Freie Universität Berlin
Fachbereich für Mathematik und Informatik
Institut für Informatik

Bachelorarbeit

Implementierung eines Partikelfilters zur Detektion von Fischen in Videobildern

29. September 2015

Bearbeitet von:
Jonas Dettlaff
Bristolstr. 1
13349 Berlin

j.dettlaff@fu-berlin.de

Betreuer: Dr. Tim Landgraf
Zweitgutachter: Prof. Dr. Raúl Rojas

1. Zusammenfassung

Das Detektieren und Tracken von Tieren in Fischschwärmen ist wichtige Vorraussetzung zum Auswerten des Verhaltens. Diese Arbeit beschreibt die Implementierung eines Partikelfilters innerhalb der Biotracker-Applikation, eines Open-Source-Frameworks des Biorobotics Lab. Zum Verarbeiten der Bilder wurde die OpenCV-Bibliothek verwendet. Beim Entwurf wurde Wert darauf gelegt, mit geringem Programmieraufwand auch beliebige andere bewegte Objekte in Videos detektieren zu können. Es wurden außerdem verschiedene Bewertungsfunktionen zum Detektieren von Fischen implementiert und ausgewertet.

Inhaltsverzeichnis

1. Zusammenfassung **3**

2. Einführung **7**

2.1. Das Biorobotics Lab 7

2.2. Das Robofish-Projekt 7

2.3. Das Biotracker-Projekt 8

3. Bisherige Arbeiten **11**

3.1. Partikelfilter im FishTracker-Programm 11

3.2. Der SimpleTracker 11

4. Implementierung **13**

4.1. Anforderungen 13

4.1.1. Performance 13

4.1.2. Zuverlässige Detektion von Fischen 13

4.1.3. Wartbarkeit 13

4.1.4. Open Source 13

4.2. Genutzte Technologien 13

4.2.1. C++ 13

4.2.2. OpenCV 14

4.2.3. Doxygen 14

4.2.4. Qt 14

4.3. Der Algorithmus 14

4.3.1. Background Subtraction 15

4.3.2. Gauß-Filter 15

4.3.3. Der Partikelfilter 17

4.4. Schnittstellen 20

4.4.1. Schnittstelle zum Biotracker-Framework 20

4.4.2. Implementierung anderer Scoring-Funktionen 21

4.5. Entwurf der Klassenarchitektur 22

5. Evaluation **23**

5.0.1. Performance 23

5.0.2. Zuverlässige Detektion von Fischen 23

6. Diskussion **25**
6.0.1. Performance 25
6.0.2. Quantifizierung der Qualität der Fisch-Detektion 25

7. Ausblick **27**
7.1. Verbesserung der Evaluation 27
7.2. Performance 27
7.2.1. Parallelisierung 27
7.2.2. Background Subtraction 27

Anhang **31**

A. Rohdaten: Evaluation der Detektion **33**

B. Selbständigkeitserklärung **35**

2. Einführung

2.1. Das Biorobotics Lab

Das Biorobotics Lab ist ein Teil der Arbeitsgruppe Intelligente Systeme und Robotik. Es ist am Institut für Informatik der Freien Universität Berlin beheimatet. Das Biorobotics Lab hat verschiedene Projekte, die interdisziplinär Biologie und Informatik verbinden.[1] Derzeit liegt der Schwerpunkt auf der Erforschung des Verhaltens und der Kommunikation von Bienen und Fischen.

Für das Projekt Robofish, in dessen Rahmen diese Arbeit entstanden ist, gibt es eine Kooperation mit dem biologischen Bereich der Humboldt-Universität Berlin.

2.2. Das Robofish-Projekt

Das Robofish-Projekt ist Teil des Biorobotics Lab der Freien Universität Berlin. Es wird das Verhalten von Fischschwärmen bzw. -gruppen untersucht. Dies wird mit Guppys und einer robotisch gesteuerten Fischattrappe in einem Becken durchgeführt.

Abbildung 2.1.: Ansicht des Versuchsbeckens mit Fischschwarm aus Perspektive der Kamera

Die Fische werden einzeln oder in Gruppen in ein Becken mit flachem Wasser gesetzt, das von oben mit einer Kamera gefilmt wird. Die Perspektive der Kamera zeigt Abbildung 2.1. Das flache Wasser beschränkt die Bewegung der Fische im Wesentlichen auf das

Zweidimensionale. Dann wird das Verhalten der Fische mit der Kamera aufgezeichnet und kann untersucht werden.

Unter dem Becken befindet sich eine Platte, auf der sich ein Roboter bewegt. Dieser Roboter bewegt mit einem Magneten eine sich im Becken befindliche Fischattrappe. Der Roboter wird mit einer Kamera von unten getrackt. Das Ziel ist hierbei die Akzeptanz der Attrappe durch die Fische. Um mit dem Schwarm schwimmen zu können, muss die Steuerung des Roboters die Position der Fische in Echtzeit erfahren. Auch für andere Auswertungen des Verhaltens der Fische ist ein Tracken der Individuen essenziell, wenn auch nicht unbedingt in Echtzeit.

