Projekt 1 Hochschule Darmstadt

Particle Tracking

Prof. Dr. Stephan Neser, Sergey Belyaev,
Sebastian Gramlich, Thomas Huxhorn,
Severin Kraft, Fabian Römer

Darmstadt, Sommersemester 2010

Contents

1 Kurzzusammenfassung (Gramlich)

Grundlage des Projektes ist die Detektion von Partikeln mit Hilfe von Floureszenzmikroskopie und Auswertung der Daten mit Methoden der Bildverarbeitung. Innerhalb der Projektgruppe wurden zwei verschiedene Teilbereiche abgedeckt. Zum einen die Verbesserung des Bildaufnahmeaufbaus durch bessere Beleuchtung, zum anderen die programmiertechnische Umsetzung einer Routine zum Vergleich verschiedener Lokalisierungsalgorithmen und anschließender Verfolgung der Trajektorien.

Anstelle der bisher verwendeten Quecksilberdampflampe wurde die Beleuchtung auf eine gepulste LED umgestellt, was in mehrerlei Hinsicht deutliche Vorteile bringt. Für die Verarbeitung der Daten wurde ein Programm in MATLAB erstellt, das in der Lage ist, Sequenzen von Bildern einzulesen, verschiedene Methoden zur Bestimmung von Schwerpunkten der Partikel durchzuführen und die Ergebnisse als Datei zu speichern. Die Verfolgung der Trajektorien nutzt die ermittelten Schwerpunkte mit Hilfe von Milan und analysiert unter Beachtung der sachlichen Probleme die Bewegung der Partikel.

2 Rahmenprogramm (Kraft)

2.1 Allgemeines

Das Rahmenprogramm, oder auch Main genannt, analysiert die ausgewählte Sequenz, d.h. die Anzahl der Bilder und den Pfad wo die Bilder abgespeichert sind. Es werden die Bilder einzeln und mit den eingegebenen Parametern und Einstellungen an die entsprechenden Funktionen übergeben und als Rückgabewert in einer .txt Datei abgelegt. Die Main erkennt selbstständig ob eine Sequenz fertig ist, oder erkennt wenn ein Bild in der Sequenz fehlt. Sollte ein Bild fehlen, beendet das Programm ganz normal und die Werte werden bis dahin abgespeichert.

2.2 Zerlegen des Dateipfads

Nach Auswahl der Sequenz über ein Userinterface wird der Pfad des ersten Bildes der ausgesuchten Sequenz mit Hilfe der Matlabfunktion *fileparts* in Pfadname, Dateiname und Dateiendung zerlegt.

Bsp.: [pathstr, name, ext]=fileparts([PathName,FileName])

Pfad (PathName, FileName):	`E:\testbilder\gauss_r15_noise0.1_nodepth000001.tif`
Pfadname (pathstr):	`E:\testbilder`
Dateiname (name):	`gauss_r15_noise0.1_nodepth000001`
Dateiendung (ext):	.tif

2.3 Parameter für die Vorverarbeitung

Über eine einfache Eingabe und Auswahl werden die Parameter und Einstellungen für die Vorverarbeitung festgelegt. Diese sind die minimale und maximale Fläche eines Partikels, ob über eine 4er oder 8er Nachbarschaft die Partikel gesucht werden sollen und ob die Binarisierung über eine feste Schwelle, über die Matlabfunktion *graytresh* oder mit 20% des höchsten Grauwertes durchgeführt werden soll.

2.4 Zerlegen des Dateinamens

Da beim späteren Hochzählen der Bildnummern der Stammname nicht von Interesse ist, muss der Dateiname in den Stammnamen und in die Bildnummer zerlegt und beides separat abgespeichert werden. Dies wird durch einen einfachen Algorithmus realisiert, der den Dateinamen von hinten Schritt für Schritt durchsucht, bis er keine Zahl mehr findet.

Bsp.:

Dateiname: `gauss_r15_noise0.1_nodepth000001` <—-
1.Durchlauf: `gauss_r15_noise0.1_nodepth00000`
2.Durchlauf: `gauss_r15_noise0.1_nodepth0000`
.
.
.
6.Durchlauf: `gauss_r15_noise0.1_nodepth`
.
Stammname: `gauss_r15_noise0.1_nodepth`
Bildnummer: 000001

2.5 Die Bildnummer

Um die Bildnummer hoch zählen zu können, da diese aus einem String besteht und man Zahlen, die vom Typ ein Charakter sind, nicht erhöhen kann, wird die Bildnummer über die Funktion str2num in einen Integer umgewandelt. Durch diese Funktion werden aber die führenden Nullen abgeschnitten.

Bsp:

Bildnummer (string): 000001

Bildnummer (int): 1

Damit aber später der ganzen Pfad wieder rekonstruiert werden kann, muss man sich die Anzahl der führenden Nullen merken, damit sie später im Programm wieder eingefügt werden können.

2.6 Funktionen

Hier wird jedes einzelne Bild der Sequenz mit den vorher eingegebenen Parametern und Einstellungen an die Funktionen der Vorverarbeitung übergeben. Die zurückgegebenen Schwerpunkte werden dann an die Funktion, die nach Crocker & Grier eine genauere Bestimmung der Schwerpunkte durchführt und die Momente berechnet, weiter gegeben. Diese genaueren Schwerpunkte und Momente werden dann in eine .txt Datei geschrieben.

Ein Beispiel dieser Datei Befinden sich im Anhang A auf Seite 22

2.7 Prüfen ob das nächste Bild existiert

Nach Durchlaufen der Funktionen wird die Bildnummer hochgezählt und für die Prüfung der Pfad mit der neuen Bildnummer rekonstruiert. Da die Anzahl der führenden Nullen variiert wenn sich die Anzahl der Stellen der Bildnummern ändert, z.B. von 9 auf 10, muss vor der Prüfung abgefragt werden, ob ein Sprung in der Anzahl der Stellen kommt und gegebenenfalls berücksichtigt werden. Kann das nächste Bild nicht aufgerufen werden, weil die Sequenz fertig ist oder sogar ein Bild in der Sequenz fehlt, ist das Programm fertig.

3 Vergleich von zwei Lokalisierungsalgorithmen (Belyaev)

3.1 Allgemeines

Eine der Zielen des Projektes war, die Lokalisierungsgenauigkeit verschiedener Objektlokalisierungsalgorithmen zu vergleichen und für vorgegebene Parameter wie z.B. Objektgröße und Rauschlevel, den optimalen Algorithmus auszuwählen. Dabei wurden drei verschiedene Methoden untersucht: einfache Blob-Analyse, Analysemethode von Crocker und Grier und Fit einer Gauß-Funktion.

3.2 Einfache Blob-Analyse

Dieser Standardalgorithmus untersucht nur die Kontur eines Objektes, d.h. für die Berechnung des Schwerpunktes sind nur die Randpunkte relevant. Die Funktion, die die Lokalisierung der Objekte durchführt, wurde 'BlobsWithMatlabs' genannt und hat fünf Parameter:

Im - ein Grauwertbild
minArea, maxArea - minimale und maximale Fläche der Objekten, die für weitere Berechnungen genommen werden
NB - die Nachbarschaft für das Labeling
TH - Wahl der Methode zu der Berechnung der Binarisierungsschwelle.

Die Rückgabe der Funktion ist eine 2xN - Matrix, die die X- und Y-Koordinate von Objekten enthält.

3.3 Binarisieren des Bildes

Im ersten Teil der Funktion wird das Bild binarisiert. Das schwierigste dabei ist die richtige Schwelle zu bestimmen. Es werden drei unterschiedliche Methoden zur Schwellenwertbestimmung benutzt.

Die erste ist eine feste Schwelle für jedes Bild zu nehmen. Diese Methode liefert die schlechteste Ergebnisse.

Weitere Möglichkeit ist eine Funktion aus MatLab zu benutzen. Die Funktion heißt **graythresh**. Da die Grauwerte im Bild in einem engen Bereich liegen, kann die Funktion auch keine gute Ergebnisse zeigen.

Die dritte Methode ist die Schwelle auf 20% des größten Grauwertes zu setzen. Damit kann man zwar auch keine perfekte Werte für die Schwelle berechnen, aber für einige Bildsequenzen hat die Methode gut funktioniert.

3.4 Labeling und Berechnen der Koordinaten

Weiter werden im binarisierten Bild die Objekte gefunden und die Schwerpunkte berechnet. Für das Labeling wird die Funktion **bwlabel** verwendet und die Koordinate von Objekten werden mit Hilfe der Funktion **regionprops** bestimmt. Eine der weiteren Möglichkeiten von dieser Funktion ist die Berechnung der Fläche von jedem Objekt. Durch Eingabe der minimalen und der maximalen Fläche kann man nur die Objekte auswählen, deren Fläche in diesen Bereich passt.

4 Lokalisierung der Partikel nach der Methode von Crocker und Grier (Gramlich)

4.1 Kurzzusammenfassung

Auf der Grundlage des Artikels "Methods of Digital Video Microscopy for Colloidal Studies" von John C. Crocker und David G. Grier (*Journal of Colloid and Interface Science, Volume 179, Issue 1, Pages 298-310, 1996*) wurde eine helligkeitsgewichtete Schwerpunktlokalisierung in MATLAB programmiert. Die dort beschriebene und von mir umgesetzte Methode stellt bis heute einen Standard in ähnlichen Projekten dar.

Zur Umsetzung wurden mehrere Funktionen programmiert, die sich im Anhang finden und im folgenden einzeln beschrieben werden sollen. Soweit möglich wurde Matrix-orientiert gearbeitet und die internen MATLAB-Routinen benutzt, um auf Geschwindigkeit zu optimieren. Vorab muss betont werden, dass die Gedankengänge des Algorithmus und die verwendeten Formeln auf dem genannten Artikel basieren und nicht meine eigene Arbeit darstellen.

