



Fachbereich 3 - Mathematik und Informatik

## Bachelor-Thesis

# 2D-Partikelerkennung aus niedrig aufgelösten Videos

*Lebelwa Danmou Job Jimmy*

19. Oktober 2022

**Erster Gutachter:** Prof. Dr. Sebastian Maneth

**Zweite Gutachterin:** Dr. Hui Shi

**Betreuer:** Prof. Dr. Sebastian Maneth

# **Eidesstattliche Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 19. Oktober 2022

---

Job Jimmy, Lebelwa Danmou

# **1 Abstract**

The needs in the fields of biology, microbiology and medicine are in constant evolution. The motivation of our bachelor thesis will be to contribute to this edifice by designing a tool capable or allowing not only to detect particles present in a video, but also to provide an optimal graphical interface for the evaluation of results. This work could therefore serve as a foundation stone for many others in this same field.

## **2 Einleitung**

Diese Arbeit befasst sich mit der Erkennung und Bewertung von Partikeln auf der Grundlage von Videodaten. Dazu werden die zu untersuchenden Videos in Sequenzen von Bildern segmentiert, durch die die Erkennung und dann die Auswertung Bild für Bild erfolgt. Dies geschieht unter Verwendung von Bibliotheken und/oder Frameworks, die für diesen Zweck konzipiert wurden.

### **2.1 Motivation**

Jenseits eines besonderen Interesses an Biologie, Mikroskopie und Medizin im Allgemeinen gibt es auf medizinischer oder biologischer Seite einen stetig wachsenden Bedarf an immer leistungsfähigeren organischen Zellanalysen. Dies geschieht natürlich mit dem Ziel, die Funktionsweise (Reaktion) von Mikrozellen besser zu verstehen und dadurch die Qualität der Behandlung zu verbessern und/oder Prognosen zu erstellen.

In diesem Sinne haben wir uns überlegt, ein Werkzeug zu entwickeln, das als Sprungbrett für weitere Arbeiten genutzt werden kann, um so einen bescheidenen Beitrag zur Entwicklung der oben genannten Bereiche zu leisten.

### **2.2 Ausgangssituation**

Diese Arbeit soll die erste in einer Reihe von zukünftigen Arbeiten sein. Deshalb ist der Ausgangspunkt hier nichts anderes als das Material, das zur Verfügung gestellt wird, um die Daten daraus zu extrahieren, nämlich die Videos.

Da es also keinen Vorgänger gibt, werden wir uns zunächst auf eine Art Vergleichsstudie verschiedener Tools konzentrieren, die uns helfen können, unsere Ziele zu erreichen (*siehe Kapitel Bibliothek/Software<sup>4</sup>*). Anschließend werden wir die Ergebnisse dieser Studie verwenden, um mit dem eigentlichen Zweck der Arbeit fortzufahren, wie er in der 2.3 beschrieben ist.

## **2.3 Zielsetzung**

Ziel dieser Bachelorarbeit ist es, wie bereits erwähnt, ein Werkzeug (Software) zu entwerfen, mit dem man Partikel in einem bestimmten Video aufspüren kann. Dabei wird die Anzahl der gefundenen Partikel in einem vorgegebenen Bereich gehalten. Anschließend soll eine grafische Benutzeroberfläche entwickelt werden, die eine schnelle Auswertung der Ergebnisse ermöglicht.

## **2.4 Abgrenzung**

In Anbetracht des Rahmens unserer Arbeit ist es notwendig, an dieser Stelle zu erwähnen, dass die Erkennung und Bewertung von Partikeln auf der Grundlage von Videos eine Reihe von Faktoren nicht einschließt. Dies bezieht sich unter anderem auf die Interpretation der Ergebnisse. Dabei handelt es sich sowohl um die Position der Partikel als auch um deren Identifizierung.

Diese Arbeit wird daher zunächst nur dazu dienen, die Partikel Bild für Bild zu erkennen. Anschließend werden die Bilder mit den erkannten Partikeln mit Hilfe einer eigens dafür entwickelten Schnittstelle visualisiert.

Schließlich können wir Parameteränderungen auf ein oder mehrere spezifische Bilder anwenden, ohne die bereits validierten Bilder zu verändern.



# Inhaltsverzeichnis

<b>1 Abstract</b>	<b>2</b>
<b>2 Einleitung</b>	<b>3</b>
2.1 Motivation . . . . .	3
2.2 Ausgangssituation . . . . .	3
2.3 Zielsetzung . . . . .	4
2.4 Abgrenzung . . . . .	4
<b>3 Grundlage</b>	<b>8</b>
3.1 Definition . . . . .	8
3.2 Bild- und Videoqualität . . . . .	9
3.2.1 Bild-/Imagequalität . . . . .	9
3.2.2 Videoqualität . . . . .	12
3.3 Verfahren . . . . .	12
3.3.1 Erkennung Methoden . . . . .	12
3.3.2 Der Fall Trackpy . . . . .	15
<b>4 Bibliotheken/Software</b>	<b>16</b>
4.1 ParticleTracker . . . . .	17
4.2 STracking . . . . .	22
4.3 SPT: Single particle tracking analysis . . . . .	24
4.4 Trackpy . . . . .	26
<b>5 Trackpy</b>	<b>30</b>
5.1 Einleitung und Installation . . . . .	30
5.2 Benutzererfahrung . . . . .	30
5.2.1 In Bezug auf das Lesen des Videos . . . . .	30
5.2.2 In Bezug auf die Einarbeitung in die Bibliothek . . . . .	31
5.2.3 In Bezug auf die Geschwindigkeit . . . . .	31
5.2.4 In Bezug auf Verwendung bestimmter Parameter der Funktion locate. . . . .	32
<b>6 Partikel-Erkennungssystem</b>	<b>33</b>
6.1 Partikel-Erkennungssystem . . . . .	33
6.1.1 Parameter der locate-Funktion . . . . .	33

6.1.2	Ermittlung der optimalen Parameterwerte der Locate-Funktion von Trackpy. . . . .	36
6.2	Automatische Parameter-Optimierung . . . . .	52
6.3	Bearbeitung eines einzelnen Bildes . . . . .	59
<b>7</b>	<b>Auswertung</b>	<b>62</b>
7.1	App . . . . .	62
7.1.1	Installation . . . . .	62
7.1.2	Bedienung der Anwendung . . . . .	63
<b>8</b>	<b>Fazit</b>	<b>66</b>
<b>9</b>	<b>Ausblick</b>	<b>67</b>
<b>Abbildungsverzeichnis</b>		<b>68</b>
<b>Literaturverzeichnis</b>		<b>69</b>
<b>Anhang</b>		<b>72</b>

# 3 Grundlage

## 3.1 Definition

In der Absicht, bestimmte Probleme zu vermeiden, die sich aus der semantischen Verwirrung bestimmter Wörter oder Ausdrücke ergeben und die später während des Lesens dieser Arbeit auftreten könnten. Wir machen uns hier die Mühe, diese Wörter oder Ausdrücke, die immer wieder verwendet werden, in diesem Kontext zu definieren.

- **Partikel/Teilchen**

Laut Pons Wörterbuch, ein Partikel ist “ein sehr kleines Teilchen von etwas”. Nun im Rahmen unserer Arbeit ist eine Partikel, jedes bewegliche oder unbewegliche Objekt oder Element, das auf einem Bild zu sehen ist.

- **Erkennung/Lokalisierung**

Unter Erkennung wird hier gemeint, der Prozess, der es ermöglichen soll, jegliche auf ein Bild existierende Partikel zu detektieren/finden und ggf. zu umzingeln. Bei der Erkennung liegt der Fokus auf das Finden von Partikeln auf einem einzigen Bild.

- **Verfolgung/Tracking**

Bei der Verfolgung handelt es sich um einen Prozess, bei dem die vorhandenen Partikel gefunden werden. Dies geschieht jedoch nicht auf einem einzelnen Bild wie bei der Erkennung, sondern auf einer Sequenz von Bildern. So kann man den Verlauf (Historie) eines Partikels von Bild 1 bis Bild  $n$  erhalten.  $n$  ist das letzte Bild der Bildsequenz.

- **Trajektorie/Verbindung/Verlinkung lokalisierter Partikel**

Hierbei handelt es sich um die Verbindungsgeraden zwischen den Punkten, die die Position der Partikel im Laufe der Zeit (der Bilder) in der Bildsequenz darstellen. So kann man für ein Partikel in jedem Bild unterschiedliche Koordinaten ( $x,y$ ) haben, vorausgesetzt, das Partikel ist beweglich.

## 3.2 Bild- und Videoqualität

Dieser Teil der Arbeit widmet sich der Klärung von Begriffen, die mit dem Bereich der Bildverarbeitung zusammenhängen (Auflösung, Pixel, Dichte, ...), so dass der Unterschied zwischen Bildern guter und schlechter Qualität gemacht werden kann.

Es ist wichtig, einige Begriffe zu verstehen, um über die Qualität von Bildern oder Videos sprechen zu können.

- **Bild/Image**

Ein Bild [14], genauer gesagt ein digitales Bild, bezeichnet eine Matrix oder ein Array von quadratischen Pixeln, die in Zeilen und Spalten geordnet sind.

- **Pixel**

Ein Pixel [30] wird als kleinste Einheit (quadratisch) eines digitalen Bildes oder einer Grafik bezeichnet, die auf einem digitalen Bildschirm angezeigt werden kann.

- **Video**

Ein Video [20] ist eine Sequenz/Abfolge von Bildern (so genannte Frames), die in einer bestimmten Frequenz aufgenommen und eventuell angezeigt wird.

### 3.2.1 Bild-/Imagequalität

Im Allgemeinen bezieht sich die Qualität eines Bildes/Videos auf seine Auflösung. Das bedeutet, dass wir uns hier mit der Auflösung befassen, und noch besser, was diese ausmacht. Wie bereits erwähnt, ist ein digitales Bild eine Darstellung von Pixeln, die in Höhe und Breite (Spalten und Zeilen) angeordnet sind. Die Anzahl der Pixel, aus denen ein Bild besteht, sowie die Bittiefe dieser Pixel und die Art und Weise, wie sie verteilt sind, machen also einen Großteil der Auflösung eines Bildes aus. Wir können also nicht nur über die Anzahl und Bittiefe der Pixel sprechen, sondern auch über ihre Dichte im Bild.

#### Anzahl der Pixel

Die Anzahl der Pixel kann einfach durch Multiplikation der Anzahl der Pixel in der Höhe mit der Anzahl der Pixel in der Breite bestimmt werden. Beispiel: Ein Bild mit (Höhe \* Breite)  $300 * 400 = 1200000\text{Pixel}$  oder  $0,12\text{Megapixel}$ .

### **Bilddichte**

Die Bilddichte wird jedoch in Pixel pro Zoll *dpi* bzw.*ppi* (pixel per inch) ausgedrückt. Das heißt, die Anzahl der Pixel, die pro Zoll angezeigt werden. Die Auflösung hängt also auch davon ab. Je niedriger der Wert von *dpi* ist, desto größer ist das Bild. Daher wird bei einem Bild mit wenigen Pixeln die Qualität bzw. Auflösung verringert, wie im folgenden Beispiel gezeigt wird.

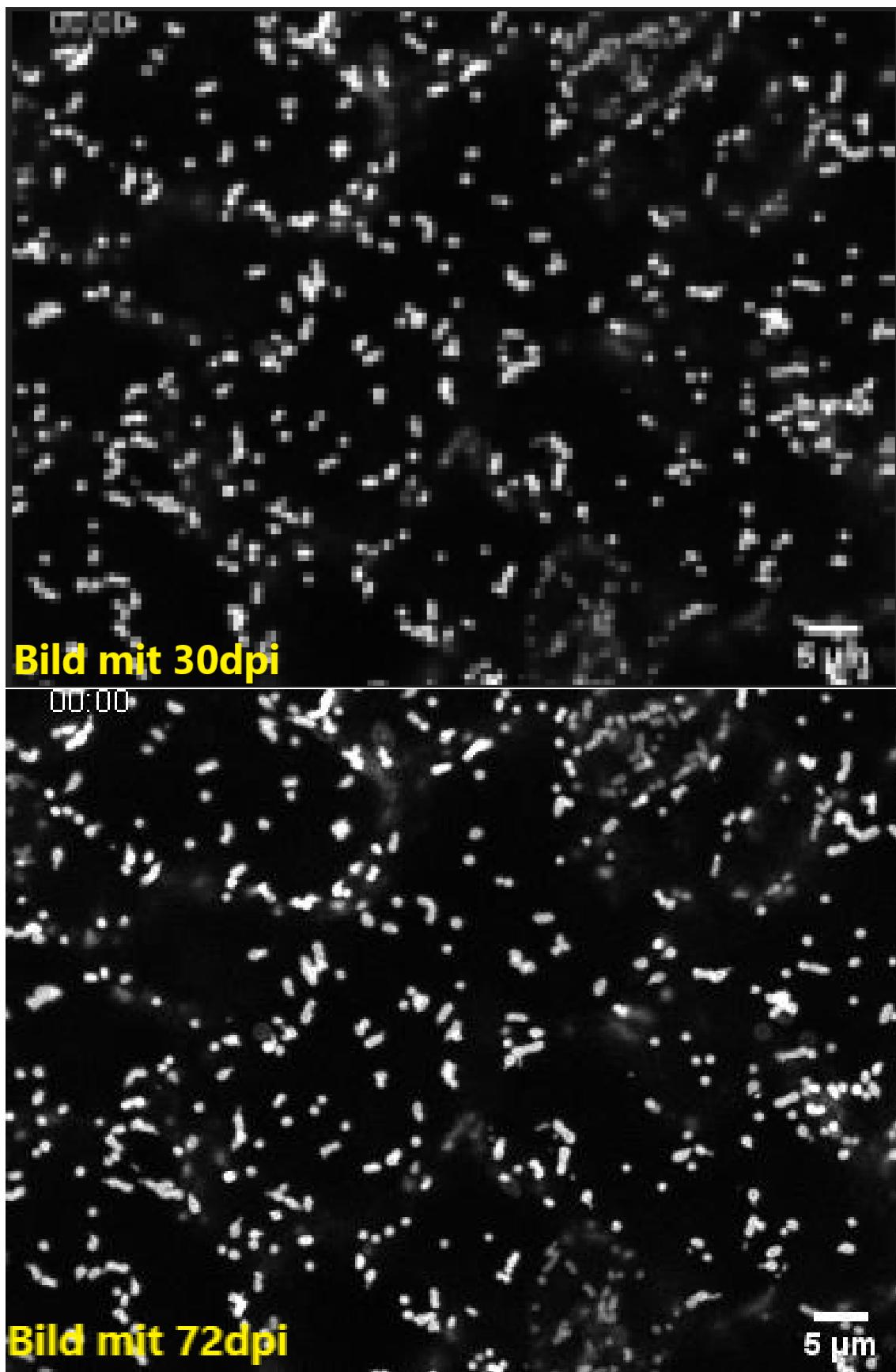


Abbildung 3.1: Vergleich zwischen Bild mit 72dpi und eins mit 30dpi.

### Bittiefe

Die Bittiefe bezieht sich auf die Gesamtanzahl der Töne (Anzahl der Farben), die das Bild haben muss. Die Gesamtzahl der Bittiefe wird durch gegeben:

$$2^n = \text{Bittiefe} \quad (3.1)$$

Dabei steht n für die Anzahl der Bits. Je höher dieser Wert ist, desto mehr Farben werden gespeichert und angezeigt. Beispiel: Ein Bild mit einer Bittiefe von 1 Bit:  $2^1 = 2$  kann nur zwei Farben speichern, nämlich Schwarz und Weiß.

### 3.2.2 Videoqualität

Ein Video ist letztlich nur eine Folge von Bildern, die mit einer bestimmten Frequenz, nämlich Frames pro Sekunde (*fps*), gerendert oder aufgenommen werden. Je höher die Anzahl der *fps*, desto flüssiger das Video. Der Punkt über die Auflösung von Bildern 3.2.1 sollte daher noch einmal gelesen werden, um das Konzept der Auflösung besser zu verstehen und auf ein Video anzuwenden.

## 3.3 Verfahren

Zur Erkennung von Partikeln gibt es eine Vielzahl von Ansätzen. Zu diesen Ansätzen oder Methoden gehören: "Maxima detection", "thresholding", "fitting", "centroid estimation", wie in "Objective comparison of particle tracking methods" erwähnt.

### 3.3.1 Erkennung Methoden

#### • Maxima detection

Die "Maxima detection", auch "Peak detection" oder "lokal maxima" genannt, ist ein Konzept, das es ermöglicht, innerhalb jeder Zeile eines Akkumulators oder Arrays in einem bestimmten Intervall das Element zu finden (im Allgemeinen ist die Position des Elements im Array die Mitte des gewählten Intervalls), das den höchsten Wert vor und nach ihm in diesem Intervall hat. Die Werte in diesen Arrays sind in der Regel die Helligkeitsintensität jedes Punktes im Bild.

Beispiel: Das Array  $a = [74, 4, 5, 71, 70, 8, 9, 67]$  und ein Intervall von 5. Das zu findende Element ist auf Position 3 (Mitte des Arrays)

74	4	5	71	70	8	9	67
----	---	---	----	----	---	---	----

Im ersten Intervall (gefondene Maxima = 0).

74	4	5	71	70	8	9	67
----	---	---	----	----	---	---	----

Im zweiten Intervall (gefondene Maxima = 71).

74	4	5	71	70	8	9	67
----	---	---	----	----	---	---	----

Im dritten Intervall (gefondene Maxima = 0).

74	4	5	71	70	8	9	67
----	---	---	----	----	---	---	----

Im vierten Intervall (gefondene Maxima = 0).

Abbildung 3.2: Beispiel von ‘maxima detection’

Dies ermöglicht es, Schritt für Schritt die Bildpunkte mit der höchsten Helligkeit zu ermitteln, bei denen es sich wahrscheinlich um Partikel oder zu erkennende Elemente handelt. "VK Yadav et al." beschreiben in ihrem Artikel Approach to accurate circle detection: "Circular Hough Transform and Local Maxima concept" (Ansatz zur genauen Kreiserkennung: "Circular Hough Transform und lokales Maxima-Konzept") [32] eine explizitere Beschreibung.

- **Thresholding**

Beim Thresholding wird jeder Pixelwert (Pixelintensität) in einem Bild mit einem bestimmten Schwellenwert (threshold) verglichen. Dadurch werden alle Pixel des Eingabebildes in zwei Gruppen unterteilt: Die Pixelgruppe mit In-

tensitätswerten unterhalb eines Schwellenwerts und eine mit Intensitätswerten über einem Schwellenwert.

Auf diese Weise ist es möglich, anhand der erhaltenen Pixelgruppen zu bestimmen, welche Bildelemente am ehesten mit den gesuchten Partikeln übereinstimmen und gegebenenfalls den Schwellenwert anzupassen.

Die Anwendungsmöglichkeiten werden in "A Multi-Thresholding Algorithm for Sizing out of Focus Particles" [16] und in "A review of thresholding strategies applied to human chromosome segmentation" [21] besprochen und implementiert.

- **Fitting**

Das Fitting ist eine Funktion, die in der Regel im Bereich des maschinellen Lernens eingesetzt wird. Die Funktion trainiert ein Modell<sup>1</sup>, um Vorhersagen zu treffen (hier Vorhersagen über die Position/Lokalisierung von Partikeln). Dazu verwendet sie Trainingsdaten, die vorab zur Verfügung gestellt werden. Wenn das Modell einmal trainiert ist, wird es die Parameterwerte finden, die am besten geeignet sind, um möglichst viele Partikel aufzuspüren.

Sollten die Vorhersagen und die Ergebnisse zu stark voneinander abweichen, sollte das Modell idealerweise erneut mit der Fitting-Methode trainiert werden.

In dem Artikel "Efficient particle swarm optimization approach for data fitting with free knot B-splines" [11] von *Y Niu et al.* sowie in "A function fitting method" [10] von *Rajesh Dachiraju* wird die Funktionstüchtigkeit des Fittings herausgearbeitet und eingehend erläutert.

- **Centroid estimation**

Ein Zentroid ist ein Punkt, der dem geometrischen Mittelpunkt eines Objekts entspricht. Je nach Form des Objekts können eine oder mehrere Koordinaten erforderlich sein, um diesen Punkt zu definieren.

Das Konzept der Zentroidenschätzung ermöglicht es also, den geometrischen Mittelpunkt eines jeden vorhandenen Partikels zu ermitteln.

Genau zu diesem Zweck haben *Tsukamoto, Marcio Michiharu et al.* in dem Artikel "*Fluid interface detection technique based on neighborhood particles centroid deviation (NP-CD) for particle methods*" [31] die Effektivität der Anwendung dieser Methode im Vergleich zu anderen, weiter verbreiteten und verwendeten Methoden demonstriert.

