Detektion, Tracking und Robotergestützte Handhabung Linearer Deformierbarer Objekte

eingereichter
Abschlussbericht zur DIPLOMARBEIT
von

cand. ing. Binh An Tran

geb. am 25.09.1983
wohnhaft in:
Willi Graf Straße 11
80805 München
Tel.: 0176 24775377

Lehrstuhl für
STEUERUNGS- und REGELUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer:	Dipl.-Inf. Thomas Müller, Dr.-Ing. Angelika Peer
Beginn:	15.11.2009
Zwischenbericht:	15.02.2010
Abgabe:	15.05.2010

Zusammenfassung

Die Arbeit ist im Transferprojekt T4 des Sonderforschungsbereichs SFB 453, „Wirklichkeitsnahe Telepräsenz und Teleaktion“ angesiedelt. Im Transferprojekt werden Forschungsergebnisse aus dem Teilprojekt I4 des SFB 453 in industrielle Anwendungen transferiert, wobei vorhandene Algorithmik und Methodik angepasst und erweitert werden müssen. Der zentrale Fokus des Projekts liegt hierbei in den Bereichen Bildverarbeitung und experimentelle industrielle Robotik. Konkrete Fragestellungen sind beispielsweise:

- Erkennung von deformierbaren (linearen) Objekten, z. B. Gummischläuche, Kabel oder Dichtungen,
- Visuelle Verfolgung der erkannten Objekte im Arbeitsraum,
- Modellierung, z. B. Vorhersagen zum Objektverhalten,
- Autonome, zielgerichtete Handhabung, z. B. Roboter-Greifen oder exakte Positionierung,
- Interaktion (teilautonomes Trainieren und Erlernen von Tasks), z. B. mit Hilfe von Telepräsenzsystemen.

Die Arbeit setzt auf einem existierenden Robotik- und Bildverarbeitungs-Framework auf. Damit können z. B. der zu verwendende Roboter (Mitsubishi RV6-S) oder sensorische Peripherie (Kameras, Kraft-Moment-Sensor, 3D Eingabegeräte) ausgelesen, bzw. angesteuert werden.

Abstract

The thesis is a part of the transfer project T4 of the special research field SFB 453 „High-Fidelity Telepresence and Teleaction“. In the transfer project, research results are transferred from the subproject I4 of the SFB 453 into industrial applications. The central focuses of the project are image processing and experimental industrial robotics. Concrete question formulations are for example:

- Recognition of deformable (linear) objects, e.g. rubber ropes, cables or gaskets,
- Visual tracking of the recognized objects in the working space,
- Modeling, e.g. forecasting the object behavior,
- Autonomous, precise Handling, e.g. picking or exact positioning,
- Communication (partially autonomous training and learning of tasks), e.g. with the aid of telepresence systems.

The work sets on an existing Robotics- and image processing framework. So that the robot (Mitsubishi RV6-S) or sensory periphery (Cameras, force-torque sensors, three dimensional input devices) can be used.

Danksagung

Ich bedanke mich herzlich bei Dipl. -Inf. Thomas Müller für die hilfreiche Unterstützung während der Bearbeitung dieser Diplomarbeit.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Einleitung

Die meisten Arbeiten in der Robotik haben sich mit starren Objekten mit einfachen, regelmäßigen Formen beschäftigt. Objektgeometrie ist in diesem Fall vorhersehbar und unterscheidet sich nur in kleinen Toleranzen von einer Instanz eines Objekts zur anderen. Konventionelle Roboter und Roboter-Programmierungssprachen stützen sich auf diese Regelmäßigkeit. Der Roboter muss die genaue Form und Lage von den Objekten in der Umgebung sowie die entsprechenden Aktionen wissen, die durchgeführt werden sollen.
Viele Objekte in vielen industriellen Anwendungen sind heutzutage aber biegsam und haben sehr unterschiedliche Formen. Flexible Dichtungen, Gummi, Stoff, Papier, und Schläuche sind gängige Komponenten in dieser Gruppe. Materialien wie Draht oder Seil sind schwer zu handhaben und nicht genau modellierbar. Konzeptionell einfache Manipulationen sind schwierig zu spezifizieren, und einfache Aktionen können komplexe Änderungen an den Objekten erzeugen. In diesem Fall sind die Roboter, die auf genaue Informationen und feine Toleranzen verlassen, schwierig zu programmieren und sehr fehleranfällig.
Flexible Objekte können in drei Kategorien von linearen, blattförmigen und dreidimensionalen Objekten eingeordnet werden. Für jede dieser Kategorien kann eine Liste der Werkstück-Komponenten gegeben werden, mit denen Versuche unternommen wurden, um sie automatisch für spezifische Anwendungen in bestimmten Bereichen in der Industrie zu manipulieren (1.1).
Unter Bezugnahme auf deformierbare lineare Objekte (DLO) implizieren wir Objekte, die in physischer Form ähnlich wie Drähten oder Spulen sind. Manipulation von deformierbaren linearen Teilen wie Seile, Drähte und Spulen für präzise Montage ist eine wichtige industrielle Anwendung, z. B. die Einfügung eines verformbaren Balkens, wie Seil, in ein starres Loch in der Strom-Industrie, die automatisierte Montage von Kabelbäumen in Endprodukten wie Autotüren in der Automobilindustrie oder die Befestigung in der Verpackungsindustrie.
Um das Objekt manipulieren zu können, muss es zuerst detektiert werden, wenn es

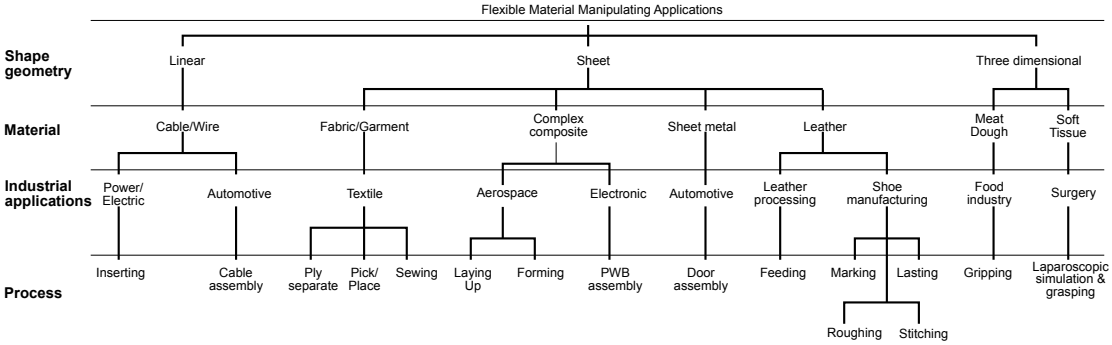


Abbildung 1.1: Kategorie der Anwendungen von deformierbaren Objekten in der Industrie [Nan02]

sich im Arbeitsbereich von dem Roboter befindet. Danach sollen die Position sowie der Zustand des Objekts verfolgt werden. Abhängig von der Lage des Objekts und der Aufgabe muss man schließlich entsprechende Aktionen planen.

1.2 Übersicht der Arbeit

Versuchsobjekte dieser Arbeit sind Dichtungen, wie sie in Haushaltgeräte z. B. Waschmaschinen eingesetzt werden (siehe Abbildung 1.2). Ziel der Arbeit ist De-

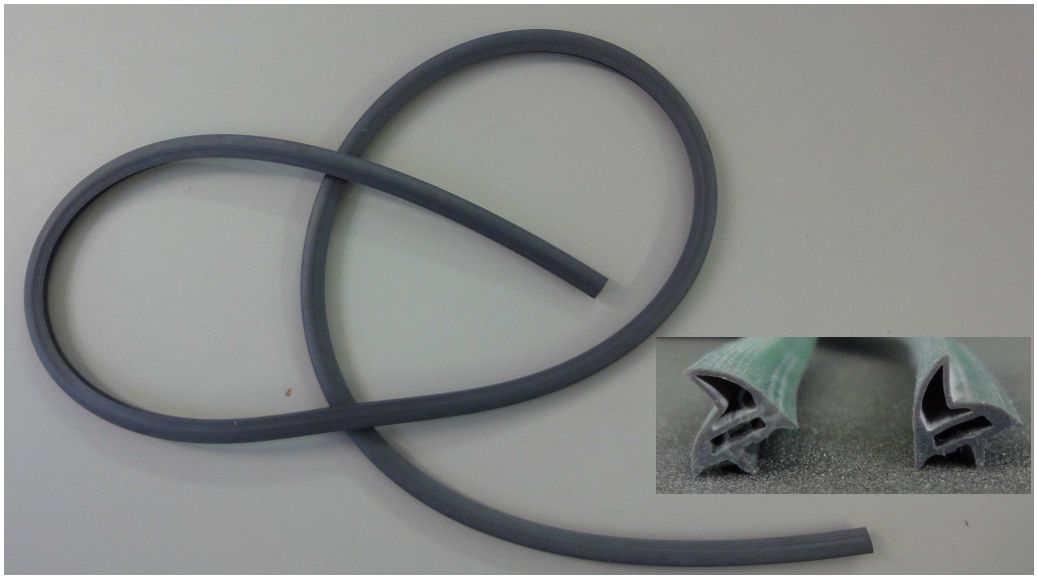


Abbildung 1.2: Die benutzte Dichtung mit ihrem Querschnitt

tektion der Dichtungen, wenn sie sich im Arbeitsbereich des Roboters befinden. Es folgt eine Strukturanalyse, um entsprechende Aktionen planen zu können. Wenn die Struktur der Dichtung keinen unauflösbaren Knoten mehr enthält, wird ein Ende

der Dichtung vom Roboter angehoben und an einen Greifer übergeben. Schließlich wird das andere Ende von dem Roboter geholt und an dem vorher befestigten Ende geklebt. Der Prozess soll garantieren, dass sich die Dichtung nicht gedreht, und die beiden Enden in richtiger Richtung zusammengeklebt werden.
Die meisten Verfolgungsanwendungen benutzen verschiedene dynamische Modelle für die Objekte, die verfolgt werden sollen. Da unser Objekt deformierbar ist, ist es schwierig, sein Modell genau zu erstellen. Die Level-Set-Methode wird deshalb gewählt, weil die Modelle des Objekts in einer energiebasierten Funktion eingebettet werden. Das Verfolgungsproblem wird zu der Minimierung einer Energiefunktion gewandelt. Ein Problem der Level-Set-Methode ist, dass eine partielle Differenzialgleichung gelöst werden muss. Dieser Prozess ist sehr rechenintensiv und zeitaufwendig. In dieser Arbeit wird ein schnelles Implementierungsschema benutzt, das nur zwei Listen von Punkten um die Kontur des Objekts bearbeitet, um die Level-Set-Methode zu beschleunigen. Dieses Schema muss keine Differenzialgleichung lösen, deshalb ist unsere Anwendung echtzeitfähig.
Die Arbeit wird demnächst in sieben Kapitel gegliedert. Im zweiten Kapitel wird die Verfolgung von Objekten gehandelt. Zuerst wird eine Literaturübersicht über Objektverfolgung gegeben. Die wichtigen Schritte sowie die üblichen Verfolgungsmethoden werden kurz vorgestellt. Anschließend wird eine Konturverfolgungsmethode, die für unser Objekt geeignet ist, ausführlich erklärt.
Bevor der Roboter anfängt, das deformierbare Objekt zu manipulieren, soll der Zustand des Objekts analysiert werden. Dieser Schritt garantiert, dass es keine festen Knoten entsteht, wenn das Objekt manipuliert wird. Um die Struktur des linearen Objekts zu erkennen, ist es erwünscht, das Skelett von dem Objekt zu bekommen. Diese Darstellung ist näher an der menschlichen Wahrnehmung, deshalb erlaubt sie eine einfache Struktur-Analyse und intuitives Design von Erkennungsalgorithmen. Darüber hinaus kann die Reduzierung von Bilddaten die Kontur-Verzerrungen beseitigen, während die wichtigen topologischen und geometrischen Eigenschaften noch gehalten werden können. Das dritte Kapitel wird die Methode um die Struktur des Objekts zu analysieren erklären.
Um die beiden Enden aneinander in richtiger Richtung kleben zu können müssen die Orientierungen von den Enden zuerst festgestellt werden. In dieser Arbeit wird Template-Matching-Methode angewendet, um die Orientierung der Enden relative zu einem vorher definierten Template zu bestimmen. Drei Methoden wurden im System implementiert und mit einander vergleichen. Der Brute-Force und Ciratefi-Algorithmus sind regionbasierte Ansätze und der SURF-Algorithmus ist eine merkmalbasierte Methode. Im vierten Kapitel werden die Implementierungsschritten von diesen Algorithmen präsentiert.
In dieser Arbeit werden der industrielle Roboter RV6-S von Mitsubishi zur Manipulation und zwei industrielle Kameras zur Bildaufnahme benutzt. Alle Algorithmen werden in einer Anwendung aufbauend auf dem Roboter-Framework FlexRF von dem Lehrstuhl Robotik und Eingebettete Systeme der Fakultät Informatik implementiert. Dieses Framework bietet verschiedene Schnittstellen für vielfältige

Hardware- sowie Software-Komponente. Aufgrund der modularen Struktur kann die Anwendung basierend auf dem Framework FlexRF schnell aufgebaut werden. In diesem Framework wurde eine komplette Bewegungsplanung für den Roboter implementiert, deshalb können wir uns in dieser Arbeit auf die Aktionsplanung konzentrieren. Das fünfte Kapitel wird alle Hardware- und Softwarekomponente, die in dieser Arbeit verwendet werden, ausführlich vorstellen.
Nachdem die Struktur des Objekts analysiert wurde, kann ein Aktionsplan für den Roboter erstellt werden. Wenn die Struktur einen auflösbaren Knoten enthält, wird dieser Knoten zuerst aufgelöst. Danach folgt es die Übergabe- und Klebe-Aktionen, mit denen zwei Endpunkte des Objekts richtig zusammengeklebt werden. Die detaillierten Schritte der Erstellung von Manipulationsplan werden ausschließlich im Kapitel fünf präsentiert.
Schließlich gibt das letzte Kapitel eine Zusammenfassung sowie die vorgeschlagenen zukünftigen Arbeiten.

Kapitel 2

DLO-Verfolgung

Objektverfolgung ist eine wichtige Aufgabe der Bildverarbeitung. Die Verbreitung von Hochleistungsrechnern sowie hochwertigen und kostengünstigen Videokameras, und die zunehmende Notwendigkeit für die automatische Videoanalyse erweckt großes Interesse von Objektverfolgungsalgorithmen. Es gibt drei wichtige Schritte in der Videoanalyse: Detektion von bewegten Objekten, Verfolgung solcher Objekte von Bild zu Bild und Analyse der Objektmerkmale, um sein Verhalten zu erkennen. Zahlreiche Ansätze zur Objektverfolgung wurden vorgeschlagen. Diese unterscheiden sich voneinander in erster Linie in der Art und Weise, wie sie die folgenden Fragen beantworten: Welche Objektdarstellung ist für die Verfolgung geeignet? Welche Bildmerkmale sollen verwendet werden? Wie sollen die Bewegung, Erscheinung und Form des Objekts modelliert werden? Die Antworten von diesen Fragen hängen von dem Kontext, in dem die Verfolgung ausgeführt wird, und der Verwendung von den aus der Verfolgung gewonnenen Informationen, ab. Eine große Anzahl von Verfolgungsmethoden sind vorgeschlagen worden, um diese Fragen für eine Vielzahl von Szenarien zu beantworten. Im nächsten Abschnitt wird eine Literatur-Übersicht präsentiert. Schließlich werden die gewählte Methode „Level Set“ und ihre echtzeitfähige Implementierung ausführlich erklärt.

2.1 Objektverfolgung

In dieser Sektion werden die Grundlagen der Objektverfolgung zuerst vorgestellt. Die gängigen Methoden zur Objektverfolgung werden präsentiert. Schließlich wird die Silhouettenverfolgungsmethode, die für unser Objekt geeignet ist, ausführlich erklärt.

2.1.1 Literatur-Übersicht

In einem Verfolgungsszenario kann ein Objekt als alles, was von Interesse für die weitere Analyse ist, definiert werden. Zum Beispiel, Boote auf dem Meer, Fische in

einem Aquarium, Fahrzeuge auf einer Straße, Flugzeuge in der Luft oder die Menschen auf einer Straße sind eine Reihe von Objekten, die wichtig zur Verfolgung sein können. Objekte können durch ihre Formen und Erscheinung beschrieben werden. Die häufig benutzten Darstellungsarten sind:

- **Punkte**: Das Objekt wird durch einen Punkt, z. B. den Schwerpunkt oder durch eine Reihe von Punkten repräsentiert. Im Allgemeinen ist die Punktdarstellung für die Verfolgung von Objekten geeignet, die nur kleinen Bereich in einem Bild besetzen.

- **Geometrische Formen**: Objektform wird durch ein Rechteck, Ellipse usw. dargestellt. Objektbewegung für solche Darstellungen wird in der Regel durch Translationen, affine oder projektive Transformationen modelliert. Obwohl geometrische Formen besser geeignet für die Darstellung von einfachen starren Objekten sind, werden sie auch für die Verfolgung von nicht-formfesten Objekten verwendet.

- **Objekt-Silhouette und Kontur**: Konturdarstellung definiert die Grenze eines Objektes. Die Region innerhalb der Kontur wird als die Silhouette des Objekts gekennzeichnet. Silhouette und Konturdarstellungen sind für die Verfolgung von komplexen deformierbaren Objekten geeignet.

- **Artikulierte-Modelle**: Artikulierte Objekte bestehen aus den Teilen, die durch Gelenke zusammengehalten werden. Zum Beispiel, der menschliche Körper ist ein gegliedertes Objekt mit Torso, Beinen, Händen, Kopf und Füßen, die durch Gelenke verbunden sind. Die Beziehung zwischen den Teilen wird durch kinematische Bewegungsmodelle geregelt, zum Beispiel, Gelenkwinkel, usw. Um ein artikuliertes Objekt darzustellen, kann man zylinder- oder ellipsenförmige Modelle benutzen.

- **Skelett-Modelle**: Objekt-Skelette können durch die Anwendung mediale Achsen-Transformation extrahiert werden. Dieses Modell wird oft als eine Form-Darstellung für die Erkennung von Objekten verwendet. Skelett-Darstellung kann verwendet werden, um sowohl artikulierte und starre Objekte zu modellieren.

Es gibt eine Reihe von Möglichkeiten, um das Erscheinungseigenschaften von Objekten darzustellen. Einige gängige Erscheinungsdarstellungen im Rahmen der Objektverfolgung sind:

- **Wahrscheinlichkeitsdichten der Objekterscheinung**: Die Schätzungen der Wahrscheinlichkeitsdichte der Objekterscheinung können entweder parametrisch, wie Gaußfunktion oder eine Mischung der Gaußfunktionen, oder nicht-parametrische wie Parzen-Fenster ([Smo02]) und Histogramme sein. Die Wahrscheinlichkeitsdichte von Merkmalen der Objekterscheinung (Farbe, Texturen) kann aus den Regionen des Bildes, die durch die Form-Modelle (Innenbereich einer Ellipse oder einer Kontur) spezifiziert werden, berechnet werden.

- **Templates**: werden aus einfachen geometrischen Formen oder Silhouetten gebildet. Ein Vorteil von Templates ist, dass sie sowohl räumliche und erscheinliche Informationen tragen. Templates beschreiben aber nur eine einzige Ansicht der Objekterscheinung. Deshalb sind Templates nur für die Verfolgung der Objekte, deren Posen (Positionen und Orientierungen der Objekte in 3D-Raum) nicht erheblich im Laufe der Verfolgung variieren, geeignet.
- **Multiview-Erscheinung-Modelle**: Diese Modelle können verschiedene Ansichten eines Objekts darstellen.

In der Regel gibt es eine starke Verbindung zwischen den Objektdarstellungen und Verfolgungsalgorithmen. Objektdarstellungen werden deshalb je nach Anwendungsdomäne gewählt. Jede Verfolgungsmethode erfordert einen Objekterkennungsmech-

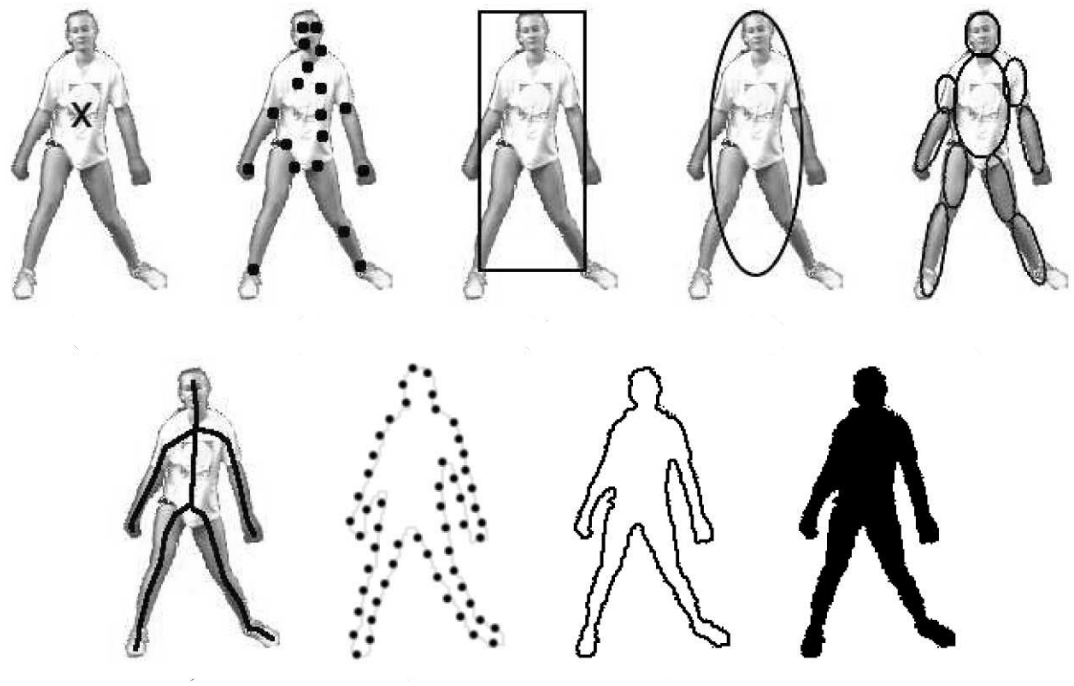


Abbildung 2.1: Beispiele der verschiedenen Erscheinungsmodelle [Sha06]

anismus entweder in jedem Frame oder wenn das Objekt zum ersten Mal im Video erscheint. Ein üblicher Ansatz zur Objekterkennung ist die Nutzung der Informationen nur in einem einzigen Frame. Andere Objekterkennungsmethoden benutzen aber Informationen aus einer Folge von Bildern, um die Falsch-Erkennungsrate zu reduzieren. Zeitliche Information kann in der Regel aus Differenzbildern gewonnen werden, die veränderten Regionen in aufeinanderfolgenden Frame hervorheben. Nachdem die Objektregion erkannt wurde, hat der Verfolgungsalgorithmus die Aufgabe, das Korrespondenz-Objekt von einem Bild zum Nächsten zu verfolgen.

Dieser Verfolgung erfolgt nur durch charakteristische Merkmale des Objekts, die sich nur leicht von Frame zum Frame ändern dürfen. Die Auswahl der richtigen Merkmale spielt deshalb eine entscheidende Rolle bei der Verfolgung. Im Allgemeinen ist die wichtigste Eigenschaft eines visuellen Merkmals seine Einzigartigkeit, sodass sich die Objekte einfach im Merkmals-Raum erkennen lassen. Diese Auswahl der Merkmale steht eng mit der Objektrepräsentation in Zusammenhang. Zum Beispiel sind Farben ein Merkmal für die Histogramm-basierten Repräsentation, während in der Regel die Objektgrenze für Kontur-basierte Repräsentation verwendet wird. In vielen Verfolgungsalgorithmen werden diese Merkmale sogar in Kombination verwendet. Die Einzelheiten der üblichen visuellen Merkmale sind wie folgt:

- **Farbe**: Die sichtbare Farbe eines Objektes ist in erster Linie durch zwei physikalische Faktoren beeinflusst: 1) die spektrale Energieverteilung der Lichtquelle und 2) die Reflexionseigenschaften der Oberfläche des Objekts. In der Bildverarbeitung wird RGB-Farbraum (Rot, Grün, Blau) oft benutzt, um Farben darzustellen. Allerdings ist der RGB-Farbraum kein wahrnehmungsmäßig einheitlicher Farbraum, das heißt, die Unterschiede zwischen den Farben im RGB-Farbraum entsprechen nicht der Farbenunterschiede der menschlichen Wahrnehmung [Pas01]. Außerdem sind die RGB-Farbraum stark korreliert. Im Gegensatz dazu sind L*U*V* und L*A*B* wahrnehmungsmäßig einheitliche Farbräume, während HSV-Farbraum (Farbton, Sättigung, Wert) ein etwa einheitlicher Farbraum ist. Allerdings sind diese Farbräume rauschempfindlich [Pet96]. Zusammenfassend kann man nicht allgemein feststellen, welcher Farbraum effizienter ist. Deshalb muss der Farbraum auch je nach Anwendung richtig gewählt werden.

- **Kanten**: Die Grenzen eines Objekts erzeugen normalerweise starken Veränderungen in Bild-Intensitäten. Kantenerkennung wird deshalb verwendet, um diese Veränderungen zu identifizieren. Eine wichtige Eigenschaft der Kanten ist, dass sie weniger empfindlich auf Änderungen der Illumination im Vergleich zu Farben. Algorithmen, die die Grenze des Objekts verfolgen, verwenden oft Kanten als die repräsentative Eigenschaft des Objekts. Aufgrund seiner Einfachheit und Genauigkeit ist Canny-Filter die beliebteste Kantendetektion-Methode [Can86].

- **Optischer Fluss**: Optischer Fluss ist ein Feld von Verschiebungsvektoren, die die Translation der einzelnen Pixel in einer Region definieren. Es wird mit der Einschränkung der Helligkeit berechnet, die die Konstanz der Helligkeit von einem Pixel in aufeinanderfolgenden Frame annimmt [Sch81]. Optischer Fluss wird oft in bewegungsbasierter Segmentierung- und Verfolgungsanwendungen verwendet.

- **Texturen**: Die Textur ist ein Maß für die Variation der Intensität von einer Fläche, die Eigenschaften wie Glätte und Gleichmäßigkeit quantifiziert.

Im Vergleich zur Farbe erfordert Textur einen Verarbeitungsschritt, um den Deskriptor zu erzeugen. Ähnlich wie bei Kante-Eigenschaft, ist die Textureigenschaft weniger empfindlich auf Änderungen von Beleuchtung im Vergleich zur Farbe.

Das Ziel des Verfolgers ist, die Trajektorie des Objekts im Laufe der Zeit zu erzeugen, indem die Position des Objekts in jedem Frame des Videos bestimmt wird. Der Objektverfolger kann auch die gesamte Region angeben, die durch das Objekt zu jeder Zeit besetzt ist. Die Aufgaben der Detektion und Verfolgung des Objekts in Frame können entweder separat oder gemeinsam durchgeführt werden. Im ersten Fall werden die möglichen Regionen des Objekts in jedem Frame mithilfe eines Objekterkennungsalgorithmus erhalten und dann sucht der Verfolger nach entsprechenden Objekten Frame nach Frame. Im letzteren Fall werden die Detektion und Verfolgung gemeinsam durch iterative Aktualisierung der Objektposition und Regionsinformationen aus vorhergehenden Frames geschätzt. In beiden Verfolgungskonzepten werden die Objekte durch die Form- und / oder Erscheinungsmodelle dargestellt. Die ausgewählten Modelle beschränken die Art der Bewegung oder Verformung von dem Objekt. Zum Beispiel, wenn ein Objekt durch einen Punkt modelliert wird, dann kann nur ein translatorisches Modell für die Bewegung verwendet werden. In den Fällen, in denen eine geometrische Formdarstellung wie eine Ellipse für das Objekt verwendet wird, sind parametrische Bewegungsmodelle wie affine oder projektive Transformationen geeignet. Diese Modelle können die Bewegung starrer Objekte approximieren. Für ein flexibles Objekt kann Silhouette oder Kontur als die Erscheinungsmodelle und beide parametrischen und nicht-parametrischen Modelle als die Bewegungsmodelle verwendet werden. In Anbetracht der oben erwähnten Diskussion hat Yilmaz et al. [Sha06] eine Taxonomie von Verfolgungsmethoden vorgeschlagen. Die Verfolgungsalgorithmen werden dann in drei Kategorien eingeordnet:

- **Punktverfolung**: Die erkannten Objekte in aufeinanderfolgenden Frames sind durch Punkte dargestellt, und die Assoziation der Punkte basiert auf dem vorherigen Objektzustand z. B. die Position und Bewegung. Dieser Ansatz erfordert einen externen Mechanismus, um die Objekte in jedem Bild zu erkennen.

- **Kernelverfolgung**: Kernel bezieht sich auf die Form und Erscheinung des Objekts. Zum Beispiel kann der Kernel eine reckeckige oder eine elliptische Form mit einem zugehörigen Histogram. Die Objekte werden durch die Berechnung der Bewegung der Kernel in aufeinanderfolgenden Bildern verfolgt. Diese Bewegung ist in der Regel in Form parametrischer Transformationen wie Translation und Rotation.

- **Silhouettenverfolgung**: Verfolgung wird durch Schätzung der Objektregion in jedem Frame durchgeführt. Silhouettenverfolgungsmethoden verwenden die Informationen innerhalb der Objektregion. Silhouetten oder Konturen können durch Kontur-Form-Matching oder Konturevolution verfolgt werden. Bei-

de Methoden lassen sich im Wesentlichen als Objektsegmentierung durch die Benutzung der aus den vorhergehenden Frames gewonnenen Informationen kennzeichnen.

Im nächsten Abschnitt wird die von uns verwendete Methode Silhouettenverfolgung ausführlicher erklärt.

2.1.2 Silhouettenverfolgung

Nicht-formfeste Objekte können komplexe Formen haben, zum Beispiel, Hände, Kopf und Schultern (siehe Abbildung 2.2), die nicht durch einfache geometrische Formen beschrieben werden können. Silhouettenbasierte Methoden liefern eine genauere Formbeschreibung für diese Objekte. Das Ziel eines Silhouettenverfolgers ist das Objekt in jedem Frame durch ein Objektmodell aus den vorangegangenen Frames zu finden. Dieses Modell kann in Form eines Farbhistogramms oder Objektkontur sein. Wir teilen die Silhouettenverfolger in zwei Kategorien auf, nämlich Form-Matching und Konturverfolgung. Form-Matching-Ansätze suchen nach der Objektsilhouette im aktuellen Frame. Auf der anderen Seite entwickelt Konturverfolger eine erste Kontur zu seiner neuen Position im aktuellen Frame, indem man entweder die Zustandsraummodelle benutzt oder einige Energiefunktionen minimiert.

- **Form-Matching**: Form-Matching kann ähnlich wie Template-Matching durchgeführt werden, indem eine Objekt-Silhouette und das damit verbundene Modell im aktuellen Frame gesucht werden. Die Suche wird durch die Berechnung der Ähnlichkeit zwischen dem Objekt und dem Modell, das aus vermuteten Silhouetten aus den früheren Frames generiert wurde, durchgeführt. Bei diesem Ansatz wird es davon ausgegangen, dass sich die Silhouette nur aus dem aktuellen Frame zum nächsten translatorisch bewegt. Die Bewegung des nicht-starren Objekts wird nicht explizit behandelt. Das Objektmodell, das normalerweise in Form einer Kantenkarte ist, wird neu initialisiert, um Veränderungen im Erscheinungsbild des Objekts in jedem Frame zu behandeln, nachdem das Objekt lokalisiert wurde. Diese Aktualisierung ist erforderlich, um Probleme bezüglich der Veränderung von Aussichtspunkt und Beleuchtung sowie der Bewegung des nicht-formfesten Objekts zu überwinden.

- **Konturverfolgung**: Im Gegensatz zu Form-Matching-Methoden entwickeln Konturverfolgungsmethoden iterativ eine erste Kontur in den vorhergehenden Frames auf die neue Position im aktuellen Frame. Diese Konturentwicklung erfordert, dass ein Teil des Objekts im aktuellen Frame mit der Objektregion im vorherigen Frame überlappt. Es gibt zwei verschiedene Ansätze für die Verfolgung durch Konturevolution. Der erste Ansatz nutzt Zustandsraummodelle, um die Form und Bewegung der Kontur zu modellieren. Der zweite Ansatz entwickelt die Kontur durch die Minimierung der Kontur-Energie-Funktion.

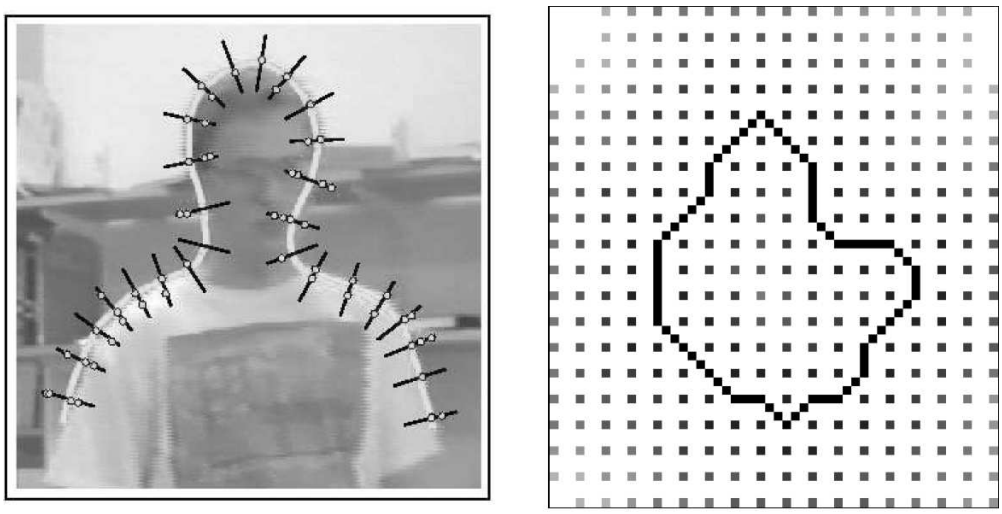


Abbildung 2.2: Beispiele der Kontur- und Silhouettenverfolgung [Sha06]

Silhouettenverfolgung wird eingesetzt, wenn die gesamte Region eines Objekts verfolgt werden soll. Wichtige Faktoren, um die verschiedenen Silhouettenverfolgungsalgirthmus zu unterscheiden, sind: Welche Funktionen werden verwendet? Wie wird die Okklusion behandelt? Ist das Training erforderlich? Darüber hinaus verwenden einige Algorithmen nur Informationen über die Silhouettengrenze für die Verfolgung, während die anderen die gesamte Region in der Silhouette benutzen. Generell sind die Region-basierten Ansätze widerstandsfähiger gegen Rauschen.
Der wichtigste Vorteil der Silhouettenverfolgung ist ihre Flexibilität, um eine große Vielfalt von Objekt-Formen zu behandeln. Die häufig benutzte Darstellungsart von Silhouetten ist in Form einer binären Indikatorfunktion, die die Objektregion durch Einsen und die Umgebung durch Nullen markieren. Für konturbasierte Methoden wird die Silhouette entweder explizit oder implizit dargestellt (siehe Abbildung 2.2). Explizite Darstellung definiert die Grenze der Silhouette durch eine Reihe von Kontrollpunkten. Implizite Darstellung definiert die Silhouette durch eine Funktion auf einem Raster. Die häufig benutzte implizite Konturdarstellung ist die Level-Set-Darstellung. Die Repräsentation von der silhouettenbasierten Verfolgungsmethode kann in Form von Bewegungsmodellen, Erscheinungsmodellen oder einer Kombination von diesen sein. Objektform kann durch eine Wahrscheinlichkeitsdichte für Kontur modelliert werden, die aus einer Reihe von möglichen Objektzuständen erzeugt wird [Isa98]. Darüber hinaus kann Objektform implizit durch eine Level-Set-Funktion, deren Zero-Level-Set die Kontur eines Objekts beschreibt [Sha04].
Okklusionsbehandlung ist ein weiterer wichtiger Aspekt der Silhouettenverfolgungsmethoden. Die meisten Methoden behandeln das Okklusionsproblem nicht explizit. Ein üblicher Ansatz ist es, eine konstante Bewegung oder Beschleunigung des Objekts anzunehmen, und wenn Okklusion auftritt, wird die Objektsilhouette aus den

vorhergehenden Frames auf seine neue hypothetische Position transformiert. Nur wenige Methoden behandeln die Okklusion explizit durch die Erzwingung der Formzwangsbedingung ([Bla00]; [Sha04]). Ein weiteres wichtiges Thema im Zusammenhang mit Silhouettenverfolgung ist die Fähigkeit, die Spaltung oder Fusionierung des Objekts zu behandeln. Ein Beispiel ist die Verfolgung der Silhouette einer Person, die ein Objekt trägt. Wenn die Person das Objekt hinterlässt, wird ein Teil der Kontur der Person auf dem gelassenen Objekt platziert. Diese Topologieänderungen der Region können durch implizite Konturdarstellungen der Level-Set-Methode sehr gut behandelt werden.