Die sich über dem Becken befindliche Kamera nimmt Videos im Format 2040 mal 2040 Pixel mit 18 Bildern pro Sekunde auf. Die so erzeugten Videos sind Grundlage für den in dieser Arbeit vorgestellten Algorithmus.

2.3. Das Biotracker-Projekt

Die Biotracker-Applikation ist ein Open-Source-Projekt des Biorobotics Lab. Es wurde mit dem Ziel entwickelt, eine einheitliche Programmierschnittstelle und Benutzeroberfläche für das Biorobotics Lab zu entwickeln, die dem Nutzer Detektion und Tracking in Videos für verschiedene Objekte (Fische, Bienen) und mit verschiedenen Algorithmen, z.B. dem hier vorgestellten Partikelfilter, zur Verfügung stellt.

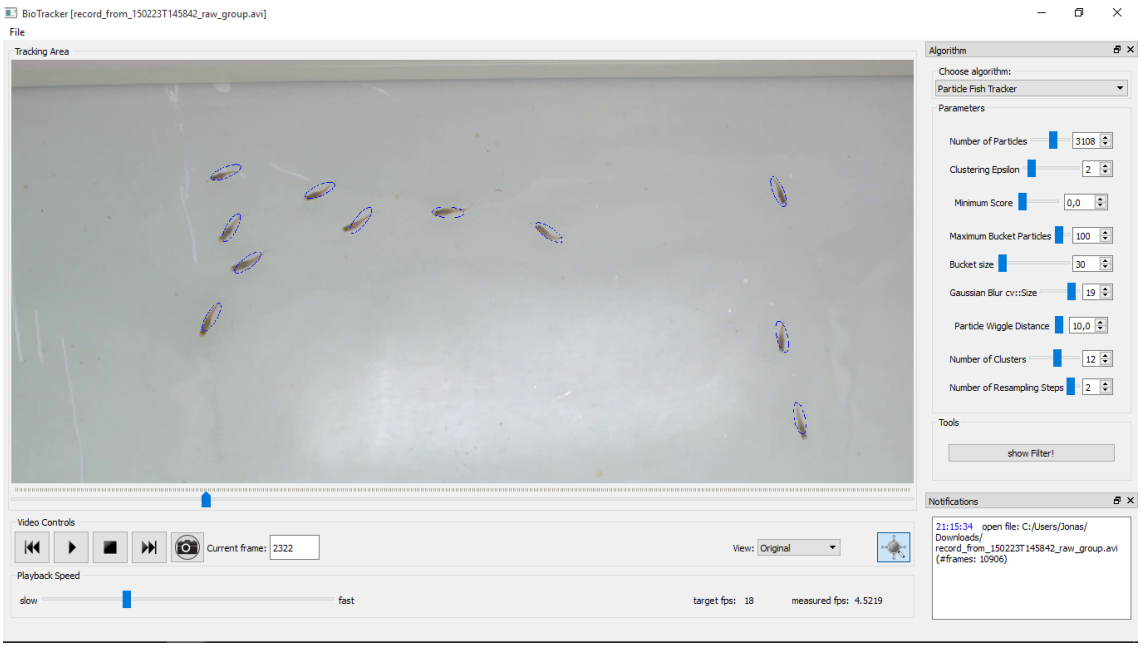

Abbildung 2.2.: Bildschirmfoto der Biotracker-Anwendung

Die Biotracker-Applikation hat eine grafische Benutzeroberfläche. In ihr können Videos und Gruppen von Einzelbildern angezeigt werden. Außerdem gibt sie Entwicklern die Möglichkeit, Parameter-Einstellungen für den gewählten Algorithmus einfach in die GUI einzubinden. Für die implementierten Algorithmen wird eine Schnittstelle zur Einbindung

in die Applikation zur Verfügung gestellt, die den Algorithmus von der GUI und dem Rendern des Videos bzw. der Bilder weitgehend entkoppelt. Die Applikation gibt dem Algorithmus außerdem die Möglichkeit, auf Mauseingaben innerhalb des angezeigten Videobilds zu reagieren.

3. Bisherige Arbeiten

3.1. Partikelfilter im FishTracker-Programm

Von A. Szengel wurde im Rahmen von [12] bereits ein Partikelfilter zum Tracking von Fischen implementiert. Dies geschah innerhalb eines für das Robofish-Projekt entwickelten GUI-Frameworks zum Tracking von Fischen (`FishTracker`). Da diese Applikation durch das generischere Biotracker-Projekt abgelöst wird, war eine neue Implementierung des Detektions-Algorithmus notwendig.