Es ist wichtig zu erwähnen, dass die meisten Partikelerkennungstools nicht nur eine dieser Methoden verwenden, sondern vielmehr eine Kombination aus mehreren, wie

---

<sup>1</sup>Ein Modell ist eine wohldefinierte Berechnung, die das Ergebnis eines Algorithmus ist, der einen Wert oder eine Reihe von Werten als Eingabe nimmt und einen Wert oder eine Reihe von Werten als Ausgabe erzeugt.

in "Objective comparison of particle tracking methods" [7] gezeigt wird.

### 3.3.2 Der Fall Trackpy

Um Partikel in einem Bild zu lokalisieren/zu erkennen, muss die Trackpy-Bibliothek drei Schritte durchführen. Der erste Schritt besteht in der Korrektur von Bildfehlern, der zweite in der tatsächlichen Lokalisierung/Erkennung und der letzte in der Verfeinerung der Lokalisierung.

- **Die Bildrekonstruktion:**

Die Mängel von digitalisierten Bildern sind in der Regel auf folgende Ursachen zurückzuführen:

- Die geometrische Verzerrung, die durch das Lichtmikroskop verursacht wird.
- Die Rauschen des Bildes
- Ein ungleichmäßiger Kontrast

In dieser Phase werden diese Bildfehler verringert oder sogar korrigiert.

- **Lokalisierung der Partikel:**

Trackpy verwendet hauptsächlich die Methode der "*Maxima-Erkennung*", um Partikel zu finden, die für die Erkennung in Frage kommen. So wird ein Pixel als Kandidat angenommen, wenn kein anderes Pixel in einer Entfernung von  $w$  heller ist.

Es ist auch wichtig zu erwähnen, dass Trackpy auch die Möglichkeit bietet, Partikel durch Schwellenwertbildung (*Thresholding*) zu lokalisieren. Die Verwendung dieser Funktion ist jedoch optional, wenn der Benutzer eine zusätzliche Genauigkeit zu den bereits vorhandenen hinzufügen möchte.

- **Verfeinerung der Lokalisierung:**

Dient dazu, die Zentroide der Partikel innerhalb eines halben Pixels zu lokalisieren.

Die detailliertere Funktionsweise der ausgeführten Schritte zur Erreichung der Lokalisierung wird in dem Artikel von Crocker, John C und Grier, David G [9] hervorgehoben.

## 4 Bibliotheken/Software

In diesem Abschnitt werden einige Möglichkeiten zur Erkennung von Partikeln in einem Video niedriger Auflösungsqualität verglichen. Es wäre natürlich ideal einen Vergleich zwischen manuellen und werkzeugbasierten Erkennung zu ziehen. Aber leider wird die manuelle Erkennung hier nicht vollzogen aufgrund der viel zu schwierig bzw. langwierige Arbeit, die sie bereitet. Es geht hier nämlich um mehrere hunderte von Partikeln pro Bild.

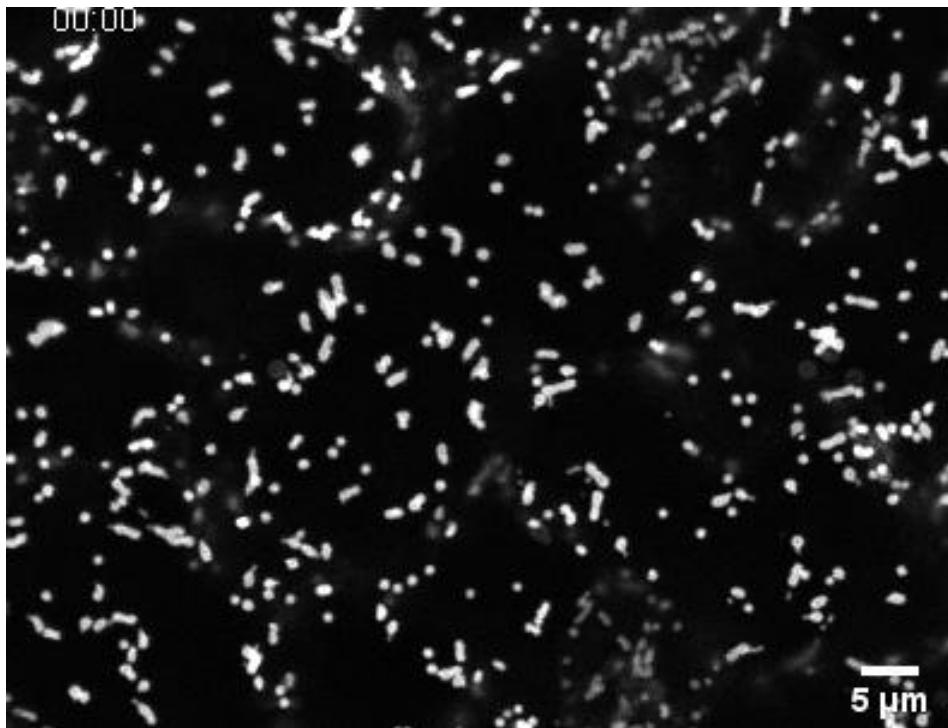


Abbildung 4.1: Rohbild mit zu erkennenden Partikeln.

In diesem Sinne widmet sich die Arbeit in der Folgezeit dem Vergleich möglicher Tools, die zur Erreichung der eigentlichen Ziele benutzt werden können. Hierbei werden aus der Vielzahl der Tools nur vier herausgegriffen. Diese werden dann

einer textuellen Analyse der Unterschiede unter ihnen unterzogen.

Es ist jedoch unerlässlich, hier zum besseren Verständnis der Arbeit zu erwähnen, dass die meisten der im Folgenden betrachteten Bibliotheken Bibliotheken sind, die es ermöglichen, Partikel durch ein Video hindurch zu verfolgen. Da das Verfolgen von Partikeln ein Schritt nach der Erkennung ist, haben wir uns dafür entschieden, mit Bibliotheken zu arbeiten, die das Verfolgen von Partikeln ermöglichen. Wir haben uns auf die Verfolgung von Partikeln konzentriert, da es gewusst ist, dass die Verfolgung erst nach der Erkennung erfolgt. Genauer gesagt, weil das Tracking das Produkt einer Reihe von Erkennungsvorgängen im Verlauf eines Videos ist. Auf diese Weise möchten wir Missverständnisse ausräumen, die durch die Verwendung des Begriffs ‘**Tracking/Verfolgung**’ und nicht ‘**Erkennung**’ entstehen könnten.

## 4.1 ParticleTracker

**ParticleTracker** ist eine vollständig grafisch bedienbare Software, die es erlaubt, Partikel aus einem Video guter oder niedriger Auflösung zu verfolgen. Diese verwendet mehrere verschiedene Verfolgungsalgorithmen mit einer Standardschnittstelle, um die Einrichtung verschiedener Partikelverfolgungsprojekte schnell und einfach zu gestalten [26]. Es verwendet drei verschiedene Methoden/Funktionen zur Erkennung von Partikeln in einem Video. Alle diese Methoden haben ihre eigenen Besonderheiten bei der Verwendung, nämlich:

- *Opencv Hough Circles:*

Es ist eine Methode, die von OpenCV[6] entwickelt und zur Verfügung gestellt wurde. Diese Option wurde hauptsächlich für die Erkennung von Objekten entwickelt, die kreisförmig aussehen oder eine kreisförmige Form haben. Dies ist in unserem Fall nicht sehr interessant, da die in unserem Video gezeigten Partikel verschiedene Formen haben können (kreisförmig, quadratisch, längs und ...). Allerdings verwendet sie nicht die klassische **Hough Transform**<sup>1</sup>, die in “THE STANDARD HT” [33] und ziemlich zeitaufwendig und speicher-kostend ist, da sie die Suche nach drei Unbekannten/Parametern erfordert, wie die mathematische Darstellung des Kreises unten zeigt:

$$(x - x_{center})^2 + (y - y_{center})^2 = r^2$$

Wobei  $(x_{center}, y_{center})$  ist der Mittelpunkt des Kreises.

---

<sup>1</sup>Es ist eine von vielen Methoden, um einen Kreis in einem Bild zu erkennen.

$r$  ist der Radius des Kreises.

Stattdessen wird der **Hough Gradient** (im Wesentlichen beschrieben in ‘21HT’ [33]) verwendet, der die Informationen des Gradienten<sup>2</sup> berücksichtigt.

- *Trackpy:*

Diese Methode ist nach der Bibliothek benannt, die sie entwickelt und zur Verfügung gestellt hat, nämlich: trackpy. Trackpy ist eine Bibliothek zur Verfolgung von Partikeln in 2D, 3D und höheren Dimensionen. Weitere Informationen zu dieser Bibliothek liefern wir in einem gesonderten Abschnitt dieser Arbeit.

- *Opencv Contour finding:*

Ist auch, wie der Name schon sagt, eine OpenCV-Methode mit dem Ziel, Konturen<sup>3</sup> in einem Binärbild<sup>4</sup> zu finden. Die Funktion ruft die Konturen des Binärbildes ab, indem sie den Algorithmus [29] verwendet. Konturen sind ein nützliches Werkzeug für die Musteranalyse und das Erkennen und Erfassen von Objekten.

## Ein-/Ausgabe

Als Eingabe akzeptiert ParticleTracker Videos in fast allen gängigen Formaten wie avi, mp4, MOV etc... .

Jedoch gibt es bis zu drei Möglichkeiten für die Ausgabe. Diese sind wie folgt:

- *DataFrame*<sup>5</sup>: Dieser Datenrahmen enthält die Werte aller im Video vorhandenen Partikel im Laufe der Zeit. Er enthält in der Reihenfolge Frame (Frame-Nummer), x(Position auf der x-Achse), y(Position auf der y-Achse), mass(Helligkeit), size, ecc, raw\_mass, ep, particle, user\_rad

---

<sup>2</sup> “The gradient is the generalization of the derivative to multivariate functions. It captures the local slope of the function, allowing to predict the effect of taking a small step from a point in any direction.” — Page 21, Algorithms for Optimization, 2019 [18].

<sup>3</sup> Eine Kontur kann einfach als eine Kurve interpretiert werden, die alle aufeinanderfolgenden Punkte (entlang des Grenzbereichs) mit derselben Farbe oder Intensität verbindet. Konturen sind nützliche Werkzeuge zur Analyse von Formen und zum Erkennen und Identifizieren von Objekten.

<sup>4</sup> Ein binäres Bild ist ein Bild, dessen Pixel nur zwei mögliche Intensitätswerte haben. Numerisch sind diese beiden Werte meist 0 für Schwarz, 1 oder 255 für Weiß.

<sup>5</sup> Ein DataFrame ist eine Datenstruktur in Python, die Daten in einer 2-dimensionalen Tabelle mit Zeilen und Spalten organisiert, ähnlich wie eine Tabellenkalkulation.

- *CSV*: Liefert ebenfalls die gleichen Ergebnisse wie beim DataFrame, allerdings im csv-Format
- *Video*: Hier handelt es sich um ein neues Video, das aus den Bildern des Videos erstellt wurde, das für die Entdeckungen verwendet wurde. Nur hier bleiben alle erkannten Partikel, die eingekreist und/oder nummeriert wurden, im gesamten (neuen) Video so eingekreist/nummeriert.

## Vorteile

### 1. GUI bedienbar:

Die Benutzeroberfläche ist eine angenehme Arbeitsmethode für Benutzer, insbesondere für Neulinge (Siehe [13]). Dies ist sicherlich eine hervorragende Möglichkeit, in diesen Bereich einzusteigen, ohne sich mit zu vielen Codezeilen auseinandersetzen zu müssen oder Programmierkenntnisse zu besitzen.

### 2. Anfängerfreundlich:

Die Anfängerfreundlichkeit ist stark auf den vorherigen Punkt zurückzuführen, nämlich die GUI-Bedienbarkeit. Zudem ist es sonderlich umstandslos, mit Hilfe einiger Klicks, sich Informationen(x(Position auf der x-Achse), y(Position auf der y-Achse), mass(Helligkeit), size, ...) über jedes Partikel in jedem Bild generieren zu lassen, die relevant sind.

### 3. Datenvisualisierung:

Ein großer Vorteil ist Datenvisualisierung. Dies liegt an der schrittweisen und automatischen Aktualisierung des Renderings(Anzeige der Ergebnisse der Partikelerkennung auf dem Bild.) in Abhängigkeit von den vorgenommenen Einstellungen.

### 4. Datenausgabe:

Der Zugang zu den Daten in der Software ist schnell und einfach. Außerdem sind die Daten in verschiedenen Formen verfügbar. Unter anderem werden die Daten als csv-Datei und auch als Video bereitgestellt, das den Fortschritt der erkannten Objekte/Partikel im Ausgangsvideo nachzeichnet. Es ist auch wichtig zu erwähnen, dass es möglich ist, die Daten (in csv) in Bezug auf ein einzelnes Bild zu exportieren.

## Nachteile

### 1. Das Hochfahren der Software:

Trotz der Dokumentation war es nicht einfach, die Software zum Laufen zu bringen. In Anbetracht der Verwendung von Bibliotheken, die nur in einer bestimmten Python-Umgebung verfügbar sind, nämlich ‘Conda’<sup>6</sup>. Daher wäre es fair zu erwähnen, dass dies für Laien noch schwieriger sein könnte.

## 2. Funktionsbegrenzt:

Die Benutzung über eine GUI ist zwar ein begeisternder Punkt. Doch genau hier liegt die Schwäche. Da die Benutzeroberfläche bereits konfiguriert ist, bietet sie nur die Möglichkeit, die verfügbaren Optionen zu nutzen. Dies könnte in manchen Fällen zu einem Mangel an Optionen führen und somit unzureichend sein. Es Beispiel hier wäre: Falls man den Wert der verwendeten Parameter jedes Mal ändern möchte, wenn die Anzahl der erkannten Partikel unter einer vordefinierten Anzahl oder sogar unter einem Prozentsatz liegt.

## 3. Nicht intuitiv:

Obwohl die Verwendung einer grafischen Oberfläche im Allgemeinen für den Benutzer attraktiver ist, sollte sie so intuitiv wie möglich sein oder gut dokumentiert werden (zumindest als Tooltip). In diesem Fall sind die Namen einiger Optionen auf der Schnittstelle nicht sehr aussagekräftig oder beschreibend(siehe 4.2).



Abbildung 4.2: Unzureichend beschriebene Optionsnamen

Es ist daher verständlich, dass man Hinweise erwartet, die die Verwendung der Optionen (Schaltflächen) im Detail beschreiben. Dies ist leider nicht der Fall. Daher erfordert die Benutzung der Software ein gewisses Hin und Her in der externen Dokumentation, um die Rolle der Optionen zu lernen.

<sup>6</sup>Conda ist ein Open-Source-Paketverwaltungssystem und Umgebungsverwaltungssystem, das auf unter Windows, macOS und Linux läuft ausgeführt wird. Es ermöglicht die schnelle Installation, Ausführung und Aktualisierung von Paketen und deren Abhängigkeiten.

## Beispiel

Wir zeigen hier ein Beispiel für die Erkennung von Partikeln auf einem bestimmten Bild.

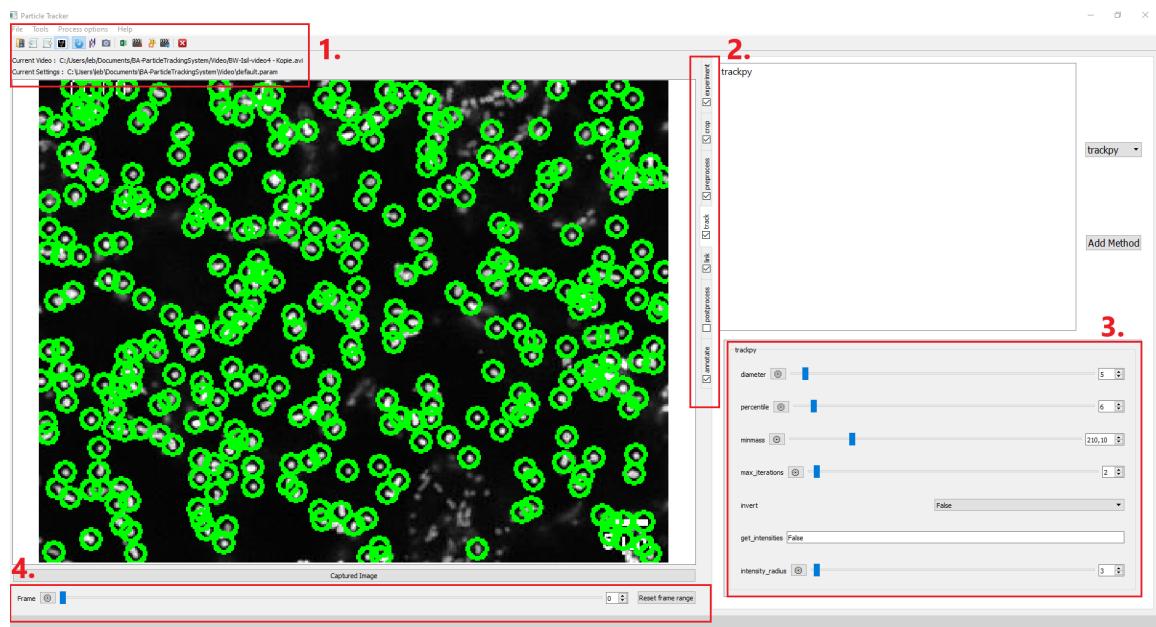


Abbildung 4.3: Beispiel einer Partikelerkennung mit der Trackpy-Methode von ParticleTracker.

1.	Hier geht es um alles, was mit der Eingabe und Ausgabe von Daten zu tun hat.
2.	Es geht hier um die Optionen, die angewendet/aktiviert werden müssen, um im Rendering-Bereich (Bereich direkt unter 1.) angezeigt zu werden.
3.	Hier können die Werte der Parameter für die Lokalisierung geändert werden.
4.	Es zeigt das Bild, auf dem man sich gerade befindet.

Auf diesem Bild sehen wir eine Menge kleiner grüner Kreise, die kleine weiße Partikel umschließen. Es ist zu erkennen, dass die meisten dieser weißen Partikel eingekreist sind. Das bedeutet in diesem Fall, dass sie entdeckt wurden. Daher scheint die Erkennung hier ziemlich gut zu sein. Obwohl sie einige Partikel redundant zu erfassen scheint.

## 4.2 STracking

**STracking** ist ein Framework, dessen Ziel es ist, eine Pipeline<sup>7</sup> für die Partikelverfolgung zu entwickeln. Diese Pipeline besteht aus folgenden voneinander unabhängigen Schritten:

- Frame-by-Frame Partikelerkennung
- Partikelverknüpfung
- Analyse von Partikeleigenschaften
- Entwurf von Track-Features
- Filterung von Tracks

Es stellt drei Methoden zur Verfügung, um die Partikelerkennung und -verfolgung durchzuführen, die jeweils auf spezifischen Algorithmen basieren, nach denen sie benannt sind. Wir haben namentlich:

- **Difference of Gaussians (DoG)**: ist ein Algorithmus zur Verbesserung von Graustufenbildern, bei dem eine unscharfe Version eines Originalgraustufenbildes von einer anderen, weniger unscharfen Version des Originals subtrahiert wird [15].

- **Determinant of Hessian (DoH)**: Der als solche funktioniert, wie in [25] dargestellt,

*'An image is scaled to a size defined by the scale parameter  $\sigma$ . Let  $H_\sigma$  denote the Hessian matrix at a specific image location in level  $\sigma$  and e.g.  $\partial_{xx} = \frac{\partial^2 I}{\partial x^2}$  denoting the second order derivative of the image  $I$  along the  $x$ -axis. We can use the normalized determinant response of the Hessian'*

$$\sigma^4 \cdot \det(H_\sigma) = \sigma^4 \cdot (\partial_{xx} \cdot \partial_{yy} \cdot \partial_{xy}^2)$$

*'to detect image features (blobs and notches) by searching for maxima in each image location across scale.'*

- **Laplacian of Gaussians (LoG)**: Er hebt Regionen mit schnellen Intensitätsänderungen hervor und wird daher häufig zur Kantenerkennung verwendet.

---

<sup>7</sup>Ein Verfahren, das es ermöglicht, die verschiedenen Schritte der Verarbeitung eines Befehls durch den Prozessor voneinander unabhängig zu machen.

det [19].

Obwohl es für jede Anwendung zur Verfolgung von Punkten in 2D+t- und 3D+t-Bildern verwendet werden kann, ist seine Hauptidee die Verfolgung intrazellulärer Objekte (alle Objekte/Partikel, die innerhalb einer Zelle sind) in Mikroskopiebildern. Sowohl in 2D+t (2-Dimension nämlich x-position und y-position) als auch in 3D+t (3-Dimension nämlich x-position, y-position und z-position) [22]. Wobei t für Zeit steht.

In Kombination mit dem Plugin **napari**[23] bietet es die Möglichkeit, den Tracking-Prozess über eine grafische Benutzeroberfläche zu verwalten.

