y, binned, and an estimated fit line was produced. This curve fitting optimization was carried out by SciPy, and involved also taking the logarithm of the data to generate a power law relationship between the normalized velocities and the binned frequencies, as described by Eq. 3.1. As expected, the power, p, of this expression fitting the velocity distribution of the SHO particle's exact position from the time series approaches $p = 2.00$, as shown in Fig. 3.5. The results of this same process, as completed on the particle tracking methods, are shown in Fig. 3.6. Ultimately, the tracking model that seemed to most closely align with the fit model was that of MLP, though we can see that there is still some error involved in its predictions with $p = 2.03$, though the fit falls within a standard deviation of $p = 2.00$. Each of the other models display some notable degree of fitting inaccuracy, with the exception of linear regression, which comes rather close to the expected fit. Something interesting to note is the negative impact that applying SPIFF had on the already poor performance of TrackPy, as the TrackPy with SPIFF fit is the worst performer with $p = 2.29$, a large departure from the expected fit.

$$\log(y) = Ae^p + C \tag{3.1}$$

We expect the velocity distribution of the particle, regardless of feature detection treatment, to follow that of a Gaussian corresponding to a fit line of $p = 2$, as described in previous research from our lab and shown in Fig. 3.4 [33]. Deviation from this Gaussian characterization must come from inaccuracies in feature detection when the movement of

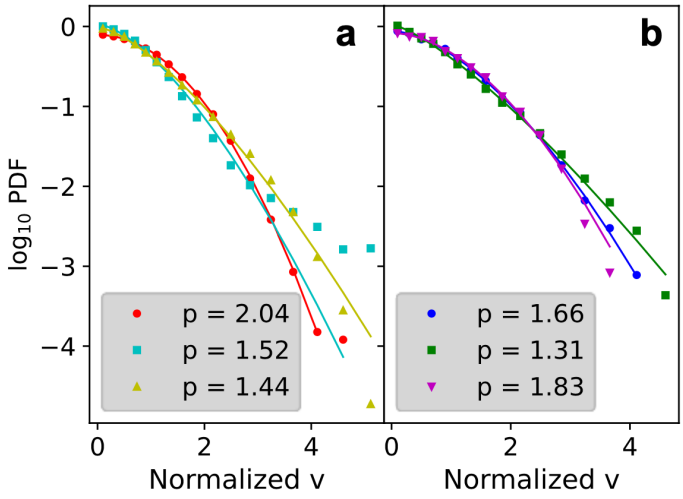

Figure 3.4: Previously found velocity distribution performance. (a) shows a comparison of simulated data with the presence of Gaussian, non-Gaussian, and additional pixel-locking noise, corresponding to the red, blue, and yellow fit lines respectively. (b) shows another comparison of experimental data trajectories with the same fitting procedure. Note that the degree of pixel-locking present in these figures is worse than what we find to be the case for each feature detection model on the simulated data set.

the particle is driven solely by the Wiener Process. In this way, fitting the velocity distributions derived from the detected positions of the SHO particle over time serves as a way to quantify the degree of pixel-locking present with each model. We cannot use a typical sub-pixel distribution as above to depict the presence of pixel-locking because the SHO is a system oscillating about equilibrium with turning points roughly a pixel away. The particle will then spend a great deal of its existence about and passing through the equilibrium, and even more accelerating and decelerating about the turning points.

It is important to note the influence of bin count and size on the curve fitting carried out by SciPy, as particularly large bin counts can still provide a fit of $p = 2.0$, though the data points therein will be widely scattered about the more central fit line. To that end, the decision to lock the bin count for each curve fit at 100 was implemented, followed by changing to a more accurately distributed but lower bin count which more closely aligned with the greater frequency of smaller velocities present. This change allowed for a more

