Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Peter Oltmann

Verfolgung von Objekten mit Aktiven Konturen und Partikelfiltern

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Peter Oltmann

Verfolgung von Objekten mit Aktiven Konturen und Partikelfiltern

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 4. Mai 2015

Peter Oltmann

Thema der Arbeit
Verfolgung von Objekten mit Aktiven Konturen und Partikelfiltern

Stichworte
Verfolgung, Objekte, Objektverfolgung, Aktive Konturen, Level Sets, Sparse Field, Konturentwicklung, Partikelfilter, Verdeckungen, partielle Verdeckungen

Kurzzusammenfassung
Diese Arbeit behandelt und entwickelt ein Verfahren zur kamerabasierten Verfolgung von Objekten. Dazu werden mittels Aktiver Konturen die exakten Objektumrisse bestimmt. Durch die Verwendung von Partikelfiltern zur Positionsverfolgung wird ein hoher Grad an Robustheit bei der Objektverfolgung erreicht. Das Verfahren erlaubt selbst im Fall partieller Verdeckungen des verfolgten Objekts eine sehr genaue Prognose der tatsächlichen Objektkonturen. Durch die Implementierung ist eine genaue und robuste Objektverfolgung möglich, bei der gleichzeitig die damit einhergehenden Echtzeitanforderungen eingehalten werden.

Peter Oltmann

Title of the paper
Tracking Objects with Active Contours and Particle Filters

Keywords
tracking, objects, tracking objects, active contours, level sets, sparse field, contour evolution, particle filters, occlusions, partial occlusions

Abstract
This thesis examines and develops a method for camera-based object tracking. For that purpose the exact object shape is determined via Active Contours. By using particle filters to track the position of an object a high degree of robustness is achieved. The procedure allows a very accurate prediction of the contours of the tracked object even if partial occlusions occur. In consequence of the implementation it is possible to realize an accurate and robust object tracking by still fulfilling real-time requirements.

Inhaltsverzeichnis

1. Einleitung

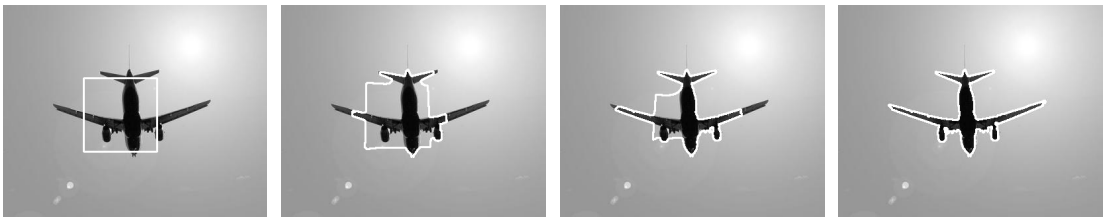
Ein hochaktuelles Thema sowohl in der Forschung als auch in der Industrie ist die Entwicklung autonomer Roboter und Fahrzeuge. Im Forschungsprojekt *Robot Vision* an der Hochschule für Angewandte Wissenschaften Hamburg im Departement Informatik wird an autonomen Assistenzrobotern geforscht und gearbeitet. Diese Roboter werden fortlaufend um neu entwickelte Fähigkeiten erweitert.

Ein Bereich der Forschung konzentriert sich auf die Verfolgung von Objekten mit Hilfe simpler zweidimensionaler Kameras. Dies soll der Interaktion mit Gegenständen, anderen Robotern oder auch Menschen dienen. Die Anwendungsgebiete einer Objektverfolgung sind breit gefächert. So können beispielsweise Roboter oder Autos selbstständig verfolgt oder Gegenstände, die mit einem Roboterarm gegriffen werden sollen, lokalisiert werden. Auch eine Gestenerkennung durch die Verfolgung von Handzeichen ist denkbar.

1.1. Zielsetzung

Ziel dieser Arbeit ist die Entwicklung eines Verfahrens zur kamerabasierten Verfolgung von Objekten in sequentiellen Aufnahmen. Damit soll es möglich sein, fortlaufend die exakten Konturen von sich verformenden Objekten zu bestimmen. Gleichzeitig ist eine robuste Objektverfolgung gegenüber partiellen Verdeckungen, sich schnell bewegenden Objekten und Hintergrundänderungen erforderlich.

Zur exakten Ermittlung von Objektkonturen in Bildern haben sich Aktive Konturen, eine Klasse von Algorithmen zur Bildsegmentierung, als sehr leistungsfähig herausgestellt. Damit ist es möglich, ausgehend von einem simplen Startzustand, etwa einem Rechteck oder Kreis, eine Kontur iterativ in Richtung der Umrisse zusammenhängender Bildregionen zu entwickeln. Die äußerst effektive Methode der Repräsentation und Entwicklung Aktiver Konturen, die sogenannte *Level-Set-Methode*, soll dafür in dieser Arbeit zum Einsatz kommen.



(a) (b) (c) (d)

Abbildung 1.1.: Exakte Bestimmung der Objektumrisse, ausgehend von einer zuvor bestimmten ungefähren Position und Größe

Um die geforderte Robustheit bei der Objektverfolgung zu erlangen, sollen Partikelfilter eingesetzt werden. Partikelfilter sind Methoden zur probabilistischen Schätzung eines unbekannten Systemzustands. Konkret bestünde dieser Systemzustand beispielsweise aus den Positions- und Größenparametern eines zu verfolgenden Objekts. Der große Vorteil von Partikelfiltern liegt in ihrem hohen Grad an Robustheit bei der Zustandsermittlung, was durch das Aufstellen und Bewerten von Zustandshypothesen erreicht wird.

Abbildung 1.1 zeigt ein Beispiel einer Aktiven Kontur, die iterativ in Richtung der Objektumrisse entwickelt wird. Als Ausgangslage hierfür dient die zuvor bestimmte ungefähre Position und Größe des Objekts.

Weiterhin soll im Rahmen der vorliegenden Arbeit eine Software entwickelt werden, die dieses Verfahren zur Objektverfolgung implementiert. Damit soll es möglich sein, beliebige sequentielle Aufnahmen eines Objekts entgegenzunehmen und dessen Konturen robust zu verfolgen. Auch die mit einer Objektverfolgung einhergehenden Echtzeitanforderungen sollen eingehalten werden.

1.2. Gliederung

Die vorliegende Arbeit ist in die im Folgenden aufgelisteten Kapitel gegliedert.

Kapitel 1 beschreibt die Motivation und Zielsetzung dieser Arbeit.

Kapitel 2 gibt einen Überblick über bisher veröffentlichte verwandte Arbeiten.

Kapitel 3 erläutert die Identifikation von Bildregionen mittels Histogrammen, Flächenmerkmalen und Fourier-Deskriptoren.

Kapitel 4 beschreibt die Theorie der Aktiven Konturen und wie es damit möglich ist, die exakten Umrisse zusammenhängender Bildregionen automatisch zu bestimmen. In dem Zusammenhang wird die *Level-Set-Methode* zur Repräsentation und Entwicklung Aktiver Konturen behandelt.

Kapitel 5 befasst sich mit Partikelfiltern zur robusten Schätzung unbekannter Systemzustände. Es wird ein Partikelfilter zur Objektverfolgung vorgestellt, mit dem eine robuste Positions- und Größenbestimmung gesuchter Bildregionen möglich ist.

Kapitel 6 stellt den in dieser Arbeit entwickelten Algorithmus zur Objektverfolgung vor. Zunächst werden die Anforderungen und Rahmenbedingungen aufgestellt und ein Überblick über den Ablauf der Objektverfolgung gegeben. Des Weiteren wird die detaillierte Umsetzung und Implementierung der Aktiven Konturen, des Partikelfilters zur Objektverfolgung sowie der Ausnahmebehandlung einschließlich des Umgangs mit partiellen Verdeckungen behandelt.

Kapitel 7 stellt die Ergebnisse dieser Arbeit zusammen und analysiert diese.

Kapitel 8 fasst die Resultate dieser Arbeit zusammen und zieht ein Fazit.

2. Verwandte Arbeiten

Dieses Kapitel behandelt verwandte Arbeiten. Dazu wird ein Überblick über relevante und grundlegende Literatur gegeben.

Eine grundlegende Behandlung der Aktiven Konturen findet sich in [BI98]. Darin werden vor allem die geometrischen Grundlagen erläutert. Außerdem werden Ansätze zur probabilistischen Modellierung von Aktiven Konturen beschrieben. Dies resultiert in einem ersten Ansatz der Objektverfolgung von Aktiven Konturen in Kombination mit der wahrscheinlichkeitstheoretischen Zustandsschätzung [IB98]. Die Grundlagen der in dieser Arbeit zum Einsatz kommenden Level-Set-Methode zur Umsetzung Aktiver Konturen werden in [Set99] behandelt.

Verschiedene Methoden zur probabilistischen Schätzung unbekannter Systemzustände werden sehr anschaulich in Bezug auf die Lokalisierung von Robotern in [Thr12] analysiert und beschrieben. Die Grundlagen der Partikelfilter finden sich in [GSS93] und [AMGC02].

Ein spezifisches Verfahren zur visuellen Verfolgung von Bildregionen mit Partikelfiltern wurde in [BD11] veröffentlicht. Das dort verwendete Modell dient als Grundlage für das in dieser Arbeit entwickelte Partikelfilter.

Ein besonderer Ansatz zur Kombination von Aktiven Konturen und Partikelfiltern ist die in [RVTY07] präsentierte Methode zur Objektverfolgung, bei der u.a. Echtzeitanforderungen im Fokus stehen. Darin wird vorgeschlagen, wie der hohe Rechenaufwand Aktiver Konturen umgangen werden kann.

Ein weiterer Ansatz ist das in den Arbeiten [LST13, LST11, DSYT10] entwickelte Verfahren. Dort wird durch die Verfolgung von Objekten mit Partikelfiltern und Aktiven Konturen versucht, deren dreidimensionale Pose zu bestimmen. Dabei werden Ansätze für die Detektion und den Umgang mit partiellen Verdeckungen vorgestellt.

Des Weiteren sind die für diese Arbeit herangezogenen Werke über die Grundlagen der Bildverarbeitung [NFHS11a], [NFHS11b] und [GW08] hervorzuheben.

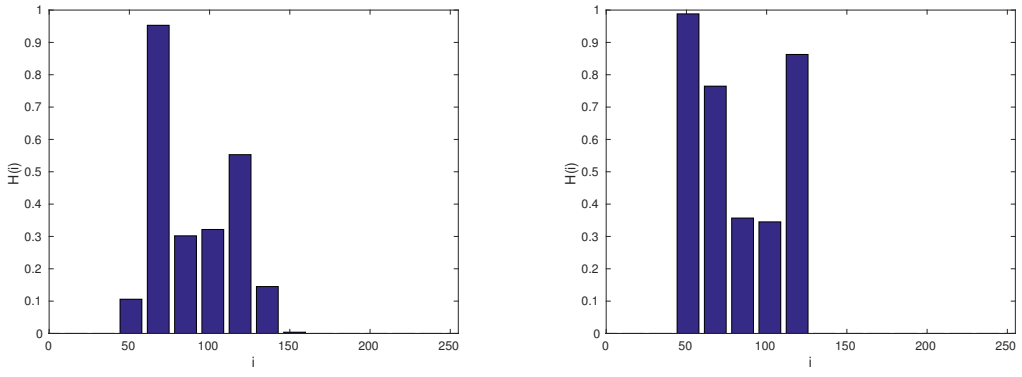
3. Identifikation von Bildregionen

Dieses Kapitel behandelt die in dieser Arbeit verwendeten Methoden zur Identifikation und zum Vergleich von Bildregionen durch Histogramme, Flächenmerkmale oder Konturformen. Sie dienen der Identifikation und Wiedererkennung von Objekten in sequentiellen Aufnahmen. Dazu existieren unterschiedliche Methoden mit dem selben Ziel: die Berechnung eines Wertes, der angibt, inwieweit die beiden verglichenen Bildregionen der Objekte übereinstimmen.

3.1. Histogrammvergleich

Dieser Abschnitt behandelt Methoden für den Vergleich von Histogrammen zur Bestimmung der Ähnlichkeit zweier Bildregionen. Zwecks der Vergleichbarkeit müssen die Histogramme zuvor normiert werden.

Abbildung 3.1 zeigt exemplarisch die Grauwert-Histogramme von zwei unterschiedlichen Bildausschnitten eines Objekts. Diese können mit einer der folgenden Methoden verglichen werden.



(a) (b)

Abbildung 3.1.: Beispiel zweier normierter Histogramme

3.1.1. Korrelation

Der Korrelationskoeffizient [NFHS11b, Kapitel 3.8.3] von zwei Histogrammen kann als Maß für deren Übereinstimmung genutzt werden. Er beschreibt den linearen Zusammenhang zweier statistischer Variablen bzw. deren Stichprobenwerte. Im Fall des Histogrammvergleichs ist dies der Zusammenhang der Histogramm-Werte. In Gleichung 3.1 ist die Berechnung des Korrelationskoeffizienten c_{correl} aufgeführt. Daraus kann ein Distanzmaß d_{correl} (Gleichung 3.2) berechnet werden.

$$
\begin{aligned}
c_{\text{correl}}(H_x, H_y) &= \frac{\sigma_{xy}^2}{\sigma_x \cdot \sigma_y} \\
\sigma_{xy}^2 &= \frac{1}{n} \cdot \sum_{i=1}^{n} (H_x(i) - \bar{H_x}) \cdot (H_y(i) - \bar{H_y})
\end{aligned}
\tag{3.1}
$$

$$
d_{\text{correl}} = 1 - |c_{correl}| \tag{3.2}
$$

In Gleichung 3.1 bis 3.4 ist:

c_{correl}	der Korrelationskoeffizient
c_{bc}	der Bhattacharyya-Koeffizient
$d_{\text{correl}}, d_{\text{bc}}$	Distanzmaß des Korrelations-/Bhattacharyya-Koeffizienten
H_x, H_y	die zu vergleichenden Histogramme
σ_x, σ_x	die Standardabweichungen der Histogramme
σ_{xy}	die empirische Kovarianz der Histogramme
n	die Anzahl der Histogramm-Klassen

Der Korrelationskoeffizient liegt im Wertebereich $[-1, 1]$. Bei einem Wert von $|d_{\text{corell}}| = 1$ sind die Histogramme vollständig linear abhängig. Ein Wert von $c_{\text{correl}} = 0$ hingegen lässt auf keine Korrelation der Histogramme schließen. Ein negativer Wert (negative Korrelation) bedeutet im Falle des Histogrammvergleich ebenfalls keine Übereinstimmung. Der Korrelationskoeffizient der beiden Histogramme aus Abbildung 3.1 beträgt beispielsweise $c_{\text{correl}} = 0.7274$ bzw. $d_{\text{correl}} = 0.2726$.

3.1.2. Bhattacharyya-Koeffizient

Der Bhattacharyya-Koeffizient [Bha43] ist ein Maß für die Ähnlichkeit zweier normierter Histogramme bzw. zweier Mengen von statistischen Stichproben. Dazu wird die Anzahl der Überschneidungen der Histogramm-Klassen in Betracht gezogen. Gleichung 3.3 zeigt die

Berechnung des Bhattacharyya-Koeffizienten. Daraus kann ein Distanzmaß d_{bc} (Gleichung 3.2) berechnet werden.

$$c_{\text{bc}} = \sum_{i=1}^{n} \sqrt{H_x(i) \cdot H_y(i)} \tag{3.3}$$

$$d_{\text{bc}} = 1 - c_{\text{bc}} \tag{3.4}$$

Der Bhattacharyya-Koeffizient liegt im Wertebereich $[0, 1]$. Nimmt c_{bc} einen Wert von 1 an, stimmen die Histogramme vollständig überein. Bei einem Wert von 0 hingegen liegt keine Übereinstimmung vor. Der Bhattacharyya-Koeffizient der beiden Histogramme aus Abbildung 3.1 beträgt beispielsweise $c_{\text{bc}} = 0.8993$ bzw. $d_{\text{bc}} = 0.1007$.

3.2. Vergleich von Flächenmerkmalen

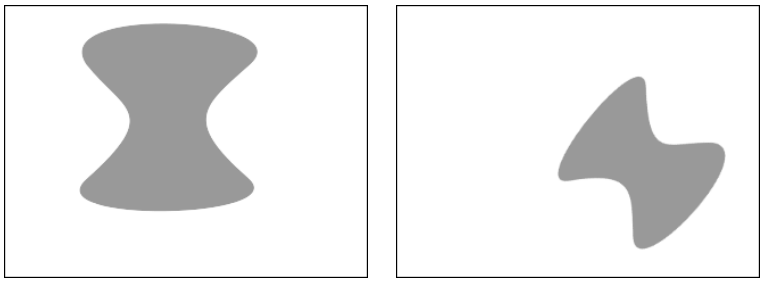
Eine weitere Methode für die Identifikation von Bildregionen ist der Vergleich von Flächenmerkmalen. Diese beschreiben die Eigenschaften einer Bildregion, beispielsweise die eines zu verfolgenden Objekts, und sollten zwecks Wiedererkennung in sequentiellen Aufnahmen invariant gegenüber Translation, Skalierung und Rotation sein.

Abbildung 3.2 zeigt exemplarisch zwei Objekte, deren Bildregionen sich nur durch eine Translation, Skalierung und Rotation unterscheiden. Ein Vergleich von Flächenmerkmalen sollte zum Ergebnis haben, dass es sich um dasselbe Objekt handelt.

Zur Übersicht sind einige in diesem Abschnitt auftretenden Formelzeichen in Tabelle 3.1 aufgeführt.

In den Gleichungen dieses Abschnitts ist:	m_{pq}	ein geometrisches Moment
	μ_{pq}	ein Zentralmoment
	η_{pq}	ein normalisiertes Zentralmoment
	x, y	die Koordinaten des betrachteten Bildpunktes
	$I(x, y)$	der Pixelwert am Bildpunkt (x, y)
	$\sum_{x,y}^{I}$	die Summe über alle Bildpunkte
	$(\bar{x}, \bar{y})$	der Objektschwerpunkt

Tabelle 3.1.: Formelzeichen: Momente



(a) (b)

Abbildung 3.2.: Beispiel von Bildregionen zweier Objekte, die sich nur durch eine Translation, Skalierung und Rotation voneinander unterscheiden

3.2.1. Geometrische Momente

Geometrische Momente sind eine bestimmte Art von Flächenmerkmalen. Sie sind ein Variante zur Beschreibung mechanischer Eigenschaften von Bildregionen. Die geometrischen Momente sind nicht translationsinvariant. Dennoch bilden sie die Basis zur Berechnung invarianter Momente.

Die Definition der geometrischen Momente m_{pq} ist in Gleichung 3.5 aufgeführt. Durch sie kann beispielsweise der Schwerpunkt $(\bar{x}, \bar{y})$ eines Objekts beschrieben werden (Gleichung 3.6).

$$m_{pq} = \sum_{x,y}^{I} I(x,y) \cdot x^p \cdot y^q \tag{3.5}$$

$$(\bar{x}, \bar{y}) = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{3.6}$$

3.2.2. Zentralmomente

Die Zentralmomente definieren sich über den Objektschwerpunkt. Sie sind translationsinvariant, aber nicht rotations- und skalierungsinvariant. Durch eine Normalisierung der Zentralmomente kann die Invarianz gegenüber Skalierung erreicht werden. Allerdings sind auch sie nicht rotationsinvariant.

Im Folgenden sind die Definitionen der Zentralmomente μ_{pq} (Gleichung 3.5) sowie der normalisierten Zentralmomente η_{pq} (Gleichung 3.8) aufgeführt.

$$\mu_{pq} = \sum_{x,y}^{I} I(x,y) \cdot (x - \bar{x})^p \cdot (y - \bar{y})^q \tag{3.7}$$

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \qquad \gamma = \frac{p+q}{2} + 1 \tag{3.8}$$

3.2.3. Hu-Momente

Die Hu-Momente [Hu62] sind sieben Momente, die aus den normalisierten Zentralmomenten berechnet werden können. Sie sind translations-, skalierungs- und rotationsinvariant. Zusätzlich sind die Momente ϕ_1 bis ϕ_6 invariant gegenüber Spiegelung. Das letzte Moment ϕ_7 wechselt bei letzterem sein Vorzeichen. Somit kann es für die Detektion einer Spiegelung verwendet werden.

Im Folgenden sind die sieben Hu-Momente ϕ_1 bis ϕ_7 definiert.

$$\phi_1 = \eta_{20} + \eta_{02} \tag{3.9}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{3.10}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{3.11}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{3.12}$$

$$\begin{aligned}\phi_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) \\ & \cdot [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]\end{aligned} \tag{3.13}$$

$$\begin{aligned}\phi_6 = {} & (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\ & + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})\end{aligned} \tag{3.14}$$

$$\begin{aligned}\phi_7 = {} & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ & - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]\end{aligned} \tag{3.15}$$

Vergleich von Hu-Momenten

Da die Hu-Momente die geforderte Invarianz aufweisen, sind sie miteinander vergleichbar, sodass Objekte, deren Bildregionen durch Hu-Momente beschrieben sind, wiedererkannt

werden können. Um einen Vergleichswert zu erhalten, wird die Distanz zwischen den einzelnen Momenten berechnet. Dazu wird der euklidische Abstand verwendet (vgl. Gleichung 3.16).

$$d_{\text{hu}} = \sqrt{\sum_{i=1}^{7}(|\phi_i| - |\phi_i'|)^2} \tag{3.16}$$

Ein Vergleich der Hu-Momente der beiden Bildregionen aus Abbildung 3.2 liefert einen euklidischen Abstand von $d_{\text{hu}} = 0$.

3.3. Konturvergleich

Eine weitere Methode, um die Form zweier Objekte zu vergleichen, ist der Konturvergleich mit Fourier-Deskriptoren. Sie sind translations-, skalierungs- und rotationsinvariant und erfüllen somit die Anforderungen an die Objektidentifikation. Außerdem können Details der Kontur, die unwesentlich für die Wiedererkennung sind oder durch Störungen verursacht wurden, durch eine Reduzierung der die Kontur beschreibenden Fourier-Koeffizienten herausgefiltert werden.

3.3.1. Fourier-Deskriptoren

Es gibt unterschiedliche Verfahren zur Berechnung von Fourier-Deskriptoren, beispielsweise [ZR72, PF77]. Das in dieser Arbeit verwendete Verfahren zur Berechnung der Fourier-Deskriptoren ist die Variante der komplexen Koordinaten [GW08, Kapitel 11.2.3].

Um die Fourier-Deskriptoren einer Kontur zu erhalten, werden alle Bildpunkte (x_i, y_i) der Kontur nach Gleichung 3.17 als komplexe Zahlen dargestellt. Die Punkte sind dabei nach ihrer Nachbarschaft auf der Kontur entweder mit oder gegen den Uhrzeigersinn sortiert.

$$\mathbf{U} = \begin{pmatrix} x_1 + jy_1 \\ x_2 + jy_2 \\ \dots \\ x_N + jy_N \end{pmatrix} \tag{3.17}$$

Diese komplexen Zahlen $u_i = x_i + jy_i$ werden mit der Diskreten Fourier-Transformation (DFT) aus dem Ortsbereich in den Frequenzbereich transformiert (Gleichung 3.18).

$$a_k = \sum_{i=0}^{N-1} u_i \cdot e^{-j2\pi ki/N} \tag{3.18}$$

Für $k = 0, 1, \ldots, N-1$ besteht das Ergebnis aus N Fourier-Koeffizienten a_k. Diese bilden zusammen den Fourier-Deskriptor $\mathbf{F}$ für die Kontur. Dabei ist zu beachten, dass die Frequenzen f_k nach einer DFT, wie in Gleichung 3.19 dargestellt, geordnet sind.

$$\mathbf{F} = \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{\lfloor \frac{N}{2} \rfloor} \\ \cdots \\ a_{N-1} \\ a_N \end{pmatrix} = \begin{pmatrix} f_0 \\ f_2 \\ \cdots \\ f_N \\ \cdots \\ f_3 \\ f_1 \end{pmatrix} \tag{3.19}$$

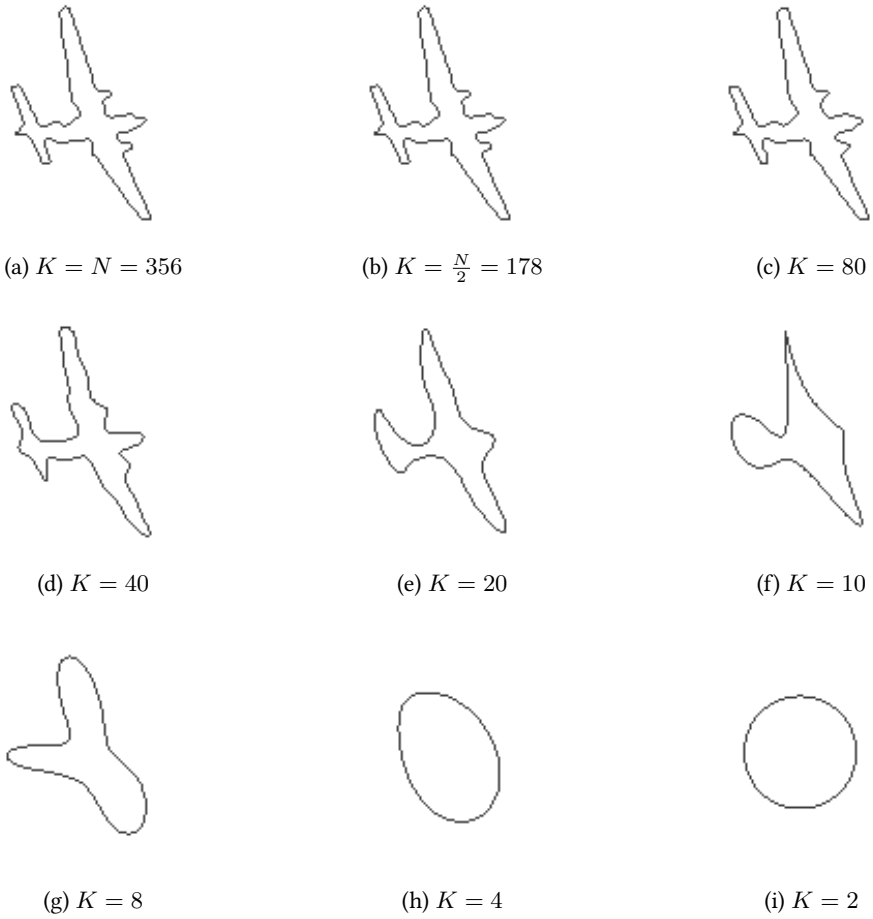
Um die ursprüngliche Kontur zu rekonstruieren, wird die inverse DFT auf die Fourier-Koeffizienten angewendet (Gleichung 3.20).

$$u_i = \frac{1}{N} \sum_{k=0}^{N-1} a_k \cdot e^{j2\pi ki/N} \tag{3.20}$$

Es sei nun angenommen, man verwendet für die Rekonstruktion der Kontur nur die K niedrigsten Frequenzen. Dazu werden alle Frequenzen $f_k = 0$ gesetzt für $k \geq K$. Die hohen Frequenzen, die für die Details der Kontur verantwortlich sind, werden somit herausgefiltert. Mit Hilfe eines solchen *Tiefpass-Filters* können z.B. kleine Unebenheiten aus der Kontur entfernt werden.