### Ein-/Ausgabe

STracking verwendet als Eingabedaten Bilder oder Sequenzen von Bildern im TIFF-Format<sup>8</sup> (Abkürzung für Tag Image File Format). Daneben ist es auch möglich, den Inhalt von csv-, xml- und StIO-Dateien zu lesen.

In Bezug auf die Ausgabe bietet es die Möglichkeit, die Ergebnisse in verschiedene Formate zu exportieren, nämlich:

- *csv*: Unterstützt keine Aufspaltungs- (Ereignis, bei dem ein Partikel sich in mehrere splittet) und Fusionseigniss (Ereignis, bei dem mehrere Partikeln zusammenführen),
- *StIO*: Dieses Format wurde von STracking entwickelt, um die Daten leichter speichern zu können.

### Vorteile

#### 1. Leicht installierbar:

Mit den in der Dokumentation beschriebenen Schritten ist es relativ einfach, die Bibliothek zu installieren. Die Dokumentation erwähnt Wege, wie die Installation durchgeführt werden könnte, nämlich über *PyPI* und/oder über die *Das Herunterladen und Installieren des Quellcodes über git*.

#### 2. GUI bedienbar:

Wie erwähnt ist es möglich, das Programm über eine grafische Benutzeroberfläche zu verwenden. Dies geschieht nach der Installation des napari-Plugins.

---

<sup>8</sup>ist eine Computerdatei, die zum Speichern von Matrixgrafiken und Bildinformationen verwendet wird

Hier müssen die ersten Schritte jedoch per Code gemacht werden. Das Plugin kann also sowohl zur Manipulation als auch zur Visualisierung der Ergebnisse verwendet werden.

## Nachteile

### 1. Lückenhafte Dokumentation:

In der Tat ist die Dokumentation in Bezug auf die Installation der Bibliothek nicht vollständig. Zum Beispiel wird nirgends erwähnt, dass **pyQt**<sup>9</sup> installiert werden muss. Obwohl dieses **Modul(pyQt)** zwingend erforderlich ist, wenn man die grafische Benutzeroberfläche verwenden möchte.

### 2. Unerwartete Reaktion der grafischen Benutzeroberfläche:

Im Gegensatz zu dem, was in der Dokumentation unter dem Punkt ‘Example 1: Stracking workflow’ beschrieben ist, schließt sich das Fenster, das die visuelle Darstellung der Tracking-Aktionen anzeigen soll, automatisch, sobald es geöffnet wird. Dies ermöglicht natürlich weder das Betrachten noch das Manipulieren der grafischen Darstellung. Es ist wichtig zu erwähnen, dass PyQt in der gleichen virtuellen Umgebung installiert werden muss, um das oben erwähnte Problem zu beheben.

## 4.3 SPT: Single particle tracking analysis

**SPT** [28] ist ein Python-Paket, das einen vollständigen Arbeitsablauf für die Analyse der Verfolgung einzelner Partikel bietet. Dabei stützt es sich im Wesentlichen auf andere Pakete wie: **picasso\_addon** [24] und **picasso** [17] für die Lokalisierung/Erkennung von Partikeln aus Video und die Analyse von immobilisierten<sup>10</sup> Partikeln und **trackpy** [4] für die Verknüpfung von Lokalisierungen oder die Erstellung von Trajektorien.

Die Lokalisierung, die hier die des *Picasso*-Pakets verwendet, konzentriert sich auf einen *gradientenbasierten Ansatz* [27]. Der Gradient oder bzw. die Gradientenkarte wird üblicherweise in der Bildverarbeitung verwendet, um die Kanten von Objekten in einem Bild hervorzuheben.

Dabei haben homogene Bereiche eines Bildes meist einen niedrigen Gradienten, der niedriger ist als der Gradient der Objektkanten in demselben Bild.

---

<sup>9</sup>PyQt ist eine Bibliothek, die die Verwendung des Qt-GUI-Frameworks in Python ermöglicht.

<sup>10</sup>Das sind Partikel, die sich während des gesamten Videos überhaupt nicht bewegen.

## Ein-/Ausgabe

SPT erlaubt es nur eine einzige Eingabemöglichkeiten, nämlich Video-Datei. Dabei werden nur zwei Arten akzeptiert: TIFF und binäre Rohdaten (Dateierweiterung „.raw“).

Bei der Datenausgabe bietet SPT eine Reihe von Möglichkeiten, was den Export von Daten angeht:

- *csv*
- *hdf5*
- *txt*
- *xyz*
- *3d*
- *roi*

## Vorteile

### 1. Zielgerichtet:

Wie in seiner Beschreibung angekündigt, stellt er alle Werkzeuge zur Verfügung, die für die Erreichung eines spezifischen Ziels (z.B. Erkennung, Verfolgung oder Trajektorienerstellung von Partikeln) notwendig sind, obwohl er sich auch auf andere Ressourcen stützen kann. Zu Illustrationszwecken verwendet er folgende Tools.

#### a) *picasso*:

Für die Lokalisierung und Auswertung von Bildern in verschiedenen Auflösungen.

#### b) *picasso\_addon*:

Für alle anderen Funktionen, die neben der Bindung von lokalisierten Partikeln genutzt werden können.

#### c) *trackpy*:

Für alles, was mit der Verbindung von zuvor lokalisierten Partikeln zu tun hat

## Nachteile

### 1. Schlecht dokumentiert:

Die Dokumentation des gesamten Handbuchs ist in der Tat sehr unterschiedlich. Sie variiert von sehr wenig Dokumentation für die Installation bis hin zu gar keiner Dokumentation für die Nutzung. Das macht die Installation und Nutzung natürlich schwierig. Da es immer zu den Paketen navigiert werden muss, die SPT verwendet (nämlich *picasso*, *picasso-addon* und *trackpy*), um die notwendigen Informationen (zu befolgenden Schritte) für die Installation und gegebenenfalls auch für die Verwendung zu erhalten.

### 2. Fehlender Benutzerleitfaden:

Das Fehlen eines Benutzerhandbuchs in einem Paket wie SPT, das sich auf andere Pakete stützt, um Ergebnisse zu liefern, schränkt sowohl die Einarbeitung in die Umgebung als auch deren Beherrschung stark ein. Der Benutzer muss die Funktionen oder Werkzeuge, die er benötigt, selbst aus den Dokumentationen der verschiedenen Pakete lernen und dann zu SPT zurückkehren, um sie anzuwenden.

## 4.4 Trackpy

Trackpy ist ein Python-Paket, das es ermöglicht aus einem Video bzw. einer Imagesequenz Partikel in unterschiedlichen Dimensionen (2D und 3D) zu erkennen und zu verfolgen. Hier wird natürlich die Zweidimensionalität anvisiert. Die Erkennung der Partikel erfolgt über eine der Funktionen des Paketes, nämlich die *locate*-Funktion. Diese verfügt über eine Reihe von Parametern, anhand derer die Qualität der Anerkennung angepasst werden kann.

### Ein-/Ausgabe

Trackpy verwendet als Eingabedaten Bilder oder Sequenzen von Bildern. Es kann auch aus Video bestimmter Formate (AVI, MOV, ...) Bildsequenz erstellten und damit arbeiten.

In Bezug auf die Ausgabe bietet es die Möglichkeit, die Ergebnisse in verschiedene Formate zu exportieren, nämlich:

- *csv*
- *HDF5*

- *SQL database*
- *Excel*

## Vorteile

### 1. Gründlich dokumentiert:

Eine ausführliche Dokumentation fast aller Funktionen der Software ist sicherlich ein großer Vorteil der Software. In der Tat trägt sie sehr dazu bei, dass sich der Benutzer schnell eingewöhnt und schnell Vertrauen fasst. Letzteres ist zweifellos eines der Hauptkriterien, auf das die Nutzer achten, wenn es um die Umsetzung geht *siehe*[12].

### 2. Benutzerleitfaden:

Im Einklang mit einer gründlichen Dokumentation gibt es auf der Website des Pakets eine Reihe von Tutorials (Anleitungen) mit reproduzierbaren praktischen Beispielen, die praktisch alle möglichen Anwendungsbereiche des Pakets abdecken. Auf diese Weise wird der neue und ungeübte Benutzer fast nie allein gelassen. Auf diese Weise wird sowohl ein schneller und allgemeiner Überblick über die Funktionen als auch die sogenannten fortgeschrittenen Funktionen abgedeckt.

### 3. Breites Spektrum an Möglichkeiten:

Die Lokalisierung von Elementen in einem Bild, die Verfeinerung der Koordinaten von Merkmalen auf eine Genauigkeit von weniger als einem Pixel und die Identifizierung von Merkmalen über die Zeit ebenso wie ihre Verknüpfung zu Trajektorien sind nur die Spitze des Eisbergs. In der Tat bietet "trackpy" weitaus mehr Funktionen, die je nach den Bedürfnissen des Nutzers von großer Bedeutung sein können. Dazu gehören unter anderem Module zur statistischen Datenanalyse, zur Bewegungsanalyse, zu Tracking-Tools und sogar zur Bewegungsvorhersage. All dies verleiht der Software also ein breites Spektrum an Möglichkeiten.

### 4. Hohe Parametrisierbarkeit

Die hohe Parametrisierbarkeit beruht hier auf der großen Anzahl an Möglichkeiten, die angeboten werden, um das Ergebnis der Funktionen zu verfeinern. In diesem Sinne bietet die Funktion *locate*[3] bis zu 18 nutzbare Parameter, die zur Verfeinerung der Suche und des Ergebnisses verwendet werden können. Obwohl einige von ihnen aus dem einen oder anderen Grund überflüssig oder unwirksam sind.

## 5. Beliebtheit in diesem Bereich

Aus all den in den vorangegangenen Punkten genannten Gründen und insbesondere dank der 3 und 4 wird sie vielfach bei der Entwicklung anderer Software verwendet, die in ihren Anwendungsbereich fällt. Dies hat dazu geführt, dass wir verschiedene Softwareprodukte finden, die seine Funktionen nutzen, wie z.B. *SPT 4.3*, *Particle Tracker 4.1* und viele andere, die im Rahmen dieser Arbeit nicht behandelt wurden.

## Nachteile

### 1. Fehlerhaftigkeit einiger Beschreibungen:

Trotz seiner gut etablierten Dokumentation muss man feststellen, dass es dennoch einige Fälle gibt, in denen die Dokumentation nicht immer korrekt war. Insbesondere hat sich die Anwendung von Prämissen aus der Dokumentation als unpraktikabel erwiesen. Auf diese Fälle wird im weiteren Verlauf dieser Arbeit eingegangen (siehe 5.2.4).

### 2. Fehlen einer grafischen Benutzeroberfläche:

Das Vorhandensein einer gut durchdachten und strukturierten grafischen Benutzeroberfläche, die nicht nur bei der Nutzung der Fülle der angebotenen Funktionen, sondern auch bei der Visualisierung der erzielten Ergebnisse helfen kann, wäre für die Nutzer natürlich noch interessanter und verführerischer gewesen.

**Beispiel:** Wir zeigen hier ein Beispiel für die Erkennung von Partikeln auf einem bestimmten Bild.

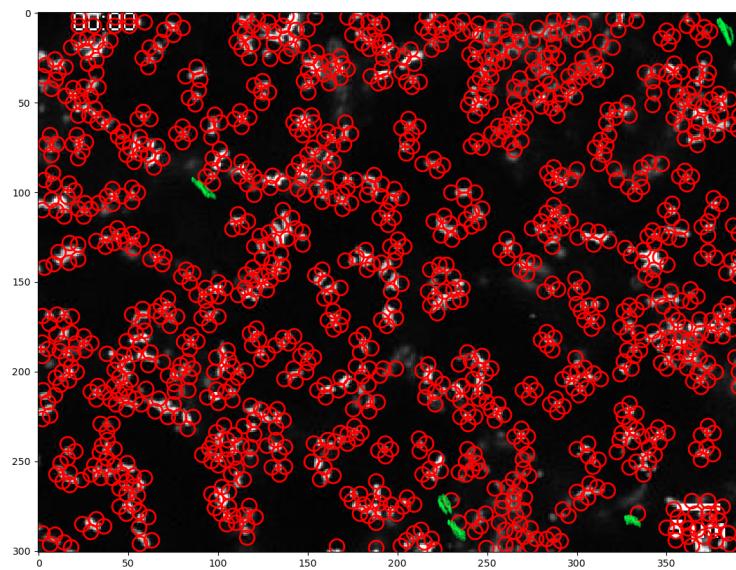


Abbildung 4.4: Beispiel von Partikelerkennung mit Trackpy.

In Anbetracht der Gesamtheit der Merkmale, die im Zusammenhang mit der Erwägung von Bibliotheken (Software), die zur Erreichung des oben genannten Ziels beitragen könnten, genannt wurden, ist **trackpy** der am besten geeignete Kandidat für unsere Arbeit.

# 5 Trackpy

Dieser Arbeitsteil wird sich hauptsächlich mit einer Einführung in die Software beschäftigen. Des Weiteren wird es um ihre Installation und die Erfahrungen des Nutzers mit der Software gehen.

## 5.1 Einleitung und Installation

Trackpy ist ein Python-Paket, das, wie im Abschnitt 4.4 beschrieben, nicht nur Informationen über die Partikel in einem Video (sowohl in hoher als auch in niedriger Auflösung) liefert, sondern auch mit ihnen interagieren (wie zum Beispiel einen Trajektorie der XY-Koordinaten zu zeichnen, die ein Partikel während der Aufnahmen erreicht hat.) kann, und zwar sowohl in 2D als auch in 3D. Das Programm verfügt über eine gut durchdachte Dokumentation, die die Installation und das Eintauchen in das Programm erleichtert. Aus diesem Grund werden wir uns bei der Installation nur auf die auf der Website [2] angegebenen Schritte stützen und eventuell weitere Schritte erwähnen, die nicht angegeben wurden.

Nachdem alle Schritte befolgt wurden, sollte die Installation ohne Probleme abgeschlossen werden können. Die Nutzung der Bibliothek verläuft jedoch noch nicht ganz so, wie im Benutzerhandbuch beschrieben. Daher werden die Unterschiede in der Nutzung im Abschnitt 5.2 behandelt.

## 5.2 Benutzererfahrung

### 5.2.1 In Bezug auf das Lesen des Videos

Die Bibliothek verspricht eine problemlose Ablesung verschiedener Videoformate wie *AVI*, *h.264*, *Tiff-Stacks* und viele andere, indem man einfach die Funktion *open()* oder *video()* aus der **PIMS**-Bibliothek [1] verwendet. Leider erfordert das Abspielen von Videos hier die Installation einer beliebigen Bibliothek, die das Abspielen

von Videos ermöglicht. In unserem Fall haben wir also **ffmpeg** und **scikit-image** verwendet.

Um dies zu tun, müssen diese beiden ebenfalls installiert werden. Wenn Pycharm wie empfohlen als IDE verwendet wird, muss die Installation folgendermaßen durchgeführt werden: **STRG+ALT+S** drücken, dann *Interpreter* in die Suchleiste eingeben und auf *Python Interpreter* klicken. Auf der rechten Seite des Fensters erscheint dann eine Liste der installierten Pakete. Jetzt muss man auf die Schaltfläche + klicken, um die zu installierenden Pakete zu finden, und dann auf *Installieren* klicken. Sicherheitshalber sollten Sie auch darauf achten, dass *trackpy*, *bokeh*, *ffmpeg* und *scikit-image* in der Liste der bereits installierten Pakete enthalten sind.

### **5.2.2 In Bezug auf die Einarbeitung in die Bibliothek**

Das Eintauchen und Einarbeiten in diese Bibliothek wird durch die Präsenz eines **”Walkthrough”** auf der Website sehr erleichtert. In diesem *Walkthrough* geht es vor allem um die grundlegende Nutzung fast aller wichtigsten Funktionen dieser Bibliothek anhand eines praktischen Beispiels, das Schritt für Schritt zeigt, wie man zu den Ergebnissen gelangt, die mit Bildern illustriert sind.

Dies ermöglichte uns einen leichten Einstieg und eine schnelle Nutzung der benötigten Funktionen. Abgesehen von der Wiedergabe von Videos, wie sie im Abschnitt 5.2.1 beschrieben ist.

### **5.2.3 In Bezug auf die Geschwindigkeit**

Was die Ausführungsgeschwindigkeit betrifft, finden wir, dass sie für die Informationen, die sie über die vorhandenen Partikel liefert, recht schnell ist. Wir sprechen hier in den meisten Fällen von wenigen Sekunden.

Bei zusätzlichem Bedarf an Leistung (Geschwindigkeit) gibt es auf ihrer Website auch eine Anleitung, wie man Trackpy noch schneller machen kann.

### 5.2.4 In Bezug auf Verwendung bestimmter Parameter der Funktion `locate`.

Tatsächlich gibt es zwar eine Dokumentation von fast allen Anwendungsfällen von Trackpy. Insbesondere hier bei den Parametern der Funktion „`locate()`“ gibt es eine Reihe von Parametern, die nicht so funktionieren, wie in der Dokumentation beschrieben. Die meisten von ihnen sind in diesem Abschnitt 6.1.1 zu finden. Aber um unsere Aussagen hier zu verdeutlichen, nehmen wir einen dieser Parameter, die nicht funktionieren, nämlich:

- **max\_iterations**: soll das Ergebnis der Erkennung in jeder Schleife verfeinern. Das Standardwert ist 10, egal welcher Wert zugewiesen wird, größer als 0, es gibt keinen sichtbaren Unterschied (siehe 5.1).

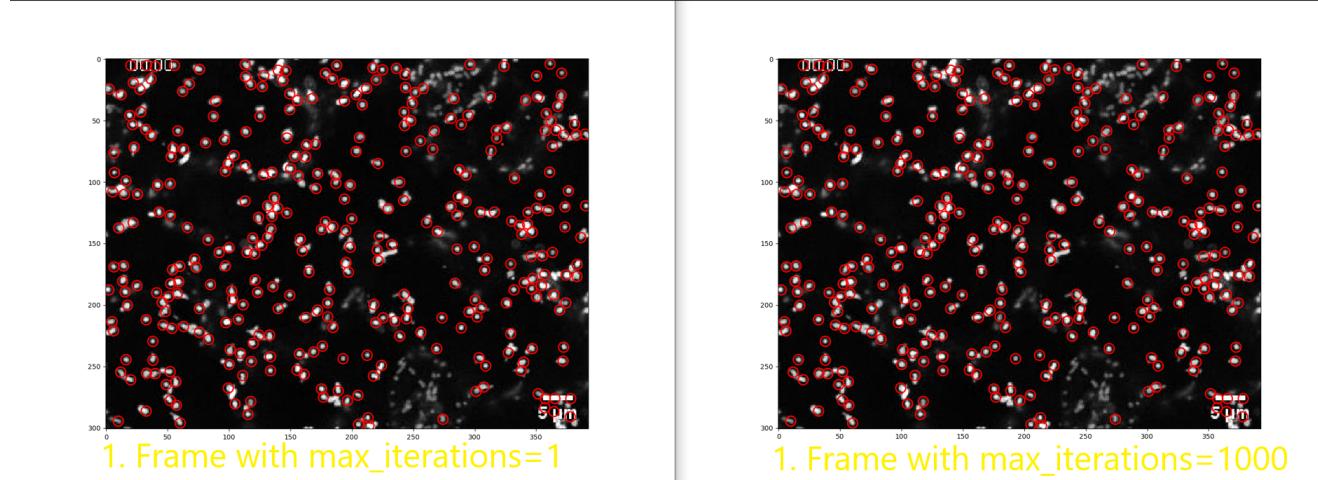


Abbildung 5.1: Vergleich zwischen `max_iterations=1` und `max_iterations=1000` auf demselben Frame.

# 6 Partikel-Erkennungssystem

## 6.1 Partikel-Erkennungssystem

PES (Partikel-Erkennungssystem) ist eine Software, die wir im Rahmen dieser Arbeit entwickelt haben und die es ermöglichen soll, die in einem Bild vorhandenen Partikel zu lokalisieren und auch die Ergebnisse zu bewerten, indem die den Parametern zugewiesenen Werte verglichen werden. Um die Partikel zu lokalisieren, verwendet das PES ausschließlich die Bibliothek *trackpy*, die es ihm mit ihrer Funktion "***locate()***" ermöglicht, die Partikel zu erkennen.

Im Folgenden werden wir die verschiedenen Parameter dieser Funktion vorstellen, die unserer Einschätzung der Effektivität und Wichtigkeit nach in zwei Gruppen eingeteilt sind.

### 6.1.1 Parameter der ***locate*-Funktion**

Folgende Parameter werden im Laufe dieser Arbeit angewandt:

1. ***raw\_image: array of images***

Wird für die endgültige Charakterisierung der zu erkennenden Partikel verwendet.

2. ***diameter: odd integer***

Entspricht der geschätzten Größe der Partikeln (in Pixel). Es wurde leider kein Grund für die Verwendung einer ungeraden Zahl angegeben.

3. ***minmass: float***

Minimale eingebaute Helligkeit. Dies ist ein Schlüsselparameter, um störende Merkmale zu entfernen. Der Standardwert ist '*None*'.