3.2. Der SimpleTracker

Der SimpleTracker ist als Übung in OpenCV entstanden. Er nutzt einige wenige OpenCV-Funktionen, um sich bewegende, geschlossene Konturen in Videos zu erkennen. Damit ist er als Vergleichsalgorithmus für die Evaluation des implementierten Partikelfilters geeignet. Wie der Partikelfilter ist er als Pipeline ausgelegt, deren Eingabe ein Bild des Videos ist. Das Ergebnis wird dann direkt in der Videoansicht gezeigt, indem detektierte Fische markiert werden.

Die Pipeline dieses Algorithmus besteht aus folgenden Stufen:

1. Background Subtraction, ein BackgroundSubtractorMOG2 von OpenCV[3]
2. Der findContours-Algorithmus aus OpenCV[10]
3. Zeichnen des Ergebnisses

Die Stufe 2 wird in dieser Arbeit nicht näher erläutert, es sei hier auf die OpenCV-Dokumentation des findContours-Algorithmus unter [10] verwiesen. Der Background-Subtraction-Algorithmus ist mit dem im Partikelfilter verwendeten identisch und ist dort weiter erläutert.

4. Implementierung

4.1. Anforderungen

4.1.1. Performance

Da das Steuerungsprogramm des mit den Fischen schwimmenden Roboters des Robofish-Projekts für Schwarmexperimente immer die aktuelle Position der Fische kennen muss, ist Echtzeitverhalten gefordert. Dies bedeutet, dass der Algorithmus 18 Einzelbilder pro Sekunde, bei einer Auflösung von 2040 mal 2040 Pixeln verarbeiten muss. Daher bleibt zur Verarbeitung eines Bildes eine Zeit von etwa 56 ms, die zu verarbeitende Bilddatenmenge liegt bei 24 bit pro Pixel bei etwa 225 MB pro Sekunde.

4.1.2. Zuverlässige Detektion von Fischen

Es ist eine zuverlässigere Detektion gefordert, als dies bei den bisher implementierten Algorithmen SimpleTracker und dem Partikelfilter im Fish-Tracking-Programm der Fall ist.

4.1.3. Wartbarkeit

Ein Schwerpunkt wird auch auf leichte Wartbarkeit und Weiterentwicklung des Algorithmus durch andere Entwickler gelegt, daher sollte eine einfache Architektur gewählt sowie eine Dokumentation angelegt werden. Dies ist insbesondere wichtig, da das gesamte BioTracker-Framework als Open-Source-Projekt ausgelegt ist.

4.1.4. Open Source

Da das BioTracker-Framework Open Source ist, ist die Verwendung von Programmierbibliotheken auf kompatible Lizenzen beschränkt.

4.2. Genutzte Technologien

4.2.1. C++

C++ ist eine auf Performance ausgelegte Programmiersprache [11]. Die Menge der zu verarbeitenden Daten und enge Beschränkung der Laufzeit ist daher ein Argument für die Sprache. Außerdem ist die verwendete OpenCV-Bibliothek für C++ verfügbar.

4.2.2. OpenCV

Diese Bibliothek stellt die nötigen Algorithmen zur Bildmanipulation zur Verfügung. OpenCV ist ein Open-Source-Projekt. Es dient dem performanten Verarbeiten von Bildern und Videos und stellt eine Vielzahl an Algorithmen zu diesem Zweck bereit. Die OpenCV-Bibliothek steht für die Programmiersprachen C, C++ und Python zur Verfügung. [2]

Für das Biotracker-Framework und den für diese Arbeit implementierten Algorithmus wurde die C++-Version verwendet.

In dieser Arbeit wurden außer der OpenCV-Datenstruktur zur Speicherung von Bildern und Matrizen cv::Mat [4], die Implementierungen des k-means-Algorithmus [5], des Mixture-of-Gaussians-Background-Subtraction-Algorithmus [3] und der Gauß-Filterung [8] verwendet.

4.2.3. Doxygen

Um die Anforderung an Wartbarkeit und einfache Weiterentwicklung zu erfüllen, musste eine umfassende Dokumentation erstellt werden. Diese Dokumentation deckt insbesondere die Schnittstellen der Komponenten untereinander ab. Um diese Dokumentation zu erstellen, wird im Biotracker-Projekt Doxygen verwendet. Doxygen ermöglicht das Dokumentieren direkt im Code, wodurch Nähe zwischen Code und Dokumentation erzeugt wird.

4.2.4. Qt

Die Biotracker-Applikation nutzt Qt, die eingebundenen Algorithmen werden von der Nutzung dieses GUI-Frameworks allerdings weitestgehend entkoppelt. Lediglich die Parameter-Einstellungen sind Qt-Elemente. Außerdem geschieht das Reagieren auf Nutzereingaben im Videofenster mittels Qt-slots.