Abbildung 3.3 zeigt eine Kontur und deren Rekonstruktion aus ihrem Fourier-Deskriptor. Dazu wurden nur die K niedrigsten Frequenzen in unterschiedlicher Anzahl verwendet. Es ist deutlich zu erkennen, dass eine relativ geringe Anzahl an Fourier-Koeffizienten ausreicht, um die Kontur zu beschreiben. Selbst bei $K = 80$ Koeffizienten (3.3c) ist kein deutlicher Unterschied zur ursprünglichen Kontur (3.3a) zu erkennen. Auch bei $K = 20$ bleibt die

ungefähre Form der Kontur erhalten. Erst bei von $K = 10$ verliert die rekonstruierte Kontur ihre Form.



(a) $K = N = 356$ (b) $K = \frac{N}{2} = 178$ (c) $K = 80$

(d) $K = 40$ (e) $K = 20$ (f) $K = 10$

(g) $K = 8$ (h) $K = 4$ (i) $K = 2$

Abbildung 3.3.: Rekonstruktion einer Kontur aus ihrem Fourier-Deskriptor: für die Rekonstruktion wurden jeweils die K niedrigsten Fourier-Frequenzen verwendet

3.3.2. Vergleich von Fourier-Deskriptoren

Die mittels komplexer Koordinaten erstellten Fourier-Deskriptoren sind nicht im direkten Wege translations-, skalierungs- und rotationsinvariant. Dennoch lassen sich aus ihnen Invarianten konstruieren, weshalb sie für eine Wiedererkennung von Objektkonturen verwendbar sind. Die mathemathischen Zusammenhänge dazu sind in [GW08, Kapitel 11.2.3] beschrieben. Eine detaillierte Behandlung des Verfahrens zum Konturvergleich findet sich in [ZL02].

Für den Vergleich zweier Fourier-Deskriptoren muss Folgendes beachtet werden: Der Fourier-Koeffizient f_0 (vgl. Gleichung 3.19, S. 11) enthält alle Translationsinformationen der beschriebenen Kontur. Alle weiteren Koeffizienten sind translationsinvariant. Aus diesem Grund wird f_0 bei einem Konturvergleich nicht berücksichtigt. Um die Skalierungsinvarianz zu erreichen, werden die Fourier-Koeffizienten mit dem Betrag des Koeffizienten $|f_1|$ normiert, sodass gilt:

$$\mathbf{G} = s\mathbf{F} \quad \Longrightarrow \quad \frac{\mathbf{G}}{|g_1|} = \frac{\mathbf{F}}{|f_1|} \tag{3.21}$$

Weiterhin hat sowohl eine Rotation der Kontur als auch eine Veränderung des Startpunktes nur Einfluss auf die Phase der Fourier-Koeffizienten, sodass für einen Vergleich nur deren Beträge $|f_k|$ verwendet werden.

Als Vergleichswert d_{fd} dient die euklidische Distanz zwischen zwei Fourier-Deskriptoren. Die Berechnung ist in Gleichung 3.22 aufgeführt. Darin werden die Beträge (Rotationsinvarianz) der Fourier-Koeffizienten f_k mit $k > 1$ (Translationsinvarianz) mit dem Koeffizienten f_1 normiert (Skalierungsinvarianz).

$$d_{\text{fd}} = \sqrt{\sum_{i=2}^{K-1} \left(\frac{|f_i|}{|f_1|} - \frac{|f'_i|}{|f'_1|} \right)^2} \tag{3.22}$$

4. Aktive Konturen

Aktive Konturen [BI98] bilden eine Klasse von Algorithmen zur Bildsegmentierung. Mit ihnen lassen sich Bilder in Vorder- und Hintergrund segmentieren und somit Objektumrisse oder -formen extrahieren. Aktive Konturen haben die Eigenschaft, sich an die Umrisse bestimmter Bildregionen anpassen zu können. Deshalb sind sie für eine sehr exakte Konturfindung auch von sich verformenden Objekten geeignet. Ausgehend von einem trivialen Startzustand, etwa einem Kreis oder Rechteck, kann eine Kontur in Richtung ihres idealen Zustands, wie etwa der Umrisse eines Objekts, gebracht werden. Dieser Ablauf wird als Konturentwicklung bezeichnet.

4.1. Konturentwicklung und -energie

Die grundlegende Vorstellung der Konturentwicklung ist, dass auf eine Kontur eine Energie einwirkt. Eine daraus ableitbare Kraft treibt die Kontur in eine bestimmte Richtung. Eine Aktive Kontur definiert sich über diese Energie, die auf unterschiedlichsten Bilddaten basieren kann. Dies können bestimmte Bildmerkmale sein wie Kanten oder auch statistische Werte wie Varianzen bestimmter Bildregionen. Je höher die Energie ist, desto stärker ist auch die Kraft, die auf die Kontur wirkt. Ist also die Energie minimal, so wirkt auch keine bzw. nur eine minimale Kraft auf die Kontur. Dies ist der ideale Zustand einer Aktiven Kontur. Ziel einer Konturentwicklung ist somit die Minimierung der Konturenergie. Abbildung 4.1 veranschaulicht dies anhand eines Beispiels.

Die Energie E einer Kontur kann mathematisch durch ein sogenanntes Energiefunktional beschrieben werden. Eine allgemeine Form davon ist in Gleichung 4.1 aufgeführt. Darin werden die Energiewerte an allen betrachteten Bildpunkten $\mathbf{x}$ aufsummiert. Je nach Beschaffenheit einer bestimmten Energie ist die Energiefunktion F zu definieren. Unterschiedliche Arten von

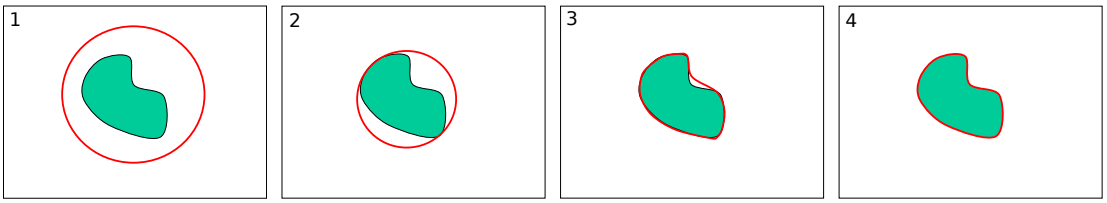


Abbildung 4.1.: Energieminimierung bei Aktiven Konturen: künstliches Beispiel eines Objekts, bei der die Energie an den Kanten minimal ist. **1**: Initialzustand, hohe Energie **2**: Kontur touchiert erstmalig die Kanten des Objekts **3**: Sehr geringe Energie der Kontur **4**: Objektumriss erreicht: minimale Energie. Bildquelle: [Coo08]

Konturenergien und spezielle Definitionen von Energiefunktionalen werden in Abschnitt 4.3 behandelt.

$$E = \sum_{\mathbf{x}}^{I} F(\mathbf{x}) \stackrel{!}{=} 0 \tag{4.1}$$

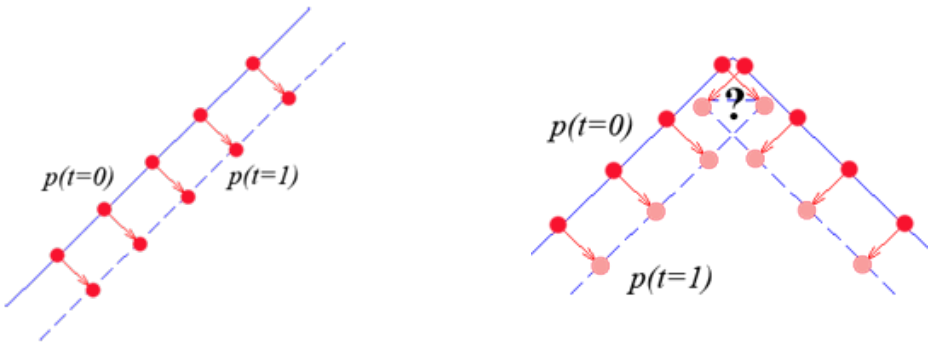
In Gleichung 4.1 und den folgenden Energiefunktionalen wird immer die Summe der Energiefunktion F über alle Bildpunkte $\mathbf{x}$ eines Bildes oder Bildausschnitts I gebildet. Das Betrachten aller Bildpunkte wird in dieser Arbeit mathematisch durch $\sum_{\mathbf{x}}^{I}$ beschrieben.

4.2. Level-Set-Methode

Eine Methode zur Durchführung der Konturentwicklung ist die sogenannte Level-Set-Methode [Set99]. Sie weist gegenüber anderen Verfahren, die auf der Parametrisierung durch *Sample Points* basieren [KWT88, BI98], diverse Vorteile auf.

Ein solches Verfahren entwickelt eine Kontur, indem es ihre *Sample Points* entwickelt (vgl. Abbildung 4.2a). Ein Problem dabei sind beispielsweise neu auftretende oder sich verschiebende Ecken, bei denen die Kontur durch Überschneidungen von Punkten in einen unbekannten Zustand gelangt (vgl Abbildung 4.2b). Außerdem kommt es vor, dass die Anzahl der *Sample Points* bei einer expandierenden Kontur nicht ausreichend ist. Mit der Level-Set-Methode können solche Formveränderungen besser erfasst werden.

Ein weiterer Vorteil ist die Unabhängigkeit von Änderungen in der Topologie. Die Anzahl der Konturen eines Objekts muss nicht vorab bekannt sein und kann sich während einer Konturentwicklung ändern. Wird der Blick auf ein Objekt, beispielsweise durch eine partielle



(a) (b)

Abbildung 4.2.: Konturentwicklung durch *Sample Points*, Bildquelle: [Lom06]

Verdeckung, geteilt, so teilt sich auch die Kontur der Level-Set-Methode. Umgekehrt können zwei Konturen wieder zu einer Kontur verschmelzen (*Split and Merge*).

4.2.1. Oberflächenfunktion

Die Level-Set-Methode wählt einen anderen Weg der Umsetzung von Aktiven Konturen. Die Kontur wird repräsentiert und entwickelt durch eine dreidimensionale Oberflächenfunktion, wodurch eine Parametrisierung der Kontur entfällt. Diese Oberflächenfunktion beschreibt für jeden Bildpunkt den Abstand zur Kontur. Dabei haben Punkte innerhalb der Kontur negative Werte und Punkte außerhalb der Kontur positive Werte. Eine solche Funktion wird auch als *Signed Distance Function (SDF)* bezeichnet. Die Kontur selbst befindet sich an den Bildpunkten, an denen die Oberflächenfunktion den Nullpunkt schneidet. Man spricht in diesem Zusammenhang auch von impliziten Konturen.

Gleichung 4.2 definiert die Kontur am Nullpunkt der Oberflächenfunktion, das sogenannte *Zero Level Set*. Abbildung 4.3 zeigt ein Beispiel einer Konturentwicklung mit der Level-Set-Methode. Eine Video-Demonstration befindet sich in Anhang B.1 (Zeile 2).

$$C_t = \{\mathbf{x} \mid \phi(\mathbf{x}, t) = 0\} \tag{4.2}$$

Die Durchführung der Konturentwicklung wird durch die Modellierung der Energieminimierung (vgl. Abschnitt 4.1) als dynamisches System erreicht, sodass dessen Vorteile genutzt

In Gleichung 4.2 ist:

C_t	die Kontur zum Zeitpunkt t
$\mathbf{x}$	der Bildpunkt mit den Koordinaten (x, y)
ϕ	die Oberflächenfunktion

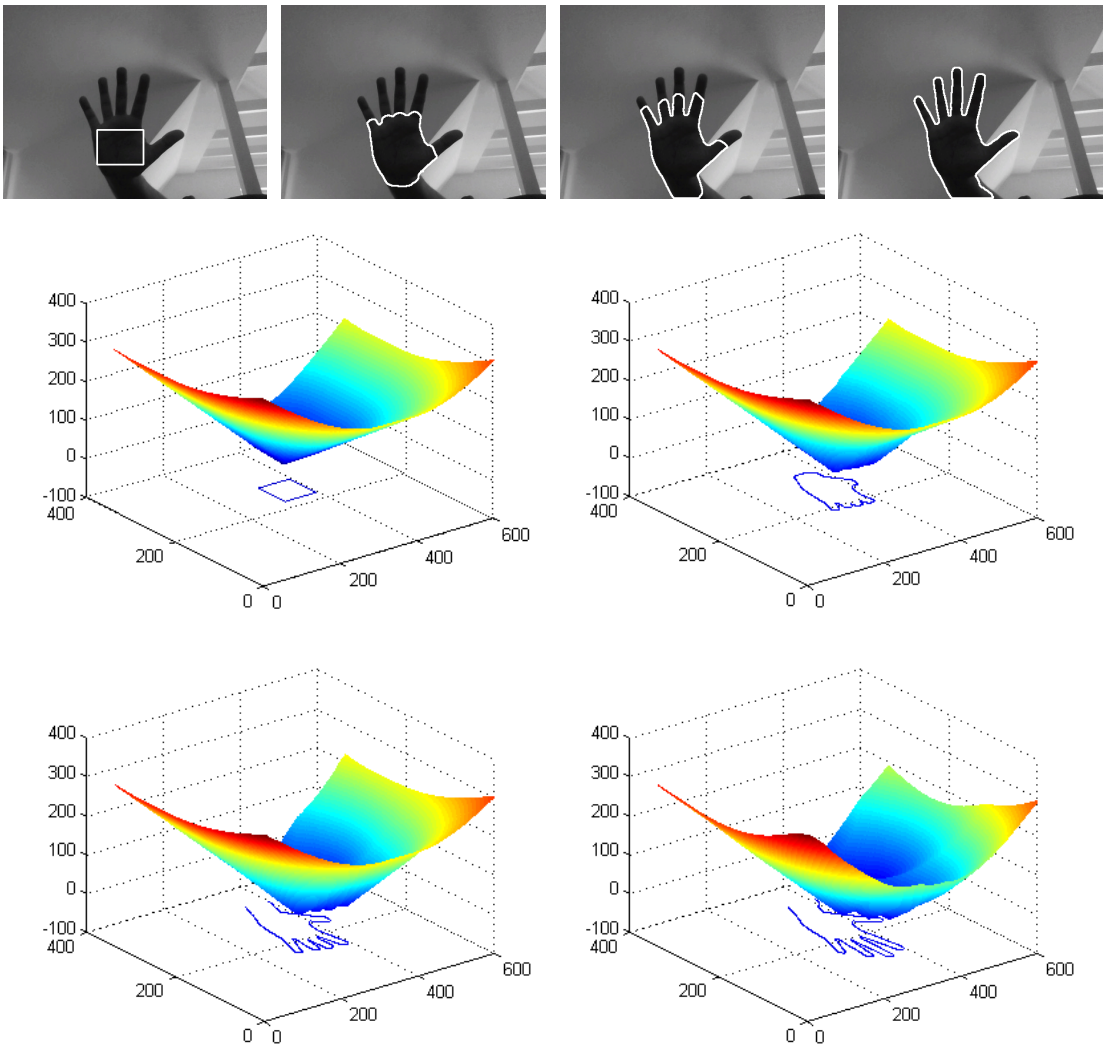


Abbildung 4.3.: Beispiel für die Konturentwicklung mit der Level-Set-Methode: dreidimensionale Oberflächenfunktion mit der impliziten Kontur am *Zero Level Set*

werden können. Die Kontur bzw. die Oberflächenfunktion wird dabei als zeitabhängiger dynamischer Zustand betrachtet. Aus der Energie lässt sich die Bewegungsgleichung des Systems und die auf die Kontur wirkende Kraft ableiten. Die Konturentwicklung kann dann durch das Lösen der Bewegungsgleichung, einer partiellen Differentialgleichung, vollzogen werden.

Die Definition der Bewegungsgleichung für die Konturentwicklung mit der Level-Set-Methode ist in Gleichung 4.3 aufgeführt.

$$\phi_t + F_\nabla \cdot |\nabla \phi| = 0 \tag{4.3}$$

Wird diese Differentialgleichung diskretisiert, lässt sie sich nach Gleichung 4.4 iterativ lösen.

$$\begin{aligned} & \frac{\phi(\mathbf{x}, t + \Delta t) - \phi(\mathbf{x}, t)}{\Delta t} + F_\nabla \cdot |\nabla \phi| = 0 \\ \Leftrightarrow \quad & \phi(\mathbf{x}, t + \Delta t) = \phi(\mathbf{x}, t) - \Delta t \cdot F_\nabla \cdot |\nabla \phi| \end{aligned} \tag{4.4}$$

In Gleichung 4.3 und 4.4 ist:		
	ϕ	die Oberflächenfunktion
	ϕ_t	die partielle Ableitung von ϕ nach der Zeit
	F_∇	die aus der Energiefunktion abgeleitete Kraft
	$\mathbf{x}$	der Bildpunkt mit den Koordinaten (x, y)
	t	der betrachtete Zeitpunkt
	Δt	der diskrete Zeitschritt
	$\|\nabla \phi\|$	der Gradient der Oberflächenfunktion

4.2.2. SDF-Bedingung

Die Oberflächenfunktion ϕ ist eine *Signed Distance Function (SDF)* im euklidischen Raum, sodass der Gradient

$$|\nabla \phi| = 1 \tag{4.5}$$

ist, was vor allem für die Implementierung der Level-Set-Methode eine wünschenswerte Bedingung ist. Dies kann durch die Verwendung des euklidischen Abstands jedes Bildpunktes zur Kontur bei der Initialisierung erreicht werden. Die Bedingung kann jedoch im Verlauf der Konturentwicklung, je nach Approximationsschema der Implementierung, gebrochen werden. Deshalb kann eine Reintialisierung notwendig sein. Es gibt aber auch Wege der Implementierung, die während der Konturentwicklung die Einhaltung der SDF-Bedingung gewährleisten.

Die in Abschnitt 6.2 behandelte *Sparse-Field-Methode* verwendet ein Approximationsschema, mit dem die SDF-Bedingung während der Konturentwicklung eingehalten wird.

4.2.3. CFL-Bedingung

Die Konturentwicklung und damit das iterative Lösen einer partiellen Differentialgleichung basiert auf diskreten Zeitschritten Δt. Diese haben direkten Einfluss auf die Kraft F_∇ (vgl. Gleichung 4.4, S. 18). Die CFL-Bedingung[1] für die Diskretisierung zeitabhängiger partieller Differentialgleichungen besagt Folgendes: die sich entwickelnde Kontur darf nicht mehr als eine Gitterzelle in einem Zeitschritt überschreiten [Set99, Kapitel 6.4.1]. Eine Gitterzelle ist dabei ein diskreter Ortsschritt im kartesischen Gitter des betrachteten dynamischen Systems. Die mathematische Definition der CFL-Bedingung ist in Gleichung 4.6 aufgeführt.

$$\max(\mathbf{F}_\nabla) \cdot \Delta t < \Delta x \tag{4.6}$$

In Gleichung 4.6 und 4.7 ist:		
	$\max(\mathbf{F}_\nabla)$	das Maximum der Kräfte aller betrachteten Bildpunkte
	Δt	der diskrete Zeitschritt
	Δx	der Abstand der Gitterzellen
	c	der Sicherheitsfaktor

Der diskrete Zeitschritt Δt kann nach Gleichung 4.6 berechnet werden. Der Sicherheitsfaktor c sollte dabei im Bereich $[0.75, 0.9]$ gewählt werden [CR07].

$$\Delta t = c \cdot \frac{\Delta x}{\max(\mathbf{F}_\nabla)} \tag{4.7}$$

4.2.4. Krümmung der Kontur

Oftmals wird die Berechnung der Konturenergie bzw. der daraus abgeleiteten Kraft durch die Miteinbeziehung der Konturkrümmung erweitert, um eine Glättung der Kontur zu erzielen [Set99]. Dazu wird die Bewegungsgleichung für die Konturentwicklung (vgl. Gleichung 4.4, S. 18) um die Krümmung κ der Kontur ergänzt (vgl. Gleichung 4.8). Diese wirkt entgegen der

[1] CFL: nach Courant-Friedrichs-Lewy

Kraft F_∇. Dabei sorgt eine starke Krümmung für eine geringere Änderung der Oberflächenfunktion, was zu einer glatteren Kontur führt.

$$\phi(\mathbf{x}, t + \Delta t) = \phi(\mathbf{x}, t) - \Delta t \cdot F_\nabla \cdot |\nabla\phi| + \alpha \cdot \kappa(\mathbf{x}) \tag{4.8}$$

In Gleichung 4.8 ist $\alpha \in [0, 1]$ ein Gewichtungsfaktor, mit dem die Miteinbeziehung der Krümmung κ in die Konturentwicklung und somit die Glättung der Kontur gesteuert werden kann. Je höher α gewählt wird, desto glatter wird die Kontur. Die Berechnung der Konturkrümmung an einem Bildpunkt $\kappa(\mathbf{x})$ ist in Gleichung 4.9 definiert. Darin ist $\phi(\mathbf{x}, t)$ vereinfacht als $\phi(x, y)$ dargestellt.

$$\kappa(\mathbf{x}) = \nabla \frac{\nabla\phi}{|\nabla\phi} = \frac{\phi_{xx}\phi_y^2 - 2\phi_y\phi_x\phi_{xy} + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}} \qquad \text{mit} \qquad \mathbf{x} = (x, y) \tag{4.9}$$

$$\phi_{xx} = (\phi(x+1, y) - \phi(x, y)) - (\phi(x, y) - \phi(x-1, y)) \tag{4.10}$$

$$\phi_{yy} = (\phi(x, y+1) - \phi(x, y)) - (\phi(x, y) - \phi(x, y-1)) \tag{4.11}$$

$$\begin{aligned}\phi_{xy} = \frac{1}{4}[&(\phi(x+1, y+1) - \phi(x-1, y+1)) \\ &- (\phi(x+1, y-1) - \phi(x-1, y-1)]\end{aligned} \tag{4.12}$$

$$\phi_x = \frac{1}{2}(\phi(x+1, y) - \phi(x-1, y)) \tag{4.13}$$

$$\phi_y = \frac{1}{2}(\phi(x, y+1) - \phi(x, y-1)) \tag{4.14}$$

4.3. Spezifische Konturenergien

Dieser Abschnitt behandelt spezifische Konturenergien und die daraus ableitbaren Kräfte. Da die Konturentwicklung in dieser Arbeit mit Hilfe der Level-Set-Methode (vgl. Abschnitt 4.2) umgesetzt wird, sind die Darstellungen und Gleichungen der Energien darauf bezogen. Alle Konturenergien bzw. Kräfte sind allerdings auch in Verbindung mit anderen Verfahren anwendbar.

In der Literatur wird generell zwischen zwei Kategorien von Konturenergien unterschieden, und zwar in kanten- und regionsbasierte Energien. Die kantenbasierten Energien [CKS97] verfolgen die Idee der minimalen Energie an den im Bild detektierten Kanten. Sie nutzen

den Bildgradienten als Grundlage ihrer Energiefunktion. Ihnen gegenüber stehen die regionsbasierten Energien. Sie haben das Ziel, eine Energie zu definieren, die das Bild in maximal homogene Bereiche segmentiert. Die Energiefunktionen werden auf Basis von Bildstatistiken, wie beispielsweise Varianzen oder quadrierten Abweichungen zum Mittelwert, definiert.

Diese Arbeit behandelt die regionsbasierten Energien [CV01, YTW02, LT08] aufgrund ihrer Vorteile bezüglich der Verfolgung von Objekten. Sie führen oftmals trotz verrauschter Bilder oder unklarer Kanten eines Objekts zu einer erfolgreichen Bildsegmentierung. Allerdings kann je nach Beschaffenheit des Objekts auch eine kantenbasierte Energie bessere Ergebnisse liefern. Abbildung 4.4 zeigt eine erfolgreiche Bildsegmentierung mit Hilfe einer regionsbasierten Konturenergie. In Abbildung 4.5 hingegen findet nur eine kantenbasierte Energie erfolgreich die Umrisse des Objekts.