*None* ist ein Typ in Python, der es ermöglicht, einer Variablen keinen festen Wert zuzuweisen. Da *minmass* optional ist, wurde *None* gewählt, damit der

Wert nicht direkt mit der Partikelerkennung interferiert. Und None ist eine gute Möglichkeit, dies in Python zu tun.

4. **maxsize: float**

Maximaler Gyrationsradius der Helligkeit.

Der Gyrationsradius [8] definiert den Abstand von der Rotationsachse zu dem Punkt, an dem die gesamte Masse eines Körpers konzentriert sein soll, der das gleiche Trägheitsmoment wie die ursprüngliche Form aufweist.

5. **separation: float**

Minimaler Abstand zwischen den Partikeln. Der Standardwert ist *diameter* + 1.

Natürlich können auch negative Werte angegeben werden, aber das würde das Ergebnis nur verfälschen. Wie in dem Punkt, in dem es um die richtige Wahl für die Separation geht, erwähnt, führt jeder Wert unterhalb des Diameters nur zu einer Vervielfachung der Anzahl der erkannten Partikel, obwohl es sich um dieselben Partikel handelt, die sich an denselben Positionen befinden.

6. **noise\_size: float or tuple**

Breite des Gaußschen Weichzeichner-kerns, in Pixeln. Der Standardwert ist 1. Wobei ein Gaußscher Weichzeichner ist nichts anderes als die Anwendung einer mathematischen Funktion auf ein Bild, um es weichzuzeichnen.

7. **topn: interger**

Gibt lediglich die N hellsten Merkmale über minmass zurück. Wenn 'None' (Voreinstellung), werden sämtliche Eigenschaften oberhalb von minmass zurückgegeben.

Die Wahl von 'None' als Standardwert ist hier nichts anderes als derselbe Wert, der auch in 3 erwähnt wird.

Neben diesen Parametern, die wir im Laufe unserer Arbeit verwenden werden, gibt es noch andere, die ebenfalls von trackpy zur Verfügung gestellt werden. Sie sind jedoch aus Gründen der Nützlichkeit oder Funktionsfähigkeit nicht aufgelistet. Im Folgenden werden diese Parameter aufgelistet:

- **smoothing\_size: float or tuple**

Die Größe der Seiten des quadratischen Kerns, der bei der Glättung verwendet wird, in Pixeln. Der Standardwert ist der *diameter*.

- **threshold: float**

Schneidet das Ergebnis des Bandpasses unterhalb dieses Wertes ab. Die Schwellwertbildung wird auf das Bild angewendet, das bereits vom Hinter-

grund subtrahiert wurde. Standardmäßig 1 für Vollbilder und 1/255 für schwächende Bilder.

- **invert:** boolean

Auf True gesetzt, wenn die Merkmale dunkler als der Hintergrund sind. Standardmäßig False. Dies wird deprecated. Es sollte stattdessen eine geeignete PIMS-Pipeline verwenden, um ein Bild oder eine Sequenz von Bildern zu invertieren.

- **percentile:** float

Die Merkmale sollten eine hellere Spitze haben als die Pixel in diesem Perzentil. Dadurch werden störende Peaks eliminiert.

- **preprocess:** boolean

Setze auf False, um die Bandpass-Vorverarbeitung zu deaktivieren.

- **max\_iterations:** integer

Maximale Anzahl der Schleifen zur Verfeinerung des Massenschwerpunkts, Standardwert 10. Er muss immer mindestens größer als 0 sein. Für den Fall, dass wir ihn verwenden wollen.

- **filter\_before:** boolean

Es wird nicht mehr unterstützt, da es die Leistung nicht verbessert.

- **filter\_after:** boolean

Diese Einstellung wurde deprecated: Verwende stattdessen minmass und maxsize.

- **characterize:** boolean

Berechnet die „Extras“: Exzentrizität, Signal, ep. Standardmäßig wahr.

- **engine:** ‘auto’, ‘python’, ‘numba’

Bestimmt, mit welchem Compiler die Funktion ausgeführt werden soll.

Ein PANDA.DATFRAME mit den Daten *y-koordinaten*, *x-koordinaten*, *mass*, *size*, *ecc*, *signal*, *raw\_mass*, *ep*, *frame* wird als Rückgabewert zurückgegeben. Dies gilt für jedes der gefundenen Partikel (Siehe 6.6). Ausführlichere Informationen zu weiteren Parametern sowie zu den Obengenannten ist auf [3] zu sehen. An dieser Stelle kann folgende Frage aufgeworfen werden: Was sind die besten Parameterwerte?

Beachte, dass einfachheitshalber, während der gesamten Parametereinstellung nur das erste Bild unseres Videos betrachtet wird.

### 6.1.2 Ermittlung der optimalen Parameterwerte der Locate-Funktion von Trackpy.

Hier wird es ein Antwortversuch auf die zuletzt gestellte Frage eingegangen. Zur Erreichung dieses Ziels wird mit der Erkennung begonnen, indem nur die geforderten Parameter(raw\_image und diameter) verwendet werden und nach und nach weitere hinzugefügt werden, um die Suche zu verfeinern.

#### 1. Ermittlung von diameter

`locate(f, d): Wobei f = raw_image und d = diameter`

Während `frames[0]` dem ersten Bild der Videoaufnahme bzw. der Imagesequenz entspricht, `Frames` bilden der Gesamtheit der Bilder des Videos im Laufe der Zeit. Diese Angabe, die vom Typ Array ist, stellt eine Voraussetzung für die Ausführung von Funktionen dar.

Für den Durchmesser wird zunächst willkürlich eine ziemlich kleine ungerade Zahl genommen, um die Ergebnisse zu sehen und eine Annäherung an den Wert, den wir verwenden sollen, zu erhalten. Zunächst nehmen wir also einen Durchmesser von drei ( $d=3$ ).

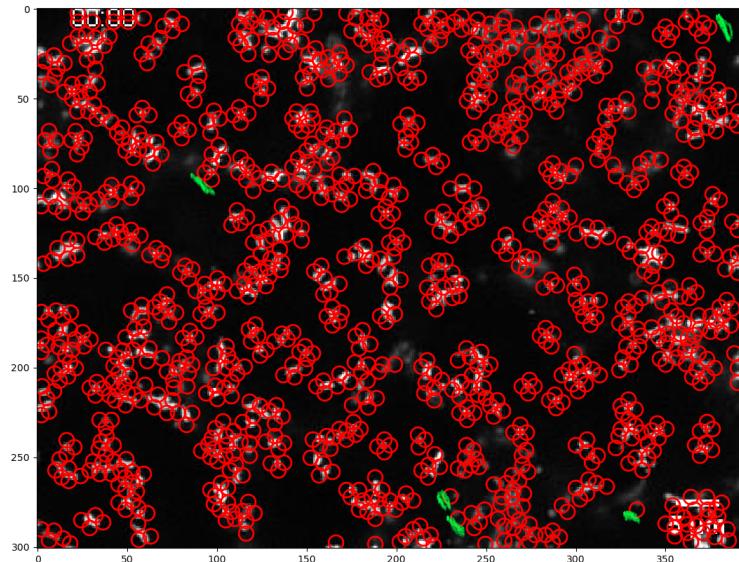


Abbildung 6.1: `locate()`-Funktion auf 0. Frame mit  $diameter=3$

Das Bild zeigt eine Lokalisierung der Partikel. Es wurde dabei fast alle Elemente des Bildes erkannt, wobei offensichtlich eine große Menge an *False Positive* erkannt wird.

In grüner Farbe haben wir auf dem Bild einige Beispiele für *false Positive* Ergebnisse markiert. Ein *false Positive* Befund liegt vor, wenn ein Partikel entdeckt wird, das nicht hätte entdeckt werden dürfen. Denn entweder sind sie zu klein, zu dunkel oder sogar zu viele Detektionskreise um das gleiche Partikel.

Eine Verfeinerung der Lokalisierung würde somit einen größeren Durchmesser erfordern. Dies erfolgt in der Folge durch die Verwendung einer immer noch ungeraden Zahl, die jedoch einen größeren Wert hat. In diesem Fall ist es neun, da es so viele *False Positives* gibt. `locate(frames[0], 9)`

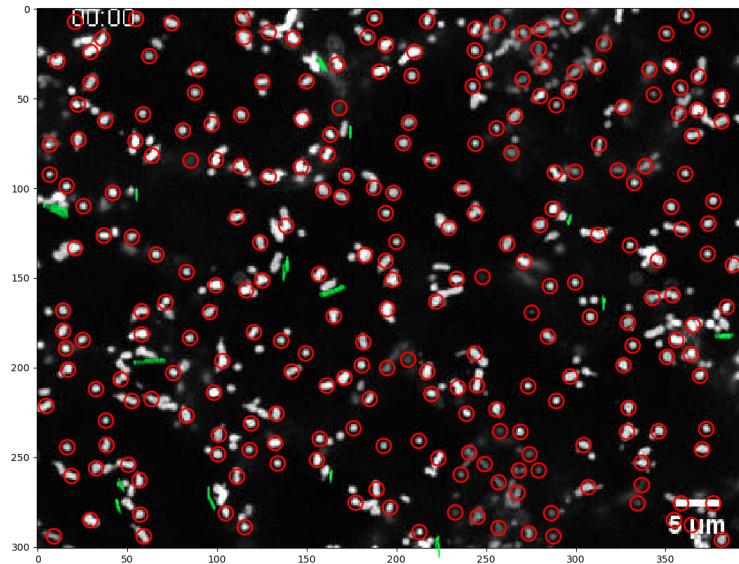


Abbildung 6.2: `locate()`-Funktion auf 0. Frame mit diameter=9

Diesmal gibt es viel weniger ungewollte Teilchen. Allerdings hat sich eine große Anzahl von *False Negative* gebildet.

Auch hier sind auf dem Bild paar Beispiele von *false Negative* Ergebnisse in grün markiert. Hier spricht man von *False Negative* Befund, wenn ein Partikel nicht entdeckt wird, das hätte entdeckt werden müssen. Aus diesem Grund wurde nacheinander der Durchmesser von sieben und dann von fünf ausprobiert.

`locate(frames[0], 7)` gefolgt `locate(frames[0], 5)`

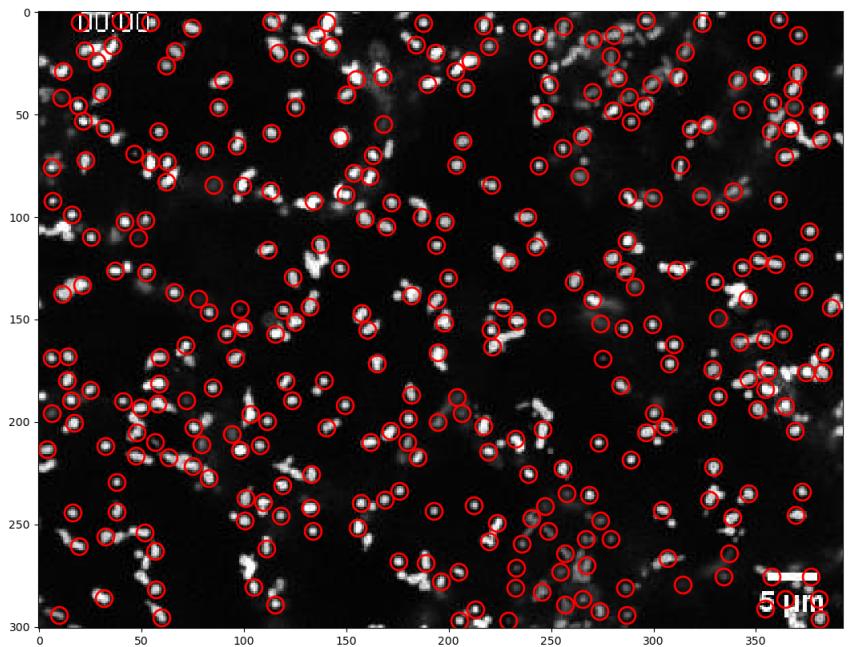


Abbildung 6.3: locate()-Funktion auf 0.  
Frame mit diameter=7

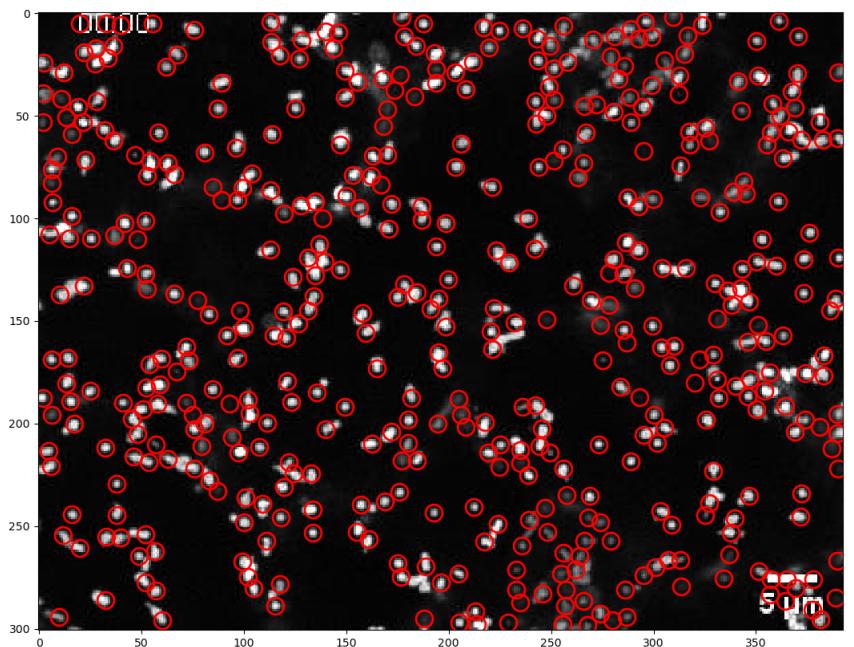


Abbildung 6.4: locate()-Funktion auf 0.  
Frame mit diameter=5

In Anbetracht des Ziels, einen Durchmesser zu finden, der die Erkennung möglichst vieler Partikel ermöglicht und gleichzeitig möglichst wenig unerwünschte Partikel enthält, ist es besser, mit dem Durchmesser 5 fortzufahren. Denn aus den zuvor verwendeten Durchmessern geht hervor, dass bei diesem Bild die Anzahl der nicht-lokalisierten Teilchen umso größer ist, je höher der Durchmesser ist.

Dies ist nicht als Allgemeingültigkeit zu verstehen, da verschiedene Videos unterschiedliche Arten von Partikeln mit variierenden Größen und Dicken aufweisen. Es wäre ratsam, die Parameter bei jedem neuen Video zu testen.

Tatsächlich lassen sich insgesamt 475 Partikel finden, von denen ca. 121 unerwünscht waren und kaum fehlten (in grün markiert). Dies entspricht einer ungefähren Rate von 25.47%.

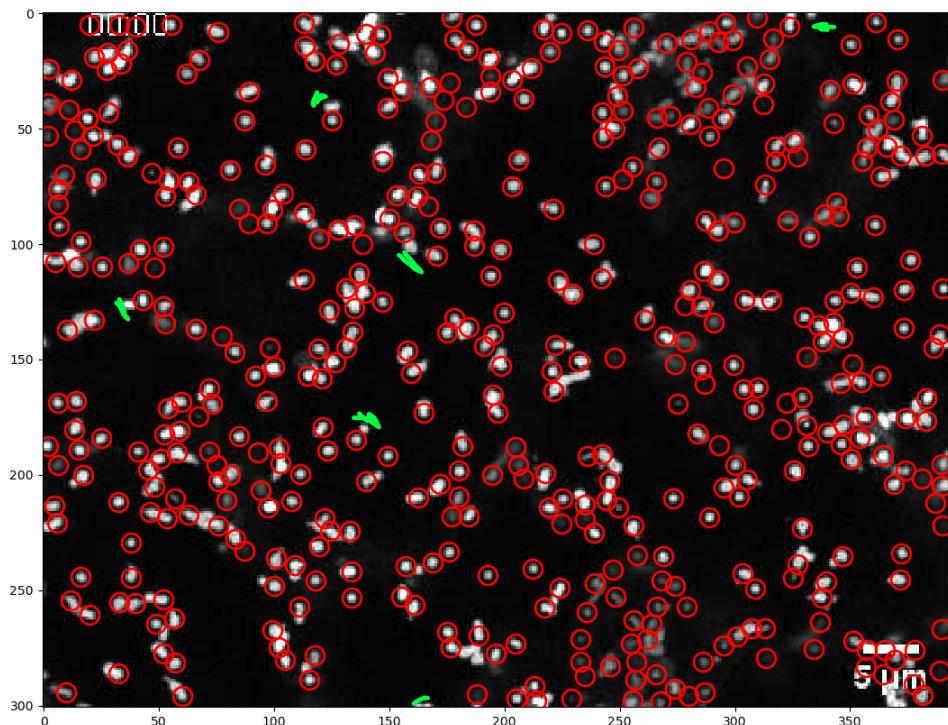


Abbildung 6.5: locate()-Funktion auf 0. Frame mit diameter=5

## 2. Ermittlung von minmass

**locate(f, d, minmass):**

Wie bereits erwähnt, spiegelt *minmass* die inhärente und eingebaute minimale Helligkeit jedes lokalisierten Partikels wider. Das Ziel ist es nun, die zuvor ermittelten, zu dunklen Partikel herauszufiltern. Daher ist es notwendig, methodisch mit dem Parameter *minmass* zu spielen, um dieses Ziel zu erreichen. Es gibt natürlich mehrere Möglichkeiten, den optimalen Wert für die gesuchte Parameter zu finden. Allerdings wird hier die folgende Logik verfolgt:

Aus dem DataFrame der letzten Suche (`locate(frames[0], 5)`) wurde ja 475 Partikel gefunden. Davon sind ca. 25.47%, also 121 unerwünscht. Diese Zahl entspricht so fast allen gefundenen zu dunklen Partikel. In anderen Worten, stellt sie die Elemente dar, deren *minmass* zu niedrig ist. So könnte der DataFrame verwendet werden und ihn nach der Spalte *mass* absteigend sortieren. Dies würde dazu führen, dass unsere dunkelsten Partikel am Ende der Liste (Tabelle) positioniert werden und die hellsten ganz oben. Dazu muss keine Funktion implementiert werden, sondern es genügt, die Funktion `sort_values()` aus der Panda. Dataframe-Bibliothek aufzurufen. Der Aufruf sowie die Tabelle sähe jeweils dann wie folgt aus:

```
dataframe.sort_values(by=['mass'], ascending=False)
```

	♦ y	♦ x	♦ mass	♦ size	♦ ecc	♦ signal	♦ raw_mass	♦ ep	♦ frame
0	1.97655	176.98325	169.30889	1.21272	0.48280	29.77794	1753.00000	0.02039	0
1	1.65487	309.41593	32.04674	1.05596	0.42957	7.08999	777.00000	0.05016	0
2	4.17298	112.93003	291.82386	1.28836	0.15063	37.43513	2308.00000	0.01524	0
3	4.02090	296.07023	339.18497	1.21756	0.28070	52.18230	2016.00000	0.01758	0
4	3.65717	361.00551	308.55623	1.20240	0.15441	51.89870	1586.00000	0.02269	0
5	4.94174	20.01910	296.92865	1.35242	0.45187	49.62991	1303.00000	0.02807	0
6	5.08451	32.98483	261.76232	1.32154	0.44768	48.49551	1345.00000	0.02712	0
7	5.04670	55.62187	249.00034	1.30314	0.12840	43.67432	1951.00000	0.01820	0
8	5.15986	187.78239	402.71125	1.26714	0.20241	53.60030	2277.00000	0.01546	0
9	5.83473	39.98536	271.12110	1.36988	0.45337	49.34631	1357.00000	0.02686	0
10	6.23242	256.08034	197.66883	1.25283	0.11802	28.07635	1451.00000	0.02498	0
11	5.92622	323.69447	353.64855	1.30356	0.02827	48.49551	2299.00000	0.01531	0
12	6.91337	216.93624	409.23404	1.28184	0.11378	55.30190	2515.00000	0.01393	0
13	7.26920	235.90578	358.18614	1.24351	0.14073	51.04791	1990.00000	0.01782	0
14	8.13769	75.89409	321.31821	1.27713	0.31391	40.83832	2282.00000	0.01543	0
15	7.75574	288.75738	172.99568	1.19356	0.36273	31.47954	1476.00000	0.02452	0
16	8.86983	139.73601	233.11877	1.34996	0.11650	27.79275	2971.00000	0.01171	0
17	9.16776	146.15543	344.85696	1.13880	0.23418	61.54109	2129.00000	0.01659	0
18	8.63941	224.71464	346.84216	1.20030	0.16519	56.15270	1833.00000	0.01944	0
19	11.25685	178.18721	248.43314	1.23172	0.15535	38.28593	1851.00000	0.01924	0
20	11.07851	280.90358	205.89322	1.27002	0.34491	26.37475	1911.00000	0.01860	0
21	10.98058	316.68689	116.84298	1.22574	0.06571	19.00116	854.00000	0.04498	0
22	11.16351	370.71685	284.45027	1.19737	0.23169	45.37592	1492.00000	0.02424	0
23	12.25278	292.80556	102.09581	1.21906	0.18829	15.59797	950.00000	0.03985	0
24	11.70541	299.01752	178.10047	1.24537	0.30674	25.52395	1457.00000	0.02486	0
25	13.11283	128.01832	294.09266	1.26826	0.44311	42.53992	2490.00000	0.01408	0
26	12.58599	243.18926	311.67582	1.20320	0.11061	51.04791	2102.00000	0.01682	0
27	12.87879	270.38567	205.89322	1.22643	0.15091	32.33034	1699.00000	0.02108	0
28	14.12027	113.24399	247.58234	1.31865	0.29931	33.74834	2255.00000	0.01562	0
29	13.64028	350.13769	329.54259	1.23419	0.17192	50.48071	1710.00000	0.02094	0
30	16.05483	36.20786	346.55856	1.27323	0.09083	46.22671	2800.00000	0.01246	0
31	16.00707	184.08798	361.02213	1.17949	0.23087	57.57069	2245.00000	0.01569	0
32	15.76045	249.24791	203.62442	1.25007	0.17406	30.91234	1562.00000	0.02307	0
33	17.05800	27.85688	303.16784	1.31942	0.38830	37.71873	2905.00000	0.01199	0
34	17.15275	143.64664	278.49468	1.26724	0.29019	37.43513	2548.00000	0.01374	0
35	16.72688	219.74946	263.74751	1.26278	0.26127	37.15153	1623.00000	0.02214	0
36	18.88504	21.89142	310.82502	1.27189	0.10055	39.98753	2396.00000	0.01466	0
37	19.25497	193.25083	342.58817	1.33874	0.26635	45.09232	2645.00000	0.01322	0