2.2 Motionsegmentierung

Wie in der vorherigen Sektion schon erwähnt, ist die Extraktion von bewegenden Objekten aus einer Videosequenz oft der erste Schritt im visuellen Verfolgungssystem. Ein üblicher Ansatz zur Extraktion von bewegenden Objekten aus dem Hintergrund ist Hintergrundsubtraktion. Die Idee von Hintergrundsubtraktion ist, das aktuelle Bild zu einem Referenz-Hintergrund-Modell zu differenzieren. Die Subtraktion identifiziert nicht-stationäre oder neue Objekte. Das einfachste Hintergrundmodell geht davon aus, dass die Intensität eines Pixels von einer einzigen unimodalen Verteilung modelliert werden kann. Dieses Grundmodell wurde in ([Wre96],[Dav99]) verwendet. Allerdings kann ein Single-Mode-Modell nicht mehrere Hintergründe, wie wehenden Bäumen modellieren. Das verallgemeinerte Mischverteilungsmodell (Mixture of Gaussians - MOG), das in [Gri99] verwendet wurde, wird zur Modellierung komplexer, nicht-statischer Hintergründe verwendet. MOG hat allerdings einige Nachteile. Schnell variierte Hintergründe sind nicht einfach mit wenigen Mittelwerten genau zu modellieren. Darüber hinaus muss MOG abhängig von der Lernrate einen Kompromiss finden, um Veränderungen der Hintergrundinformationen anzupassen. Für eine geringe Lernrate erzeugt es ein Modell, das schwierig plötzliche Änderungen des Hintergrunds erkennt. Wenn sich das Modell zu schnell anpasst, werden langsame bewegende Vordergrundpixel in das Hintergrundmodell absorbiert, was zu einer hohen falsch negativen Rate führt. Dies wurde als der Vordergrund-Blende-Problem in [Mey99] beschrieben.
Um diese Probleme zu überwinden, wurden nicht-parametrische Verfahren vorgeschlagen. Sie sind in der Lage, sich sehr schnell an Veränderungen der Hintergrund anzupassen und Ziele mit hoher Sensitivität zu erkennen.
In dieser Arbeit benutzen wir auch ein nicht-parametrisches Verfahren, das Codebuch genannt wird. Das ist ein regionbasiertes Verfahren. D. h., die Pixel werden nicht getrennt, sondern in „Blob“ betrachtet. Das Hintergrundmodell, das durch Codebuch erstellt wurde, ist komprimiert. Die einfachste Methode um dieses Modell zu erstellen ist der Vergleich eines neuen Bildwertes für ein Pixel mit dem vorher betrachteten Wert. Wenn der neue Wert in der Nähe vom alten Wert liegt, wird es als eine Perturbation von der Farbe modelliert. Wenn es nicht der Fall ist, wird eine neue

Gruppe von Farben entsprechend dem neuen Wert erstellt. Man bekommt schließlich als Ergebnis eine Gruppe von Blobs in einem Farbraum. Jeder Blob repräsentiert ein separates Hintergrundvolumen.

2.2.1 Codebuch-Methode

Der Hintergrundsubtraktionsalgorithmus Codebuch lernt den Hintergrund über längere Zeit, ohne ein parametrisches Modell zu erstellen. Ein komplexer Hintergrund kann von mehreren Codewörtern modelliert werden. Die wichtigsten Merkmale des Algorithmus sind:

- ein adaptives und kompaktes Modell, das einen sich bewegenden strukturellen Hintergrund über einen längeren Zeitraum unter begrenzten Speicher aufzeichnen kann.
- die Handlungsfähigkeit der lokalen und globalen Veränderungen der Beleuchtung;
- geschichtete Modellierung und Erkennung ermöglichen, mehrere Schichten von Hintergrundinformationen, die verschiedene Hintergründe repräsentieren, zu erstellen.

Die in OpenCV implementierte Methode wird aus der Methode von Kim, Chalidabhongse, Harwood und Davis [Dav05] abgeleitet. Dies benutzt den RGB-Farbraum nicht, sondern den YUV-Farbraum. Schnelles Verfahren zur Verfeinerung des daraus resultierenden Hintergrundbildes kann in Martins [Haz99] gefunden werden.
Ein Blob wird einfach durch eine Box im YUV-Farbraum modelliert. Dieses Modell ist besonders effektiv bezüglich Speicherung und Rechenaufwand sowie der Überprüfung, ob sich ein Pixel innerhalb von einem Blob befindet. Die Codebuch-Methode wird durch ein einfaches Beispiel erklärt. Ein Codebuch wird von mehreren Boxen zusammengestellt, die im Laufe der Zeit wachsen, um die gemeinsam gehörigen Werte zu decken. Der obere Teil der Abbildung 2.3 zeigt eine Wellenform über die Zeit. Im unteren Teil werden die Boxen erstellt, um einen neuen Wert zu decken und dann langsam wachsen, um Werte in der Nähe zu decken. Wenn ein Wert zu weit weg ist, dann wird eine neue Box erstellt, um die neuen Werte und die dazu nahliegenden Werte zu decken.
Im Falle unseres Hintergrundmodells deckt eine Box die Werte in den drei Dimensionen (die drei Farbkanäle von unserem Bild an jedem Pixel) zu decken. Codebuch-Methode kann mit Pixeln umgehen, die sich im Laufe der Zeit dramatisch verändern (z. B. wegen Änderung von Illumination).
In Codebuch-Methode wird jede Box durch zwei Schwellenwerte (max und min) auf jede der drei Farbe-Achsen definiert. Diese Schwellenwerte werden erweitert (max wird größer, min wird kleiner), wenn sich neue Hintergrund-Pixel innerhalb eines Lernschwellenwertes ($learnHigh$ und $learnLow$: $learnHigh > max$ und $learnLow < min$) aber außerhalb von der Box befinden. Im anderen Fall wird eine

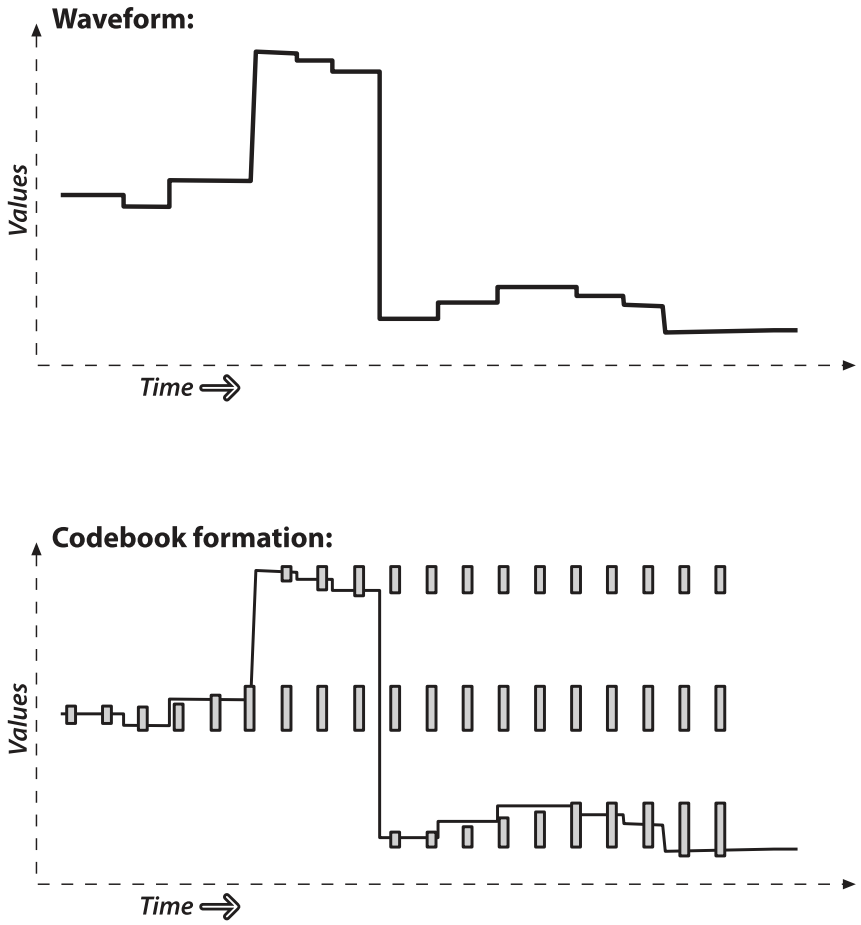
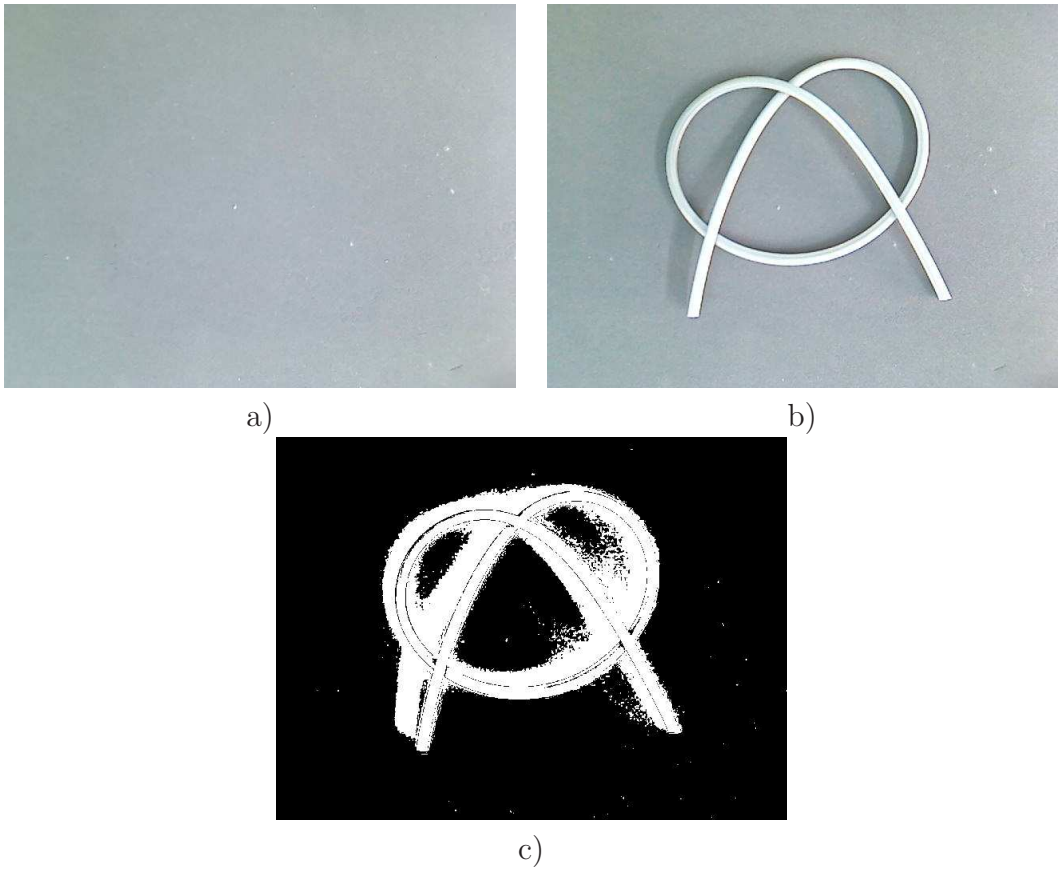


Abbildung 2.3: Codewörter sind Boxen, die einen bestimmten Bereich der Farbwerte enthalten. Eine box wird erstellt, um einen neuen Farbwert abzudecken. Die Box vergrößert sich langsam, um die nah liegenden Werte auch abzudecken. Wenn der Wert zu weit von der Box ist, wird eine neue Box erstellt. [Kae08]

neue Box erstellt. Im Hintergrundsubtraktionsteil gibt es Akzeptanz-Schwellenwerte $maxMod$ und $minMod$. Mit diesen Schwellenwerten können wir feststellen, ob ein Pixel „nah genug“ zu einer Box ist. Die genaue Implementierung kann man mehr im Buch „Learning OpenCV“ [Kae08] lesen. Die Abbildung 2.4 zeigt Ergebnis von dem Codebuch-Algorithmus, um unser Objekt zu segmentieren, wenn sich es im Arbeitbereich des Roboters befindet:



a) b)

c)

Abbildung 2.4: Ergebnis vom Codebuch-Algorithmus: a) Das Hintergrundmodell. b) Das aufgenommene Bild wenn das Objekt im Arbeitbereich des Roboters ist. c) Das segmentierte Bild aus der Codebuch-Methode

2.2.2 Hautfarbe, Schatten

Aufgrund der Beleuchtung enthält das Bild nach der Hintergrundsubtraktion oft auch Schatten. Um die originale Form des Objekts zu bekommen, müssen wir diese Schattenpixel eliminieren. In [Ebr04] wird eine einfache Methode zur Schattensegmentierung vorgeschlagen. Diese Methode benutzt den RGB-Farbraum und den normalisierten RGB-Farbraum.

Im ersten Schritt wird ein Referenzmodell M für den Hintergrund gemacht. Es soll hier beachtet werden, dass das Referenzmodell M nicht relevant zum Hintergrundmodell von der Codebuch-Methode ist. Das Referenzmodell wird oft von den ersten n Frames der Videosequenz berechnet, das heißt:

$$M = \sum_{i=1}^{n} w_i F_i, \tag{2.1}$$

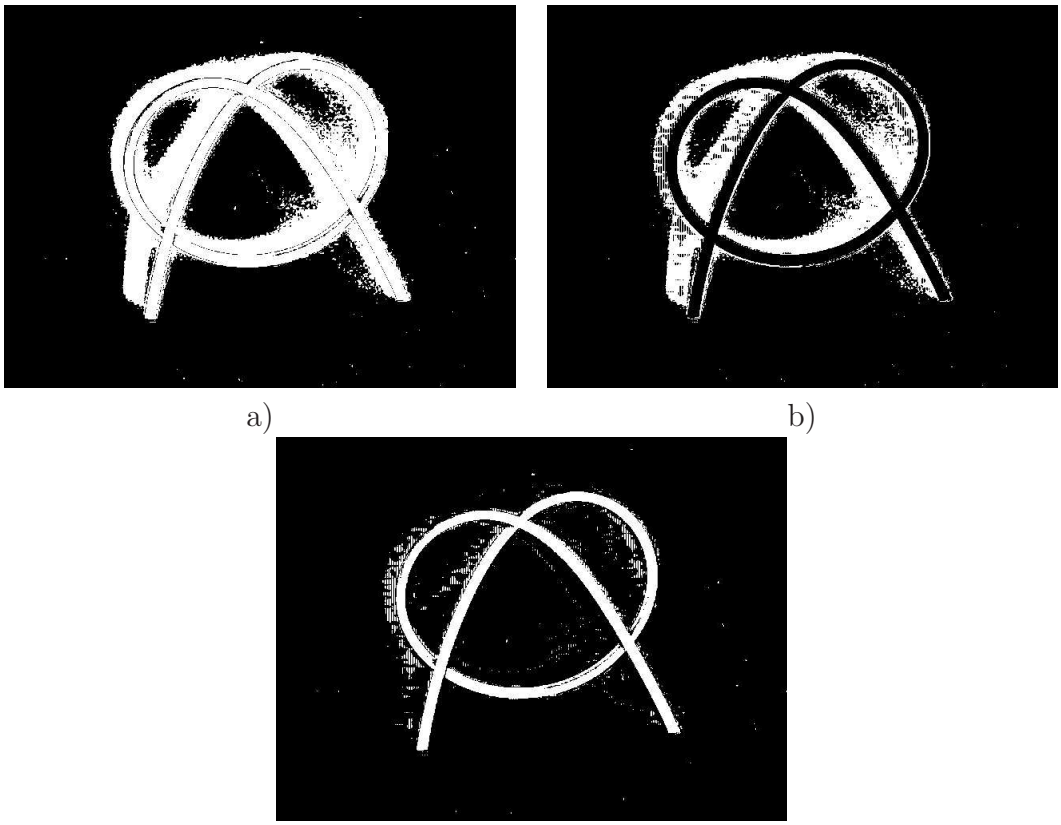
wobei F_i das i-te Frame ist und w_t ist ein Gewichtungsfaktor und wird oft als $w_i = \frac{1}{n}$ gewählt. Die Transformation von RGB-Farbraum zum normalisierten RGB-Farbraum wird wie folgt durchgeführt:

$$r = \frac{R}{R+G+B} \quad , \quad g = \frac{G}{R+G+B} \quad , \quad \frac{B}{R+G+B}, \tag{2.2}$$

wobei R, G, B die absoluten Werte in RGB-Farbraum und r, g, b die normalisierten Werte sind. c_1 und c_2 werden entsprechend als die größte Komponente und die mittlere Komponente in RGB-Farbraum definiert. Ähnlich werden c_3 und c_4 entsprechend als die größte Komponente und die mittlere Komponente in rgb-Farbraum definiert.
Ein Pixel (x, y) von dem Frame F_t wird dann als Schattenpixel erkannt, wenn die beiden folgenden Bedingungen erfüllt werden.

- **Bedingung 1**: $c_1(F_t(x,y)) < c_1(M(x,y))$ und $c_2(F_t(x,y)) < c_2(M(x,y))$,
- **Bedingung 2**: $|c_3(F_t(x,y)) - c_3(M(x,y)| < k_1$ und $|c_4(F_t(x,y)) - c_4(M(x,y)| < k_2$,

wobei k_1 und k_2 zwei kleine Konstanten sind und werden oft experimentell bestimmt. Die Abbildung 2.5 zeigt Ergebnis von der Schattensegmentierung:
In unserer Anwendung wird das Objekt in den Arbeitsbereich gelegt, nachdem das Modell für den Hintergrund erstellt wurde. Es ist erwünscht, dass das Objekt auch in dieser Phase verfolgt wird. Da die Hände des Menschen zusammen mit dem Objekt als Vordergrund segmentiert werden, müssen wir die Hände des Menschen aus dem Vordergrundbild löschen. Eine mögliche Methode dafür ist Haupfarbe-Segmentierung, d. h., die Pixel, die ähnliche Farbe wie Hautfarbe haben, werden aus dem Vordergrundbild gelöscht. Das verbleibende Vordergrundbild enthält ausschließlich nur noch die Pixel von dem Objekt. In [Sch06] sind einige Hautfarbe-Segmentierung-Algorithmen präsentiert. Die Methoden segmentieren die Hautfarben durch einen stückweisen linearen Schwellenwert. Diese expliziten Hautfarbe-Segmentierung-Methoden schlagen verschiedene feste Schwellenwerte in einem bestimmten Farbraum vor. Einige Farbräume erlauben die Suche nach hautfarbigen Pixeln im 2D-chromatischen Raum, mit dem die Empfindlichkeit von Beleuchtungsänderung verringert werden kann. Sechs verschiedene Algorithmen, die in verschiedenen Farbräumen arbeiten, wurden implementiert und analysiert. Sie sind nach den



a) b)

c)

Abbildung 2.5: Schatten-Subtraktion

Farbraum-Namen benannt: YCbCr, RGB, HSV1, HSV2, HSI und rgb. Die Einzelheiten ihrer Implementierungen können in [Sch06] gelesen werden. Nach ihrer Analyse haben die Methoden HSV1 und rgb die beste Präzision. In dieser Arbeit wurde die HSI-Methode implementiert.
Zuerst wird das RGB-Bild zum HSI-Bild mit folgenden Gleichungen transformiert:

$$\begin{aligned} I_1 &= \frac{1}{3}(R+G+B)\ ;\ I_2 = \frac{1}{2}(R-B)\ ;\ I_3 = \frac{1}{4}(2G-R-B), \\ I &= I_1, \qquad (2.3) \\ S &= \sqrt{I_2^2 + I_3^2}, \qquad (2.4) \\ H &= tan^{-1}(\frac{I_3}{I_2}), \qquad (2.5) \end{aligned}$$

Danach werden die folgenden Bedingungen geprüft. Die Pixel, deren Farbenwerte die Bedingungen erfüllen, werden als Hautfarbe-Pixel markiert:

- **Bedingung 1**: $I > 40$.

- **Bedingung 2**: wenn $13 < S < 110$, $0° < H < 28°$ und $332° < H < 360°$; wenn $13 < S < 75$, $309° < H < 331°$.

2.2.3 Verbundene Komponenten

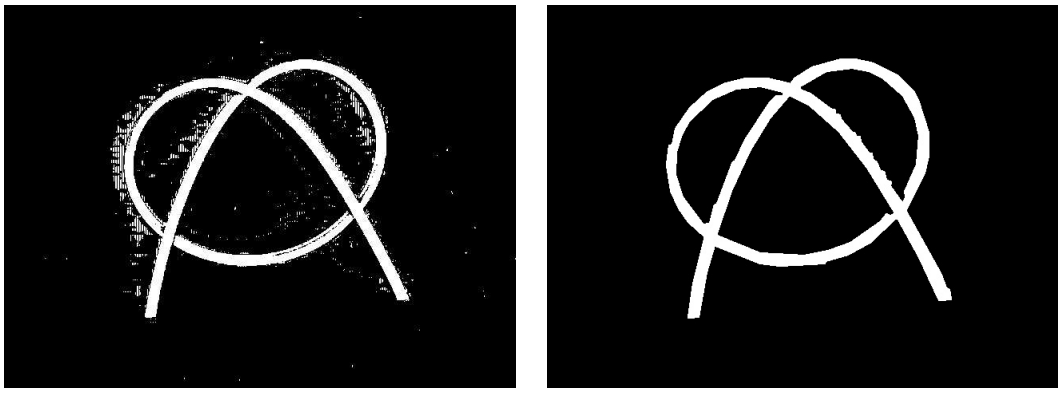
Das Ergebnis der Codebuch-Methode sollte eigentlich nur das sich bewegende Objekt sein. In der Praxis enthält dieses Ergebnis aber auch die Pixel, die eigentlich zum Hintergrund gehören. Wir müssen deshalb diese Pixel rausfiltern, bevor wir unser Objekt aus den Rohpixeln rekonstruktieren. In dieser Arbeit wird dieser Prozess von der Verbundene-Komponenten-Methode [Haz99] durchgeführt. Diese Methode nimmt als Eingang das Rausch-behaftete Bild. Die morphologische Operation „Öffnen“ wird dann durchgeführt, um die Bereiche von kleinen Rauschen zu schrumpfen und die morphologische Operation „Schließen“ wird anschließend durchgeführt, um die überlebenden Komponenten zu wiederherzustellen. Danach können wir die „genug großen“ Konturen der überlebenden Segmente finden. Jetzt können wir entweder alle übrigen Komponenten zurückgeben oder nur die Komponente, die größer als ein Schwellenwert sind, um die Rauschen weiter zu unterdrücken.
Der in dieser Arbeit implementierte Ansatz basiert auf der Implementierung von OpenCV und enthält mehrere Auswahlmöglichkeiten, um das wirkliche Objekt zu rekonstruieren. Details über die Implementierung von OpenCV können mehr in [Kae08] gelesen werden. Da die Implementierung von [Kae08] kein Objekt mit Löchern rekonstruieren kann, wird unser Ansatz erweitert, um dieses Problem zu überwinden. Die größte konvexe Hülle von dem Objekt wird zuerst mit dem Wert 255 überfüllt. Danach werden die Löcher mit dem Wert 0 gezeichnet. Schließlich haben wir als Ausgang von diesem Schritt ein binäres Bild, dessen Pixel den Wert 255 haben, wo sich das Objekt befindet und den Wert 0, wo der Hintergrund ist. Die Abbildung 2.6 zeigt das segmentierte Bild nach dem Verbundenen-Komponenten-Prozess.

2.3 Konturverfolgung

Diese Sektion erklärt ausführlich die Level-Set-Methode, die wir zur Verfolgung unseres Objekts benutzen. Die Level-Set-Methode hat mehrere Vorteile im Vergleich zu anderen Verfolgungstechniken. Zuerst werden die Grundlagen von Level-Set-Methode präsentiert. Danach wird die Level-Set-Funktion, die die Evolution der Kontur regelt, erklärt. Schließlich wird die Schnellimplementierung, die unsere Echtzeit-Anforderung erfüllt, vorgestellt.

2.3.1 Level-Set-Methode

Level-Set-Methoden sind implizite Formulierungen des aktiven Konturmodells, das ursprünglich von Kass et. al. [Ter88] vorgeschlagen wurde. Sie wurden von Osher und



a) b)

Abbildung 2.6: Ergebnis nach dem Verbundenen-Komponenten-Prozess

Sethian [Set88] für die Simulation der Flammenausbreitung und Strömungsmechanik eingeführt. Die inkorporierten Ausbreitungsschemen sind leicht für eine Vielzahl von Problemen zu integrieren. Das führt zu vielen Anwendungen, die diese Methoden benutzen, in vielen Forschungsbereichen. Beispiele gibt es in Computer-Grafik, Bewegungsverfolgung, Strömungssimulation, Segmentierung, Berechnung des kürzesten Wegs und Segmentierung.
Die Level-Set-Methode benutzt „Propagation der Front“ um eine bestimmte geschlossene Kontur innerhalb eines bestimmten Bereiches mit der Zeit zu entwickeln. Die Bewegung der Kontur wird von einer Geschwindigkeitsfunktion geregelt, die die Eigenschaften des simulierten Prozesses widerspiegelt. Mit dieser Geschwindigkeitsfunktion ist es möglich, die Geschwindigkeit jedes Pixels auf der Kontur in jedem beliebigen Zeitpunkt zu berechnen. Im Falle der Bildsegmentierung sind diese Eigenschaften in der Regel aus der Erscheinung des gewünschten Objekts im Bild abgeleitet und basieren sich daher oft auf den Bildeigenschaften wie Gradient oder Textur. Der Prozess ist initialisiert, indem eine geschlossene Kontur irgendwo im Bild wird. Oft soll diese Anfangsposition innerhalb des Objekts sein, das segmentiert werden sollte. Aber andere Arten von Initialisierung sind auch möglich.
Die aktive Kontur entwickelt sich gemäß seiner Geschwindigkeitsfunktion, die so definiert ist, dass die Front sich über das Bild mit einer hohen Geschwindigkeit bewegt, solange die Eigenschaften des Objekts erfüllt sind. Die Geschwindigkeit reduziert aber auf nahezu Null, wenn die Front an der Grenze des gewünschten Objekts kommt (z. B., die vordefinierten Eigenschaften sind nicht mehr wahr). Die geschlossene Kurve konvergiert sich daher langsam zu der Form des gewünschten Objekts.
Mathematisch werden die impliziten aktiven Konturen mit Hilfe von Techniken aus der Strömungsmechanik und hyperbolischer Erhaltungssätze modelliert. Betrachten wir eine geschlossene Kurve, die sich im Laufe der Zeit mit einer bestimmten Geschwindigkeit entlang seines normalen Vektorfelds bewegt. Dies führt zu einer

Menge von Kurven $\gamma(t)$ mit $t \in [0, \infty)$. Die Grundidee besteht nun darin, dass die Bewegung von γ durch die Annahme, dass γ in einer höher dimensionalen Funktion ϕ eingebettet ist, implizit zu verfolgen. Diese Funktion ϕ entwickelt sich durch eine partielle Differenzialgleichung, sodass $\phi(t) = 0$ die Front zum Zeitpunkt t ergibt. Ein einfaches Beispiel für diesen Prozess ist in Abbildung 2.7 veranschaulicht.
Um die obige Idee zu generalisieren, setzen wir Ω als eine beschränkte Teilmenge von

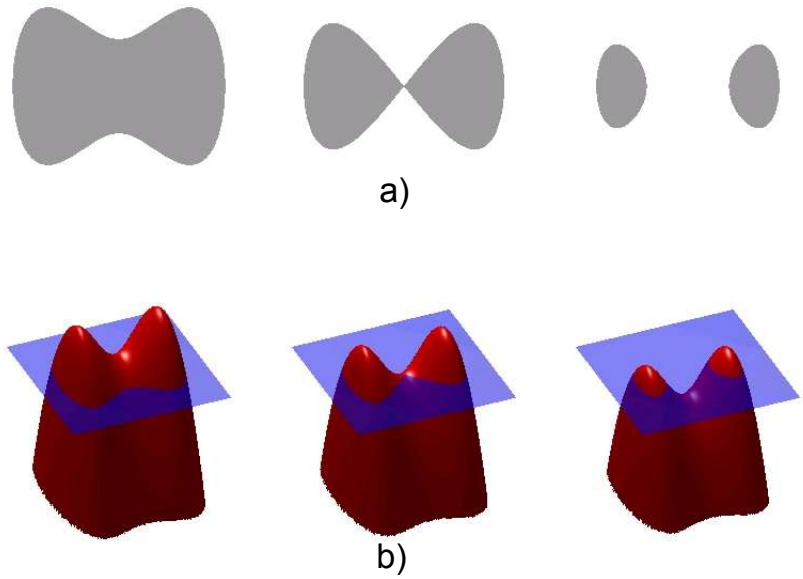


Abbildung 2.7: Konturverfolgung durch Evolution der Front. Änderung der Topologie wird implizit behandelt. a) Zero-Level-Set b) Energiefunktion und die Front

$\Re^N$ und $x(s) \in \Re^N, 0 \leq s \leq S$ als der Parametrierungsvektor einer geschlossenen Hyperfläche γ. Diese Hyperfläche wird nun als das Zero-Level-Set in einer $(N+1)$-dimensionalen Lipschitzfunktion $\phi(x,t) \in \Omega \times (0, \infty)$ eingebettet.

$$\phi(x, t = 0) = \pm d, \tag{2.6}$$

wobei $d = d(x, \gamma)$ der vorzeichenbehaftete Abstand von x zu der initialen Kurve $\gamma(t = 0)$ ist. Das Zeichen zeigt an, ob x innerhalb oder außerhalb von γ ist. Normalerweise ergibt $d < 0$ Punkte in der Innerseite der Kurve $\gamma(t = 0)$ an und $d > 0$ die Punkte in der Außenseite. Insbesondere gilt es auch:

$$\gamma(t = 0) = \{x | \phi(x, t = 0) = 0\}. \tag{2.7}$$

Natürlich ist es möglich, einen beliebigen Wert $\phi(x) = k, k \in \Re$ für das Zero-Level-Set zu definieren. Die Wahl $k = 0$ hat aber verschiedene Vorteile, z. B. die Bestimmung, ob ein Punkt x innerhalb oder außerhalb von γ ist, kann einfach durch

die Beobachtung von $sgn(d)$ ermöglicht werden sowie der Abstand von einem beliebigen Punkt zur Front kann auch ohne weitere Berechnungen bekommen werden. Da ϕ eine Charakterisierungsfunktion von der Entwicklung der gegebenen Kurve γ kennzeichnen soll, folgt es daraus, dass es für jeden auf einen Pfad x bewegenden Punkt gilt:

$$\phi(x(t), t) = 0. \tag{2.8}$$

Diese Bewegung von der Front kann durch die Ableitung von ϕ in Bezug auf t ausgedrückt werden. Da $x = x_1, ..., x_N$ folgt es aus 2.8:

$$\frac{\partial \phi}{\partial t} + \sum_{i=1}^{N} \frac{\partial \phi}{\partial x_i} \frac{\partial x_i}{\partial t} = 0. \tag{2.9}$$

Wie bereits erwähnt, bewegt sich jeder Punkt x auf der Kurve γ (z. B. das Zero-Level-Set) in Richtung der Flächennormalen $\overrightarrow{n}$ mit einer bestimmten Geschwindigkeit $F(x)$. Eine sehr einfache Geschwindigkeitsfunktion F ist also gegeben durch:

$$\overrightarrow{F} = \frac{\partial x}{\partial t} \cdot \overrightarrow{n} = \frac{\partial x}{\partial t} \frac{\nabla \phi}{|\nabla \phi|}. \tag{2.10}$$

Die Verwendung der euklidischen Distanzen ergibt sich $|\nabla \phi| = 1$.
Es wird später gezeigt, dass die Definition von F sehr komplex sein kann und diese Definition bestimmt auch die Art und Weise von der Entwicklung des impliziten Modells. Mit F können wir die Gleichung 2.9 zu der klassischen Level-Set-Gleichung vereinfachen:

$$\phi_t + F|\nabla \phi| = 0, \tag{2.11}$$

mit $\phi_t = \frac{\partial \phi}{\partial t}$ Es soll beachtet werden, dass solange die Geschwindigkeit der Front nur von der ersten Ableitung abhängt, die Gleichung 2.11 eine Hamilton-Jacobi-Differenzialgleichung ist. Gleichungen dieser Art haben bestimmte Eigenschaften, die für diskrete Berechnungen sorgfältig beachtet werden müssen.
Die obige Formulierung hat verschiedene Vorteile: Da dieser Ansatz die Topologie in der Level-Set-Funktion ϕ integriert, kann γ seine Topologie ändern. Außerdem ist es sehr einfach, geometrische Eigenschaften wie die Oberflächennormal oder Krümmung an jedem Punkt der Front zu bestimmen. Und schließlich sind alle erforderlichen Gleichungen von der Anzahl der Dimensionen unabhängig, sodass die Level-Set-Methoden in einer breiten Palette von Anwendungen benutzt werden können.
Nachteilig ist die Benutzung der Funktion ϕ rechenintensiv, weil eine weitere Dimension in die Berechnung hinzugefügt wird. Es ist möglich, diesen Effekt teilweise durch den Einsatz effizienter Schemen für die Berechnung von ϕ wie die Berechnung von einem Schmalband (Narrow-Band) auszugleichen. In diesem Zusammenhang sollten auch noch zwei andere Probleme angegangen werden: Kleine numerische Fehler des kontinuierlichen Ausbreitungsprozesses in der diskreten Domäne werden kumulativ addiert, bis zum Zeitpunkt, bei dem die Bewegung der Kontur γ der Level-Set-Funktion ϕ nicht mehr entspricht. Außerdem ist es schwierig, die Geschwindigkeit

der Front für künftige Schritte basierend auf lokalen Bild-Merkmalen vorherzusagen. Beide Probleme können durch Re-Initialisierung der Level-Set-Funktion ϕ nach einer gewissen Zeit vermieden werden. Dies ist genau die zentrale Idee des oben genannten Narrow-Band-Ansatzes. Die Anwendung dieser Technik spart nicht nur Rechenzeit, sondern auch sichergestellt, dass die Level-Set-Funktion zurückgesetzt wird, sodass sich γ richtig entwickelt.
In der Theorie bricht der Ausbreitungsprozess ab, wenn die aktive Kontur zu der Lösung, die durch die Eigenschaften der Level-Set-Funktion festgelegt wurde, konvergiert hat. Leider wird die Geschwindigkeit der Front in den meisten Fällen nie genau Null. Daher muss ein Konvergenz-Kriterium in den Prozess einbezogen werden, um zu spezifizieren, wann der Algorithmus gestoppt werden soll. In den meisten Ansätzen basiert diese Entscheidung auf dem Verhältnis zwischen der kumulativen Distanz, über die die Punkte auf der Front zurückgelegt haben, und der Anzahl der Pixel, die aktualisiert wurden. „Konvergenz“ wird feststellt, wenn dieser Wert unterhalb eines vorgegebenen Schwellenwertes ist. Ein weiterer Ansatz zum Stoppen des Algorithmus ist, wenn das Verhältnis zwischen dem neuen Voxel und der gesamten Anzahl der Voxel im Volumen (pro Iteration) unterhalb eines bestimmten Schwellenwertes ist. In [Des01] wurde ein „Einfrieren-Schema“ eingeführt, das basierend auf geodätischen Distanzen die Ausbreitung regelt. Schließlich ist es auch sehr beliebt, einfach den Algorithmus nach einer vordefinierten Anzahl von Iterationen zu stoppen. Eine umfassende Einführung in die verschiedenen Aspekte der Level-Set-Methode und ihre mathematischen Grundlagen kann man in [Set99] lesen.
Berechnung von der Ausbreitung der Front durch eine aktive Kontur ist eine zeitaufwendige Aufgabe. In ihrer ursprünglichen Form (Osher und Sethian [Set88]) muss die partielle Differenzialgleichung über den gesamten Bildbereich gelöst werden. Auch die Aktualisierungsschemen, die verwendet werden, erfordern sehr kleine Zeitschritte, um sicherzustellen, dass keine Bilder-Merkmale im Ergebnis fehlen, oder die Front sich nicht schwingt. Diese beiden Probleme sind in verschiedenen Arbeiten behandelt, von denen einige hier erwähnt werden.
Im Bereich der Bildsegmentierung oder Bildverfolgung ist es normalerweise nicht nötig, explizit die Level-Set-Funktion ϕ über das ganze Bild zu berechnen, weil nur ihre Grenze (d. h. das Zero-Level-Set) als das Ergebnis verwendet wird. Darüber hinaus sind in den meisten Fällen nur sehr wenige Punkte in der Nähe der Grenze für die Berechnung der verschiedenen Geschwindigkeitsfunktionen erforderlich. Deshalb hat Adalsteinson in [Set94] einen Ansatz zur Berechnung der Level-Set-Funktion vorgeschlagen, der nicht im ganzen Bildbereich, sondern nur auf einem „schmalen Band“ um die Zero-Ebene eingesetzt wird. D. h. die Berechnung der Level-Set-Funktion beschränkt sich auf die Punkte, die sich in einen kleinen Bereich um Zero-Ebene befinden. Dieser Ansatz spart Rechenaufwand erheblich, zum Beispiel in 3D von $O(n^3)$ und $O(nk^2)$ pro Zeitschritt, wobei n die Anzahl der Pixel in jeder Dimension und k die Breite des schmalen Bands ist. Darüber hinaus muss die vorzeichenbehaftete Abstandfunktion nicht bei jedem Schritt neu berechnet werden, sondern nur, wenn die Front nah an den Rand des Schmalbands kommt. Das

heißt, das Schmalband wird neu initialisiert, wenn Werte, die für die Berechnung von bestimmten Geschwindigkeitsfunktionen erforderlich sind, nicht mehr verfügbar sind. Die Aktualisierung der vorzeichenbehafteten Abstandfunktion werden mit Fast-Marching-Methode durchgeführt. Details, numerische Verfahren und Beweise für die verschiedenen Aspekte des Schmalband-Ansatzes können in Adalsteinsson gefunden werden.
Whitaker et al. [Whi02] hat diesen Ansatz noch weiter entwickelt und die „Sparse-Feld-Methode“ vorgeschlagen, die insbesondere für Bildverarbeitung-Anwendungen interessant ist: Da die partiellen Ableitungen für die Berechnung der Ausbreitung für den Schritt für viele Anwendungen nur erste oder zweite Ordnung benötigen, genügt es, nur zwei Ebenen auf jeder Seite der Front zu betrachten. Das heißt, zu jedem Zeitschritt berechnet der Algorithmus nur die tatsächlichen Werte, die für den nächsten Schritt benötigt sind. Natürlich erfordert dies eine Neuinitialisierung zu jedem Zeitschritt, aber seitdem das Schmalband extrem schmal ist, hat dies keine negativen Auswirkungen auf die Rechenzeit. Dieser Ansatz vermeidet auch mögliche Abweichung zwischen dem Zero-Level-Set und der vorzeichenbehafteten Abstandfunktion. In der gleichen Arbeit wurde eine Reihe der Umsetzungsschwierigkeiten präsentiert, um eine effiziente Berechnung der Level-Set-Methode zu ermöglichen.
In Kühne et al. [KWBE02] wurde eine Lösung für das Problem der sehr kleinen Zeitschritte präsentiert. Das vorgeschlagene Schema basiert auf Additiv-Operator-Splitting (AOS) und ermöglicht der Propagation viel größere Zeitschritte, die wiederum den gesamten Prozess der Segmentierung beschleunigen. Dies wird durch eine andere Formulierung des Aktualisierungssystems für einen bestimmten Bildpunkt mit Hilfe von Techniken der nichtlinearen Diffusion-Filterung realisiert. Das Aktualisierungssystem benutzt harmonische Mittelung anstelle von Finite-Differenzen-Systemen, um numerische Probleme zu vermeiden. Das vorgeschlagene Schema von Kühne ist stabil und ermöglicht einen ziemlich großen Zeitschritt. Die experimentellen Ergebnisse wurden auch in Kühne vorgestellt. Die Anzahl der Iterationen für die Segmentierung mit dem AOS-System wurde im Durchschnitt um den Faktor 20 im Vergleich zum traditionellen Schema reduziert. Die Begrenzung dieses Algorithmus ist, dass er mit einem bestimmten Modell eingeschränkt ist, und es muss die vorzeichenbehaftete Distanz-Funktion als Level-Set-Funktion benutzt werden, die aber sehr rechenintensiv zu berechnen ist.