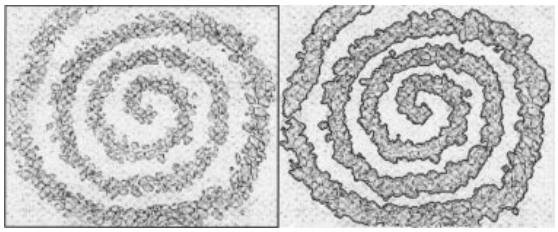


Abbildung 4.4.: Beispiel einer Aktiven Kontur mit einer regionsbasierten Energie: erfolgreiche Bildsegmentierung bei verrauschten Bildern [CV01]

(a) Initiale Kontur

(b) Fehlgeschlagene regionsbasierte Segmentierung

(c) Erfolgreiche kantenbasierte Segmentierung

Abbildung 4.5.: Vergleich von regions- und kantenbasierten Aktiven Konturen: Künstliches Bild eines Objekts mit heterogener Intensität auf einem Hintergrund mit ähnlich heterogener Intensität [LT08]

4.3.1. Uniform Modeling Energy

Die am meisten verbreitete regionsbasierte Konturenergie ist die als *Uniform Modeling Energy* oder auch als *Chan-Vese Energy* bezeichnete Energie [CV01]. Sie basiert auf den quadrierten

Abweichungen zu den Mittelwerten innerhalb und außerhalb der Kontur. Die Definition des Energiefunktionals ist in Gleichung 4.15 aufgeführt.

$$E_{\text{UM}} = \sum_{\mathbf{x}}^{I} F_{\text{UM}}(\mathbf{x}) = \sum_{\mathbf{x}}^{I} \underbrace{(I(\mathbf{x}) - \mu_{\text{in}})^2}_{\text{falls } \phi(\mathbf{x},t) < 0} + \underbrace{(I(\mathbf{x}) - \mu_{\text{out}})^2}_{\text{falls } \phi(\mathbf{x},t) > 0} \tag{4.15}$$

In Gleichung 4.15 und 4.16 ist:		
	ϕ	die Oberflächenfunktion
	F_{UM}	die Energiefunktion
	$I(\mathbf{x})$	der Pixelwert am Bildpunkt $\mathbf{x}$
	$\mathbf{x}$	der Bildpunkt mit den Koordinaten (x, y)
	t	der betrachtete Zeitpunkt
	μ_{in}, μ_{out}	der Mittelwert innerhalb, außerhalb der Kontur

Die Beudeutung des Energiefunktionals ist folgende: es werden die quadrierten Abweichungen der Pixelwerte zum Mittelwert innerhalb bzw. außerhalb der Kontur aufsummiert. Dies führt dazu, dass die Kontur eine minimale Energie aufweist, wenn die Varianzen der durch die Kontur getrennten Bildregionen minimal sind. Somit trennt die Kontur in ihrem idealen Zustand zwei maximal homogene Bildregionen voneinander.

Die für die Bewegungsgleichung der Level-Set-Methode benötigte Kraft F_∇ ist in Gleichung 4.16 aufgeführt. Für eine Herleitung sei auf [CV01] verwiesen.

$$F_\nabla^{\text{UM}}(\mathbf{x}) = -(I(\mathbf{x}) - \mu_{\text{in}})^2 + (I(\mathbf{x}) - \mu_{\text{out}})^2 \tag{4.16}$$

4.3.2. Mean Separation Energy

Eine weitere regionsbasierte Konturenergie ist die als *Mean Separation Energy* bezeichnete Energie [YTW02]. Ähnlich wie die *Uniform Modeling Energy* basiert sie auf den Mittelwerten der Pixelwerte innerhalb und außerhalb der Kontur. Das Energiefunktional der *Mean Separation Energy* ist in Gleichung 4.17 definiert.

$$E_{\text{MS}} = F_{\text{MS}} = -\frac{1}{2}(\mu_{\text{in}} - \mu_{\text{out}})^2 \tag{4.17}$$

Die Energie basiert, anders als die *Uniform Modeling Energy*, nicht auf homogenen Pixelwerten. Sie beruht auf der Annahme, dass Vorder- und Hintergrund des Bildes, also ein Objekt und seine Umgebung, stark voneinander abweichende Mittelwerte aufweisen. Eine Energieminimierung

In Gleichung 4.17 und 4.18 ist:	ϕ	die Oberflächenfunktion
	F_{MS}	die Energiefunktion
	$I(\mathbf{x})$	der Pixelwert am Bildpunkt $\mathbf{x}$
	$\mathbf{x}$	der Bildpunkt mit den Koordinaten (x, y)
	t	der betrachtete Zeitpunkt
	μ_{in}, μ_{out}	Mittelwert innerhalb, außerhalb der Kontur
	A_{in}, A_{out}	die Anzahl der Bildpunkte innerhalb, außerhalb der Kontur

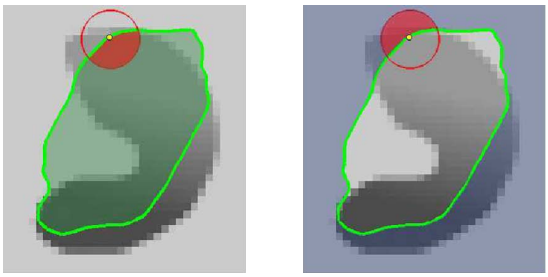
führt zu einer Kontur, die ein Bild in zwei Regionen mit maximal unterschiedlichen Mittelwerten segmentiert.

Die für die Bewegungsgleichung der Level-Set-Methode benötigte Kraft F_∇ ist in Gleichung 4.18 aufgeführt. Für eine Herleitung sei auf [YTW02] verwiesen.

$$F_\nabla^{\text{MS}}(\mathbf{x}) = (\mu_{\text{in}} - \mu_{\text{out}}) \cdot \left(\frac{I(\mathbf{x}) - \mu_{\text{in}}}{A_{\text{in}}} + \frac{I(\mathbf{x}) - \mu_{\text{out}}}{A_{\text{out}}} \right) \tag{4.18}$$

4.3.3. Lokalisierte regionsbasierte Energien

Die lokalisierten regionsbasierten Energien [LT08] stellen eine Methode zur Erweiterung der regionsbasierten Konturenergien dar. Im Grunde kann jede regionsbasierte Energie dafür verwendet werden. Dazu werden für die Berechnung der Kräfte, die auf die Kontur einwirken, nicht mehr global die Regionen des gesamten Bildes, sondern nur noch die lokalen Regionen eines kleinen Bereichs um den jeweiligen Bildpunkt betrachtet. Dies ist in Abbildung 4.6 veranschaulicht.



(a) Lokale innere Region (b) Lokale äußere Region

Abbildung 4.6.: Lokalisierung der Regionen: für jeden Bildpunkt wird ein lokaler Bereich (Kreis) betrachtet. Die Kontur trennt diesen in eine lokale innere und äußere Region [LT08]

Das Energiefunktional der lokalisierten regionsbasierten Energien ist in Gleichung 4.20 definiert. Darin wird für jeden Bildpunkt $\mathbf{x}$ eine lokalisierte Energie berechnet. Dazu werden alle Bildpunkte $\mathbf{y}$, die sich in einem bestimmten Radius r um den Punkt $\mathbf{x}$ befinden, miteinbezogen. Dieser Bereich muss nicht zwangsläufig kreisförmig sein. Ein quadratischer Bereich ist ebenfalls denkbar.

Die für die Bewegungsgleichung der Level-Set-Methode benötigte Kraft F_{∇}^{L} ist in Gleichung 4.21 aufgeführt. Für eine Herleitung sei auf [LT08] verwiesen.

$$\mathcal{B}(\mathbf{x},\mathbf{y}) = \begin{cases} 1, & ||\mathbf{x}-\mathbf{y}|| < r \\ 0, & \text{sonst} \end{cases} \tag{4.19}$$

$$E_{\mathrm{L}} = \sum_{\mathbf{x}}^{I} \sum_{\mathbf{y}}^{I} \mathcal{B}(\mathbf{x},\mathbf{y}) \cdot F(\mathbf{y}) \tag{4.20}$$

$$F_{\nabla}^{L}(\mathbf{x}) = \sum_{\mathbf{y}}^{I} \mathcal{B}(\mathbf{x},\mathbf{y}) \cdot F_{\nabla}(\mathbf{y}) \tag{4.21}$$

In Gleichung 4.19 bis 4.21 ist:	ϕ	die Oberflächenfunktion
	$\mathcal{B}(\mathbf{x},\mathbf{y})$	die Auswahlfunktion für die lokale Regionszugehörigkeit
	$I(\mathbf{x}), I(\mathbf{y})$	der Pixelwert am Bildpunkt $\mathbf{x}$, $\mathbf{y}$
	$\mathbf{x}, \mathbf{y}$	der Bildpunkt mit den Koordinaten (x_1, y_1), (x_2, y_2)
	F	die Energiefunktion der verwendeten regionsbasierten Energie
	F_{∇}	die Kraft der verwendeten regionsbasierten Energie

Eine Lokalisierung der Konturenergien hat den großen Vorteil, dass die Beschaffenheit eines zu verfolgenden Objekts nicht vollkommen homogen sein muss. Allerdings wird deutlich mehr Rechnenaufwand als bei den globalen Energien benötigt. Abbildung 4.7 zeigt ein Beispiel, bei dem die lokalisierte Variante der *Mean Separation Energy* zu einer erfolgreichen Bildsegmentierung führt. Die globale Variante hingegen schlägt fehlt.

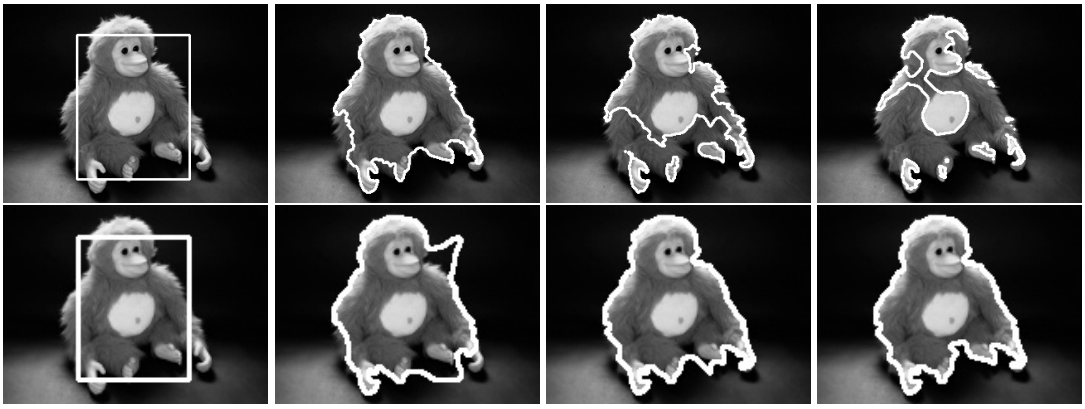


Abbildung 4.7.: Vergleich der globalen (oben) und lokalisierten (unten) *Mean Separation Energy*

5. Partikelfilter

Dieses Kapitel[1] behandelt sogenannte Partikelfilter als Methode zur Lokalisierung und Verfolgung von Objekten in Bildsequenzen. Mit ihrer Hilfe soll Robustheit gegenüber sich schnell bewegenden und partiell verdeckten Objekten erlangt werden.

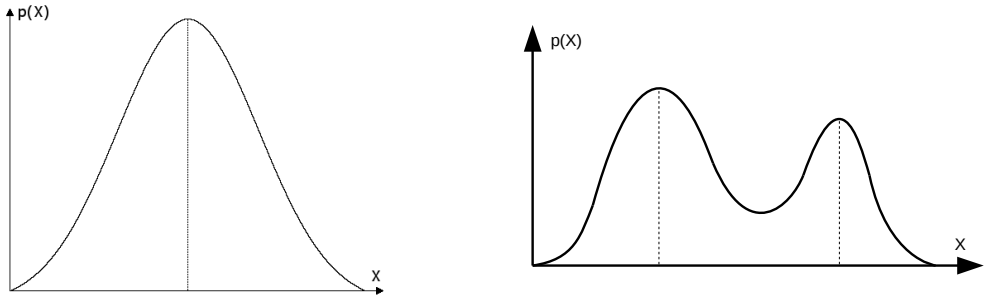
5.1. Grundlegende Partikelfilter

Ein Partikelfilter ist eine probabilistische Methode zur Schätzung eines unbekannten dynamischen Systemzustands. Diese Methoden werden weitestgehend in der Robotik zur Lokalisierung eingesetzt. Ein Kernproblem bei der Lokalisierung ist die damit einhergehende Unsicherheit durch beispielsweise Messungenauigkeiten, Rauschen oder Umgebungsänderungen. Probabilistische Methoden versuchen dies zu bewältigen, indem sie die Unsicherheit explizit mit Hilfe der Wahrscheinlichkeitstheorie modellieren. Dazu repräsentieren sie das System durch eine Wahrscheinlichkeitsverteilung über einen Raum von möglichen Hypothesen.

Eine beliebte Methode zur Modellierung der Unsicherheit ist die Verwendung der Gauß'schen Normalverteilung. Diese sogenannten parametrischen Filter beruhen auf einer festen funktionalen Form der Verteilung und sind unimodal. Partikelfilter hingegen sind nicht-parametrisch und haben somit keine zuvor festgelegte Form der Wahrscheinlichkeitsverteilung. Sie approximieren den Systemzustand durch eine bestimmte Anzahl gewichteter Stichproben (*Samples*) und sind multimodal. Dadurch ist es ihnen möglich, einen viel umfassenderen Bereich an Verteilungen zu repräsentieren. Die Qualität der Approximation hängt dabei von der Anzahl der *Samples* ab. Geht die Anzahl gegen unendlich, konvergieren nicht-parametrische Methoden im Laufe der Zeit zur korrekten Wahrscheinlichkeitsverteilung [TBF05, Kapitel 4].

[1] *Anm.:* In diesem und den folgenden Kapiteln werden bezüglich der Partikelfilter die Begriffe der englischsprachigen Literatur verwendet, wie bspw. *(Re-)Sampling* oder *Prediction*. Zwecks der Grammatik werden die Verben mit „neu auswählen" (*to resample*) und „prognostizieren" (*to predict*) übersetzt.

Abbildung 5.1 zeigt zur Veranschaulichung eine unimodale Gauß'sche Normalverteilung (5.1a) und eine multimodale Verteilung (5.1b). Darin werden die Wahrscheinlichkeiten $p(X)$ verschiedener Zustände X beschrieben. Die multimodale Verteilung hat zwei lokale Wahrscheinlichkeits-Maxima, wohingegen die unimodale Verteilung nur ein Maximum aufweist.



(a) Unimodale Gauß'sche Normalverteilung

(b) Multimodale Wahrscheinlichkeitsverteilung

Abbildung 5.1.: Vergleich unimodaler und multimodaler Wahrscheinlichkeitsverteilungen

Die grundlegende Idee eines Partikelfilters ist es, die Wahrscheinlichkeitsverteilung mittels einer Menge von Zustandshypothesen (*Samples*) zu repräsentieren. Diese Hypothesen werden auch als Partikel bezeichnet. Den Partikeln wird mit Hilfe von Messungen eine Gewichtung zugeordnet, die die Wahrscheinlichkeit widerspiegelt, mit der die jeweilige Zustandshypothese den tatsächlichen Zustand darstellt. Basierend auf dieser Gewichtung werden die Partikel neu ausgewählt (*Resampling*). Dieser Ablauf wird auch als *Importance Sampling* bezeichnet. Er wiederholt sich in jedem Zeitschritt des betrachteten Systems. Dadurch verdichten sich die Partikel immer mehr um einen Bereich hoher Wahrscheinlichkeit.

Ein Iterationsschritt eines Partikelfilters, also ein Zeitschritt im betrachteten System, besteht im Wesentlichen aus zwei Teilschritten: *Prediction* und *Measurement & Update*.

Prediction Die Partikel werden durch ein Bewegungsmodell abgeändert (*Adaption*). Dadurch wird in jedem Iterationsschritt der gesuchte Systemzustand prognostiziert (*to predict*). Zusätzlich wird die *Prediction* durch ein Störungsmodell verzerrt, um die Partikel voneinander unterscheidbar zu machen.

Measurement & Update Jedes Partikel wird durch eine Messung des Systems gewichtet. Aus dieser Messung muss die Korrektheit der Zustandshypothese ableitbar sein. Die Gewichtung beschreibt die Wahrscheinlichkeit, mit der ein Partikel den gesuchten Systemzustand darstellt. Auf Basis dieser Gewichtung werden die Partikel neu ausgewählt

(*to resample*). Das hat zur Folge, dass Partikel mit einer höheren Gewichtung häufiger ausgewählt werden als gering gewichtete Partikel.

Abbildung 5.2 illustriert die Funktionsweise des *Importance Sampling*. Darin beschreibt die Größe der Kreise die Höhe der Gewichtung der Partikel. Auf Basis ihrer Gewichtung werden die Partikel neu ausgewählt und prognostiziert (*hier: drift, diffuse*). Durch eine Messung wird die Gewichtung neu berechnet und die A-posteriori Verteilung erzeugt.

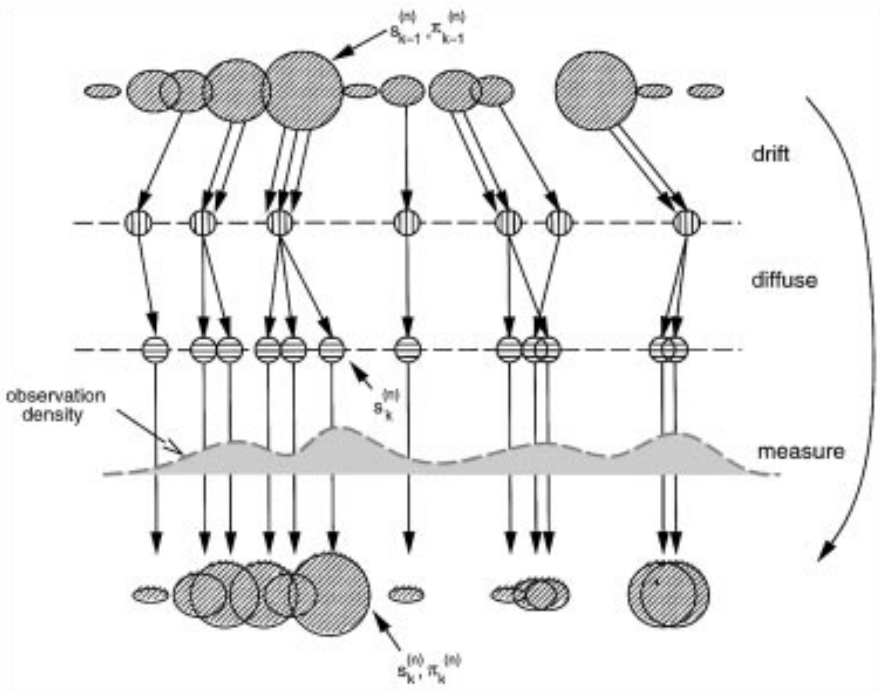
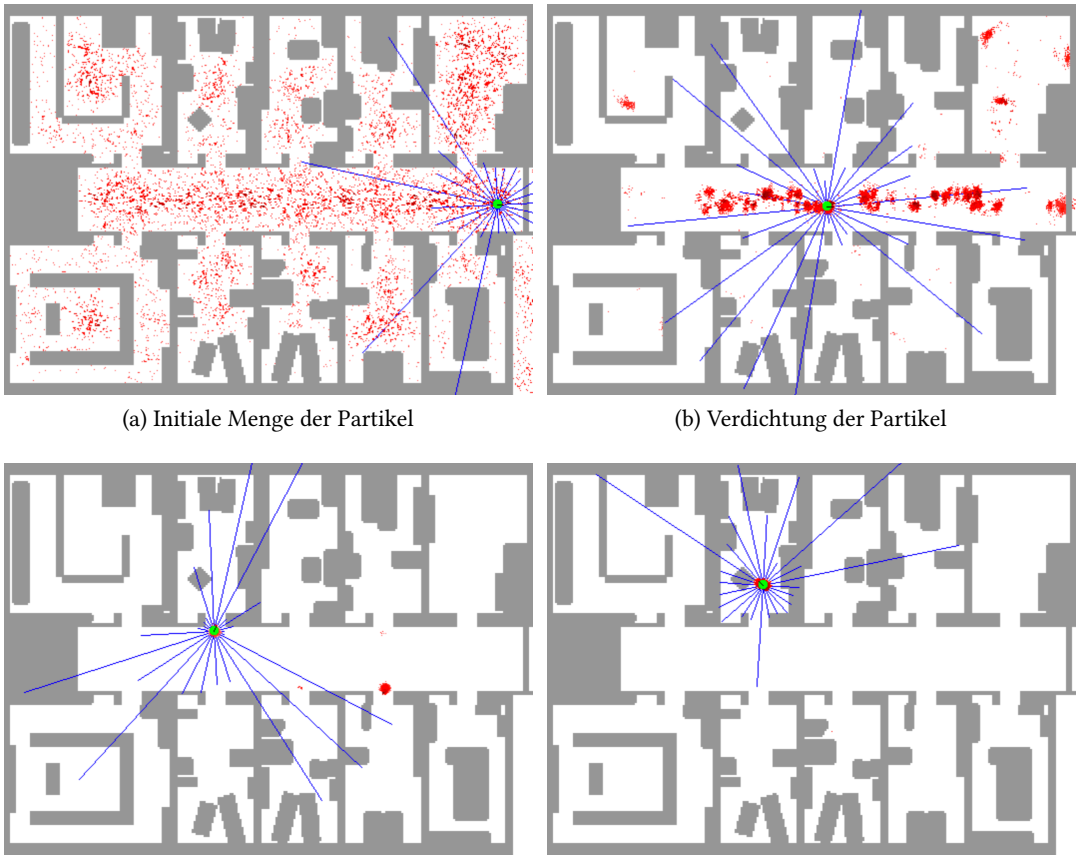


Abbildung 5.2.: Importance Sampling: Erzeugen einer A-posteriori Wahrscheinlichkeitsverteilung [IB98]

Abbildung 5.3 zeigt ein Anwendungsbeispiel für die Lokalisierung eines Roboters auf einer Karte mittels eines Partikelfilters. Darin sind die unterschiedlichen Partikel (rot) in Form ihrer Position dargestellt. Zunächst ist die initiale Menge der Partikel gleichmäßig verteilt (5.3a). Der Roboter, dessen tatsächliche Position (gelb) unbekannt ist, bewertet und gewichtet diese Zustandshypothesen anhand von Abstandsmessungen (blau). Durch die multimodale Wahrscheinlichkeitsverteilung des Partikelfilters sind mehrere Bereiche hoher Wahrscheinlichkeit zu erkennen, die aus der Symmetrie der Karte resultieren (5.3b und 5.3c). Erst wenn der Roboter sich in einen eindeutigen Bereich der Karte bewegt, ist die Zustandsschätzung eindeutig (5.3d). Eine Video-Demonstration dieses Anwendungsbeispiels befindet sich in Anhang B.1 (Zeile 5).



(a) Initiale Menge der Partikel

(b) Verdichtung der Partikel

(c) Zwei Bereiche hoher Wahrscheinlichkeit

(d) Eindeutige Zustandsschätzung

Abbildung 5.3.: Anwendungsbeispiel für die Lokalisierung eines Roboters mittels eines Partikelfilters [TBF05]

5.2. Partikelfilter zur Objektverfolgung

In dieser Arbeit wird ein Partikelfilter für die Verfolgung von Objekten in sequentiellen Aufnahmen eingesetzt. Dazu wird die Position und Größe einer rechteckigen Bildregion verfolgt, in dessen Umfeld sich das gesuchte Objekt befindet. Ausgehend davon kann eine Konturentwicklung (vgl. Kapitel 4) gestartet werden, um die exakten Umrisse des Objekts zu finden.

Dieser Abschnitt beschreibt die Modellierung des in dieser Arbeit entwickelten Partikelfilters. Dazu wird zunächst der Zustandsraum definiert, auf dessen Basis eine Menge von Partikeln erstellt wird. Weiterhin wird das Bewegungsmodell, dass für die *Prediction* der Partikel und damit des gesuchten Systemzustands benötigt wird, erläutert. Außerdem wird die Berechnung der Gewichtung der Partikel zur Erzeugung einer Wahrscheinlichkeitsverteilung sowie die Ermittlung des aktuellen Systemzustands definiert. Zu guter Letzt wird die grundlegende Funktionsweise des *Resampling* beschrieben.

5.2.1. Zustandsraum

Das Partikelfilter hat zum Ziel, einen rechteckigen Bereich zu verfolgen, der die ungefähre Position und Größe des gesuchten Objekts hat. Die initiale Form, also die relative Höhe und Breite des Rechtecks, ist dabei vorab bekannt bzw. parametrierbar. Der Zustandsraum $\mathbf{s}$ des Partikelfilters besteht aus den Positionsparametern x, y sowie einem Skalierungsfaktor s für die Größe des Rechtecks. Weiterhin werden auf Grund des verwendeten Bewegungsmodells (vgl. Abschnitt 5.2.2) die Geschwindigkeiten $\dot{x}, \dot{y}$ bzw. die Positionsänderungen miteinbezogen.

$$\mathbf{s} = \begin{pmatrix} x & y & \dot{x} & \dot{y} & s \end{pmatrix}^T \tag{5.1}$$

Auf Basis dieses Zustandsraums wird eine Menge von N Partikeln $\mathbf{p}_{t=0}^{(i)}$ mit $i = 0, 1, \ldots, N-1$ erstellt.

5.2.2. Bewegungsmodell

Für den *Prediction*-Schritt eines Partikelfilters muss die Änderung des Systemzustands in einem diskreten Zeitschritt näherungsweise bekannt sein. Im Fall des in dieser Arbeit verwendeten Partikelfilters ist dies die Bewegung des zu verfolgenden Objekts. Diese wird auf alle Partikel

angewendet und so der gesuchte Systemzustand prognostiziert. Jedem diskreten Zeitpunkt ist dabei ein Bild der Aufnahmesequenz zugeordnet.

Da die Bewegung allerdings unbekannt ist, wird ein Bewegungsmodell verwendet. In [RVTY07] wird beispielsweise ein *auto-regressives* Modell verwendet, das der Verfolgung von Parametern einer affinen Transformation dient. Ein solches Modell muss allerdings zuvor durch Trainingsdaten erlernt werden. Um unabhängig von Trainingsdaten zu sein, wird in dieser Arbeit ein *einfaches konstantes Bewegungsmodell* ähnlich dem Modell aus [BD11] verwendet.