4.3. Der Algorithmus

Die Verarbeitung eines Bildes erfolgt schrittweise durch eine Pipeline-Architektur. Grob lässt sich diese Pipeline in drei Teilbereiche spalten:

1. **Vorbearbeitung des Bildes** 4.4.1 Background Subtraction, 4.4.2 Gauss-Filterung
2. **Partikelfilter** 4.4.3 Seeding, 4.4.4 Scoring, 4.4.5 Resampling, 4.4.6 k-means-Clustering
3. **Ergebnisse sammeln und anzeigen** Aufbereiten der Ergebnisdaten und Anzeige in der graphischen Benutzeroberfläche, direkt in der Videoansicht.

Ein typisches Originalbild des Beckens ist in Abbildung 4.1 gezeigt, am Beispiel dieses Bildes wird im Folgenden die Pipeline des Algorithmus schrittweise vorgestellt.

Abbildung 4.1.: Typisches Eingabebild für den Algorithmus

4.3.1. Background Subtraction

Wir gehen von der Annahme aus, dass das Versuchsbecken einen relativ statischen Hintergrund darstellt, vor dem sich die Fische bewegen und nicht länger als wenige Sekunden stationär bleiben. Daher bietet sich Background Subtraction an.

Es wird hier der in OpenCV implementierte Mixture-of-Gaussians-Algorithmus `cv::BackgroundSubtractorMOG2` verwendet, diese Klasse ist unter [3] dokumentiert. Background Subtraction baut ein Bild des Hintergrunds auf, das vom Eingabebild subtrahiert wird. [7] So entsteht das Vordergrundbild, hier als schwarz-weißes Bild der Fische. Dadurch, dass ein Hintergrundbild aufgebaut werden muss, dauert es einige (ca. 30) Frames, bis ein Vordergrundbild stabil sichtbar wird. Das Hintergrundbild wird langsam weiter angepasst, sodass langsame Schwankungen in z.B. der Helligkeit des Videos mit in den Hintergrund eingehen. Es musste eine Balance gefunden werden, damit nicht auch kurz stillstehende Fische als Hintergrund erfasst werden und verschwinden. Als guter Parameter hat sich ein `nMixtures`-Wert von 30 bis 60 erwiesen.

Abbildung 4.2 zeigt das Beispielbild nach Durchführung der Background Subtraction. Fische sind scharf abgegrenzte weiße Objekte vor schwarzem Grund.

4.3.2. Gauß-Filter

Da der Partikelfilter eine gewisse Unschärfe der Bewertungsfunktion benötigt, hat sich die Gauß-Filterung des Bildes als nützlich erwiesen. Sie lässt die scharf abgegrenzten Fische im Bild zu etwa ellipsenförmigen Objekten mit hellweißem Zentrum und grauem, ins Schwarze verlaufendem Rand werden. Dadurch entsteht im Bild ein kontinuierlicher Gradient der Helligkeit zum Zentrum des Fisches hin.

Wir nutzen die in OpenCV implementierte Gauß-Filterung `cv::GaussianBlur`, unter

Abbildung 4.2.: Vergrößertes Beispielbild nach der Background Subtraction.

[8] dokumentiert.

Die Gauß-Filterung ist ein Algorithmus zum Weichzeichnen eines Bildes. Dabei wird der neue Wert eines Pixels aus seinem alten sowie den alten Werten der Pixel in seiner Umgebung gebildet. Hierfür wird eine zweidimensionale Matrix (Kernel) gebildet, deren Werte nach Art der Normalverteilung zum Zentrum steigen. Nutzt man nun diese Matrix, um den Wert eines Pixels mit den Werten seiner Nachbarn gewichtet nach den Werten in der Matrix zu mischen, erhält man den neuen, gefilterten Wert des Pixels.

Beispiel: Bei einer Kernel-Matrix von $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ ist der neue Wert eines Pixels ein Gemisch aus seinen direkten Nachbarn, sowie dem dreifach gewichteten eigenen Wert.

Das so entstandene Bild ist die Eingabe und Bewertungsgrundlage des eigentlichen Partikelfilters.

Abbildung 4.3.: Vergrößertes Beispielbild nach der Gauß-Filterung.

4.3.3. Der Partikelfilter

Der Partikelfilter ist ein Monte-Carlo-Verfahren, das im Seeding zunächst sogenannte Partikel über das Bild streut. Diese Partikel stellen jeweils eine Hypothese dar, dass sich an dem von ihnen repräsentierten Punkt tatsächlich ein Fisch befindet. Diese Hypothese wird anschließend durch das Scoring anhand einer Funktion bewertet, die beurteilt, wie gut die Hypothese zutrifft. Im Resampling wird dann durch wiederholtes Streuen und Bewerten erreicht, dass sich die Partikel um die tatsächlichen Fischpositionen herum anhäufen. Im Clustering werden dann mittels k-means-Algorithmus die Clusterzentren gefunden, die dann die gefundenen Fische repräsentieren.