4.2 Skizze des Algorithmus

Ausgehend von potentiellen Schwerpunkten der Partikel in einem Bild, deren Bestimmung vorhergehenden Routinen obliegt, wird der helligkeitsgewichtete Schwerpunkt innerhalb einer kreisförmigen Umgebung definierter Größe bestimmt. Sollte der Schwerpunkt hierbei näher am Ausgangspixel als an jedem anderen Pixel liegen, so wird die Berechnung akzeptiert. Andernfalls wird das am nächsten liegende Pixel als neuer möglicher Schwerpunkt gesehen und die Berechnung fortgesetzt, bis die zuerst genannte Bedingung erfüllt wurde oder 100 Suchdurchläufe vollendet wurden.

Rückgabewert der Funktion ist ein 1x4-Array mit folgenden Einträgen (Nummerierung nach Index, alle vom Datentyp float):

1. x-Koordinate des Schwerpunktes
2. y-Koordinate des Schwerpunktes
3. Moment nullter Ordnung
4. Moment zweiter Ordnung

Die genauen Schritte, Parameter und internen Funktionalitäten sind in den Beschreibungen der Funktionen niedergelegt.

4.3 Die Hauptfunktion FindSingleParticleGrier.m

Diese Funktion wird im Hauptprogramm aufgerufen und führt unter Zuhilfenahme der restlichen - untergeordneten - Funktionen alle Berechnungen durch. Sie erhält als Parameter

1. "myimage": das aktuell zu bearbeitende Bild in der Form eines Grauwertbildes, also ein zweidimensionales Array. Das Bild wird nicht verändert, ist aber notwendig, da neben dem potentiellen Schwerpunkt (Punkte 3 & 4) auch dessen Umgebung für die Algorithmen benötigt wird.
2. "w": der Durchmesser einer kreisförmigen Umgebung für alle weiteren Berechnungen, die die Umgebung einbeziehen
3. "x": die x-Koordinate des vermuteten Schwerpunktes.
4. "y": die y-Koordinate des vermuteten Schwerpunktes.

Zunächst wird durch Aufruf der Funktion CreateMask die kreisförmige Umgebung des Durchmessers w berechnet und in eine Matrix eingebettet. Dieser Schritt beschleunigt die benötigten Berechnungen, da MATLAB für Matrizen ausgelegt und optimiert wurde, gegenüber Schleifendurchläufen. Anschließend wird durch einmaligen Schleifendurchlauf die Anzahl der von Null verschiedenen Elemente in den Teilmasken gezählt und die Werte in Variablen abgelegt. Auch dieser Schritt dient der Beschleunigung des eigentlichen Lokalisierungsalgorithmus. Zu Beginn der Schwerpunktbestimmung wird geprüft, ob der Startpunkt so nah am Rand des Bildes liegt, dass die Umgebung über den Rand hinausreichen würde. In diesem Fall ist die Berechnung des Schwerpunktes nur mit zusätzlichen Annahmen zur Randbehandlung möglich und wird daher nicht durchgeführt. In diesem Fall wird die weitere Berechnung des helligkeitsgewichteten Schwerpunktes abgebrochen und der Rückgabevektor enthält den zu diesem Zeitpunkt abgelegten Stand der Koordinaten sowie die Momente dieser Koordinaten. Zur Berechnung des helligkeitsgewichteten Schwerpunktes relativ zu den Koordinaten des letzen Startpunktes wird gemäß Crocker und Grier die Rechenvorschrift

$$\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \end{pmatrix} = \frac{1}{m_0} \sum_{i^2+j^2 \leq w^2} \begin{pmatrix} i \\ j \end{pmatrix} A(x+i, y+i) \tag{1}$$

verwendet. Aus oben angeführten Gründen der Beschleunigung wird sie zerlegt in komponentenweise Matrixmultiplikationen der Teilmasken mit einem Bildausschnitt und anschließende Normierung.

Ist die Abweichung vom potentiellen Schwerpunkt, der als Startpunkt verwendet wurde, in mindestens einer Dimension betraglich größer oder gleich einem halbes Pixel, so wird

das am nächsten liegende Pixel als neuer Startpunkt angesehen. Die Schwerpunktbestimmung wird sodann erneut durchlaufen, wobei ein eingebauter Zähler dafür sorgt, dass spätestens nach 100 Iterationen abgebrochen wird und das letzte Ergebnis akzeptiert und verwertet wird.

Schließlich wird in das Ergebnisarray der Dimension 1x4 folgendes geschrieben und das Array zurückgegeben (Nummerierung nach Index, alle vom Datentyp float):

1. x-Koordinate des Schwerpunktes
2. y-Koordinate des Schwerpunktes
3. Moment nullter Ordnung
4. Moment zweiter Ordnung

Die Berechnung der Punkte 3 und 4 erfolgt durch Aufruf der Funktionen Calculatem0 und Calculatem2, jeweils mit den Ergebniskoordinaten (Punkt 1 & 2).

4.4 Die Funktion CreateMask.m

Diese Funktion dient der Erstellung einer Matrix, die die Umgebung definiert. Parameter der Funktion ist der Durchmesser der kreisförmigen Umgebung w. Zwei Teilmasken in X- und Y-Richtung werden durch eine Doppelschleife erstellt und dann als Matrix der Form [w x w x 2] zurückgegeben. Einziger Zweck dieser Funktion ist die Beschleunigung der Berechnungen in der Hauptfunktion.

4.5 Die Funktion Calculatem0.m

Diese Funktion berechnet das Moment nullter Ordnung nach der Vorschrift

$$m_0 = \sum_{i^2+j^2 \leq w^2} A(x+i, y+j) \tag{2}$$

und gibt es zurück. Als Parameter werden der Funktion folgende Daten übergeben:

1. “myimage“: Das Bild in der bereits bei der Hauptfunktion dargestellten Form
2. “w“: Der Durchmesser der Umgebung
3. “xc“: x-Koordinate des Zentrums (Schwerpunktes)
4. “yc“: y-Koordinate des Zentrums (Schwerpunktes)

4.6 Die Funktion Calculatem2.m

Diese Funktion berechnet das Moment zweiter Ordnung nach der Vorschrift

$$m_2 = \frac{1}{m_0} \sum_{i^2+j^2 \leq w^2} (i^2 + j^2) A(x+i, y+j) \tag{3}$$

und gibt es zurück. Als Parameter werden der Funktion folgende Daten übergeben:

1. "myimage": Das Bild in der bereits bei der Hauptfunktion dargestellten Form
2. "w": Der Durchmesser der Umgebung
3. "xc": x-Koordinate des Zentrums (Schwerpunktes)
4. "yc": y-Koordinate des Zentrums (Schwerpunktes)

Bei der Umsetzung der Rechenvorschrift wurde zunächst die Summe gebildet und dann die Normierung vorgenommen.

4.7 Die Funktionen FindParticleGrier.m und FindBWCentroids.m

Für den Einsatz in dem für dieses Projekt fertiggestellten Gesamtprogramm sind die bisher genannten Funktionen ausreichend. Im Laufe des Projektes entstanden aber die im Titel genannten zusätzlichen Funktionen. Sie erlauben die Schwerpunktbestimmung aller aufgefundenen Objekte in einem Bild und nutzen hierzu die Bildvorverarbeitung und Auswahl potentieller Schwerpunkte gemäß dem eingangs erwähnten Artikel. Sie benötigen die Hilfsfunktionen CreateMask, Calculatem0 und Calculatem2.

FindParticleGrier erhält als Parameter das Bild als Grauwertbild und die Umgebungsgröße analog zur Funktion FindSingleParticleGrier. Rückgabewert ist anstelle des 1x4-Arrays ein Nx4-Array mit N Schwerpunktskoordinaten und Momenten.

FindBWCentroids übernimmt die Aufgabe der Hauptfunktion FindSingleParticleGrier mit dem Unterschied, dass ein Array von Schwerpunktskoordinaten übergeben wird und als Rückgabewert demzufolge auch ein Array verwendet werden muss.

Die Funktion FindParticleGrier führt die Bildvorverarbeitung durch, detektiert potentielle Schwerpunktkoordinaten und ruft dann FindBWCentroids auf, um schließlich das Ergebnisarray zu erzeugen und zurückzugeben.

Die Bildvorverarbeitung umfasst Glättungen mit einem Mittelwertfilter und einem Gaussfilter mit $\sigma = 1$. Hierzu werden die in MATLAB bereits implementierten Routinen verwendet und auf Separierung verzichtet.

Zur Bestimmung potentieller Schwerpunkte werden zwei Kriterien zu Rate gezogen:

- Die Differenz zwischen dem einer Dilatation unterzogenen Bild und dem Originalbild ist an Stellen lokaler Maxima Null.
- In Frage kommen nur Pixel, deren Helligkeit mindestens 70% des maximalen Grauwertes im Bild erreicht.

Positionen, auf die beide Kriterien zutreffen, werden als Ausgangspunkt für den unter FindSingleParticleGrier ausführlich beschriebenen Algorithmus verwendet.

4.8 Probleme und Fortsetzungsmöglichkeiten

Der Programmteil ist vollständig lauffähig, es gibt jedoch an einigen Punkten noch die Möglichkeit, den Code auf geringere Laufzeit zu optimieren. Inhaltlich wäre es wünschenswert, nur sinnvolle Umgebungsgrößen anzuerkennen. Dazu böte sich eine Umstellung auf den Radius der kreisförmigen Umgebung anstelle des Durchmessers an. Weiterhin wäre eine Information über abgebrochene Schwerpunktbestimmungen hilfreich, um erkennen zu können, ob der Algorithmus aufgrund der Nähe zum Rand des Bildes beendet wurde. Diese Daten sollten dann verworfen werden oder besser die nötigen Zusatzbedingungen für die Randbehandlung eingefügt werden.

Hinzuweisen ist auch auf das Phänomen, dass durch Schwerpunkte ausgelöst wird, die in mindestens einer Dimension exakt in der Mitte zwischen zwei Pixeln liegen. Hierbei springt der Algorithmus zwischen den beiden Pixeln hin und her. Welcher der beiden Pixel dann fälschlicherweise als Schwerpunkt zurückgegeben wird, hängt nur vom Startpunkt des Suchvorgangs ab (Anmerkung: Auch von der Anzahl der Iterationen, die aber auf 100 fixiert wurde). An diesem Punkt gibt es noch fachliche Optimierungsmöglichkeiten, wobei dieses Problem bei simulierten Bildern leicht beobachtet werden kann, in der Praxis aber nur sehr selten auftreten sollte.