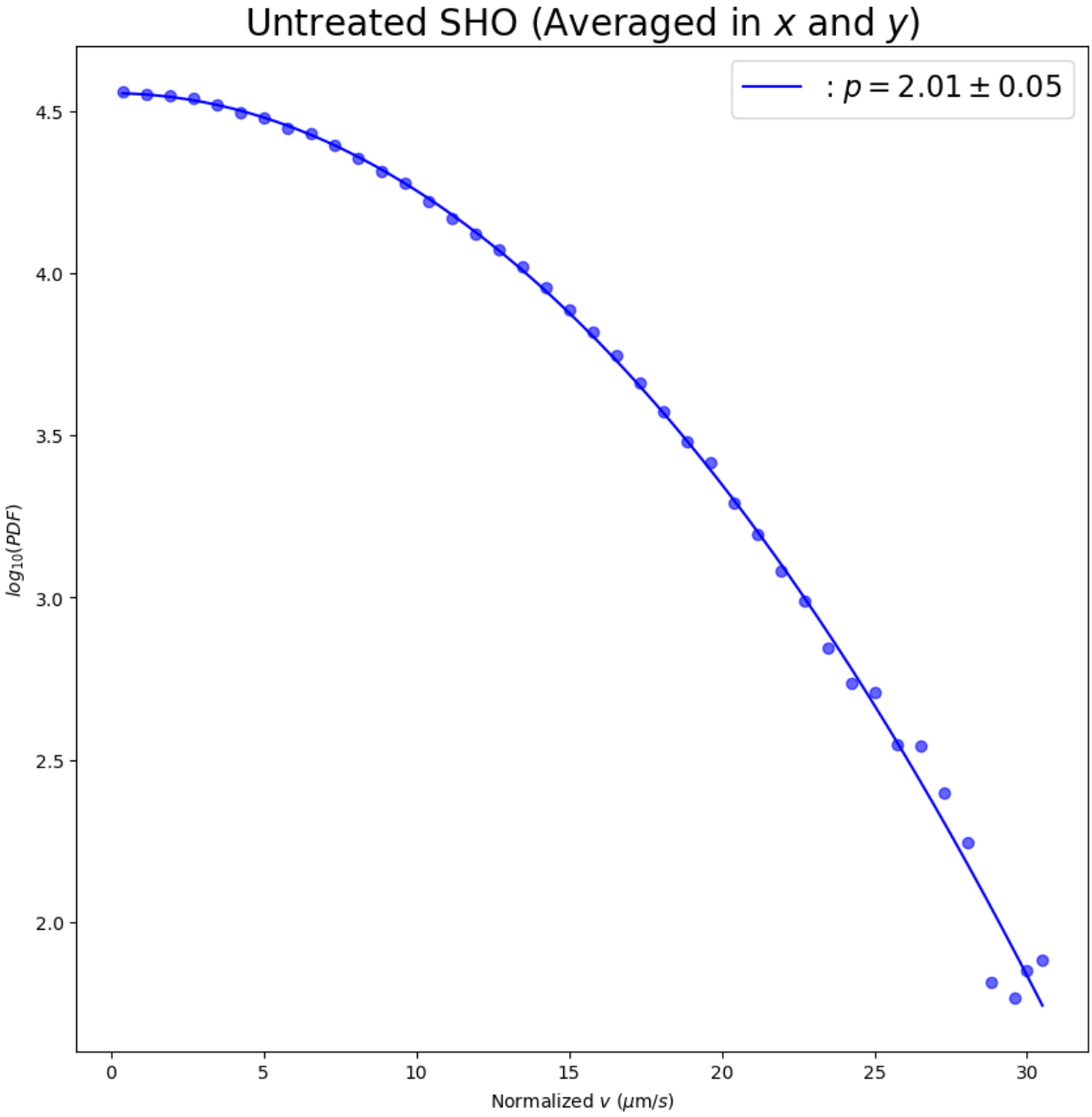

Figure 3.5: Result of curve fitting procedure on the SHO time series. Velocity values have been normalized and averaged across x and y.