Gleichung 5.2 definiert das Bewegungsmodell. Eine Zustandsänderung und damit eine *Prediction* des gesuchten Systemzustands erfolgt durch die Multiplikation des Bewegungsmodells mit den aktuellen Partikeln. Die Geschwindigkeiten $\dot{x}, \dot{y}$ werden genutzt, um die Positionsparameter abzuändern. Zusätzlich wird eine zufällige Störung auf die Partikel angewendet. Der Skalierungsfaktor bleibt erhalten und wird nur von der Störung beeinflusst.

Durch das Miteinbeziehen der Änderungen der Positionsparameter, also der Geschwindigkeiten in x- und y-Richtung, in den Zustandsraum werden diese ebenfalls durch das Partikelfilter mitbestimmt und verfolgt. Dadurch wird umgangen, dass die Änderungen der Zustandsparameter näherungsweise bekannt sein müssen.

$$\mathbf{p}_t^{(i)} = \mathbf{T} \cdot \mathbf{p}_{t-1}^{(i)} + \mathbf{u}_t^{(i)} = \begin{pmatrix} 1 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ s \end{pmatrix} + \mathbf{u}_t^{(i)} \tag{5.2}$$

In Gleichung 5.2 ist:

$\mathbf{T}$	die Zustandsübergangsmatrix
$\mathbf{p}_t^{(i)}$	das i-te Partikel zum Zeitpunkt t
$\mathbf{u}_t^{(i)}$	die Störung der Prediction zum Zeitpunkt t
t	der betrachtete Zeitpunkt
Δt	der diskrete Zeitschritt
x, y	die x- und y-Koordinate der Position
$\dot{x}, \dot{y}$	die Geschwindigkeit in x- und y-Richtung
s	der Skalierungsfaktor

5.2.3. Gewichtung

Nach der *Prediction* werden die Zustandshypothesen bewertet, um die Wahrscheinlichkeitsverteilung zu erzeugen. Dazu wird jedem Partikel eine Gewichtung zugeordnet. Die Berechnung der Gewichtung basiert auf dem Histogrammvergleich mittels des Bhattacharyya-Koeffizienten (vgl. Abschnitt 3.1.2). Dabei wird das Histogramm der durch einen Partikel beschriebenen Bildregion mit einem Template-Histogramm verglichen. Dieses Template-Histogramm sollte die optimale Bildregion beschreiben.

Ein Template-Histogramm kann auf unterschiedliche Arten ermittelt werden. Ist die Beschaffenheit der Bildregion, beispielsweise durch ein ähnliches Bild des zu verfolgenden Objekts, gegeben, kann deren Histogramm verwendet werden. In dieser Arbeit wird die ungefähre Startposition des zu verfolgenden Objekts als bekannt angenommen. Damit kann in der ersten Aufnahme der Bildsequenz durch eine Konturentwicklung (vgl. Kapitel 4) die optimale Bildregion gefunden werden.

Gleichung 5.3 definiert die Funktion zur Gewichtungsberechnung der Partikel. Darin wird die Exponentialfunktion verwendet, um aus den Werten des Distanzmaßes des Bhattacharyya-Koeffizienten d_{bc} angemessen verteilte Wahrscheinlichkeitswerte zu erzeugen. Das Distanzmaß beschreibt die Übereinstimmung des Partikel-Histogramms und des Template-Histogramms nach Gleichung 3.4 (S. 7). Durch λ kann die Varianz der Gewichtungsfunktion gesteuert werden.

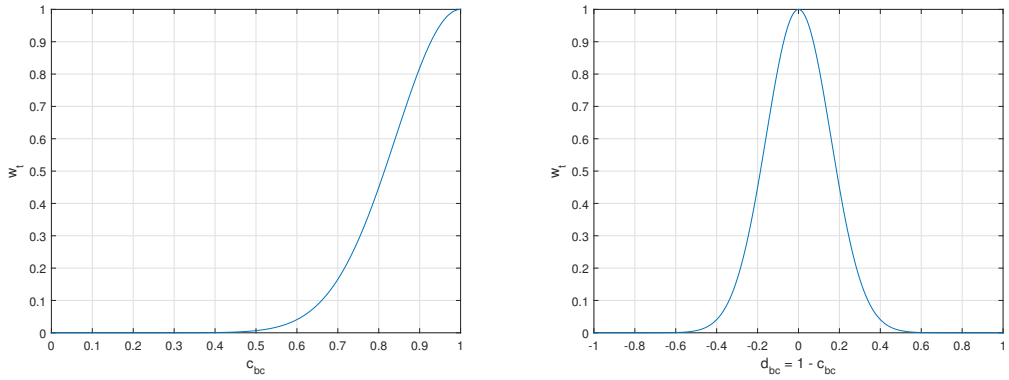
$$w_t^{(i)} = e^{-\lambda d_{\text{bc}}^2} \tag{5.3}$$

In Gleichung 5.3 und Abbildung 5.4 ist:

$w_t^{(i)}$	die Gewichtung des i-ten Partikels zum Zeitpunkt t
c_{bc}	der Bhattacharyya-Koeffizient des Partikel-Histogramms und des Template-Histogramms
d_{bc}	Distanzmaß des Bhattacharyya-Koeffizienten
λ	dient der Steuerung der Varianz

Abbildung 5.4 zeigt die Gewichtungsfunktion in zwei unterschiedlichen Darstellungen: mit c_{bc} als Variable (5.4a) sowie mit $d_{\text{bc}} = (1 - c_{\text{bc}})$ als Variable (5.4b). Je höher der Bhattacharyya-Koeffizient ausfällt, je besser also die Histogramme übereinstimmen, desto höher fällt die Gewichtung des Partikels aus.

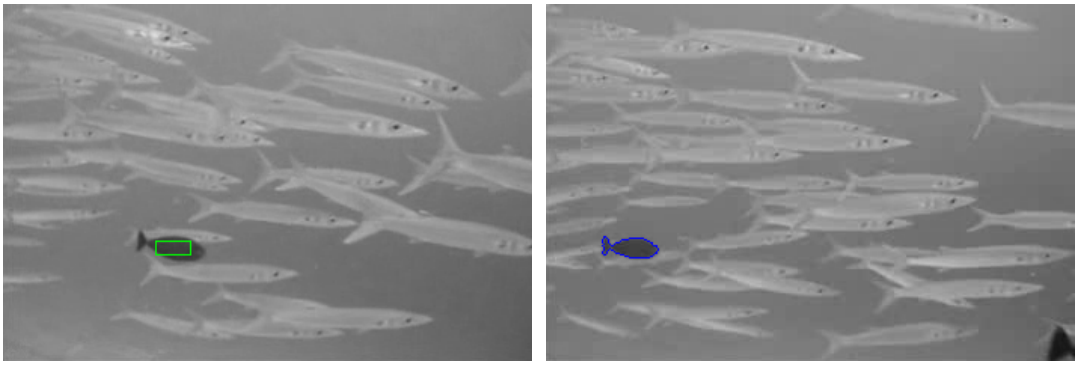
Abbildung 5.5 zeigt ein Beispiel für die Gewichtungsberechnung durch den Vergleich mit dem Histogramm einer Template-Kontur. Die Template-Kontur (5.5b) wurde durch eine Konturentwicklung in der ersten Aufnahme der für die Objektverfolgung verwendeten Bildsequenz



(a) Gewichtungsfunktion mit c_{bc} als Variable (b) Gewichtungsfunktion mit $d_{\mathrm{bc}} = 1 - c_{\mathrm{bc}}$ als Variable

Abbildung 5.4.: Gewichtungsfunktion nach Gleichung 5.3

ermittelt. Abbildung 5.5a zeigt das durch ein Partikel beschriebene Rechteck in Bild 124 der Sequenz. Der Bhattacharyya-Koeffizient wurde aus den Histogrammen dieser beiden Bildregionen (grün bzw. blau umschlossenen) berechnet.



(a) Ein Partikel (b) Template-Kontur

Abbildung 5.5.: Gewichtungsberechnung durch den Histogrammvergleich zweier Bildregionen

5.2.4. Zustandsschätzung

Ist die Gewichtung für alle Partikel berechnet, also die Wahrscheinlichkeitsverteilung erzeugt, kann daraus die Schätzung des Systemzustands zum Zeitpunkt t ermittelt werden. Dazu

wird das gewichtete arithmetische Mittel aller Partikel verwendet. Gleichung 5.4 zeigt die Berechnung des Systemzustands durch den gewichteten Mittelwert.

$$\mathbf{s}_t = \frac{1}{N} \sum_{i=0}^{N-1} w_t^{(i)} \cdot \mathbf{p}_t^{(i)} \tag{5.4}$$

In Gleichung 5.4 ist:

$\mathbf{s}_t$	der ermittelte Systemzustand zum Zeitpunt t
$\mathbf{p}_t^{(i)}$	das i-te Partikel
$w_t^{(i)}$	die Gewichtung des i-ten Partikels
N	die Anzahl der Partikel

5.2.5. Resampling

Im letzten Schritt des Partikelfilters wird eine neue Wahrscheinlichkeitsverteilung erzeugt. Dies geschieht durch sogenanntes *Resampling*. Dazu wird die Menge der Partikel auf Basis ihrer Gewichtung aktualisiert. Aus der bestehenden Menge der N Partikel wird N-mal ein Partikel ausgewählt, sodass eine neue Menge von Partikeln mit derselben Anzahl entsteht. Welches Partikel dabei gewählt wird, hängt von der Höhe seiner Gewichtung ab. Dies führt dazu, dass Partikel mit einer hohen Gewichtung mehrfach ausgewählt werden können und Partikel mit einer niedrigen Gewichtung möglicherweise verworfen werden.

Gleichung 5.5 zeigt die Menge der Partikel $\mathbf{p}_i$ und deren Gewichtungen w_i sowie die Wahrscheinlichkeit α_i, mit der ein bestimmtes Partikel ausgewählt wird (mit $i = 0 \dots N-1$). Die Wahrscheinlichkeit entspricht dabei den normierten Gewichtungen. Abbildung 5.6 zeigt ein *Resampling*-Beispiel. Darin sind in der linken Menge die Partikel, repräsentiert durch ihre Resampling-Wahrscheinlichkeiten α_i, abgebildet. Die Größe der Kreise beschreibt dabei die Höhe der Wahrscheinlichkeit. Die rechte Menge enthält die neu ausgewählten Partikel. Dabei wurde p_2 mit der höchsten Wahrscheinlichkeit dreimal übernommen. p_1 und p_5 mit jeweils einer sehr geringen Wahrscheinlichkeit wurden aussortiert.

$$\begin{array}{ccc} \mathbf{p}_1 & w_1 & \alpha_1 = \frac{w_1}{W} \\ \mathbf{p}_2 & w_2 & \alpha_2 = \frac{w_2}{W} \\ \vdots & \vdots & \vdots \\ \mathbf{p}_{N-1} & w_{N-1} & \alpha_{N-1} = \frac{w_{N-1}}{W} \\ & W = \sum w_i & \sum a_i = 1 \end{array} \tag{5.5}$$

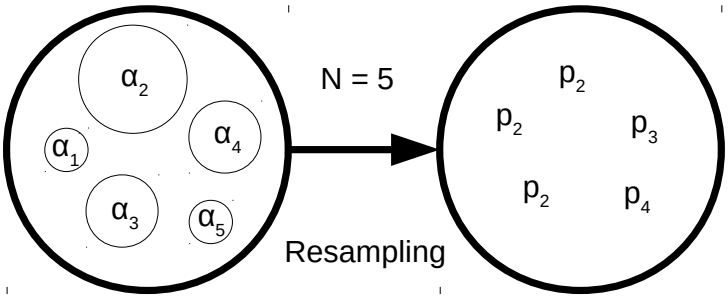


Abbildung 5.6.: Resampling-Beispiel

Es gibt unterschiedliche Methoden zur Umsetzung des *Resampling*. Das in dieser Arbeit verwendete Verfahren wird in Abschnitt 6.3.5 im Zusammenhang mit der Implementierung behandelt.

6. Objektverfolgung

Dieses Kapitel behandelt den in dieser Arbeit entwickelten Algorithmus zur Verfolgung von sich verformenden Objekten in sequentiellen Aufnahmen sowie dessen Implementierung. Dieser hat das Ziel, die Kontur eines Objekts möglichst exakt zu bestimmen und dabei gleichzeitig robust gegenüber partiellen Verdeckungen und schnellen Bewegungen des Objekts zu sein. Der Algorithmus verwendet dazu sowohl Aktive Konturen (Kapitel 4) als auch Partikelfilter (Kapitel 5) sowie die Methoden zur Identifikation von Bildregionen (Kapitel 3). Besondere Beachtung gilt dabei dem Zusammenspiel dieser drei Bereiche.

Für die Konturermittlung wird auf die Aktiven Konturen zurückgegriffen. Für deren Umsetzung wird die sogenannte Sparse-Field-Methode, eine Implementierung der Level-Set-Methode (vgl. Abschnitt 4.2), verwendet, da sie den Echtzeitanforderung der Objektverfolgung am nächsten kommt. Dabei wird die Theorie und Implementierung der Sparse-Field-Methode detailliert erläutert. Die geforderte Robustheit wird durch das Verfolgen der Position und Größe des Objekts erlangt. Dies geschieht mit Hilfe des Partikelfilters aus Abschnitt 5.2. Die Implementierung der einzelnen Schritte des Partikelfilters wird ebenfalls behandelt. Zusätzliche Robustheit gegenüber partiellen Verdeckungen eines Objekts wird durch Fourier-Deskriptoren (vgl. Abschnitt 3.3) bzw. Hu-Momente (vgl. Abschnitt 3.2) erlangt. Dazu wird beschrieben, wie diese für die Unterscheidung zwischen Verformungen und Verdeckungen genutzt werden können.

Um die Details der Implementierung zu verdeutlichen, wird direkter Bezug auf die in dieser Arbeit entwickelte Software genommen. Dies geschieht u.a. durch Source-Code Beispiele. Dazu ist anzumerken, dass die C++ Open-Source Bibliothek OpenCV[1] verwendet wurde, weshalb in den Beispielen oftmals Datenstrukturen und Funktionen daraus zu finden sind. In Anhang A ist der gesamte Source-Code der Software zu finden.

[1] OpenCV: http://opencv.org/

6.1. Algorithmus zur Objektverfolgung

Dieser Abschnitt gibt einen Überblick über den Algorithmus zur Objektverfolgung sowie über die entwickelte Software. Dazu werden zunächst die konkreten Anforderungen an eine erfolgreiche Objektverfolgung definiert. Weiterhin wird der Ablauf des Algorithmus beschrieben und insbesondere das Zusammenspiel der einzelnen Komponenten zur Positionsverfolgung, Konturfindung und dem Umgang mit Verdeckungen erläutert. Die Umsetzung und Implementierung der Teilschritte wird in den Folgeabschnitten behandelt.

6.1.1. Anforderungen und Rahmenbedingungen

Dieser Abschnitt beschreibt die Anforderungen des Algorithmus zur Objektverfolgung. Für eine erfolgreiche Objektverfolgung sind außerdem einige Rahmenbedingungen erforderlich.

Kamera
: Ein zu verfolgendes Objekt wird über eine einfache, nicht-stationäre 2D-Kamera aufgenommen. Die sequentiellen Aufnahmen des Objekts dienen als Grundlage für die Objektverfolgung. Dabei kann es sich sowohl um Farb- als auch um Grauwertaufnahmen handeln, wobei mit Farbaufnahmen eine robustere Verfolgung möglich ist.

Hintergrundänderungen
: Das zu verfolgende Objekt darf sich in einer sich verändernden Umgebung befinden. Dies kann beispielsweise durch Kamerabewegungen entstehen oder durch andere sich im Hintergrund bewegende Objekte.

Verformungen
: Das Objekt darf alle ihm möglichen Formen annehmen, ohne dass eine Verfolgung fehlschlägt. Die exakten Objektumrisse sollen unabhängig von der aktuellen Form des Objekts immer gefunden werden können. Wird beispielsweise die Bewegung und Form einer Hand verfolgt, so kann diese alle unterschiedlichen Formen annehmen, Anzahl an Fingern zeigen o.Ä., ohne dass dies Einfluss auf eine erfolgreiche Konturfindung hat.

Verdeckungen
: Das zu verfolgende Objekt darf während der Objektverfolgung partiellen Verdeckungen ausgesetzt sein. Wird ein Objekt partiell verdeckt, kann zwar die exakte Kontur nicht mehr ermittelt werden, allerdings können sowohl die Form als auch die Position und Größe prognostiziert werden. Um eine Verdeckung von einer Verformung zu

unterscheiden, werden sogenannte *Characteristic Views* verwendet. Dies sind typische Konturformen, repäsentiert durch Fourier-Deskriptoren, die das Objekt annehmen kann. Wahlweise können auch Hu-Momente anstelle der Fourier-Deskriptoren genutzt werden. Behandelt werden *Characteristic Views* in Abschnitt 6.4.3 im Rahmen der Detektion und dem Umgang mit Verdeckungen.

Objektbeschaffenheit

Die Beschaffenheit des Objekts muss sich so deutlich von seiner Umgebung unterscheiden, dass eine Identifikation durch einen Histogrammvergleich vollzogen werden kann. Diese kann je nach Art des Histogramms oder Art der Kameraufnahmen variieren.

Konturenergie

Je nach Beschaffenheit des zu verfolgenden Objekts muss eine entsprechende Konturenergie gewählt oder definiert werden. Nur mit der korrekt gewählten Energie können durch eine Konturentwicklung in den Einzelaufnahmen der Bildsequenz die Objektumrisse gefunden werden.

Startposition

Die ungefähre Startposition des zu verfolgenden Objekts wird als bekannt angenommen. Dies ist notwendig, damit in der ersten Aufnahme der Bildsequenz ein Template für die Gewichtungsberechnung des Partikelfilters (vgl. Abschnitt 5.2.3) und die spätere Konturbewertung ermittelt werden kann.

6.1.2. Algorithmus

Dieser Abschnitt gibt einen Überblick über den Ablauf des Algorithmus zur Objektverfolgung. Auf Basis der eingehenden sequentiellen Aufnahmen eines Objekts wird dessen Verfolgung gestartet. Durch das Partikelfilter aus Abschnitt 5.2 wird die ungefähre Position und Größe des Objekts prognostiziert. Daraufhin werden mit Hilfe der Konturentwicklung (Aktive Konturen, Kapitel 4) die exakten Umrisse des Objekts ermittelt. Abschließend wird die daraus resultierende Kontur bewertet. Dazu wird festgestellt, ob das Ergebnis der Konturentwicklung die tatsächlichen Objektumrisse widerspiegelt. Im Falle partieller Verdeckungen soll durch eine Detektion ebendieser die Prognose der verdeckten Objektkontur ermöglicht werden. Dazu werden sogenannte *Characteristic Views*, vorab bekannte charakteristische Ansichten bzw. Konturformen des Objekts, verwendet.

Zur genauen Erläuterung sind die einzelnen Teilschritte des Algorithmus im Folgenden aufgelistet und beschrieben. In Abbildung 6.1 ist dazu außerdem der grobe Ablauf des Algorithmus als Flussdiagramm veranschaulicht.

Parametrisierung

Die Parametrisierung des Algorithmus ist relativ umfangreich. Die wichtigsten zu spezifizierenden Parameter sind die verwendete Konturenergie, die Anzahl der Schritte der Konturentwicklung, die ungefähre Position und Größe des Objekts im ersten Bild der Sequenz sowie die Anzahl der Partikel. Eine ausführliche Liste und Beschreibung aller Parameter befindet sich in Anhang A.3.

Initialisierung

Die erste Aufnahme der verwendeten Bildsequenz dient der Findung eines Templates. Das Template ist die Grundlage sowohl für die Gewichtungsberechnung des Partikelfilters als auch für die spätere Konturbewertung. Ausgehend von der bekannten ungefähren Startpostion und Größe (rechteckige Bildregion) wird eine Konturentwicklung gestartet, deren Ergebnis als Template-Kontur genutzt wird.

Das Histogramm der von der Kontur umschlossenen Bildregion wird zur Gewichtungsberechnung des Partikelfilters verwendet. Außerdem dient der Fourier-Deskriptor (oder alternativ die Hu-Momente) dieser Kontur als einer der *Characteristic Views* zwecks Detektion und Umgang mit partiellen Verdeckungen.

Aufgrund der initial ermittelten Objektkontur ist es nicht nötig, die initiale Menge der Zustandshypothesen des Partikelfilters über den gesamten Zustandsraum zu verteilen. Stattdessen dient das die Template-Kontur minimal umgebende Rechteck als Basis für die Initialisierung des Partikelfilters. Im näheren Bereich um dieses Rechtecks wird die initiale Menge der Partikel erstellt.

Prediction

Im *Prediction*-Schritt des Partikelfilters werden alle Partikel prognostiziert. Dazu wird das Bewegungsmodell aus Gleichung 5.2, S. 31 angewendet.

Gewichtungsberechnung

Die Gewichtungsberechnung zur Bewertung aller Partikel erfolgt nach Gleichung 5.3, S. 32.

Zustandsschätzung

Nachdem die Gewichtungsberechnung erfolgt ist und damit die Wahrscheinlichkeits-

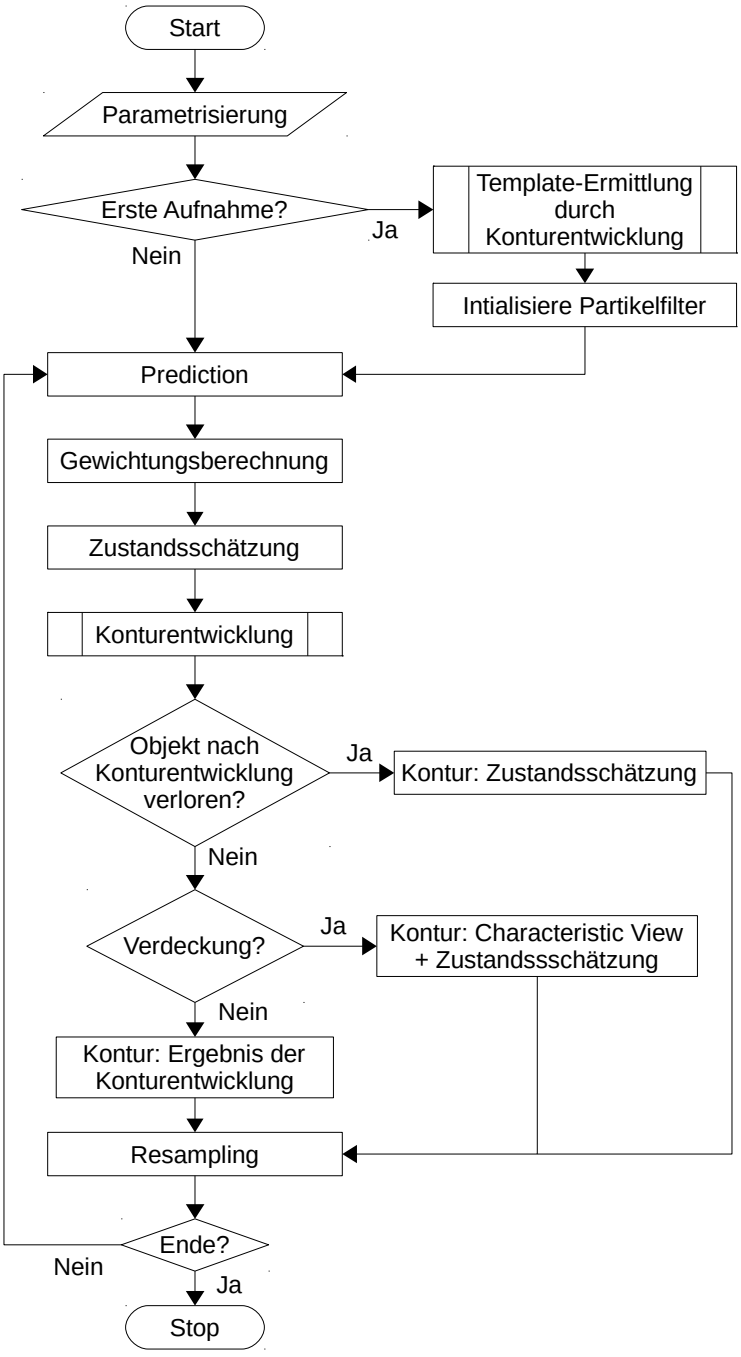


Abbildung 6.1.: Ablauf des Algorithmus zur Objektverfolgung

verteilung erzeugt wurde, lässt sich die Schätzung des Systemzustands des aktuellen Iterationsschritts ermitteln. Dazu wird das gewichtete Mittel aller Partikel gebildet (vgl. Gleichung 5.4, S. 34).

Konturentwicklung

Ausgehend von der rechteckigen Bildregion, die durch die Zustandsschätzung des Partikelfilters beschrieben ist, wird eine Konturentwicklung durchgeführt. So wird aus der robust verfolgten Position und ungefähren Größe die exakte Kontur des Objekts ermittelt. Die entwickelte Kontur dient als Ergebnis des Algorithmus für den aktuellen Iterationsschritt.

Konturbewertung (Objekt verloren)

Nach der Konturentwicklung wird geprüft, ob die Ergebniskontur die tatsächlichen Umrisse des Objekts beschreibt. Dies wird durch einen Vergleich der Histogramme der Ergebniskontur und der Template-Kontur ermittelt. Gegebenenfalls wird das Template-Histogramm adaptiert (vgl. Abschnitt 6.4.1).

Es kann zeitweise vorkommen, dass das Ergebnis der Konturentwicklung nicht zufriedenstellend ist. So kann beispielsweise durch eine kurzzeitige totale Verdeckung des Objekts oder sich stark ändernde Lichtverhältnisse die Konturentwicklung fehlschlagen. In diesem Fall wird die entwickelte Kontur verworfen. Als Ergebniskontur wird stattdessen das durch die Zustandsschätzung des Partikelfilters beschriebene Rechteck verwendet. Für diesen Iterationsschritt, also für dieses Bild der Sequenz, ist die exakte Konturbestimmung fehlgeschlagen, doch wird durch das Partikelfilter weiterhin die ungefähre Position und Größe des Objekts verfolgt.