5 Bahnverfolgung/Visualisierung (Huxhorn)

5.1 Allgemeines

Die Bahnverfolgung beschäftigt sich damit, aus den einzelnen Schwerpunkte Trajektorien zu erstellen.

Um die Daten zu visualisieren wurde ein Plugin für das schon vorhandene Milan Programm geschrieben. Es gibt nun die Möglichkeit Schwerpunktdaten sowie Bilder zu laden, Trajektorien zu erstellen und Informationen über sie abzurufen. Es ist weiterhin möglich die Bilder und Trajektorien vergrößert zu betrachten um den genauen Verlauf zu erkunden. Auch kann so kontrolliert werden, ob die Trajektorie richtig erstellt wurde.

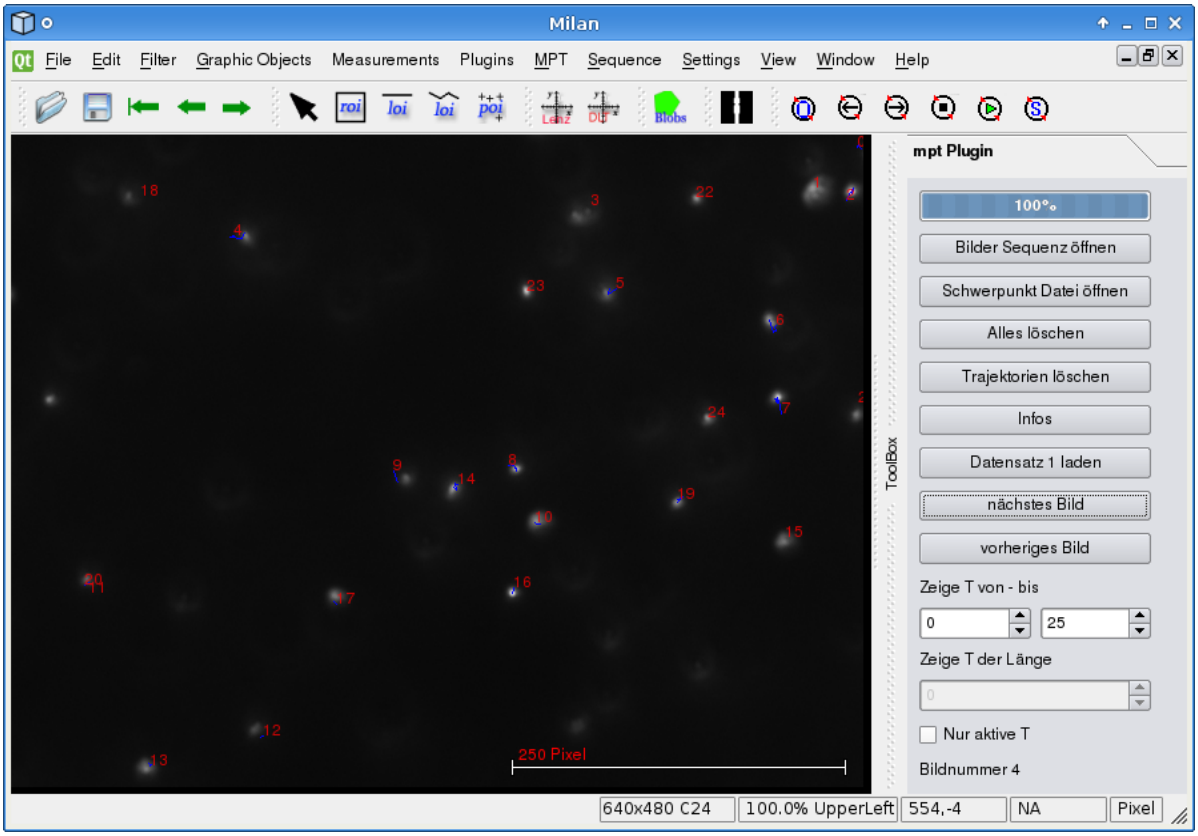


Figure 1: mpt Plugin mit bereits erstellten Trajektorien (in Blau)

In Figure 1 ist das Programm zu sehen. Auf der rechten Seiten befinden sich die Bedienelemente des mpt Plugins.

Erklärung der Elemente

- Bildersequenz öffnen: Öffnet einen Fileopen Dialog für die Bilder. Es sind alle Bilder zu markieren die geöffnet werden sollen. Es langt nicht nur das erste Bild zu markieren.
- Schwerpunktdatei öffnen: Öffnet einen Fileopen Dialog für die Schwerpunkt Textdatei.
- Alles löschen: Löscht Bilder, Schwerpunktdaten und Trajektorien
- Trajektorien löschen: Es werden nur die Trajektorien gelöscht. Bilder und Schwerpunktdaten bleiben geladen.
- Infos: Zeigt ein paar Daten zu den Trajektorien. Darunter der Faktor ”drift”. Das Verhaeltnis zwischen der Distance zwischen Start und Endpunkt einer Trajektorie

und den gesamt zurückgelegten Weg. Damit sollte sich erkennen lassen ob eine Trajektorie lokal beleibt oder sich stark fortbewegt.

- Datensatz 1 laden: Es werden 10 vorgegebene Bilder + Schwerpunktedaten geladen.
- nächstes Bild:
- vorheriges Bild: Zeigt das entsprechende Bild an und erweitert die Trajektorien entsprechend.
- Zeige Trajektoren von/bis: Hier ist es möglich nur bestimmte Trajektorien zu untersuchen.
- Es kann auch nützlich sein nur Trajektoren darzustellen, die eine gewisse Mindestlänge haben. Diese Funktion ist noch nicht implementiert.
- Nur aktive Trajektorien: Es werden nur aktive Trajektorien angezeigt. Eine Trajektorie ist dann aktiv, wenn die derzeit angezeigte Bildnummer zwischen dem Start- und Endbild der Trajektorie liegt.
- Zuletzt wird die aktuelle Bildnummer angezeigt

Es ist nicht notwenig die Bilder vor den Schwerpunktdaten zu laden. Das Programm füllt seine internen Datenstruckturen automatisch richtig. Auch werden 'Dummy'-Bilder erzeugt, sollte es mehr Schwerpunktdaten als Bilder geben.

Es sollten alle gänige Bildformate unterstützt sein.

Die Schwerpunkte sind in einer Textdatei gespeichert. Ein Beispiel dieser Datei Befinden sich im Anhang A auf Seite 22

Hat eine Zeile mehr als zwei Spalten, wird eine Warnung ausgegeben und der Lesevorgang fortgesetzt.

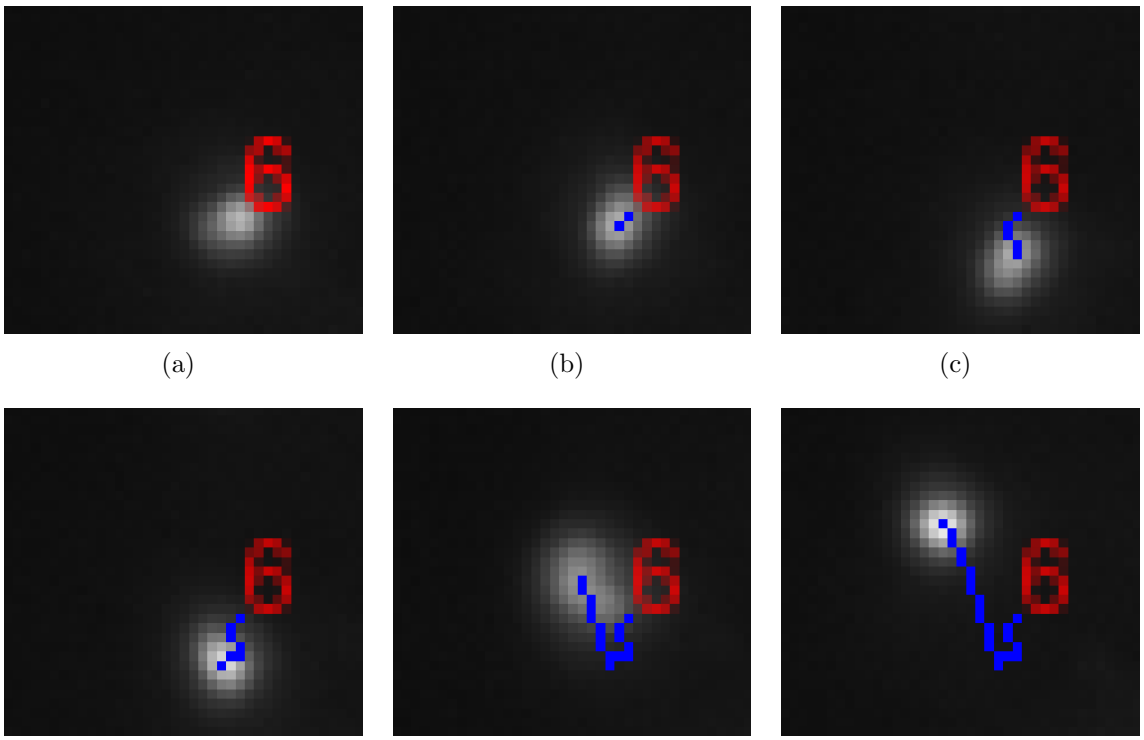
Figure 2 zeigt das Entstehen einer Trajektorie. Wie genau der Algorithmus funktioniert wird im nächsten Kapitel erklärt.

5.2 Der Algorithmus

In diesem Kapitel wird nun der zugrunde liegende Algorithmus erklärt, der die Trajektorien erstellt.

Ein Bild beinhaltet relativ wenige detektierte Objekte. Um das selbe Objekt im nächsten Bild zu finden wird ein Suchradius definiert. Befindet sich im nächsten Bild ein Objekt innerhalb vom Suchradius, wird es der Trajektorie hinzugefügt. Andernfalls nicht.

Der Suchradius ist wie folge festgelegt: Fläche des Bildes (in Pixel) geteilt durch die Anzahl der Objekte. Das ergibt die Fläche, die jedes Objekt zur Verfügung hat. Wird die Fläche als ein Kreis gedeutet, kann ein Radius bestimmt werden. Dieser Wert wird



(a) (b) (c)

(d) (e) (f)

Figure 2: Trajektorie 6 entsteht

noch durch 5 geteilt. Ein genaues Kriterium dafür wurde nicht gefunden, sondern experimentell ermittelt. Der Suchradius ist in Figure 3 schematisch dargestellt.