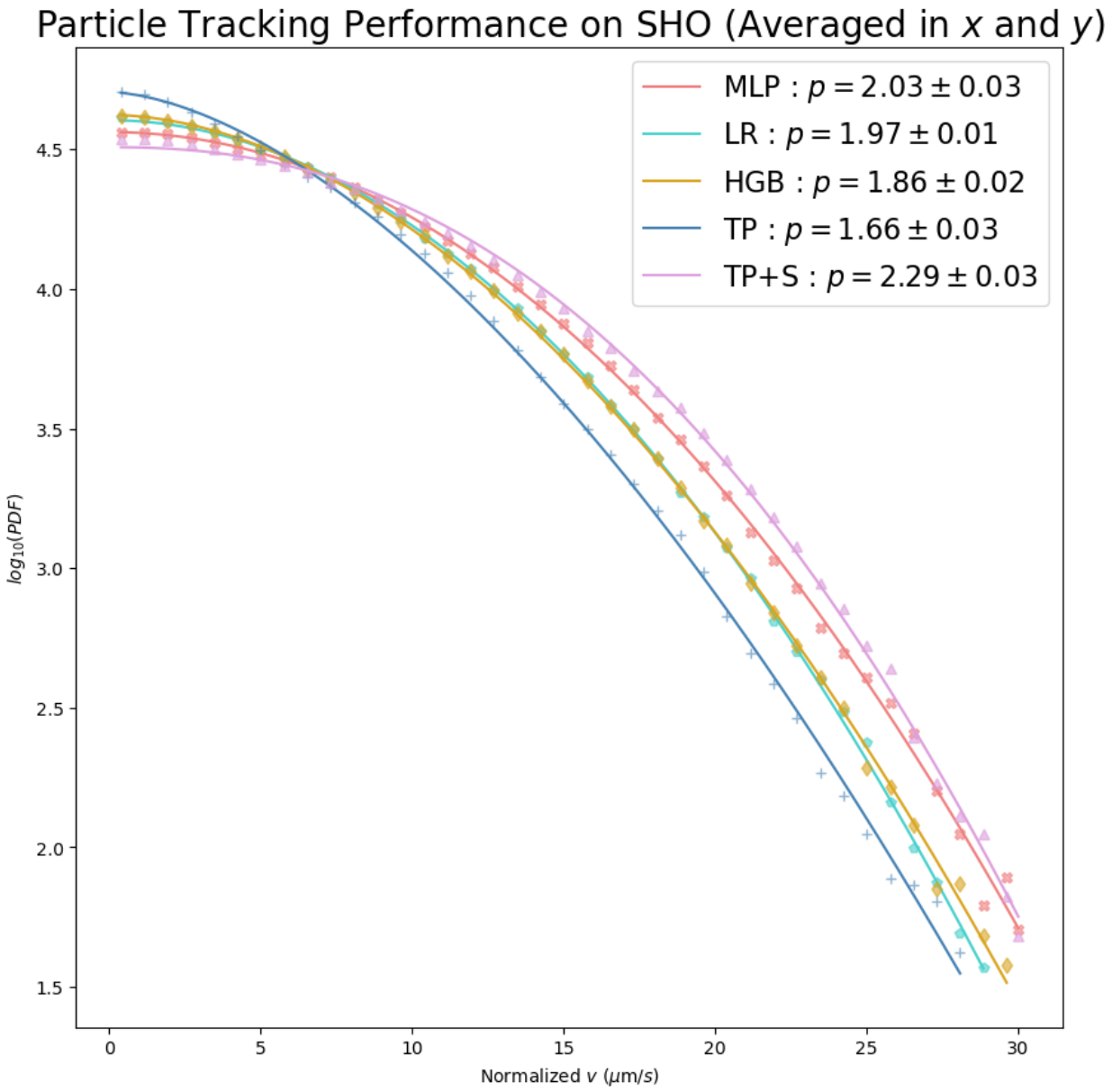

Figure 3.6: Result of curve fitting procedure on the SHO time series with each particle tracking method. Velocity values have been normalized and averaged across x and y.