Partikel

Die Partikel werden in Abbildung 4.6 zwar nur als Punkte im Zweidimensionalen dargestellt (grün markiert), sie haben allerdings bei den meisten eingesetzten Bewertungsfunktionen (s. unten "Scoring") mehr Dimensionen. Je mehr Dimensionen von einer Bewertungsfunktion in die Bewertung mit einbezogen werden, desto mehr Informationen werden über die detektierten Fische herausgefunden. So kann der ParticleBrightnessObserver lediglich die x- und y-Koordinaten der Fische bestimmen, der LineScorer zusätzlich den Winkel und der EllipseScorer zusätzlich Länge und Breite des Fisches. Im Folgenden eine Liste mit den Dimensionen eines Partikels und welche Bewertungsfunktionen diese Dimension nutzen.

- **x-Koordinate** Die x-Koordinate des Partikels im Bild. (ParticleBrightnessObserver, LineScorer, EllipseScorer)
- **y-Koordinate** Die y-Koordinate des Partikels im Bild. (ParticleBrightnessObserver, LineScorer, EllipseScorer)
- **Winkel** Der Winkel des Partikels im Bild, dies umfasst lediglich 0 bis 180 Grad, da die Bewertungsfunktionen nicht zwischen Kopf und Schwanz des Fisches unterscheiden. (LineScorer, EllipseScorer)
- **Ellipsenlänge** Die Länge des Partikels. (EllipseScorer)
- **Ellipsenbreite** Die Breite des Partikels. (EllipseScorer)

Seeding

Dies ist der erste Schritt des Partikelfilters. Ein Parameter des Algorithmus ist die Anzahl der zu streuenden Partikel, $n_{particles}$. Im ersten Bild des Videos werden zunächst diese $n_{particles}$ Partikel zufällig gleichverteilt über das gesamte Bild gestreut. In folgenden Bildern werden nur 10 Prozent der Partikel neu zufällig über das Bild verstreut.

Ziel ist nun, dass durch das Scoring und Resampling erreicht wird, dass sich die so gestreuten Partikel um die Fische herum clustern.

Scoring

Wie bereits beschrieben, wurden verschiedene Bewertungsfunktionen implementiert, die verschieden komplex sind und auch qualitativ unterschiedliche Ergebnisse liefern. Die Bewertungsfunktion wird für jeden Partikel auf dem Bild aufgerufen und setzt sein Score-Attribut. Hierbei ist das Ziel, Partikeln, die gute Hypothesen zur Position eines Fisches repräsentieren, eine hohe Score zuzuordnen.

Die einfachste Scoring-Funktion ist der **ParticleBrightnessObserver**, der aus A. Szengels Arbeit [12] übernommen ist. Diese Funktion nutzt nur 2 Dimensionen der Partikel: x- und y-Koordinate im Bild. Da das Zentrum eines Fisches durch die Gauß-Filterung hell ist und die Helligkeit nach außen abnimmt, wird als Score die Helligkeit des Pixels an der Position des Partikels verwendet.

Die Klasse **LineScorer** implementiert eine Scoring-Funktion, die zusätzlich den Winkel der Partikel bewertet. Dazu wird nicht nur die Helligkeit eines Pixels als Score gesetzt, sondern alle Pixelhelligkeiten auf einer Linie im Winkel des Partikels werden aufsummiert und als Score verwendet. Da ein Fisch eine längliche Form hat, haben daher Partikel, deren Winkel in Richtung Kopf-Schwanz eines Fisches liegt, eine höhere Bewertung. Um durch alle Pixel auf der Linie zu iterieren, wird die OpenCV-Klasse `LineIterator` [6] verwendet.

Die komplexeste Scoring-Funktion ist **EllipseScorer**. Diese Funktion basiert direkt auf der Beobachtung, dass sich Fische im bearbeiteten Bild als Ellipsen darstellen, die zur Mitte hin heller werden. Es werden Partikel direkt auf Basis dieses Modells bewertet. Ein Partikel stellt eine Ellipse auf dem Bild dar. Wir bilden zu dem Bild nun zuerst die Gradienten in x- und y-Richtung. Dies funktioniert wie folgt:
Der Sobel-Filter ist eine Algorithmus zum Ermitteln der Gradienten in x- und y-Richtung. Zunächst wird das Bild I Pixelblock für Pixelblock mit einem Kernel K gefaltet:

$$\text{Für die x-Richtung: } K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Für die y-Richtung: } K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Danach erhalten wir 2 Matrizen $G_x = K_x * I$, $G_y = K_y * I$, die jeweils die Gradienten in x- und y-Richtung enthalten. [9] Hieraus können wir für beliebige Koordinaten (x, y) den Gradientenvektor aus $G = \sqrt{G_x^2 + G_y^2}$ bestimmen.

Abbildung 4.4 zeigt eine graphische Darstellung eines Partikels. Die blaue Ellipse stellt die durch den Partikel repräsentierte hypothetische Fischposition dar. Die weißen Punkte auf der Ellipse sind die Stellen, die von der Bewertungsfunktion zur Bewertung des Partikels genutzt werden. An jeder dieser acht Stellen wird der Gradientenvektor bestimmt. Dann wird bewertet, wie genau dieser Vektor in die Mitte der Ellipse zeigt. So entsteht die Score eines Partikels.