Ist nun der Abstand des neuen Objektes kleiner als der Suchradius, gehört es zur Trajektorie. Es wird jedes Objekt mit jedem verglichen um den kleinsten Abstand zu finden.

Im zweiten Schritt wird für die übrig gebliebenen Objekte jeweils eine neue Trajektorie gestartet.

5.3 Datenstruktur

Um die Trajektorien zu speichern wurde eine eigene Datenstruktor entwickelt, die folgenden Aufbau hat:

```
class myObject
{
public:
```

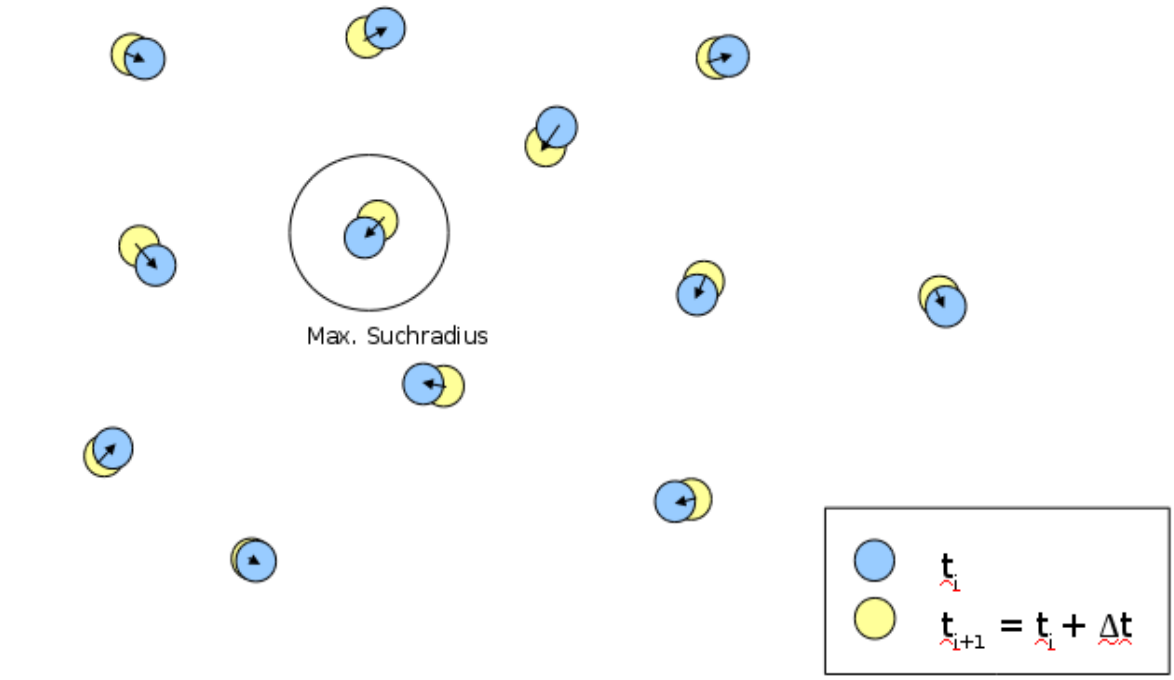


Figure 3: Schematische Darstellung des Suchradius

```
    bool schonBesucht;
    QPoint schwerpunkt;
    float anzPixel;
    int nummer;
};


class myTrajektorie
{
public:
    // Bei welchen Bild faengt sie an und hoehrt auf
    int startBild, endBild;

    // Alle Schwerpunkte die zu dieser Trajektorie gehoeren
    QVector<myObject*> objects;
};
```

```
QVector<myTrajektorie> trajektorien;
```

Erklärung:

Die Variable ‘trajektorien‘ vom Typ ‘myTrajektorie‘ repräsentiert alle gefundenen Trajektorien. Gespeichert werden die Start- und Endbild Nummer in der die Trajektorie existiert sowie die Objekte, die zu dieser Trajektorie gehören. Ein Objekt hingegen enthält seinen Schwerpunkt. Eine Nummer, das wievielte Objekt es in einem Bild ist. Die Anzahl der Pixel, also die Helligkeit des Objekts. Diese Information wird zur Trajektorienerstellung momentan nicht verwertet. Sowie eine Variable ‘schonBesucht‘, die aussagt, ob das Objekt schon einer Trajektorie zugeordnet wurde.

5.4 Problemfälle

Beim Erstellen der Trajektorien gibt es gewisse Probleme wie in Figure 4 dargestellt. Wie Abbildung a) zeigt, entfernt sich ein Objekt aus der Fokusebene der Kamera. Es ist nur noch eine unscharfe Abbildung zu sehen, die nicht mehr detektiert werden kann. Hier ist die Trajektorie einfach zu Ende. Abbildung b) zeigt einen ähnlichen Fall. Hier wandert ein Objekt in die Fokusebene und kann nun detektiert werden. Eine neue Trajektorie entsteht.

Problematischer ist es, wenn die Objekte zu nah beisammen sind, wie Abbildung c) zeigt. Es ist nicht mehr eindeutig erkenntbar, welches das richtige nächste Objekt ist. Aber es ist möglich mit einem geeigneten Algorithmus diese Situation zu erkennen und die Trajektorie vorzeitig zu beenden.

Der schlimmste Fall ist in Abbildung d) gezeigt. Es taucht gleichzeitig ein Objekt ab und ein Neues kommt hinzu. Hier ist nicht zu erkennen, dass es ein Fehler gibt. Mit dem bloßem Auge ist die Situation nur zu erkennen, wenn das abtauchende Objekte noch eine unscharfe Abbildung hinterlässt. Einen Algorithmus zu schreiben, der auch noch die falsch abgebildeten Objekte mit in die Analyse einbezieht, erscheint recht schwierig.

Zum Beispiel Figure 2 auf Seite 12. Das Objekt hat nach jedem Bild eine neue Richtung bis auf die letzen beiden Bilder e) und f). Hier bewegte sich das Objekte in die selbe Richtung weiter und ist auch wieder deutlich heller. Auch wenn es sehr wahrscheinlich noch das ein und selbe Objekt ist, was hier beobachtet wurde. Könnte so ein gleichzeitiges Auf- und Abtauchen aussehen..

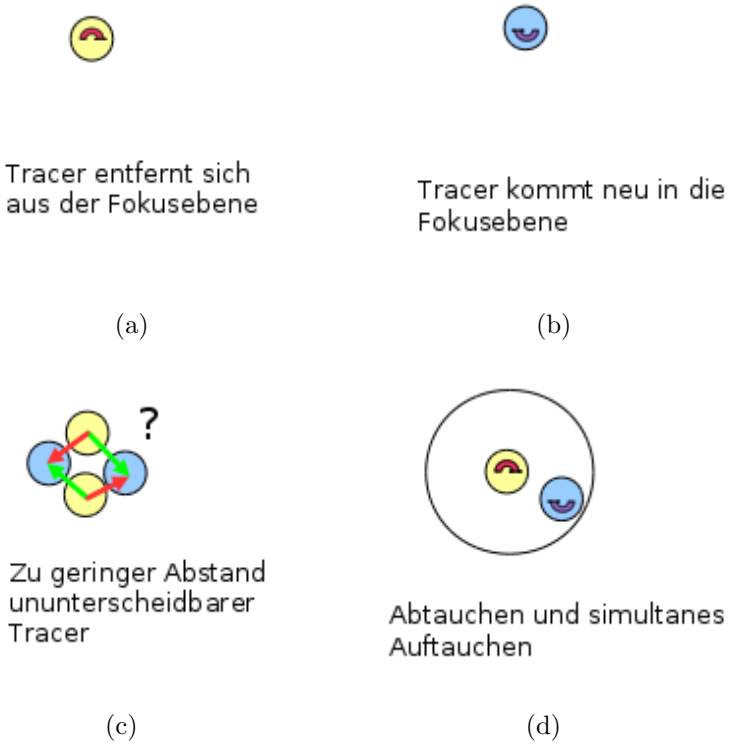


Figure 4: Probleme beim erstellen von Trajektorien

5.5 Sprünge

Sind die Trajektorien erstellt worden, sind weitere Test nötig um mögliche Fehler zu finden. Figure 5 zeigt eine Trajetorie mit etwa 100 detektierten Objekten. Auf der Abszisse sind die Objekte mit ihrer jeweiligen Nummer aufgetragen. Die Ordinate zeigt den Abstand zweier nachfolgender Objekte in Pixel.

Es ist zu sehen, dass der Abstand von Objekt zu Objekt kleiner als 0.2 Pixel ist. Bis auf eine einzige Stelle. Im Bild mit einem roten Kreis markiert. Hier beträgt der Abstand 1.8 Pixel. Es ist leicht zu sehen, dass dieses Objekt nicht zur Trajektorie gehört. Vielmehr beginnt hier eine neue, zweite Trajektorie. Sie wurde nur fälschlicherweise mit der ersten verbunden. Solch ein Fehler wird mit ‘Sprung‘ betitelt.