Abbildung 6.6: Ein Teil des initialen Dataframes

	↓ y	↓ x	↓ mass	↓ size	↓ ecc	↓ signal	↓ raw_mass	↓ ep	↓ frame
174	102.21197	41.90898	464.25234	1.28760	0.11428	56.71990	2730.00000	0.01279	0
85	45.77866	18.99019	462.55074	1.25893	0.15586	62.39188	2571.00000	0.01362	0
427	268.07495	175.17676	454.04276	1.23326	0.20214	66.07868	2367.00000	0.01485	0
199	124.11196	43.00509	445.81838	1.26461	0.09220	62.39188	2642.00000	0.01323	0
371	225.21720	238.93567	445.25118	1.25936	0.22805	60.97389	2454.00000	0.01430	0
441	275.71162	377.90687	441.56438	1.29819	0.07056	58.98869	2804.00000	0.01244	0
352	213.94323	97.17624	429.65321	1.33176	0.15137	47.07751	2862.00000	0.01218	0
285	181.05692	58.93051	428.51881	1.33402	0.17592	47.36111	2951.00000	0.01180	0
457	286.25316	364.42193	426.81721	1.24639	0.10282	70.33267	1833.00000	0.01944	0
97	53.13449	21.26831	425.96641	1.23826	0.22768	59.55589	2500.00000	0.01402	0
446	280.24000	104.81133	425.39921	1.30307	0.03800	54.16750	2585.00000	0.01354	0
359	218.25936	288.91110	424.26482	1.22393	0.12786	63.52628	2183.00000	0.01616	0
148	84.93373	99.08568	423.69762	1.32148	0.09162	47.07751	2970.00000	0.01172	0
132	73.04088	62.78887	423.13042	1.28926	0.06547	53.03310	2654.00000	0.01317	0
183	110.10858	352.86394	423.13042	1.27383	0.09383	55.30190	2272.00000	0.01550	0
348	211.84859	32.24428	421.42882	1.26449	0.18456	55.86910	2332.00000	0.01508	0
334	204.10700	368.93876	421.42882	1.29446	0.26090	49.62991	2557.00000	0.01369	0
319	198.18801	180.20957	420.86162	1.28803	0.18012	53.03310	2347.00000	0.01498	0
383	239.02259	109.09582	414.33883	1.27928	0.34046	55.58550	2721.00000	0.01283	0
264	166.83436	383.18480	414.33883	1.29708	0.06546	52.46590	2676.00000	0.01306	0
258	162.19162	309.94368	412.92084	1.24780	0.14322	56.71990	2239.00000	0.01574	0
125	69.37276	162.78473	412.35364	1.29675	0.14442	52.18230	2529.00000	0.01385	0
344	210.03854	273.02684	412.07004	1.18228	0.19113	69.19827	2022.00000	0.01752	0
177	106.92808	376.10996	410.08484	1.28697	0.14053	51.04791	2263.00000	0.01556	0
169	98.83679	16.06846	410.08484	1.21795	0.23059	62.39188	2309.00000	0.01524	0
377	233.93763	372.22869	409.23404	1.24314	0.17808	57.85429	2165.00000	0.01630	0
12	6.91337	216.93624	409.23404	1.28184	0.11378	55.30190	2515.00000	0.01393	0
301	189.31897	123.22307	408.09965	1.28983	0.22348	53.60030	2209.00000	0.01596	0
260	162.87447	303.82118	404.41285	1.23756	0.14141	55.86910	2197.00000	0.01605	0
8	5.15986	187.78239	402.71125	1.26714	0.20241	53.60030	2277.00000	0.01546	0
347	210.98449	244.24048	402.14406	1.28707	0.03669	51.61510	2549.00000	0.01374	0
294	186.81016	346.01694	401.86046	1.23778	0.14596	57.85429	2089.00000	0.01693	0
462	290.63726	377.56951	401.86046	1.23064	0.08276	68.63107	1810.00000	0.01971	0
437	274.78233	176.79364	401.29326	1.25819	0.22714	53.60030	2482.00000	0.01413	0
211	131.75337	329.99291	400.15886	1.21180	0.24847	62.95908	2037.00000	0.01739	0
162	92.96596	172.05390	399.87526	1.33049	0.14158	42.25632	2360.00000	0.01489	0
221	136.89062	65.98580	399.30806	1.24773	0.21941	56.71990	2321.00000	0.01515	0
366	222.07402	255.86548	398.45726	1.33045	0.07965	44.52512	2573.00000	0.01360	0

Abbildung 6.7: Ein Teil des sortierten Dataframes

Jetzt wird es auf dem sortierten Dataframe eine weitere Funktion aufgerufen, um lediglich nur die 354 (also 475 – 121) gewünschte Partikel bzw. hellsten

zu behalten. Eine solche Funktion `head()` wird auch von Panda-Dataframe bereitgestellt. Aus diesem Ergebnis reicht es aus, die von Python angebotene Funktion `min()` auszuführen, um den kleinsten Wert in der Spalte `mass` zu erhalten. Die Aufrufe sähen dann wie folgt aus:

```
dataframe.head(354)
min(dataframe['mass'])
```

**189.72805** ist hier das Ergebnis der vorherigen Vorgänge und damit auch der minimale Wert von `mass`, den ein Partikel haben muss. Es sei der `minmass` Parameter der Funktion `locate()`.

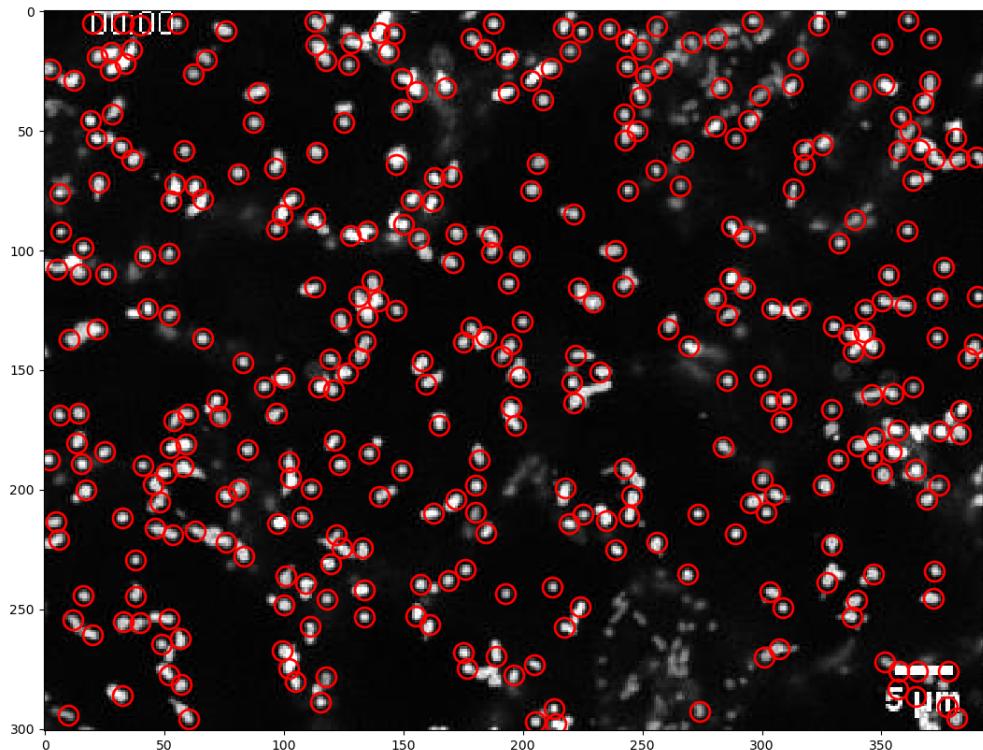
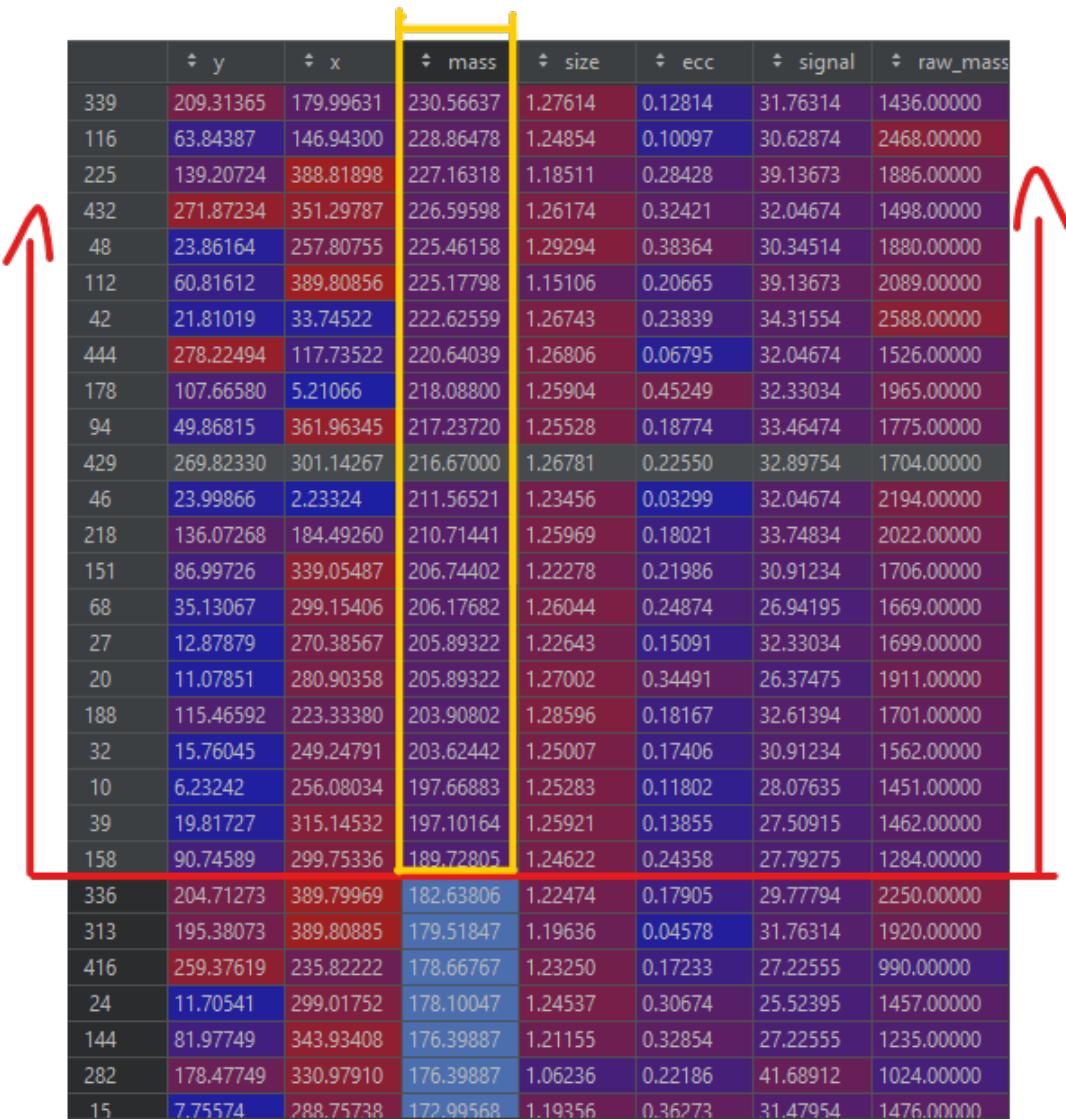


Abbildung 6.8: `locate()`-Funktion auf 0. Frame mit 'mimass=189.72805'

Aus diesem Bild geht hervor, dass fast alle gewünschten Partikel lokalisiert bleiben. Allerdings werden einige von ihnen doppelt gezählt, was Gegenstand der Verwendung anderer Parameter sein wird. Dennoch gibt es noch einige

Partikel, die nicht lokalisiert werden sollten. Es handelt sich dabei um etwa 12 Teilchen. Es wäre daher ratsam, die *minmass* schrittweise zu erhöhen, bis ein zufriedenstellendes Ergebnis erzielt wird.

Zum Festlegen des Wertes, der in jedem Schritt verwendet werden soll, schauen Sie in der Tabelle (Dataframe), die mit *dataframe.sort\_values(by=['mass'], ascending=False)* erstellt wurde, beginnend mit der 121. Zeile von unten nach oben.(121. Zeile entspricht der Anzahl der Teilchen mit geringer Helligkeit)



	↓ y	↓ x	↓ mass	↓ size	↓ ecc	↓ signal	↓ raw_mass
339	209.31365	179.99631	230.56637	1.27614	0.12814	31.76314	1436.00000
116	63.84387	146.94300	228.86478	1.24854	0.10097	30.62874	2468.00000
225	139.20724	388.81898	227.16318	1.18511	0.28428	39.13673	1886.00000
432	271.87234	351.29787	226.59598	1.26174	0.32421	32.04674	1498.00000
48	23.86164	257.80755	225.46158	1.29294	0.38364	30.34514	1880.00000
112	60.81612	389.80856	225.17798	1.15106	0.20665	39.13673	2089.00000
42	21.81019	33.74522	222.62559	1.26743	0.23839	34.31554	2588.00000
444	278.22494	117.73522	220.64039	1.26806	0.06795	32.04674	1526.00000
178	107.66580	5.21066	218.08800	1.25904	0.45249	32.33034	1965.00000
94	49.86815	361.96345	217.23720	1.25528	0.18774	33.46474	1775.00000
429	269.82330	301.14267	216.67000	1.26781	0.22550	32.89754	1704.00000
46	23.99866	2.23324	211.56521	1.23456	0.03299	32.04674	2194.00000
218	136.07268	184.49260	210.71441	1.25969	0.18021	33.74834	2022.00000
151	86.99726	339.05487	206.74402	1.22278	0.21986	30.91234	1706.00000
68	35.13067	299.15406	206.17682	1.26044	0.24874	26.94195	1669.00000
27	12.87879	270.38567	205.89322	1.22643	0.15091	32.33034	1699.00000
20	11.07851	280.90358	205.89322	1.27002	0.34491	26.37475	1911.00000
188	115.46592	223.33380	203.90802	1.28596	0.18167	32.61394	1701.00000
32	15.76045	249.24791	203.62442	1.25007	0.17406	30.91234	1562.00000
10	6.23242	256.08034	197.66883	1.25283	0.11802	28.07635	1451.00000
39	19.81727	315.14532	197.10164	1.25921	0.13855	27.50915	1462.00000
158	90.74589	299.75336	189.72805	1.24622	0.24358	27.79275	1284.00000
336	204.71273	389.79969	182.63806	1.22474	0.17905	29.77794	2250.00000
313	195.38073	389.80885	179.51847	1.19636	0.04578	31.76314	1920.00000
416	259.37619	235.82222	178.66767	1.23250	0.17233	27.22555	990.00000
24	11.70541	299.01752	178.10047	1.24537	0.30674	25.52395	1457.00000
144	81.97749	343.93408	176.39887	1.21155	0.32854	27.22555	1235.00000
282	178.47749	330.97910	176.39887	1.06236	0.22186	41.68912	1024.00000
15	7.75574	288.75738	172.99568	1.19356	0.36273	31.47954	1476.00000

Abbildung 6.9: Sortierte Tabelle

So werden nach und nach die Werte 197,1016 (d. h. 197), 197,6688 (d. h. 198) und so weiter verwendet.

Die Verwendung der Funktion `locate(frames[0], 5, minmass=197)` ergibt so gut wie keine Änderung der Lokalisierung. Da insgesamt immer noch 353 Partikel erkannt wurde. Daher wird sich der nächste Versuch mit dem folgenden Wert beschäftigen. Genauer gesagt `minmass = 198`.

Auch hier wurden nur zwei Teilchen weniger gefunden. Das sind insgesamt 351 Partikel. Obwohl es fast unmöglich ist, diese beiden Teilchen auf dem Bild zu erkennen. Es wäre klug, den nächsten Wert unserer Tabelle zu nehmen, der derzeit bei 203,6244 (also 204) liegt, und erneut zu versuchen, eine Lokalisierung durchzuführen. (Siehe Bild 6.11)

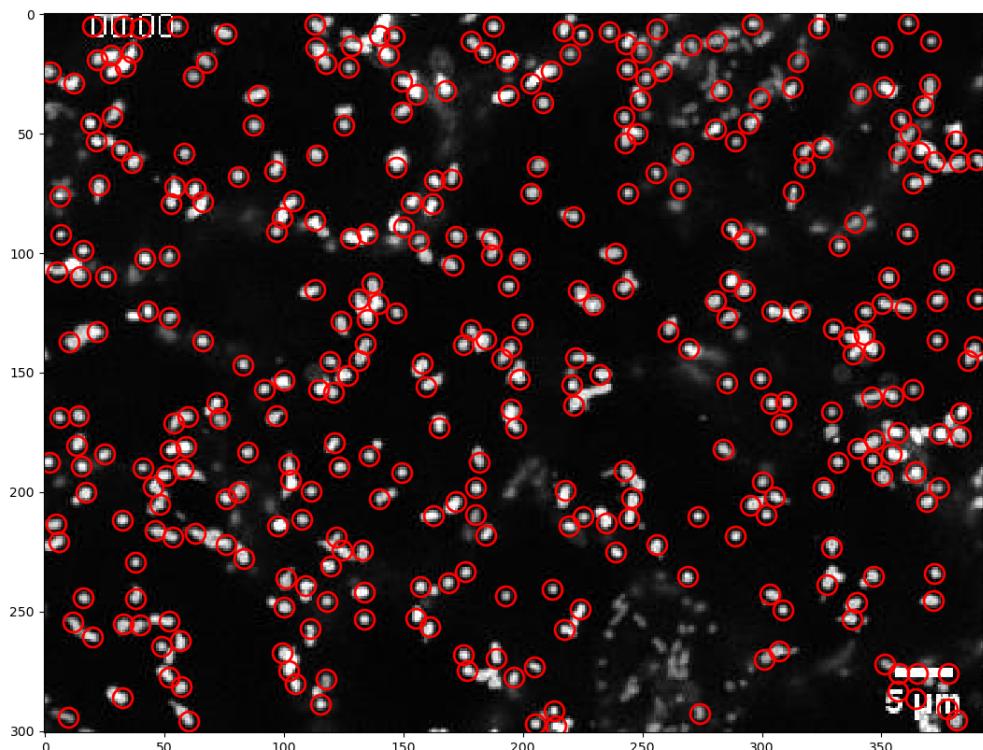


Abbildung 6.10: `locate()`-Funktion auf 0. Frame mit 'minmass=197'

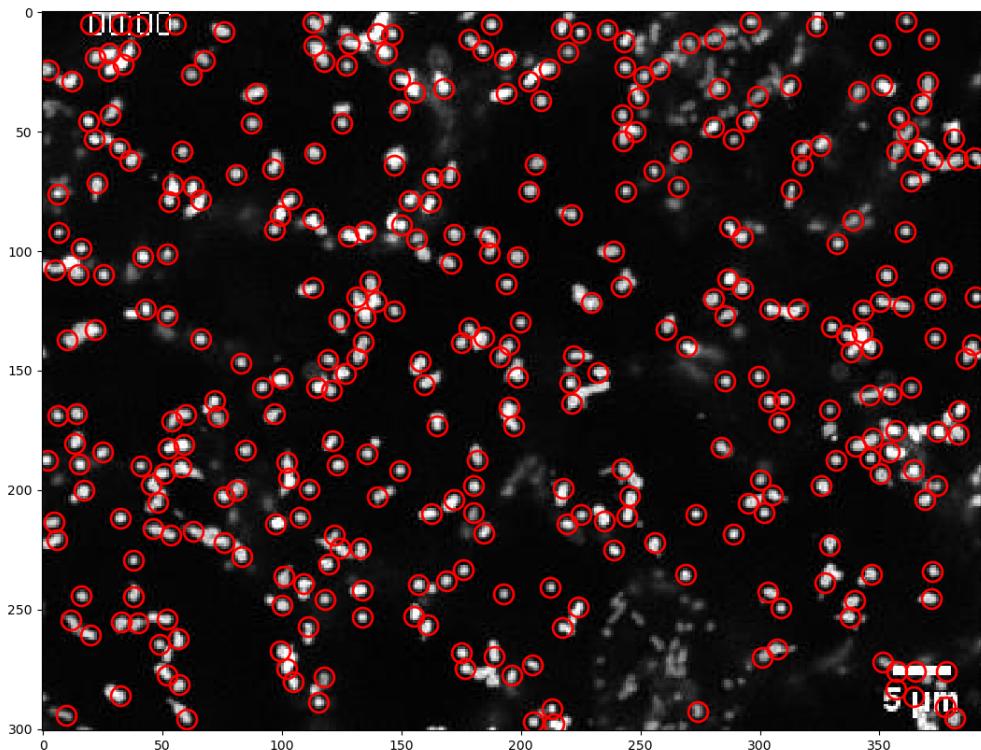


Abbildung 6.11: `locate()`-Funktion auf 0. Frame mit 'mimass=204'

Bei der Verfolgung dieser Logik kommen wir schnell auf einen *minmass* Wert von **210**.