2.3.2 Level-Set-Funktion

Im vorherigen Abschnitt wurden die grundlegenden Prinzipien des impliziten aktiven Konturmodells beschrieben. Es wurde betont, dass die Geschwindigkeitsfunktion F die Propagation der Level-Set-Funktion ϕ regelt. In diesem Abschnitt werden die Auswirkungen der verschiedenen Geschwindigkeitsfunktionen diskutiert und schließlich die Geschwindigkeitsfunktionen, die in dieser Arbeit benutzt wird, im Detail behandelt.
Im Gegensatz zu vielen expliziten Methoden zur Segmentierung in der Bildverar-

beitung muss das Zero-Level-Set nicht in der Nähe von der Grenze des gewünschten Objekts initialisiert werden (obwohl dies Zeit und Rechenleistung spart und in der Regel das Verfahren robuster macht). Stattdessen können die ersten Konturen fast überall auf dem Bild gelegt werden. Nach dem Start des Evolutionsprozesses bewegt sich die Kontur entlang des nach innen oder außen gerichteten Normals und die Geschwindigkeit, mit der sich jeder Punkt auf der Kontur bewegt, wird in diese Richtung durch die Geschwindigkeitsfunktion F errechnet.
Es gibt zwei verschiedene Möglichkeiten, um eine solche Geschwindigkeitsfunktion zu definieren. Der erste Ansatz wird von Osher und Sethian [Set88] eingeführt. Dies erfordert, dass die Kontur vollständig innerhalb oder außerhalb des Objekts, das segmentiert werden sollte, platziert wird. Da externe Kräfte für die Kontur nur aus den Bilddaten um das Zero-Level-Set berechnet werden, muss sich die Front so erweitern oder verkleinern, bis sie zu einer geschlossenen Grenze, die durch die Eigenschaften der Geschwindigkeitsfunktion F definiert wird, konvergiert (d. h., die Ausbreitungskraft wird gedämpft, wenn sich die Front am Ziel annähert). Deshalb, wenn Teile der ursprünglichen Kontur innerhalb und außerhalb des gewünschten Objekts platziert werden, ist es oft nicht möglich, die korrekte geschlossene Grenze zu finden und eine Segmentierung wird in der Regel nicht gelungen. Aufgrund seiner Abhängigkeit von dem Zero-Level-Set (und darauffolgend an der Grenze des gesuchten Objekts) wird dieser Ansatz als Grenze-basierte Methode gekennzeichnet. Details, wie man mit einer solchen Geschwindigkeitsfunktion definiert, können in [Set88], [Set99] gelesen werden.
Der zweite Ansatz, der erstmals von Chan und Vese [Ves01] vorgeschlagen wird, basiert auf Regionsinformationen und bestimmt die Geschwindigkeit für jeden Punkt basierend auf der relativen Position (innerhalb oder außerhalb) aller Pixel zu der Kontur. Dadurch ändert sich die Evolution des Level-Sets zu einem Optimierungsprozess auf Basis von Gradienten-Abstiegs. Je nach Definition des jeweiligen Geschwindigkeitsterms soll die initiale Kontur entweder über eine Objektgrenze platziert oder es spielt keine Rolle, wo diese Kontur initialisiert wird. Da in dieser Arbeit diese Art von Geschwindigkeitsfunktion benutzt wird, werden demnächst die Grundlagen dieses Konzepts ausführlich vorgestellt.
Ein bekanntes Problem bei Bildsegmentierung mit der obigen Beschreibung der Level-Set-Methode ist die Tatsache, dass ein Gradient-basierter Geschwindigkeitsterm nie genau 0 an den Grenzen der Objekte wird und damit die Verbreitung der Front nicht an diesen Grenzen angehalten wird. Darüber hinaus ist es schwierig, Objekte zu segmentieren, deren Grenzen nicht durch Gradient definiert werden.
Formulierungen der Region-basierten Level-Set-Methode wurden erst mal von Chan und Vese [Ves01] und Tsai [Wil01a] vorgeschlagen. Diese Ansätze beruhen sich auf der Arbeit von Mumford und Shah [Sha89], die eine Energiefunktion zur Segmentierung eines Signals u in nahezu homogenen Regionen getrennt durch eine glatte Kurve C. Unter der Annahme, dass eine Lösung f existiert, ist diese Energiefunktion

gegeben durch:

$$E(u,C) = \alpha \int_{\Omega} (f-u)^2 dx + \beta \int_{\Omega \backslash C} |\nabla f|^2 dx + \gamma \oint_{C} ds \tag{2.12}$$

Hier sorgt der erste Term dafür, dass f eine gute Approximation des Bildes u wird, und der zweite Term enthält die Glättung der Kurve durch die Begünstigung homogener Gebiete innerhalb und außerhalb von C, und der letzte Term bestraft übermäßige Bogenlänge. Offensichtlich hängt eine Segmentierung, die auf die Eigenschaften aller Pixel innerhalb und außerhalb einer geschlossenen Kontur C basieren, nicht mehr von Gradientinformation der Kante ab.
Eine Funktion, die sehr ähnlich wie Gleichung 2.12 ist, kann auch in Level-Set-Methode integriert werden und ermöglicht somit den Einsatz von Regionseigenschaften in einer Geschwindigkeitsfunktion. Weitere Informationen über das ursprüngliche Mumford-Shah-Framework können in [Sha89] ausführlich gelesen werden.
Die folgende detaillierte Beschreibung einer Region-basierten Level-Set-Formulierung basiert auf dem Papier von Chan und Vese [Ves01]. Sei Ω eine beschränkte offene Teilmenge von $\Re^2$. Wie bereits erwähnt, wird es davon ausgegangen, dass ein Bild $u : \Omega \rightarrow R$ aus zwei Regionen mit etwa stückweisen konstanten Intensitäten besteht. Angenommen, dass das gewünschte Objekt durch $u_i = \omega$ repräsentiert wird, dann ist der Bereich außerhalb des Objekts $u_o = u \backslash \omega$. Die durchschnittliche Intensität der Pixel innerhalb dieser Regionen werden entsprechend von c_i und c_o bezeichnet. Die Grenze zwischen den beiden Regionen ist C_0. Zur Evaluierung dieser Grenze betrachten wir die folgende Gleichung:

$$F_1(C) + F_2(C) = \int_{inside(C)} |u(x) - c_i| dx + \int_{outside(C)} |u(x) - c_o| dx \tag{2.13}$$

wobei C eine geschlossene Kurve in u ist. Es ist offensichtlich, dass die Kurve C_0 die Gleichung 2.13 minimiert.

$$inf\{F_1(C) + F_2(C)\} \approx F_1(C_0) + F_2(C_0) \approx 0 \tag{2.14}$$

Um sicherzustellen, dass C_0 hinreichend glatt ist, ist es notwendig, dass einige Regularisierungsterme basierend auf der Länge der Kurve C oder dem Bereich innerhalb von C in der Energie-Gleichung enthalten werden.
Die resultierende Gleichung wird dann gegeben durch

$$\begin{aligned} F(c_i, c_o, C) &= \mu.Length(C) + \nu.Area(inside(C)) \\ &+ \lambda_1 \int_{inside(C)} |u(x) - c_i|^2 dx \\ &+ \lambda_2 \int_{outside(C)} |u(x) - c_o|^2 dx \end{aligned} \tag{2.15}$$

mit den Gewichtungen u, ν, λ_1, $\lambda_2 \geq 0$. Nach Chan und Vese [Ves01] werden oft $\lambda_1 = \lambda_2 = 1$ und $v = 0$ gesetzt. Die obige Formulierung ist offenbar ein Spezialfall der Mumford-Shah-Funktion mit dem Region-basierten Term, der theoretisch für eine weitere Verfeinerung der Lösung zuständig ist.
Wenn Gleichung 2.15 in Level-Set-Methode integriert wird, wird das Zero-Level-Set durch die Kontur C repräsentiert. Schließlich haben wir:

$$\begin{aligned} C = \partial\omega &= \{x \in u : \phi(x) = 0\}, \\ inside(C) = \omega &= \{x \in u : \phi(x) > 0\}, \\ outside(C) = \Omega\backslash\omega &= \{x \in u : \phi(x) < 0\}. \end{aligned} \tag{2.16}$$

Mit der Heaviside-Funktion

$$H(x) = \begin{cases} 1, & \text{wenn} \quad x \geq 0, \\ 0, & \text{wenn} \quad x < 0 \end{cases} \tag{2.17}$$

sowie der Dirac-Funktion:

$$\delta(x) = \frac{d}{dx} H(x) \tag{2.18}$$

ist es jetzt möglich, die Gleichung 2.15 umzuschreiben

$$\begin{aligned} F(c_i, c_o, \Phi) &= \mu \int_\Omega \delta(\Phi(x)) |\nabla\phi(x)| dx \\ &+ \nu \int_\Omega H(\Phi(x)) dx \\ &+ \lambda_1 \int_\Omega |u(x) - c_i|^2 H(\phi(x)) dx \\ &+ \lambda_2 \int_\Omega |u(x) - c_o|^2 (1 - H(\phi(x))) dx \end{aligned} \tag{2.19}$$

Die Konstanten $c_i = c_i(\phi)$ und $c_o = c_o(\phi)$ können als durchschnittliche Intensitäten der Pixel, wo die Heaviside-Funktionen entsprechend die Werte 1 und 0 annehmen. Die entsprechende Euler-Lagrange-Gleichung für Φ ist dann:

$$\frac{\partial\phi}{\partial t} = \delta_\epsilon [\mu \nabla \frac{\nabla\phi}{|\nabla\phi|} - \nu - \lambda_1 (u - c_i)^2 + \lambda_2 (u - c_o)^2] = 0 \tag{2.20}$$

mit

$$\begin{aligned} \phi(x, 0) &= \phi_0(x) \quad \text{in} \quad \Omega \\ \frac{\delta_\varepsilon(\phi))}{|\nabla\phi|} \frac{\partial\phi}{\partial\overrightarrow{n}} &= 0 \quad \text{auf} \quad \Omega \end{aligned} \tag{2.21}$$

Die Gleichung 2.20 kann auch als eine Level-Set-Gleichung mit anderer Geschwindigkeitsfunktion F betrachtet werden. Das heißt, $\lambda_1(u - c_i)$ und $\lambda_2(u - c_o)$ ersetzen den

Advektionsterm der Geschwindigkeitsfunktion, sodass die Front an den Stellen expandiert, wo der Unterschied vom Pixelwert zum Hintergrund (c_o) groß ist und schrumpft an den Stellen, wo der Unterschied vom Pixelwert zum segmentierten Objekt (c_i) groß ist.
Die Gleichung 2.20 löst viele Segmentierungsprobleme bei der Verwendung von nur drei bis vier Parameter (je nachdem ob ν benutzt wird). Die Werte von c_i und c_o können vor dem Segmentierungsprozess vordefiniert werden, wodurch die Anzahl der Parameter auf fünf steigert. Eine Approximation von c_i und c_o führt zu einem höheren Risiko, dass C abhängig von der Initialisierung der Kontur zu einem lokalen Minimum konvergiert (d. h. nicht die gewünschte Lösung). Numerische Berechnung für alle Gleichungen wurden auch in Chan und Vese [Ves01] und Chen [Wil01b] gegeben.
Der oben beschriebene Ansatz kann verallgemeinert werden, um mehrphasige (oder Multi-Flow) Segmentierung zu ermöglichen, d. h. die Segmentierung von 2^n Klassen von Regionen in einem Bild mithilfe von n Level-Set-Funktionen (siehe [Cha02]).

2.4 Schnelle Implementation

Der ursprüngliche Ansatz von Chan und Vese [Ves01] ist eine gute Lösung für das Segmentierungsproblem, in dem das Bild aus zwei etwa konstanten Regionen besteht. Die Implementierung von diesem Ansatz muss aber eine partielle Differenzialgleichung über das ganze Bild lösen. Diese Implementierung erfüllt damit nur schwierig die Echtzeitbedingung. Shi und Karl [Kar05] haben eine schnelle Implementierung für Level-Set-Methode vorgestellt, die nur die Punkte in zwei Listen bearbeitet. Diese Implementierung beschleunigt die traditionelle Level-Set-Methode deutlich.

2.4.1 Grundimplementierung der Level-Set-Methode

In diesem Abschnitt analysieren wir die Bewegung einer implizit repräsentierten Oberfläche im allgemeinen $K(K \geq 2)$ dimensionalen euklidischen Raum $\Re^K$. Basierend auf dieser Analyse wird eine neue Strategie für die Implementierung der Level-Set-basierte Oberflächenevolution vorgeschlagen.
Eine Kurve C wird implizit als das Zero-Level-Set-Funktion ϕ repräsentiert. Hier wählen wir $\phi < 0$ für die Punkte innerhalb der Kurve C und positive für die außerhalb C. Wir gehen davon aus, dass ϕ über eine Domäne $D \subset \Re^K$ definiert wird, und es ist in einem Gitter der Größe $M_1 \times M_2 \times ... \times M_K$ diskretisiert. D bezeichnet die Menge aller Punkte im Gitter. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass ein Raster die Größe 1×1 hat. Für einen Punkt x im Gitter bezeichnen wir seinen Koordinaten als $x = (x_1, x_2, ..., x_K)$.
Angesichts dieser impliziten Repräsentation können wir die zwei Listen benachbarter

Punkte L_{in} und L_{out} für die Kurve C wie folgt definieren:

$$L_{out} = \{x|\phi(x) > 0 \text{ und } \exists y \in N(x) \text{ sodass } \phi(y) < 0\},$$
$$L_{in} = \{x|\phi(x) < 0 \text{ und } \exists y \in N(x) \text{ sodass } \phi(y) > 0\}. \quad (2.22)$$

wobei $N(x)$ die Menge aller Nachbarpunkte von x ist, die wie folgt definiert wird:

$$N(x) = \{y \in D| \sum_{k=1}^{K} |y_k - x_k| = 1\} \forall x \in D. \quad (2.23)$$

Wenn wir die Menge der Punkte innerhalb der Kurve C als Ω bezeichnen, können wir die Definition von L_{in} und L_{out} in Bezug auf die Menge Ω und seine Ergänzung D
Ω im Gitter D wie folgt ausdrücken:

$$L_{out} = \{x|x \in (D \backslash \Omega) \text{ und } \exists y \in N(x) \text{ sodass } y \in \Omega\}.$$
$$L_{in} = \{x|x \in \Omega \text{ und } \exists y \in N(x) \text{ sodass } y \in (D \quad (2.24)$$
$$\Omega)\}. \quad (2.25)$$

Wie man aus Abbildung 2.8 sehen kann, ist L_{in} die Liste der benachbarten Gitterpunkte, die innerhalb von C sind und L_{out} ist die Liste der benachbarten Gitterpunkte, die außerhalb von C sind. Für eine feste Kurve C kann die Definition von ϕ beliebig sein, und es gibt unendlich viele Möglichkeiten, aber die beiden Listen der benachbarten Gitterpunkte L_{in} und L_{out} werden eindeutig durch C bestimmt. Zunächst werden wir die Beziehung zwischen den Bewegungen der Kurve und den beiden Listen L_{in}, L_{out}.
Um die Kurve C mit einer Geschwindigkeit F auszubreiten, wird in der klassischen Level-Set-Methode von Osher und Sethian [Set88] die folgende Differenzialgleichung numerisch über das ganze Bild gelöst:

$$\frac{d\phi}{dt} + F|\nabla\phi| = 0 \quad (2.26)$$

Da sich die Funktion ϕ ständig weiter entwickelt, breitet sich die Kurve C auch ständig aus. Wir veranschaulichen dies in Abbildung 2.8 das Ergebnis des Entwicklungsprozesses der Kurve C. Am Punkt A bewegt sich die Kurve nach außen, indem der Wert von ϕ am Punkt A von positiv auf negativ wechselt. Am Punkt B bewegt sich die Kurve nach innen und spaltet sich in zwei Kurven, indem sich der Wert von ϕ am Punkt B von negativ auf positiv verändert. Das alles geschieht auch normal in der klassischen Level-Set-Methode, außer dass sie die rechenintensive partielle Differenzialgleichung in 2.26 lösen muss, sodass der Ansatz nicht echtzeitfähig ist. Aber wenn wir nur darum sorgen, welche Punkte innerhalb und außerhalb der Kurve C sind, können wir das gleiche Ergebnis durch Analyse des Verhältnisses zwischen C und L_{in}, L_{out} erreichen. Um die Kurve am Punkt A nach außen zu bewegen, müssen wir den Punkt A einfach von L_{out} zu L_{in} wechseln. Ebenso müssen wir nur

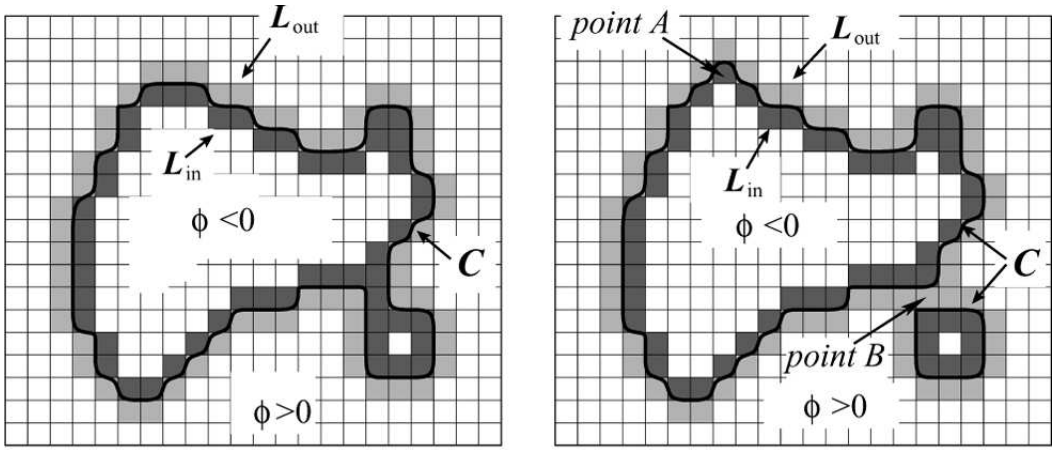


Abbildung 2.8: a) Implizite Repräsentation der Kurve C durch zwei Listen von Punkten L_{in} und L_{out} in der Level-Set-Methode. b) Evolution der Kurve C durch Tauschen der Punkte zwischen zwei Listen [Kar05]

den Punkt B von L_{in} zu L_{out} wechseln, um die Kurve nach innen zu bewegen. Um Punkte zwischen den beiden Listen L_{in} und L_{out} zu wechseln, müssen wir die Werte an den bestimmten Stellen der Level-Set-Funktion ϕ mit dem entsprechenden Vorzeichen setzen. Durch die Anwendung eines solchen Verfahrens auf alle Punkte von L_{in} und L_{out} können wir C einen Punkt überall entlang der Kurve nach außen oder nach innen mit einer minimalen Rechenzeit bewegen. Diese Beobachtung bildet die Grundlage die schnelle Implementierung der Level-Set-Methode.
Basierend auf den obigen Beobachtungen wird die schnelle Implementierung für die Level-Set-basierte Kurvenevolution durch einfache Wechseln-Operationen zwischen den beiden Listen L_{in} und L_{out} erklärt. In dem Algorithmus muss man die partiellen Differenzialgleichungen nicht lösen, sodass signifikante Beschleunigungen im Vergleich zur „Narrow-Band-Methode“ erreicht werden können.
In diesem ersten Schritt wird die Implementierung der Level-Set-Methode für eine allgemeine Geschwindigkeitsfunktion F präsentiert.
Die Datenstruktur für die Implementierung ist einfach und wird folgt definiert:

- Ein Array für die Level-Set-Funktion ϕ;
- Ein Array für die Geschwindigkeitsfunktion F;
- Zwei Listen der benachbarten Punkte: L_{in} und L_{out}.

Die Punkte innerhalb von C aber nicht in L_{in} werden als innere Punkte benannt und die Punkte außerhalb von C aber nicht in L_{out} werden als äußere Punkte benannt. Für eine schnellere Berechnung werden die Werte von ϕ aus einer begrenzten Anzahl

von ganzen Zahlen $(-3, -1, 1, 3)$ gewählt. Die genauere Definition für Level-Set-Funktion ϕ wird wie folgt definiert:

$$\phi(x) = \begin{cases} 3 \text{ wenn } x \text{ ein äußerer Punkt ist ;} \\ 1 \text{ wenn } x \in L_{out} \text{ ist ;} \\ -1 \text{ wenn } x \in L_{in} \text{ ist ;} \\ -3 \text{ wenn } x \text{ ein innerer Punkt ist ;} \end{cases} \tag{2.27}$$

Durch die obige Definition von ϕ kann die relative Position eines Punktes in Bezug auf das Zero-Level-Set einfach bestimmt werden. Für die Evolutionsgeschwindigkeit wird nur das Vorzeichen im Algorithmus verwendet, sodass F auch nur die Werte 1, 0 oder -1 annimmt. Die beiden Listen L_{in} und L_{out} sind bidirektional verbunden, sodass das Einfügen und Löschen leicht gemacht werden kann. Bevor wir in Details des Algorithmus eingehen, werden zuerst zwei grundlegende Operationen, die auf unsere Datenstruktur angewendet werden, definiert. Die Operation *switch_in*() für einen Punkt $x \in L_{out}$ ist wie folgt definiert:

switch_in(x):

- Schritt 1: Löschen x aus L_{out} und fügen es in L_{in} hinzu. Setze $\phi(x) = -1$.
- Schritt 2: $\forall y \in N(x)$ erfüllt $\phi(y) = 3$, fügen y in L_{out} hinzu, und setze $\phi(y) = 1$.

Der erste Schritt in *switch_in*() wird x aus L_{out} zu L_{in} gewechselt. Mit $x \in L_(in)$ werden jetzt alle seine Nachbarn, die vorher äußere Punkte waren, in L_{out} in der zweiten Stufe hinzugefügt. Durch die Anwendung einer *switch_in*()-Operation auf jeden Punkt in L_{out} wird die Kurve an dieser Stelle um einen Punkt nach außen verschoben. Auch die *switch_out*()-Operation für einen Punkt $x \in L_{in}$ ist wie folgt definiert:

switch_out(x):

- Schritt 1: Löschen x aus L_{in} und fügen es in L_{out} hinzu. Setze $\phi(x) = 1$.
- Schritt 2: $\forall y \in N(x)$ erfüllt $\phi(y) = -3$, fügen y in L_{in} hinzu, und setze $\phi(y) = -1$.

Durch die Anwendung einer *switch_out*()-Operation auf einen Punkt von L_{in} verschieben wir die Kurve um einen Punkt nach innen.

Bei jeder Iteration berechnen wir zuerst die Geschwindigkeitsfunktion an allen Punkten von L_{out} und L_{in} und speichern die Vorzeichen der Werte im Array F. Danach scannen wir sequenziell durch die beiden Listen, um die Kurve zu entwickeln. Genauer gesagt scannen wir zunächst durch die Liste L_{out} und wenden die $switch\_in()$-Operation auf den Punkt an, wenn $F > 0$ an diesem Punkt ist. Dieser Scan verarbeitet die Teile der Kurve mit positiven Geschwindigkeiten und verschiebt sie um einen Punkt nach außen. Nach diesem Scannen werden einige Punkte in L_{in} zu inneren Punkten umgewandelt, und werden aus L_{in} gelöscht. Wir scannen dann durch die Liste L_{in} und wenden die $switch\_out()$-Operation auf den Punkt mit $F < 0$ an. Dieser Scan verschiebt die Teile der Kurve mit negativen Geschwindigkeiten um einen Punkt nach innen. Ebenso werden äußere Punkte von L_{out} nach diesem Scan gelöscht. Nach einem Scan durch die beiden Listen wird eine Stopbedingung überprüft. Die Iteration stoppt, wenn diese Bedingung erfüllt ist. In der Implementierung wird folgender Stopbedingung verwendet:

Stopbedingung:
(a) Die Geschwindigkeitsfunktion an allen Punkten in zwei Listen:

$$\begin{aligned} F(\underline{x}) &\leq 0 \ \forall x \in L_{out}; \\ F(\underline{x}) &\geq 0 \ \forall x \in L_{in}; \end{aligned} \tag{2.28}$$

oder (b) Eine maximale Anzahl der Iterationen erreicht.

Die Komplexität dieses Algorithmus ist linear zu der Anzahl der benachbarten Punkte in zwei Listen. Nehmen wir die maximale erforderliche Anzahl von Operationen für eine $switch\_in()$ oder $switch\_out()$-Operation als P_1 und die Anzahl der Operationen für das Löschen des inneren oder äußeren Punktes von L_{in} oder L_{out} als P_2 an. Dann ist die gesamte Anzahl der Operationen pro Iteration $(P1 + P2)(length(L_{out}) + length(L_{in}))$, wobei $length(L_{out})$ und $length(L_{in})$ die Anzahl der Punkte in den Listen sind.
Im Vergleich zu früheren Narrow-Band-Methoden beschränkt der Algorithmus die Berechnungen nur auf beiden Listen L_{in} und L_{out}. Der grundlegende Unterschied ist, dass der Algorithmus die Kurve ohne Lösung einer partiellen Differenzialgleichung entwickelt, während die großen Vorteilen der Level-Set-Methode gehalten werden, z. B. die automatische Behandlung von topologischen Veränderungen und die Allgemeingültigkeit des numerischen Verfahrens für beliebige Dimensionen. Da wir keine partielle Differenzialgleichung lösen müssen, wird die numerische Stabilität von dieser Methode gut garantiert und die Reinitialisierung ist auch kein Problem mehr. Da außerdem alle Berechnungen nur mit Ganzzahlen sind, kann der Algorithmus noch effizienter arbeiten.

2.4.2 Schnelle Glätten-Regulasierung

Bei der Anwendung der Kurvenevolution in vielen Bildverarbeitungsanwendungen besteht die Evolutionsgeschwindigkeit F oft aus zwei Teilen: einer von den Bildeigenschaften abhängigen externen Geschwindigkeit F_{ext} und einer von Glätten-Regularisierung abhängigen internen Geschwindigkeit F_{int}. Eine übliche Wahl von F_{int} ist die Krümmung der Kurve (Mumford und Shah [Sha89]; Chan und Vese [Ves01]; Tsai [Wil01a]). In der Level-Set-Methode ist die numerische Berechnung der Krümmung mit der Funktion ϕ sehr rechenintensiv. Wenn ϕ als vorzeichenbehaftete Abstandfunktion gewählt wird, entspricht die Krümmung dem Laplace-Operator von ϕ. Aus der Theorie des Scale-Spaces (Hummel [Hum86]; Koenderink [Koe84];) wissen wir, dass die Evolution einer Funktion bezogen aus ihrer Laplace einer Gauß-Filterung der selbenen Funktion entspricht.
Es wurde auch gezeigt, dass Diffusionsprozesse auch eine Krümmung-basierte Evolution erzeugen können (Merriman [Osh94]). Motiviert durch diese Beobachtungen, wurde eine neue Methode zur Glätten-Regularisierung im Evolutionsprozess der Kurve basierend auf Gauß-Filterung der Level-Set-Funktion vorgestellt.
In der Methode von Shi et al. [Kar05] werden die von Bildeigenschaften abhängig Geschwindigkeit F_{ext} und die Glätten-Regularisierung in zwei Zyklen getrennt. Dies unterscheidet sich von früheren Ansätzen (Mumford und Shah [Sha89]; Chan und Vese [Ves01]; Tsai [Wil01a]), wo die Geschwindigkeit, die die Regularisierung der Krümmung verursacht, auch in F_{ext} behaltet wird. Deshalb wird die rechenintensive Krümmungsregularisierung in jeder Iteration durchgeführt. In der schnellen Implementierung wird die Glätte-Regularisierung mit einer einfachen Gauß-Filterung realisiert. Da diese Glättung von der durch F_{ext} getriebene Evolution getrennt ist, kann sie viel weniger häufig durchgeführt werden, wenn es nicht zu viel Rausch gibt. Dieser Ansatz reduziert nicht nur den Rechenaufwand drastisch, sondern ermöglicht auch tiefer konkave Regionen der Kurve ohne die Glätte der Kontur zu verlieren.
Der Algorithmus hat jetzt schließlich zwei Stufen. In einem Zyklus des Algorithmus werden N_a Iterationen für die Kurvenevolution durch die allgemeine Geschwindigkeit F_{ext} durchgeführt. In der zweiten Stufe wird Gauß-Filterung auf der Level-Set-Funktion angewendet, um die Kurve zu glätten. Wir wählen einen isotropen und diskreten gaußförmigen Filter G der Größe $N_g \times N_g \times ... \times N_g$ im K-dimensionalen Raum. In dieser Arbeit wurde der folgende 2-D Gauß-Filter mit $N_g = 9$ verwendet:

$$G = \begin{bmatrix} \frac{1}{1600} & \frac{2}{1600} & \frac{4}{1600} & \frac{8}{1600} & \frac{16}{1600} & \frac{8}{1600} & \frac{4}{1600} & \frac{2}{1600} & \frac{1}{1600} \\ \frac{2}{1600} & \frac{4}{1600} & \frac{8}{1600} & \frac{16}{1600} & \frac{32}{1600} & \frac{16}{1600} & \frac{8}{1600} & \frac{4}{1600} & \frac{2}{1600} \\ \frac{4}{1600} & \frac{8}{1600} & \frac{16}{1600} & \frac{32}{1600} & \frac{64}{1600} & \frac{32}{1600} & \frac{16}{1600} & \frac{8}{1600} & \frac{4}{1600} \\ \frac{8}{1600} & \frac{16}{1600} & \frac{32}{1600} & \frac{64}{1600} & \frac{128}{1600} & \frac{64}{1600} & \frac{32}{1600} & \frac{16}{1600} & \frac{8}{1600} \\ \frac{16}{1600} & \frac{32}{1600} & \frac{64}{1600} & \frac{128}{1600} & \frac{256}{1600} & \frac{128}{1600} & \frac{64}{1600} & \frac{32}{1600} & \frac{16}{1600} \\ \frac{8}{1600} & \frac{16}{1600} & \frac{32}{1600} & \frac{64}{1600} & \frac{128}{1600} & \frac{64}{1600} & \frac{32}{1600} & \frac{16}{1600} & \frac{8}{1600} \\ \frac{4}{1600} & \frac{8}{1600} & \frac{16}{1600} & \frac{32}{1600} & \frac{64}{1600} & \frac{32}{1600} & \frac{16}{1600} & \frac{8}{1600} & \frac{4}{1600} \\ \frac{2}{1600} & \frac{4}{1600} & \frac{8}{1600} & \frac{16}{1600} & \frac{32}{1600} & \frac{16}{1600} & \frac{8}{1600} & \frac{4}{1600} & \frac{2}{1600} \\ \frac{1}{1600} & \frac{2}{1600} & \frac{4}{1600} & \frac{8}{1600} & \frac{16}{1600} & \frac{8}{1600} & \frac{4}{1600} & \frac{2}{1600} & \frac{1}{1600} \end{bmatrix} \tag{2.29}$$

Da wir nur Zero-Level-Set glätten wollen, berechnen wir das Ergebnis der Filterung von Level-Set-Funktion ϕ durch den G nur auf die Gitterpunkte in L_{out} und L_{in}. Um die Integerwerte der Funktion ϕ wie in (6.5) festzuhalten, benutzen wir diese Ergebnisse des Gauß-Filters nicht direkt. Stattdessen, wenden wir $switch\_in()$ oder $switch\_out()$-Operation auf den Punkt an, nur wenn das Vorzeichen des Filterausgangs im Vergleich zum ursprünglichen Vorzeichen von ϕ an diesem Punkt geändert wird, weil nur in diesem Fall die Kurve verschoben werden soll. Eine Iteration der Glättung wird wie folgt zusammengefasst:

Eine Glättung-Iteration:

- Für jeden Rasterpunkt $x \in L_{out}$, berechnen $G \otimes \phi(x)$. Wenn $G \otimes \phi(x) < 0$, $switch\_in(x)$;
- Für jeden Rasterpunkt $x \in L_{in}$, berechnen $G \otimes \phi(x)$. Wenn $G \otimes \phi(x) > 0$, $switch\_out(x)$.

Im zweiten Zyklus führen wir die Glättung für N_s Iterationen durch. Wenn die Stopbedingung während des ersten Zyklus erfüllt ist, beenden wir den Algorithmus, andernfalls gehen wir zum ersten Zyklus zurück und führen dieses iterativen Prozesses noch mal durch.

2.4.3 Schnelle Aktuallisierung der Geschwindigkeitsfunktion

Frühere Arbeiten von schnellen Implementierungen des Chan-Vese Modells (Lakshmanan et al. [Lak06]; Pan et al. [YPBJD06]) haben nicht erwähnt, wie die mittleren Intensitäten der Pixel außerhalb und innerhalb der Kurve berechnet werden sollen. In der traditionellen Level-Set-Methode, in der partielle Differenzialgleichungen gelöst werden müssen, wird die Level-Set-Funktion ϕ über das gesamte Bild oder über ein Schmalband ausgewertet. Es gibt auch keine direkte Kenntnisse über die Pixel, die nach außen oder nach innen verschoben werden sollen. Als Folge werden die mittleren Intensitäten innerhalb und außerhalb der Kurve für jede Iteration der Kurvenevolution über das gesamte Bild berechnet. Da wir mit der Shis Implementierung genau wissen können, welche Punkte von L_{in} und L_{out} verschoben werden, können wir die mittleren Intensitäten effektiv aktualisiert werden, ohne die Echtzeitbedingung zu verletzen. Die Werte von c_1 und c_2 werden wie folgt initialisiert:

$$c_1 = \frac{i_i(-3)}{tp_i(-3)} \quad , \quad c_2 = \frac{i_o(3)}{tp_o(3)} \tag{2.30}$$

wobei v, $i_i(v)$, $i_o(v)$, $tp_o(v)$ und $tp_i(v)$ wie folgt definiert werden:

$$i_i(v) = \int\limits_{\Omega_v} I(x,y)H(\phi(x,y))dxdy; \tag{2.31}$$

$$tp_i(v) = \int\limits_{\Omega_v} H(\phi(x,y))dxdy; \tag{2.32}$$

$$i_o(v) = \int\limits_{\Omega_v} I(x,y)(1 - H(\phi(x,y))dxdy; \tag{2.33}$$

$$tp_o(v) = \int\limits_{\Omega_v} (1 - H(\phi(x,y))dxdy; \tag{2.34}$$

und $H(x)$ Heaviside-Funktion ist:

$$H(x) = \begin{cases} 1 & \text{wenn} \quad x > 1 \\ 0 & \text{wenn} \quad x < -1 \end{cases} \tag{2.35}$$

Zusammenfassend sind $tp_i(-3)$ und $i_i(-3)$ entsprechend die Anzahl und die Summe der Intensitäten der Pixel, für die $\phi = -3$ ist; $tp_o(3)$ und $i_i(3)$ entsprechend die Anzahl und die Summe der Intensitäten der Pixel, für die $\phi = 3$ ist.
Sobald die ersten Werte für c_1 und c_2 berechnet wurden, können sie effizient für die Evolution der Kurve aktualisiert werden, jedes Mal, wenn *switch_in*()- oder *switch_out*()-Operationen durchgeführt wird, oder wenn ein Pixel aus einer Liste gelöscht wird, ohne dass dieses Pixel in eine andere Liste hinzugefügt wird. Die zwei Operationen *switch_in*(x) und *switch_out*(x) werden deshalb entsprechend modifiziert, um die mittleren Pixel-Intensitäten effektiv online aktualisiert werden zu können. Die modifizierte *switch_in*(x)-Operation ist wie folgt:

Die modifizierte *switch_in*$(\underline{x})$

- Löschen $\underline{x}$ aus L_{out} und fügen es in L_{in} hinzu. Setze $\phi(\underline{x}) = -1$;
- $\forall \underline{y} \in N(\underline{x})$ erfüllt $\phi(\underline{y}) = 3$, fügen $\underline{y}$ in L_{out} hinzu und setze $\phi(\underline{y}) = 1$; $tp_o \rightarrow tp_o - 1$ und $i_o \rightarrow i_o - I(\underline{y})$,

und die modifizierte *switch_out*(x)-Operation enthält folgende Schritte:

Die modifizierte *switch_out*$(\underline{x})$

- Löschen $\underline{x}$ aus L_{in} und fügen es in L_{out} hinzu. Setze $\phi(\underline{x}) = 1$;
- $\forall \underline{y} \in N(\underline{x})$ erfüllt $\phi(y) = -3$, fügen $\underline{y}$ in L_{in} hinzu und setze $\phi(\underline{y}) = -1$; $tp_i \rightarrow tp_i - 1$ und $i_i \rightarrow i_i - I(\underline{y})$,

wobei $N(x)$ die diskreten Nachbarn von x ist, und i_i, tp_i, i_o, tp_o werden durch die Gleichungen 2.31 bis 2.34 definiert.
Nach der Operation *switch_in*() oder *switch_out*() gibt es oft die redundanten Pixel in L_{in} und L_{out}, die gelöscht werden müssen. Bei dem Löschen dieser Pixel müssen wir die Variablen i_i, tp_i, i_o, tp_o auch aktualisieren. Die Schritte zum Löschen der redundanten Punkte in L_{in} und L_{out} werden in zwei Operationen: *remove_in*() und *remove_out* zusammengefasst. Die Schritte von *remove_in*() werden wie folgt definiert:

remove_in(x)**:**

- wenn $\forall y \in N_4(x)$, $\phi(y) < 0$, löschen x von L_{in} und setze $\phi(x) = -3$;
- $tp_i \rightarrow tp_i + 1$ und $i_i \rightarrow i_i + x$,

und entsprechend werden die Schritte der Operation *remove_out*(x) wie folgt definiert:

remove_in(x)**:**

- wenn $\forall y \in N_4(x)$, $\phi(y) > 0$, löschen x von L_{out} und setze $\phi(x) = 3$;
- $tp_o \rightarrow tp_o + 1$ und $i_o \rightarrow i_o + x$

Durch die Addition und Subtraktion der Variablen i_i, tp_i, i_o und tp_o können wir die notwendigen Informationen für die Berechnung der durchschnittlichen Intensitäten c_1 und c_2 effektiv verfolgen, ohne dass wir über das ganze Bild iterieren müssen.
Die obigen Änderungen werden nur für Segmentierungsänderung vorgeschlagen, in der die Bilddaten konstant im ganzen Segmentierungsprozess bleiben. In Verfolgungsanwendungen wird ein neues Bild nach jeder Segmentierungsiteration geladen. Deshalb müssen die Variablen i_i, tp_i, i_o, tp_o entsprechend nach diesem Laden mit den neuen Bilddaten aktualisiert.
Schließlich wird eine Übersicht des modifizierten Algorithmus von Shi und Karl [Kar05] in Tabelle 2.1 zusammengefasst. Das Ergebnis der Verfolgung des Objekts durch die Level-Set-Methode wird im nächsten Kapitel in Kombination mit dem Thinning-Prozess präsentiert.