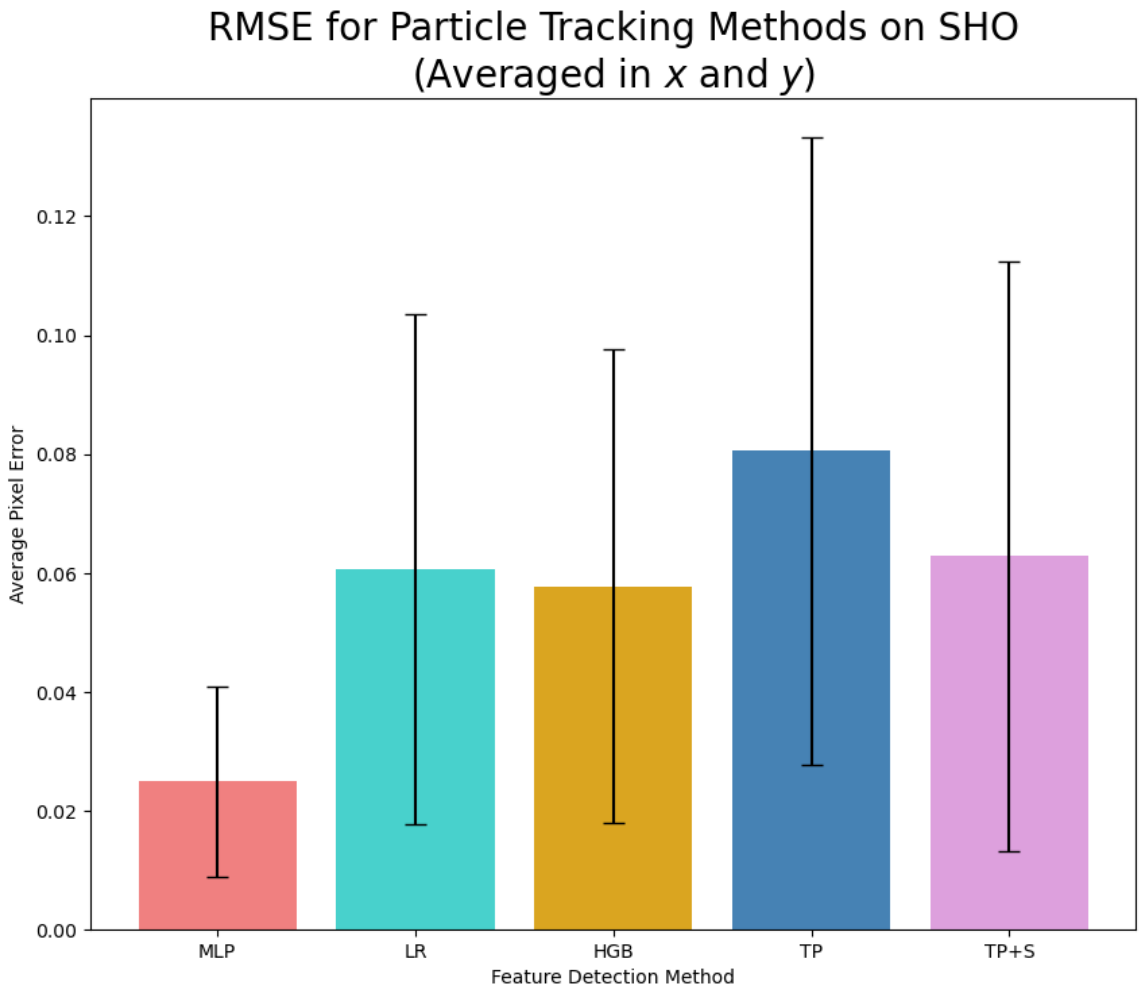

Figure 3.7: Average error comparison between the feature detection models' performance on SHO particle.

accurate comparison of deviations from the stochastic noise to be detected across the models. However, because the deviation from the expected fit line cannot communicate the whole story of the models' performance on the SHO particle set, we need to also carry out another error analysis comparison, as shown in Fig. 3.7.