Partielle Verdeckung

War die Konturentwicklung erfolgreich, sind also die Objektumrisse erfolgreich gefunden, wird untersucht, ob das verfolgte Objekt partiell verdeckt ist. Dies geschieht durch einen Konturvergleich mittels Fourier-Deskriptoren (oder alternativ Hu-Momenten). Dabei wird die Ergebniskontur mit den *Characteristic Views* des Objekts verglichen (vgl. Abschnitt 6.4.3). Anhand dieser bekannten Ansichten bzw. Konturformen kann beurteilt werden, ob es sich um eine partielle Verdeckung oder eine Verformung des Objekts handelt. Zusätzlich zu den *Characteristic Views* wird das Kontur-Template, das in der ersten Aufnahme der Bildsequenz ermittelt wurde, als bekannte Konturform verwendet.

Wurde eine partielle Verdeckung detektiert, so wird eine Ersatz-Kontur erstellt. Dazu wird auf den durch den Konturvergleich zuletzt getroffenen *Characteristic View* zugrück-

gegriffen. Die Kontur des *Characteristic View* wird auf die aktuelle Position des Objekts verschoben und auf die entsprechende Größe skaliert. Dies dient als Ergebniskontur für den aktuellen Iterationsschritt.

Resampling

Die Menge der gewichteten Partikel wird neu ausgewählt und so eine neue Wahrscheinlichkeitsverteilung erzeugt (vgl. Abschnitt 5.2.5). Eine detaillierte Behandlung des in dieser Arbeit angewendeten Resampling-Verfahrens ist in Abschnitt 6.3.5 zu finden.

6.1.3. Komponenten

Dieser Abschnitt gibt einen Überblick über die in dieser Arbeit entwickelte Software. Die folgende Auflistung gibt eine einleitende Beschreibung der fünf Hauptkomponenten. Abbildung 6.2 zeigt das zugehörige Komponentendiagramm. Die *Doxygen*-Dokumentation der Software ist in Anhang A zu finden.

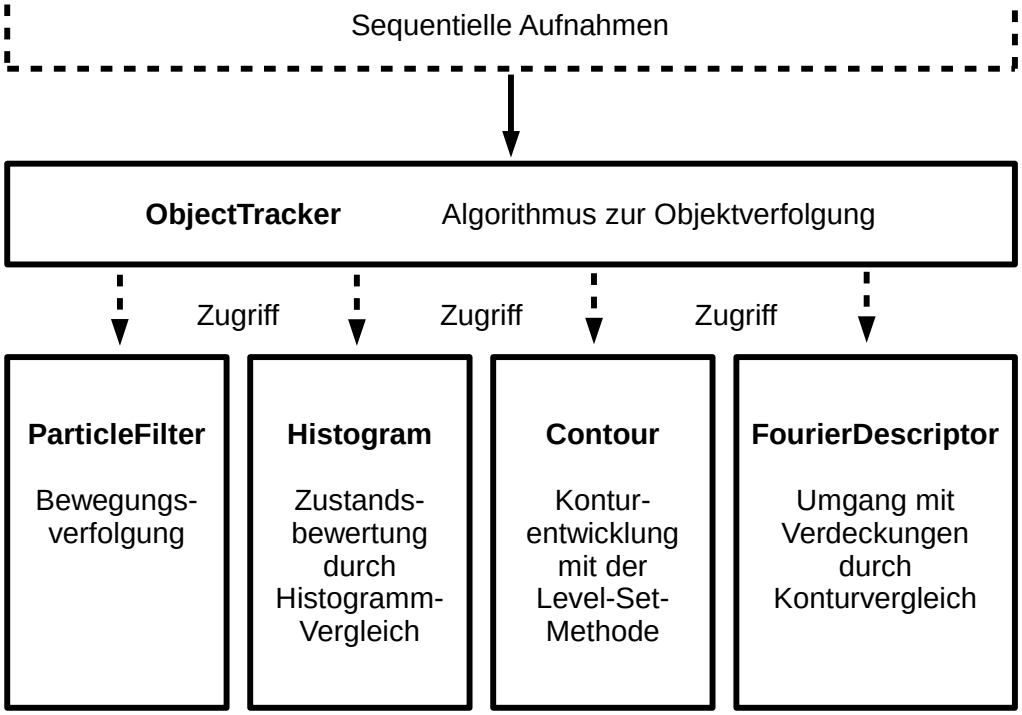


Abbildung 6.2.: Überblick über die Haupt-Komponenten der Software

ObjectTracker
ist die für die Steuerung der Objektverfolgung zuständige Komponente. Sie implementiert den Algorithmus aus Abschnitt 6.1.2. Dazu nimmt sie die sequentiellen Aufnahmen des zu verfolgenden Objekts als Eingabeparameter entgegen und greift auf die benötigten und durch die anderen Komponenten bereitgestellten Teilfunktionen zu.

ParticleFilter
ist die für die Positions- und Größenverfolgung zuständige Komponente. Sie implementiert das Partikelfilter aus Abschnitt 5.2 und stellt dessen Teilfunktionen zur Verfügung.

Contour
ist die für die Konturentwicklung mit der Level-Set-Methode zuständige Komponente. Sie bietet Funktionen zur Ausführung der Konturentwicklung. Neben Datenstrukturen zur Verwaltung von Aktiven Konturen werden alle in Abschnitt 4.3 beschriebenen Konturenergien zur Verfügung gestellt.

Histogram
bietet Datenstrukturen zur Verwaltung und Funktionen zur Berechnung von Histogrammen. Außerdem stellt diese Komponente die Methode zum Histogrammvergleich mit dem Bhattacharyya-Koeffizienten nach Gleichung 3.3, S. 7 zur Verfügung.

FourierDescriptor
bietet Datenstrukturen zur Verwaltung und Funktionen zur Berechnung der Fourier-Deskriptoren aus Abschnitt 3.3.1. Außerdem stellt diese Komponente die Methode zum Konturvergleich nach Abschnitt 3.3.2 zur Verfügung.

6.2. Sparse-Field-Methode

Dieser Abschnitt behandelt die Implementierung Aktiver Konturen mit der Level Set-Methode (vgl. Abschnitt 4.2) durch die sogenannte *Sparse-Field-Methode*.

Es gibt unterschiedliche Ansätze zur konkreten Umsetzung der Konturentwicklung. Dies ist beispielsweise die sogenannte *Full-Matrix-Methode*, bei der die Bewegungsgleichung der Konturentwicklung für jeden einzelnen Bildpunkt iterativ gelöst wird. Einen anderen Weg wählt die *Narrow-Band Methode*, bei der nicht mehr jeder einzelne Bildpunkt, sondern nur einer kleiner Bereich um die Kontur, das sogenannte *Narrow-Band*, entwickelt wird [Set99]. Eine weitere Methode ist die in diesem Abschnitt behandelte Sparse-Field-Methode. Sie wurde

ursprünglich in [Whi98] vorgestellt, doch dort nur sehr knapp und theoretisch beschrieben. [Sha09] hingegen beschreibt die Sparse-Field-Methode und die praktische Umsetzung sehr detailliert. Diese drei Methoden wurden ausführlich in [Olt14a] bezüglich ihrer praktischen Möglichkeiten als auch ihres Rechenaufwands untersucht. Dabei hat sich die Sparse-Field-Methode als die effizienteste Methode herausgestellt.

Die Sparse-Field-Methode beruht auf der Idee, die Bewegungsgleichung für die Konturentwicklung nur für die Bildpunkte zu lösen, die sich auf der Kontur befinden. Das bedeutet, die Oberflächenfunktion der Level Set-Methode wird nur für das *Zero Level Set* entwickelt. Die Umgebung der Kontur wird entsprechend angepasst, sodass die SDF-Bedingung (vgl. Abschnitt 4.2.2) eingehalten wird. Dazu wird das *Zero Level Set* (die Kontur) sowie die angrenzenden *Level Sets* (die Umgebung) durch Listen von Bildpunkten repräsentiert. Je nach Änderung der Kontur durch die Bewegungsgleichung werden die Bildpunkte in die entsprechenden Listen verschoben. Insgesamt werden fünf Listen genutzt, um das *Zero Level Set* sowie die zwei ins Konturinnere und die zwei ins Konturäußere angrenzenden *Level Sets* zu repräsentieren und zu verwalten. Die Gleichungen 6.2 bis 6.6 definieren die verwendeten Listen. Die Wertebereiche für die Listenzugehörigkeit sind so gewählt, dass der Abstand der Gitterzellen bzw. ein diskreter Ortsschritt im dynamischen System

$$\Delta x = 0.5 \tag{6.1}$$

entspricht (vgl. Abschnitt 4.2.3).

$$L_0 = \{\mathbf{x} \mid \phi(\mathbf{x}, t) \in [-0.5, 0.5]\} \tag{6.2}$$
$$L_{-1} = \{\mathbf{x} \mid \phi(\mathbf{x}, t) \in [-1.5, -0.5)\} \tag{6.3}$$
$$L_1 = \{\mathbf{x} \mid \phi(\mathbf{x}, t) \in (0.5, 1.5]\} \tag{6.4}$$
$$L_{-2} = \{\mathbf{x} \mid \phi(\mathbf{x}, t) \in [-2.5, -1.5)\} \tag{6.5}$$
$$L_2 = \{\mathbf{x} \mid \phi(\mathbf{x}, t) \in (1.5, 2.5]\} \tag{6.6}$$

In Gleichung 6.2 bis 6.7 ist:		
	ϕ	die Oberflächenfunktion
	$\mathbf{x}$	der Bildpunkt mit den Koordinaten (x, y)
	t	der betrachtete Zeitpunkt
	L_{-2} bis L_2	die Listen für die jeweiligen *Level Sets*

Neben diesen Listen wird eine *Label-Map* in derselben Dimension und Größe wie die Oberflächenfunktion ϕ benötigt. Darin wird die Listenzugehörigkeit der Bildpunkte mit den ganzzahligen Werten $-3, -2, -1, 0, 1, 2, 3$ markiert. Dabei markieren -3 und 3 alle Bildpunkte

$$\{\mathbf{x} \mid \phi(\mathbf{x}, t) \notin [-2.5, 2.5]\} \tag{6.7}$$

d.h. alle Bildpunkte, die soweit von der Kontur entfernt liegen, dass sie keiner der Listen zugeordnet werden. Abbildung 6.3 veranschaulicht das *Sparse-Field*, also die Listen der unterschiedlichen *Level Sets*, die für die Repräsentation der Kontur und die Durchführung der Konturentwicklung verwendet werden, durch ein Beispiel.

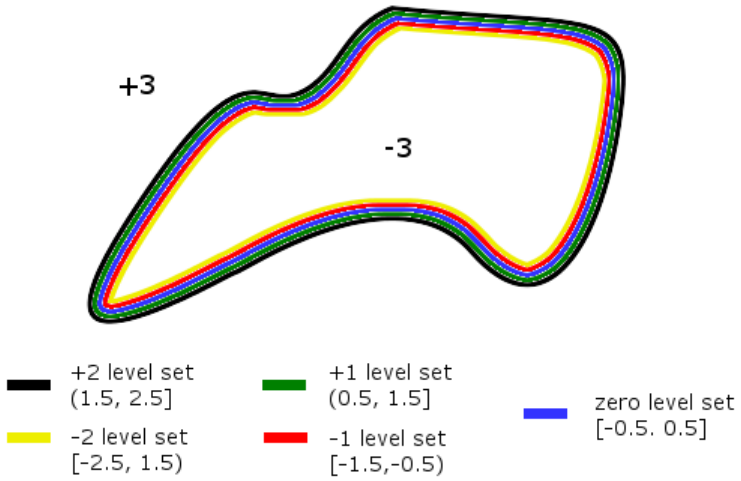


Abbildung 6.3.: Sparse-Field: die Listen der unterschiedlichen *Level Sets*

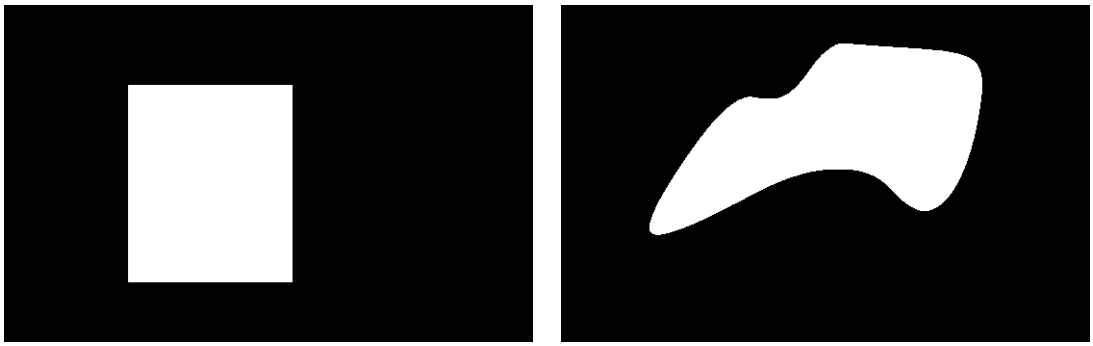
Im weiteren Verlauf dieses Abschnitts wird immer wieder Pseudo-Code zur Verdeutlichung des Ablaufs und der Umsetzung der Sparse-Field-Methode verwendet. Im Folgenden sind zur Übersicht einige Variablen aufgelistet, die darin genutzt werden.

- `init_mask`: die Initialisierungsmaske (Binärbild)
- `phi`: die Oberflächenfunktion
- `F`: die auf die Kontur wirkende Kraft
- `label`: die *Label-Map* in derselben Dimension und Größe wie `phi`
- `lz`, `ln1`, `lp1`, `ln2`, `lp2`: die *Level Set* Listen L_0, L_{-1}, L_1, L_{-2}, L_2.

- `x, y`: ein Bildpunkt mit den Koordinaten (x, y)
- `N(x, y)`: jeder direkte Nachbar eines Bildpunktes (x, y) – oben $(x, y-1)$, rechts $(x+1, y)$, unten $(x, y+1)$, links $(x-1, y)$

6.2.1. Initialisierung

Zur Initialisierung der Sparse-Field-Methode wird ein Binärbild verwendet, das den Startzustands der Aktiven Kontur beschreibt. Dieses Binärbild dient als Initialisierungsmaske. Darin haben alle Bildpunkte außerhalb der Kontur den Wert 0 und alle Punkte innerhalb der Kontur einen Wert $\neq 0$. Auf Basis der Initialisierungsmaske werden die Listen der *Level Sets* erstellt und die *Label-Map* sowie die Oberflächenfunktion initialisiert. Abbildung 6.4 zeigt exemplarisch zwei Initialisierungsmasken. Dabei handelt es sich zum einen um den typischen, rechteckigen Startzustand (6.4a) und zum anderen um die Initialisierungsmaske für die Kontur aus Abbildung 6.3 (6.4b).



(a) (b)

Abbildung 6.4.: Beispiel zweier Initialisierungsmasken als Binärbilder

Listing 6.1 zeigt den Pseudo-Code für den Initialisierungsvorgang. Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 1 Initialisierung auf Basis eines Binärbildes. Sowohl der Oberflächenfunktion als auch der *Label-Map* wird innerhalb der Kontur der Wert -3 und außerhalb der Kontur der Wert 3 zugewiesen.

Zeile 10 Suche nach dem *Zero Level Set*. Dies ist der äußere Rand des Konturinneren, also des Bereichs der Initialisierungsmaske mit einem Wert $\neq 0$. Sowohl der Oberflächenfunktion als auch der *Label-Map* wird an diesen Bildpunkten der Wert 0 zugewiesen.

Zeile 17 Suche nach dem -1 und $+1$ *Level Set*. Dabei werden alle Nachbarn des zuvor gefundenen *Zero Level Set* betrachtet. Sowohl der Oberflächenfunktion als auch der *Label-Map* wird der Wert -1 bzw. 1 zugewiesen.

Zeile 29 und 37 Suche nach dem -2 und $+2$ *Level Set*. Dies geschieht analog zur Suche nach dem -1 und $+1$ *Level Set*.

```
1  // initialize label map and phi
2  for each image coordinate x, y:
3      if (init_mask(x, y) == 0):
4          label(x, y) = 3
5          phi(x, y) = 3.0
6      else:
7          label(x, y) = -3
8          phi(x, y) = -3.0
9
10 // find zero level set
11 for each point x, y in image:
12     if (init_mask(x, y) != 0 && at least one N(x, y) == 0):
13         lz.push_back(Point(x, y))
14         label(x, y) = 0
15         phi(x, y) = 0
16
17 // find the -1 and +1 level set
18 for each point x, y in lz:
19     for each neighbor xn, yn in N(x, y):
20         if (label(xn, yn) == -3):
21             ln1.push_back(Point(xn, yn))
22             label(xn, yn) = -1
23             phi(xn, yn) = -1
24         else if (label(xn, yn) == 3):
25             lp1.push_back(Point(xn, yn))
26             label(xn, yn) = 1
27             phi(xn, yn) = 1
28
29 // find the -2 level set
30 for each point x, y in ln1:
31     for each neighbor xn, yn in N(x, y):
32         if (label(xn, yn) == -3):
```

```
33              ln2.push_back(xn, yn)
34              label(x, y) = -2
35              phi(x, y) = -2
36
37  // find the +2 level set
38  for each point x, y in lp1:
39      for each neighbor xn, yn in N(x, y):
40          if (label(xn, yn) == 3):
41              lp2.push_back(xn, yn)
42              label(x, y) = -2
43              phi(x, y) = -2
```

Listing 6.1: Initialisierung der Sparse-Field-Methode

6.2.2. Konturentwicklung

Dieser Abschnitt behandelt die Implementierung der Konturentwicklung mit der Sparse-Field-Methode. Dazu werden zunächst die Feinheiten der Berechnung der Kraft für die Bewegungsgleichung erläutert. Weiterhin wird im Detail beschrieben, wie die Oberflächenfunktion aktualisiert und somit die Kontur entwickelt wird.

Sind alle Datenstrukturen initialisiert, kann die Kontur entwickelt werden. Dies geschieht durch das iterative Lösen der Bewegungsgleichung, einer partiellen Differentialgleichung (vgl. Gleichung 4.4, S. 18). Dazu muss zunächst die auf die Kontur wirkende Kraft für alle Konturpunkte berechnet werden, sodass die Oberflächenfunktion aktualisiert werden kann. Alle angrenzenden *Level Sets* werden entsprechend angepasst.

Für die Konturentwicklung werden zusätzlich fünf temporäre Listen benötigt. Sie dienen der Zwischenspeicherung der Bildpunkte, die den Wertebereich ihres *Level Set* verlassen haben und deshalb in eine neue Liste verschoben werden müssen. In den Gleichungen 6.8 bis 6.12 sind die verwendeten temporären Listen definiert.

$$S_0 = \text{Punkte, die sich nach } L_0 \text{ bewegen} \tag{6.8}$$
$$S_{-1} = \text{Punkte, die sich nach } L_{-1} \text{ bewegen} \tag{6.9}$$
$$S_1 = \text{Punkte, die sich nach } L_1 \text{ bewegen} \tag{6.10}$$
$$S_{-2} = \text{Punkte, die sich nach } L_{-2} \text{ bewegen} \tag{6.11}$$
$$S_2 = \text{Punkte, die sich nach } L_2 \text{ bewegen} \tag{6.12}$$

Listing 6.2 zeigt den Pseudo-Code zur Berechnung der Kraft für die Konturentwicklung, die zur Aktualisierung der Oberflächenfunktion benötigt wird. Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 4 Berechnung der gemittelten Pixelwerte innerhalb (`meanInt` $= \mu_{\text{in}}$) und außerhalb (`meanExt` $= \mu_{\text{out}}$) der Kontur, die für die Berechnung der Kräfte der regionsbasierten Energien benötigt werden.

Zeile 11 Berechnung der Kraft $-F_{\nabla}$, die für die Konturentwicklung benötigt wird. An dieser Stelle sind die Kräfte der *Uniform Modeling Energy* (Gleichung 4.16, S. 22) sowie der *Mean Separation Energy* (Gleichung 4.18, S. 23) zu erkennen. Die lokalisierten Varianten sind zur Vereinfachung nicht aufgeführt, jedoch ebenfalls in der entwickelten Software implementiert.

Zeile 22 Berechnung des Betragsmaximums aller Kräfte zwecks Normalisierung. Dies ist notwendig, um die Oberflächenfunktion als *Signed Distance Function (SDF)* im euklidischen Raum zu erhalten (vgl. Abschnitt 4.2.2).

Zeile 27 Normalisierung der Kräfte und Berechnung der Konturkrümmung. Letztere ist an dieser Stelle zugunsten der Übersichtlichkeit nicht aufgeführt. Die Berechnung erfolgt jedoch exakt nach Gleichung 4.9, S. 20.

```
1 // I = current image
2 // phi (surface function) and F (force) in same size and type
3
4 meanInt = 0.0, meanExt = 0.0, sumInt = 0.0, sumExt = 0.0
5 // ... calculate mean
6
7 // calculate force
```

```
8  maxF = 0.0
9  for each point x, y in lz:
10
11     // unified modeling energy
12     if (method == CHAN_VESE):
13         diffInt = I(x, y) - meanInt
14         diffExt = I(x, y) - meanExt
15         F(x, y) = diffInt*diffInt - diffExt*diffExt
16
17     // mean separation energy
18     else if (method == YEZZI):
19         F(x, y) = -((meanInt - meanExt) * ((I(x, y) - meanInt) / sumInt
20                                          + (I(x, y) - meanExt) / sumExt))
21
22     // get maxF for normalization
23     absFi = fabs(F(x, y))
24     if (absFi > maxF):
25         maxF = absFi
26
27 // normalize force and calculate curvature
28 for each point x, y in lz:
29     // calculate curvature
30     curvature = /* ... */
31     F(x, y) = F(x, y)/maxF + alpha*curvature
```

Listing 6.2: Konturentwicklung mit der Sparse-Field-Methode: Berechnung der auf die Kontur wirkenden Kraft

Listing 6.3 zeigt die Aktualisierung der Oberflächenfunktion für das *Zero Level Set* bzw. die Anpassung der Oberflächenfunktion für die anderen *Level Sets*. Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 6 Berechnung des Zeitschrittfaktors Δt der Bewegungsgleichung nach Gleichung 4.6, S. 19 mit $c = 0.9$ und $\Delta x = 0.5$. Dies dient der Einhaltung der CFL-Bedingung (vgl. Abschnitt 4.2.3).

Zeile 9 Aktualisierung der Oberflächenfunktion für das *Zero Level Set*, also für jeden Bildpunkt in der Liste L_0. Dazu wird die zuvor berechnete Kraft $-F_\nabla$ zu den bestehenden Werten der Oberflächenfunktion ϕ addiert. Außerdem werden alle sich aus dem Wertebereich

von L_0 heraus bewegende Bildpunkte in die entsprechende temporäre Liste S_{-1}/S_1 verschoben.

Anm.: An dieser Stelle ist es sinnvoll, sich die Bildpunkte zu merken, die sich vom Konturinneren (`phi(x, y) < 0`) ins Konturäußere (`phi(x, y) > 0`) bewegt haben. Dadurch können die inneren und äußeren Mittelwerte, die den regionsbasierten Konturenergien zugrunde liegen, effizienter aktualisiert werden.

Zeile 20 und 36 Aktualisierung der Oberflächenfunktion für das -1 und $+1$ *Level Set*, also für jeden Bildpunkt in der Liste L_{-1} bzw. L_1. Dabei wird die Oberflächenfunktion so angepasst, dass die Werte an den betrachteten Bildpunkten genau um $|1|$ von ihrem nächsten Nachbarn in L_0 entfernt sind. Daraufhin werden die sich dadurch aus dem Wertebereich von L_{-1} bzw. L_1 bewegenden Bildpunkte in die entsprechenden temporären Listen verschoben. Hat der Bildpunkt keine Nachbarn mehr in L_0, wird er ins nächste *Level Set* verschoben, also in die temporäre Liste S_{-2} bzw. S_2.