Wo es deutlich zu erkennen ist, dass weitere unerwünschte Partikel nicht mit erkannt wurde. Es wurde insgesamt hier **345** Partikel gefunden. Jedoch muss es festgestellt werden, dass ab **211** nach oben, werden zwar weniger unerwünschte Partikel gefangen, aber auch erwünschte. Wie es die Visualisierung auf Bild (Bild 212) zeigt. Deswegen wird es dem weiteren den Parameter **Separation** gewidmet, um weitere unerwünschte zu eliminieren.

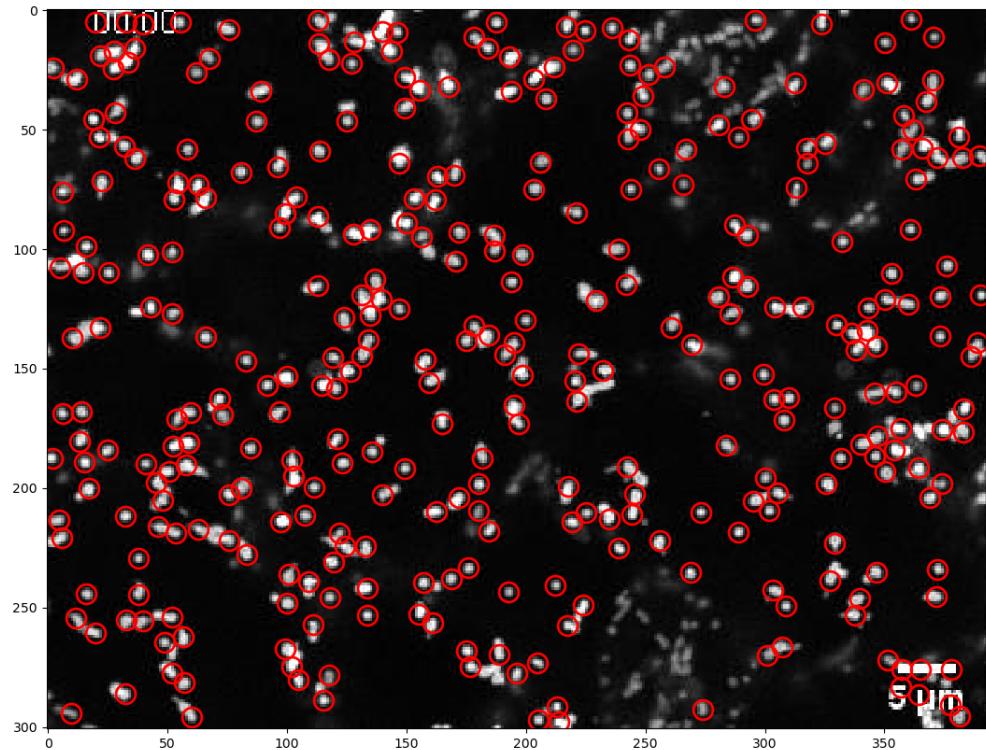


Abbildung 6.12: locate()-Funktion auf 0. Frame mit 'mimass=210'

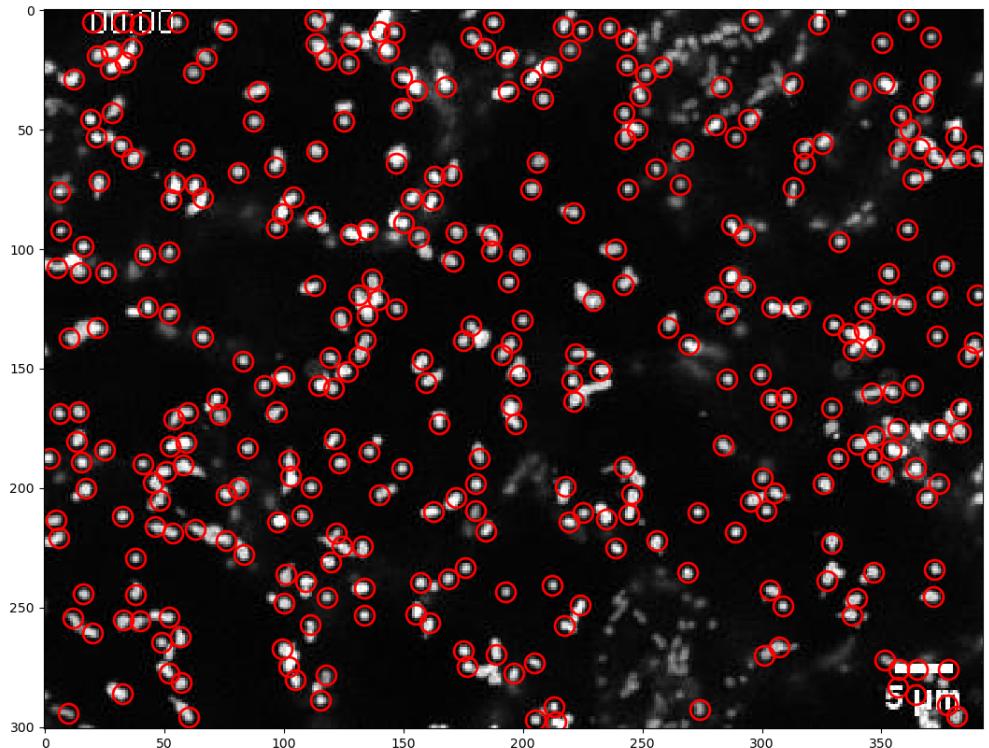


Abbildung 6.13: `locate()`-Funktion auf 0. Frame mit 'mimass=212'

### 3. Ermittlung von separation `locate(f, d, minmass, separation)`

Hier werden wir Werte für `separation` ausprobiert, um auf bessere Resultate zu kommen. Es wäre interessant zu erwähnen, dass es sich dabei um den minimalen Abstand zwischen zwei Teilchen handelt. Der Standardwert ist *Durchmesser + 1*. In diesem Fall ist es 6. Mit diesem neuen Parameter werden wir versuchen, all jene Partikel zu eliminieren, die doppelt erkannt werden. Ohne die Qualität der bisherigen Erkennung zu beeinträchtigen.

Denn es scheint offensichtlich, dass, wenn der Wert zu hoch ist, mehrere bisher erkannte Partikel nicht mehr erkannt werden. Weil sie zu nahe beieinander liegen. Bei `separation = 6` bleibt die Erkennung unverändert und ergibt auch eine Anzahl von **345**.

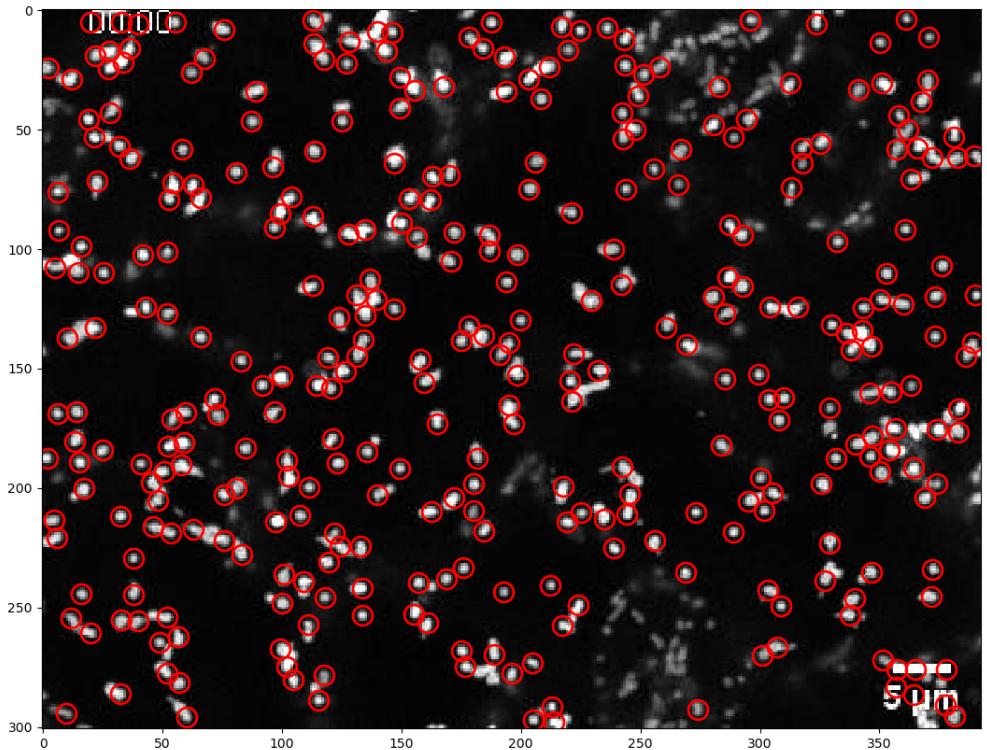


Abbildung 6.14: `locate()`-Funktion auf 0. Frame mit 'separation=6.0'

Es wird dann als nächstes zuerst **7** als Parameterwert ausprobieren. Trotz der relativ kleinen Anzahl an insgesamt erkannten Partikel also **312**. Was auf dem folgenden Bild sichtbar ist, wird es ungefähr **10** gewünschte Partikel verloren. Diese sind in gelb auf dem Bild markiert. Natürlich hat der Parameterwert nicht nur Verschlechterung gezogen sondern auch positive Effekte. So ist es auch leicht in grün auf dem Bild Ausbesserungen zu sehen. Es handelt sich hier nämlich um **6** Partikel, die sich mehrfach erkennen lies. Allerdings, da die Anzahl an nicht mehr erkannte gewünschte Elemente größer ist als die von nicht gewünschten die eliminiert wurde, wird der Wert des Parameters dann nach unten korrigiert.

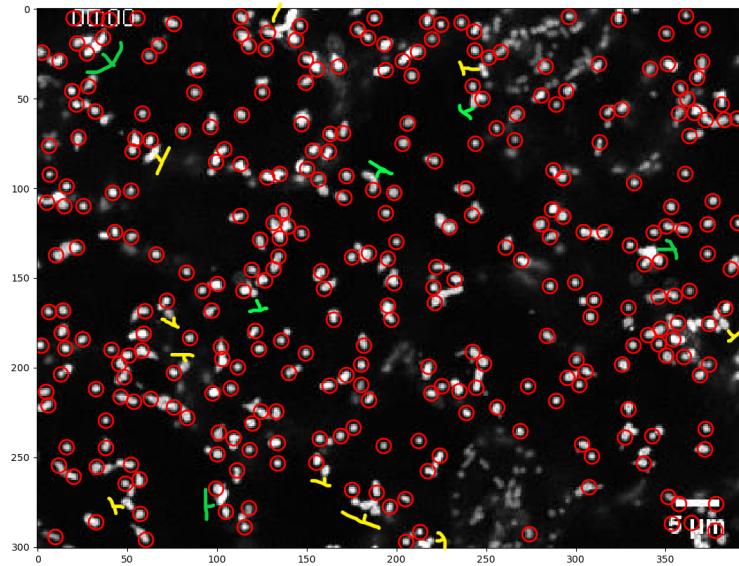


Abbildung 6.15: `locate()`-Funktion auf 0. Frame mit 'separation=7.0'

Mit den aufeinanderfolgenden Werten **6.8**, **6.7**, **6.5 und 6.4** kommt es neben einigen Verbesserungen immer zu mehreren Verschlechterungen, die später mit anderen Parametern nur schwer zu korrigieren sind. Genau aus diesem Grund wird hier als **separation** der Wert **6.3** betrachtet. Wobei fast lediglich nur Verbesserungen zu notieren sind.

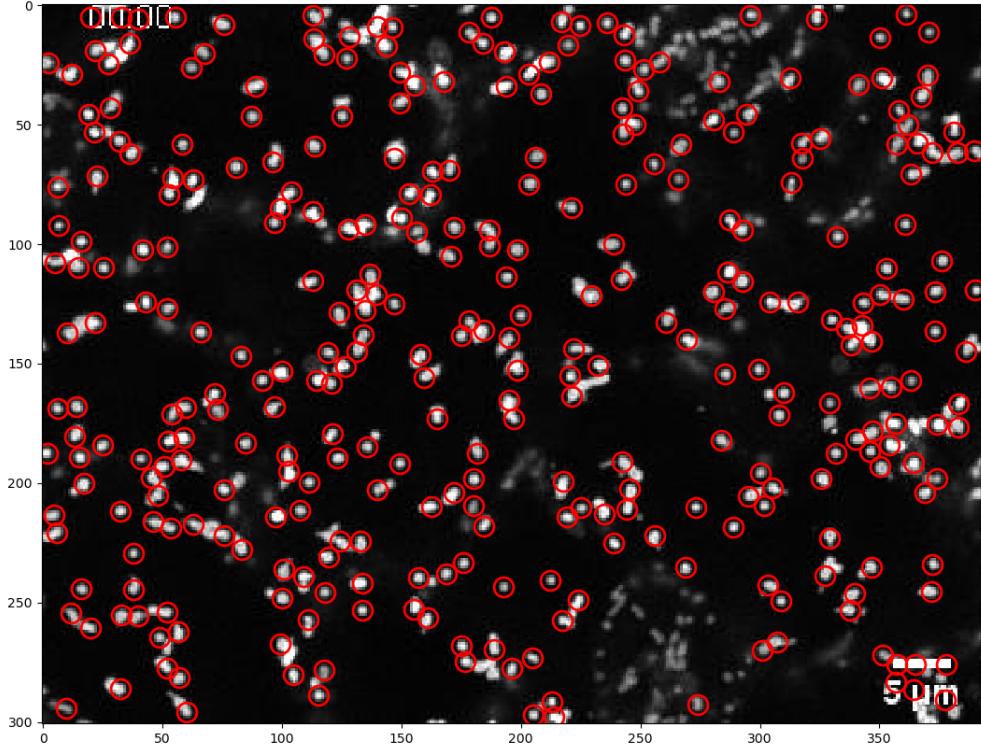


Abbildung 6.16: `locate()`-Funktion auf 0. Frame mit 'separation=6.3'

Es scheint hier wichtig zu erwähnen, dass es kein Parameter gibt, dessen Wert adäquat für alle Arten von Bildern oder Partikeln funktioniert. So sollte immer eine Parametrisierung als erster Schritt durchgeführt werden. Allerdings scheint im konkreten Fall dieses Bildes (nulltes Frames) die Parameterwerte, die am besten geeignet erscheinen, wie folgt:

```
locate(frames[0], 5, minmass=210, separation=6.3)
```

Diese Werte können jedoch nicht unverändert bleiben, da jedes neue Bild seine eigenen Eigenschaften hat und die Verwendung von Werten, die im Bild *Frame* hervorragende Ergebnisse erzielen, im Bild *Frame+1* zu chaotischen Ergebnissen führen kann.

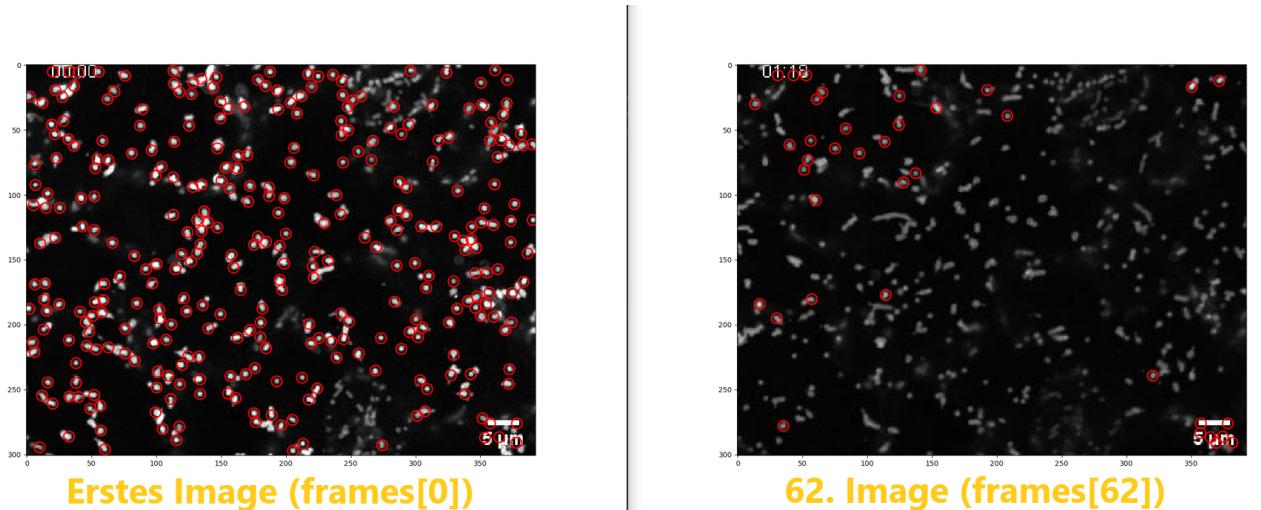


Abbildung 6.17: Vergleich von Bild 1 und Bild 62 mit denselben Parameterwerten.

Tatsächlich wurden auf dem rechten Bild ungefähr 38 Partikel erkannt, während es auf dem linken Bild 335 Partikel waren. Das macht einen Unterschied von 297 Partikeln, obwohl die gleichen Parameter und Werte verwendet wurden.

Die Sicherstellung, dass diese Art von Partikeln, die zwischen den Bildern gefunden werden, nicht mehr reproduziert werden, wird Gegenstand unserer Beschäftigung im nächsten Gliederungspunkt sein.

## 6.2 Automatische Parameter-Optimierung

Zur Lösung des oben genannten Problems haben wir im Rahmen dieser Arbeit eine Methode namens `get_particles_per_image_as_array` implementiert, die auf alle Bilder des Videos angewendet wird (ein Bild nach dem anderen) und die nach der Bestimmung eines Intervalls sicherstellt, dass die Anzahl der detektierten Partikel in diesem Intervall verbleibt.

Das folgende Struktogramm zeigt, wie die Methode sicherstellt, dass die Anzahl der gefundenen Partikel immer innerhalb des zuvor definierten Intervalls bleibt.

An dieser Stelle sei erwähnt, dass das Ergebnis und die Parameter, die zu diesem Ergebnis geführt haben, in einer Variablen mit dem Namen `particle_per_frame` gespeichert werden. Diese Variable wird auch der Rückgabewert der Funktion sein.

Außerdem ist der einzige Parameter, der bei der automatischen Optimierung des Ergebnisses berücksichtigt und geändert wird, der "minmass" Parameter. Dieser wird verwendet, weil wir im Laufe unserer Versuche festgestellt haben, dass ein Großteil der Ergebnisse nur mit diesem Wert erzielt werden kann.