5.6 Statistik

Das mpt Milan Plugin bietet auch die Möglichkeit mehr Informationen über die einzelnen Trajektorien zu erlangen wie in Figure 6 zu sehen ist. Es stehen Informationen

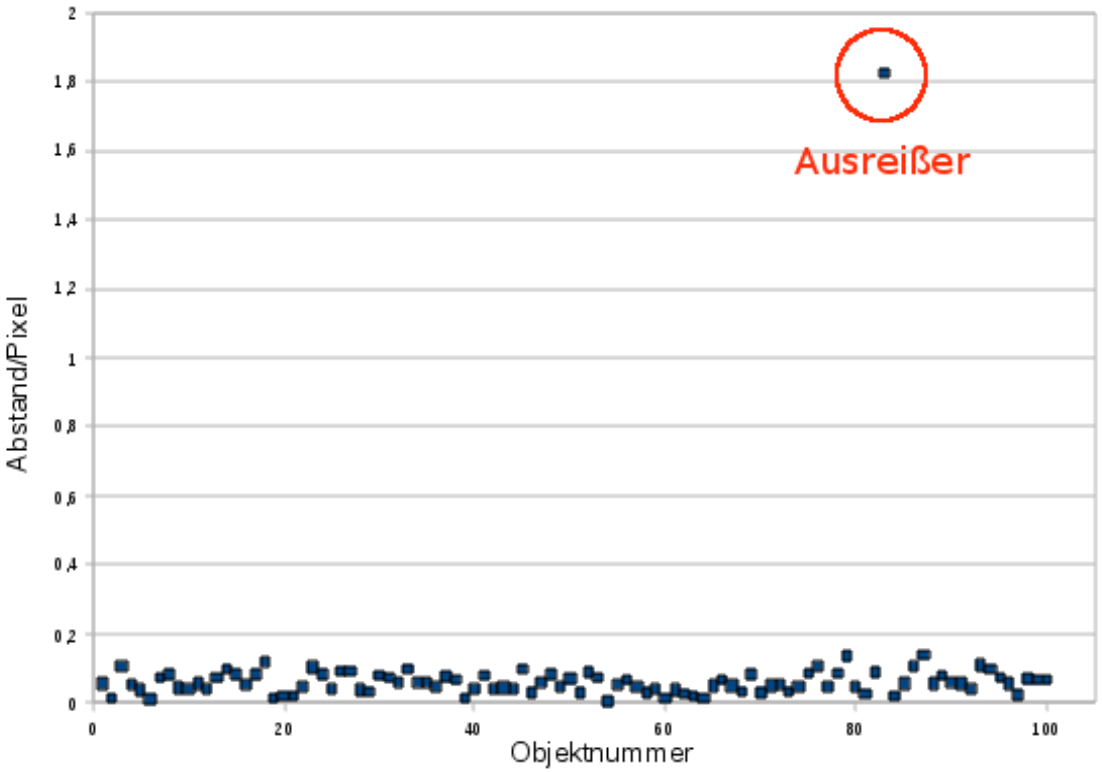


Figure 5: Sprung in der Trajektorie

zu Start/Endbild, Länge, Geschwindigkeit, mittleres Verschiebungsquadrat und einen eigens entwickelten Parameter 'drift' bereit. Dabei wird die mittlere Geschwindigkeit nochmals getrennt in X und Y Richtung angegeben. Zur Berechnung wurde wegen Zeitmangel die Qt eigene Manhatten Distance Funktion herangezogen statt des noch zu implementierenden Pythagoras.

Auffällig ist Trajektorie 1. Sie geht über 6 Bilder, hat aber eine Geschwindigkeit von 0. D.h. sie 'klebt' fest.

Und Trajektorie 6 bewegt sich doppelt so schnell in Y Richtung, als in X Richtung. Siehe Figure 2 auf Seite 12

Das mittleres Verschiebungsquadrat berechnet sich wie folgt [PCI1]

$$\langle \Delta x^2 \rangle = \frac{1}{N} \sum_{n=1}^{N} \Delta x_n^2 \tag{4}$$

N ist die Anzahl der Objekte in einer Trajektorie
Δx der Abstand von Objekt zu Objekt

Der Parameter ‘drift‘ sagt aus, ob eine Trajektorie relativ nah bei ihrem Entstehungsort bleibt oder weit weg wandert. Der Paramter bewegt sich in dem Intervall [0,1] und ist einfach die Relation zwischen dem derzeitigen Abstand zum Entstehungspunkt und dem gesammt zurückgelegten Wert.

Bewegt sich die Trajektorie z.B. relativ wenig von ihrem Entstehungsort weg, ist der Abstand zum Entstehungsort klein, der gesamte zurückgelegte Weg wird aber immer größer. Der Quotient wandert also gegen 0. Gegenbeispiel: Die Trajektorie wandert auf einer exakt geraden Linie, ohne Knicke und Kurven. Hier ist der gesamte Weg gleich der Distanz zum Ursprung. Der Quotient geht gegen 1.

Wie sich dieser Wert für lange Trajektorien entwickelt ist unklar. Sicher wird der gesamte zurückgelegte Weg immer größer sein als der Abstand zum Entstehungspunkt. Aber durch die zufällige Bewegung könnte der Quotient für alle Trajektorien sich einem festen Wert annähern, z.B. 0.5.

	T	Start	End	Anz	meanSpeed	meanSpeedXY	mitVerQuad	drift
1	1	0	5	6	0.00	0.00 0.00	0	nan
2	2	0	5	6	5.20	1.60 3.60	34	0.230769
3	6	0	5	6	6.00	2.00 4.00	51	0.6
4	7	0	5	6	6.20	2.20 4.00	47	0.548387
5	8	0	5	6	3.60	1.40 2.20	14	0.333333
6	16	1	5	5	2.00	0.25 1.75	6	0.25
7	0	0	3	4	2.67	1.00 1.67	9	0.5
8	4	0	3	4	6.67	4.67 2.00	51	0.5
9	5	0	3	4	5.00	2.33 2.67	25	0.333333

Figure 6: Trajektorien Informationen

5.7 Verbesserungen/Erweiterungen

Es gibt einige Ansätze für Verbesserungen

- QList statt QVector. Ein Vector ist zwar beliebig erweiterbar, doch wenn seine interne Größe überschritten wird, muss der gesamte Inhalt kopiert werden. Das fällt bei QList weg. Dennnoch bietet QList den Indexzugriff
- Den Milan Sequenzer nutzen. Es müss sich nicht mehr um das Laden der Bilder gekümmert werden. Auch die Gefahr, dass der RAM voll läuft wenn zuviele Bilder geladen werden
- Es braucht ein besseres Kriterium für den Suchradius
- Nicht mit der Manhatten Distance rechnen. Pytagoras ist genauer

6 Hardware Optimierung (Römer)

6.1 Allgemeines

Dieses Teilprojekt befasst sich mit der Umrüstung der Fluoreszenzbeleuchtung eines Leica DM IRB Mikroskops. Motivation hierbei ist es, die übliche Quecksilberdampflampe gegen eine geeignete LED zu tauschen und somit Verbesserungen und neue Möglichkeiten in den folgenden Bereichen der Fluoreszenzmikroskopie zu erzielen.

- Pulsen der LED im Moment der Bildaufnahme zur Erhöhung der Lebensdauer von Fluorochromen.
- Erhöhung der Lichtleistung zur Senkung der Belichtungszeit, erhöhte Bildfrequenz
- Kostensenkung, Lebensdauerverbesserung und Erhöhung des Wirkungsgrades

1) Da eine Quecksilberdampflampe nach dem Betrieb eine lange Abkühlzeit benötigt bevor sie erneut gezündet werden kann, eignet sie sich nur zum Dauerbetrieb und ist zum pulsen nicht geeignet.

Da eine Quecksilberdampflampe nach dem Betrieb eine lange Abkühlzeit benötigt bevor sie erneut gezündet werden kann, eignet sie sich nur zum Dauerbetrieb und ist zum pulsen nicht geeignet.

2) Durch die Erhöhung der optischen Ausgangsleistung besteht die Möglichkeit einer Reduzierung der Belichtungszeit, was unter Verwendung einer geeigneten Kamera die Möglichkeit bietet, bei gleicher Messzeit eine erhöhte Bilderserie zu erzielen und somit eine genauere Messreihe zu reproduzieren.

3) Im Folgenden werden noch technische Gründe, die für eine Umrüstung sprechen aufgelistet:

Lebensdauer:	LED ca. 10.000 Std.	Quecksilberlampe ca. 150 Std.
Anschaffungskosten:	LED ca. 10 Euro	Quecksilberlampe ca. 200- 300 Euro
Wirkungsgrad:	Quecksilber: ca. $\frac{12mw}{50W} = 0,4\%$	LED: ca. $\frac{1,2W}{5W} = 28\%$

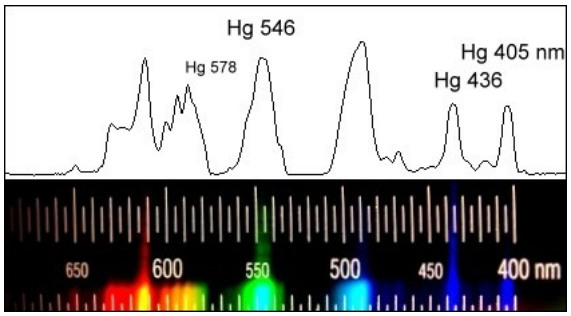


Figure 7: Spektrum der Quecksilberlampe

Der geringe Wirkungsgrad der Quecksilberlampe ergibt sich durch das breite Spektrum. Siehe Figure 7. Die in diesem Projekt verwendeten Fluorochrome (dragon green), fluoreszieren bei einer Erregerwellenlänge von 450nm. Aus der spektralen Verteilung ist ersichtlich, dass bei dieser Wellenlänge nur ein geringer Anteil der optischen Gesamtleistung emittiert wird. Eine geeignete LED dagegen, emittiert überwiegend monochromatische Strahlung in einem sehr kleinen Spektralbereich. Das hat zur Folge, dass keine Strahlung am Filterblock des Mikroskops absorbiert wird und fast die gesamte optische Leistung genutzt werden kann.

6.2 Vorbereitendes

Vor der eigentlichen Realisierung der Umrüstung von einer Quecksilberdampflampe auf eine geeignete LED wurde zum einen die optische Ausgangsleistung der vorhandenen Beleuchtung ermittelt. Unter Verwendung eines Leistungsmessgerätes(Coherent Fieldmaster) und dem 5xObjektiv mit NA=0,15 für kleinst mögliche Divergenz ergab sich eine Leistung von: **ca. 12 mW**

Zum anderen wurde die ausgewählte LED (Diamond Dragon) auf ihre exakte Spektralbreite vermessen. Dies geschah mit Hilfe eines Spektrometers, zum einen mit und zum andern ohne den Filterblock (Nr.513819) zur Selektierung der Erregerstrahlung. In Figure 8 Seite 37 sind beide Kurven aufgetragen:

- Dunkelblau mit Filterblock
- Hellblau ohne Filterblock

Die Kurven zeigen, dass die LED für die Fluoreszenzmikroskopie geeignet ist. Da vom Filterblock nur die geringfügigen Randbereiche des Spektrums der LED abgeschnitten

werden und der Großteil der Strahlung durch den Filterblock transmittiert und somit als Erregerstrahlung genutzt werden kann.