Analyzing the result of this comparison, we find that the lowest average error is again produced by the MLP model, which outperforms both the other machine-learning models and the TrackPy methods, with and without the SPIFF correction. Note also that SPIFF appears to do even less to reduce the average error in prediction for TrackPy here, as the average error for the TrackPy with SPIFF method is still above that of MLP, and its σ_{error} is

still greater than HGB and LR with their smaller average error values. This could be due to the fact that SPIFF attempts to correct for non-uniform subpixel distributions, but again, the subpixel distribution of the SHO time series is biased due to the nature of the system itself more than pixel-locking. The standard deviation from this average error value is also worth discussing, as we see that σ_{error} smallest for MLP, but it is still almost as large as the value of the average error in detection. This would suggest a large spread in the error values produced by MLP that largely overlaps with the spread for the rest of the methods on the lower end, while still capturing much lower values on the higher end of the average with σ_{error}. This suggests that MLP ends up performing more accurately than TrackPy under the conditions of this time series test, while the velocity distribution for these predicted positions was more accurate than TrackPy.

Chapter 4

Conclusion

Our research finds that the performance of relatively simple machine learning models trained on randomized simulated data in detecting feature positions of simulated particles is comparable to if not better than the performance of existing, widely used tracking methodologies. Note that this is just beginning to scratch the surface of machine learning implementation possible in this field, especially as the capability of machine learning processes continues to increase at lightning speed. More extensive testing of both the parameters that influence pixel-locking and the performance of machine learning models on dynamic systems like the simple harmonic oscillator could show a greater advantage over the older methods. Of course, the ultimate test will be the application of these techniques towards the study of actual experimental data, though this would likely require greater training set diversity to reflect further variations in particle characteristics. With said validation, greater accuracy in biological studies of cells and cellular processes, the dynamics created by expansion microscopy, or the movement of dusty plasma particles could be achieved and influence further scientific discoveries for years to come.

4.1 Future Research

As far as future directions in this research are concerned, the promotion of the machine learning implementation to use more advanced customizable models through Pytorch or TensorFlow stands as a probable next step, along with the introduction of additional particle variation or even imaging multiple simulated particles at once. More powerful libraries for machine learning in Python give access to convolutional neural network models, not to mention the ability to pass image intensities into the models during the training stage as a 2D array instead of a vector. Providing expanded spatial context for the task instead of making the models develop that connection on their own could yield greater model accuracy and remove possible dimensional differences of the aforementioned "wraparound problem." On the other hand, a theoretically more accurate data set extends to include the possibility of cylindrical particles with independent values for the width of the particle in each dimension by including both σ_x and σ_y in the Gaussian function. In this instance, we would also need to modify our model to allow for particle rotation, as different orientations influence the shape of the intensity pattern created in images.

Bibliography

[1] Sagar Adatrao, Michele Bertone, and Andrea Sciacchitano. Multi-Δt approach for peak-locking error correction and uncertainty quantification in PIV. *Measurement Science and Technology*, 32(5):054003, May 2021. doi: 10.1088/1361-6501/abdcde.

[2] R. J. Adrian. Particle-imaging techniques for experimental fluid-mechanics. *Annual Review of Fluid Mechanics*, 23:261–304, January 1991. doi: 10.1146/annurev.fl.23.010191.001401.

[3] Daniel B. Allan, Thomas Caswell, Nathan C. Keim, Casper M. van der Wel, and Ruben W. Verweij. soft-matter/trackpy: v0.6.1, February 2023. URL `https://doi.org/10.5281/zenodo.7670439`.

[4] E. Alpaydin. *Introduction to Machine Learning, second edition.* Adaptive Computation and Machine Learning series. MIT Press, 2009. ISBN 9780262303262. URL `https://books.google.com/books?id=TtrxCwAAQBAJ`.

[5] Stanislav Burov, Patrick Figliozzi, Binhua Lin, Stuart A. Rice, Norbert F. Scherer, and Aaron R. Dinner. Single-pixel interior filling function approach for detecting and correcting errors in particle tracking. *Proceedings of the National Academy of Science*, 114(2):221–226, January 2017. doi: 10.1073/pnas.1619104114.

[6] Marco A. Catipovic, Paul M. Tyler, Josef G. Trapani, and Ashley R. Carter. Improving the quantification of brownian motion. *American Journal of Physics*, 81(7):485–491, 2013. doi: 10.1119/1.4803529. URL `https://doi.org/10.1119/1.4803529`.