Zusammenfassung des Tracking-Algorithmus

- **Schritt 1:** Wenn der Algorithmus zum ersten Mal aufgerufen wird: Initialisieren ϕ, F_d, die zwei Listen L_{out} und L_{in}, die Summen der Intensitäten i_i, i_o, und die Anzahl der Punkten innerhalb und außerhalb der Kontur tp_i, tp_o. Wenn schon in Tracking-Iteration: Aktualisieren i_i, i_o, tp_i und tp_o basierend auf dem aktuellen Bild I_curr und dem gespeicherten Bild I_save.
- **Schritt 2:** For $i = 1 : N_a$ do
  - Berechnen die Geschwindigkeitsfunktion F_d für die Pixel in L_{out} und L_{in};
  - Für jeden Pixel $\underline{x} \in L_{out}$, $switch\_in(\underline{x})$ wenn $F_d(\underline{x}) > 0$.
  - Für jeden Pixel $\underline{x} \in L_{in}$, $remove\_in(\underline{x})$.
  - Für jeden Pixel $\underline{x} \in L_{in}$, $switch\_out(x)$ wenn $F_d(\underline{x}) < 0$.
  - Für jeden Pixel $\underline{x} \in L_{out}$, $remove\_out(\underline{x})$.
  - Überprüfen die Stopping-Kondition, wenn diese Kondition erfüllt ist, gehen zum Schritt 3.
- **Schritt 3:** For $i = 1 : N_g$ do
  - Für jeden Pixel $\underline{x}$ in L_{out}, berechnen $G \otimes \phi(\underline{x})$. Wenn $G \otimes \phi(\underline{x}) < 0$, $switch\_in(\underline{x})$;
  - Für jeden Pixel $\underline{x} \in L_{in}$, wenn $\forall \underline{y} \in N(\underline{x})$, $\phi(\underline{y}) < 0$, löschen $\underline{x}$ aus L_{in}, und setzen $\phi(\underline{x}) = -3$.
  - Für jeden Pixel $\underline{x}$ in L_{in}, berechnen $G \otimes \phi(\underline{x})$. Wenn $G \otimes \phi(\underline{x}) > 0$, $switch\_out(\underline{x})$.
  - Für jeden Pixel $\underline{x} \in L_{out}$, wenn $\forall \underline{y} \in N(\underline{x})$, $\phi(\underline{y}) > 0$, löschen $\underline{x}$ aus L_{out}, und setzen $\phi(\underline{x}) = 3$
- **Schritt 4:** Wenn die Stopping-Kondition in Step 2 erfüllt ist, terminiert den Algorithmus, sonst gehen zurück zum Schritt 2.
- **Schritt 5:** Speichern die Intensität-Bild I_{curr} zu I_{save}

Tabelle 2.1: Zusammenfassung der schnellen Implementation der Level-Set-Methode

Kapitel 3

DLO-Struktur

Um die Struktur des linearen Objekts zu erkennen, ist es erwünscht, das Skelett vom Objekt zu bekommen. Diese Darstellung ist näher an der menschlichen Wahrnehmung, deshalb erlaubt sie eine einfachere Methode zur Strukturanalyse und ein intuitives Design der Erkennungsalgorithmen. Darüber hinaus kann die Reduzierung der Bilddaten auf das Wesentliche die Konturverzerrungen unter Beibehaltung der wichtig topologischen und geometrischen Eigenschaften beseitigen. In Praxis, dünne Liniendarstellungen von länglichen Strukturen sind nützlich, kritische Eigenschaften wie Endpunkten, Knotenpunkten sowie die topologischen Verbindungen zwischen den einzelnen Komponenten. Natürlich soll ein effektiver Thinning-Algorithmus im Idealfall die zu verarbeiteten Daten komprimieren, wichtige Merkmale des Objekts behalten, lokales Rauschen beseitigen, und keine eigene Verzerrungen machen. Der Begriff „Skelett“ ist in der Regel verwendet, um die Darstellung eines Objekts durch eine Sammlung von dünnen (oder fast dünnen) Linien und Kurven zu bezeichnen. In dieser Sektion werden einige Grundlagen von Thinning-Algorithmen vorgestellt. Danach wird der Ansatz, der die Information aus der Level-Set-Methode benutzt um den Thinning-Prozess zu beschleunigen, vorgestellt. Schließlich wird der Analysealgorithmus um die Struktur des Objekts zu erkennen, ausführlich erläutert.

3.1 Thinning

Die meisten Thinning-Algorithmen sind iterativ oder, genauer gesagt, sie löschen aufeinanderfolgenden Schichten von Pixeln am Rand des Objekts, bis nur noch ein Skelett bleibt. Das Löschen oder Beibehaltung eines Pixels p sind von der Konfiguration der Pixel in einer lokalen Umgebung von p abhängig. Abhängig von der Art und Weise, wie sie Pixel prüfen, lassen sich diese Algorithmen als sequenziell oder parallel kategorisieren. In den sequenziellen Algorithmen werden Konturpunkte für die Löschung in einer vorgegebenen Reihenfolge geprüft, und dies kann entweder durch Konturverfolgung durchgeführt werden. Parallele Thinning-Algorithmen werden Pixel zum Löschen auf der Grundlage der Ergebnisse der vorherigen Iteration überprüft. Aus diesem Grund sind diese Algorithmen für die Implemen-

tierung auf parallelen Prozessoren, wo die die Löschbedingung erfüllenden Punkte gleichzeitig gelöscht werden können, geeignet ist. Es gibt auch nicht-iterative Thinning-Methoden, die eine bestimmte Mittellinie des Objekts direkt in einer Iteration ohne Prüfung aller einzelnen Pixel produzieren. Ein umfangreicher Survey über verschiedene Thinning-Algorithmen gibt es in [Sue92].
Da die meisten Thinning-Algorithmen über das ganze Bild, oder eine rechteckige Begrenzungsbox des interessierten Objekts iterieren, ist es schwierig, die Echtzeitbedingung zu erfüllen. In dieser Arbeit wurde ein sequenzieller Algorithmus implementiert, der das Ergebnis von Level-Set-Methode direkt benutzen kann. Da wir nur die Punkte in zwei Listen L_{in} und L_{out} benutzen, und unser Objekt nicht zu dicht ist, können wir den Thinning-Prozess beschleunigen, um die Echtzeitbedingung unserer Anwendung zu erfüllen.
Unser Algorithmus funktioniert mit dem Prinzip, dass wir die $switch\_out()$-Operation iterativ auf alle Punkte in L_{in} anwenden, d. h. die Kontur des Objekts schrumpft, bis es eine 1-Pixel-Linie entsteht. Ein negativer Effekt von dem obigen Algorithmus ist, dass die beiden Endpunkte verkürzt werden, wenn die Anzahl der Iteration zu groß ist. Wenn die Anzahl der Iteration aber zu klein ist, bekommt man keine 1-Pixel-Linie. Dieses Problem könnte gelöst werden, indem wir nach zwei Endpunkten suchen und wenden keine Operation auf die Punkte in den Bereichen um diese Punkte anwenden. Das Suchen nach zwei Endpunkten ist aber rechenintensiv, da wir im Worst-Case über das ganze Bild iterieren müssen. In dieser Arbeit wird eine andere Methode implementiert. Wir gliedern unsere Iteration in drei Teilen. In dem ersten Teil werden N_1 Iterationen durchgeführt, in denen alle Punkte in L_{in},die die Löschbedingung erfüllen, mit der Operation $switch\_out()$) nach innen gezogen werden. Die Anzahl der Iterationen N_1 wird groß genug gewählt, damit alle artifizielle Äste identifiziert werden können. Die Punkte, die nur einen Nachbar haben, werden mit ihren originalen Positionen registriert. In dem zweiten Teil werden die Iterationen ähnlich wie im ersten Teil durchgeführt, bis wir nur noch zwei Endpunkte haben. Im dritten Teil werden N_2 ähnliche Iterationen auf alle Punkte in L_{in} außer zwei Endpunkten angewendet. Diese Bedingung garantiert, dass unsere zwei Endpunkte nicht mehr verkürzt werden, sondern nur die Punkte, an denen das Objekt mehr als 2-Pixel dicht ist. Da der erste Iterationszyklus unser Objekt ein bisschen verkürzt, zeichnen wir im Ergebnisbild an zwei Endpunkten zwei Linienstücke zurück, um die verkürzten Längen des Skelettes an zwei Endpunkten zu kompensieren. Diese Maßnahme wiederherstellt die gelöschten Längen unseres Objekts. Das resultierende Skelett ist dann 1-Pixel dicht und hat keine artifiziellen Äste mehr.
Die Abbildung 3.1 zeigt einige Bilder des Thinning-Prozesses. In Abbildung 3.1-a zeigt das Ergebnis von dem Level-Set-Methode. Die grünen Linien sind die Punkte von der Liste L_{out}, die roten Linien sind die Punkte von der Listen L_{in}, und die blauen Punkte sind die Punkte innerhalb der Kontur des Objekts. Die Abbildung 3.1-b zeigt die Listen L_{in} und L_{out} nach dem Thinning-Prozess. Das Ergebnis des Thinning-Prozess ist schließlich die Punkte von der Liste Lin.
Nach dem Thinning-Prozess werden die Punkte bezüglich der Anzahl ihrer Nach-

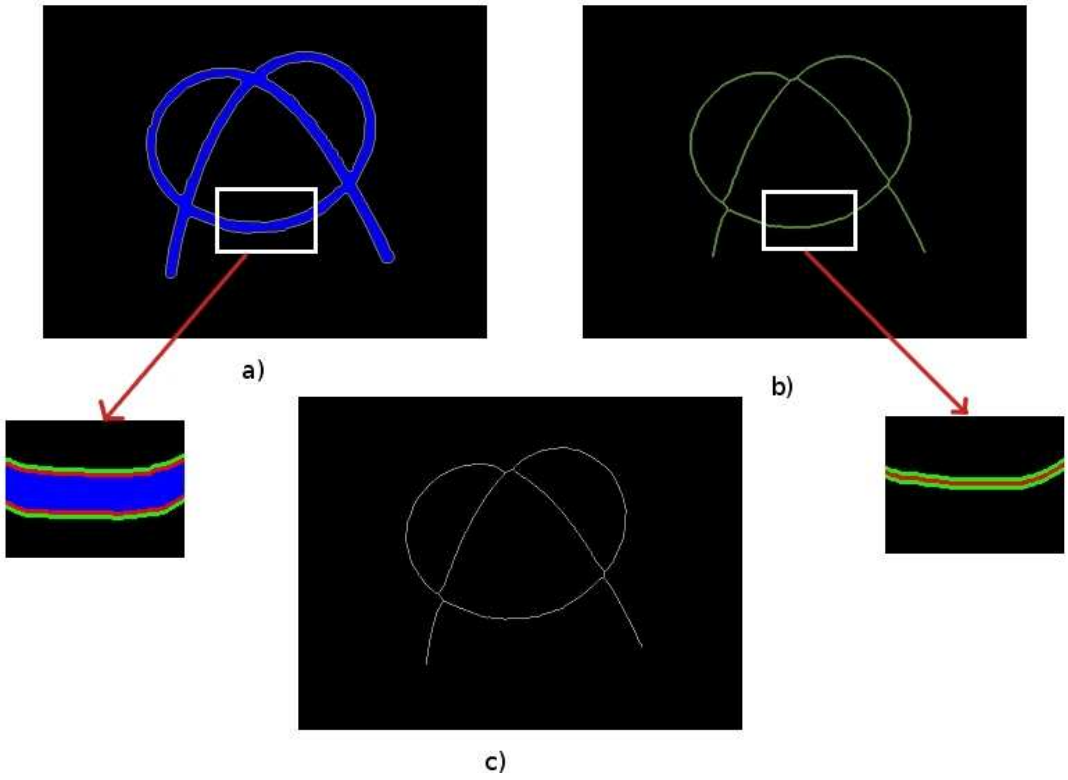


Abbildung 3.1: Der Thinning-Prozess.

barpunkte in 4 Kategorien klassifiziert.

- Die Punkte, die nur einen Nachbar haben, sind die Endpunkte.
- Die Punkte, die zwei Nachbarn haben, sind normale Punkte.
- Die Punkte, die drei Nachbarn haben, sind Bifurkationspunkte.
- Die Punkte, die vier Nachbarn haben, sind Kreuzpunkte.

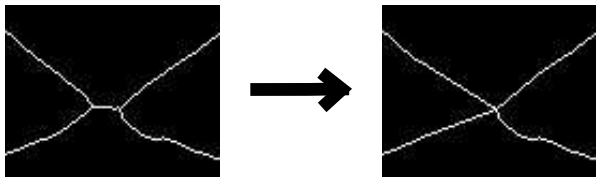


Abbildung 3.2: Korrektur der Birfukation-Effekt.

Bevor wir in Details des Algorithmus eingehen, wird es über einige Charakteristika des Skelettbildes diskutiert.

Wir definieren zwei Hilfsfunktionen für unseren Algorithmus. In der ersten Funktion werden zwei nah an einander liegende Bifurkationspunkte durch einen Kreuzpunkt ersetzt. Wegen dem Effekt vom Thinning-Algorithmus wird ein Kreuzpunkt im Silhouettenbild in zwei Bifurkationspunkten im Skelettbild aufgeteilt (Siehe Abbildung 3.2). Der Abstand zwischen zwei Bifurkationspunkten ist abhängig von der Steigung zweier Objektsegmente am Kreuzpunkt. In unserer Funktion wird ein Schwellenwert für Abstand zwischen zwei Bifurkationspunkten, die eigentlich zu einem Kreuzpunkt gehören, eingeführt. Wenn ein Bifurkationspunkt gefunden wird, wird ein anderer Bifurkationspunkt innerhalb von einem Kreis mit dem Radius gleich dem oben genannten Schwellenwert gesucht. Anschließend werden beide Bifurkationspunkte durch einen Kreuzpunkt ersetzt.
Da der Algorithmus jeden Punkt aus einem Endpunkt zu dem anderen Endpunkt verfolgt, muss er immer wissen, welcher Nachbarpunkt von dem aktuellen Punkt weiter bearbeitet werden soll. Es ist aber schwierig, den nächsten Punkt zu finden, wenn der aktuelle Punkt ein Kreuzpunkt ist, weil der Kreuzpunkt vier Nachbarpunkte hat. Die zweite Funktion hat deshalb die Aufgabe, den entsprechenden nächsten Punkt zu finden, wenn der aktuelle Punkt ein Kreuzpunkt ist. Das Skelett schneidet einen Kreis, der den aktuellen Punkt als Zentrumpunkt annimmt und einen groß genug Radius hat, in 4 Punkten. Je nachdem welche von 4 schon geprüft wurden, können wir den nächst zu bearbeitenden Punkt feststellen.
Das Skelett wird dann in mehreren Segmenten aufgeteilt. Ein Segment wird als ein Array von Punkten zwischen einem Endpunkt und einem Kreuzpunkt oder zwischen zwei Kreuzpunkten. Die Struktur von dem Skelett wird mit einer Sequenz von Endpunkten und Kreuzpunkten kodiert. Ein Beispiel wird in Abbildung 3.3 veranschaulicht.
Basierend auf den zwei beschriebenen Funktionen können wir jetzt die Methode zur Analyse des Skelettes implementieren. Der Algorithmus startet mit der Suche nach einem Endpunkt. Danach bearbeitet der Algorithmus dann den Nachbarpunkt von dem aktuellen Punkt. Ab hier haben wir mehrere Möglichkeiten:

- Wenn der aktuelle Punkt ein normaler Punkt ist, wird der nächste Punkt der noch nicht markierte Punkt.

- Wenn der aktuelle Punkt ein Bifurkationspunkt ist, wird die erste Funktion durchgeführt, um den anderen entsprechenden Punkt zu finden und schließlich beide Bifurkationspunkte in einen Kreuzpunkt zu fusionieren. Die zweite Funktion wird danach durchgeführt, um den nächst zu bearbeitenden Punkt zu finden. Der Algorithmus läuft dann weiter mit dem gefundenen Punkt. Dieser Schritt garantiert, dass das Endergebnis nur die Endpunkte und die Kreuzpunkte als die speziellen Punkte enthält und der negative Effekt von dem Thinning-Algorithmus praktisch eliminiert wird.

- Wenn der aktuelle Punkt ein Kreuzpunkt ist, wird die zweite Funktion gerufen, um den nächsten Punkt zu finden.

Abbildung 3.3: Struktur des Objekts. Die Knoten und Enden werden nummeriert.

Der Algorithmus wird angehalten, wenn der zweite Endpunkt gefunden wird.
Die durch den Algorithmus gefundene Knotensequenz für das Objekt in Abbildung 3.3 ist:

$$0 - 1 - 2 - 3 - 1 - 2 - 3 - 4 \tag{3.1}$$

Die Abbildung 3.4 zeigt das Ergebnis, wenn wir den ganzen Bildverarbeitung-Algorithmus anwenden. Das Ergebnis von dem Algorithmus ist die Struktur des deformierbaren linearen Objekts, die in zwei Arrays gespeichert wird. Das Segment-Array enthält die Informationen von allen Segmenten und das Sequenz-Array speichert die Sequenz von Indexzahlen der speziellen Punkte entlang des Objekts von einem Endpunkt zum anderen Endpunkt. Die Positionen der möglichen Greifpunkte für jedes Segment können mit Hilfe von Segment-Array berechnet werden. In dieser Arbeit werden die Mittelpunkte von den Segmenten als Greifpunkte gewählt.
Die Abbildung 3.4 zeigt das Ergebnis von dem Bildverarbeitungsalgorithmus zur Analyse des deformierbaren linearen Objekts.

3.2 Entknoten

In unserer Anwendung muss es garantiert werden, dass die Dichtung sich nicht gedreht, und sie enthält auch keinen Knoten, bevor ihre zwei Endpunkte zusam-

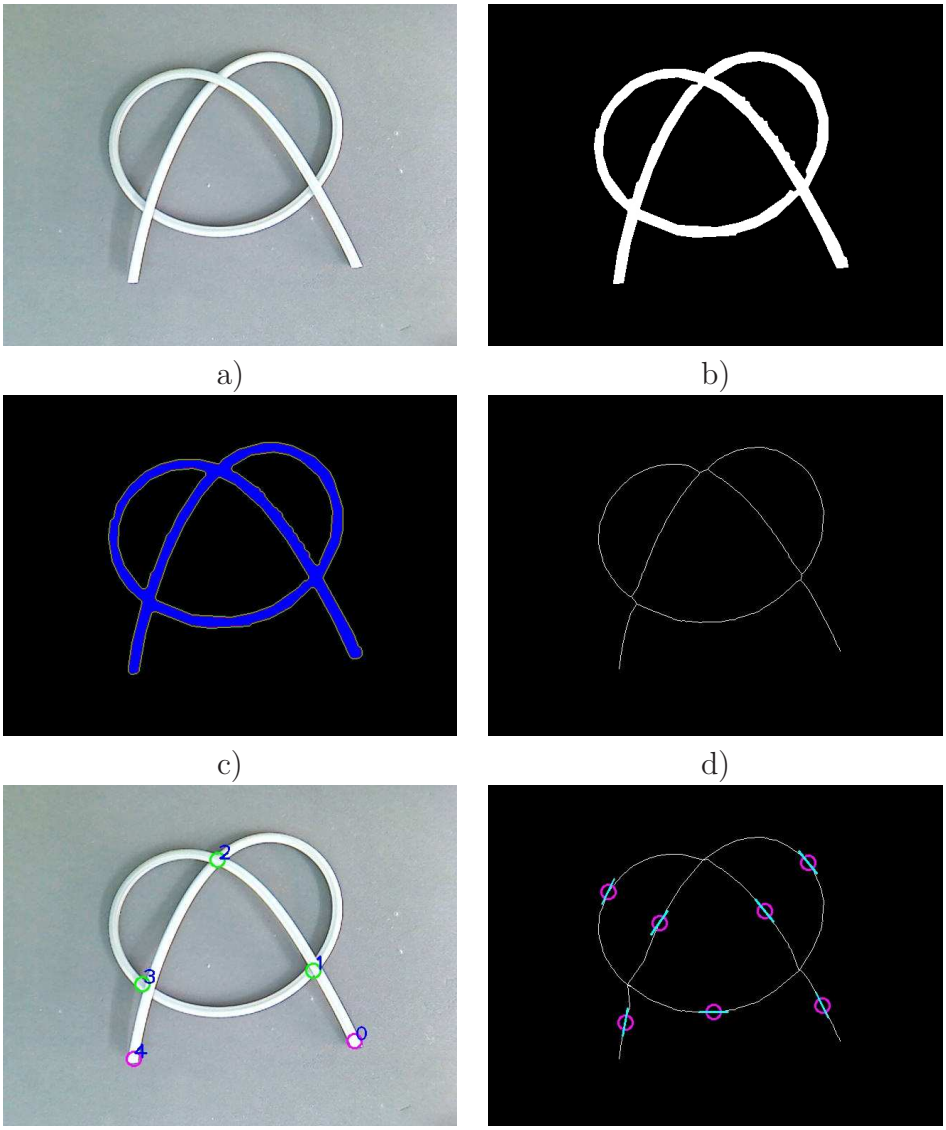


Abbildung 3.4: Ergebnisse von jedem Schritt des Bildverarbeitung-Algorithmus: a) das originale Objekt; b) Hintergrundsubtraktion; c) Verfolung mit Level-Set-Methode; d) Thinning; e) Das Ergebnis; f) Segmente mit Greifpunkten

mengeklebt werden. Basierend auf der Struktur des Objekts hat Wakamatsu ([Hir06a]) eine Methode vorgeschlagen, die sicherstellt, dass kein Knoten erstellt wird, wenn der Roboter zwei Endpunkte des DLOs manipuliert. Demnächst wird diese Methode von Wakamatsu ausführlich erläutert.
In den meisten industriellen Anwendungen manipuliert der Roboter die deformierbaren linearen Objekte durch ihre zwei Endpunkte. Es ist oft erwünscht vor diesen Manipulationen, dass das Objekt „frei“ ist, d. h., es kein Knoten entsteht, wenn die beiden Endungen des Objekts gezogen werden. In der Manipulation der deformierbaren linearen Objekte muss die Verwickelung vermieden werden. Tritt eine unerwartete Verwickelung auf, benötigt es viel Zeit, um das Objekt zu entwirren.
Entwirren ist äquivalent zu Aufknoten. Allerdings kann der Zustand eines verwirrten Objekts komplexer als der eines geknoteten Objekts sein. Darüber hinaus ist es schwierig, den Zustand eines verwirrten Objekts zu erkennen, weil es sich winden kann. Aus diesem Grund sind Erkennung des Objektzustands und Planung der Manipulation sehr wichtig für die Entwirren-Aktion. Die 3D-Informationen von dem Objekt sind nützlich für das Auflösen des verwirrten Objekts. Das Problem ist aber, dass diese Informationen schwierig zu erhalten sind. In der Arbeit von Wakamatsu ([Hir06a]) wurde eine Methode zur Planung der Auflösen-Aktion eines linearen Objekts ohne 3D-Informationen mithilfe von Objektstruktur vorgeschlagen. Der Zustand des Objekts wird zuerst basierend auf der Objektstruktur nach Verwirrung identifiziert. Die Zustandsübergänge werden durch die Einführung von vier grundlegenden Operationen definiert. Dann kann ein möglicher Auflösen-Prozess generiert werden, wenn der aktuelle Zustand der Kreuzung absolut identifiziert ist. Ein Objekt mit auflösbarer Sequenz kann schließlich durch Ziehen seiner beiden Enden auflösen.
Der Zustand eines linearen Objekts kann topologisch durch drei Merkmale nach der Projektion seiner Form auf eine Projektionsebene dargestellt werden. Das erste Merkmal ist die Kreuzungssequenz. Es wird durch Nummerierung der Kreuzungen von einem Endpunkt zum anderen bestimmt. Die i-te Kreuzung wird als C_i symbolisiert. Der erste Endpunkt wird als E_l genannt und der andere Endpunkt ist E_r. Das zweite Merkmal ist die relative Position zwischen zwei Punkte an der Kreuzung, das heißt, welcher Punkt oberhalb/untehalb des anderen liegt. Der obere Punkt wird als C_i^u und der untere Punkt als C_i^u gekennzeichnet. Das dritte Merkmal ist die Helix jeder Kreuzung. In [Hir06a] haben die Autoren vier Paare von grundlegenden Operationen (siehe Abbildung 3.5), die benutzt werden können, um ein lineares deformierbares Objekt zu entwirren. Details über diese Operationen können mehr in der genannten Arbeit gelesen werden. Da wir keine 3D-Information von den Kreuzungen haben, deshalb können wir das erste Merkmal benutzen, um die vorhandenen Knoten aufzulösen. Wir können dann nur zwei Paare von Grundoperation verwenden, die wie in [Hir06a] als CO_I, UO_I und CO_{IV}, UO_{IV} genannt werden.
Wir betrachten zuerst zwei Beispiele von zwei Knoten. Abbildung 3.6-(a-1) zeigt die Silhouette eines Knotens mit 3 Kreuzungen. Die Kreuzungssequenz ist wie folgt

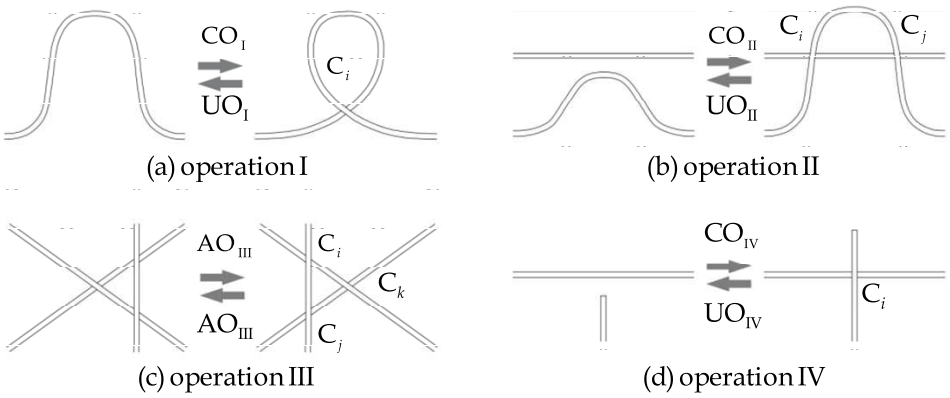


Abbildung 3.5: Grundoperationen [Hir06a]

beschrieben:

$$E_l \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } C_3 \text{ - } C_2 \text{ - } C_1 \text{ - } E_r.$$

Knoten in Abbildung 3.6-(a-2) bis (a-4) haben die gleiche Kreuzungssequenz. Sie enthalten die Teilsequenz ... - C_3 - C_3 - ... , auf die die Operation UO_I angewendet werden kann. Wenn die Kreuzung C_3 gelöscht wird, kann die Kreuzung C_2 auch durch Anwendung der Operation UO_I gelöscht werden. Nach dem Löschen der Kreuzung C_2 können wir die Kreuzung C_1 durch die Anwendung UO_I löschen. Dies bedeutet, dass Knoten in Abbildung 3.6 (a-2) zu (a-4) auflösbar sind. Jeder Knoten mit der Kreuzungssequenz wie in Gleichung 3.2 kann aufgelöst werden, indem ihre beiden Endpunkte unabhängig von der Lage an den einzelnen Kreuzungen gezogen werden. In [Hir06a] wird diese Kreuzungssequenz als auflösbare Kreuzungssequenz definiert. Ein auflösbarer Knoten verfügt über eine auflösbare Kreuzungssequenz. Ein Knoten in Abbildung 3.6-(b-1) hat auch drei Kreuzungen. Sein Kreuzungssequenz ist wie folgt:

$$E_l \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } E_r.$$

Die Knoten in Abbildung 3.6-(b-2) bis (b-4) haben auch diese Kreuzungssequenz. Die Knoten in Abbildung3.6-(b-3) und (b-4) sind beide auflösbar, aber der Knoten in Abbildung 3.6-(b-2) entspricht einer Überhandknoten, das heißt, es ist spannbar. Dies bedeutet, dass ein spannbarer Knoten mit der Kreuzungssequenz wie Gleichung 3.2 vorhanden ist. Daher ist eine solche Kreuzungssequenz nicht auftrennbar. Es wird beachtet, dass die Knoten in Abbildung 3.6-(b-3) und (b-4) aufgelöst werden können, aber sie können nicht von den Knoten in Abbildung3.6-(b-2) unterschieden werden, weil die 3D-Informationen von den Kreuzungen nicht vorhanden sind. So können wir die Kreuzungssequenz von einem Knoten in zwei Arten kategorisieren: auftrennbar und nicht-auftrennbar. Die Knoten in der ersten Kategorie können unabhängig von der Lage der einzelnen Kreuzungen aufgelöst werden, indem ihre beiden Endpunkte

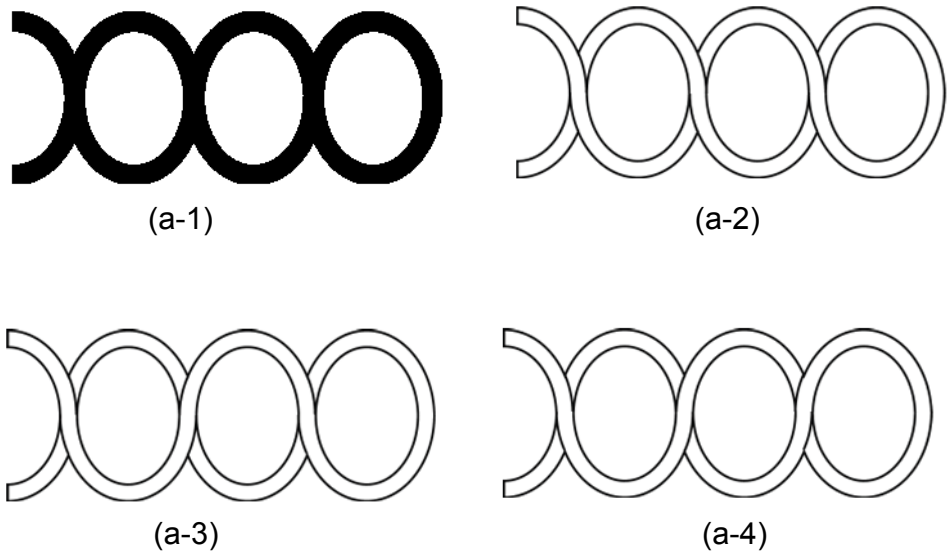


Abbildung 3.6: Auflösbare und nicht-auflösbare Sequenzen der Kreuzpunkte [Hir06a]

gezogen werden, während die Knoten in der zweiten Kategorie je nach Situation durch Ziehen der beiden Endpunkte aufgelöst werden können.
Mit der obigen Betrachtung wird demnächst ein Verfahren zu Auflösen eines linearen

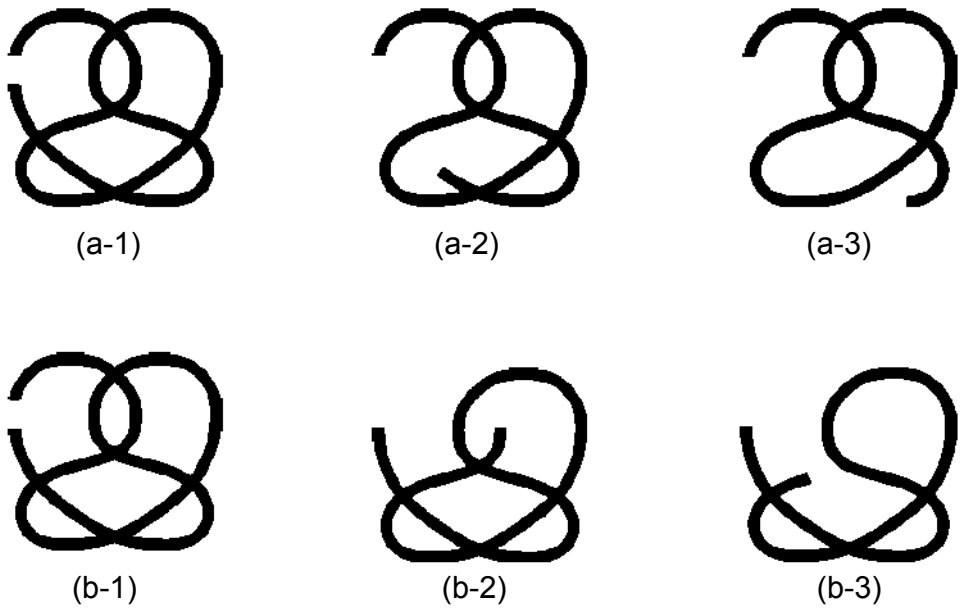


Abbildung 3.7: Der Prozess der Entlösen-Aktion [Hir06a]

deformierbaren Objekts vorgestellt. Nehmen wir an, dass nur die Silhouette eines linearen Objekts wie in Abbildung 3.7-(a-1) gegeben ist. Die Kreuzungssequenz wird dann wie folgt beschrieben:

$$E_l \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } C_4 \text{ - } C_5 \text{ -} C_3 \text{ - } C_2 \text{ - } C_1 \text{ - } C_4 \text{ - } C_5 \text{ - } E_r.$$

Die oben gezeigte Sequenz entspricht einer nicht-auflösbaren Sequenz mit fünf Kreuzungen. Das bedeutet, dass das Objekt sich verwirren kann, wenn seine beiden Endpunkte gezogen werden. Dann betrachten wir die Anwendung der Operation UO_{IV}, sodass das Objekt keine nicht-auftrennbare Sequenz mehr enthält. Wenn wir die Operation UO_{IV} auf den Endpunkt E_l anwenden, ändert sich das Objekt zu dem Zustand wie in Abbildung 3.7. Die Kreuzungssequenz wird wie folgt beschrieben:

$$E_l \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } C_4 \text{ - } C_2 \text{ - } C_1 \text{ - } C_3 \text{ - } C_4 \text{ - } E_r$$

Wir wenden eine weitere Operation UO_{IV} auf den Endpunkt E_l an. Dann wird die folgende Sequenz abgeleitet:

$$E_l \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } E_r$$

Die oben genannte Sequenz entspricht der nicht-auflösbaren Sequenz mit 3 Kreuzungen. Dies bedeutet, dass zusätzliche UO_{IV}-Operationen erforderlich sind, um das

Objekt zu entwirren. Im Gegenteil dazu, wenn wir zwei aufeinanderfolgenden UO_{IV}-Operationen auf der rechten Seite der Folge anwenden, wie in Abbildung 3.7-(b), wird die Kreuzungssequenz wie folgt:

$$E_l \text{ - } C_1 \text{ - } C_2 \text{ - } C_3 \text{ - } C_3 \text{ - } C_2 \text{ - } C_1 \text{ - } E_r$$

Da diese Sequenz ähnlich wie die auflösbare Sequenz in Abbildung 3.6-(a-1) ist, können wir abschließen, dass ein Entwirren-Prozess wie in Abbildung 3.7-(a) durch Analyse automatisch erstellt werden kann. Der generierte Prozess löst die Knoten sequenziell durch Anwendung der grundlegenden Operation auf zwei Enden des Objekts auf. Dies zeigt auch, dass wir ein lineares Objekt ohne Stereokamera entwirren können.
Unauflösbare Sequenzen können aus der Liste der Primknoten in Knotentheorie (Rolfsen et al. [Rol76]) entnommen werden. Wir definieren die folgende Teilsequenz als nicht unauflösbare Teilfolge mit 3 Kreuzungen:

$$\text{... - } C_i \text{ - ... - } C_j \text{ - ... - } C_k \text{ - ... - } C_i \text{ - ... - } C_j \text{ - ... - } C_k \text{ - ... } (i < j < k)$$

Wenn der Kreuzungszustand die oben genannten Kreuzungsfolgen enthält, hat das Objekt einen Teil, der festgeknotet werden kann. Die unauflösbare Sequenz von Gleichung 3.2 ist eine Art dieser Teilfolgen. Wir können auch allgemein unlösbare Teilfolgen für n Kreuzungen definieren. Wenn eine solche unlösbare Teilfolge in der Kreuzungssequenz erkannt wird, löschen wir die in dieser Teilfolge erhaltenen Kreuzungen durch die Anwendung Operationen UO_{IV} auf einen von beider Endpunkten. Wir definieren C_{li} und C_{rj} als $i - ten$ und $j - ten$ Kreuzung, die entsprechend am nähsten des linken und rechten Endpunktes sind. Wenn das Objekt n Kreuzungen hat, gehen wir davon aus, dass nur UO_{IV}-Operation angewandt werden soll, um die k ($k = 1..n$) Kreuzungen zu löschen. Dann überprüfen wir die verbleibenden unauflösbaren Teilfolgen nach dem Löschen der Kreuzungen C_{li} ($i = k, k-1, ..., 1, 0$) und C_{rj} ($j = k-i$). Wenn der Kreuzungssequenz keine unauflösbare Teilfolge mehr enthält, können die restlichen Kreuzungen durch Ziehen der beiden Endpunkte aufgelöst werden. Dies bedeutet, dass das Objekt effizient aufgelöst werden kann. Zum Beispiel die Kreuzung Sequenz von Gleichung 3.2 enthält drei folgende unauflösbare Teilsequenzen:

$$\text{... } - C_1 - \text{ ... } - C_4 - C_5 - \text{ ... } - C_1 - C_4 - C_5 - \text{ ..., .} \tag{3.2}$$
$$\text{... } - C_2 - \text{ ... } - C_4 - C_5 - \text{ ... } - C_2 - \text{ ... } - C_4 - C_5 - \text{ ..., } \tag{3.3}$$
$$\text{... } - C_3 - C_4 - C_5 - C_3 - \text{ ... } - C_4 - C_5 - \text{ ... } \tag{3.4}$$

In diesem Fall kann die Kreuzung $C_{l1} = C_1$ oder $C_{r1} = C_5$ durch Operation UO_{IV} gelöscht werden. Wenn die Kreuzung C_5 gelöscht wird, werden alle diese Teilsequenzen gestrichen. Dann wird das Objekt auflösbar. Teilsequenzen von Gleichung 3.4 und 3.4 bleiben aber unauflösbar, auch wenn der Kreuzung C_1 schon gelöscht wurde. Daher wenden wir die Operation UO_{IV} zuerst besser auf Kreuzung C_5 an. Danach

wird das Objekt vollständig auflösbar, indem ihre beiden Endpunkte gezogen werden. So kann ein effizienter Prozess zum Auflösen eines linearen Objekts abgeleitet werden, selbst wenn nur die Kreuzungssequenz ohne ihre 3D-Informationen identifiziert werden kann.