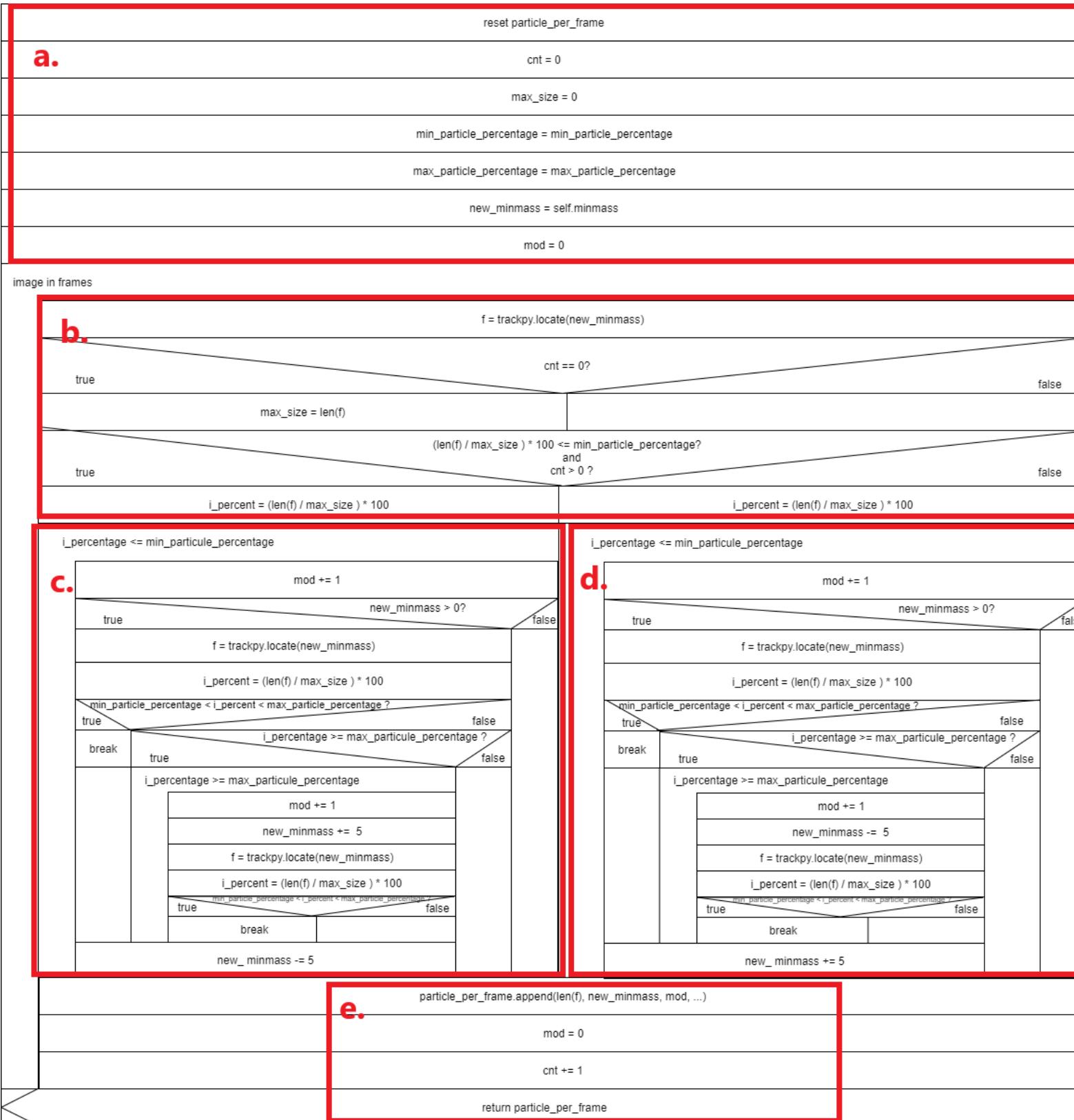


Abbildung 6.18: Struktogramm der Methode.: `get_particles_per_image_as_array()`

---

```
reset particle_per_frame
```

---



---

```
cnt = 0
```

---



---

```
max_size = 0
```

---



---

```
min_particle_percentage = min_particle_percentage
```

---



---

```
max_particle_percentage = max_particle_percentage
```

---



---

```
new_minmass = self.minmass
```

---



---

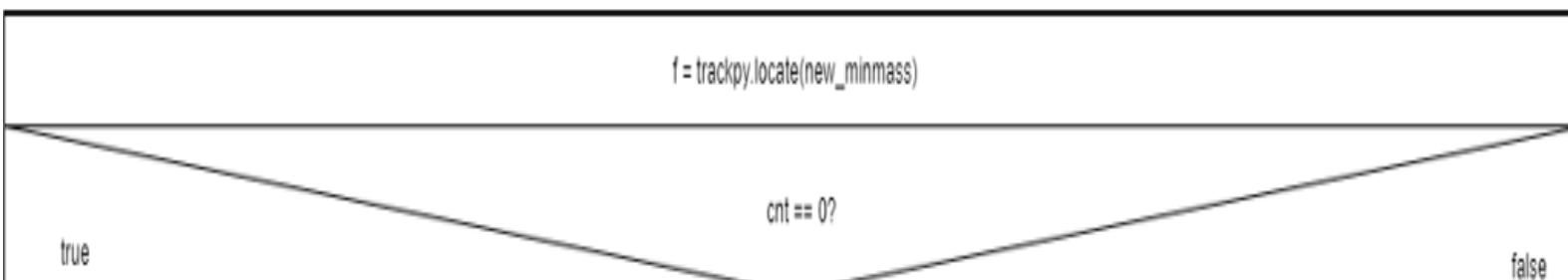
```
mod = 0
```

**a.**

---

```
f = trackpy.locate(new_minmass)
```

---



```
cnt == 0?
```

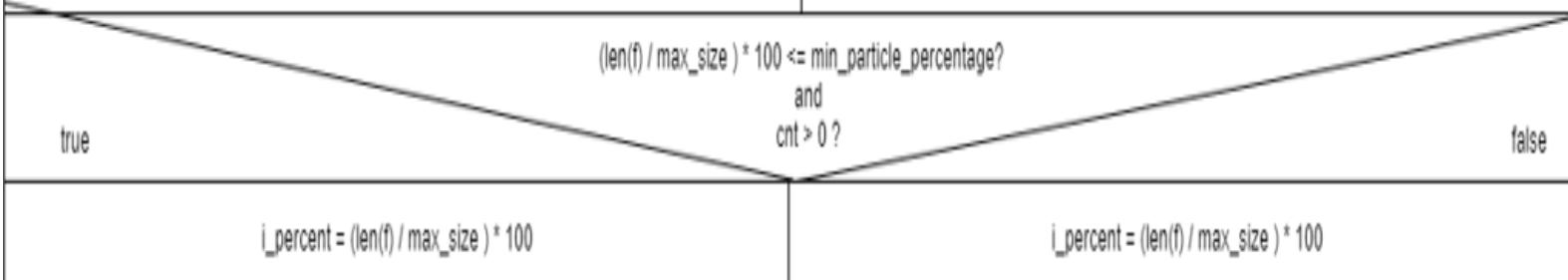
true

false

---

```
max_size = len(f)
```

---



```
(len(f) / max_size ) * 100 <= min_particle_percentage?
```

and

cnt > 0 ?

true

false

---

```
i_percent = (len(f) / max_size) * 100
```

---



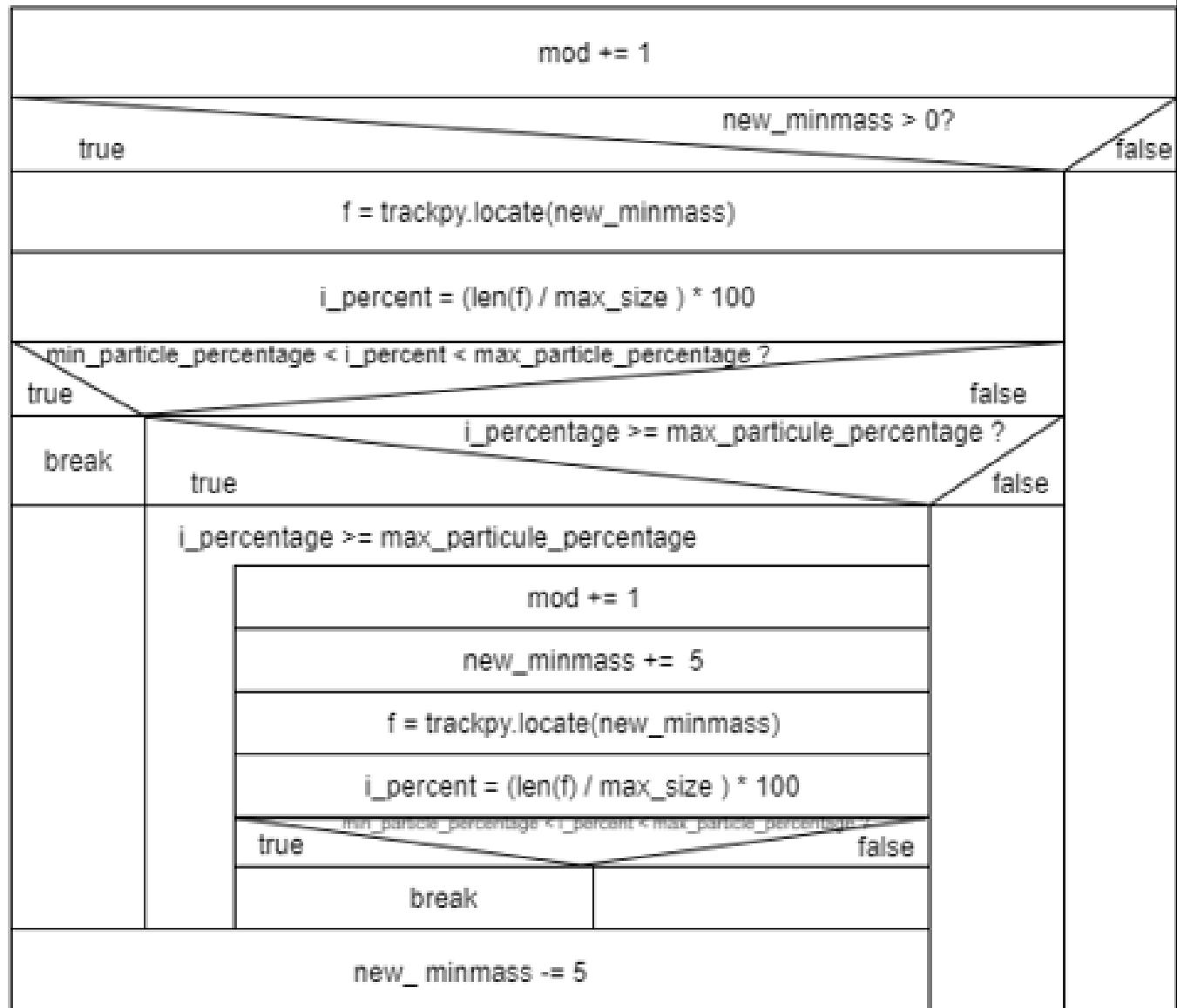
---

```
i_percent = (len(f) / max_size) * 100
```

---

**b.**

i\_percentage <= min\_particule\_percentage



c.

```
i_percentage <= min_particule_percentage
```

```
mod += 1
```

```
true
```

```
new_minmass > 0?
```

```
false
```

```
f = trackpy.locate(new_minmass)
```

```
i_percent = (len(f) / max_size ) * 100
```

```
min_particle_percentage < i_percent < max_particle_percentage ?
```

```
true
```

```
false
```

```
break
```

```
true
```

```
i_percentage >= max_particule_percentage ?
```

```
false
```

```
i_percentage >= max_particule_percentage
```

```
mod += 1
```

```
new_minmass -= 5
```

```
f = trackpy.locate(new_minmass)
```

```
i_percent = (len(f) / max_size ) * 100
```

```
min_particle_percentage < i_percent < max_particle_percentage ?
```

```
true
```

```
false
```

```
break
```

```
new_minmass += 5
```

d.

---

```
particle_per_frame.append(len(f), new_minmass, mod, ...)
```

---

```
mod = 0
```

---

```
cnt += 1
```

---

```
return particle_per_frame
```

---

**e.**

Das Intervall sollte in Prozent angegeben werden, wobei der Standardwert *80%* für den unteren Grenzwert und *110%* für den oberen Grenzwert ist. Die 100% werden durch die Anzahl der im allerersten Bild erkannten Partikel bestimmt. Es wird natürlich davon ausgegangen, dass ein erster Schritt, der manuelle und systematische Suche nach geeigneten Parametern für dieses erste Bild, bereits durchgeführt wurde (*siehe Abschnitt 6.1.2*).

Nach der Ausführung der Methode mit den Standardprozentwerten für die untere und obere Grenze des Intervalls erhalten wir schließlich das folgende Ergebnis. Dieses Ergebnis wird mit demselben Bild verdeutlicht, bei dem keine automatische Optimierung der Parameters durchgeführt wurde.

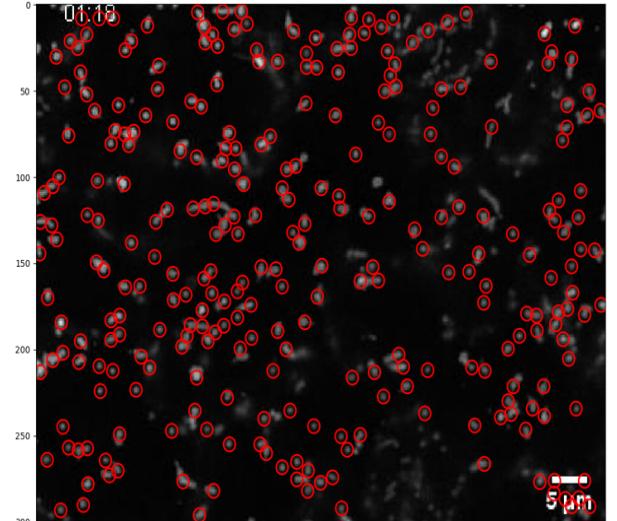
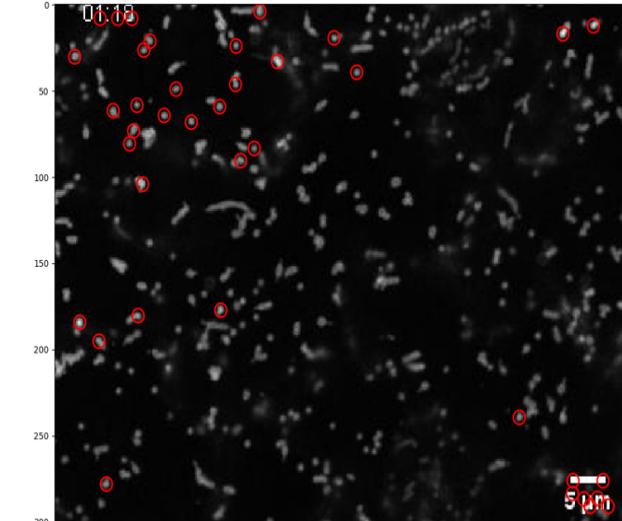


Abbildung 6.19: Vergleich von Bild 62 ohne und mit automatischer Optimierung der Parameter.

Es ist jedoch möglich, verschiedene Werte für bestimmte Parameter zu testen, und zwar für ein bestimmtes Bild mit Hilfe einer anderen Funktion **update\_frame**.

### 6.3 Bearbeitung eines einzelnen Bildes

Die automatische Optimierung hilft zwar dabei, eine gewisse Anzahl von Partikeln in den Ergebnissen zu behalten, aber es kann vorkommen, dass man bei einem bestimmten Bild andere Werte für die Parameter anwenden möchte, ohne die Gesamtheit der enthaltenen Bilder zu verändern. Wir haben daher eine Methode namens **update\_frame** entwickelt, mit der wir die gewünschten Änderungen an einem einzelnen Bild vornehmen können. Sie kann insgesamt 7 Parameter aufnehmen, von denen 2 erforderlich und die anderen 5 optional sind. Die obligatorischen Parameter sind: *frames* und *f\_no*(Die Nummer des zu bearbeitenden Bildes) Die optionalen sind *minmass*, *separation*, *maxsize*, *topn* und *engine*.

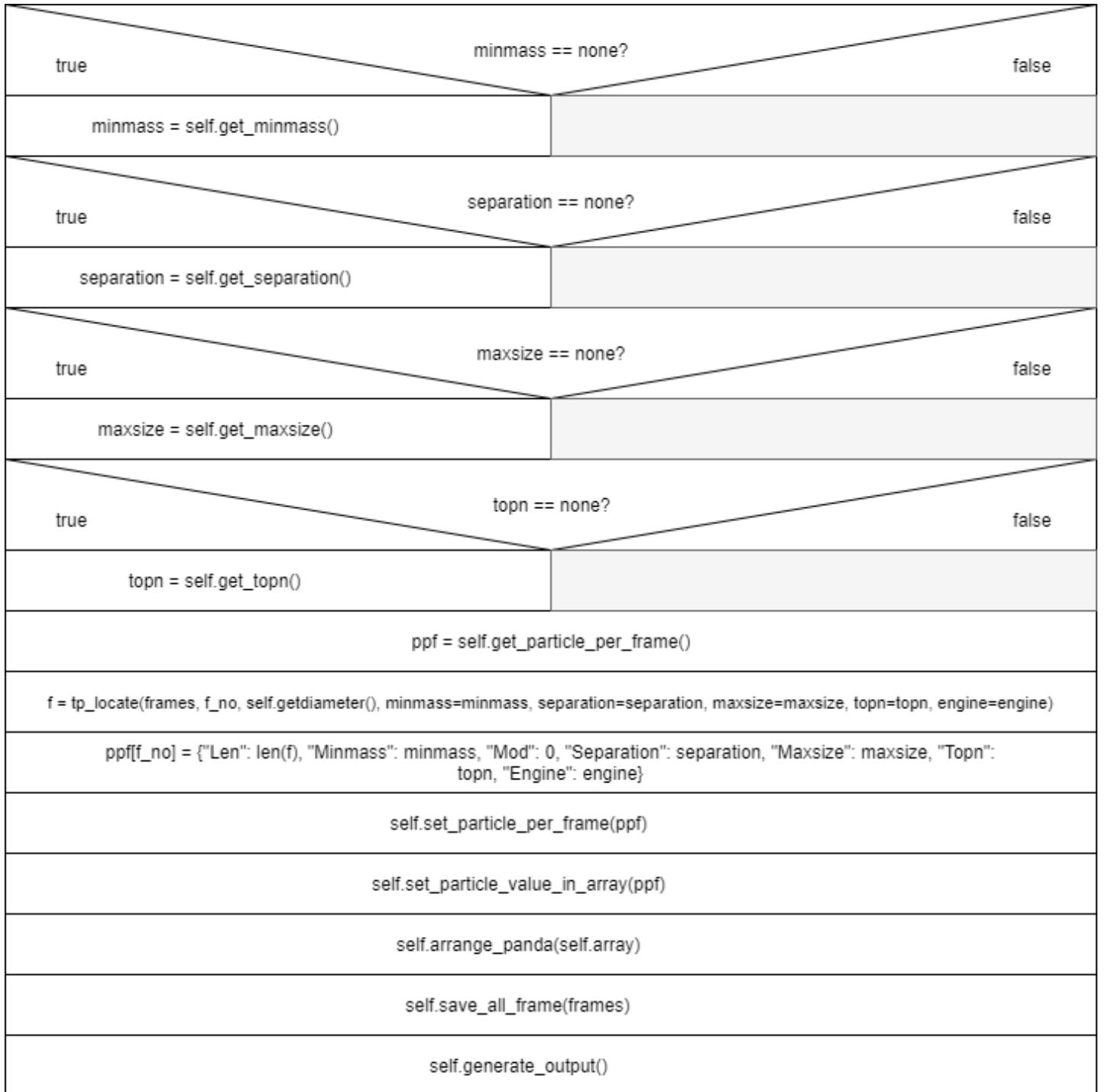


Abbildung 6.20: Struktogramm der Methode: `update_frame()`

Der obige Struktogramm zeigt, welche Schritte in welcher Reihenfolge ausgeführt werden müssen, um eine Änderung an einem einzelnen Frame vorzunehmen.

So führt die Ausführung des folgenden Codes:

**update\_frame(frames, 66, minmass=170, maxsize=100, topn=150)**

Dies bewirkt, dass zuerst die minimale eingebaute Helligkeit auf 170 gesetzt wird, dann der maximale Radius der Helligkeit auf 100 gesetzt wird (so dass keiner der gefundenen Partikel über diesem Wert liegt) und schließlich nur die 150 hellsten Partikel angezeigt werden, die die vorgenannten Bedingungen erfüllen und dies ausschließlich in Frame 66 unseres Videos.

# 7 Auswertung

Dieser Teil widmet sich nicht nur dem Starten der Anwendung(Partikel-Erkennungssystem) sondern auch der Auswertung der Ergebnisse bzw. Bilder mit lokalisierten Partikeln aus der Verwendung des Partikel-Erkennungssystems, um die Beurteilung des Renderings (Ergebnisse) zu erleichtern. In diesem Sinne haben wir im Rahmen unserer Arbeit die Python-Bibliothek **bokeh** verwendet. Diese ermöglicht es, interaktive Visualisierungen für moderne Webbrowser zu erstellen und "hilft dabei, wunderschöne Grafiken zu erstellen, von einfachen Plots bis hin zu komplexen Dashboards mit kontinuierlichen Datensätzen", wie es auf ihrer Website beschrieben wurde [5].