Zeile 52 und 70 Aktualisierung der Oberflächenfunktion für das -2 und $+2$ *Level Set*, also für jeden Bildpunkt in der Liste L_{-2} bzw. L_2. Dabei wird die Oberflächenfunktion so angepasst, dass die Werte an den betrachteten Bildpunkten genau um $|1|$ von ihrem nächsten Nachbarn in L_{-1} bzw. L_1 entfernt sind. Daraufhin werden die sich dadurch aus dem Wertebereich von L_{-2} bzw. L_2 bewegenden Bildpunkte in die entsprechenden temporären Listen verschoben. Hat der Bildpunkt keine Nachbarn mehr in L_{-1} bzw. L_1, wird er aus seiner Liste entfernt. Zusätzlich werden die *Label-Map* und die Oberflächenfunktion auf -3 gesetzt und so der Bildpunkt aus dem Bereich der Listen entfernt.

```
1  maxF = 0.0
2  for each point x, y in lz:
3      if (F(x, y) > maxF):
4          maxF = F(x, y)
5
6  // maintain the CFL condition with c = 0.9 and delta x = 0.5
7  dt = .45f / maxF // 0.9*0.5 = 0.45
8
9  // update zero level set and phi
10 for each point x, y in lz:
11     phi(x, y) += dt * F(x, y)
```

```
12
13      if (phi(x, y) > 0.5):
14          sp1.push_back(Point(x, y))
15          lz.remove(Point(x, y))
16      else if (phi(x, y) < -.5f):
17          sn1.push_back(Point(x, y))
18          lz.remove(Point(x, y))
19
20  // update -1 level set
21  for each point x, y in ln1:
22      if (each label(N(x, y)) != 0):
23          sn2.push_back(Point(x, y))
24          ln1.remove(Point(x, y))
25      else:
26          m = max(phi(N(x, y) with label >= 0))
27          phi(x, y) = m - 1;
28
29          if (phi(x, y) >= -0.5):
30              sz.push_back(Point(x, y))
31              ln1.remove(Point(x, y))
32          else if (phi(x, y) < -1.5):
33              sn2.push_back(Point(x, y))
34              ln1.remove(Point(x, y))
35
36  // update +1 level set
37  for each point x, y in lp1:
38      if (each label(N(x, y)) != 0):
39          sp2.push_back(Point(x, y))
40          lp1.remove(Point(x, y))
41      else:
42          m = min(phi(N(x, y) with label <= 0))
43          phi(x, y) = m + 1;
44
45          if (phi(x, y) <= 0.5)
46              sz.push_back(Point(x, y))
47              lp1.remove(Point(x, y))
48          else if (phi(x, y) > 1.5)
49              sp2.push_back(Point(x, y))
50              lp1.remove(Point(x, y))
51
```

```
52 // update -2 level set
53 for each point x, y in ln2:
54     if (each label(N(x, y)) != -1):
55         ln2.remove(Point(x, y))
56         label(x, y) = -3
57         phi(x, y) = -3.0
58     else:
59         m = max(phi(N(x, y) with label >= -1))
60         phi(x, y) = m - 1
61 
62         if (phi(x, y) >= -1.5)
63             sn1.push_back(Point(x, y))
64             ln2.remove(Point(x, y))
65         else if (phi(x, y) < -2.5)
66             ln2.remove(Point(x, y))
67             label.at<int>(x, y) = -3
68             phi.at<float>(x, y) = -3.0
69 
70 // update +2 level set
71 for each point x, y in lp2:
72     if (each label(N(x, y)) != 1):
73         lp2.remove(Point(x, y))
74         label(x, y) = 3
75         phi(x, y) = 3.0
76     else:
77         m = min(phi(N(x, y) with label <= 1))
78         phi(x, y) = m + 1
79 
80         if (phi(x, y) <= 1.5)
81             sp1.push_back(Point(x, y))
82             lp2.remove(Point(x, y))
83         else if (phi(x, y) > 2.5)
84             lp2.remove(Point(x, y))
85             label.(x, y) = 3
86             phi(x, y) = 3.0
```

Listing 6.3: Konturentwicklung mit der Sparse-Field Methode: Aktualisierung bzw. Anpassung der Oberflächenfunktion

Listing 6.4 zeigt die Handhabung der Bildpunkte, die das *Level Set* wechseln. Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 1 Verschieben aller Bildpunkte aus der temporären Liste S_0 in die Liste L_0 sowie Aktualisierung der *Label-Map*.

Zeile 6 und 14 Verschieben aller Bildpunkte aus der temporären Liste S_{-1} bzw. S_1 in die Liste L_{-1} bzw. L_1 sowie Aktualisierung der *Label-Map*. Zusätzlich werden alle Nachbarn, deren Werte außerhalb des Listenbereichs (< -2.5 und > 2.5) liegen, aktualisiert, sodass ihre Werte genau um $|1|$ vom aktuell betrachteten Bildpunkt entfernt sind. Diese werden dann in die entsprechende temporäre Liste S_{-2} bzw. S_2 verschoben. Dadurch wird sichergestellt, dass die benachbarten Bildpunkte auch im benachbarten *Level Set* liegen.

Zeile 22 und 27 Verschieben aller Bildpunkte aus der temporären Liste S_{-2} bzw. S_2 in die Liste L_{-2} bzw. L_2 sowie Aktualisierung der *Label-Map*.

```
1  // move points into zero level set
2  for each point x, y in sz:
3      lz.push_back(Point(x, y));
4      label(x, y) = 0;
5
6  // move points into -1 level set and ensure -2 neighbours
7  for each point x, y in sn1:
8      ln1.push_back(Point(x, y));
9      label(x, y) = -1;
10     for each point xn, yn in N(x, y):
11         if (phi(xn, yn) < -2.5):
12             phi(xn, yn) = phi(x, y) - 1
13
14 // move points into +1 level set and ensure +2 neighbours
15 for each point x, y in sp1:
16     lp1.push_back(Point(x, y));
17     label(x, y) = 1;
18     for each point xn, yn in N(x, y):
19         if (phi(xn, yn) > 2.5):
20             phi(xn, yn) = phi(x, y) + 1
21
22 // move points into -2 level set
23 for each point x, y in sn2:
```

```
24      ln2.push_back(Point(x, y));
25      label(x, y) = -2;
26
27  // move points into -2 level set
28  for each point x, y in sp2:
29      lp2.push_back(Point(x, y));
30      label(x, y) = 2;
```

Listing 6.4: Konturentwicklung mit der Sparse-Field Methode: Handhabung der Bildpunkte, die das *Level Set* wechseln

6.3. Implementierung des Partikelfilters

Dieser Abschnitt behandelt dies Implementierung des Partikelfilters zur Objektverfolgung aus Abschnitt 5.2. Es wird die konkrete Umsetzung der einzelnen Teilschritte des Partikelfilters beschrieben und erläutert. Neben der *Prediction*, der Gewichtungsberechnung und der Zustandsschätzung gilt dabei besondere Aufmerksamkeit der Implementierung des *Resampling*.

In den Source-Code-Beispielen dieses Abschnitts sind einige wiederkehrende Variablen zu finden:

- `frame`: die aktuell betrachtete Aufnahme des zu verfolgenden Objekts in der Bildsequenz
- `state`: die aktuell ermittelte Zustandsschätzung (1 x 5 Matrix)
- `p[i]`: das i-te Partikel (1 x 5 Matrix)

6.3.1. Initialisierung

Damit eine erfolgreiche Objektverfolgung durchgeführt werden kann, muss das Partikelfilter initialisiert werden. Dies geschieht mittels der Template-Kontur, die in der ersten Aufnahme des zu verfolgenden Objekts durch eine Konturentwicklung ermittelt wurde (vgl. Abschnitt 6.1).

Das die Template-Kontur umschließende Rechteck dient als Grundlage für die Erstellung der initialen Menge von Partikeln. Dabei ist das Zentrum des Rechtecks die nun bekannte Startposition. Die initiale Geschwindigkeit wird $\dot{x} = \dot{y} = 0$ angenommen und der Skalierungsfaktor $s = 1$ gesetzt. Als Störungsmodell wird ein zufälliges Gauß'sches Rauschen verwendet. Es dient dazu, die Partikel zu verzerren, um sie voneinander unterscheidbar zu machen und damit

die Unsicherheit des Systemzustands zu modellieren. Gleichung 6.13 zeigt die Berechnung der initialen Menge von $i = 0, 1, \ldots, N-1$ Partikeln.

$$\mathbf{p}_{t=0}^{i} = \begin{pmatrix} x = x_c \\ y = y_c \\ \dot{x} = 0 \\ \dot{y} = 0 \\ s = 1 \end{pmatrix} + \text{random}(\,\mathcal{N}(\mu = 0, \sigma)) \tag{6.13}$$

In Gleichung 6.13 ist:

$\mathbf{p}_{t=0}^{(i)}$	das i-te Partikel zum Zeitpunkt $t = 0$
x_c, y_c	das Zentrum des die Template-Kontur umschließenden Rechtecks
x, y	die x- und y-Koordinate der Position
$\dot{x}, \dot{y}$	die Geschwindigkeit in x- und y-Richtung
s	der Skalierungsfaktor
random()	die Zufallsfunktion
$\mathcal{N}(\mu = 0, \sigma)$	die Gauß'sche Normalverteilung mit einem Mittelwert von $\mu = 0$
σ	die Standardabweichung der Gauß'schen Normalverteilung

Listing 6.5 zeigt die Initialisierung des Partikelfilters. Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 3 Definition des initialen Zustands auf Basis des zuvor ermittelten Kontur-Templates. Das Zentrum des umschließenden Recktecks (`templ_rect`) dient als Startposition.

Zeile 6 Definition der Standardabweichungen (`sigma`) für das Gauß'sche Rauschen.

Zeile 8 Initialisierung der Partikel `p[i]`. Jedes einzelne Partikel wird durch ein zufälliges Gauß'sches Rauschen verzerrt.

```
1 void ParticleFilter::init(const cv::Rect templ_rect)
2 {
3     // initial state: [x = x_c, y = y_c, x_vel = 0, y_vel = 0, scale = 1]
4     state = {templ_rect.x + templ_rect.width/2.0,
5              templ_rect.y + templ_rect.height/2.0, 0.0, 0.0, 1.0};
6     sigma = {2.0, 2.0, 0.5, 0.5, 0.1}; // distortion deviation
7
```

```
8     // init particles
9     for (int i = 0; i < N; i++) {
10        for (int j = 0; j < NUM_PARAMS; j++) {
11            float noise = random.gaussian(sigma[j]);
12            p[i](j) = state[j] + noise;
13        }
14    }
15 }
```

Listing 6.5: Initialisierung des Partikelfilters

6.3.2. Prediction

Im *Prediction*-Schritt des Partikelfilters wird der gesuchte Systemzustand prognostiziert. Dazu wird das konstante Bewegungsmodell (vgl. Gleichung 5.2, S. 31) auf alle Partikel angewendet.

Listing 6.6 zeigt die *Prediction* der Partikel. Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 1 Definition des Bewegungsmodells.

Zeile 10 *Prediction* der Partikel durch Anwendung des Bewegungsmodells. Dabei wird jedes einzelne Partikel durch ein zufälliges Gauß'sches Rauschen verzerrt. Dies geschieht auf die gleiche Art und Weise wie bei der Initialisierung des Partikelfilters (vgl. Abschnitt 6.3.1).

Zeile 17 Sicherstellung, dass der Skalierungsfaktor nicht unterhalb von 0.1 fällt. Dies ist notwendig, da ein Partikel, der eine rechteckige Bildregion beschreibt, im Fall einer zu geringen Größe nicht verwendbar bzw. verwertbar ist.

```
1  // state transitions matrix with constant velocity model
2  const float DT = 1;
3  T = (cv::Mat_<float>(NUM_PARAMS, NUM_PARAMS) << 1, 0, DT,  0, 0,
4                                                  0, 1,  0, DT, 0,
5                                                  0, 0,  1,  0, 0,
6                                                  0, 0,  0,  1, 0,
7                                                  0, 0,  0,  0, 1);
8  void ParticleFilter::predict()
9  {
10    for (int i = 0; i < N; i++) {
```

```
11          cv::Mat_<float> noise(NUM_PARAMS, 1);
12          for (int j = 0; j < NUM_PARAMS; j++) {
13              noise(j) = random.gaussian(sigma[j]); // calc noise vector
14          }
15          p[i] = T * p[i] + noise; // predict particles
16
17          // assert min scale
18          float scale = std::max(.1f, p[i](PARAM_SCALE));
19          p[i](PARAM_SCALE) = scale;
20      }
21  }
```

Listing 6.6: *Prediction* der Partikel

6.3.3. Berechnung der Gewichtung

Für die Gewichtungsberechnung der Partikel wird der Histogrammvergleich herangezogen (vgl. Abschnitt 3.1). Dazu wird das Histogramm der vom Kontur-Template umschlossenen Bildregion mit dem Histogramm des durch ein Partikel beschriebenen Rechtecks verglichen (vgl. Abschnitt 5.2.3).

Listing 6.7 zeigt die Gewichtungsberechnung der Partikel. Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 8 Mittels `state_rect()` wird aus dem i-ten Partikel bzw. dessen Positions- und Skalierungsparametern ein Rechteck erzeugt. `frame_roi` ist die durch dieses Rechteck beschriebene Bildregion in der aktuell betrachteten Aufnahme des Objekts.

Zeile 10 Das Histogramm der Bildregion wird berechnet, normiert und mit dem Histogramm-Template verglichen. `templ_hist.match(hist)` liefert als Ergebnis das Distanzmaß d_{bc} nach Gleichung 3.4, S. 7.

Zeile 14 Die Gewichtung des Partikels wird nach Gleichung 5.3, S. 32 berechnet.

```
1  void ParticleFilter::calc_weight(cv::Mat& frame, cv::Size templ_size,
2                                   Histogram& templ_hist, float sigma)
3  {
4      cv::Rect bounds(0, 0, frame.cols, frame.rows);
```

```
5     static Histogram hist;
6
7     for (int i = 0; i < N; i++) {
8         cv::Mat frame_roi(frame, state_rect(templ_size, bounds, i));
9
10        hist.calc_hist(frame_roi, RGB);
11        hist.normalize();
12        float bc = templ_hist.match(hist);
13
14        w[i] = std::exp(-sigma * bc*bc);
15    }
16 }
```

Listing 6.7: Berechnung der Gewichtung

6.3.4. Zustandsschätzung

Mit Hilfe der gewichteten Partikel kann die Schätzung des Systemzustand ermittelt werden. Dies geschieht durch die Berechnung des gewichteten arithmetischen Mittels aller Partikel (vgl. Gleichung 5.4, S. 34). Listing 6.8 zeigt die Berechnung des gewichteten mittleren Systemzustands.

```
1  void ParticleFilter::weighted_mean_estimate()
2  {
3      float sum = 0.f;;
4      cv::Mat_<float> tmp = cv::Mat_<float>::zeros(NUM_PARAMS, 1);
5      for (int i = 0; i < N; i++) {
6          tmp += p[i] * w[i];
7          sum += w[i];
8      }
9      state = tmp / sum;
10 }
```

Listing 6.8: Zustandsschätzung durch die Berechnung des gewichteten arithmetischen Mittels

6.3.5. Resampling

Zum Abschluss des aktuellen Iterationsschritts des Partikelfilters wird die Menge der Partikel neu ausgewählt (*to resample*) und damit die Wahrscheinlichkeitsverteilung aktualisiert. Dieser Abschnitt behandelt das in dieser Arbeit verwendete *Resampling*-Verfahren nach [Thr12], das sogenannte *Resampling Wheel*, bei dem eine Normierung der Gewichtungen nicht notwendig ist. Dazu wird die detaillierte Funktionsweise des Verfahrens im Zusammenhang mit der Implementierung sowie anhand eines exemplarischen Ablaufs erläutert. Für die grundlegende Funktionsweise des *Resampling* sei auf Abschnitt 5.2.5 verwiesen.

Listing 6.9 zeigt zunächst die Implementierung des verwendeten *Resampling*-Verfahrens (*Resampling Wheel*). Im Folgenden sind einige auf einzelne Zeilen bezogene Erläuterungen aufgelistet.

Zeile 3 Der Startindex, bei dem der *Resampling*-Prozess beginnen soll, wird zufällig im Bereich $[0, N]$ gewählt. N ist dabei die Anzahl der verwendeten Partikel. Der Index verweist auf das jeweilige Partikel `p[index]` sowie die ihm zugeordnete Gewichtung `w[index]`.

Zeile 4 Definition der für das *Resampling* benötigten Variablen. Dies ist zum einen die Hilfsvariable `beta` und zum anderen die maximale Gewichtung aller Partikel `w_max`.

Zeile 12 Jeder Iterationsschritt beginnt damit, dass zu der Hilfsvariable `beta` ein zufälliger Wert im Bereich $[0, 2w_{\max}]$ addiert wird. Dies dient der Modellierung der statistischen Unsicherheit, ob ein Partikel ausgewählt wird oder nicht. Je höher die Gewichtung eines Partikels ist, desto höher ist auch die Wahrscheinlichkeit, mit der er im *Resampling*-Prozess ausgewählt wird. Allerdings ist weder garantiert, dass ein bestimmtes Partikel in die neue Menge der Partikel übernommen noch dass er verworfen wird.

Zeile 12 Solange `beta` die aktuell betrachtete Gewichtung `w[index]` überschreitet, wird diese von `beta` subtrahiert und der Index inkrementiert. Erst wenn `beta` klein genug ist, wird das entsprechende Partikel `p[index]` zur neuen Menge der Partikel hinzugefügt. Dies führt dazu, dass Partikel mit einer hohen Gewichtung auch mit einer hohen Wahrscheinlichkeit mehrfach ausgewählt werden und dass niedrig gewichtete Partikel mit einer hohen Wahrscheinlichkeit verworfen werden.

```
1 void ParticleFilter::resample()
2 {
3     int index = random.uniform(0, N);
4     float beta = 0.f;
```

```
5     float w_max = 0.f;
6     for (int i = 0; i < N; i++) { // find maximum weight
7         if (w[i] > w_max)
8             w_max = w[i];
9     }
10
11    for (int i = 0; i < N; i++) {
12        beta += random.uniform(0.f, 2.f * w_max);
13        while (beta > w[index]) {
14            beta -= w[index];
15            index = (index + 1) % N;
16        }
17        p_new[i].push_back(p[index]);
18    }
19    p = p_new;
20 }
```

Listing 6.9: Implementierung des *Resampling Wheel*

Abbildung 6.5 veranschaulicht die Funktionsweise des *Resampling Wheel* durch einen exemplarischen Ablauf. Dieser ist im Folgenden erläutert.

(a) Zunächst wird der Startindex, also die Nummer des Partikels, bei dem das *Resampling* beginnt, zufällig ausgewählt. In diesem Beispiel wird mit dem Partikel `p[index = 6]` bzw. seiner Gewichtung `w[index = 6]` begonnen. Die Hilfsvariable `beta = 0` wird initialisiert. Zu `beta` wird eine zufälliger Wert im Bereich $[0, 2w_{\max}]$ addiert.

(b) Jetzt ist `beta` größer als die Gewichtung des aktuell betrachteten Index (`beta > w[6]`). Deshalb wird diese Gewichtung von `beta` subtrahiert (`beta -= w[6]`). Außerdem wird der Index inkrementiert. Die nun betrachtete Gewichtung `w[7]` ist kleiner als `beta`. In diesem Fall wird das Partikel `p[i]` zur neuen Menge der Partikel `p_new` hinzugefügt.

Durch diesen Ablauf wurde Partikel `p[6]` aufgrund seiner zu geringen Gewichtung aussortiert. `p[7]` hingegen wurde als erstes Partikel in die neue Menge der Partikel übernommen.

(c) Der nächste Iterationsschritt zur Auswahl des zweiten Partikels beginnt. Der aktuell betrachtete Index ist noch immer `index = 7`. Erneut wird ein zufälliger Wert mit Bereich $[0, 2w_{\max}]$ zu `beta` addiert. Dieses Mal wird der Wert von `beta` größer als die nächsten beiden Gewichtungen `w[7]` und `w[8]` zusammen.

(d) Durch den hohen Wert von `beta` werden sowohl `p[7]` als auch `p[8]` bei der Auswahl der Partikel „übersprungen“. Erneut wird die aktuell betrachtete Gewichtung von `beta` subtrahiert und der Index inkrementiert. Dies geschieht solange, bis...

(e) ...der Wert von `beta` wieder kleiner als die aktuell betrachtete Gewichtung ist (`beta <= w[index]`). Dies ist bei einem Index von 1 der Fall, sodass `p[index = 1]` zur neuen Menge der Partikel hinzugefügt wird.

(f) Im nächsten Iterationsschritt ist der zu `beta` addierte zufällige Wert so gering, dass `p[1]` direkt noch einmal ausgewählt wird.

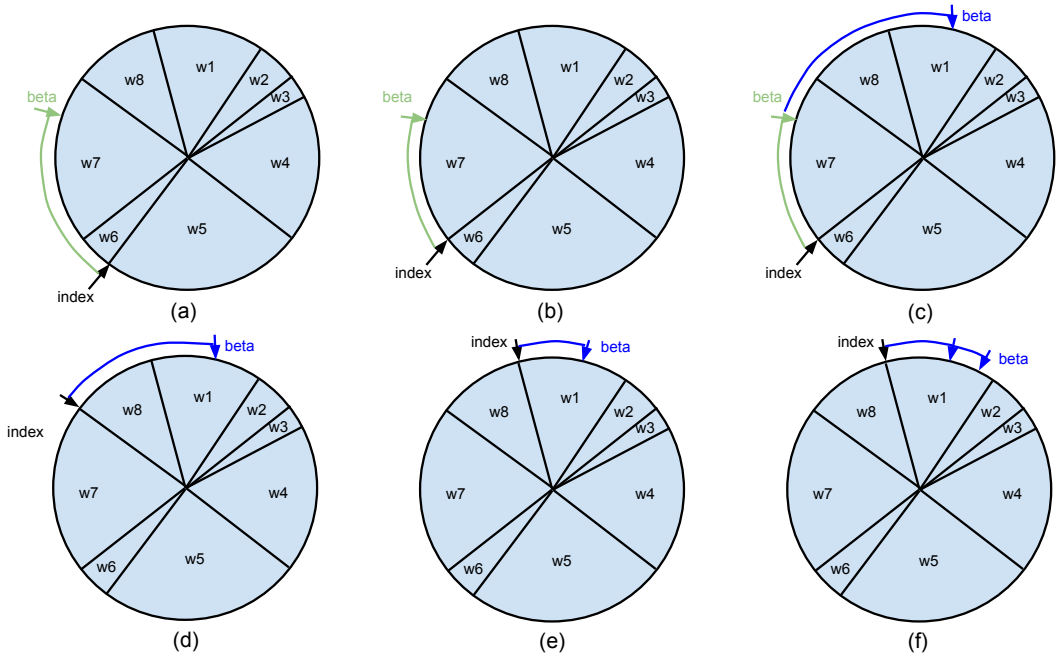


Abbildung 6.5.: Exemplarischer Ablauf des *Resampling Wheel* nach [Thr12]

6.4. Konturbewertung und Ausnahmebehandlung

Dieser Abschnitt behandelt die Bewertung der Kontur in der aktuell betrachteten Aufnahme des zu verfolgenden Objekts, die mit Hilfe des Partikelfilters und der Aktiven Konturen ermittelt wurde. Nachdem ausgehend von der Zustandsschätzung des Partikelfilters eine Konturentwicklung durchgeführt wurde, liegt eine Kontur vor, die im idealen Fall die exakten Objektumrisse beschreibt. Nun gilt es zu bewerten, ob diese Kontur die tatsächliche Konturen

des Objekts widerspiegelt. Dabei soll zum einen festgestellt werden, ob das Objekt nach der Konturentwicklung verloren wurde. Zum anderen sollen eventuelle partielle Verdeckungen detektiert werden. Beide Situationen müssen entsprechend gehandhabt werden.

6.4.1. Verlust des Objekts

Es kann unter gewissen Umständen passieren, dass das Objekt nach der Konturentwicklung verloren wird, d.h. die Kontur nicht den Objektumrissen entspricht. Ist das Objekt z.B. zeitweise vollständig verdeckt, kann die Konturentwicklung fehlschlagen. Auch sich kurzzeitig stark ändernde Lichtverhältnisse können zu einem Verlust des Objekts führen.

An dieser Stelle sei angemerkt, dass die ungefähre Position und Größe des Objekts zuvor korrekt durch das Partikelfilter ermittelt wurde, also das Objekt nicht gänzlich verloren wird. Schlägt die Verfolgung durch das Partikelfilter, auf dessen Zustandsschätzung die Konturermittlung aufbaut, fehl, sind die Grundvoraussetzungen des Algorithmus zur Objektverfolgung (vgl. Abschnitt 6.1.1) nicht erfüllt.

Der Verlust des Objekts nach der Konturentwicklung wird detektiert, indem ein Histogrammvergleich mittels des Bhattacharyya-Koeffizienten vollzogen wird (vgl. Abschnitt 3.1.2). Dazu wird das Histogramm der Ergebnis-Kontur mit dem Histogramm der Template-Kontur verglichen. Das Ergebnis der Konturentwicklung wird als korrekt angesehen, wenn die Bedingung

$$d_{\mathrm{bc}} < threshold_{bc} \tag{6.14}$$

erfüllt ist. Überschreitet das aus dem Histogrammvergleich resultierende Distanzmaß d_{bc} einen bestimmten Schwellenwert $threshold_{bc}$, stimmen die Histogramme also nicht überein, gilt das Objekt als verloren.

Ist der Verlust des Objekts detektiert, ist also die Konturentwicklung fehlgeschlagen, wird eine Ersatz-Kontur als Ergebnis für die aktuell betrachtete Aufnahme des Objekts benötigt. In diesem Fall wird das durch die Zustandsschätzung des Partikelfilters ermittelte Reckteck, das die ungefähre Position und Größe des Objekts beschreibt, als Ergebnis-Kontur verwendet.

6.4.2. Histogramm Adaption

Wurde die Kontur des Objekts erfolgreich gefunden, also Bedingung 6.14 erfüllt, kann das Template-Histogramm ggf. adaptiert werden. Dies hat den Vorteil, dass sich das Histogramm

an eine sich nach und nach verändernde Objektbeschaffenheit anpassen kann. Diese kann beispielsweise durch sich langsam ändernde Lichtverhältnisse verursacht werden. Ein kleiner Unterschied der Lichtverhältnisse zwischen zwei dicht beieinander liegenden Zeitpunkten fällt nicht sonderlich ins Gewicht, doch im Laufe der Zeit kann sich die Beschaffenheit des Objekts immer mehr verändern.

Das Histogramm-Template H_t wird dazu, wie in Gleichung 6.15 definiert, durch das Histogramm der Ergebnis-Kontur H_c adaptiert. Dabei kann die Stärke der Adaption durch den Faktor a gesteuert werden. Die Adaption wird für alle Histogramm-Klassen $i = 1 \ldots n$ durchgeführt.

$$H_t(i) := (1 - a) \cdot H_t(i) + a \cdot H_c(i) \tag{6.15}$$

Weiterhin empfiehlt es sich, das Template-Histogramm nur zu adaptieren, wenn d_{bc} einen bestimmten Schwellenwert $threshold_{adapt}$ nicht überschreitet. Dieser sollte unterhalb des Schwellenwertes zur Detektion des Objektverlusts liegen (vgl. Gleichung 6.16).

$$d_{\text{bc}} < threshold_{adapt} < threshold_{bc} \tag{6.16}$$

6.4.3. Umgang mit partiellen Verdeckungen

Ist die Kontur des zu verfolgenden Objekts erfolgreich durch die Konturentwicklung ermittelt, wird überprüft, ob das Objekt partiell verdeckt ist. Dies kann beispielsweise durch einen Gegenstand oder ein anderes Objekt im Vordergrund verursacht sein. Für die Detektion einer partiellen Verdeckung muss unterschieden werden, ob das Objekt tatsächlich verdeckt ist oder sich nur verformt hat. Dazu werden typische Konturformen bzw. Ansichten, die das Objekt annehmen kann, verwendet. Diese werden auch als *Characteristic Views* bezeichnet. Neben diesen vorab bekannten Ansichten wird zusätzlich das Kontur-Template als einer der *Characteristic Views* benutzt. Damit ist selbst in dem Fall, dass das zu verfolgende Objekt vollkommen unbekannt ist, mindestens eine bekannte Ansicht vorhanden.