Kapitel 4

DLO-Orientierung

Um zwei Ende der Dichtung korrekt mit einander kleben zu können, müssen wir zuerst die Orientierung des Querschnittes von den Enden bestimmen. In dieser Arbeit werden Template-Matching-Methoden benutzt um dieses Problem zu lösen. Drei Methoden wurden implementiert um die Effizienz von jeder Methode mit einander vergleichen zu können. Der Brute-Force-Algorithmus benutzt das Refermenzbild T und macht eine Reihe von normalen Template-Matching-Prozessen, um das aufgenommene Bild I zu analysieren. Das Referenzbild T muss zuerst mit jedem Skalierungsfaktor skaliert und in jeder Orientierung rotiert werden. Die skalierten rotierten Bilder werden dann auf das ganze Bild I gesucht. Der Brute-Force liefert höchstwahrscheinlich das beste Ergebnis, ist aber sehr rechenintensiv. Der neue Algorithmus, der als Ciratefi genannt wurde, liefert ein ähnliches Ergebnis wie das von dem Brute-Force-Algorithmus aber mit deutlich kürzerer Laufzeit. Die dritte Methode benutzt einen anderen Ansatz als der von zwei vorherigen Algorithmen. Die Feature-Punkte vom Referenzbild T und vom aktuellen Bild I werden zuerst extrahiert. Diese Feature-Punkte sollen invariant zu Skalierung, Rotation, Translation sein. Die korrespondierenden Punkte zwischen zwei Punktlisten werden gesucht. Die relative Position und Orientierung von dem Objekt werden schließlich aus diesen korrespondierenden Punkten berechnet. Im nächsten Abschnitt werden zuerst allgemeine Informationen über Template-Matching vorgestellt. Danach werden drei Methoden ausführlich erklärt.

4.1 Template Matching

Eine einfache Definition von Template Matching ist wie folgende: Gegeben ein Bild S und ein Bild T, wo die Dimension von S größer als T ist. Die Ausgabe von Template-Matching-Prozess ist, ob S ein Bild I enthält, indem I und T entsprechend ähnlich sind und wenn es das Bild I gibt, wird die Position von I in S gezeigt [Tak03]. Template Matching ist ein wichtiges Thema in der Forschung seit den 80er Jahren. Seitdem haben die Forscher verschiedene Methoden vorgschlagen, die eine Vielzahl von Anforderungen aus Industrie erfüllen sollen. Das Hauptproblem bei der Template-

Matching ist die verbleibende Toleranz gegenüber verschiedenen Verzerrungen, die bei Bildaufnahme auftreten können. Solche Verzerrungen sind z. B. Rotation, Veränderungen in Größe, lineare und nicht-lineare Veränderungen der Helligkeit, perspektivische Verzerrungen und Rauschen. Abgesehen von der Genauigkeit und Präzision, ist Effizienz auch ein wichtiges Faktor beim Aufbau des vollständigen Algorithmus für Template-Matching. Für echtzeitfähige Algorithmen dürfen die Ausführungszeiten nur maximal 100ms sein.

Ein Template kann zusätzlich eine gewisse Variabilität aufweisen: nicht alle ihre In-

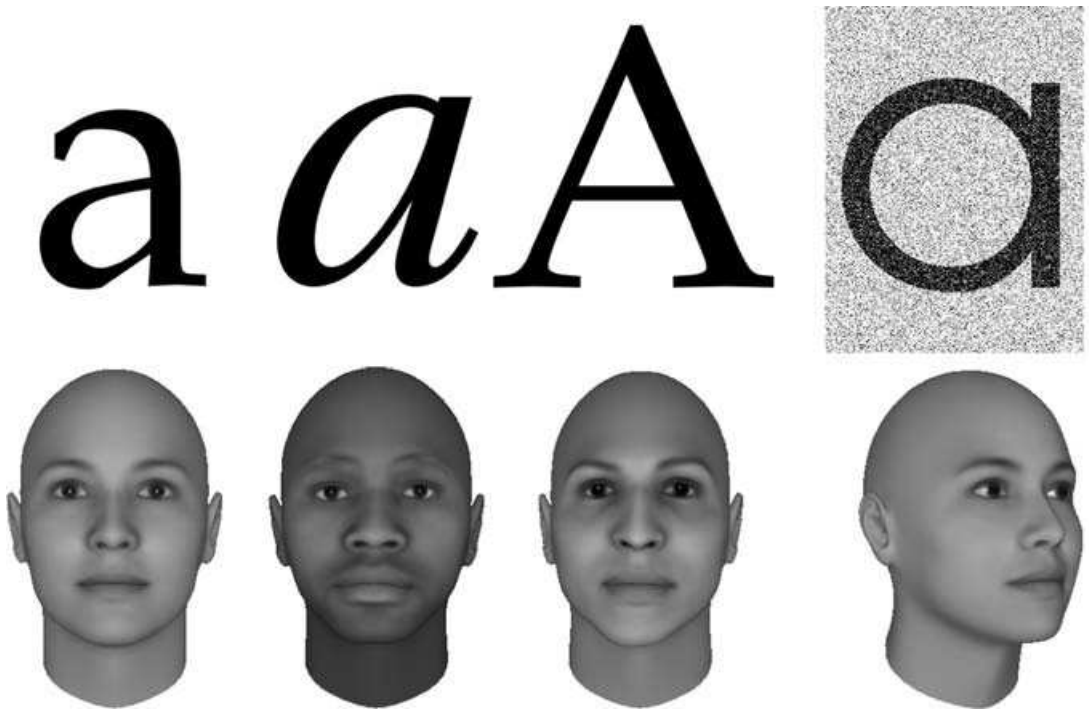


Abbildung 4.1: Einige Templates für zwei übliche Objektklassen: Buchstaben und Gesichte [Bru09]

stanzen sind exakt gleich (Siehe Abbildung 4.1). Ein wichtiges Beispiel für die Variabilität ist aufgrund der unterschiedlichen Aussichtspunkte, aus denen das Objekt beobachtet wird. Änderungen der Beleuchtung, Bildsensor oder Sensor-Konfiguration können auch zu erheblichen Schwankungen führen. Eine weitere Form der Variabilität ergibt sich aus intrinsischen Änderungen der physischen Objekt-Instanzen, die die Änderung des entsprechenden Erscheinungsbilds des Objekts verursachen. Man betrachtet z.B viele Variationen von Gesichtern, die alle einem gemeinsamen Grundstruktur behalten, aber auch deutliche Unterschiede aufweisen. Eine weitere wichtige Quelle der Variabilität ergibt sich aus der zeitlichen Entwicklung eines einzelnen Objekts. Ein interessantes Beispiel dafür ist der Mund beim Sprechen.

Matching-Techniken lassen sich grob in zwei große Gruppen: als Feature-basiertes Matching (feature-based matching (FBM)) und Bereich-basiertes Matching (area-

based matching (ABM)) zugeordnet werden.
FBM extrahiert spezifische Merkmale aus dem Template und speichert dies als Informationen des Template-Bilds. Daher hat FBM eine hohe Toleranz gegenüber Rotation, Größenvariation, und perspektivische Verzerrungen, während sie eine faire Toleranz gegenüber Veränderungen in der Beleuchtung und der geringen Rauschen ergibt.
ABM, auf der anderen Seite, speichert das Template-Bild als eine geordnete Menge von Pixeln mit unterschiedlichen Grauwerten als Template-Informationen. Eine Korrelationsmethode oder Methode der kleinsten Quadrate wird häufig verwendet, um die Daten des Templates in allen möglichen Position im Suchbild zu vergleichen. ABM ist deshalb sehr rechenintensiv, und schwach gegen Verdrehen, Größenvariation und perspektivische Verzerrungen. Seine Stärke sind aber, wenn die Korrelationsmethode verwendet wird, wird ABM völlig unempfindlich gegen lineare Veränderungen in der Helligkeit. Die Toleranz gegenüber Rauschen ist besser im Vergleich zu FBM.
Die meisten Template-Matching-Methoden bestehen aus vier folgenden Schritte

- **Merkmale-Erkennung:** Hervortretende und unverwechselbare Objekte (geschlossene Regionen, Kanten, Konturen, Linien-Kreuzungen, Ecken, etc.) sind manuell oder, vorzugsweise, automatisch erkannt. Zur weiteren Verarbeitung können diese Funktionen durch ihre Kontrollpunkte repräsentiert werden (Schwerpunkte, Zeilenenden, markante Punkte).

- **Merkmale-Matching:** In diesem Schritt wird die Übereinstimmung zwischen den Merkmalen vom aktuellen Bild und vom Referenzbild gesucht. Verschiedene Deskriptoren für Merkmale und Ähnlichkeitsmaß können für diesen Zweck verwendet werden.

- **Schätzung des Transform-Model:** Die Art und die Parameter der so genannte Mapping-Funktionen, die das aktuelle Bild zu dem Referenzbild abbildet, wird geschätzt. Die Parameter der Mapping-Funktionen sind durch die Korrespondenz der Merkmale berechnet

- **Resampling und Transformation:** Das aktuelle Bild wird durch die Mapping-Funktionen transformiert. Werte in nicht-ganzzahligen Koordinaten werden durch entsprechende Interpolationstechnik berechnet.

In den nächsten Abschnitten werden zwei Ansätze ABM und FBM ausführlich diskutiert. Die Implementierung jedes Schrittes hat eigene typischen Probleme und soll entsprechend je nach den spezifischen Aufgabe ausgewählt werden. Die Leser können die Details in der Diskussion [Flu03] ausführlicher lesen.
In den nächsten Abschnitten werden zwei Ansätze ABM und FBM ausführlich diskutiert.

4.1.1 Region-basierte Methode

Region-basierte Methoden verschmelzen die zwei Schritte: Merkmale-Erkennung und Merkmale-Matching. Diese Methoden bearbeiten die Bilder ohne hervortretende Merkmale zu finden. Die Beschränkung von ABM stammt aus ihrer Grundidee. Zuerst ist das rechteckige Fenster, das am häufigsten verwendet wird, nur geeignet für die Bilder, die nur translatorische Unterschiede ausweisen. Wenn Bilder durch mehr komplexere Transformationen (Rotation, Projektion) deformiert sind, ist diese Art vom Fenster nicht in der Lage, die gleichen Teile der Szene im Referenzbild und im aktuellen Bild decken zu können.
Ein weiterer Nachteil der ABM bezieht sich auf die „Auffälligkeit“ des Fensterinhaltes . Es besteht eine hohe Wahrscheinlichkeit, dass ein Fenster mit einer glatten Fläche ohne prominente Details fälschlicherweise mit anderen glatten Fläche im Referenzbild übereingestimmt wird. Die Merkmale für die Template-Matching sollte vorzugsweise in charakteristischen Teile des Bildes detektiert werden.
Klassische Bereich-basierte Methoden wie Kreuzkorrelation benutzen direkt die Intensitäten und benötigen keine strukturelle Analyse. Daher sind sie empfindlich auf Veränderungen der Intensität, zum Beispiel durch Rauschen, unterschiedliche Beleuchtung, und durch den Einsatz verschiedener Sensortypen.
Gegeben sind ein Referenzbild T und ein zu analysierendes Bild I. Das Referenzbild T wird auf dem Bild I verschoben und der überlappende Bereich von zwei Bildern werden verglichen. Oft kommen Kreuzkorrelation oder Methode der kleinsten Quadrate zum Einsatz. Einige Varianten von diesen Ansätzen werden hier gelistet:

- Kleinste der Quadraten:

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2 \tag{4.1}$$

- Normalisierte Kleinste der Quadraten:

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \sum_{x',y'} \cdot I(x+x',y+y')^2}} \tag{4.2}$$

- Kreuzkorrelation

$$R(x,y) = \sum_{x',y'} (T(x',y') + I(x+x',y+y')) \tag{4.3}$$

- Normalsierte Kreuzkorrelation

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \sum_{x',y'} \cdot I(x+x',y+y')^2}} \tag{4.4}$$

- Kreuzkorrelation-Koeffiziente

$$R(x,y) = \tag{4.5}$$

wobei

$$T'(x',y') = T(x',y') - \frac{1}{w \cdot h} \cdot \sum_{x'',y''} T(x'',y'') \tag{4.6}$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{1}{w \cdot h} \cdot \sum_{x'',y''} I(x+x'',y+y'') \tag{4.7}$$

- Normalisierte Kreuzkorrelation-Koeffiziente

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \sum_{x',y'} \cdot I(x+x',y+y')^2}} \tag{4.8}$$

wobei (x',y') Position des Pixels auf dem Referenzbild und (x,y) Position des Pixels auf dem aufgenommenen Bild ist.
Dieses Ähnlichkeitsmaß wird für das Fenster-Paar aus dem aktuellen Bild und dem Referenzbild berechnet und sein Maximum (Kreuzkorrelation, normalisierte Kreuzkorrelation) oder Minimum (Kleinste der Quadraten) werden durchgesucht. Das Fenster-Paar, bei dem das Maximum/Minimum erreicht wird, wird als Korrespondenz festgelegt. Obwohl Kreuzkorrelation-basierte Methoden nur die Translation behandeln können, beeinflüssen kleine Orientationsänderung und Skalierung die Ergebnisse auch nicht erheblich. Die Ergebnisse gelten daher als akzeptabel.
Zwei wesentliche Nachteile der Korrelation-basierten Methoden sind die Flachheit der Ähnlichkeitsmessung von Maxima (aufgrund der Selbstähnlichkeit der Bilder) und hohe Komplexität. Die Maxima können durch Vorverarbeitung oder die Benutzung von Vektor-Korrelation geschärft werden. Pratt et al. [Pra74] hat einen Filter vor dem Template-Matching-Prozess eingesetzt, um die Performanz der Korrelation zu verbessern. Van Wie et al. [vWMS77] und Anuta et al. [Anu70] haben Kante-basierte Korrelation anstatt der ursprünglichen Bilder selbst verwendet. Auf diese Weise ist das Verfahren weniger empfindlich auf die Intensitätsunterschiede zwischen dem Referenzbild und dem aktuellen Bild.
Trotz der oben genannten Einschränkungen, sind die Korrelation-basierte Methoden auch heute noch oft im Einsatz, vor allem durch ihre einfache Implementierung. Mit der Entwicklung von Hardwarekomponenten können diese Methode auch in Echtzeitanwendungen benutzt werden.

4.1.2 Merkmal-basierte Methode

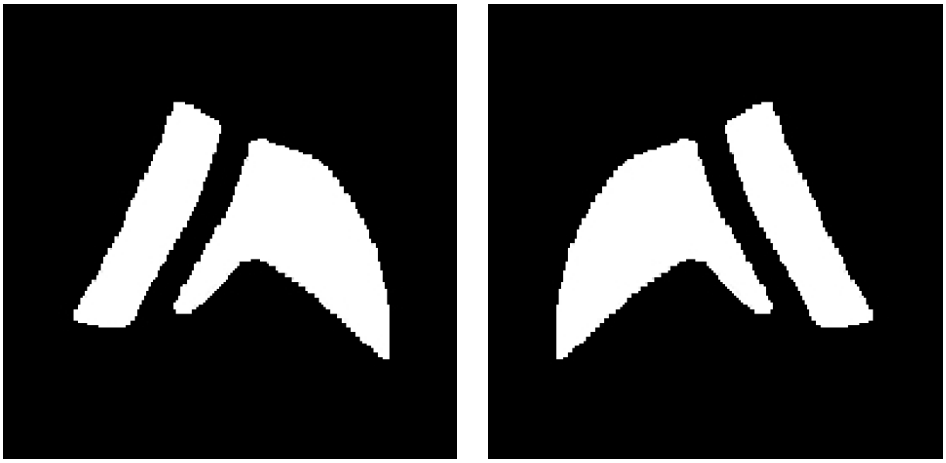
Wir gehen davon aus, dass zwei Gruppen von Merkmalen im Referenzbild und dem aktuellen Bild von den Kontrollpunkten repräsentiert werden. Das Ziel ist es, die paarweise Korrespondenz zwischen ihnen mit deren räumliche Beziehungen oder verschiedene Deskriptoren von Merkmalen zu finden.

Methoden, die in erster Linie auf die räumlichen Beziehungen zwischen den Eigenschaften basieren, werden in der Regel angewendet, wenn detektierte Merkmale mehrdeutig sind. Die Informationen über die Abstände zwischen den CPs und über ihre räumliche Verteilung werden genutzt. Einige Methoden können in [Sto85] (Graph Matching-Verfahren) oder [Ben82] (Clustering-Technik) gesehen werden.
Als Alternative zu den Methoden, die die räumlichen Beziehungen benutzen, kann die Korrespondenz der Merkmale mittels ihrer Deskriptoren, vorzugsweise invariant gegen die erwartete Bildverformung geschätzt werden (siehe Abbildung. 4). Die Deskriptoren sollen mehrere Bedingungen erfüllen. Die wichtigsten sind Invarianz (die Deskriptoren der entsprechenden Merkmale aus dem Referenzbild und dem aktuellen Bild müssen identisch sein), Einzigartigkeit (zwei verschiedene Merkmale sollen unterschiedlichen Deskriptoren haben), Stabilität (der Deskriptor eines Merkmals, der leicht verformt wird, soll möglichst nah am Deskriptor des ursprünglichen Merkmals liegen), und Unabhängigkeit (wenn der Deskriptor eines Merkmals ein Vektor ist, sollen deren Elemente funktional unabhängig sein). Allerdings können in der Regel nicht alle diese Bedingungen gleichzeitig erfüllt sein und es ist notwendig, einen geeigneten Kompromiss zu finden. Merkmale aus dem aktuellen Bild und dem Referenzbild mit den meisten ähnlichen invarianten Deskriptoren paaren sich zusammen und werden als Korrespondenz markiert. Die Wahl der Art der invarianten Deskriptoren hängt von den Eigenschaften der Merkmalen und der angenommenen geometrischen Verformungen der Bilder. Auf der Suche nach den besten passenden Merkmal-Paaren im Merkmale-Raum kommt oft der Mindestabstand (z. B. die Mahalanobis oder Eucliedean-Abstand) mit einem Schwellenwert zum Einsatz.
Eine breite Vielfalt von Detektoren und Deskriptoren wurden in der Literatur bereits vorgeschlagen. Außerdem wurden detaillierte Vergleiche und Auswertungen über das Benchmarking-Datensätzen durchgeführt ([Sch05], [vG05]). Der am häufigsten verwendete Detektor ist wahrscheinlich die Harris-Ecke-Detektor [Ste88], der bereits im Jahr 1988 vorgeschlagen wurde. Es basiert auf die Eigenwerte der zweiten Moment-Matrix. Allerdings sind Harris Ecken nicht Skala-invariant. Lindeberg et al. [Lin98] hat das Konzept einer automatischen Auswahl von Skala vorgestellt. Dies erlaubt es, jeden Interessenpunkte in einem Bild mit ihren eigenen, charakteristischen Skalen zu erkennen. Mikolajczyk und Schmid in [Sch01] verfeinerte diese Methode zu einem stabilen und Skala-invarianten Detektor mit hoher Wiederholung. Sie verwendeten die Determinante der Hesse-Matrix, um die Position auszuwählen, und Laplace-Operator, um die Skala zu berechnen. Mit der Konzentration auf die Geschwindigkeit hat Lowe in [Low99] eine Approximation des Laplace des Gaussians (Laplace of Gaussians-LOG) durch Differenz des Gaussians (Different of Gaussian-DOG)-Filter vorgeschlagen. Nach der Untersuchung der Literatur hat Bay in [Goo06] beschlossen, dass Hessian-basierte Detektoren stabiler und widerholbarer als Harris-basierte Detektoren. Darüber hinaus ist die Benutzung von Determinanten der Hesse-Matrizen, vorteilhaft gegenüber der Benutzung seiner Spur (der Laplace-Operator), da es die längliche, schlecht lokalisierte Strukturen vermeiden kann. Es wird auch festgestellt, dass Annäherungen durch DoG mehr Geschwindigkeit zu einem günstigen Kosten

in Form von verlorener Genauigkeit bringen können. Eine noch größere Vielfalt von Merkmal-Deskriptoren wurde vorgeschlagen. Es wurde gezeigt, dass der von Lowe et al. [Low99] eingeführte Deskriptor besser als die anderen ist [Sch05]. Dies lässt sich durch die Tatsache erklären, dass der Deskriptor eine beträchtliche Menge von Informationen über den räumlichen Intensität-Muster erfasst, während er gleichzeitig robust gegen kleine Verformungen oder Lokalisierungsfehler ist. Der Deskriptor in [Low04], der als SIFT genannt wurde, berechnet ein Histogramm der lokalen orientierten Gradienten um dem Interessenpunkt und speichert die Bins in einem 128-dimensionalen Vektor (8 Orientierung-Bins für jeweils 4×4 Position-Bins).
Der SIFT-Deskriptor scheint immer noch der beste Deskriptor für praktische Anwendungen und damit auch ist bisher der am meisten verwendete Deskriptor. Er ist ziemlich schnell, die entscheidend für Online-Anwendungen ist. Auf der anderen Seite ist die hohe Dimensionalität des Deskriptors ein Nachteil von SIFT im Matching-Schritt.
SURF (Speeded-Up Robust Features) wurde vor kurzem von Bay et al. [Goo06] vorgestellt. Wie SIFT, enthält der SURF-Ansatz einen Schlüsselpunkt-Detektor und einen Deskriptor. Schlüsselpunkte werden mit Hilfe eines so genannten beschleunigten Hesse-Detektors festgestellt, der sich auf einer Approximation der Hesse-Matrix für einen bestimmten Bildpunkt beruht. Die Antworten auf Haar-Wavelets werden zur Orientierungszuordnung verwendet, bevor der Deskriptor für den Schlüsselpunkt durch die Wavelet-Antworten in einer bestimmten Umgebung gebildet wird. SURF wurde später gezeigt, dass er ähnliche Performanz wie von SIFT aber mit weniger Laufzeit liefert. [Pro07].

4.2 Brute-Force-Algorithmus

Um ein Rotation-, Skalierung-, Translation-invariantes Template-Matching zu erhalten, muss das Referenzbild T in jedem Winkel gedreht und mit jedem Faktor skaliert werden. In der Praxis ist es nicht möglich, das Referenzbild T in jedem Winkel und mit dem Faktor zu dreken und zu skalieren, sondern nur in einer diskreten Menge von Winkeln und Skalen. Abbildung 4.3 zeigt einige der Templates, die in $m = 36$ verschiedenen Winkeln ($\alpha_0 = 0, \alpha_1 = 10, ..., \alpha_{35} = 350$) gedreht und mit $n = 5$ verschiedene Faktoren ($s_0 = 0,9, s_1 = 0,95, ..., s_4 = 1.1$) skaliert werden. Ein Low-Pass-Filter (z. B. die Gauß-Filter) wird oft nach verwendet, um die beiden Bilder I und T zu glätten. Dieser Low-Pass-Filter verringert die Fehler, die durch Verwendung diskreter Skalen und Winkel eingeführt werden (siehe Abbildung 4.3).
Jedes Pixel p von I wird gegen alle verwandelte Vorlagen getestet. Der Brute-Force-Algorithmus gibt dann die genaue Skalierungsfaktor und Drehwinkel für jede Trefferzeile.



a) b)

Abbildung 4.2: Die Referenzbilder für zwei Ende des Objekts im Brute-Force-Algorithmus

Abbildung 4.3: Einige skalierte, rotierte Referenzbilder

4.3 Ciratefi-Algorithmus

Der Ciratefi-Algorithmus besteht aus drei kaskadierten Filtern. Jeder Filter schließt nacheinander die Pixel aus, die keine Möglichkeit haben, zur weiteren Bearbeitung zu passen. Der erste Filter, der als CIFI (circular sampling filter) genannt wird, nutzt die Projektion von Bildern I und T auf Kreise, um die Pixel von I in zwei Gruppen zu kategorieren: diejenigen, die keine Chance haben zum Refenzbild T zu passsen und diejenigen, die „vielleicht“ zum Referenzbild T passen. Dieser Filter ist für die Festlegung der Skalierungsfaktoren ohne Isolierung der Formen zuständig. Er bestimmt einen „wahrscheinlichen Skalierungsfaktor“ für jedes der erststufigen Kandidaten-Pixel. Der zweite Filter RAFI (radial sampling filter) nutzt die Projektion von Bildern I und T auf den radialen Linien und die von CIFI berechneten „wahrscheinlich Skalenfaktoren“, um einige der erststufigen Kandidaten-Pixel zu den zweistufigen Kandiaten-Pixeln zu aktualisieren. Auch weist er einen “wahrscheinlich Drehwinkel„ zu jedem zweitstufigen Kandidaten-Pixel zu. Die Pixel, die nicht aktualisiert werden, werden ausgeschlossen. Der dritte Filter mit der Bezeichnung Tefi (Template-Matching-Filter) ist ein konventionelles Bereich-basiertes Template-Matching. Die Anzahl der zweitstufigen Kandidaten-Pixel ist in der Regel geringer.

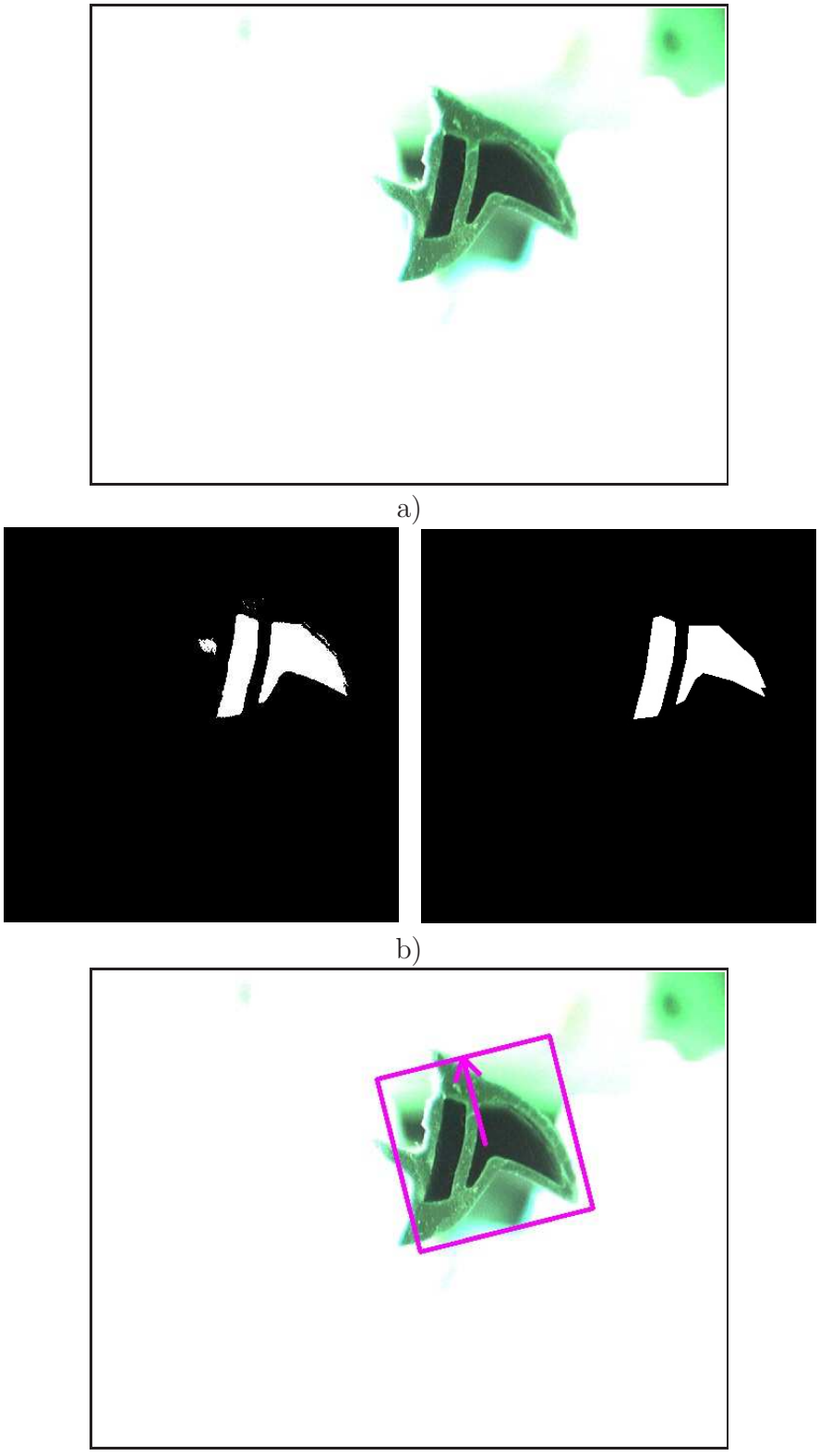


Abbildung 4.4: Der Brute-Force-Algorithmus: a) Das aufgenommene Bild. b) Das segmentierte Bild. c) Das Ergebnis von dem Brute-Force-Algorithmus

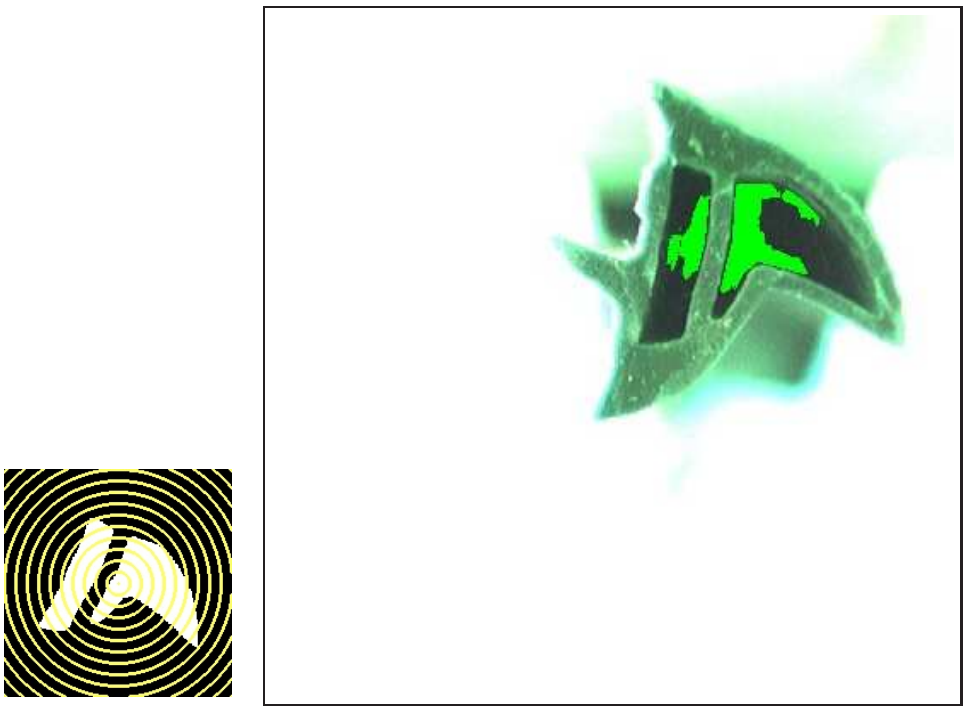
CIFI und RAFI haben außerdem bereits ihre wahrscheinlichen Skalierungsfaktoren und Drehwinkel berechnet. Deshalb kann das Template-Matching schnell die richtige Pixel aus allen zweitstufigen Kandidaten-Pixel aussuchen.