## 7.1 App

Nachdem wir in den vorherigen Abschnitten gesehen haben, wie man die Parameterwerte für das allererste Bild auswählt und die wichtigsten Methoden der Anwendung wie *update\_frame* und *get\_particle\_per\_frame* kennengelernt haben, ist es uns wichtig zu zeigen, wie man die Anwendung startet und wo man was einsetzt, um sie zu benutzen. Entsprechend werden wir uns im nächsten Abschnitt mit der Installation des Systems befassen.

### 7.1.1 Installation

Zuerst müssen Sie sich das Paket besorgen. Sie können das Projekt entweder aus dem github-Repository [?] klonen oder es einfach downloaden. Sobald das Repository lokal auf Ihrem Computer verfügbar ist, ist der nächste Schritt, sich eine IDE zu besorgen. Es ist zwar möglich, das Projekt über das Terminal zu starten, aber wir empfehlen dringend die Verwendung einer IDE, vorzugsweise PyCharm.

Danach müssen Sie nur noch das Repository mit dem von Ihnen gewählten Editor öffnen. Da die Installation des Systems größtenteils auf der Installation von Trackpy basiert, die im Abschnitt 5.1 durchgeführt wurde, müssen Sie zuerst die

Schritte zur Installation von Trackpy befolgen und anschließend unbedingt auch "bokeh installieren. Wie auf ihrer Seite beschrieben, können wir je nach Arbeitsumgebung entweder **conda install bokeh** für eine conda-Umgebung oder **pip install bokeh** verwenden (*siehe [5]*). Nach der Installation ist das System betriebsbereit und einsatzbereit.

### 7.1.2 Bedienung der Anwendung

#### 1. Backendseitig

Nach der erfolgreichen Installation müssen Sie in die Datei **ParticleTrackingSystem/launcher.py** gehen, die hauptsächlich zur Ausführung der Funktionen dient, die hier zur Verfügung gestellt werden. Die Funktion "set\_path", die im Programm beschrieben wird, ermöglicht es, den Ort der Datei anzugeben, genauer gesagt den Ort des Videos, das wir auf die Bewegung der vorhandenen Partikel untersuchen wollen. Mit Hilfe der Variable "tracker", die nichts anderes als ein Objekt vom Typ **Tracker** ist, müssen wir ihr jeweils die Elemente zuweisen, die sie wissen muss:

- Den Durchmesser der Partikel über den Konstruktor
- Die zu manipulierende Bildsequenz (wird automatisch erstellt).
- Die Minmasse der Partikel.
- Schließlich die Trennung

Alle Werte sollten aus dem ersten Bild oder einem Modellbild gemäß 6.1.2 ausgewählt werden.

Anschließend kann man den maximalen und minimalen Prozentsatz der zu findenden Partikel festlegen.

Wenn Sie all dies getan haben, müssen Sie nur noch die Datei **launcher.py** ausführen und einige Sekunden warten, bis das Programm ausgeführt wird. Dies ist alles für die Ausführung des Programms. Jetzt müssen wir nur noch die Visualisierung (Frontend) über Bokeh starten.

#### 2. Frontendseitig

Das Frontend lässt sich starten, indem man ein Terminal öffnet, in das **ParticleTrackingSystem** wechselt und dort den folgenden Befehl ausführt:

**python -m bokeh serve --show .\ParticleTrackingSystem\.**

Es erscheint eine Seite mit einem Button, der eine Diashow der Bilder mit

den lokalisierten Partikeln und den Werten der angewandten Parameter, die zu dem zu sehenden Ergebnis geführt haben, startet.

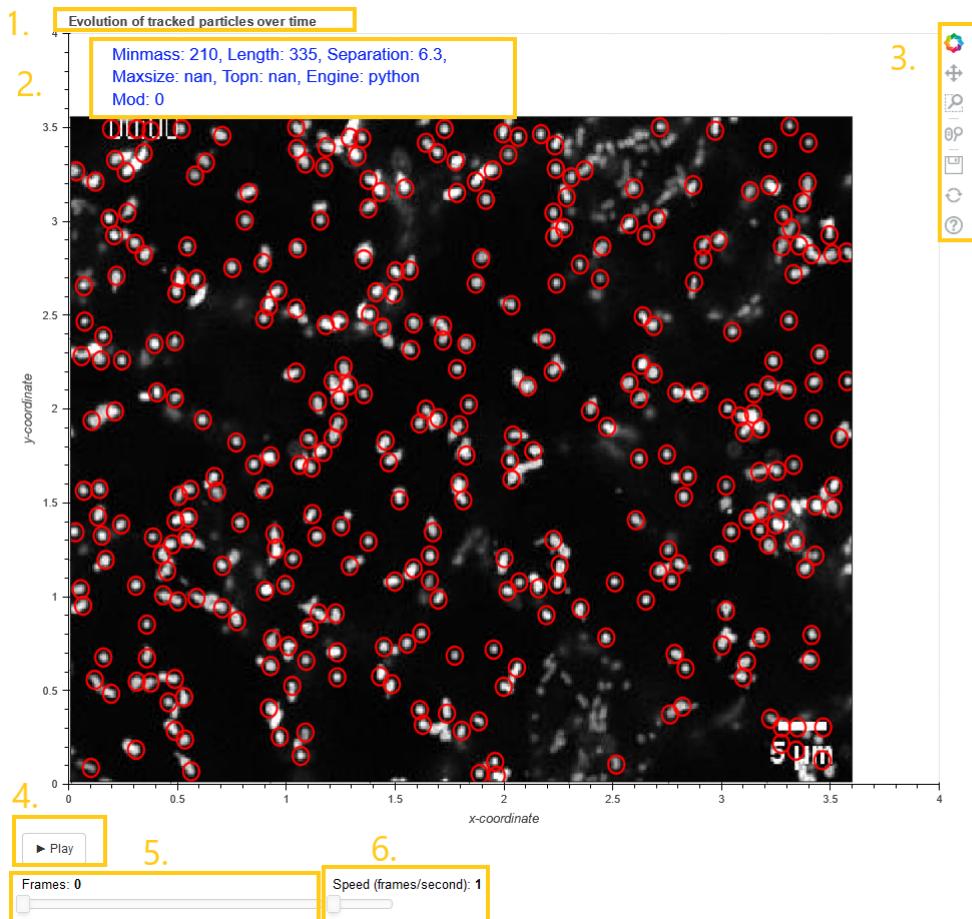


Abbildung 7.1: Frontend presentation

1.	Titel der Seite (bleibt unverändert).
2.	Parameterwerte, die zum Ergebnis geführt haben (+ length).
3.	Von Bokeh bereitgestellte Tools (Zoomen, Verschieben, Speichern und Neuladen, ...).
4.	Play- und Pause-Button zum Starten und Pausieren der Diashow.
5.	Zeigt den Frame an, in dem man sich gerade befindet, und ermöglicht es, zu einem anderen Frame zu wechseln.
6.	Reguliert die Reihenfolge der Bilder (z. B. Bild nach Bild oder jedes zweite Bild).

Sollte das Ergebnis (eines bestimmten Bildes) nicht zufriedenstellend sein,

kann es mit der Funktion *update\_frame*, wie im vorherigen Kapitel beschrieben, geändert werden. Allerdings muss der Bokeh-Server nach der Änderung neu gestartet werden, um sicherzustellen, dass das neue Bild und seine Parameter angezeigt werden können.

# 8 Fazit

Das Ziel unserer Arbeit war es, ein Werkzeug (Software) zu entwickeln, mit dem man in einem bestimmten Video Bild für Bild die vorhandenen Partikel erkennen kann. Anschließend sollte die Anzahl der gefundenen Partikel automatisch konstant gehalten werden (z.B. um zu verhindern, dass durch Lichtveränderungen im Video zu viele Partikel entdeckt werden).

Danach sollte eine Benutzeroberfläche entwickelt werden, die es ermöglicht, die Ergebnisse aus den vorherigen Punkten optimal zu bewerten und zu beurteilen. Schließlich sollte auch die Möglichkeit bestehen, die Werte der Parameter, die zu den in den ersten Punkten erzielten Ergebnissen geführt haben, anzupassen, ohne die bereits validierten Ergebnisse zu verändern.

Im Kapitel 4 haben wir zunächst eine vergleichende Studie verschiedener Werkzeuge durchgeführt, die die Möglichkeit bieten, das erste Ziel (Partikeldetektion) zu erreichen. Dies war **trackpy**. Im Kapitel 5 stellten wir trackpy vor und berichteten über unsere Erfahrungen als Nutzer. Erst in Kapitel 6 machten wir praktische Fortschritte, indem wir die zu verwendenden Parameter (siehe 6.1.1) vorstellten und einen systematischen Weg aufzeigten, wie man angemessene Werte für das erste Bild(Frame) finden kann (siehe 6.1.2).

Ebenfalls in diesem Kapitel, im Teil 6.1.2, wird erklärt, wie die Lösung, die wir gefunden haben, um die Anzahl der Partikel in einem Intervall zu halten, funktioniert. Der Abschnitt 6.3 seinerseits behandelt die Frage der Änderung von Parameterwerten und deren Anwendung in unserem Kontext.

Schließlich zeigen wir im Kapitel 7 den Installationsprozess sowohl des Frontends als auch des Backends unseres Tools (Software). Wir haben auch die von uns entworfene Benutzeroberfläche vorgestellt, die unserer Meinung nach optimal für die Visualisierung und Bewertung der Parameterwerte ist, die zur Erzielung von Ergebnissen verwendet werden.

## 9 Ausblick

Aus den Ergebnissen und dem derzeitigen Zustand unseres Tools, das uns nur erlaubt, Partikel zu erkennen, die Anzahl der Partikel im Laufe des Videos automatisch zu stabilisieren und die Ergebnisse angemessen anzuzeigen, ohne dabei die Änderung eines bestimmten Bildparameters zu vergessen, wird klar, dass es noch viele Dinge gibt, die wir verbessern und hinzufügen können. Leider hatten wir im Rahmen dieser Bachelorarbeit nicht die Zeit, dies zu tun. Dem Gedankengang folge sehen wir die möglichen Verbesserungen des Tools wie folgt:

- Die Einfügung einer Identifikation für jeden Partikel. So dass jedes Partikel einen ID-Code hat, der im Laufe der Zeit im Video nachverfolgt werden kann.
- Die Bewegungsverläufe der einzelnen Partikel mithilfe dieser ID durch das gesamte Video verfolgen
- Erkennung von verschiedenen Ereignissen, die im Video stattfinden, ebenfalls anhand dieser ID. Genauer gesagt, Ereignisse der Spaltung (wenn sich ein Partikel in zwei oder mehr Partikel aufspaltet) und der Verschmelzung (wenn zwei oder mehr Partikel zusammenkommen, um ein einziges zu bilden) von Partikeln.
- Damit die Ergebnisse eine höhere Genauigkeit haben, wäre es interessant, sich mit dem 3D-Aspekt der Erkennung zu beschäftigen. Neben der horizontalen und vertikalen Position, die durch x bzw. y angegeben wird, gibt es noch eine dritte Position, die die z-Richtung bestimmt. Dies würde die Zuverlässigkeit und Genauigkeit der Ergebnisse signifikant erhöhen.

# Abbildungsverzeichnis

3.1	Vergleich zwischen Bild mit 72dpi und eins mit 30dpi. . . . .	11
3.2	Beispiel von ‘maxima detection’ . . . . .	13
4.1	Rohbild mit zu erkennenden Partikeln. . . . .	16
4.2	Unzureichend beschriebene Optionsnamen . . . . .	20
4.3	Beispiel einer Partikelerkennung mit der Trackpy-Methode von ParticleTracker. . . . .	21
4.4	Beispiel von Partikelerkennung mit Trackpy. . . . .	29
5.1	Vergleich zwischen max_iterations=1 und max_iterations=1000 auf demselben Frame. . . . .	32
6.1	locate()-Funktion auf 0. Frame mit diameter=3 . . . . .	36
6.2	locate()-Funktion auf 0. Frame mit diameter=9 . . . . .	37
6.3	locate()-Funktion auf 0. Frame mit diameter=7 . . . . .	38
6.4	locate()-Funktion auf 0. Frame mit diameter=5 . . . . .	38
6.5	locate()-Funktion auf 0. Frame mit diameter=5 . . . . .	39
6.6	Ein Teil des initialen Dataframes . . . . .	41
6.7	Ein Teil des sortierten Dataframes . . . . .	42
6.8	locate()-Funktion auf 0. Frame mit ‘mimass=189.72805’ . . . . .	43
6.9	Sortierte Tabelle . . . . .	44
6.10	locate()-Funktion auf 0. Frame mit ‘mimass=197’ . . . . .	45
6.11	locate()-Funktion auf 0. Frame mit ‘mimass=204’ . . . . .	46
6.12	locate()-Funktion auf 0. Frame mit ‘mimass=210’ . . . . .	47
6.13	locate()-Funktion auf 0. Frame mit ‘mimass=212’ . . . . .	48
6.14	locate()-Funktion auf 0. Frame mit ‘separation=6.0’ . . . . .	49
6.15	locate()-Funktion auf 0. Frame mit ‘separation=7.0’ . . . . .	50
6.16	locate()-Funktion auf 0. Frame mit ‘separation=6.3’ . . . . .	51
6.17	Vergleich von Bild 1 und Bild 62 mit denselben Parameterwerten. .	52
6.18	Struktogramm der Methode.: get_particles_per_image_as_array() .	54
6.19	Vergleich von Bild 62 ohne und mit automatischer Optimierung der Parameter. . . . .	59
6.20	Struktogramm der Methode: update_frame() . . . . .	60

7.1 Frontend presentation . . . . .	64
-------------------------------------	----

# Literaturverzeichnis

- [1] Daniel B. Allan, Thomas Caswell, Nathan C. Keim, Casper M. van der Wel, and Thomas Dimiduk. Soft-matter/pims: Python image sequence: Load video and sequential images in many formats with a simple, consistent interface., 2021.
- [2] Daniel B. Allan, Thomas Caswell, Nathan C. Keim, Casper M. van der Wel, and Ruben W. Verweij. Installing trackpy, 2021.
- [3] Daniel B. Allan, Thomas Caswell, Nathan C. Keim, Casper M. van der Wel, and Ruben W. Verweij. Parameter of the locate funktion. <http://soft-matter.github.io/trackpy/v0.5.0/generated/trackpy.locate.html#rd8297186acd3-1>, May 2021.
- [4] Daniel B. Allan, Thomas Caswell, Nathan C. Keim, Casper M. van der Wel, and Ruben W. Verweij. soft-matter/trackpy: Trackpy v0.5.0, April 2021.
- [5] Bokeh Development Team. *Bokeh: Python library for interactive visualization*, 2018.
- [6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [7] Nicolas Chenouard, Ihor Smal, Fabrice De Chaumont, Martin Maška, Ivo F Sbalzarini, Yuanhao Gong, Janick Cardinale, Craig Carthel, Stefano Coraluppi, Mark Winter, et al. Objective comparison of particle tracking methods. *Nature methods*, 11(3):281–289, 2014.
- [8] Mech Content. Radius of gyration: Definition, equation, derivation, units, explained, Apr 2022.
- [9] 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, 1996.
- [10] Rajesh Dachiraju. A function fitting method. *Journal of Applied Analysis*, 26(1):59–65, 2020.
- [11] Akemi Gálvez and Andrés Iglesias. Efficient particle swarm optimization approach for data fitting with free knot b-splines. *Computer-Aided Design*, 43(12):1683–1692, 2011.

- [12] Golara Garousi, Vahid Garousi, Mahmoud Moussavi, Guenther Ruhe, and Brian Smith. Evaluating usage and quality of technical software documentation: an empirical study. In *Proceedings of the 17th international conference on evaluation and assessment in software engineering*, pages 24–35, 2013.
- [13] Morten Hertzum and Erik Frøkjær. Browsing and querying in online documentation: A study of user interfaces and the interaction process. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(2):136–161, 1996.
- [14] Info@noirlab.edu. Technical document: Introduction to image processing, Mar 2021.
- [15] tuto java. Difference of gaussians edge enhancement algorithm, 2022.
- [16] Dehao Ju, John S Shrimpton, and Alex Hearn. A multi-thresholding algorithm for sizing out of focus particles. *Particle & Particle Systems Characterization*, 29(2):78–92, 2012.
- [17] team Jungmannlab. Jungmannlab/picasso: A collection of tools for painting super-resolution images, 2019.
- [18] Mykel J Kochenderfer and Tim A Wheeler. *Algorithms for optimization*. Mit Press, 2019.
- [19] Hui Kong, Hatice Cinar Akakin, and Sanjay E Sarma. A generalized laplacian of gaussian filter for blob detection and its applications. *IEEE transactions on cybernetics*, 43(6):1719–1733, 2013.
- [20] Leonardo. What is the difference between an image and a video frame?, 2019.
- [21] Enea Poletti, Francesca Zappelli, Alfredo Ruggeri, and Enrico Grisan. A review of thresholding strategies applied to human chromosome segmentation. *Computer methods and programs in biomedicine*, 108(2):679–688, 2012.
- [22] Sylvain Prigent. Sylvainprigent/stracking: Python library for particles tracking in 2d+t and 3d+t scientific images, 2020.
- [23] Sylvain Prigent. Sylvainprigent/napari-stracking: Napari plugin for particles tracking, 2021.
- [24] team Schwille-Paint. Schwille-paint/picasso\_addon: Extensions to picasso python package., 2020.
- [25] Jan Sellner. Milania’s blog, 2017.
- [26] Mike I. Smith and James G. Downs. Particletracker: a gui based particle tracking software. *Journal of Open Source Software*, 6(66):3611, 2021.

- [27] Vera Soboleva and Oleg Shipitko. Raindrops on windshield: Dataset and light-weight gradient-based detection algorithm. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2021.
- [28] Florian Stehr and Johannes Stein. Schwillen-paint/spt: Complete single particle tracking analysis workflow., 2020.
- [29] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.
- [30] Techopedia. What is a pixel? - definition from techopedia, Aug 2020.
- [31] Marcio Michiharu Tsukamoto, Liang-Yee Cheng, and Fabio Kenji Motezuki. Fluid interface detection technique based on neighborhood particles centroid deviation (npcd) for particle methods. *International Journal for Numerical Methods in Fluids*, 82(3):148–168, 2016.
- [32] Virendra Kumar Yadav, Saumya Batham, Anuja Kumar Acharya, and Rahul Paul. Approach to accurate circle detection: Circular hough transform and local maxima concept. In *2014 International Conference on Electronics and Communication Systems (ICECS)*, pages 1–5. IEEE, 2014.
- [33] HK Yuen, John Princen, John Illingworth, and Josef Kittler. Comparative study of hough transform methods for circle finding. *Image and vision computing*, 8(1):71–77, 1990.

# Anhang

## *Inhalt der beigefügten USB-Stick*

Software (Partikel-Erkenung-System) .....	62
Vergleich zwischen Bild mit 72dpi und eins mit 30dpi .....	11
Rohbild mit zu erkennenden Partikeln .....	16
Unzureichend beschriebene Optionsnamen .....	20
Beispiel einer Partikelerkennung mit der Trackpy-Methode von ParticleTracker .....	21
Beispiel von Partikelerkennung mit Trackpy .....	29
Vergleich zwischen max iterations=1 und max iterations=1000 auf demselben Frame .....	32
locate()-Funktion auf 0. Frame mit diameter=3 .....	36
locate()-Funktion auf 0. Frame mit diameter=9 .....	37
locate()-Funktion auf 0. Frame mit diameter=7 .....	38
locate()-Funktion auf 0. Frame mit diameter=5 .....	39
Ein Teil des initialen Dataframes .....	41
Ein Teil des sortierten Dataframes .....	42
locate()-Funktion auf 0. Frame mit 'mimass=189.72805' .....	43
Sortierte Tabelle .....	44
locate()-Funktion auf 0. Frame mit 'mimass=197' .....	45
locate()-Funktion auf 0. Frame mit 'mimass=204' .....	46
locate()-Funktion auf 0. Frame mit 'mimass=210' .....	47
locate()-Funktion auf 0. Frame mit 'mimass=212' .....	48
locate()-Funktion auf 0. Frame mit 'separation=6.0' .....	49
locate()-Funktion auf 0. Frame mit 'separation=7.0' .....	50
locate()-Funktion auf 0. Frame mit 'separation=6.3' .....	51
Vergleich von Bild 1 und Bild 62 mit denselben Parameterwerten. ....	52
Struktogramm der Methode.: get particles per image as array() .....	54
Vergleich von Bild 62 ohne und mit automatischer Optimierung der Parameter .....	59
Struktogramm der Methode: update frame() .....	60
Frontend presentation .....	64