Die Detektion einer partiellen Verdeckung erfolgt durch einen Konturvergleich mittels Fourier-Deskriptoren (vgl. Abschnitt 3.3). Dabei wird das Ergebnis der Konturentwicklung mit jedem der *Characteristic Views* verglichen. Nur wenn die Ergebnis-Kontur mit keinem der *Characteristic Views* übereinstimmt, gilt das Objekt als partiell verdeckt. Zwei Konturen stimmen überein,

wenn das Ergebnis des Konturvergleichs d_{fd} einen bestimmten Schwellenwert $threshold_{fd}$ nicht überschreitet (vgl. Gleichung 6.17).

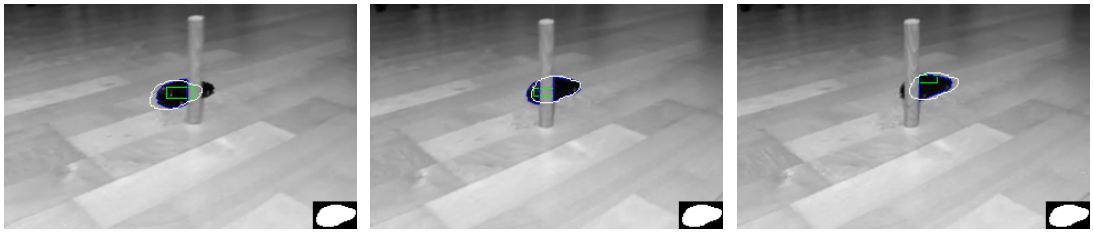
$$d_{\text{fd}} < threshold_{fd} \tag{6.17}$$

Alternativ können für den Vergleich auch die Hu-Momente verwendet werden (vgl. Abschnitt 3.2). Dabei würden die von den Konturen umschlossenen Bildregionen verglichen werden. Allerdings hat die Verwendung von Fourier-Deskriptoren den Vorteil, dass durch das Filtern der hohen Frequenzen (vgl. Abschnitt 3.3.1) nur die ungefähre Form der Konturen übereinstimmen muss, wodurch bessere Ergebnisse erzielt werden können.

Wurde festgestellt, dass eine partielle Verdeckung des Objekts vorliegt, wird eine Ersatz-Kontur als Ergebnis für die aktuell betrachtete Aufnahme des Objekts erstellt. Dazu wird auf die letzte Aufnahme der Bildsequenz, in der keine partielle Verdeckung detektiert wurde, geschaut. Die Grundlage für die Ersatz-Kontur bildet der *Characteristic View*, der in dieser Aufnahme die höchste Übereinstimmung beim Konturvergleich geliefert hat. Es wird also die Form verwendet, die das Objekt zuletzt angenommen hat, bevor es verdeckt wurde.

Zur Erstellung der Ersatz-Kontur wird aus dem Fourier-Deskriptor des verwendeten *Characteristic View* dessen ursprüngliche Kontur rekonstruiert. Diese Kontur wird dann auf die Position und Größe des Ergebnis der Konturentwicklung verschoben und skaliert.

Abbildung 6.6 zeigt exemplarisch die Verfolgung eines Autos, das durch einen Pfeiler partiell verdeckt wird. Darin sind drei unterschiedliche Aufnahmen zu erkennen, in denen die Objektumrisse zwar erfolgreich gefunden wurden, jedoch entweder durch den Pfeiler abgeschnitten (6.6a und 6.6c) oder geteilt wurden (6.6b). Die Zustandsschätzung des Partikelfilters, die als Startzustand für die Konturentwicklung dient, ist in grün und das Ergebnis der Konturentwicklung in blau eingezeichnet. Die weiße Kontur ist die Ersatz-Kontur, die aus dem zuletzt vor der Verdeckung erkannten *Characteristic View* (jeweils rechts unten) erstellt wurde.



(a) (b) (c)

Abbildung 6.6.: Beispiel einer Objektverfolgung unter Auftreten partieller Verdeckungen

7. Ergebnisse

Dieses Kapitel beschreibt die in dieser Arbeit erzielten Ergebnisse. Anhand typischer Szenarien werden die Möglichkeiten des entwickelten Verfahrens zur Verfolgung von Objekten mit Aktiven Konturen und Partikelfiltern in sequentiellen Aufnahmen aufgezeigt. Außerdem wird die Praxistauglichkeit des Verfahrens in Bezug auf die Echtzeitanforderungen einer Objektverfolgung analysiert.

7.1. Anwendungsszenarien

Dieser Abschnitt stellt die Ergebnisse der Objektverfolgung zusammen. Es werden typische Anwendungsszenarien gezeigt, um die Möglichkeiten des Verfahrens nach den in Abschnitt 6.1.1 aufgestellten Anforderungen und Rahmenbedienungen zu veranschaulichen. Die Abbildungen dieses Abschnitts zeigen repräsentative Ausschnitte der Ergebnisse der Objektverfolgung. Die vollständigen Bildsequenzen sind sowohl in Form von Einzelaufnahmen als auch als Video in Anhang B.2 zu finden.

7.1.1. Fisch-Sequenz

Das erste typische Anwendungsszenario ist die Verfolgung eines Fisches, der durch einen Schwarm anderer Fische schwimmt. Dieses Beispiel der Objektverfolgung zeigt, wie es das in dieser Arbeit entwickelte Verfahren erlaubt, die Konturen von sich verformenden Objekten zu verfolgen und dabei gleichzeitig robust gegenüber partiellen Verdeckungen zu sein. Dabei treten Hintergrundänderungen während der Verfolgung auf, die durch die anderen sich bewegenden Fische sowie die für die Aufnahme verwendete nicht-stationäre Kamera verursacht sind.

Die Resultate dieser Objektverfolgung sind in Abbildung 7.1 dargestellt[1]. Darin ist zunächst die im ersten Bild der Sequenz ermittelte Template-Kontur abgebildet (7.1a), die für die Gewich-

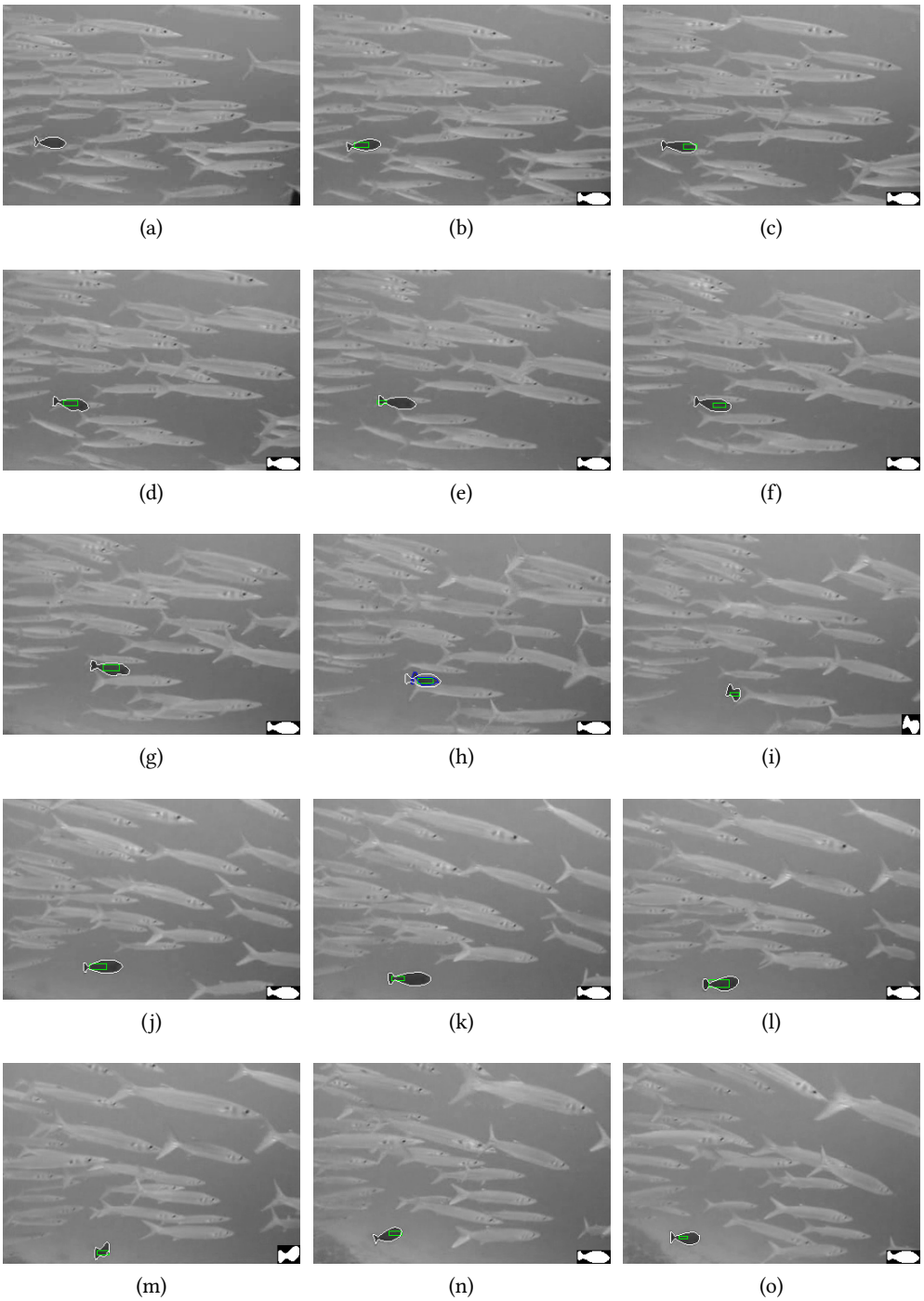
[1] Originalvideo: http://www.ece.iastate.edu/~namrata/research/ContourTrack.html, Zugriff: 26.04.2015

tungsberechnung des Partikelfilters sowie für die Konturbewertung benötigt wird. In allen weiteren Aufnahmen beschreibt das eingezeichnete Rechteck die Zustandsschätzung des Partikelfilters (grün). Ausgehend davon wurde in jedem Bild der Sequenz eine Konturentwicklung zur Findung der Objektkontur durchgeführt. Das Ergebnis der Konturentwicklung dient als Ergebnis des Algorithmus zur Objektverfolgung (weiß), es sei denn es wurde eine partielle Verdeckung detektiert (blau). Im diesem Fall wurde eine Ersatz-Kontur auf Basis des zuletzt erkannten *Characteristic View* erstellt (7.1h). Die in den Aufnahmen mittels der *Characteristic Views* erkannten Konturformen sind jeweils unten rechts eingeblendet. In den meisten Fällen ist dies die Form des Kontur-Templates. In Abbildung 7.1i und 7.1m hingegen hat der Fisch andere Formen angenommen.

Die für diese Objektverfolgung verwendeten Parameter sind in Tabelle 7.1 aufgelistet.

Parameter	**Wert**
Anzahl der Partikel N	100
Gewichtungsberechnung: σ	20
Konturenergie	UM
Iterationen der Konturentwicklung	150
Konturkrümmung: α	0.2
Konturbewertung: $threshold_{bc}$	0.25
Histogramm Adaption: $threshold_{adapt}$	0.1
Histogramm Adaption: a	0.1
Partielle Verdeckungen: $threshold_{fd}$	0.15
Anzahl verwendeter Fourier-Frequenzen K	14

Tabelle 7.1.: Parameter der Objektverfolgung (Fisch-Sequenz)



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

(m) (n) (o)

Abbildung 7.1.: Verfolgung eines sich verformenden, zeitweise partiell verdeckten Fisches

7.1.2. Auto-Sequenz

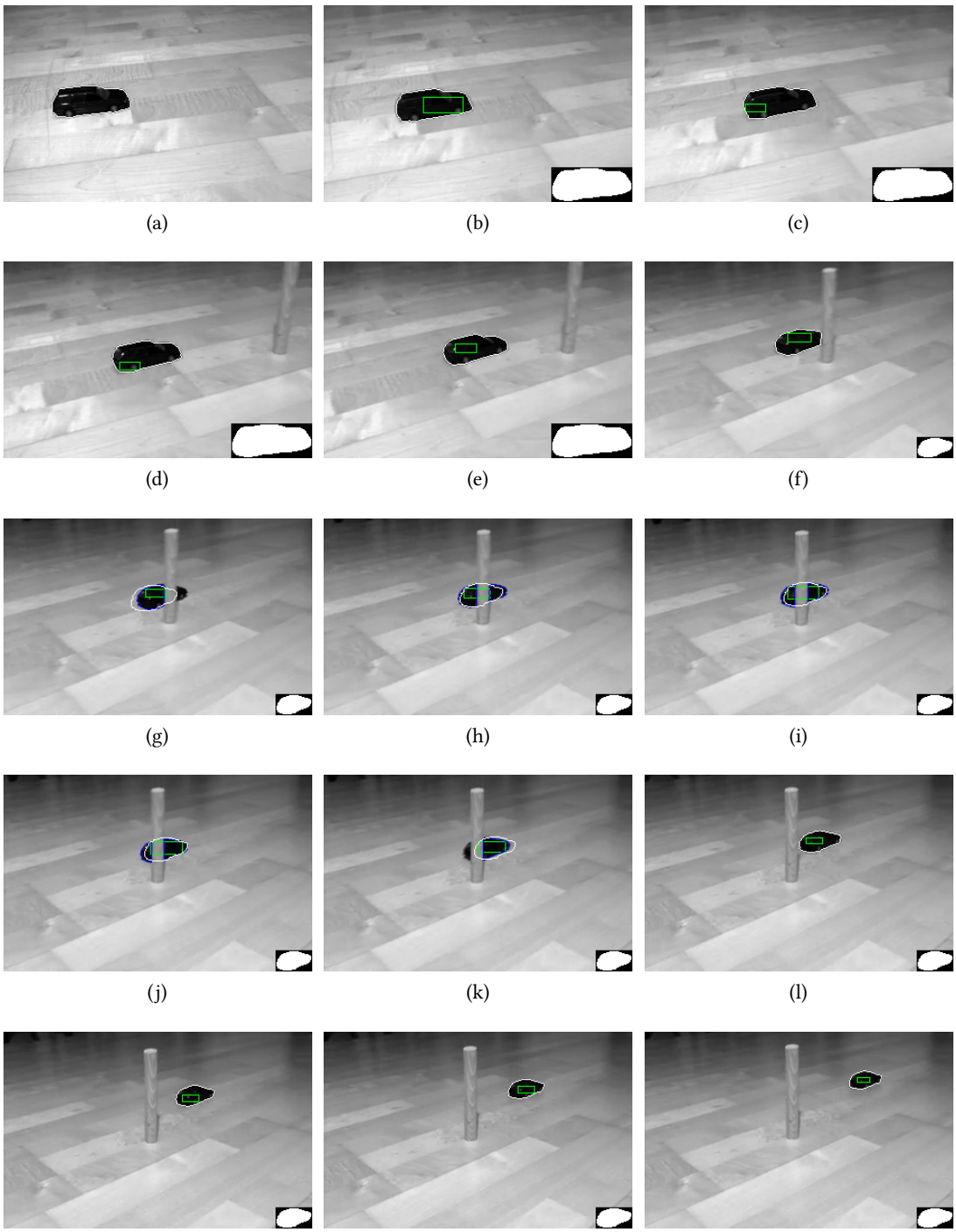
Ein weiteres Anwendungsbeispiel ist die Verfolgung eines Autos, das durch einen Pfeiler kurzzeitig verdeckt wird. Dies ist ein typisches Szenario für die Detektion und den Umgang mit partiellen Verdeckungen während der Objektverfolgung.

In Abbildung 7.2 sind die Resultate dieser Objektverfolgung dargestellt. Auch hier ist im ersten Bild der Sequenz das durch eine Konturentwicklung ermittelte Template abgebildet. Die Zustandsschätzung des Partikelfilters ist als grünes Rechteck eingezeichnet, das als Startzustand für eine Konturentwicklung dient. In den Abbildungen 7.2b bis 7.2f können dadurch die genauen Umrisse des Autos verfolgt werden (weiß). Im weiteren Verlauf (7.2g bis 7.2k) wird das Auto dann von einem Pfeiler partiell verdeckt, was durch einen Konturvergleich mit den *Characteristic Views* detektiert wird. Mit Hilfe des zuletzt erkannten *Characteristic View* (unten rechts) gelingt es, eine Ersatz-Kontur zu erstellen, die nur minimal von den exakten tatsächlichen Konturen des Autos abweicht (weiß). Das Ergebnis der Konturentwicklung ist dort zusätzlich in blau eingezeichnet. Nachdem das Auto den Pfeiler passiert hat (7.2l bis 7.2o), erfolgt die Konturfindung wieder wie vor der partiellen Verdeckung.

Die für diese Objektverfolgung verwendeten Parameter sind in Tabelle 7.2 aufgelistet.

Parameter	**Wert**
Anzahl der Partikel N	200
Gewichtungsberechnung: σ	20
Konturenergie	UM
Iterationen der Konturentwicklung	300
Konturkrümmung: α	0.2
Konturbewertung: $threshold_{bc}$	0.25
Histogramm Adaption: $threshold_{adapt}$	0.1
Histogramm Adaption: a	0.1
Partielle Verdeckungen: $threshold_{fd}$	0.15
Anzahl verwendeter Fourier-Frequenzen K	20

Tabelle 7.2.: Parameter der Objektverfolgung (Auto-Sequenz)



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

(m) (n) (o)

Abbildung 7.2.: Verfolgung eines zeitweise partiell verdeckten Autos

7.1.3. Flugzeug-Sequenz

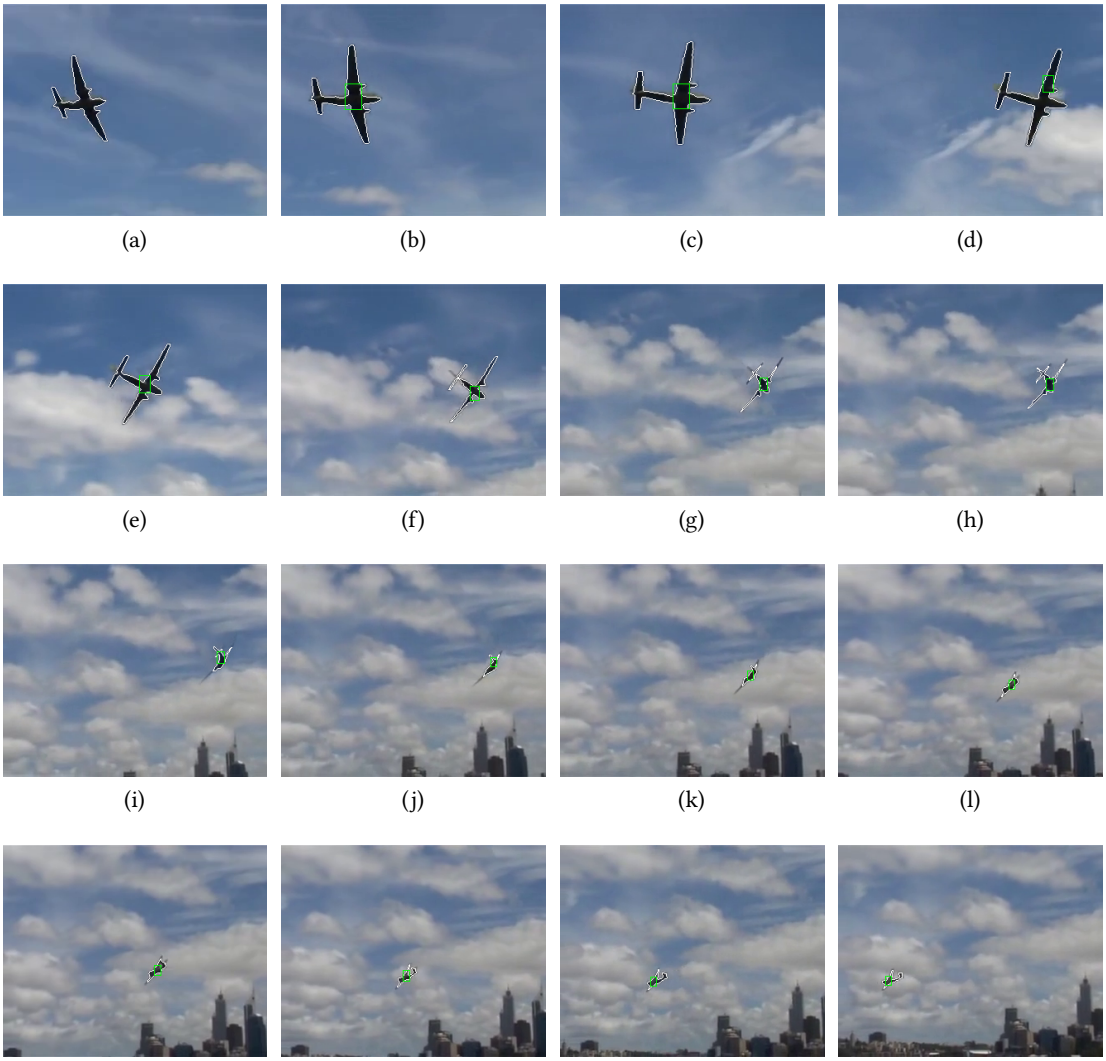
Das nächste Anwendungsszenario ist die Verfolgung eines Flugzeugs. Im Laufe der Bildsequenz ändert sich aufgrund der Flugkurve dessen Ansicht bzw. Form. Dieses Beispiel zeigt die Möglichkeit zur robusten Verfolgung und dennoch exakten Konturfindung von sich schnell bewegenden Objekten mit dem in dieser Arbeit entwickelten Verfahren. Für das Partikelfilter wurden dabei Farbaufnahmen verwendet, was zu einer robusteren Verfolgung der Position und Größe des Objekts führt. Mit in Graustufen konvertierten Aufnahmen hingegen würde diese Objektverfolgung fehlschlagen. Außerdem wird auf die Verwendung von *Characteristic Views* verzichtet, da in diesem Szenario weder partielle Verdeckungen zu erwarten sind noch bestimmte Konturformen erkannt werden sollen.

Abbildung 7.3 zeigt die Resultate dieser Objektverfolgung[2]. Darin sind das ermittelte Template (7.3a), die Zustandsschätzung des Partikelfilters (grün) und das Ergebnis der von dort startenden Konturentwicklung abgebildet (weiß). Die für diese Objektverfolgung verwendeten Parameter sind in Tabelle 7.3 aufgelistet.

Parameter	**Wert**
Anzahl der Partikel N	200
Gewichtungsberechnung: σ	20
Konturenergie	*UM*
Iterationen der Konturentwicklung	300
Konturkrümmung: α	0.2
Konturbewertung: $threshold_{bc}$	0.25
Histogramm Adaption: $threshold_{adapt}$	0.25
Histogramm Adaption: a	0.1

Tabelle 7.3.: Parameter der Objektverfolgung (Flugzeug-Sequenz)

[2] Originalvideo: http://www.csse.uwa.edu.au/~ajmal/databases.html, Zugriff: 26.04.2015



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

Abbildung 7.3.: Verfolgung eines Flugzeugs aus unterschiedlichen Ansichten

7.1.4. Hand-Sequenz

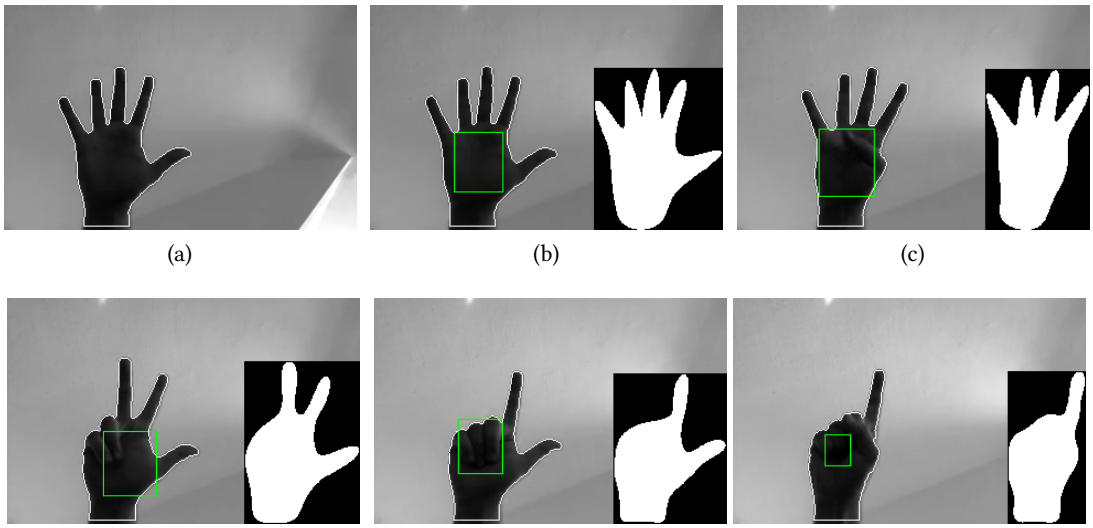
Eine weiteres typisches Anwendungsszenario ist die Verfolgung einer sich verformenden Hand, die im Laufe der Bildsequenz die Formen bestimmter Gesten annimmt. Dieses Beispiel der Objektverfolgung zeigt, wie *Characteristic Views* auch abseits des Umgangs mit partiellen Verdeckungen für die Detektion bestimmter Formen genutzt werden können. So erlaubt es das in dieser Arbeit entwickelte Verfahren, zusätzlich zur Verfolgung der exakten Kontur der Hand, auch vorab definierte Gesten zu erkennen.