4.3.1 Zircular-Sampling-Filter

CIFI nutzt die Projektionen der Bilder I und T auf einer Reihe von Kreisen (siehe Abbildung 4.5-a) um die erststufigen Kandidaten-Pixel und ihre wahrscheinlichen Skalen zu erkennen. Wie es in [dA07] gezeigt wird, ist die richtige Wahl der Anzahl der Kreise l nicht unbedingt wichtig, denn RAFI und TEFI filtern die erststufigen Kandidaten-Pixel weiter aus. Abbildung 4.5-b zeigt die Ausgabe von CIFI, wo die erststufige Kandidaten-Pixel in magenta dargestellt werden.
Gegeben ein Bild B. Wir definieren das Zirkular-Sampling $Cis_B(x, y, r)$ als den durchschnittlichen Grauwert der Bildpunkte B im Abstand r von dem Pixel in (x, y):

$$Cis_B(x, y, r) = \frac{1}{2\pi r} \int_0^{2\pi} B(x + rcos\theta, y + rsin\theta)d\theta \tag{4.9}$$

In der Praxis muss eine Summe das Integral ersetzen. Und ein Algorithmus zum Ze-



a) b)

Abbildung 4.5: a) Die Kreise, mit denen das Referenzbild abgetastet wird. b) Das Ergebnis von CIFI mit den erststufigen Punkten in grün

ichnen von Kreisen wie in [Bre77] muss auch genutzt werden, um effizient alle Pixel, die zu einem bestimmten Kreis gehören, zu bestimmen. Gegeben ein Referenzbild T und die Menge der n Skalen (in unserem Beispiel ist $s_0 = 0,9$; $s_1 = 1,0$; $s_2 = 1,1$). Düas Referenzbild T wird um Faktor s_i skaliert. Nach diesem Schritt erhalten wir eine Menge der skalierten Bilder T_0, T_1, ..., T_{n-1}. Danach wird jedes Bild T_i durch eine Reihe von vordefinierten l Kreisen (in unserem Beispiel, $l = 13$, und $r_0 = 0$, $r_1 = 2$, ..., $r_12 = 24$ Punkte) abgetastet. Das Ergebnis von diesem Schritt ist eine 2-D Matrix der Skalen-, Rotation-invarianten Merkmalen C_T mit n Zeilen (Skalen) und l Spalten (Radius):

$$C_T[i,k] = Cis_{T_i}(x_0, y_0, r_k),\ 0 \leq i \leq n \text{ und } 0 \leq k < l \tag{4.10}$$

wobei (x_0, y_0) das Zentrum-Pixel von T ist.
Gegeben ist ein Bild I, wir berechnen ein 3-D Bild $C_I[x,y,k]$:

$$C_I[x,y,k] = Cis_T(x,y,r_k),\ 0 \leq k < l \text{ und } (x,y) \in I \tag{4.11}$$

CIFI benutzt die Matrizen C_T und C_I um die Zirkular-Sampling-Korrelation $CisCorr$ für beste Übereinstimmung von Skalen für jedes Pixel (x,y) zu bestimmen:

$$CisCorr_{I,T}(x,y) = \underset{i=0..n-1}{\text{MAX}}[|Corr(C_T[i], C_I[x,y]|] \tag{4.12}$$

Ein Pixel (x,y) wird als ein erststufiges Pixel klassifiziert, wenn $CisCorr_{I,T}(x,y) >= t_1$ für einen Schwellenwert t_1. Wie es in [dA07] gezeigt wird, ist die Wahl von t_1 nicht kritisch. t_1 muss nur klein genug sein, damit die richtigen Pixel nicht ausgeschlossen werden. Die „wahrscheinliche Skalen“ von dem erststufigen Pixel (x,y) ist die am besten übereinstimmenden Skalen:

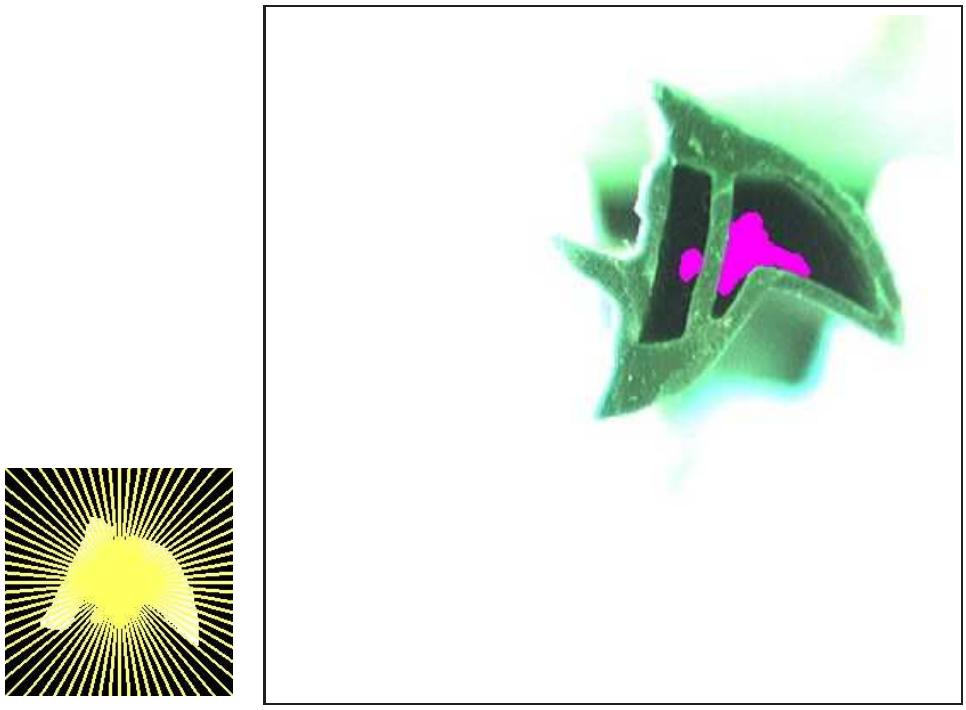
$$CisPS_{I_T}(x,y) = \underset{i=0..n-1}{\text{ARGMAX}}[|Corr(C_T[i], C_I[x,y]|] \tag{4.13}$$

4.3.2 Radial-Sampling-filter

Der zweite Filter wird als Radial-Sampling-Filter (RAFI) genannt und nutzt die Projektion von Bildern I und T auf eine Reihe von radialen Linien um einige der erststufigen Kandidaten-Pixel zu den zweitenstufigen Kandidaten-Pixel zu aktualisieren. Die Pixel, die nicht aktualisiert werden, werden ausgeschlossen. Auch weist er ein „wahrscheinlich Drehwinkel“ jedem zweistufigen Kandidaten-Pixel. In Abbildung 4.6-a werden die radiale Linien gelb gezeichnet und in Abbildung 4.6-b die zweistufigen Pixel werden pink markiert. Die Anzahl der radiale Linien muss gleich die Anzahl m der gewählten Drehwinkel sein (in unserem Beispiel, $\alpha_0 = 0, \alpha_1 = 10, ..., \alpha_{35} = 350$).
Gegeben ist ein Bild B. Wir definieren die Radial-Sampling $Ras_B^\lambda(x,y,\alpha)$ als den durchschnittlichen Grauwert der Pixel von B auf der radialen Linie mit einem Vertex in (x,y), Länge λ und Inkrement α:

$$Ras_B^\lambda(x,y,\alpha) = \frac{1}{\lambda}\int_0^\lambda B(x + tcos\alpha, y + tsin\alpha)dt \tag{4.14}$$

In der Praxis kann das Integral durch eine Summe ersetzt werden. Und ein Linie-



a) b)

Abbildung 4.6: a) Die Linien, mit denen das Referenzbild abgetastet wird. b) Das Ergebnis von RAFI mit den zweistufigen Punkten in pink

Zeichnung-Algorithmus (wie in [Bre65]) kann auch verwendet werden, um effizient alle Pixel, die zu einer Linie gehören, zu finden.
Gegeben ist das Referenzbild T und eine Menge von m Winkeln $(\alpha_0, \alpha_1, ..., \alpha_{m-1})$, T wird mit $\lambda = r_{l-1}$ (der größte Radius in CIFI) radial abgetastet. Das Ergebnis von diesem Schritt ist ein Vektor R_T mit m Elementen:

$$R_T[j] = Ras_T^{r_{l-1}}(x_0, y_0, \alpha_j),\ 0 \leq j < m \tag{4.15}$$

wobei (x_0, y_0) der Zentrumpunkt von T ist. Für jedes erststufige Kandidaten-Pixel (x, y) von I wird es mit dessen „wahrscheinlichen Skalen“ $i = CisPS_{I,T}(x, y)$ abgtastet. Der größte Radius r_{l-1} wird um Faktor s_i skaliert und wird $\lambda = s_i r_{l-1}$. Deshalb:

$$\begin{gathered} R_I[x, y, j] = Ras_T^{s_i r_{l-1}}(x, y, \alpha_j), \\ 0 \leq j < m \text{ und } (x, y) \text{ ist erststufiges Kandidaten-Pixel} \end{gathered} \tag{4.16}$$

An jedem erststufigen Kandidaten-Pixel (x, y) verwendet RAFI die Vektoren $R_I[x, y]$ und R_T um die Radial-Sampling-Korrelation RasCorr mit dem am besten passenden Winkel zu berechnen:

$$\begin{gathered} RasCorr_{I_T}(x, y) = \mathrm{MAX}_{j=0..m-1}[|Corr(R_I[x, y], cshift_j(R_T)|], \\ (x, y) \text{ ist erststufiges Kandidaten-Pixel} \end{gathered} \tag{4.17}$$

wobei „$cshift_j(R_T)$“ zirkulare Verschiebung um j Stellen der Elementen von R_T Vektor bedeutet. Ein erststufiger Pixel (x, y) von I wird dann zur zweiten Stufe aktulallisiert:

$$RasCorr_{I,T}(x, y) \geq t_2 \tag{4.18}$$

Nochmal wurde es in [dA07] gezeigt, dass die Wahl von t_2 unkritisch ist, sofern es nicht zu klein ist, damit die richtigen Pixel zur zweiten Stufe aktualisiert werden können. Der „wahrscheinliche Drehwinkel“ an dem zweitstufigen Pixel (x, y) ist:

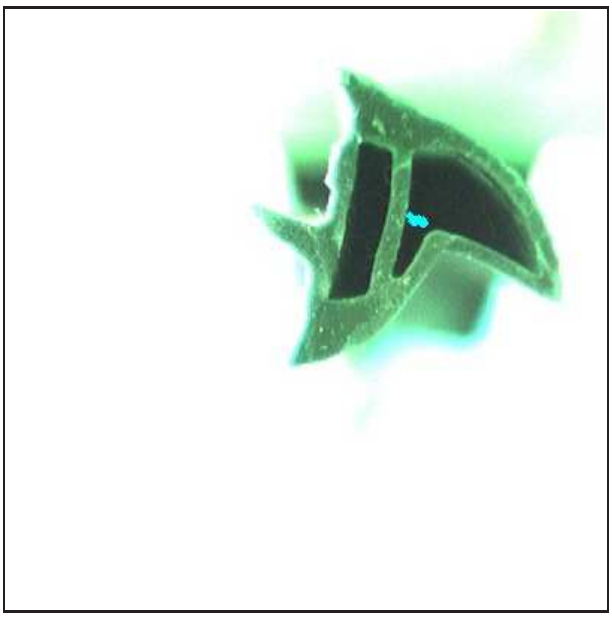
$$\begin{gathered} RasAng_{I,T}(x, y) = \text{ARGMAX}_{j=0..m-1}[|Corr(R_I[x, y], cshift_j(R_T)|], \\ (x, y) \text{ ist erststufiges Kandidaten-Pixel} \end{gathered} \tag{4.19}$$

4.3.3 Template-Matching-Filter

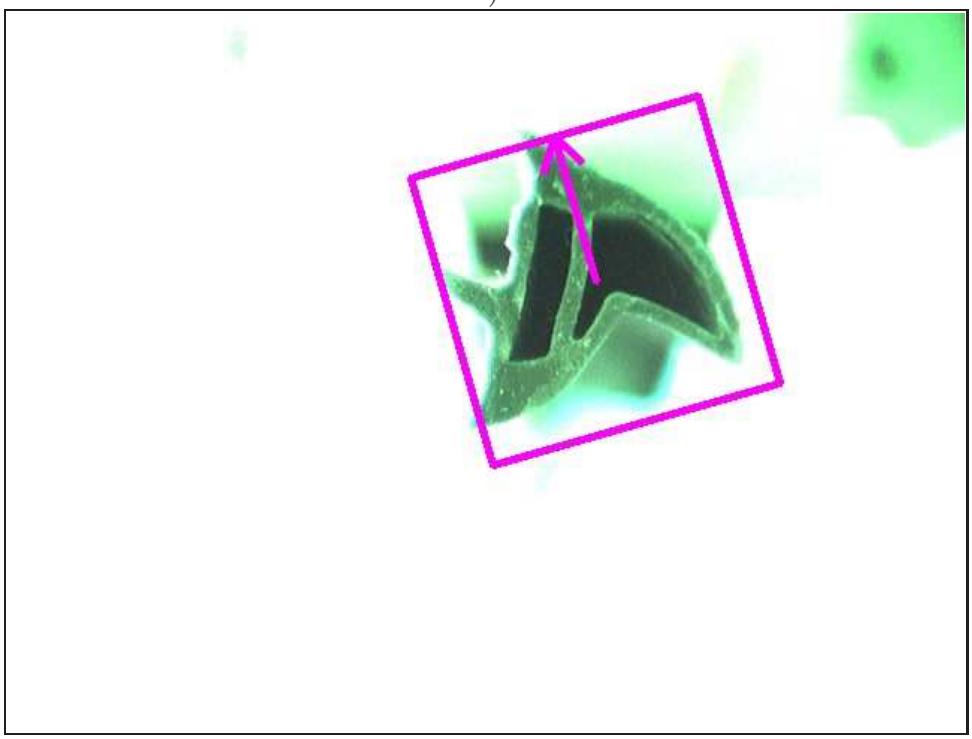
Der dritte Filter wird als TEFI (Template-Matching-Filter) genannt und es ist einfach das Region-basierte Template-Matching, das nur an den zweitstufigen Kandidaten-Pixel angewendet wird, wobei die voraussichtlich durch CIFI und RAFI berechneten Skalierungsfaktoren und Drehwinkeln benutzt werden sollen. Abbildung 4.7-a zeigt die Ausgabe von TEFI.
Zuerst skaliert TEFI das Referenzbild mit allen n Skalierungsfaktoren und dreht die skalierten Bilder wieder auf allen m Winkeln. Sei (x, y) eine zweitstufiges Kandidaten-Pixel, mit seiner wahrscheinlichen Skalen $i = CisPS_{I,T}(x, y)$ und wahrscheinlichem Winkel $j = RasAng_{I,T}(x, y)$. TEFI berechnet die Korrelation zwischen dem um Faktor s_i skalierten und um einen Winkel α_j gedrehten Referenzbild $T_{i,j}$ und dem Bild I am Pixel (x, y). Ist der absolute Wert der Korrelation größer als ein Schwellenwert t_3, wird das Template am Pixel (x, y) als „gefunden“ markiert. Wie in [dA07] gezeigt wird, ist die Komplexität von Ciratefi $O(NM)$, während die Komplexität der Brute-Force-Algorithmus $O(NM^2)$ ist , wobei N die Anzahl der Pixel des Bildes I und M die Anzahl der Pixel des Referenzbildes T ist. Mit dieser Analyse kann man leicht sehen, dass Ciratefi schneller als Burte-Force-Algorithmus laufen soll.

4.4 SURF-basierter Algorithmus

Der SURF-Algorithmus von Bay et al. [Goo06] ist ein Algorithmus zum effizienten Detektieren von Interessenpunkten in einem Bild und zur Berechnung eines Deskriptors zu jedem dieser Interessenpunkte. Die Position des Interessenpunktes und dessen SURF-Deskriptor bilden ein SURF-Merkmal . Der Deskriptor ist dabei für jedes SURF-Merkmal einzigartig, aber dennoch auch invariant gegenüber Skalierung, Rotation, Beleuchtungsänderung, Bildrauschen und bis zu einem gewissen Grad auch gegen perspektivische Verzerrung. Die Funktionsweise von SURF ist angelehnt an den SIFT-Algorithmus, ist in der Berechnung jedoch deutlich schneller. Ein Großteil des Geschwindigkeitsvorteils von SURF ist auf die Verwendung von Integralbildern



a)



b)

Abbildung 4.7: a) Das Ergebnis von TEFI, die gefundenen Punkte sind blaugrün markiert. b) Das Ergebnis vom Ciratefi-Algorithmus.

[VJ01] zurückzuführen. Im Folgenden werden die einzelnen Elemente des SURF-Detektors näher beschrieben. Die Berechnung des SURF-Deskriptors und das Vergleichen von SURF-Merkmalen ist nicht Gegenstand dieser Arbeit und wird hier daher nicht näher betrachtet.

4.4.1 Integralbilder

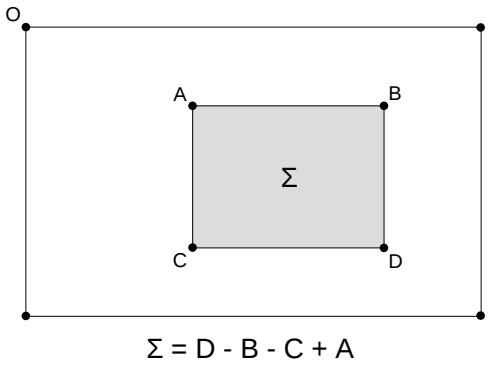


Abbildung 4.8: Berechnung der Summe aller Pixelwerte im grau markiertem Rechteck mit Hilfe eines Integralbildes [Goo06]

In einem Vorverarbeitungsschritt wird aus dem Eingabebild ein Integralbild [VJ01] berechnet. In einem Integralbild ist jeder Pixelwert die Summe aller Grauwerte zwischen dem korrespondierendem Pixel im Eingabebild und dem Ursprung. Seien ein Bild I und die Pixelkoordinate (x, y) gegeben. Der Wert im Integralbild I an der Position (x, y) ist dann formal definiert durch:

$$I_\Sigma(x, y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j). \tag{4.20}$$

Die Summe der Pixelwerte eines beliebigen achsenparallelen Rechtecks in Bild I lässt sich dann mit lediglich drei Additionen und vier Speicherzugriffen auf I berechnen. Dieses Vorgehen ist schematisch in Abbildung 4.8 dargestellt. Formal ist der Zusammenhang zwischen einer Rechtecksumme in einem Bild und dessen Integralbild wie folgt definiert (Variablennamen wie in Abbildung 4.8):

$$\Sigma = \sum_{i=A_x}^{i \leq D_x} \sum_{j=A_y}^{j \leq D_y} I(i, j). = I_\Sigma(D) - I_\Sigma(B) - I_\Sigma(C) + I_\Sigma(A) \tag{4.21}$$

4.4.2 Beschleunigter Hesse-Detektor

Zur Detektion von Interessenpunkten wird im Bild nach Positionen gesucht an denen die Determinante der Hesse-Matrix ein lokales Maximum erreicht. Um die

Elemente der Hesse-Matrix an einer Position (x, y) im Eingabebild zu berechnen, wird es bei SURF mit Marr-Hildreth-Operatoren, besser bekannt als Laplacian of Gaussians (LoG), für die x-, y- und xy-Richtung gefalten. Diese Operatoren werden durch Anwenden des Laplace-Operators auf eine Gaußfunktion berechnet, sind also im Grunde partielle Ableitungen zweiter Ordnung von dieser. Durch die Verwendung der Gaußfunktion kann mittels der Varianz die Glättung des Eingabebildes beeinflusst und somit die Determinante auf verschiedenen Skalierungsebenen des Bildes berechnet werden. Auf diese Weise wird die Skalierungsinvarianz eines SURF-Features gewährleistet. Die Hesse-Matrix lässt sich nun als eine Funktion H der Position $x = (x, y)$ und der Skalierung (scale) σ definieren:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xx}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \tag{4.22}$$

wobei $L_{xx}(x, \sigma)$ die Faltung der zweiten partiellen Ableitung der Gaußfunktion

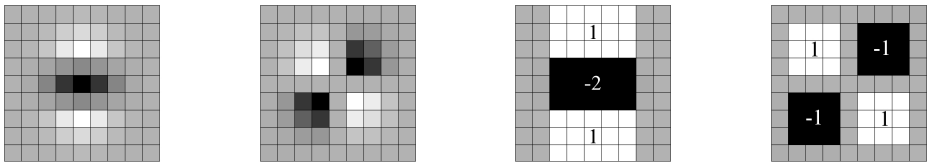


Abbildung 4.9: Von links nach rechts: Die diskretisierten Filtermasken für die zweiten partiellen Ableitung der Gaußfunktion in $y - (L_{yy})$ und xy-Richtung (L_{xy}) und deren in SURF angewandten Approximationen D_{yy} und D_{xy}. Die grauen Regionen entsprechen einer Gewichtung mit 0. [Goo06]

$\frac{\partial^2}{\partial x} g(\sigma)$ nach x mit dem Eingabebild an Position x ist. L_{yy} und L_{xy} sind analog dazu definiert. Ob sich im Bild an der Position x ein Extremum, also ein möglicher Interessenpunkt befindet, lässt sich an der Determinante der Hesse-Matrix ablesen:

$$det(H) = L_{xx} L_{yy} - L_{xy}^2 \tag{4.23}$$

Ist diese negativ, so handelt es sich um einen Sattelpunkt. Ist sie positiv, so wurde ein Extremum gefunden. Nähere Informationen über das Extremum liefert die Spur der Hesse-Matrix, welche mit dem Laplace-Operator korrespondiert:

$$tr(H) = L_{xx} + L_{yy} \tag{4.24}$$

Das Vorzeichen des Werts der Spur gibt Auskunft über den Kontrast des Extremums, also ob es sich um einen hellen Blob auf dunklem Hintergrund oder einen dunklen Blob auf hellem Hintergrund handelt. Für die Faltung mit einem Bild werde L_{xy}, L_{yy}, und L_{xy} in der Regel in Filtermasken diskretisiert. Bay et al. [Goo06] schlagen für *SURF* eine noch weitergehende Approximation durch die Mittelwertfilter D_{xx}, D_{yy} und D_{xy} vor, da diese schneller berechnet werden können. In Abbildung 4.9 sind

die diskretisierten LoG-Filter und ihre entsprechenden Approximationen dargestellt. Um die Summen der Filter unabhängig von der Filtergröße zu berechnen, werden sie bezüglich der Filterfläche normiert. Die Determinante der so approximierten Hesse-Matrix lässt sich durch folgende Formel berechnen:

$$det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2, \tag{4.25}$$

wobei w ein Gewicht ist, um den Unterschied der approximierten Filtermasken zu den ursprünglichen auszugleichen. Das Gewicht w variiert geringfügig abhängig von der Filtergröße, ist in SURF aber ein konstanter Wert ($w = 0,9$). Der Wert der Determinante wird als Blob an Position $x = (x, y, \sigma)^T$ bezeichnet.

4.4.3 Konstruktion des Skalenraumes

Interessenpunkte müssen, damit Skalierungsinvarianz gewährleistet werden kann, in verschiedenen Skalierungsebenen detektiert werden. In SIFT werden diese Skalierungsebenen in einer Skalenraum-Pyramide (scale-space pyramid) implementiert: Durch iteratives Gaußfiltern und Verkleinern des Eingabebildes wird eine Pyramide von verschiedenen Skalierungsebenen des Bildes erstellt. In SURF hingegen wird nicht das Bild verkleinert, sondern werden die Filtermasken zur Berechnung der Blob-Werte für jede Skalierungsebene entsprechend vergrößert. Da die approximierten LoG-Filter mit Hilfe des Integralbildes unabhängig von ihrer Größe in nahezu konstanter Zeit berechnet werden können, führt diese Methodik zu keinen erhöhten Rechenaufwand. Abbildung 4.10 veranschaulicht diese unterschiedlichen Vorgehensweisen. Der Skalenraum ist unterteilt in Oktaven. Eine Oktave repräsentiert eine Reihe

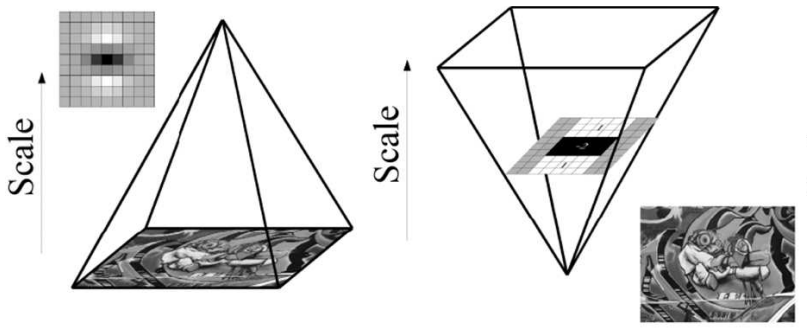


Abbildung 4.10: Anstatt das Eingabebild iterativ zu verkleinern (links), erlauben Integralbilder das Anwenden eines iterativ wachsenden Filters (rechts) auf das gleich bleibende Bild bei konstantem Aufwand [Goo06]

von Filterungen des Eingabebildes mit wachsender Filtergröße und umfasst einen Skalierungsfaktor von mindestens 2. Jede Oktave ist in eine konstante Anzahl von Skalierungsebenen, sogenannten Intervallen, unterteilt, die jeweils eine schrittweise Erhöhung der Skalierung repräsentieren.

Die 9×9 Filtermasken D_{xx}, D_{yy} und D_{xy} approxiieren die LoGs mit der Varianz $\sigma = 1,2$ und berechnen die Blob-Werte für das erste Intervall der ersten Oktave mit einer Skalierung von 1,2. Vergrößen dieser LoG-Filter vergrößert auch die Skalierung. Vorausgesetzt, dass das Filter-Layout im Verhältnis konstant bleibt, lässt sich die Skalierung aus der Filtergröße berechnen:

$$\sigma = \text{aktuelle Filtergröße.}\frac{\text{Bezugs-Filterskalierung}}{\text{Bezugs-Filtergröße}} = \text{aktuelle Filtergröße.}\frac{1,2}{9}. \tag{4.26}$$

Da die LoG-Filter diskretisiert sind und ein Zentrumspixel (d.h. eine ungerade

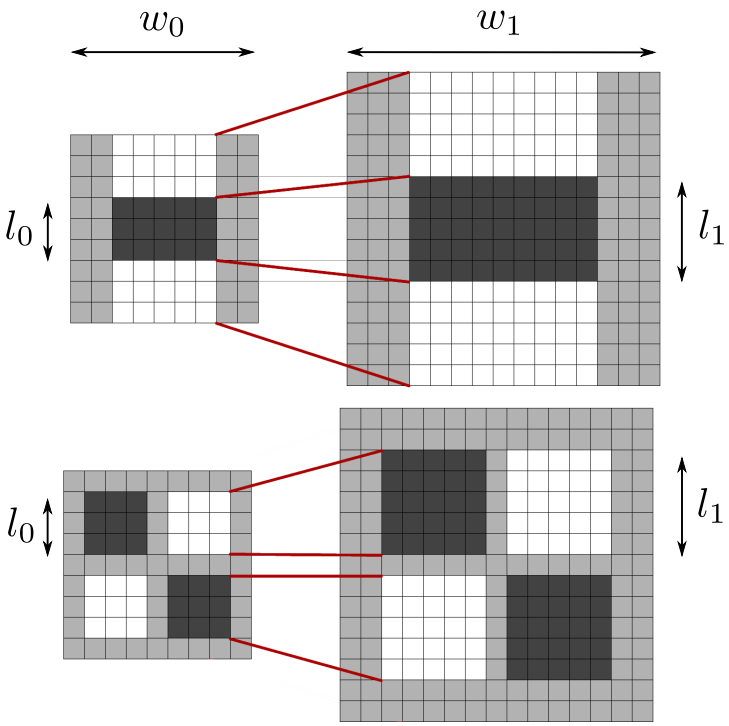


Abbildung 4.11: Um das Filter-Layout zu erhalten, dürfen die LoG-Filter jeweils nur um 6 Pixel oder mehr vergrößert werden [Goo06]

Maskengröße) haben müssen, lassen sie sich nur in eingeschränkten Schrittweiten vergrößern. Diese Schrittweite, und damit auch die möglichen Skalierungsschritte, sind abhängig von der Größe der Teilflächen1 der LoG-Filter (in Abbildung 4.9 schwarz oder weiß gekennzeichnet). Die jeweils kürzere Länge einer Teilfläche eines Filters beträgt stets ein Drittel der Filtergröße und wird als l bezeichnet. Verdeutlicht ist dies in Abbildung 4.11: Für die initialen LoG-Filter mit der Filtergröße $w_0 = 9$ (linke Seite der Abbildung), ist die Länge der Teilflächen $l_0 = 3$. Um das Zentrumspixel der schwarzen Teilfläche in D_{yy} (oben links) zu erhalten, muss l_0 um mindestens 2 Pixel vergrößert werden. Da sich in Dyy insgesamt drei dieser Teilflächen befinden, ergibt sich für w_0 eine Vergrößerung von 6 Pixeln auf $w_1 = 15$.

Daraus resultiert nach Gleichung 7 für das zweite Intervall der ersten Oktave eine Skalierung von $\sigma_1 = 2 = 15\frac{1,2}{9}$. Die Filtergrößen der nächsten Intervalle sind $w_2 = 21$ mit der Skalierung $\sigma_2 = 2,8$ und $w_3 = 27$ mit $\sigma_3 = 3,6$. Insgesamt sind das vier Intervalle pro Oktave, die in der ersten Oktave eine Skalierung von 1,2 bis 3,6 umfassen. Für die folgenden Oktaven wird der Vergrößerungsschritt der Filtergröße w

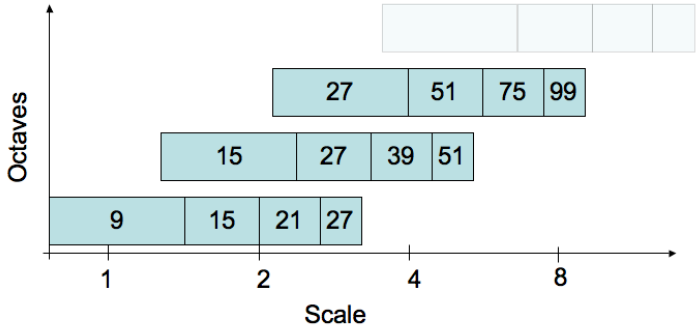


Abbildung 4.12: Die verschiedenen Filtergrößen in den einzelnen Oktaven. [Goo06]

jeweils verdoppelt. In der zweiten Oktave von 6 Pixel auf 12, in der dritten von 12 auf 24 und so weiter. Daraus ergeben sich die in Abbildung 4.12 dargestellten Filtergrößen. Zusätzlich wird mit jeder neuen Oktave die Samplingrate auf das Eingabebild halbiert bzw. der Samplingintervall verdoppelt.

4.4.4 Suche lokaler Extrema

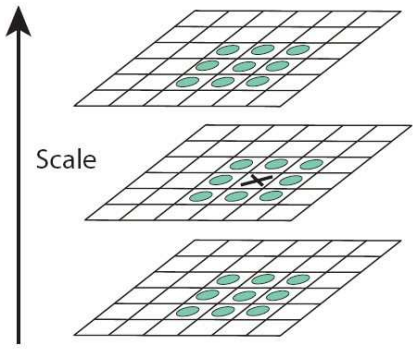


Abbildung 4.13: Die $3 \times 3 \times 3$ Nachbarschaft eines Blobs im Skalenraum, bestehend aus den acht räumlichen Nachbarn und den jeweils neun Nachbarn der nächstgrößeren und der nächstkleineren Skalierungsebene. [Goo06]

Es gilt nun innerhalb der Oktaven der Skalenraum-Pyramide Interessenpunkte zu finden. Als „interessant“ gelten lokale Extrema im Skalenraum. Dazu wird die $3 \times 3 \times 3$ Nachbarschaft jedes Blobs untersucht: Ist sein Blob-Wert betragsmäßig größer als alle seinen 26 Nachbar-Blobs, so ist er ein lokales Extremum. Ist sein Blob-Wert ebenfalls größer als ein vordefinierter Schwellwert, dann wird diese Skalenraum-Position $x = (x, y, \sigma)^T$ als ein Interessenpunkt vermerkt. Abbildung 4.13 veranschaulicht wie sich eine $3 \times 3 \times 3$ Nachbarschaft im Skalenraum zusammensetzt.

4.4.5 Präzise Ortsbestimmung der Interessenpunkte

Im finalen Schritt des SURF-Detektors werden die Positionen der gefundenen Interessenpunkte auf Subpixel-Genauigkeit interpoliert, was vor allem in höheren Skalierungsebenen nötig ist, um die Interessenpunkte präzise zu lokalisieren. Das dazu angewandte Verfahren in [Low04] besteht darin die Hesse-Matrix als ein Funktion $H(x)$ über Skalenraum-Koordinaten $x = (x, y, \sigma)^T$ durch eine Taylor-Reihe mit den Koordinaten x_e des detektieren Interessenpunktes als Entwicklungspunkt anzunähern:

$$H(x) = H(x_e) + (\frac{\partial H(x_e)}{\partial x})^T \Delta x + \frac{1}{2} \Delta x^T (\frac{\partial^2 H(x_e)}{\partial x^2}) \Delta x \tag{4.27}$$

wobei Δx der Abstand (offset) zwischen der ursprünglichen (diskreten) und der interpolierten Position des Interessenpunktes ist. Dieser Abstand lässt sich durch Ableiten dieser Funktion und Gleichsetzen mit 0 berechnen. Nach Umformung ergibt sich das lineare System

$$\Delta x = -(\frac{\partial^2 H(x_e)}{\partial x^2})^{-1} \frac{\partial H(x_e)}{\partial x} \tag{4.28}$$

aus dessen Lösungsvektor Δx sich mit

$$\hat{x} = x_e + \Delta x \tag{4.29}$$

die interpolierte Position $\hat{x}$ des Interessenpunktes bei x_e berechnen lässt. Die Ableitungen in den obigen Gleichungen werden als Differenzen zu den umgebenden Pixeln von (x_e) approximiert. Ist der Abstand x eines Interessenpunktes zu seiner interpolierten Position in einer der Dimensionen x, y oder nicht kleiner als 0,5, so wird er als zu kontrastarm (d. h. als zu empfindlich gegenüber Rauschen) verworfen. Die finale Ausgabe des SURF-Detektors ist eine Liste von Interessenpunkten mit subpixel-genauen Positionen im Eingabebild und einer zugeordneten Skalierung

4.4.6 Korrespondenz-Analyse

Bei der Korrespondenz-Suche kommt es darauf an äquivalente bzw. gleiche Merkmale in unterschiedlichen Frames oder zwischen dem aktuellem Frame und den gespeicherten Referenzen wiederzufinden. Auf Grund von Kamerarauschen oder unterschiedlichen Lichtverhältnissen haben dieselben Merkmale oftmals nicht denselben

Deskriptor und es kommt somit zu einer falschen Zuordnung der Merkmale. Nach dem die Korrespondenz-Punkte gefunden werden, können die Homographie zwischen dem Referenz-Frame und dem aktuellen Frame approximiert werden. Schließlich folgt die Homographie-Dekomposition um die relative Position und Orientierung von dem aktuellen Frame zu dem Template zu finden.

4.4.6.1 Korrespondenz-Punkte

Bevor eine Korrespondenz-Analyse durchgeführt werden kann, muss eine Zuweisung, also ein Matching der extrahierten Merkmale erfolgen. Ein merkmalgestütztes Matching vergleicht die Deskriptoren der jeweiligen Merkmale. Der Deskriptor eines jeden Merkmals besteht aus 64 Fließkommazahlen und die Ähnlichkeit der Features wird über den Abstand der jeweiligen Deskriptoren bestimmt. Da es unterschiedliche Abstandsmaße gibt, existieren auch mehrere Möglichkeiten die Merkmale zuzuordnen. In dieser Arbeit wird nur auf die „Nearest-Neighbour Ratio“-Methode näher eingegangen. Die „Nearest-Neighbour-Ratio“ Matching Methode basiert auf der Euk-

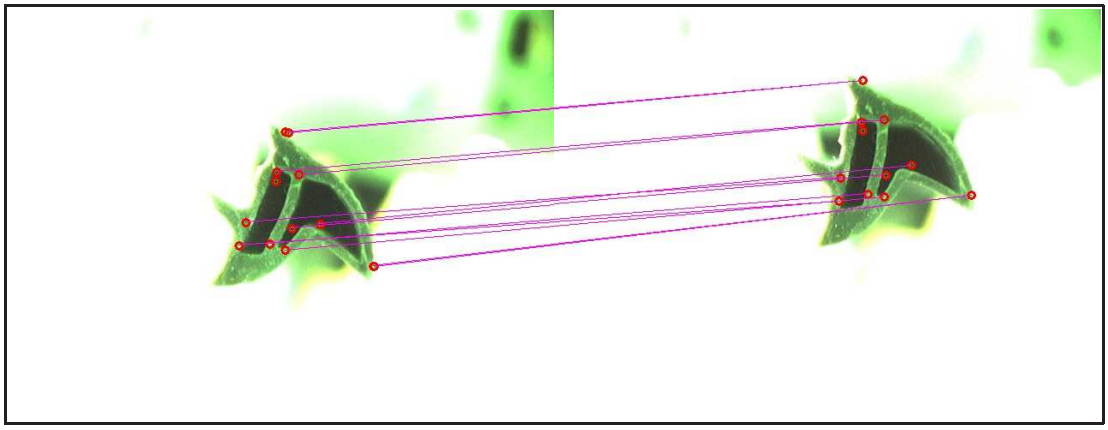


Abbildung 4.14: Die Korrespondenz-Punkte zwischen dem Referenzbild und dem aufgenommenen Bild.

lidischen Distanz zwischen zwei Eigenschaftsvektoren. Wobei hier nicht nur die zwei nächsten „Nachbarn“ innerhalb eines bestimmten Thresholds als positives Match gewertet werden (wie bei der „Nearest Neighbour“ Methode), sondern zusätzlich auch der zweit nächste „Nachbar“ berücksichtigt wird. Dies geschieht in der Form, dass zuerst der nächste und zweit nächste „Nachbar“ für den eben untersuchten Schlüsselpunkt gesucht werden. Daraufhin werden die Distanzen zwischen den Punkten berechnet. Liegt der Wert des Quotienten aus diesen Distanzen unterhalb eines Thresholds, wird Punkt zwei als positives Match gewertet. Mit Hilfe dieser Untersuchung wird sichergestellt, dass der zweit nächste „Nachbar“ zum untersuchten Punkt weit genug entfernt ist und somit der nächste Nachbar ein eindeutiges Match-Ergebnis darstellt. Folgende Formel stellt diesen Zusammenhang dar, wobei d_a den untersuchten Schlüsselpunkt, d_b den nächsten „Nachbarn“ und d_c den zweit nächsten

„Nachbarn“ darstellen.

$$\frac{|d_a - d_b|}{|d_a - d_c|} < r_{th} \tag{4.30}$$

Das Formelzeichen r_{th} steht für den Ratio-Threshold, welcher üblicherweise im Bereich von $r_{th} = 0,6 - 0,7$ liegt. In dieser Arbeit wird $r_{th} = 0.65$ gewählt. Auf diese Weise werden alle Eigenschaftsvektoren der Schlüsselpunkte des ersten Bildes mit den Eigenschaftsvektoren der Schlüsselpunkte aus dem zweiten Bild verglichen. Daraus ergibt sich gegebenenfalls eine Summe von positiven Matches, welche weiter ausgewertet werden können. Nähere Informationen zu den Matching Methoden bzw. Analysen bezüglich der Performance und Wiederholbarkeit können aus der Literatur wie [Sch05] entnommen werden.
Beim SURF Algorithmus gibt es noch eine zusätzliche Möglichkeit zur Auswertung der Matches. Neben der Euklidischen Distanz zwischen den Eigenschaftsvektoren kann auch noch das Vorzeichen der Spur der Hesse-Matrix ausgewertet werden. Dieser Wert steht durch die Berechnung der Blob-Response Map ohnehin zur Verfügung und benötigt somit keine extra Rechenzeit. Die Spur der Hesse-Matrix weist auf den Kontrast der Umgebung im Vergleich zum Schlüsselpunkt hin. (d. h.: Ob der Punkt in einer helleren Umgebung oder einer dunkleren Umgebung detektiert wurde.) Folgende Abbildung 4.15 verdeutlicht diesen Zusammenhang: Wie auch in

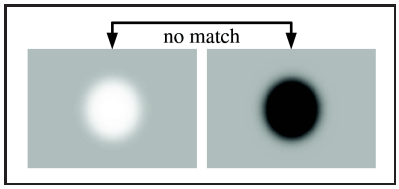


Abbildung 4.15: Kontrast von Schlüsselpunkten

[Goo06] beschrieben, können durch die Verwendung der Spur der Hesse-Matrix die Schlüsselpunkte beim Matching-Prozess bereits vor den eigentlichen Berechnungen ausgeschlossen werden, wodurch sich die Rechenzeit beim Matching verringert und die Anzahl der positiven Matches erhöht wird.