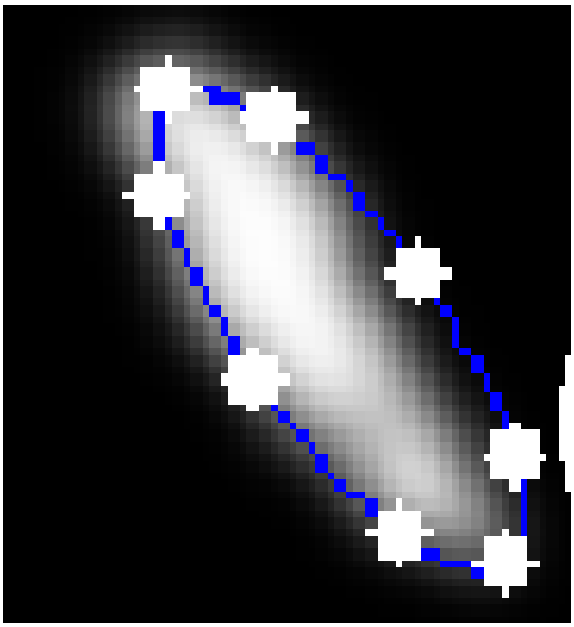

Abbildung 4.4.: Beispiel für ein Partikel (blaue Ellipse und weiße Punkte).

Resampling

Das Resampling hat zum Ziel, dass sich die Partikel um die Fische herum clustern, also ansammeln, sodass lokale Maxima der Partikeldichte auf den Positionen der Fische entstehen. Es startet mit den alten Partikeln und erstellt aus deren Daten neue. Dazu wird $n_{particles}$ oft ein zufälliges der alten Partikel gewählt. Dabei sind die Partikel proportional zu ihrer Score gewichtet, sodass hoch bewertete Partikel (wahrscheinlich) oft und schlecht bewertete (wahrscheinlich) nie gewählt werden. Von einem gewählten Partikel wird eine Kopie erstellt, die allerdings in allen Dimensionen normalverteilt leicht verändert wird. Die so erstellten Kopien sind die neuen Partikel. Nach jedem Resampling-Schritt werden die neuen Partikel bewertet.

Ein Problem beim Resampling ist die Tendenz, dass nur ein Cluster von Partikeln auf dem gesamten Bild entsteht, also nur ein Fisch gefunden wird. Daher muss vermieden werden, dass zu viele Partikel nah beieinander entstehen. Dies wird in diesem Algorithmus durch die Klasse `GridParticleBuckets` erreicht. Diese Klasse teilt das Bild in Quadrate und verhindert, dass in einem dieser Quadrate mehr als eine bestimmte (einstellbare) Anzahl an Partikeln erzeugt werden. Sollte diese Situation entstehen, so wird der Partikel stattdessen wie beim ursprünglichen Seeding neu gestreut.

Wird das Video abgespielt, so erhält der Algorithmus nacheinander sehr ähnliche Bilder, da sich Fische im Vergleich zur Bildrate der Kamera sehr langsam bewegen. Daher wird nach dem ersten Bild beim Abspielen kein neues Seeding mehr durchgeführt, sodass die Partikel aus dem vorherigen Bild ins nächste übernommen werden. Dadurch muss der Algorithmus nicht bei jedem Bild mit völlig gleichverteilten Partikeln neu anfangen, sondern hat bereits vom ersten Resampling-Schritt an viele gut bewertete Partikel. Dadurch entstehen schon bei nur 1 bis 2 Resampling-Schritten stark ausgeprägte Partikelcluster auf den

Fischen. Abbildung 4.5 zeigt solche Cluster von Partikeln, nach nur einem Resampling-Schritt.

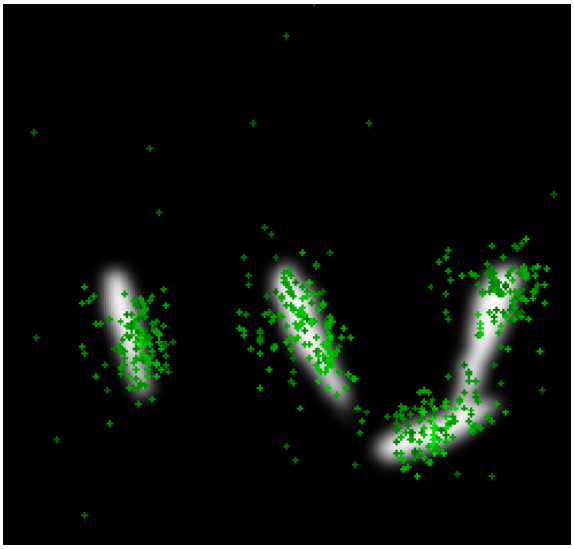

Abbildung 4.5.: Vergrößertes Beispielbild mit Partikeln (grün markiert) nach dem Resampling.

k-means-Clustering

Im letzten Schritt des Algorithmus werden die Mittelwerte aller Partikel in einem Cluster bestimmt. Diese so entstehenden Mittelwerte sollen dann die Positionen, Winkel, Länge und Breite der Fische sein. Um dies tun zu können, müssen zuerst die Partikel den Clustern zugeordnet werden. Dazu verwenden wir den von OpenCV bereitgestellten k-means-Algorithmus [5]. Dabei werden die Partikel nur nach ihren x- und y-Koordinaten geclustert, die anderen Dimensionen werden beim Clustering ignoriert.