6.3 Angaben zur verwendeten LED

Hersteller:	Osram
Typ	Diamond Dragon LBW5AP
Lichtfluss	1250mW
Wellenlänge:	455nm
Abstrahlwinkel:	120°
Wirkungsgrad:	28%

6.4 Realisierung

Zur Einkopplung der Strahlung der LED in das Mikroskop wurde ein baugleiches Lampengehäuse verwendet. Gründe hierfür sind eine vorgegebene optimale Einkopplung der Strahlung in das Mikroskop und vorgegebene Verstellmöglichkeiten in allen drei Achsen. Zunächst wurde der für die LED nutzlose Parabolspiegel entfernt und ein passendes Kühlgerippe passend geschnitten, auf dem die LED dann platziert wurde. Hierbei wurde darauf geachtet, dass sowohl LEDs mit Stern als auch mit Rechteckplatinen montiert werden können. Das Kühlgerippe samt LED wurde so konstruiert, dass sich die LED am Punkt der vorherigen Glühwendel der Quecksilberlampe befindet.

Zusätzlich wurde ein Steuerkasten samt Platine angebracht, welche es ermöglicht die LED über einen Computer anzusteuern und somit Leistung und Pulsung der LED zu steuern.

6.5 Ergebnisse

Zunächst wurde die LED optimal ausgerichtet, dass heißt auf maximale Ausgangsleistung Justiert und hierbei eine Leistung von **12,8 mW** erzielt.

Des Weiteren wurde eine Probe mit Fluorochromen von 500 nm und 200 nm Durchmesser unter dem Mikroskop betrachtet und mit Hilfe einer schwarz/weiß Kamera aufgenommen. Siehe Figure 9 auf Seite 38. Im Vergleich zur Betrachtung mit Hilfe der Quecksilberlampe ist bereits eine Kontrastverbesserung festzustellen.

6.6 Anmerkungen und Aussichten

Das erbrachte System ist funktionsfähig und macht eine Quecksilberlampe bereits überflüssig. Wobei folgendes anzumerken ist:

- Die LED wurde aufgrund eines Universalnetzteils nicht bei maximaler Leistung betrieben und lässt somit eine Leistungsoptimierung offen.
- Zur Aufnahme der Bildserien sollte eine anwendungsspezifische Kamera verwendet werden, dass heißt eine Kamera mit großer Lichtempfindlichkeit.
- Die verwendete LED wurde ohne optischen Aufbau genutzt. Ihr großer Divergenzwinkel von ca.120 Grad macht eine optimale Einkopplung nicht möglich. Ein Aufbau zur Divergenzminderung würde eine Leistungssteigerung mit sich bringen und das Pulsen ermöglichen.
- Zur Ansteuerung der LED, zum Pulsen und zur Leistungseinstellung, ist es notwendig ein geeignetes Programm über MATLAB zu schreiben.

7 Fazit (Gramlich)

Die Verbesserung der Beleuchtung darf als klarer Erfolg bewertet werden, da sich hierdurch Kosten reduziert und Messmethoden verbessert haben.

Auch die Programme zur Lokalisierung der Partikel und Verfolgung der Trajektorien sind lauffähig und können genutzt werden, um verschiedene Ansätze zur Verbesserung der Auswertung gegeneinander zu testen.

Insofern sind die Ziele des Projektes erfüllt. Es bestehen weiterhin konkrete Verbesserungs- und Erweiterungsmöglichkeiten, von denen einige bereits an den entsprechenden Stellen in unserer Dokumentation beschrieben werden.

References

[PCI1] Mittleres Verschiebungsqudrat Kapitel 2.2.4.1 `http://www2.uni-siegen.de/~pci/lehre/Script-PCI.pdf`

[] Methods of Digital Video Microscopy for Colloidal Studies“ von John C. Crocker und David G. Grier (*Journal of Colloid and Interface Science, Volume 179, Issue 1, Pages 298-310, 1996*)

A Beispiel Schwerpunktdatei

```
SP.x      SP.y      M0      M2      Pixel
//1
52.828   51.204  235.000    1.374

//2
47.963   53.073   41.000    1.195

//3
46.000   54.859   32.000    1.281

//4
44.103   54.948   29.000    1.414

//5
45.000   51.957   35.000    1.400

//6
45.888   53.075   40.000    1.325

//7
45.917   53.000   36.000    1.333

//8
43.930   51.791   43.000    1.302

//9
42.052   55.103   29.000    1.207

//10
42.036   55.000   42.000    1.405
```