4.4.6.2 Homographie

Betrachten wir zwei Bilder der Punkte p auf einem 2D-Ebene P in 3D-Raum. Nehmen wir nun an, dass es zwei Bilder (x_1, x_2) für einen Punkt $p \in P$ gibt. Die Koordinaten-Transformation zwischen den beiden Frames ist

$$X_2 = RX_1 + T, \tag{4.31}$$

wobei X_1, X_2 entsprechend die relative Koordination von p zu Kamera-Frame 1 und 2 sind. Die zwei Bilder x_1 und x_2 von p müssen die Epipolar-Randbedingungen erfüllen

$$x_2^T E x_1 = x_2^T \hat{T} R x_1 = 0 \tag{4.32}$$

Für Punkte auf der gleichen Ebene P müssen ihre Bilder eine zusätzliche Bedingung erfüllen, sodass die Erfüllung der Epipolar-Randbedingungen allein nicht mehr ausreicht. Sei $N = [n1, n2, n2]^T \in S^2$ die Einheit-Normalenvektor der Ebene P mit Bezug auf die erste Kamera-Frame, und sei $d > 0$ der Abstand von der Ebene P zum optischen Mittelpunkt des ersten Kamera. Dann haben wir:

$$N^T X_1 = n_1 X + n_2 Y + n_3 Z = d \Leftrightarrow \frac{1}{d} N^T X_1 = 1, \forall X_1 \in P \tag{4.33}$$

Setze Gleichung 4.33 in Gleichung 4.31 ein:

$$X_2 = RX_1 + T = RX_1 + T\frac{1}{d}N^T X_1 = (R + \frac{1}{d}TN^T)X_1 \tag{4.34}$$

Wir definieren die Matrix:

$$H = R + \frac{1}{d}TN^T \in \Re^{3\times 3} \tag{4.35}$$

als (planare) Homographie-Matrix, weil sie eine lineare Transformation zwischen $X_1 \in \Re^3$ und $X_2 \in \Re^3$ beschreibt.

$$X_2 = HX_1 \tag{4.36}$$

Es muss geachtet werden, dass die Matrix H auf die Bewegung-Parametern $\{R, T\}$ sowie die Struktur-Parameter $\{N, d\}$ der Ebene P abhängt. Aufgrund der inhärenten Skalierungsmehrdeutigkeit des Begriffs $\frac{1}{d}T$ in Gleichung 4.35, kann man nur am meisten erwarten, aus der Matrix H eine skalierte Translation T zu berechnen. Aus

$$\lambda_1 x_1 = X_1,\ \lambda_2 x_2 = X_2,\ X_2 = HX_1, \tag{4.37}$$

wobei λ_1 und λ_2 entsprechend Skalierungsfaktoren für das erste und zweite Kamera-Frame,
haben wir:

$$\lambda_2 x_2 = H\lambda_1 x_1 \Leftrightarrow x_2 \sim Hx_1 \tag{4.38}$$

Schließlich haben wir die Beziehung zwischen zwei Kamera-Frames

$$x_2 \sim Hx_1 \tag{4.39}$$

Anhand von zwei Korrespondenz-Punkt-Listen zwischen zwei Kamera-Frames kann die Homographie approximiert werden. Es gibt bisher in Literatur verschiedene Algorithmen um die Homographie aus zwei Korrespondenz-Punkt-Listen zu berechnen. In dieser Arbeit können zwei Algorithmen eingesetzt werden. Das erste Verfahren basiert auf dem RANSAC-Algorithmus ([Fis80]). Das Zweite beruht auf der Methode der kleinsten Quadrate. Beide Methoden untersuchen die Eingabedaten auf Ausreißer unter Vorgabe eines bestimmten Modells. Wobei in diesem Zusammenhang Ausreißer als falsche Korrespondenzen aufgefasst werden und das Modell die Homographie der beiden Ansichten ist. Da in dieser Arbeit der in OpenCV implementierte Befehl cvFindHomographie benutzt wird, werden die ausführlichen Details von dem Homographie-Approximation-Algorithmus vermieden. Die Leser können mehr in [Sas05] über die detailierten Implementationsschritte lesen.

4.4.6.3 Hormographie-Dekomposition

Nachdem wir die Matrix H in Form $H = (R + \frac{1}{d}TN^T)$ zwischen zwei Kamera-Frames approximiert haben, versuchen wir jetzt die relative räumliche Beziehung, also $(R, \frac{1}{d}T, N)$ zwischen zwei Kamera-Frames aus der Matrix H zu berechnen.
In [Sas05] wurde ein analytischer Algorithmus zur Dekomposition von Homographie-Matrix entwickelt. Die ausführliche Schritte des Algorithmus werden in Tabelle 4.1 präsentiert. Der Beweis von dem Algorithmus kann in der oben genannten Referenz gelesen werden. Nach der Homographie-Dekomposition bekommt man schließlich die Orientierung und Translation von dem aufgenommenen Bild relativ zu dem Referenzbild wie in Abbildung 4.16. Der pinke Pfeil zeigt die Orientierung des Objekts relative zu der waagerechten Richtung, und der grüne Pfeil zeigt die Verschiebung des Objekts zu dem Mittelpunkt des Bildes.

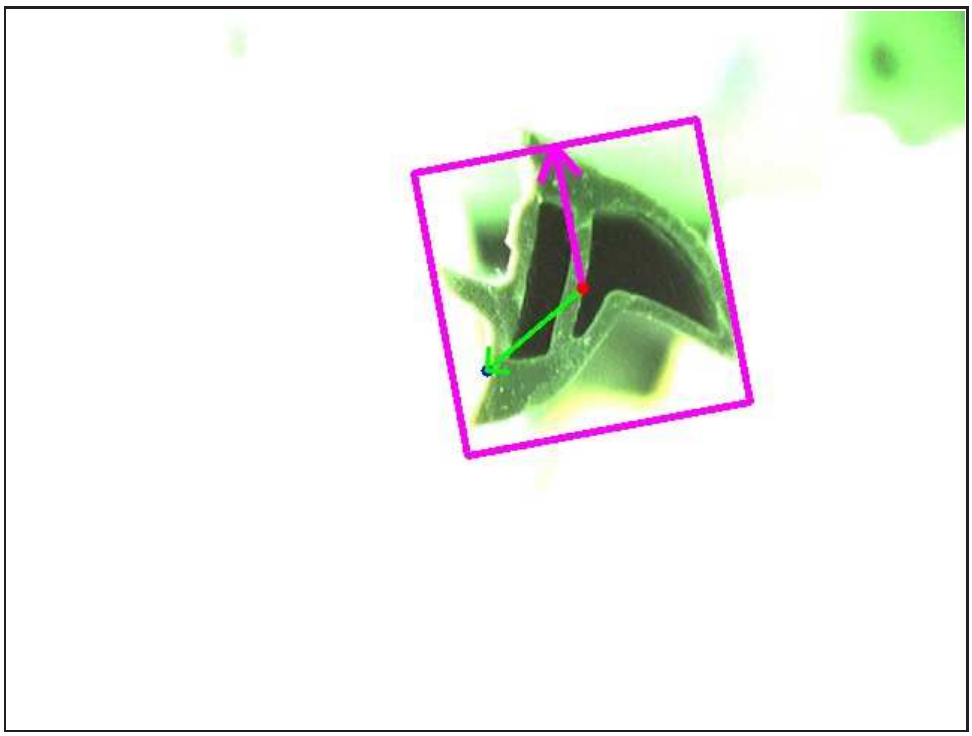


Abbildung 4.16: Das Ergebnis von dem SURF-Algorithmus nach der Homographie-Dekomposition

Zerlegung der Homographie in Struktur und Bewegung

- Berechnung der Singulärwertzerlegung

$$H^T H = V\Sigma^2 V^T,\ V \in SO(3),\ \Sigma^2 = diag\{\sigma_1^2, \sigma_2^2, \sigma_3^2\}$$

- Mit den Spaltenvektoren der Matrix V

$$V = [\upsilon_1, \upsilon_2, \upsilon_3]$$

- schreiben wir

$$H^T H\upsilon_1 = \sigma_1^2\upsilon_1, H^T H\upsilon_2 = \sigma_2^2\upsilon_2, H^T H\upsilon_3 = \sigma_3^2\upsilon_3.$$

- Somit ist der Vektor υ_2 orthogonal zu N und T , und seine Länge bleibt erhalten unter der Wirkung der Abbildung H.
- diese Längenerhaltung gilt auch für die beiden Vektoren

$$u_1 = \frac{\sqrt{1-\sigma_3^2}\upsilon_1 + \sqrt{\sigma_1^2-1}\upsilon_3}{\sqrt{\sigma_1^2-\sigma_3^2}}, u_2 = \frac{\sqrt{1-\sigma_3^2}\upsilon_1 - \sqrt{\sigma_1^2-1}\upsilon_3}{\sqrt{\sigma_1^2-\sigma_3^2}},$$

- Dies gilt für alle Vektoren aus den Unterräumen

$$S1 = span(v2, u1), S2 = span(v2, u2)$$

- Wir definieren die Matrizen

$$U1 = [\upsilon_2, u_1, \widehat{v_2 u_1}],\ W1 = [H\upsilon_2, Hu_1, H\widehat{v_2 H u_1}]$$
$$U2 = [\upsilon_2, u_2, \widehat{v_2 u_2}],\ W2 = [H\upsilon_2, Hu_2, H\widehat{v_2 H u_2}]$$

- Damit ergibt sich

$$RU_1 = W_1,\ RU_2 = W_2.$$

- Lösung

$R_1 = W_1 U_1^T, N_1 = \widehat{v_2 u_1}$ $\frac{1}{d}T_1 = (H - R_1)N_1$	$R_3 = W_1 U_1^T, N_3 = -\widehat{v_2 u_1}$ $\frac{1}{d}T_3 = -(H - R_1)N_1$
$R_2 = W_2 U_2^T, N_2 = \widehat{v_2 u_2}$ $\frac{1}{d}T_2 = (H - R_2)N_2$	$R_4 = W_2 U_2^T, N_4 = -\widehat{v_2 u_2}$ $\frac{1}{d}T_4 = -(H - R_2)N_2$

- die physikalisch sinnvolle Lösung kann ausgewählt werden durch die Nebenbedingung, dass alle sichtbaren Punkte eine positive Tiefe aufweisen sollen, also $N^T e3 = n3 > 0$

Tabelle 4.1: Homographie-Dekomposition

Kapitel 5

Versuchsaufbau

In diesem Kapitel werden die Hardware- sowie die Software-Komponenten, die in dieser Arbeit benutzen werden, präsentiert. Zuerst werden die Spezifikationen des Roboters sowie der Kamera beschrieben. Schließlich wird das Framework FlexRF von dem Lehrstuhl Robotik und Eingebettete Systeme der Fakultät Informatik ausführlich vorgestellt.

5.1 Hardware-Komponenten

Das Apparat wird in Abbildung 5.1 abgebildet. In der Arbeit wird der industrielle Roboter RV6-S von Mitsubishi benutzt. Der Knickarmroboter RV6-S hat die folgenden Kennzahlen:

Achse	6
Traglast	6 kg
Reichweite	696 mm
Wiederholgenauigkeit	±0.02 mm
Masse	58 kg
Struktur	vertikale Mehrfachgelenke

Zwei Kameras wurden zur Bildaufnahme verwendet. Eine Kamera wird in Top-View-Stellung montiert, um das Objekt zu verfolgen, und die andere wird zur Orientierungsbestimmung verwendet. Mit diesem Kamerasystem kann man Eye-to-Hand Visual-Sevoring ([Mar00]) machen.
Unser Programm läuft auf einem Quad-Core Rechner unter Linux-Betriebssystem.

5.2 Software-Komponenten

Für die Bildverarbeitung wird die Bibliothek OpenCV [Bra00] angewendet. Diese Bibliothek liefert zahlreiche low-level Bildverarbeitungsroutine sowie komplexe Algorithmen zur Segmentierung, Verarbeitung und Extraktierung von Informationen

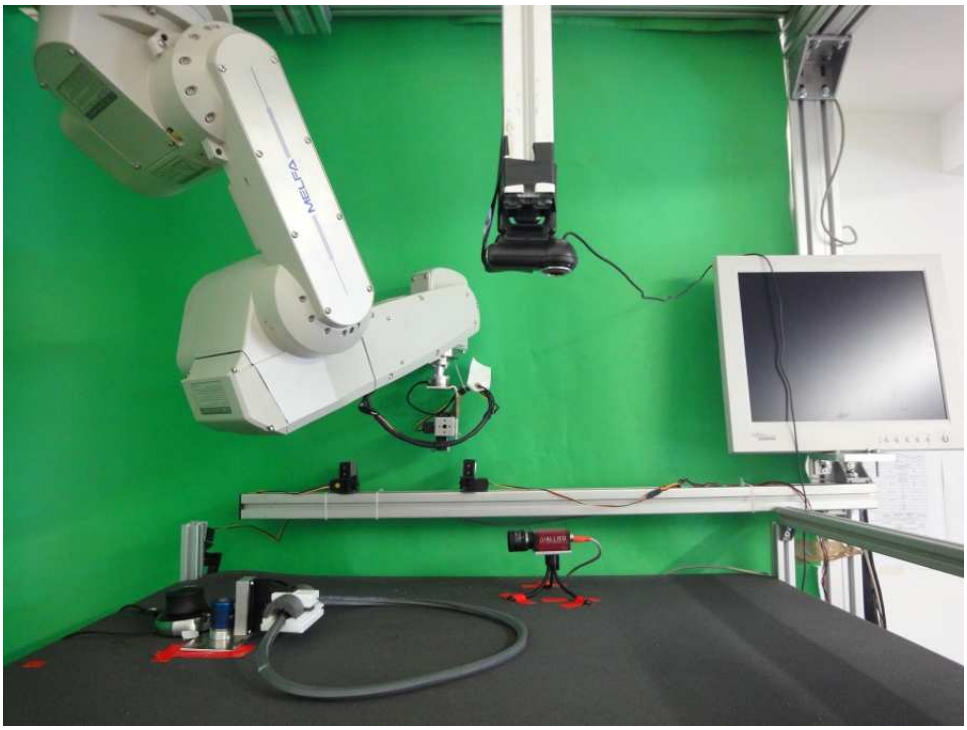


Abbildung 5.1: Der Versuchsaufbau

aus Bildern.
Aus der Perspektive der Modularität besteht das Framework FlexRF aus den vier Bausteinen wie in Abbildung 5.2.
Jede Anwendung, die mit dem FlexRF-Framework entwickelt wird, wird schrittweise

Abbildung 5.2: Grundbausteine des Frameworks FlexRF [Kno10]

aus diesen Bausteinen zusammengesetzt. Eine typische Anwendung von FlexRF ist eine gekoppelte Gruppe der Processing Unit, die sich Daten über die „information storage“ miteinander austauschen. Diese Units können die Generic Interface (generische Schnittstellen) nutzen, um auf externe Hardware- oder Softwarekomponenten zuzugreifen. Automatisch erlauben Widgets die Interaktionen zwischen Processing Unit und dem Benutzer sowie der Visualisierung der verarbeiteten Ergebnisse.

- **Recheneinheiten (Processing Units - PU)**: werden von den Neuronen in der menschlichen Gehirn inspiriert. Jede PU implementiert eine Thread-basierte Schnittstelle, die die Parallelisierung der Informationsverarbeitung

im Framework ermöglicht. Die PUs bieten außerdem eine automatische GUI-basierte Konfiguration und Feedback-Einrichtung sowie die Möglichkeit, seine Informationen mit den anderen Einheiten zu teilen.

- **Datenspeicherung (Information Storages - IS)**: ermöglicht das Austauschen von Daten sowie verarbeiteten Ergebnissen zwischen den Processing Units. Es gibt zwei Arten von Kommunikation in FlexRF: asynchrone Kommunikation und Signal-basierte Kommunikation. FlexRF unterstützt deshalb zwei Modalitäten für die Bereitstellung von Daten für PUs: rein asynchronen Abruf und Signalsubskription. Diese Flexibilität ermöglicht der FlexRF-basierte Anwendung die Handhabung komplexer Situationen, wenn PUs die Daten aus der Informationsspeicherung in spezifischer Weise erhalten müssen.

- **Generische Schnittstellen (Generic interfaces - GI)**: sind die dritte Basiskomponente von FlexRF. Diese Schnittstellen ermöglichen bequemen Zugriff auf externe Hardware-Komponenten wie Bildsensoren, Eingabegeräte (Maus, Tastatur, usw.) oder andere sensorische Geräte, z. B. Kraft-Momenten-Sensoren. Die Implementierung einer generischen Schnittstelle ist nicht für den Zugriff auf Eingabegeräten beschränkt, sondern kann man in der Tat eine Schnittstelle für beliebige externe (Software- oder Hardware-) Komponente schreiben.

- **Interaktion-Widgets (Interaction widgets - IW)**: bieten eine grafische Schnittstelle für Benutzer-Interaktion und Visualisierung. IWs ermöglichen dem Benutzer, textuelle oder grafische Ergebnisse zu kontrollieren sowie die Parameter zu ändern.

Aufgrund der modulatorischen und parallelen Struktur von dem Framework FlexRF können die auf ihm aufbauenden Anwendungen schnell von den Grundbausteinen zusammengesetzt werden und die Vorteile von z. B. Multicore-Rechner-Systemen benutzen. Ein Beispiel für eine Anwendung, die auf FlexRF aufgebaut wird, wird in Abbildung 5.3 veranschaulicht. Zwei Recheneinheiten verbinden mit Peripheriegeräten wie dem Roboter, der Spacemouse und der Kamera. Durch Interaktion-Widgets können die Benutzer die Ergebnisse aus den Geräten in GUI sehen oder mit Hilfe von GUI-Elementen mit diesen Geräten interagieren.

Alle unsere Algorithmen wurden in einer Anwendung aufbauend auf dem FlexRF implementiert. Low-Level-Interaktionen mit dem Roboter, Kommunikationen mit Peripheriegeräten und grafische Simulation-Oberfläche wurden von FlexRF bereitgestellt. Die Bildverarbeitung wurde in einer Recheneinheit implementiert, die nötigen Informationen für die Kognition-Einheit liefert. Die Kognition-Einheit plant dann die entsprechenden Visual-Servoing-Aktionen für den Roboter. Der Roboter führt schließlich diese entsprechenden Aktionen aus, um unser Problem zu lösen. Die Oberfläche des Programms wird in Abbildung 5.4 veranschaulicht. Die wichtigsten Komponente von der GUI werden in Abbildung 5.5 präsentiert.

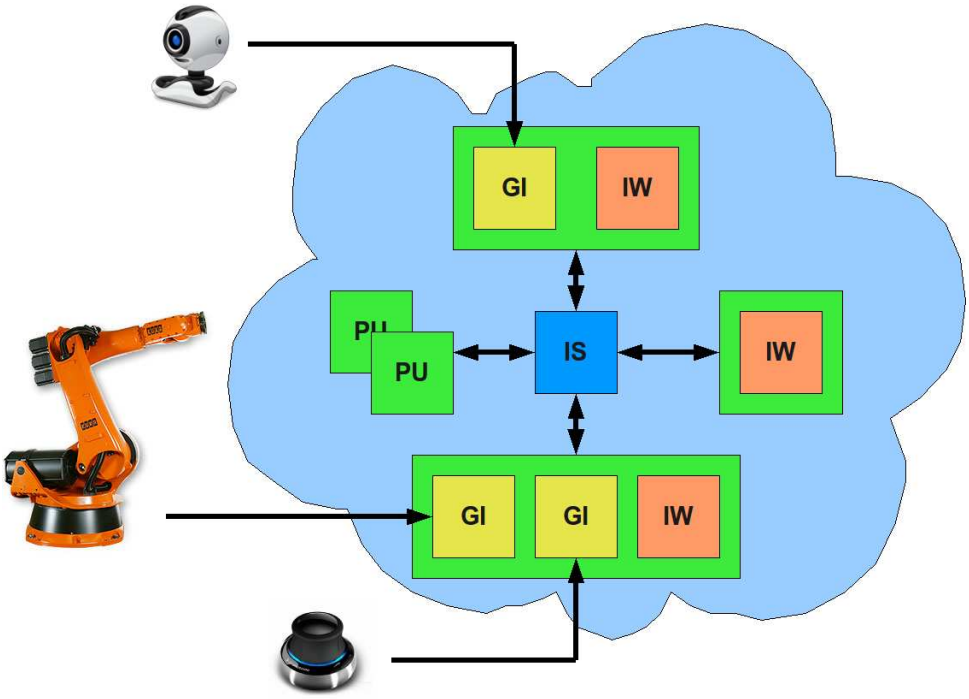


Abbildung 5.3: Ein Beispiel einer Anwendung basierend auf dem Framework FlexRF [Kno10]

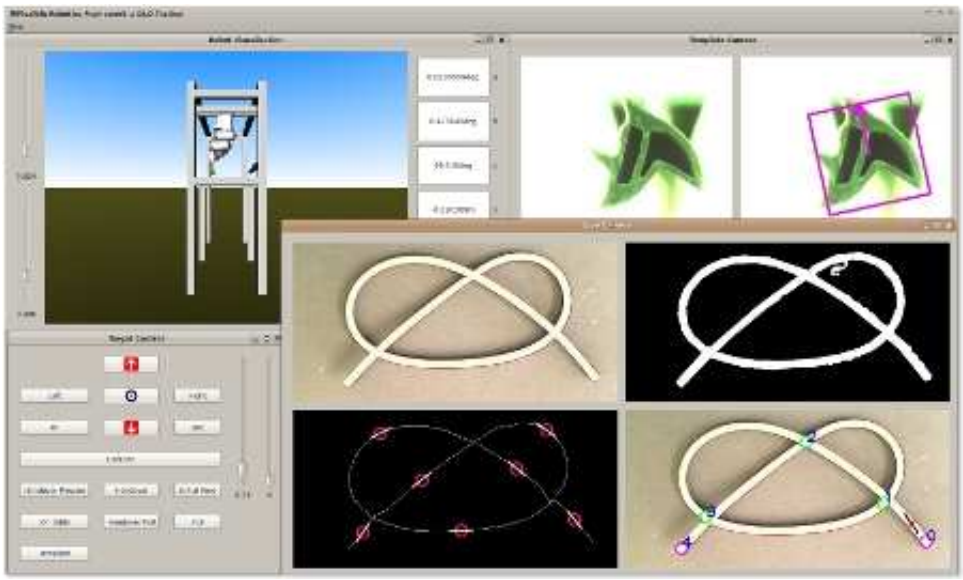
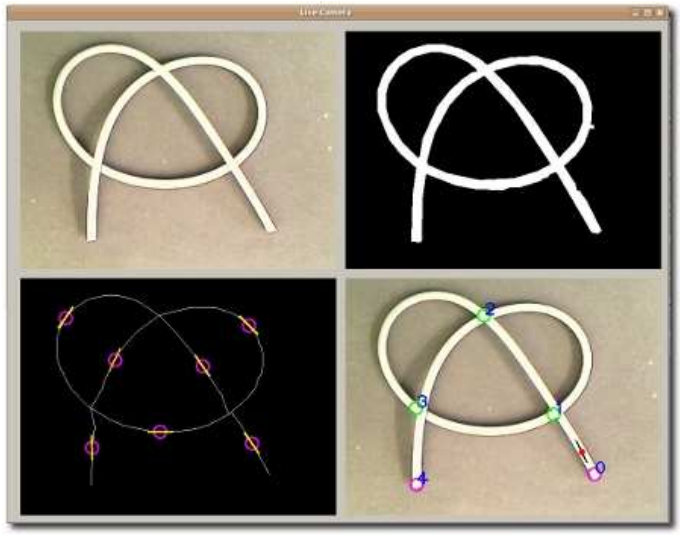
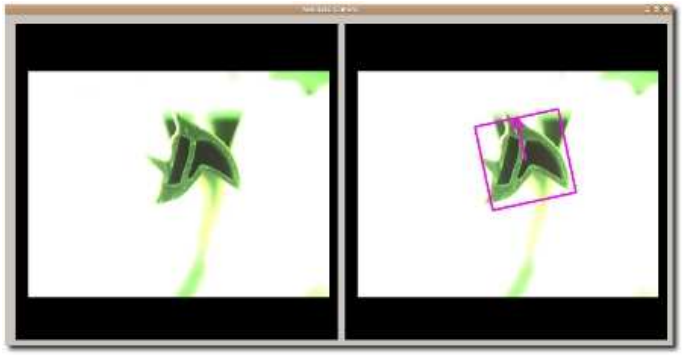


Abbildung 5.4: Die Oberfläche des Programms

a)



b)



c)

Abbildung 5.5: Die Komponenten der GUI: a) Simulation. b) Strukturanalyse. c) Bestimmung der Orientierung

Kapitel 6

Robotmanipulation

Nachdem die Struktur des Objekts erkannt wurde, kann Manipulationsplan für den Roboter erstellt werden. Die Position von dem Objekt in Roboterkoordinaten kann man durch die relative Position zwischen Kamera und Roboter und die absolute Position in Bildkoordination berechnen : ${}^{Roboter}T_{Objekt} = {}^{Roboter}T_{Bild} . {}^{Bild}T_{Objekt}$. Die komplette Aktion zur Kombinieren beider Ende des Objekts wird in mehreren Unteraktion aufgegliedert. Die Gliederung wird in der Abbildung 6.1 veranschaulicht. Die Struktur-Analyse-Routine wird zuerst aufgerufen, um zu überprüfen ob eine Entknoten-Aktion benögtig ist, und wenn ja, wird die Entknoten-Aktion solange aufgerufen, bis die Knotensequenz der Objektstruktur keine wiederholte Untersequenz mehr enthält. Danach wird die Greifpunkt eines Ende berechnet und an den Roboter geschickt. Der Roboter fährt zu diese Position und hebt das Ende ab. Es folgt dann die Übergabe des Objekts an den Greifer-System. Der Roboter zieht das Objekts durch den speziellen Greifer durch. Dieser Schritt stellt sicher, dass sich die Dichtung nicht gedreht. Folglich wird ein Ende wieder von dem Roboter aufgehoben und vor der Kamera gestellt, um seine Orientierung zu bestimmen. Der Roboter dreht sich dann in richtiger Richtung und fügt das Ende in den Muster ein. Für das andere Ende werden die gleichen Schritte durchgeführt. Am Ende werden die zwei Ende in richtiger Richtung zusammengeklebt und das Objekt windet sich auch nicht. In diesem Abschnitt werden die Ergebnisse von allen Aktionplänen an echten Roboter präsentiert.

6.1 Entknoten

Die Entknoten-Aktion garantiert, dass kein Knoten entsteht wird, wenn der Roboter die beiden Ende des Objekts manipuliert. Die Methode im Abschnitt 3.2 wird benutzt, um zu überprüfen, ob diese Aktion benötigt ist. Die Abbildung 6.2 zeigt z. B. einen Fall, in dem diese Aktion zuerst ausgeführt werden muss. Die Knoten-Sequenz in diesem Fall ist: $0-1-2-3-1-2-3$. Diese Sequenz gehört zu der Gruppe, die potenziell Knoten entsteht, wenn die beiden Enden des Objekts gezogen werden. Der Entknoten-Algorithmus in der Sektion 3.2 sucht nach dem zu bearbeitenden Ende

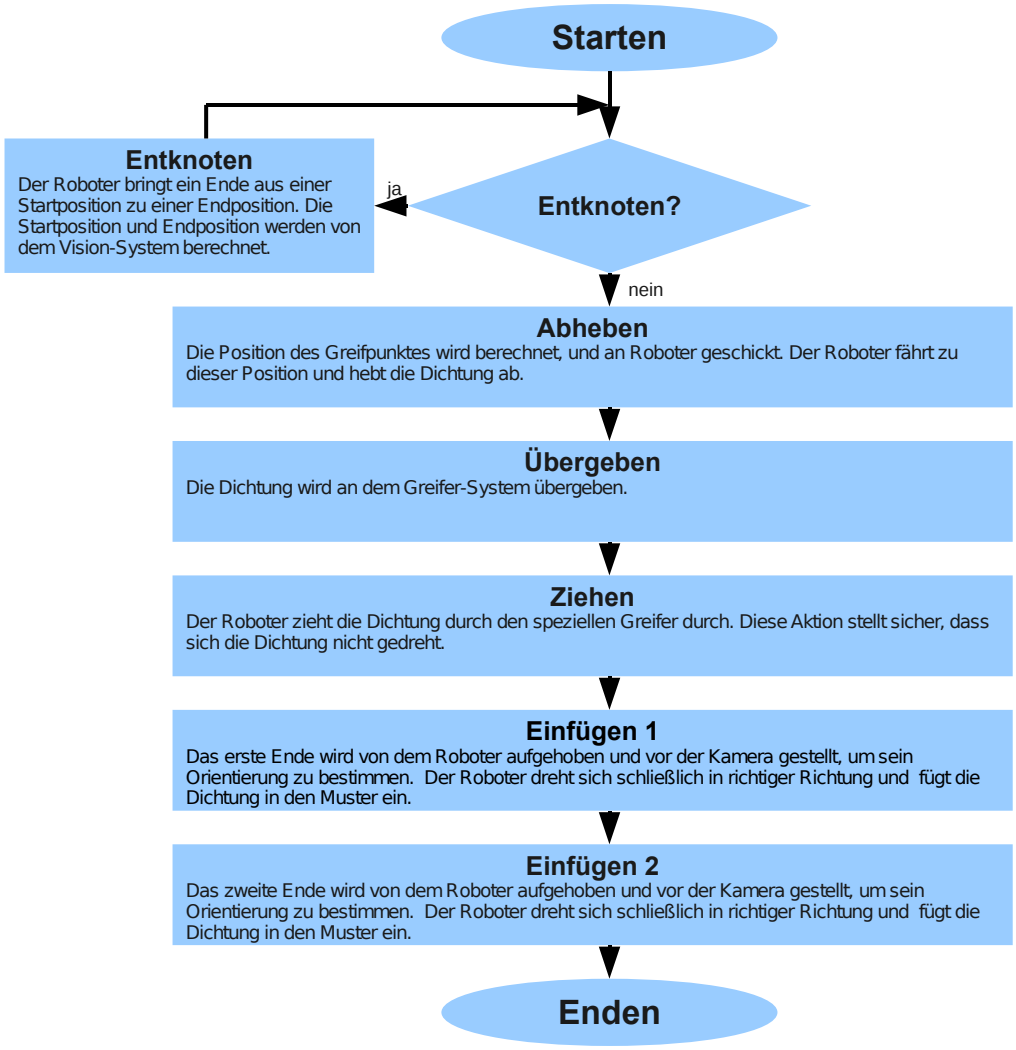
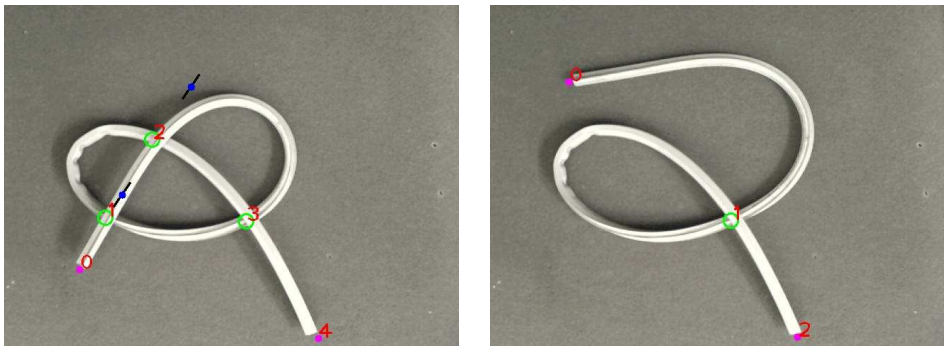


Abbildung 6.1: Die Gliederung des Aktionplan

und berechnet den dazugehörigen Greifpunkt und den Absetzpoint. Diese Aktion wird wiederholt, bis die Knotensequenz keine wiederholte Subsequenz mehr enthält. Die entsprechenden Bildsequenzen von der Entknoten-Aktion werden in Abbildung 6.5 gezeigt.

6.2 Übergeben

Um sicherzustellen, dass sich das Objekt nicht dreht, wird ein speizelles Greifer-System wie in Abbildung 6.3 benutzt. Das Objekt wird wiederholt von dem Roboter durch den speziellen Greifer durchgezogen. Der spezielle Greifer hat eine spezielle Konstruktion, damit das Objekt nur in einer Richtung durchgelaufen werden kann.



a) b)

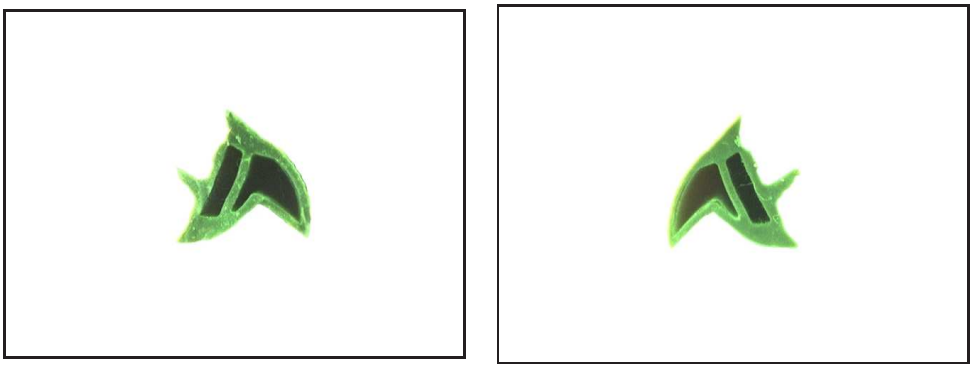
Abbildung 6.2: Die Knoten-Sequenzen: a) Die Sequenz vor der Entknoten-Aktion. b) Die Sequenz nach der Entknoten-Aktion

Endeffekt sind beide Ende in gleicher Richtung, wenn das Objekt komplett durchgezogen wird.
Die Abbilding 6.6 zeigt die Bildsequenz von der Übergeben-Aktion.

Abbildung 6.3: Der spezielle Greifer

6.3 Kleben

Um die beiden Enden richtig an einander kleben zu können, müssen die Orientierungen von beiden Endungen relative zu einander bestimmt werden. Template-Matching-Methoden werden benutzt um diese Orientierungen zu berechnen. In deiser Arbeit wurden drei im Kapitel 4 erwähnten Ansätze implementiert. Der Brute-Force- und Ciratefi-Algorithmus sind Region-basierte Ansätze und der SURF-basierte Algorithmus ist ein Merkmal-basierter Ansatz. In dieser Arbeit wurde die freie Bibliothek OpenSURF ([Eva09]) benutzt. Die OpenSURF benutzt auch die OpenCV-Bibliothek und liefert komplette Funktionen zum Erstellen der Deskriptoren und



a) b)

Abbildung 6.4: Die Referenzbilder für zwei Ende des Objekts im SURF-Algorithmus

zum Korrespondieren der Punkte in zwei Punktlisten. Deshalb lässt sich die Bibliothek OpenSURF einfach im Framwork FlexRF integrieren. Die Implementierung vom Ciratefi-Algorithmus basiert auf dem Referenzcode von XPlab ([Sac10]). Der komplette Algorithmus wurde zu OpenCV umgeschrieben und in FlexRF integriert. Da das zu erkennende Template nicht symmetrisch ist und wir auch keinen einheitlichen Hintergrund haben, ist es schwierig ein Graustufen-Template zu erstellen. Um dieses Problem zu überwinden müssen wir zuerst das aufgenommene Bild binarisieren. Mit der Betrachtung, dass die inneren Texturen von dem Querschnitt des Objekts dunkler als die Umgebung sind, können wir hier dies durch eine Threshold-Operation segmentieren. Das aus der Segmentierung resultierende Bild ist in Abbildung 4.4-b) in der Seite 61 zu sehen.
Für den SURF-Algorithmus sind die Template-Bilder die Bilder in Abbildung 6.4, in dem das Objekt genau in der Mitte steht. Drei Algorithmen wurden in Framework FlexRF implementiert und werden in Tabelle 6.1 nach Zeit, Genauigkeiten, CPU-Belastung verglichen.

Wie in Tablle 6.1 gesehen werden kann, ist der SURF-basierte Algorithmus am schnellsten. Und er belastet den CPU auch am wenigsten. Das Problem von dem SURF-basierten Algorithmus ist aber die Genauigkeit und der Anzahl der invarianten Feature-Punkte. Für die Approximation der Homographie-Matrix müssen mindesten vier Paare der Korrespondenz-Punkte zwischen dem Referenzbild und dem aufgenommenen Bild gefunden werden. Da der Querschnitt des Objekts auch nicht viele Texturen enthält, kann es nicht garantiert werden, dass genug vier Paarre von Punkten gefunden werden können. Deshalb kann der SURF-basierte Algorithmus nicht immer ausgeführt werden. Außerdem beeinflusst der akkumulative Fehler durch mehrere Approximationen das Endergebnis. Die Ergebnisse von Brute-Force- und Ciratefi-Algorithmus unterscheiden sich nicht viel. Der Ciratefi-Algorithmus ist aber schneller als der Brute-Force-Algorithmus. Das Problem von dem Brute-Force- und Ciratefi-Algorithmus ist neben der Laufzeit die Segmentierung von den Merkmalen. Da es schwierig ist, ein graustufiges Referenzbild zu erstellen, muss ein

binäres Referenzbild benutzt werden. Das aufgenommene Bild muss deshalb vor dem Template-Matching-Prozess binarisieret werden. Dieser Segmentierungsschritt ist aber nicht robust gegen Illuminationsänderungen. Wenn das aufgenommene Bild falsch segmentiert wurde, wird das Ergebnis von dem Template-Matching-Prozess auch nicht mehr richtig. Ein anderes Problem von Burte-Force-Algorithmus ist die hohe Belastung auf CPU. In dieser Arbeit wird der Ciratefi-Algorithmus benutzt und einen Kompromiss zwischen Genauigkeit und Geschwindigkeit zu erhalten.