[7] Mirela T. Cazzolato, Agma J.M. Traina, and Klemens Böhm. Establishing trajectories of moving objects without identities: The intricacies of cell tracking and a solution. *Information Systems*, 105:101955, 2022. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2021.101955. URL `https://www.sciencedirect.com/science/article/pii/S0306437921001502`.

[8] Subhash Challa, Mark R. Morelande, Darko Mušicki, and Robin J. Evans. *Introduction to object tracking*, page 1–21. Cambridge University Press, 2011. doi: 10.1017/CBO9780511975837.002.

[9] Nicolas Chenouard, Ihor Smal, Fabrice Chaumont, Martin Maška, Ivo Sbalzarini, Yuanhao Gong, Janick Cardinale, Craig Carthel, Stefano Coraluppi, Mark Winter, Andrew Cohen, William Godinez, Karl Rohr, Yannis Kalaidzidis, Liang Liang, James Duncan, Hongying Shen, Yingke Xu, Klas Magnusson, and Erik Meijering. Objective comparison of particle tracking methods. *Nature methods*, 11, 01 2014. doi: 10.1038/nmeth.2808.

[10] K. T. Christensen. The influence of peak-locking errors on turbulence statistics computed from PIV ensembles. *Experiments in Fluids*, 36(3):484–497, January 2004. doi: 10.1007/s00348-003-0754-2.

[11] John C. Crocker and David G. Grier. Methods of Digital Video Microscopy for Colloidal Studies. *Journal of Colloid and Interface Science*, 179(1):298–310, April 1996. doi: 10.1006/jcis.1996.0217.

[12] Y. Feng, J. Goree, and Bin Liu. Accurate particle position measurement from images. *Review of Scientific Instruments*, 78(5):053704, may 2007. doi: 10.1063/1.2735920. URL `https://doi.org/10.1063%2F1.2735920`.

[13] Federico S. Gnesotto, Grzegorz Gradziuk, Pierre Ronceray, and Chase P. Broedersz. Learning the non-equilibrium dynamics of brownian movies. *Nature Communications*,

11(1), oct 2020. doi: 10.1038/s41467-020-18796-9. URL https://doi.org/10.1038%2Fs41467-020-18796-9.

[14] Guram Gogia and Justin C. Burton. Emergent bistability and switching in a nonequilibrium crystal. *Phys. Rev. Lett.*, 119:178004, Oct 2017. doi: 10.1103/PhysRevLett.119.178004. URL https://link.aps.org/doi/10.1103/PhysRevLett.119.178004.

[15] R. J. Hearst and B. Ganapathisubramani. Quantification and adjustment of pixel-locking in particle image velocimetry. *Experiments in Fluids*, 56:191, October 2015. doi: 10.1007/s00348-015-2062-z.

[16] Bhanu Jena, Domenico Gatti, S. Arslanturk, Sebastian Pernal, and Douglas Taatjes. Human skeletal muscle cell atlas: Unraveling cellular secrets utilizing 'muscle-on-a-chip', differential expansion microscopy, mass spectrometry, nanothermometry and machine learning. *Micron*, 117, 11 2018. doi: 10.1016/j.micron.2018.11.002.

[17] M. Mandrysz and B. Dybiec. Energetics of the Undamped Stochastic Harmonic Oscillator. *Acta Physica Polonica B*, 49(5):871, January 2018. doi: 10.5506/APhysPolB.49.871.

[18] Joshua Méndez Harper, Guram Gogia, Brady Wu, Zachary Laseter, and Justin C. Burton. Origin of large-amplitude oscillations of dust particles in a plasma sheath. *Phys. Rev. Res.*, 2:033500, Sep 2020. doi: 10.1103/PhysRevResearch.2.033500. URL https://link.aps.org/doi/10.1103/PhysRevResearch.2.033500.