In Abbildung 7.2 sind die Resultate dieser Objektverfolgung dargestellt. Darin nimmt die verfolgte Hand nacheinander die Formen zuvor definierter Gesten an. Dies sind in diesem Anwendungsbeispiel die Handzeichen für die Zahlen 1 bis 5. Auch hier ist wieder das ermittelte Kontur-Template (7.4a), die Zustandsschätzung des Partikelfilters (grün) und das Ergebnis der von dort startenden Konturentwicklung dargestellt (weiß). Die jeweils erkannte Geste ist in Form ihres *Characteristic View* unten rechts eingeblendet.

Die für diese Objektverfolgung verwendeten Parameter sind in Tabelle 7.4 aufgelistet.

Parameter	**Wert**
Anzahl der Partikel N	200
Gewichtungsberechnung: σ	20
Konturenergie	UM
Iterationen der Konturentwicklung	400
Konturkrümmung: α	0.2
Konturbewertung: $threshold_{bc}$	0.25
Histogramm Adaption: $threshold_{adapt}$	0.1
Histogramm Adaption: a	0.1
Konturvergleich: $threshold_{fd}$	0.15
Anzahl verwendeter Fourier-Frequenzen K	22

Tabelle 7.4.: Parameter der Objektverfolgung (Hand-Sequenz)



(a) (b) (c)

(d) (e) (f)

Abbildung 7.4.: Verfolgung einer sich verformenden Hand inklusive Gestenerkennung

7.2. Echtzeitanforderungen

Dieses Kapitel analysiert das in dieser Arbeit entwickelte Verfahren in Bezug auf die mit einer Objektverfolgung einhergehenden Echtzeitanforderungen. Um diese einzuhalten, muss die Verarbeitungsrate des Algorithmus zur Objektverfolgung unter der Bildrate der eingehenden Bildsequenz liegen.

Als Grundlage für die Analyse wurden für die in Abschnitt 7.1 präsentierten Anwendungsszenarien Zeitmessungen[3] durchgeführt. Aus der durchschnittlich benötigten Zeit für die Objektverfolgung und der Anzahl der Aufnahmen der verwendeten Bildfrequenz wurde der erreichte Durchsatz an Bildern pro Sekunde (FPS: *Frames Per Second*) errechnet. Die Ergebnisse der Zeitmessungen sind in Tabelle 7.5 enthalten.

An den Zeitmessungen und dem errechneten Durchsatz ist zu erkennen, dass die bewältigte Bildrate stark variieren kann. Die Ursache dafür ist im Wesentlichen der unterschiedliche Rechenaufwand, der für die Konturentwicklung benötigt wird. Dieser resultiert zum einen aus der Anzahl der dafür verwendeten Iterationsschritte und zum anderen aus der Größe des zu verfolgenden Objekts. Da die Konturentwicklung mit der Sparse-Field-Methode nur für die

[3] Die Zeitmessungen wurden auf einem System mit den folgenden Komponenten durchgeführt:
CPU: Intel Core i5-5257U 2.7 GHz, *Speicher*: 8192 MB, LPDDR3-1866, Dual-Channel, *Festplatte*: SSD SM0256G, PCIe 3.0 4x

Sequenz	Frames	Zeit [s] (i. D.)	Durchsatz [FPS]
Fisch	357	6.98607	51.10171
Auto	143	6.62984	21.56916
Flugzeug	325	18.86468	17.22796
Hand	241	31.24566	7.71307

Tabelle 7.5.: Zeitmessungen für die Objektverfolgung

Punkte auf der Kontur (*Zero Level Set*) und die angrenzenden *Level Sets* (vgl. Abschnitt 6.2) durchgeführt wird, bestimmt die Größe der Kontur den benötigten Rechenaufwand und nicht die Größe des betrachteten Bildes.

Trotz der variierenden möglichen Verarbeitungsrate an Bildern pro Sekunde verlässt diese den Bereich einer möglichen Echtzeitanwendung nicht. Während eine Objektverfolgung in der Fisch-Sequenz mit 50.1 FPS deutlich den höchsten Durchsatz aufweist, liegt die erreichte Bildrate bei der Auto-Sequenz (21.6 FPS) und der Flugzeug-Sequenz (17.2 FPS) in einem akzeptablen Bereich. Lediglich die Verarbeitungsrate der Hand-Sequenz liegt mit 7.7 FPS an der unteren Grenze.

8. Fazit

In dieser Arbeit wurde ein Verfahren entwickelt und untersucht, mit dem es durch einfache sequentielle Kameraaufnahmen erreicht werden kann, die Konturen eines sich verformenden Objekts zu verfolgen. Damit ist es möglich, die genauen Objektumrisse zu bestimmen und gleichzeitig robust gegenüber partiellen Verdeckungen, schnellen Bewegungen des Objekts und Hintergrundänderungen zu sein.

Aktive Konturen erlauben es, die Konturen von Objekten exakt zu finden. Durch ihre Anpassungsfähigkeit an bestimmte Bildregionen sind Aktive Konturen ein effektives Mittel zur Verfolgung von Objekten unterschiedlichster Formen. Sie bieten die Möglichkeit, je nach Beschaffenheit des Objekts oder der verwendeten Aufnahmen, spezifiziert zu werden und sind dadurch vielseitig einsetzbar.

Die zusätzliche Verwendung von Partikelfiltern ermöglicht es, den geforderten Grad an Robustheit zur Objektverfolgung beizusteuern. Mit ihnen kann eine zuverlässige Verfolgung auch von sich schnell bewegenden Objekten erreicht werden. Durch die explizite Handhabung partieller Verdeckungen können selbst in diesem Fall sehr genaue Prognosen der tatsächlichen Objektkonturen aufgestellt werden.

Die Implementierung des entwickelten Verfahrens ermöglicht die Anwendung der Objektverfolgung in beliebigen sequentiellen Aufnahmen. Durch das in dieser Arbeit verwendete Approximationsschema zur Implementierung Aktiver Konturen ist es möglich, die mit einer Objektverfolgung einhergehenden Echtzeitanforderungen zu erfüllen, ohne dabei an Genauigkeit zu verlieren.

A. Entwickelte Software

Dieser Teil des Anhangs enthält Informationen über die in dieser Arbeit entwickelte Software. Der Source-Code sowie die *Doxygen*-Dokumentation der Software ist im Verzeichnis `/software` der beiliegenden CD enthalten (s. Anhang B).

Das Git-Repository mit dem Source-Code der entwickelten Software ist unter dem folgenden Link zu finden:

`https://bitbucket.org/p_oltmann/masters-degree-project`

A.1. Voraussetzungen

Für die Verwendung der entwickelten Software sind folgende Komponenten erforderlich:

- **C++11**: C++-Standard
- **CMake**: Zur Compilierung des Source-Codes (2.8+)
- **OpenCV**: Die OpenCV Bildverarbeitungsbibliothek (v2.9.11)
- **Boost**: Die Boost-Bibliothek (v1.57.0)

A.2. Compilierung

Um die Software zu compilieren, müssen folgende Schritte ausgeführt werden:

1. `cmake` aus dem `build`-Verzeichnis ausführen
    - Build-Verzeichnis frei wählbar, es sollte lediglich nicht das Source-Verzeichnis sein.

- Systemabhängige Makefiles generieren, z.B.:
  `-G 'Unix Makefiles'`
  `-G 'MinGW Makefiles'`
- Build-Type spezifizieren:
  `-DCMAKE_BUILD_TYPE=Debug`
  `-DCMAKE_BUILD_TYPE=Release`
- Compiler-Flags setzen (optional):
  `-DSHOW_CONTOUR_EVOLUTION` – Anzeige der Konturentwicklung
  `-DTIME_MEASUREMENT` – Zeitmessung

2. `make` ausführen

Listing A.1 zeigt die Befehle für eine typische Compilierung.

```
mkdir build && cd build
cmake -G 'Unix Makefiles' -DCMAKE_BUILD_TYPE=Release ..
make
./pf_hist_ac -h # print help information
```

Listing A.1: Befehle für eine typische Compilierung

A.3. Parametrisierung

Die Parametrisierung der Software wird über eine YAML-Datei vorgenommen. Diese kann bei der Ausführung mittels der Argumente `-f <filename>.yml` spezifiziert werden. Alternativ kann die Standard-Datei `../parameterization.yml` (ein Verzeichnis übergeordnet dem Build-Verzeichnis) verwendet werden.

Eine Liste und Beschreibung aller Parameter ist in Listing A.2 zu finden.

```
%YAML:1.0

# number of particles, default: 100
num_particles: 100

# number of iterations (contour evolution steps), default: 10
num_iterations: 150

```

```
# sigma for weight calculation, default: 20
sigma: 20

# histogram matching threshold, default: 0.25
bc_threshold: 0.25

# histogram matchimg threshold for adpation, default: 0.1, 0: no adaption
bc_threshold_adapt: 0.1

# histogram adaption factor (0 for using default)
a: 0.1

# number of fourier frequencies for filtering, default: 10
# the value used ist 2 * num_fourier to always obtain an even-numbered value
num_fourier:  7

# fourier descriptor matching threshold, default: 0.15, 1: none
fd_threshold: 0.15

# 1: select starting rectangle with mouse, 0: use specified
select_start_rect: 0

# starting rectangle for contour evolution on first frame
start_rect: [  35, 150,  40,  20 ] # fish

# path to input video or image sequence
input_path: "../input/palau2_gray_cropped/palau2_frames_%04d.png"

# number of input device, only used if input_path is empty, default: 0
input_device: 0

# charateristic views, list of strings to filepaths
char_views: ["../input/char_views/cv_fish/cv_fish_back_right_down_s.png",
             "../input/char_views/cv_fish/cv_fish_back_right_up_s.png"]

# file name for video and image output, leave empty to use 'out'
# file extension is added automatically
# output_file_name: "fish_result"

# path to output video directory, leave empty for no output
```

```
# output_path: "../output/fish/"

# frames per second for saved video, <= 0 or empty for taking source fps
fps: 25.0

# path to output directory for each frame as an image
# detailed view is added to 'details/' that needs to be created in advance
# save_img_path: "../output/fish/fish_img/"

# matching values that can be read in with matlab, leave empty for no output
# matlab_file_path: "../matlab/matlab_out.m"

# contour evolution detailed parameters, defaults used if not specified

# 0: CHAN_VESE (UM), 1: YEZZI (MS), default: 0
method: 0

# true/false (0/1), default: 0
localized: 0

# radius of localized regions, default: 18
rad: 18

# curvature weight (higher -> smoother), default: 0.2
alpha: 0.2
```

Listing A.2: Exemplarische Parametrisierung

B. Inhalt der beiliegenden CD

Dieser Arbeit liegt eine CD mit den im Folgenden aufgelisteten Inhalten bei.

`/pdf` Die Masterarbeit im PDF-Format

`/results` Die vollständigen Bildsequenzen der in Kapitel 7 präsentierten Ergebnisse sowohl in Form von Einzelaufnahmen als auch als Video

`/software` Der Source-Code und die *Doxygen*-Dokumentation der entwickelten Software

`/videos` Videos zur Veranschaulichung der Funktionsweise von Partikelfiltern (Kapitel 5) sowie der Konturentwicklung mit der Level-Set-Methode (Abschnitt 4.2)

B.1. Videos

Im Verzeichnis `/videos` der beiliegenden CD sind folgende Videos enthalten:

```
1 videos
2 |-- active_contours_example
3 |   |-- plane_evolving_contour.mp4
4 |   `-- plane_level_set.mp4
5 `-- robot_localization
6     `-- robot_localization.mp4
```

Listing B.1: Liste der dieser Arbeit angefügten Videos

Videoquellen:

Zeile 2 [Lan09]

Zeile 5 [TBF05]

B.2. Ergebnisse

Im Verzeichnis `/results` der beiliegenden CD sind die vollständigen Bildsequenzen der in Kapitel 7 präsentierten Ergebnisse sowohl in Form von Einzelaufnahmen als auch als Video enthalten. Im Folgenden ist der Inhalt des Verzeichnisses aufgelistet.

```
results
|-- car
|   |-- car_char_views
|   |-- car_img
|   |-- car_init.mp4
|   |-- car_result.mp4
|   |-- car_result_details.mp4
|   `-- info.txt
|-- fish
|   |-- fish_char_views
|   |-- fish_img
|   |-- fish_init.mp4
|   |-- fish_result.mp4
|   |-- fish_result_details.mp4
|   `-- info.txt
|-- hand_count
|   |-- hand_count_char_views
|   |-- hand_count_img
|   |-- hand_count_init.mp4
|   |-- hand_count_result.mp4
|   |-- hand_count_result_details.mp4
|   `-- info.txt
`-- plane
    |-- info.txt
    |-- plane_img
    |-- plane_init.mp4
    |-- plane_result.mp4
    `-- plane_result_details.mp4
```

Listing B.2: Liste der dieser Arbeit angefügten Ergeniss-Dateien

Abbildungsverzeichnis

Listings

Glossar

CFL-Bedingung	Bedingung für die Diskretisierung zeitabhängiger partieller Differentialgleichungen nach Courant-Friedrichs-Lewy (vgl. Abschnitt 4.2.3)
Characteristic View	Typische Ansicht eines Objekts, das zur Identifikation von Konturformen eines Objekts verwendet wird
DFT	Diskrete Fourier-Transformation (vgl. Gleichung 3.18, S. 11)
Doxygen	Werkzeug für die Generierung einer Dokumentation aus kommentiertem Source-Code.
Energiefunktional	Mathematische Beschreibung einer Konturenergie (vgl. Abschnitt 4.1 und 4.3)
Fourier-Deskriptor	Eine Reihe von aus Konturpunkten umgewandelten Kennwerten, um die Konturform zu identifizieren.
Fourier-Frequenzen	Die Kennwerte eines Fourier-Deskriptors sortiert nach Einflussstärke auf die Kontur
Fourier-Koeffizient	Ein Kennwert eines Fourier-Deskriptors
Hu-Momente	Flächenmerkmale für die Identifikation von Bildregionen
Konturenergie	Haupteigenschaft einer Aktiven Kontur (vgl. Abschnitt 4.1)
Konturentwicklung	Abänderung einer Aktiven Kontur zur Findung der Umrisse bestimmter Bildregionen (vgl. Abschnitt 4.1)
Level Set	Menge von Punkten einer dreidimensionalen Oberfläche, die auf derselben horizontalen Ebene liegen (vgl. Abschnitt 4.2)
Level-Set-Methode	Methode zur Repräsentation und Entwicklung Aktiver Konturen (vgl. Abschnitt 4.2)
MS	Mean Separation Energy, eine spezifische Konturenergie (vgl. Abschnitt 4.3.2)
OpenCV	Eine Softwarebibliothek für Bildverarbeitung und maschinelles Sehen

Partikel	Eine Zustandshypothese zur Repräsentation einer Wahrscheinlichkeitsverteilung (vgl. Kapitel 5)
Prediction	Adaption von Partikeln zur Prognose eines gesuchten Systemzustands (vgl. Kapitel 5)
Resampling	Aktualisierung einer Wahrscheinlichkeitsverteilung durch Erneuerung der Menge der Samples
Samples	Gewichtete Stichproben zur Repräsentation einer Wahrscheinlichkeitsverteilung
SDF	Signed Distance Function (vgl. Abschnitt 4.2.1)
Sparse-Field-Methode	Methode zur Implementierung Aktiver Konturen mit der Level-Set-Methode
Template	In dieser Arbeit: Kontur oder Histogramm einer optimalen Bildregion
UM	Uniform Modeling Energy, eine spezifische Konturenergie (vgl. Abschnitt 4.3.1)
Zero Level Set	Menge von Punkten einer dreidimensionalen Oberfläche, die auf der Nullebene liegen. Implizite Kontur, die durch die Oberflächenfunktion der Level-Set-Methode repräsentiert wird (vgl. Abschnitt 4.2)

Literaturverzeichnis

[AMGC02] Arulampalam, M.S. ; Maskell, S. ; Gordon, N. ; Clapp, T.: A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. In: *Signal Processing, IEEE Transactions on* 50 (2002), Nr. 2, S. 174–188. http://dx.doi.org/10.1109/78.978374. – DOI 10.1109/78.978374. – ISSN 1053–587X

[BD11] Bimbo, Alberto D. ; Dini, Fabrizio: Particle filter-based visual tracking with a first order dynamic model and uncertainty adaptation. In: *Computer Vision and Image Understanding* 115 (2011), Nr. 6, 771 – 786. http://dx.doi.org/http://dx.doi.org/10.1016/j.cviu.2011.01.004. – DOI http://dx.doi.org/10.1016/j.cviu.2011.01.004. – ISSN 1077–3142

[Bha43] Bhattacharyya, A.: On a measure of divergence between two statistical populations defined by their probability distributions. In: *Bulletin of the Calcutta Mathematical Society* 35 (1943), S. 99–109

[BI98] Blake, Andrew ; Isard, M.: *Active Contours: The Application of Techniques from Graphics,Vision,Control Theory and Statistics to Visual Tracking of Shapes in Motion.* 1st. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1998 http://www.robots.ox.ac.uk/~contours/. – ISBN 3540762175

[CKS97] Caselles, Vicent ; Kimmel, Ron ; Sapiro, Guillermo: Geodesic Active Contours. In: *Int. J. Comput. Vision* 22 (1997), Februar, Nr. 1, 61–79. http://dx.doi.org/10.1023/A:1007979827043. – DOI 10.1023/A:1007979827043. – ISSN 0920–5691

[Coo08] Cootes, Tom: *Snakes (Lecture Notes).* http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/CS3432/index.html. Version: 2008

[CR07] Chaudhury, Kunal N. ; Ramakrishnan, K. R.: Stability and convergence of the level set method in computer vision. In: *Pattern Recognition Letters* 28 (2007), Nr. 7, 884 – 893. http://dx.doi.org/http://dx.doi.org/10.1016/j.patrec.2006.12.003. – DOI http://dx.doi.org/10.1016/j.patrec.2006.12.003. – ISSN 0167–8655

[CV01] Chan, T.F. ; Vese, L.A.: Active Contours Without Edges. In: *Image Processing, IEEE Transactions on* 10 (2001), Nr. 2, S. 266–277. http://dx.doi.org/10.1109/83.902291. – DOI 10.1109/83.902291. – ISSN 1057–7149

[DSYT10] Dambreville, Samuel ; Sandhu, Romeil ; Yezzi, Anthony ; Tannenbaum, Allen: A Geometric Approach to Joint 2D Region-Based Segmentation and 3D Pose Estimation Using a 3D Shape Prior. In: *SIAM J. Img. Sci.* 3 (2010), März, Nr. 1, 110–132. http://dx.doi.org/10.1137/080741653. – DOI 10.1137/080741653. – ISSN 1936–4954

[GSS93] Gordon, N.J. ; Salmond, D.J. ; Smith, A. F M.: Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In: *Radar and Signal Processing, IEE Proceedings F* 140 (1993), Nr. 2, S. 107–113. – ISSN 0956–375X

[GW08] Gonzalez, Rafael C. ; Woods, Richard E.: *Digital Image Processing*. 3. Prentice Hall, 2008. – ISBN 978–0–13–168728–8

[Hu62] Hu, Ming-Kuei: Visual pattern recognition by moment invariants. In: *Information Theory, IRE Transactions on* 8 (1962), Februar, Nr. 2, S. 179–187. http://dx.doi.org/10.1109/TIT.1962.1057692. – DOI 10.1109/TIT.1962.1057692. – ISSN 0096–1000

[IB98] Isard, Michael ; Blake, Andrew: CONDENSATION — Conditional Density Propagation for Visual Tracking. In: *International Journal of Computer Vision* 29 (1998), Nr. 1, 5–28. http://dx.doi.org/10.1023/A:1008078328650. – DOI 10.1023/A:1008078328650. – ISSN 0920–5691

[KWT88] Kass, Michael ; Witkin, Andrew ; Terzopoulos, Demetri: Snakes: Active contour models. In: *International Journal of Computer Vision* 1 (1988), Nr. 4, 321–331. http://dx.doi.org/10.1007/BF00133570. – DOI 10.1007/BF00133570. – ISSN 0920–5691

[Lan09] Lankton, S.: *Thesis proposal: Local Statistics for Medical Image Processing*. http://www.shawnlankton.com/2009/03/proposal-presentation/. Version: März 2009

[Lom06] Lombaert, Hervé: *Level Set Method: an Explanation*. http://step.polymtl.ca/rv101/levelset/. Version: März 2006

[LST11] Lee, Jehoon ; Sandhu, R. ; Tannenbaum, A.: Monte Carlo sampling for visual pose tracking. In: *Image Processing (ICIP), 2011 18th IEEE International Conference on*, 2011, S. 501–504

[LST13] Lee, Jehoon ; Sandhu, Romeil ; Tannenbaum, Allen: Particle Filters and Occlusion Handling for Rigid 2D-3D Pose Tracking. In: *Computer Vision and Image Understanding* 117 (2013), Nr. 8, 922 – 933. http://dx.doi.org/http://dx.doi.org/10.1016/j.cviu.2013.04.002. – DOI http://dx.doi.org/10.1016/j.cviu.2013.04.002. – ISSN 1077–3142

[LT08] Lankton, S. ; Tannenbaum, A.: Localizing Region-Based Active Contours. In: *Image Processing, IEEE Transactions on* 17 (2008), Nr. 11, S. 2029–2039. http://dx.doi.org/10.1109/TIP.2008.2004611. – DOI 10.1109/TIP.2008.2004611. – ISSN 1057–7149

[NFHS11a] Nischwitz, Alfred ; Fischer, Max ; Haberäcker, Peter ; Socher, Gudrun: *Computergrafik und Bildverarbeitung: Band I: Computergrafik*. Bd. 2. 3. Vieweg+Teubner Verlag, 2011 dx.doi.org/10.1007/978-3-8348-8300-1. – ISBN 978–3–8348–1304–6

[NFHS11b] Nischwitz, Alfred ; Fischer, Max ; Haberäcker, Peter ; Socher, Gudrun: *Computergrafik und Bildverarbeitung: Band II: Bildverarbeitung*. Bd. 2. 3. Vieweg+Teubner Verlag, 2011 dx.doi.org/10.1007/978-3-8348-8300-1. – ISBN 978–3–8348–1712–9

[Olt14a] Oltmann, Peter: Implementation von Aktiven Konturen mit der Level-Set-Methode, Projekt 1 / HAW Hamburg. 2014. – Forschungsbericht

[Olt14b] Oltmann, Peter: Visuelles Tracking unter der Verwendung von aktiven Konturen und Partikelfiltern, Anwendungen 2 / HAW Hamburg. 2014. – Forschungsbericht

[PF77] Persoon, E. ; Fu, King-Sun: Shape Discrimination Using Fourier Descriptors. In: *Systems, Man and Cybernetics, IEEE Transactions on* 7 (1977), März, Nr. 3, S. 170–179. http://dx.doi.org/10.1109/TSMC.1977.4309681. – DOI 10.1109/TSMC.1977.4309681. – ISSN 0018–9472

[RVTY07] Rathi, Y. ; Vaswani, N. ; Tannenbaum, A. ; Yezzi, A.: Tracking Deforming Objects Using Particle Filtering for Geometric Active Contours. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29 (2007), Nr. 8, 1470–1475. http:

//dx.doi.org/10.1109/TPAMI.2007.1081. – DOI 10.1109/TPAMI.2007.1081. – ISSN 0162–8828

[Set99] Sethian, J.A.: *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999 (Cambridge Monographs on Applied and Computational Mathematics). http://math.berkeley.edu/~sethian/2006/Publications/Book/2006/book_1999.html. – ISBN 9780521645577

[Sha09] Shawn Lankton: Sparse Field Methods - Technical Report. Version: Juli 2009. http://www.shawnlankton.com/2009/04/sfm-and-active-contours/. 2009. – Forschungsbericht

[SSO94] Sussman, Mark ; Smereka, Peter ; Osher, Stanley: A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow. In: *Journal of Computational Physics* 114 (1994), Nr. 1, 146 – 159. http://dx.doi.org/http://dx.doi.org/10.1006/jcph.1994.1155. – DOI http://dx.doi.org/10.1006/jcph.1994.1155. – ISSN 0021–9991

[TBF05] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic Robotics*. MIT Press, 2005 http://mitpress.mit.edu/books/probabilistic-robotics. – ISBN 9780262201629

[Thr12] Thrun, Sebastian: *Artificial Intelligence for Robotics (Udacity Course)*. Udacity, 2012 https://www.udacity.com/course/cs373

[Whi98] Whitaker, Ross T.: A Level-Set Approach to 3D Reconstruction from Range Data. In: *International Journal of Computer Vision* 29 (1998), Nr. 3, 203–231. http://dx.doi.org/10.1023/A:1008036829907. – DOI 10.1023/A:1008036829907. – ISSN 0920–5691

[YTW02] Yezzi, Anthony J. ; Tsai, Andy ; Willsky, Alan: A Fully Global Approach to Image Segmentation via Coupled Curve Evolution Equations. In: *Journal of Visual Communication and Image Representation* 13 (2002), 195 – 216. http://dx.doi.org/http://dx.doi.org/10.1006/jvci.2001.0500. – DOI http://dx.doi.org/10.1006/jvci.2001.0500. – ISSN 1047–3203

[ZL02] Zhang, D. ; Lu, G.: A Comparative Study on Shape Retrieval Using Fourier Descriptors with Different Shape Signatures. In: *Fifth Asian Conference on Computer*

Vision (ACCV2002), Asian Federation of Computer Vision Societies (AFCV), Januar 2002, S. 646–651

[ZR72] Zahn, Charles T. ; Roskies, Ralph Z.: Fourier Descriptors for Plane Closed Curves. In: *Computers, IEEE Transactions on* C-21 (1972), März, Nr. 3, S. 269–281. http://dx.doi.org/10.1109/TC.1972.5008949. – DOI 10.1109/TC.1972.5008949. – ISSN 0018–9340

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 4. Mai 2015 Peter Oltmann