B Quelltext

```
%Mainprogramm
%Sven Severin Kraft
%h_da (Hochschule Darmstadt)
%Martrikelnummer 708288
```

```
clc, clear all;

%Aufruf der Bilddaten
[FileName,PathName,FilterIndex] = uigetfile...
    ({'*.jpg;*.tif;*.png;*.gif;*.tiff','All Image Files';...
    '*.*','All Files' },'Startbild der Sequenz');

%Titel in der .txt
titel1=['SP.x      SP.y      M0      M2      Pixel\r\n'];

%Anzahl der Dateien im Ordner
NumberDatafiles=length(dir(PathName));

%Zerlegung des ganzen Pfades in Pfad, Dateiname und Endung
[pathstr, name, ext]=fileparts([PathName,FileName]);

% Angaben für die Vorverarbeitung
minArea=input('minimale Fläche eines Partikels: ');
maxArea=input('maximale Fläche eines Partikels: ');
%NB=input('Bitte ''4'' für 4er oder ''8'' für 8er Nachbarschaft eingeben: ')
NB=menu('Nachbarschaft','4er','8er')*4

TH = menu('Art der Schwelle','fester Schwelle','mit Graythreshfunktion'...
    ,'mit 20% des größten Grauwertes')

rStrel=input...
    ('Radius des strukturierenden Elements für die Hintergrundkorrektur')

[filename, pathname] = uiputfile(...
    {'*.txt'},...
    'Where centroids save');

fid=fopen([pathname,filename],'w');
fprintf(fid,titel1);

% Radius für Masken festlegen
kernelRadius = 1;
l=1;

%Zerlegen des Namens in Zahlen und Anhang
for j=0:NumberDatafiles
    for i=length(name):-1:0
        A=name(i)
        if ((name(i)~='0')&&(name(i)~='1')&&(name(i)~='2')&&...
```

```
            (name(i)~='3')&&(name(i)~='4')&&(name(i)~='5')&&...
            (name(i)~='6')&&(name(i)~='7')&&(name(i)~='8')&&...
            (name(i)~='9'));
        Digits=name(i+1:length(name));
        linename=name(1:i);
        break
    end
end

%Bildnummer des ersten Bildes
imagenumber=str2num(Digits);

%Bildnummer hochzählen, und nächstes Bild laden zum prüfen
imagenumber=imagenumber+1;
lengthimnumber=length(num2str(imagenumber));

%Anzahl der Nullen
ZeroDigits=Digits(1:length(Digits)-lengthimnumber);
newDigits=[ZeroDigits,num2str(imagenumber)];
name=[linename,newDigits];
image=imread([PathName,linename,Digits,ext]);

%Vorverarbeitung
Im=imPreProcessing(image, rStrel );
Matrix=BlobsWithMatlab( Im, minArea, maxArea, NB, TH );
sizeof=size(Matrix);
fprintf(fid,'//%i\r\n',1);

%Übergabe der Schwerpunkte pro Bild an die Verarbeitungsfunktionen
for k=1:sizeof(1)

    %Hier werden Schwerpunkte und Momente der Partikel gesucht
    Centroids=FindSingleParticleGrier...
        (Im, 2*kernelRadius+1, Matrix(k,1), Matrix(k,2));
    Werte=[Centroids(1),Centroids(2),Centroids(3),Centroids(4)];

    % Fit an eine Gausskurve anhand der berechneten
    % Schwerpunktkoordinaten
    Uebergabekoordinaten = Werte(1:2);
    sigma = 1;
    gaussFit = fitGaussian...
        (Im, kernelRadius,Uebergabekoordinaten,sigma);

    % Schwerpunkte und Momente werden in eine .txt geschrieben
```

```
        fprintf(fid,'%5.3f %8.3f %8.3f %8.3f\r\n',Werte);
        fprintf(fid,'\r\n');
    end

    %Wenn die aufgerufene Datei nicht existiert, ist die Verarbeitung
    %fertig
    if((exist([PathName,linename,newDigits,ext]))==0);
        Last_Sequenz=[PathName,linename,Digits,ext]
        break
    end
    l=l+1;
end
```

```
% Hintergrundausgleich
% Autor: Sergey Belyaev
% Datum: 07.05.2010


%Ein Bild wird mit in Matlab integrierten Funktionen vorverarbeitet. Es
%wird ein Hintergrundausgleich mit Funktionen 'imopen' und 'imsubtract'
%durchgeführt.
%Die Funktion gibt ein Grauwertbild zurück.
%
%Parameter der Funktion:     I - Grauwertbild
%                            rStrel - Radius des strukturierenden Elements
%                                     für die Hintergrundkorrektur



function [ ImS ] = imPreProcessing( I, rStrel )

    Im8 = uint8(I);      %Das Bild wird zu einem 8bit-tiefen Bild konvertiert

    S = size(Im8);
    N = numel(S);

    if N == 3,
        Im = rgb2gray(Im8);
    else
        Im = Im8;
    end

    se = strel('disk', rStrel);
```

```
    ImO = imopen(Im, se);
    ImS = imsubtract(Im, ImO);

end
```

```
img = imread('img_000002.tif');
img = img(:,:,1);
%imtool(img);
x = fitGaussian(img, 3, [46.286, 54 ]' , 2);
```

```
function [x] = fitGaussian( image, r, sp0, sigma)

    % initial parameters
    p0 = zeros(5,1);
    p0(1) = sp0(1);
    p0(2) = sp0(2);
    p0(3) = sigma;
    p0(4) = 120;
    p0(5) = 80;
    %p0(5) = 120;

    % lower and upper bounds
    lb = zeros(0);
    ub = zeros(0);
    % options
    %options = optimset('PlotFcns',@optimplotstepsize,'Algorithm','levenberg-ma
    options = optimset('Algorithm','levenberg-marquardt','TolX',1e-6,'MaxFunEva


    % Optimize!
    [x,resnorm] = lsqnonlin(@optiFunc,p0,lb,ub,options);

    % Nested function for function evaluation
    function F = optiFunc(x)
        x0  = x(1);
        y0  = x(2);
        sx  = x(3);
        A   = x(4);
        C   = x(5);

        % Squares for better readability
        sx2 = sx*sx;

        % Range for image evaluation
        x1 = ceil(x0-r); x2 = ceil(x0+r);
```

```
        y1 = ceil(x0-r); y2 = ceil(x0+r);

        % sx,sy = varianzen in x und y-Richtung
        % rho    = Korrelationiskoeffizient

        F = 0;
        for i =  x1:x2
            for j = y1 : y2
                ival = double(image(j,i));
                %F = F + ival - 1/(2*pi*sx*sy*sqrt(1-rho2))*exp(-1/(2*rho2)*((i-
                G = A*exp(-1/2*((i-x0)*(i-x0)/sx2+(j-y0)*(j-y0)/sx2))+C;
                F = F +abs(ival - G);
            end
        end
        F;
    end
end
```

```
function [ result ] = FindSingleParticleGrier( myimage, w, x, y )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
%   myimage = image; w = kernelsize; x,y = coordinates of candidate
%   centroid
% result returns 1x4 array of [x,y,m0,m2]
% x,y = coordinates of centroid
% m0, m2 = Moments of centroid


% please look at FindBWCentroids.m for detailled commentaries !
% this is nothing but a simplified version
r = floor(w/2);

Mask = CreateMask(w);
countx = 0;
county = 0;

for i = 1 : w
    for j = 1 : w
        if(Mask(i,j,1) ~= 0)
            countx = countx + 1;
        end

        if(Mask(i,j,2) ~= 0)
            county = county + 1;
        end
```

```
    end
end

epsx = double(0.0);
epsy = double(0.0);

result = zeros(1,4);


if (x >= r && y >= r && x <= (size(myimage,1) - r) && y <= (size(myimage,2) - r)
    cutPart = myimage(x - r : x + r , y - r : y + r);
    epsx = sum(sum(Mask(:,:,1) .* double(cutPart)));
    epsy = sum(sum(Mask(:,:,2) .* double(cutPart)));

    if(epsx ~= 0.0)
        epsx = epsx / ( countx * Calculatem0(myimage, w, x, y )/double(w));
    end
    if(epsy ~= 0.0)
        epsy = epsy / ( county * Calculatem0(myimage, w, x, y )/double(w));
    end



end

% iterative correction if absolute of any eps is >0.5
count = 0;
while (((abs(epsx) >= 0.5) || (abs(epsy) >= 0.5)) && count < 100)
    x = x + round(epsx);
    y = y + round(epsy);

    if (x >= r && y >= r && x <= (size(myimage,1) - r) && y <= (size(myimage,2)
        cutPart = myimage(x - r : x + r , y - r : y + r);
        epsx = sum(sum(Mask(:,:,1) .* double(cutPart)));
        epsy = sum(sum(Mask(:,:,2) .* double(cutPart)));

        if(epsx ~= 0.0)
            epsx = epsx / ( countx * Calculatem0(myimage, w, x, y )/double(w));
        end
        if(epsy ~= 0.0)
            epsy = epsy / ( county * Calculatem0(myimage, w, x, y )/double(w));
        end

    end
```

```
    count = count + 1;
end

% create result array
result(1) = (x + epsx);
result(2) = (y + epsy);
help1 = round(x+epsx);
help2 = round(y+epsy);

result(3) = Calculatem0(myimage, w, help1, help2 );
result(4) = Calculatem2(myimage, w, help1, help2 );


end
```

```
function [ result ] = FindParticleGrier( myimage, w )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
% myimage = Image to be analysed
% w = kernel size
% result = Array [x y m0 m2]
% x,y = coordinates of centroids
% m0, m2 = Moments of centroids
% Analysing a whole grayscale image. detecting particles
% and calculating centroids and moments as shown by Crocker & Grier


% using matlab routines instead of separating the filter
% kernel mean
meanfilter = fspecial('average',w);
A = imfilter(myimage, meanfilter, 'symmetric');

% kernel gaussian, sigma = 1 according to the paper
gaussianfilter = fspecial('gaussian',w,1);
B = imfilter(A,gaussianfilter, 'symmetric');

% mean of entire image
x = mean(mean(myimage));

% create dilated image
DilMask = strel('disk',w,0);
Dil = imdilate(B,DilMask);

% Combining the image with the dilated image
C = Dil - B;
```

```
% Upper 30% Criterium as proposed in the paper
imax = max(max(myimage));
threshold = 0.7*imax;

% container for up to 500 candidate centroids
xy = zeros(500,2);

% Combining the criteria to get candidate centroids
count = 1;
[m,n] = size(myimage);
for i = 1 : m,
    for j = 1 : n,
        if C(i,j) == 0
            if myimage(i,j) > threshold
                xy(count,1) = i;
                xy(count,2) = j;
                count = count + 1;
            end
        end
    end
end

 % cut array to minimum size
xy = xy(1:count - 1,:);

% Calculating brightness-weighted centroids
BWC = FindBWCentroids(myimage,w,xy);

% Combining results to the result vector
result = zeros(size(xy,1),4);
for i = 1 : size(xy,1),
   xcor = BWC(i,1);
   ycor = BWC(i,2);

   if (xcor ~= 0.0 & ycor ~= 0.0)
    result(i,1) = xcor;
    result(i,2) = ycor;
    result(i,3) = Calculatem0(myimage,w,round(xcor),round(ycor));
    result(i,4) = Calculatem2(myimage,w,round(xcor),round(ycor));
   end
end
```

```
function [ BWCentroids ] = FindBWCentroids( myimage, w, cc)
%UNTITLED Summary of this function goes here
```

```
%   Detailed explanation goes here
%   BWCentroids = [x y]
%   w = radius of kernel, cc = coordinates of candidates ( Array nx2)

% define kernel radius
r = floor(w/2);

% empty container for centroids
BWCentroids = zeros(size(cc,1),2);

% create a Mask of size wxw
Mask = CreateMask(w)
countx = 0;
county = 0;

% sum up non-zero elements of Mask
for i = 1 : w
    for j = 1 : w
        if(Mask(i,j,1) ~= 0)
            countx = countx + 1;
        end

        if(Mask(i,j,2) ~= 0)
            county = county + 1;
        end
    end
end



% calculate brightness-weighted centroids for every pair of
% coordinates in cc
for n = 1 : size(cc,1)

    % initialize deviations from cc
    epsx = double(0.0);
    epsy = double(0.0);

    % get a pair of candidate coordinates
    ccn = cc(n,:);
    xx = ccn(1);
    yy = ccn(2);

    % ignore edge of image depending on kernel size
```

```
    if (xx >= r & yy >= r & xx <= (size(myimage,1) - r) & yy <= (size(myimage,2)

        % extract part of the image and calculate epsilons
        cutPart = myimage(xx - r : xx + r , yy - r : yy + r)
        epsx = sum(sum(Mask(:,:,1) .* double(cutPart)));
        epsy = sum(sum(Mask(:,:,2) .* double(cutPart)));

        % normalize epsilons while avoiding illegal operation
        if(epsx ~= 0.0)
        epsx = epsx / ( countx * Calculatem0(myimage, w, x, y )/double(w));
        end
        if(epsy ~= 0.0)
        epsy = epsy / ( county * Calculatem0(myimage, w, x, y )/double(w));
        end
    end

% iterative correction if absolute of any epsilon is >0.5
% breaks after maximum of 100 iteration
count = 0;
while (((abs(epsx) >= 0.5) | (abs(epsy) >= 0.5)) & count < 100)

    % refined centroid coordinates
    xx = ccn(1) + round(epsx);
    yy = ccn(2) + round(epsy);

    % same as before with redefined coordinates
    if (xx >= 1 & yy >= 1 & xx <= size(myimage,1) & yy <= size(myimage,2))
        cutPart = myimage(xx - r : xx + r , yy - r : yy + r)
        epsx = sum(sum(Mask(:,:,1) .* double(cutPart)));
        epsy = sum(sum(Mask(:,:,2) .* double(cutPart)));

        if(epsx ~= 0.0)
        epsx = epsx / ( countx * Calculatem0(myimage, w, x, y )/double(w));
        end
        if(epsy ~= 0.0)
        epsy = epsy / ( county * Calculatem0(myimage, w, x, y )/double(w));
        end


    end
    count = count + 1;
end

% create array of final coordinates
```

```
epsi = [xx + epsx, yy + epsy];
BWCentroids(n,:) = epsi;


end

end
```

```
function [ Mask ] = CreateMask( w )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
% Create a wxw-mask for further usage

% kernel radius und square of radius
r = floor(w/2);
r2 = r*r;

% containers for both dimensions
KX = zeros(w,w);
KY = zeros(w,w);

% fill the containers with weights
for i = -r : r
    for j = -r : r
        if((i*i + j*j) <= r2)
            KX(i+r+1,j+r+1) = i;
            KY(i+r+1,j+r+1) = j;
        end
    end
end

% combine containers to a single mask array
Mask = zeros(w,w,2);
Mask(:,:,1) = KX;
Mask(:,:,2) = KY;
end
```

```
function [ m2 ] = Calculatem2( myimage, w , xc , yc)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
%   m2 = 2nd Moment at the origin coordinates
%   myimage = grayscale image to be used
%   w = kernel Size
%   xc,yc = coordinates of kernel centre in the image
```

```
m2 = 0;

r = floor(w/2);

%   Sum up the parts of m2 in each pixel
for i = -r : r
    for j = -r : r
        if (xc >= r & yc >= r & xc <= (size(myimage,1) - r) & yc <= (size(myimag
            m2 = m2 + ((i*i + j*j) * double(myimage(xc + i, yc + j)));
        end
    end
end

%   normalize m2
if (m2 ~= 0)
    m2 = m2/Calculatem0(myimage,w,xc,yc);
end

end
```

```
function [ m0 ] = Calculatem0(myimage,w, xc, yc)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
%   myimage = image; w = kernelSize, xc,yc = coordinates
%   m0 = Momentum 0 at position xc,yc

m0 = double(0.0);

r = floor(w/2);

% sum up elements up m0
for i = -r : r
    for j = -r : r
        if (xc >= r & yc >= r & xc <= (size(myimage,1) - r) & yc <= (size(myimag
            m0 = m0 + double(myimage(xc + i, yc + j));
        end
    end
end

end
```

```
% Segmentierung und Objekterkennung
% Autor: Sergey Belyaev
% Datum: 07.05.2010
```

```
%BlobsWithMatlab führt Segmentierung und Objekterkennung durch.
%Alle verwendete Methoden sind in Matlab integriert.
%Die Funktion gibt die Koordinaten der in einem Bild gefundenen Objekte
%zurück. Rückgabewert ist eine 2xN Matrix, wo die N für die Anzahl der
%Objekte ist. Die erste Spalte enthält die X-Koordinaten, die zweite die
%Y-Koordinaten.
%
%Parameter der Funktion:     Im - Grauwertbild
%                            minArea - minimale Fläche des gesuchten Objektes
%                            maxArea - maximale Fläche des gesuchten
%                                                                  Objektes
%                            NB - Nachbarschaft (8 oder 4)
%                            TH - Wahl der Schwellenwertbestimmung
%                                 1 - fester Wert
%                                 2 - mit 'graythresh' wird optimale
%                                     Swelle berechnet
%                                 3 - 20% des größten Grauwertes im Bild

function [ Blobs ] = BlobsWithMatlab( Im, minArea, maxArea, NB, TH )



    k =  0.25*[1 2 1];
    K = k' * k;                          % Glättungsfilter



    %Binarisieren des Bildes

    if TH == 1,
        threshold = 0.1;
    end
    if TH == 2,
        threshold = graythresh(Im);
    end
    if TH == 3,
        threshold = (double(max(max(Im)))*0.2)/255.0;
    end

    B = im2bw(Im, threshold);
    Bf = imfilter(B, K);
```

```
    %Labeling

    labeledImage = bwlabel(Bf, NB);


    blobMeasurements = regionprops(labeledImage, 'Centroid', 'Area');
%Koordinaten und Flache
    numberOfAllBlobs = size(blobMeasurements,1);
%von Objekten werden
                                                                %ermitte

    blobCentroid = zeros(numberOfAllBlobs, 2);
    blobArea = zeros(numberOfAllBlobs, 1);


    for k = 1 : numberOfAllBlobs
        blobCentroid(k,:) = round(blobMeasurements(k).Centroid);
        blobArea(k) = blobMeasurements(k).Area;                 %Koordinaten
    end                                                         %in ein Vek

    %Zählen von Objekten, die geeignete Fläche haben
    NumberOfBlobs = 0;
    for k=1 : numberOfAllBlobs
        if   blobArea(k) > minArea && blobArea(k) < maxArea, ...
             NumberOfBlobs = NumberOfBlobs + 1;
        end
    end

    %Koordinaten von geeigneten Objekten ablegen
    Blobs = zeros(NumberOfBlobs,2);
    for k=1 : numberOfAllBlobs
        if   blobArea(k) > minArea && blobArea(k) < maxArea, ...
                Blobs(k,:) = blobCentroid(k,:);
        end
    end


end
```