Sind die Partikel Clustern zugeordnet, werden die Koordinaten aller Partikel im Cluster gemittelt (nun inklusive Winkel, Länge und Breite). Diese Mittel werden dann als Ergebnis auf das Anzeigebild gezeichnet, wie in Abbildung 4.6 gezeigt.

4.4. Schnittstellen

4.4.1. Schnittstelle zum Biotracker-Framework

GUI zur Einstellung der Parameter durch den Nutzer

Der Algorithmus stellt eine einfache Qt-GUI zur Einstellung relevanter Parameter durch den Nutzer zur Verfügung.

Einbindung des Algorithmus in die Applikation

Die Einbindung der verschiedenen Algorithmen in das Framework geschieht durch Vererbung und eine Registrierung des Algorithmus im Framework.

Abbildung 4.6.: Vergrößerte fertige Ausgabe des Algorithmus, mit markierten Ellipsen, wo Fische detektiert wurden (Clusterzentren). Dies ist mit der Bewertungsfunktion "EllipseScorer" erstellt.

Um den Algorithmus in die Applikation einzubinden, muss die Hauptklasse des Algorithmus von der Klasse `TrackingAlgorithm` erben. Dies ist eine abstrakte Klasse, sodass die Hauptklasse des Algorithmus mindestens diese Interface-Methoden zur Verfügung stellen muss:

- `void track(uint frameNumber, cv::Mat& frame)` Dies ist die zentrale Methode des Algorithmus, die die Detektion durchführt.
- `void paint(ProxyPaintObject&, View)` Diese Methode gibt dem Algorithmus die Möglichkeit, das Ergebnis in der Videoansicht zu zeichnen.

4.4.2. Implementierung anderer Scoring-Funktionen

Die Scoringfunktion des Partikelfilters ist leicht austauschbar. Eine Scoring-Funktion muss folgendes Interface zur Verfügung stellen:

- Einen parameterlosen Konstruktor.
- `void setData(cv::Mat image, ParticleParams* params)` Die Methode, mit der das Bild, auf dessen Basis die Bewertung der Partikel vorgenommen wird, gesetzt wird. Die Klasse ParticleParams kapselt die in der GUI eingestellten Parameter des Algorithmus.
- `void score(Particle& particle)` Setzt auf Basis dieser Bewertungsfunktion und der zuvor mit setData gesetzten Bild und Parameter die Score des Partikels.

4.5. Entwurf der Klassenarchitektur

Wert wurde hier auf ein klares Interface zwischen den einzelnen Stufen der Pipeline des Algorithmus gelegt. Einzelne Klassen stellen daher im Wesentlichen die Schichten der Pipeline dar, wobei die Klasse `ParticleFishTracker` als Hauptklasse des Algorithmus mehr Aufgaben übernimmt, insbesondere das Übergeben der Daten zwischen den Pipeline-Schichten und den Aufruf der entsprechenden Funktionalität der einzelnen Stufen, sowie das Seeding, Resampling und Zeichnen des Ergebnisses in der Videoansicht.

5. Evaluation

5.0.1. Performance

Die Performance bleibt relativ weit hinter den Anforderungen zurück, die Bildrate liegt auf einem handelsüblichen PC bei etwa 5 Frames pro Sekunde statt der benötigten 20. Damit liegt der Algorithmus hinter dem `SimpleTracker` aber ist performanter als der in [12] implementierte Partikelfilter. Die meiste Zeit verbraucht in allen sinnvollen Szenarien die Background Subtraction.

5.0.2. Zuverlässige Detektion von Fischen

In einem Beispielvideo hat der Algorithmus beim Tracking von 12 Fischen folgende Leistungen erreicht, Rohdaten dazu s. Anhang A1:

Zeit, bis Fische detektiert werden

Es wurde untersucht, wie viele Frames vom Start des Videos vergehen, bis die einzelnen Fische detektiert werden. Die durchschnittliche Anzahl lag hier bei 9 Frames, bei einer Standardabweichung von etwa 3.

Obwohl die Background Subtraction Zeit braucht, um ein zuverlässiges Modell des Hintergrunds zu erstellen, ist diese Zeit sehr kurz und es gibt keine "Ausreißer", die deutlich länger brauchen, bis sie erstmals detektiert werden.