[19] Carl Nelson, Patrick Duckney, Tim Hawkins, Michael Deeks, Philippe Laissue, Patrick Hussey, and Boguslaw Obara. Blobs and curves: Object-based colocalisation for plant cells. *Functional Plant Biology*, 42:471, 05 2015. doi: 10.1071/FP14047.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Marius Pfeifer, Jessica Agarwal, and Matthias Schröter. On the trail of a comet's tail: A particle tracking algorithm for comet 67P/Churyumov-Gerasimenko. , 659:A171, March 2022. doi: 10.1051/0004-6361/202141953.

[22] M.Y. Pustylnik, A.A. Pikalev, A.V. Zobnin, I.L. Semenov, H.M. Thomas, and O.F. Petrov. Physical aspects of dust–plasma interactions. *Contributions to Plasma Physics*, 61(10):e202100126, 2021. doi: https://doi.org/10.1002/ctpp.202100126. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/ctpp.202100126. e202100126 ctpp.202100126.

[23] Wolfram Research. WienerProcess. https://reference.wolfram.com/language/ref/WienerProcess.html, 2012. Accessed: 18-March-2023.

[24] Wolfram Research. ItoProcess. https://reference.wolfram.com/language/ref/ItoProcess.html, 2016. Accessed: 19-March-2023.

[25] Melissa Rinaldin, Ruben W. Verweij, Indrani Chakraborty, and Daniela J. Kraft. Colloid supported lipid bilayers for self-assembly. *Soft Matter*, 15:1345–1360, 2019. doi: 10.1039/C8SM01661E. URL http://dx.doi.org/10.1039/C8SM01661E.

[26] Katie A. Rose, Mehdi Molaei, Michael J. Boyle, Daeyeon Lee, John C. Crocker, and Russell J. Composto. Particle tracking of nanoparticles in soft matter. *Journal of Applied Physics*, 127(19):191101, 2020. doi: 10.1063/5.0003322. URL https://doi.org/10.1063/5.0003322.

[27] A. Sauret, F. Boulogne, J. Cappello, E. Dressaire, and H. A. Stone. Damping of liquid sloshing by foams. *Physics of Fluids*, 27(2):022103, feb 2015. doi: 10.1063/1.4907048. URL https://doi.org/10.1063%2F1.4907048.

[28] I.F. Sbalzarini and P. Koumoutsakos. Feature point tracking and trajectory analysis for video imaging in cell biology. *Journal of Structural Biology*, 151(2):182–195, 2005.

ISSN 1047-8477. doi: https://doi.org/10.1016/j.jsb.2005.06.002. URL https://www.sciencedirect.com/science/article/pii/S1047847705001267.

[29] Hao Shen, Lawrence J. Tauzin, Rashad Baiyasi, Wenxiao Wang, Nicholas Moringo, Bo Shuang, and Christy F. Landes. Single particle tracking: From theory to biophysical applications. *Chemical Reviews*, 117(11):7331–7376, 2017. doi: 10.1021/acs.chemrev.6b00815. URL https://doi.org/10.1021/acs.chemrev.6b00815. PMID: 28520419.

[30] Casper van der Wel and Daniela J. Kraft. Automated tracking of colloidal clusters with sub-pixel accuracy and precision. *Journal of Physics: Condensed Matter*, 29, 2016.

[31] Denis Wirtz. Particle-tracking microrheology of living cells: Principles and applications. *Annual Review of Biophysics*, 38(1):301–326, 2009. doi: 10.1146/annurev.biophys.050708.133724. URL https://doi.org/10.1146/annurev.biophys.050708.133724. PMID: 19416071.

[32] Yuval Yifat, Nishant Sule, Yihan Lin, and Norbert F. Scherer. Analysis and correction of errors in nanoscale particle tracking using the single-pixel interior filling function (spiff) algorithm. *Scientific Reports*, 7, 2017.

[33] Wentao Yu, Jonathan Cho, and Justin C. Burton. Extracting forces from noisy dynamics in dusty plasmas. *Phys. Rev. E*, 106:035303, Sep 2022. doi: 10.1103/PhysRevE.106.035303. URL https://link.aps.org/doi/10.1103/PhysRevE.106.035303.