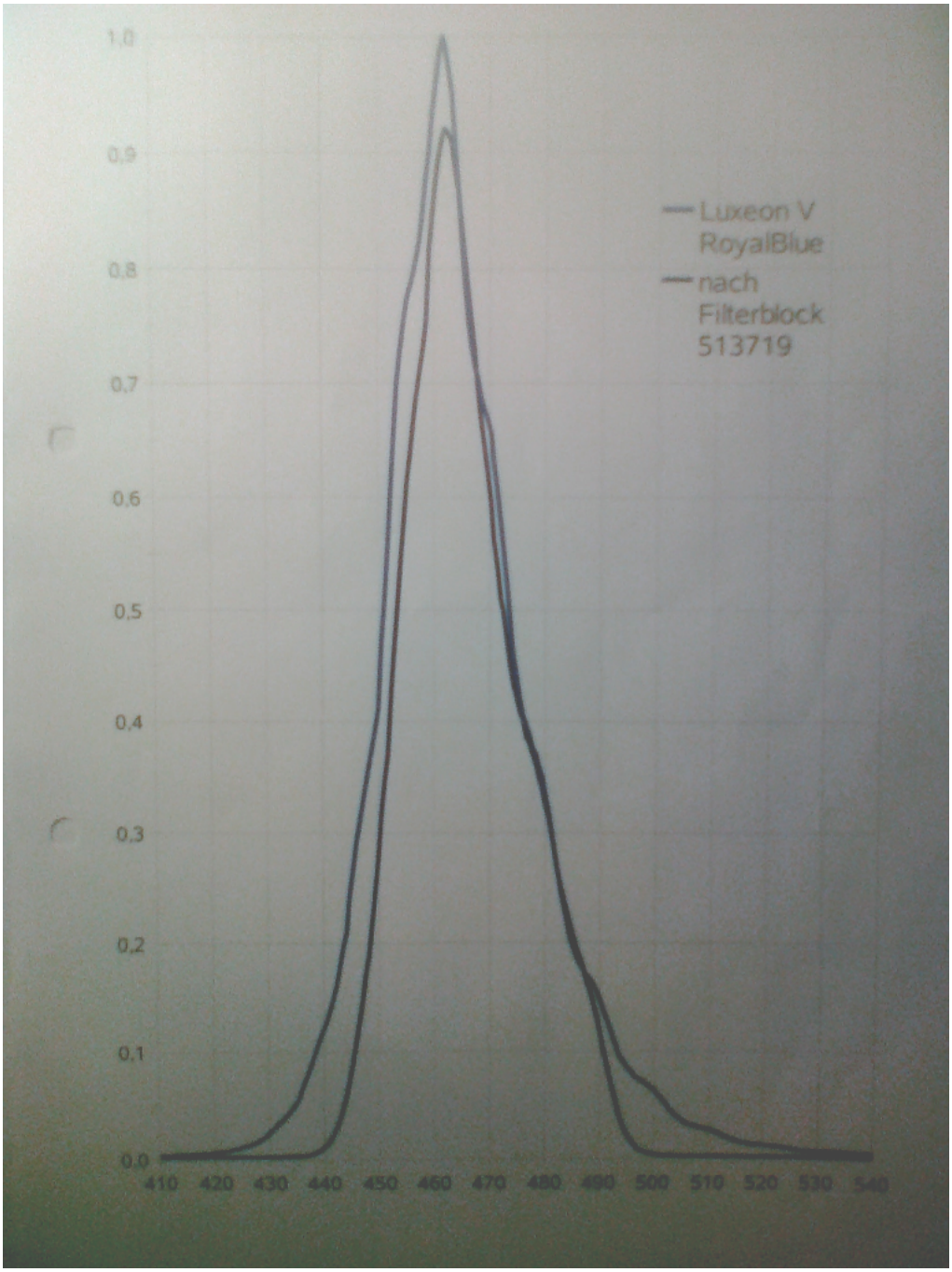


Figure 8: Spektrum der LED

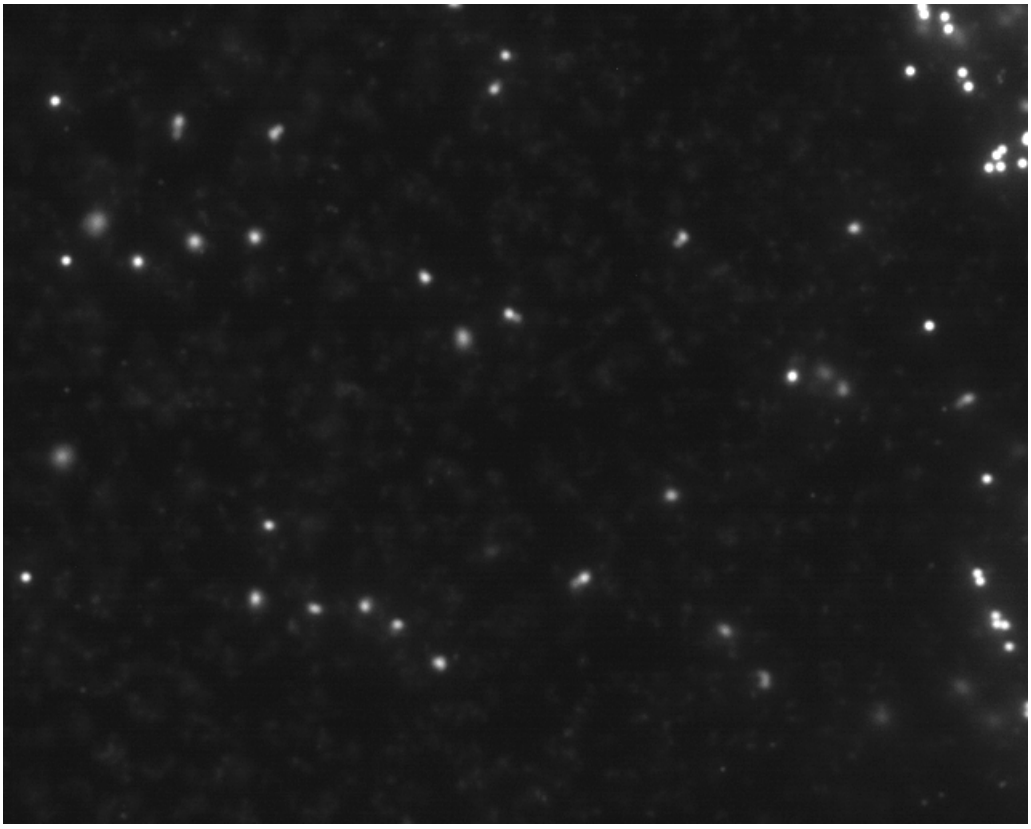


Figure 9: Aufnahme mit neuer Beleuchtung