Zeit, die ein Fisch detektiert bleibt

Es wurde untersucht, wie viele Frames vergehen, bis ein Fisch nach einer erstmaligen Detektion in einem Frame nicht mehr gefunden wird. Der Durchschnitt lag bei 22 Frames, bei einer Standardabweichung von 29.

Hier fällt insbesondere die hohe Standardabweichung auf. Es gab von den 12 Fischen 2, die nur einen Frame lang detektiert blieben, und weitere 2, die 82 Frames lang detektiert blieben. Es fällt außerdem auf, dass es oft "Aussetzer" in der Detektion von meist 1 bis 3 Frames gibt.

6. Diskussion

6.0.1. Performance

Die Performance des Algorithmus bleibt hinter den Anforderungen zurück. Dies ist insofern nicht allzu besorgniserregend, da bei der Entwicklung Überlegungen zur Performance nur marginal eine Rolle gespielt haben. Die Background Subtraction hat den größten Laufzeitverbrauch.

6.0.2. Quantifizierung der Qualität der Fisch-Detektion

Die Evaluation hätte schon früher Bestandteil des Entwicklungsprozesses sein sollen, um genauer messbar zu machen, ob Veränderungen an Algorithmus oder Parametern eine Verbesserung bringen.

Ein Vergleich auch mit den anderen Tracking-Algorithmen, nicht nur was die Laufzeit betrifft, wäre hier aufschlussreich gewesen.

7. Ausblick

7.1. Verbesserung der Evaluation

Weitere quantifizierbare Maße der Fisch-Detektion, so zum Beispiel: Wie viel Prozent der Zeit ist ein Fisch detektiert?

7.2. Performance

7.2.1. Parallelisierung

Partikelfilter haben den Vorteil, leicht parallelisierbar zu sein (die Bewertungsfunktion kann parallelisiert werden), sodass der Zeitverbrauch der Bewertungsfunktion sich noch reduzieren lassen sollte.

7.2.2. Background Subtraction

Die Background Subtraction hat den mit Abstand größten Laufzeitverbrauch, sollte sich allerdings relativ problemlos performanter gestalten lassen, indem zum Beispiel das Hintergrund-Modell nicht dauerhaft in jedem Frame aktualisiert wird.

Literaturverzeichnis

[1] *About Biorobotics Lab.* http://biorobotics.mi.fu-berlin.de/wordpress/?page_id=18, Abruf: 2015-09-22

[2] *About OpenCV.* http://opencv.org/about.html, Abruf: 2015-09-22

[3] *cv::BackgroundSubtractorMOG2 Class Reference.* http://docs.opencv.org/master/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html, Abruf: 2015-09-22

[4] *cv::Mat Class Reference.* http://docs.opencv.org/master/d3/d63/classcv_1_1Mat.html, Abruf: 2015-09-22

[5] *OpenCV: Clustering.* http://docs.opencv.org/master/d5/d38/group__core__cluster.html#ga9a34dc06c6ec9460e90860f15bcd2f88&gsc.tab=0, Abruf: 2015-09-22

[6] *OpenCV: Drawing Functions.* http://docs.opencv.org/modules/core/doc/drawing_functions.html#lineiterator, Abruf: 2015-09-22

[7] *OpenCV: How to Use Background Subtraction Methods.* http://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html, Abruf: 2015-09-22

[8] *OpenCV: Image Filtering.* http://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1&gsc.tab=0, Abruf: 2015-09-22

[9] *OpenCV: Sobel Derivatives.* http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html, Abruf: 2015-09-22

[10] *OpenCV: Structural Analysis and Shape Descriptors.* http://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a&gsc.tab=0, Abruf: 2015-09-22

[11] Stroustrup, B. : Evolving a language in and for the real world: C++ 1991-2006. In: *HOPL III Proceedings of the third ACM SIGPLAN conference on History of programming languages*, 2007

[12] Szengel, A. : *Implementierung eines Partikelfilters für das Tracking von Fischen*, Freie Universität Berlin, Bachelorarbeit, 2014

Anhang

A. Rohdaten: Evaluation der Detektion

Tabelle1

Video:
record_from_150223T145842_raw_group.avi
Ab Frame 1000

Zeit (Frames), bis Fisch n gefunden wird

Algorithmen	1	2	3	4	5	6	7	8	9	10	11	12	**Durchschnitt**	
EllipseScorer	7	9	6	15	6	13	13	7	7	7	7	11	**9**	3,16227766

Zeit (Frames), die Fisch n konsekutiv detektiert bleibt

Algorithmen	1	2	3	4	5	6	7	8	9	10	11	12	**Durchschnitt**	
EllipseScorer	1	1	18	10	17	10	82	5	5	18	82	17	**22,16666667**	28,65098232

B. Selbständigkeitserklärung

Selbstständigkeitserklärung

Freie Universität Berlin

Name:	Dettlaff
Vorname:	Jonas
geb.am:	30.11.1990
Matr.Nr.:	4515808

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: _____ Unterschrift: _____