	Ground truth	Brute-Force Algorithmus	Ciratefi Algorithmus	Surf Algorithmus
Mittelpunkt	(320,240)	(320,240)	(320,240)	(320,240)
Winkel	0	0°	0°	0°
Skala:		1.0	1.0	1.0
Laufzeit:		83s	28s	0.472132s
CPU-Belastung:		85%	40%	30%
Mittelpunkt	(320,240)	(320,240)	(320,240)	(328,246)
Winkel	45	45°	46°	44.4322°
Skala:		1	1.0	0.937885
Laufzeit:		84s	27s	0.486376s
CPU-Belastung:		85%	40%	30%
Mittelpunkt	(320,240)	(320,240)	(320,240)	(321,241)
Winkel	90	90°	90°	89.3595
Skala:		1.0	1.0	0.984798
Laufzeit:		81s	27s	0.471331s
CPU-Belastung:		85%	40%	30%
Mittelpunkt	(320,240)	(320,240)	(320,241)	(315,236)
Winkel	135	135°	136°	134.249
Skala:		1.0	1.0	0.995554
Laufzeit:		84s	27s	0.477495s
CPU-Belastung:		85%	40%	30%
Mittelpunkt	(320,240)	(320,240)	(319,239)	(314,236)
Winkel	180	180°	180°	179.775
Skala:		1.0	1.0	1.02006
Laufzeit:		84s	25s	0.469763s
CPU-Belastung:		85%	40%	30%
Mittelpunkt	(320,240)	(320,240)	(321,240)	(326,244)
Winkel	225	225°	226°	223.994
Skala:		1.0	1.0	0.997188
Laufzeit:		83s	28s	0.48936s
CPU-Belastung:		85%	40%	30%
Mittelpunkt	(320,240)	(320,240)	(320,240)	(325,244)
Winkel	270	270°	270°	269.586
Skala:		1.0	1.0	0.965407
Laufzeit:		82s	28s	0.491212s
CPU-Belastung:		85%	40%	30%
Mittelpunkt	(320,240)	(320,240)	(320,239)	(314,235)
Winkel	315	316°	316	317.112
Skala:		1.0	1.0	1.01942
Laufzeit:		82s	26s	0.521948s
CPU-Belastung:		85%	40%	30%

Tabelle 6.1: Der Vergleich zwischen Brute-Force-, Ciratefi-, und SURF-Algorithmus

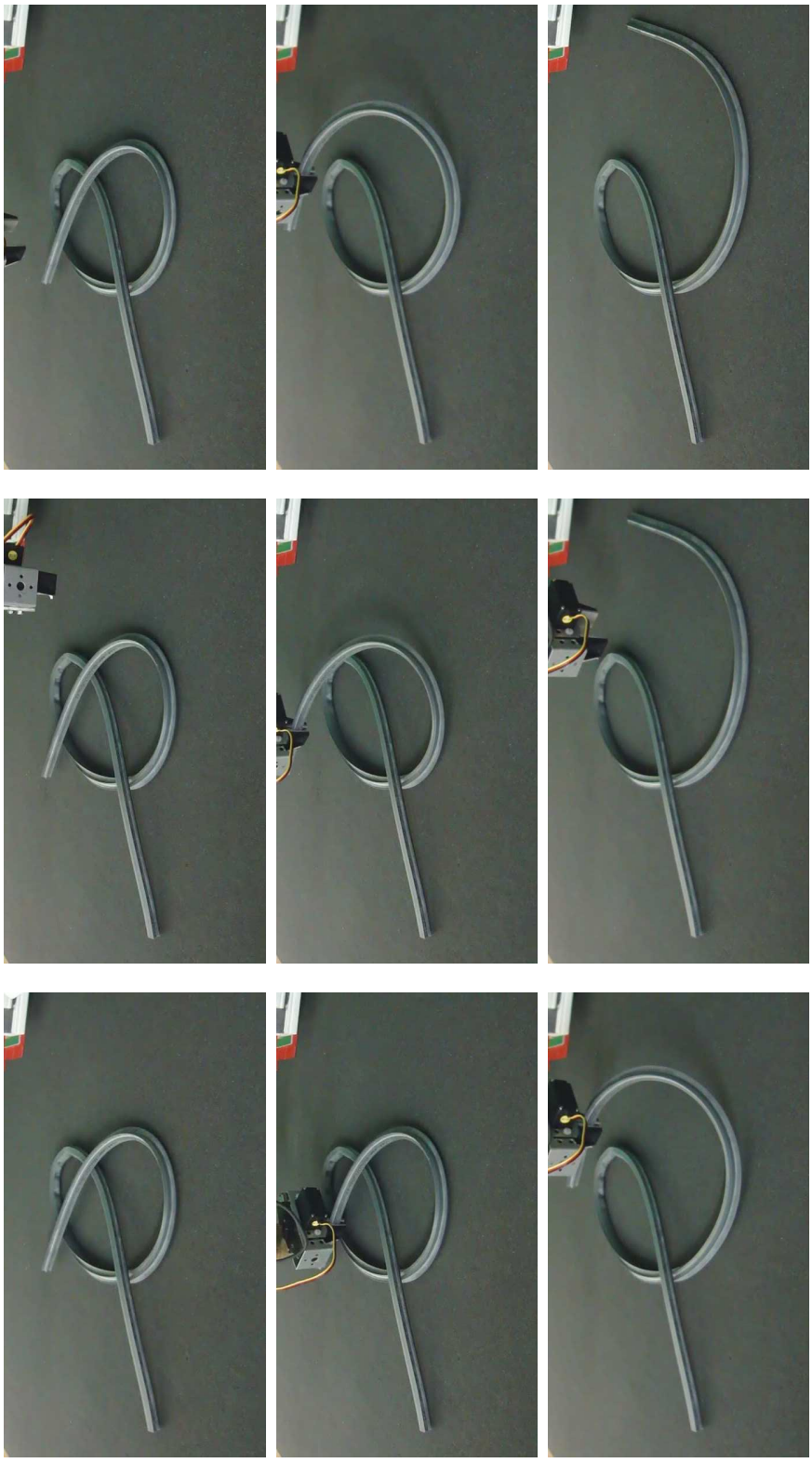


Abbildung 6.5: Bildsequenz von Entknoten-Aktion.

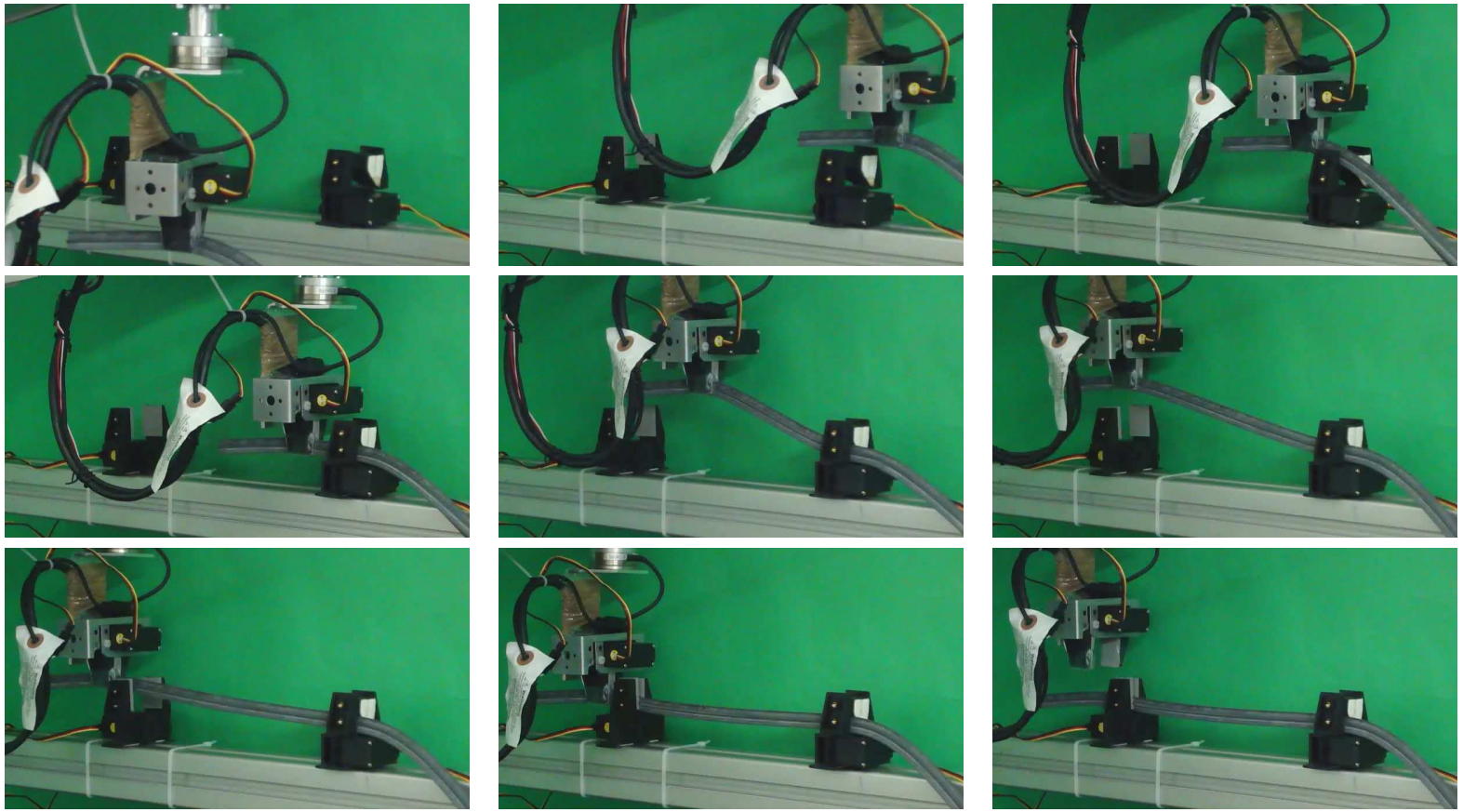


Abbildung 6.6: Bildsequenz von Übergeben-Aktion.

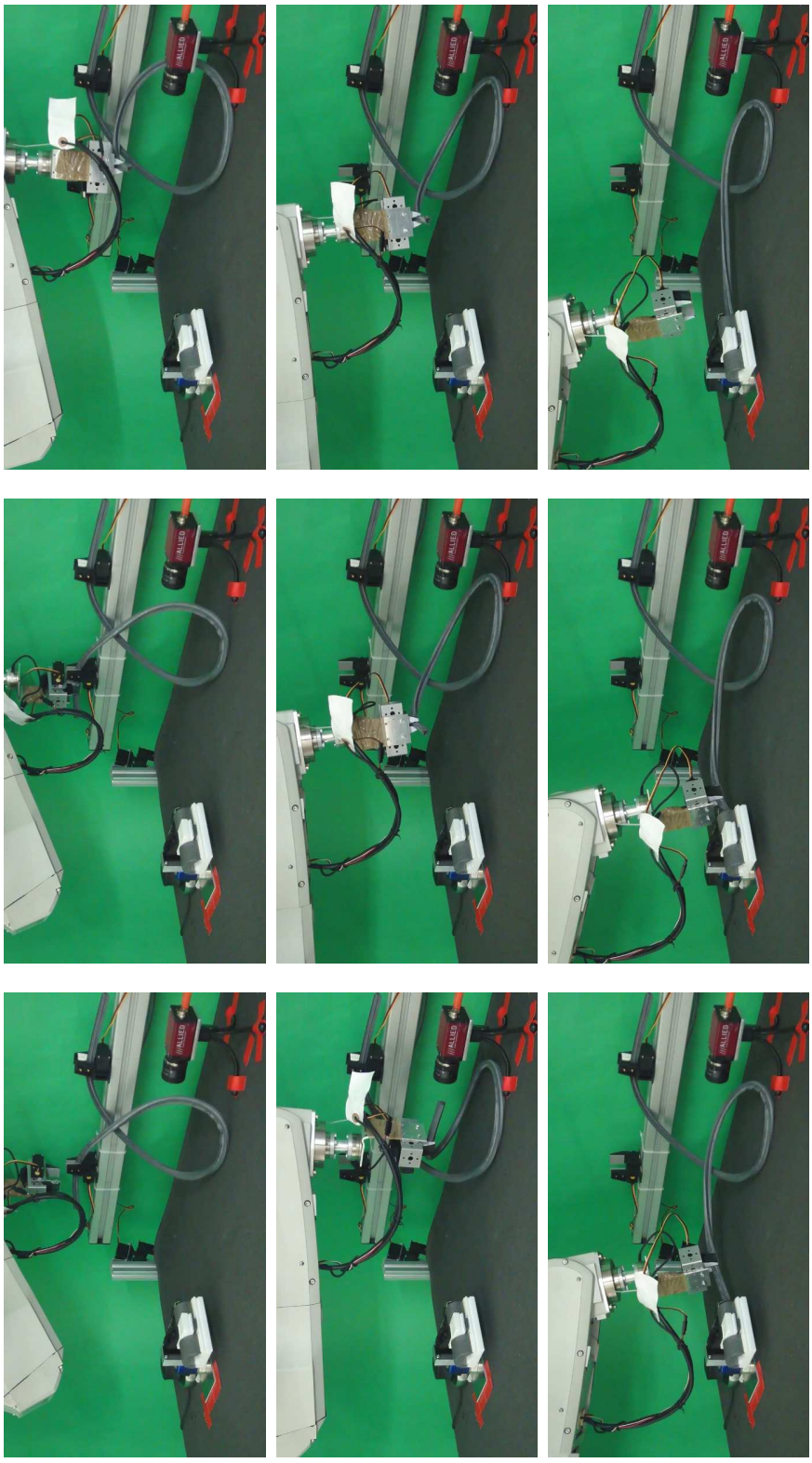


Abbildung 6.7: Bildsequenz von Einfügen-Aktion für das erste Ende.

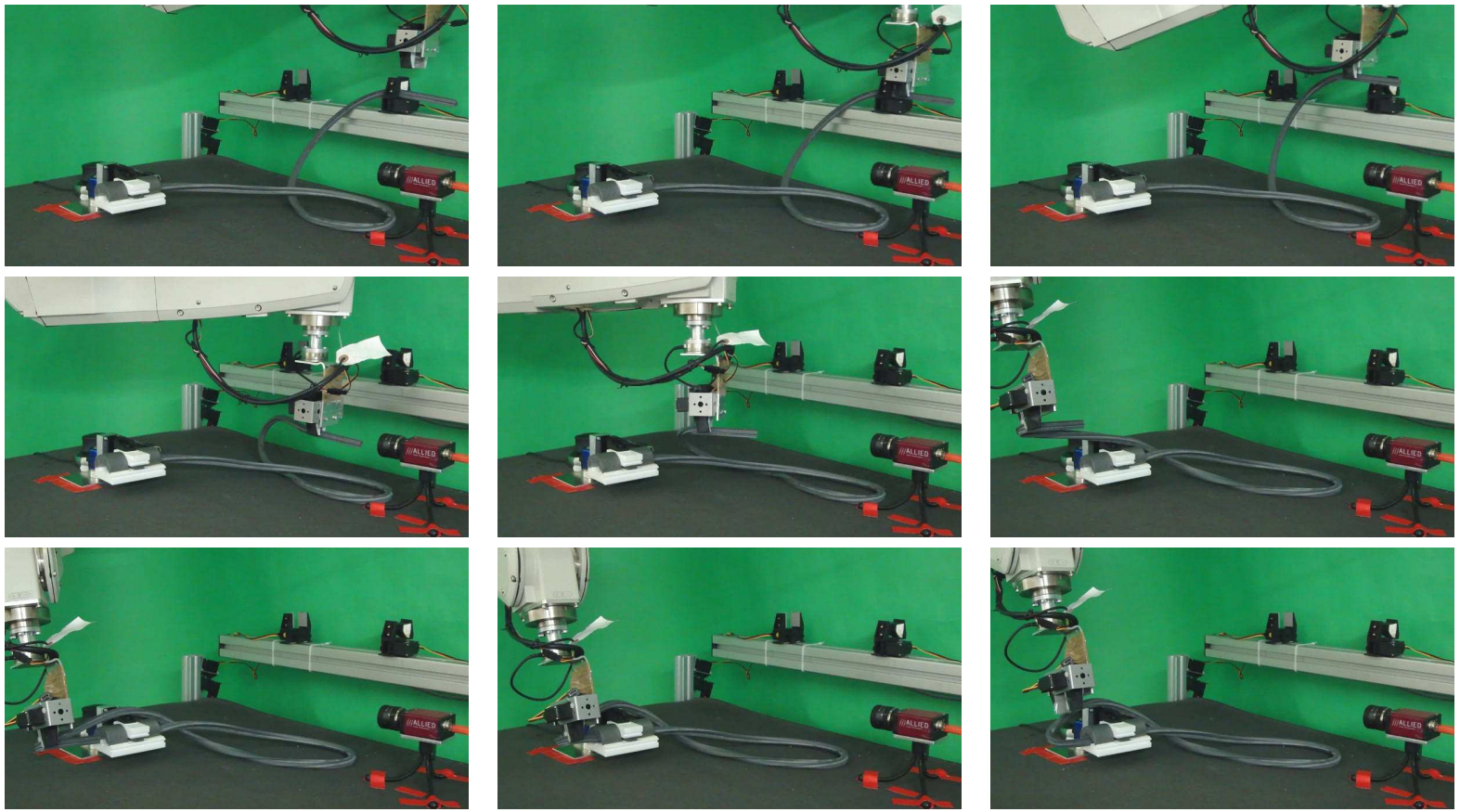


Abbildung 6.8: Die Bildsequenz von Einfügen-Aktion für das zweite Ende

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Ziel dieser Arbeit ist eine komplete Anwendung, die auf ein vorhandenes Roboter-Framework basiert, zur Detektion, Verfolgung und autonome Handhabung von deformierbaren linearen Objekten. Versuchsobjekte dieser Arbeit sind Dichtungen, wie sie in Haushaltgeräte z. B. Waschmaschinen eingesetzt werden. In dieser Arbeit wird die OpenCV-Bibliothek zur low-level Bildverarbeitung benutzt. Die ganze Anwendung wird mit Hilfe von dem Roboter-Framework FlexRF implementiert, das vom Lehrstuhl Robotik und Eigebettete Systeme der Fakultät Informatik entwickelt wurde. Das Framework FlexRF garantiert die Parallelisierungsfähigkeit der Anwendung und liefert einfache Verbindungen mit peripherischen Geräten wie Kamera und dem Mitsubishi-Roboter RV6-S.
Das Objekt wird zuerst durch Hintergrundsubtraktion detektieren, wenn es sich im Arbeitsbereich befindet. Der nicht-parametrische Codebok-Algorithmus wird zur Hintergrundsubtraktion verwendet. Der Vorteil von dieser Methode ist, dass er nicht sensitiv zu Illuminationsänderungen ist. Zur Verfolgung des Objekts wird die Level-Set-Methode verwendet. Für die Level-Set-Methode muss kein Modell für das Objekt erstellt werden. Dies ist besonders wichtig für unsere Anwendung, weil es schwierig ist, ein Modell für ein deformierbares lineares Objekt zu erstellen, oder es ist sehr aufwändig, das dynamische Verhalten von einem deformierbaren linearen Objekt durch ein energybasiertes Modell zu simulieren [Mat08]. Der Nachteil von der klassischen Level-Set-Methode ist, dass sie wegen der Lösung von partiellen Differenzialgleichungen sehr rechenintensiv ist. Deshalb ist die Methode nicht für echtzeitfähige Anwendungen geeeignet. Verschiedene Variante von Level-Set-Methode wurden entwickelt um die Geschwindigkeit zu erhöhen. Die Methode von Yongshi ist besonders besser als andere Methode wie Narrow-Band-Methode oder Fast-Marching-Methode, weil der Algorithmus nur mit zwei Punktlisten arbeitet, die die Grenze des Objekts definieren. Außerdem wird Topologie-Änderung implizit behandelt. Ein neues Thinning-Verfahren wurde in dieser Arbeit entwickelt, die auf Level-Set-Methode basiert. Die Kontur des Objekts, die implizit in zwei Punktlisten aus der

schnellen Implementation der Level-Set-Methode gespeichert wird, wird geschrumpft bis das Objekt nur noch 1-Pixel dicht ist. Dieses Verfahren ist besser als traditionelle Thinning-Algorithmen, in denen, alle Punkte im Bild untersucht werden müssen. Die Kombination zwischen der schnellen Implementation von Level-Set-Methode sowie dem neunen Thinning-Verfahren macht es möglich, das ganze Programm echtzeitfähig laufen zu lassen. Nach dem das Skelett des Objekts durch Thinning generiert wurde, muss die Struktur des Objekts analysiert werden. Da der Roboter nur zwei Ende des Objekts manipuliert, muss es zuerst sichergestellt werden, dass kein Knoten erstellt wenn beide Ende gegen einander gezogen werden. Die Analyse-Methode von Wakamatsu et al. [Hir06a] wurde für diesen Zweck benutzt. Diese Methode braucht keine 3D-Informationen von dem Objekt sondern nur die projektive Silhouette auf eine Ebene. Die Knoten werden sequenziell von einem Ende zum dem anderen nummeriert. Schließlich wird ein Sequenzmuster in dieser diese Nummersequenz gesucht, mit dem es möglicherweise Knoten erstellt, wenn der Roboter beide Enden des Objekts manipuliert. Falls dieser Muster in der Sequenz vorhanden ist, muss der Roboter das Objekts zuerst manipulieren, damit sich dieser Muster nicht mehr in der Knotensequenz des Objekts befindet. Nachdem die Struktur des Objekts analysiert wurde, und die mögliche Greifpunkte berechnet wurden, wird ein kompletter Aktionplan zum Kleben zwei Ende des Objekts erstellt. Das Objekt wird zuerst durch einen speziellen Greifer gezogen, damit es sichergestellt wird, dass sich das Objekt nicht mehr gedreht. Danach werden beide Ende mit Hilfe von einem speizellen Muster aneinander geklebt. Um die Lage der Enden zu bestimmen wurde Template-Matching eingesetzt. In dieser Arbeit wurde drei Methoden implementiert und miteinander verglichen. Der Brute-Force-Algorithmus und der Ciratefi-Algorithmus sind Region-basierte Template-Matching-Ansätze und der SURF-Algorithmus ist ein Merkmal-basierter Ansatz. In dem Brute-Force-Algorithmus wird das Referenzbild mit jemdem Skalierungsfaktor skaliert und auf jeden Winkel gedreht. Die Grauwertverteilung des modifizierten Referenzbildes wird mit allen gleichgrößen Suchfenstern auf dem aktuellen Bild verglichen. Der Winkeln sowie Skalierungsfaktor werden schließlich aus dem besten übereingestimmten Suchfenster gelesen. Der Vorteil von dieser Methode ist die Genauigkeit, der Nachteil ist aber die Laufzeit. Kim et al. [dA07] hat eine alternative Methode vorgeschlagen, um der Prozess zu vergleichen. Der Ciratefi-Algorithmus besteht aus drei Filtern. Der Circular-Sampling-Filter sucht mögliche Skalierungsfaktoren für alle Punkte auf dem Bild. Die möglich zur Weiterverarbeitung markierten Punkte werden als erststufige Punkte genannt. Der Radial-Sampling-Filter sucht für alle erststufige Punkte die möglichen Orientierungen und aktualisiert die potentiell richtigen Punkte zu zweistufigen Punkten. Der letzte Template-Matching-Filter ist ein normales Region-basiertes Template-Matching, das die wahrscheinlichen Skalierungsfaktoren und Orientierungen aus den beiden voherigen Filtern benutzen um die richtigen Punkte zu finden. Der Ciratefi-Algorithmus liefert mit kürzerer Laufzeit das gleiche Ergebnis wie der Brute-Force-Algorithmus. Wie schon erwähnt im Kapitel 4, ist der Vorteil von Region-basierten Ansätzen die

Genauigkeit. Neben der längeren Laufzeit haben diese Ansätze noch einen weiteren Nachteil. Diese Ansätze funktionieren nur mit der Annahme, dass die Transformation zwischen dem aktuellen Bild und dem Referenzbild affin ist. Der dritte Algorithmus zur Lagebestimmung basiert auf dem mermalbasierten SURF-Algorithmus, der kürzlich von Bay et al. [Goo06] vorgestellt wurde. Der SURF-Algorithmus sucht zuerst im Referenzbild und im aktuellen Bild nach Feature-Punkten, die robust gegen Translation, Rotation und Skalierungsänderung sind. Die Feature-Punkte von beiden Bildern werden miteinander verglichen und entsprechend korrespondiert. Anhand von zwei gepaarten Punktlisten werden die Homographie zwischen zwei Bildern approximiert. Ein Homographie-Dekompisition-Algorithmus ([Sas05]) wird schließlich eingesetzt, um die relative Position- und Orientirungsverschiebung zwischen dem aktuellen Bild und dem Referenzbild zu bestimmen. Die Vorteile vom SURF-Algorithmus gegen die zwei vorherigen Region-basierten Algorthmen sind die Geschwindigkeit sowie 3D-Informationen der Lage des Objekts. Da der SURF-Algorithmus nur mit Feature-Punkten und nicht mit allen Punkten auf dem aktuellen Bild arbeitet, wird der Prozess deutlich beschleunigt. Die Homographie liefert die Orientierungsänderung in allen drei Richtungen der kartesischen Koordinaten. Die drei Algorithmen zur Orientierungsbestimmung wurden mit einander verglichen. Der Brute-Force-Algorithmus ist am genausten aber auch am langsamsten. Der SURF-Algorithmus liefert das ungenauste Ergebnis in der kurzesten Laufzeit. Der Ciratefi ist ein Kompromiss von beiden vorherigen Algorithmen, weil der schneller als der Brute-Force-Algorithmus ist und er hat bessere Ergebnisse als die von dem SURF-Algorithmus. Nachdem die Orientierungen beider Ende bestimmt wurden, werden die Ende mit Hilfe von einem speziellen Muster aneinander geklebt.

7.2 Ausblick

Die Analyse von Knotensequenz benutzt in dieser Arbeit nur die projektive Silhouette von dem Objekts auf eine Ebene. Wakamatsu et al. [Hir06b] hat auch eine Methode zur Analyse der Struktur des deformierbaren linearen Objekts anhand von 3D Informationen der Knoten präsentiert. Mit dieser Analyse können wir noch genauer feststellen ob Knoten entstehen, wenn wir beide Ende des Objekts manipulieren.
Mehrere Modelle für deformierbare, lineare Objekte wurden von Remde in [Hen00], Wakamatsu in [Hir05] oder Huang in [Mat08] vorgeschlagen. Die meisten Modelle versuchen, die interne Energie des Objekts bei Robotermanipulation zu minimieren. Mit diesen Modellen kann man das Verhalten des Objekts durch die Manipulationen des Roboters abschätzen und entsprechend einen effizienten Aktionplan erstellen. Man kann auch durch diese Modell mit Hilfe von Open Tracking Library (OpenTL [Kno08]) effizientere Algorithmen zur Verfolgung der deformierbaren linearen Objekte implementieren. Dies ermöglicht uns, das Objekt auch in komplex strukturierter Umgebung zu verfolgen.

Fujiki et al. [Mac] hat auch ein probalitisches Modell zur autonomen Erkennung der Topologie von mehreren Kabeln entwickelt. Das Kabel-Modell wird mit eine probalitische Intensität-Funktion beschrieben und seine Parameter wird durch einen EM-Algorithmus bestimmt. Mit diesem Modell kann man die Anwendung zur Manipulation von mehreren Objekten erweitern.
Der SURF-Algorithmus weist eine bessere Performanz im Vergleich zu zwei anderen Region-basierte Template-Matching-Algorithmen aus. Das Problem von dem SURF-Algorithmus ist die Anzahl der übereingestimmten Punkte zwischen dem Referenzbild und dem aufgenommenen Bild. Der kürzlich von Yu et al. [Yu09] vorgestellte ASIFT-Algorithmus kann deutlich mehr korrespondierende Punkte zwischen den Bildern finden. Obwohl der ASIFT-Algorithmus langsamer als der SURF-Algorithmus ist, ist er aber immer noch viel schneller als der Brute-Force-Algorithmus oder der Ciratefi-Algorithmus.
In [Fuk06] hat Matsuno ein Dual-Hand-System benutzt, um einen Kabel zu manipulieren. Das gleiche System kann auch bei unserer Anwendung eingesetzt werden, um die Flexibilität und Geschwindigkeit der Anwendung zu erhöhen.

Abbildungsverzeichnis

Literaturverzeichnis

[Anu70] P. E. Anuta. Spatial registration of multispectral and multitemporal digital imagery using fast fourier transform techniques. *IEEE Trans. Geoscience Electronics*, 8:353–368, October 1970.

[Ben82] G. C. Stockman; S. Kopstein; S. Benett. Matching images to models for registration and object detection via clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(3):229–241, 1982.

[Bla00] J. P. MacCormick; A. Blake. A probabilistic exclusion principle for tracking multiple objects. *International Journal of Computer Vision*, 39(1):57–71, August 2000.

[Bra00] G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 25(11):120, 122–125, November 2000.

[Bre65] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, July 1965.

[Bre77] J. Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20:100–106, February 1977.

[Bru09] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, May 2009.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–698, 1986.

[Cha02] L. A. Vese; T. F. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *International Journal of Computer Vision*, 50(3):271–293, December 2002.

[dA07] H. Y. Kim; S. A. de Araujo. Grayscale template-matching invariant to rotation, scale, translation, brightness and contrast. In *PSIVT*, pages 100–113, 2007.

[Dav99] T. Horprasert; D. Harwood; L. S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. In *FRAME-RATE: Frame-rate Applications, Methods and Experiences with Regularly Available Technology and Equipment*, 1999.

[Dav05] K. Kim; T. H. Chalidabhongse; D. Harwood; L. S. Davis. Real-time foreground-background segmentation using codebook model. *Real-Time Imaging*, 11(3):172–185, June 2005.

[Des01] T. Deschamps. *Curve and Shape Extraction with Minimal Path and Level-Sets techniques*. PhD thesis, L'Univerite de Paris-Dauphine, 2001.

[Ebr04] A. Cavallaro; E. Salvador; T. Ebrahimi. Detecting shadows in image sequences, January 20 2004.

[Eva09] C. Evans. Notes on the opensurf library, 8 2009.

[Fis80] R. C. Bolles;M. A. Fischler. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Image Understanding Workshop*, pages 71–88, 1980.

[Flu03] B. Zitova; J. Flusser. Image registration methods: A survey. *Image and Vision Computing*, 21(11):977–1000, October 2003.

[Fuk06] T. Matsuno; D. Tamaki; F. Arai; T. Fukuda. Manipulation of deformable linear objects using knot invariants to classify the object condition based on image sensor information. *IEEE/ASME Transactions on Mechatronics*, 11:401–408, 2006.

[Goo06] H. Bay; T. Tuytelaars; L. V. Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[Gri99] C. Stauffer; W. Eric; L. Grimson. Adaptive background mixture models for real-time tracking. In *CVPR*, pages 2246–2252. IEEE Computer Society, 1999.

[Haz99] F. C. M. Martins; B. R. Nickerson; V. Bostrom; R. Hazra. Implementation of a real-time foreground/background segmentation system on the intel architecture. In *FRAME-RATE: Frame-rate Applications, Methods and Experiences with Regularly Available Technology and Equipment*, pages xx–yy, 1999.

[Hen00] A. Remde; D. Henrich. Direct and inverse simulation of deformable linear objects, 2000.

[Hir05] H. Wakamatsu; K. Takahashi; S. Hirai. Dynamic modeling of linear object deformation based on differential geometry coordinates. In *ICRA*, pages 1028–1033. IEEE, 2005.

[Hir06a] H. Wakamatsu; A. Tsumaya; E. Arai; S. Hirai. Unraveling of deformable linear objects based on 2D information about their crossing states. In *IROS*, pages 3873–3878. IEEE, 2006.

[Hir06b] H. Wakamatsu; E. Arai; S. Hirai. Knotting/unknotting manipulation of deformable linear objects. *I. J. Robotic Res*, 25(4):371–395, 2006.

[Hum86] R. A. Hummel. Representations based on zero crossings in scale space. In *Proceedings, CVPR '86 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, June 22–26, 1986)*, IEEE Publ.86CH2290-5, pages 204–209. IEEE, 1986.

[Isa98] A. Blake; M. Isard. *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer-Verlag, 1998.

[Kae08] G. Bradski; A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 1st edition, October 2008.

[Kar05] Y. G. Shi; W. C. Karl. Real-time tracking using level sets. In *CVPR*, pages II: 34–41, 2005.

[Kno08] G. Panin; C. Lenz; M. Wojtczyk; S. Nair; E. Roth; T Friedlhuber; A. Knoll. A unifying software architecture for model-based visual tracking. volume 6813, pages 681303+. SPIE, 2008.

[Kno10] T. Müller; A. Knoll. A generic approach to realtime robot control and parallel processing for industrial scenarios. In *International Conference on Industrial Technology*. IEEE, 2010.

[Koe84] J. J. Koenderink. The structure of images. *Biological Cybernetics*, 50:363–370, 1984.

[KWBE02] G. Kühne, J. Weickert, M. Beier, and W. Effelsberg. Fast implicit active contour models. In *DAGM*, page 133 ff., 2002.

[Lak06] M.; Chan K. L.; Zhou J. Lakshmanan, A.; Thida. Incorporation of active contour without edges in the fast level set framework for biomedical image segmentation. *Biomedical and Pharmaceutical Engineering*, pages 296–300, 2006.

[Lin98] T. Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):79–116, November 1998.

[Low99] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.

[Low04] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[Mac] R. Fujiki; H. Tanaka; Y. Kawahara; T. Yairi; K. Machida.

[Mar00] G. Flandin; F. Chaumette; E. Marchand. Eye-in-hand/eye-to-hand cooperation for visual servoing. In *ICRA*, pages 2741–2746. IEEE, 2000.

[Mat08] J. Huang; P. Di; T. Fukuda; T. Matsuno. Dynamic modeling and simulation of manipulating deformable linear objects. pages 858 – 863. IEEE, 2008.

[Mey99] K. Toyama; J. Krumm; B. Brumitt; B. Meyers. Wallflower: Principles and practice of background maintenance. In *ICCV*, pages 255–261, 1999.

[Nan02] M. Saadat; P. Nan. Industrial applications of automatic manipulation of flexible materials, 2002.

[Osh94] B. Merriman; J. K. Bence; S. J. Osher. Motion of multiple junctions: a level set approach. *J. Comput. Phys.*, 112:334–363, 1994.

[Pas01] G. Paschos. Perceptually uniform color spaces for color texture analysis: An empirical evaluation. *IEEE Trans. Image Processing*, 10(6):932–937, June 2001.

[Pet96] K. Y. Song; J. V. Kittler; M. Petrou. Defect detection in random color textures. *Image and Vision Computing*, 14(9):667–683, October 1996.

[Pra74] W. K. Pratt. Correlation techniques image registration. *IEEE Transactions on Aerospace and Electronic Systems*, (10):353–358, 1974.

[Pro07] J. Bauer; N. Sünderhauf; P. Protzel. Comparing several implementations of two recently published feature detectors. 2007.

[Rol76] D. Rolfson. *Knots and links*. Berkeley, CA, Publish or Perish, Inc., 1976.

[Sac10] E. Sack. Xplab, 3 2010. `http://www.pattern-lab.de`.

[Sas05] Y. Ma; S. Soatto; J. Kosecka; S. S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer-Verlag, 2005.

[Sch81] B. Horn; B. Schunk. Determining optical flow. *Artificial Intelligence*, 20, 1981.

[Sch01] K. Mikolajczyk; C. Schmid. Indexing based on scale invariant interest points. In *ICCV*, pages 525–531, 2001.

[Sch05] K. Mikolajczyk; C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, October 2005.

[Sch06] F. Gasparini; R. Schettini. Skin segmentation using multiple thresholding. volume 6061. SPIE, 2006.

[Set88] S. Osher; J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *JOURNAL OF COMPUTATIONAL PHYSICS*, 79(1):12–49, 1988.

[Set94] D. Adalsteinsson; J. A. Sethian. A fast level set method for propagating interfaces, March 29 1994.

[Set99] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[Sha89] D. Mumford; J. Shah. Optimal approximations by piecewise smooth functions and variational problems. *Comm. on Pure and Applied Math.*, 42(5):677–685, 1989.

[Sha04] A. Yilmaz; X. Li; M. Shah. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(11):1531–1536, November 2004.

[Sha06] A. Yilmaz; O. Javed; M. Shah. Object tracking: A survey. *ACM Comput. Surv*, 38(4), 2006.

[Smo02] B. Schoelkopf; A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

[Ste88] C. Harris; M. Stephens. A combined corner and edge detector. pages 147–151. The Plessey Company plc., 1988.

[Sto85] A. Goshtasby; G. C. Stockman. Point pattern matching using convex hull edges. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(5):631–636, September 1985.

[Sue92] L. Lam; S.-W. Lee; C. Suen. Thinning methodologies: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, 1992.

[Tak03] R. Takei. A new grey-scale template image matching algorithm using the cross-sectional histogram correlation method, 2003.

[Ter88] M. Kass; A. Witkin; D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[Ves01] T. F. Chan; L. A. Vese. Active contours without edges. *IEEE Trans. Image Processing*, 10(2):266–277, February 2001.

[vG05] K. Mikolajczyk; T. Tuytelaars; C. Schmid; A. Zisserman; J. Matas; F. Schaffalitzky; T. Kadir; L. van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(7):43–72, November 2005.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Proc. CVPR*, 1:511–518, 2001.

[vWMS77] P. van Wie; M. Stein. A landsat digital image rectification system. *IEEE Trans. Geoscience Electronics*, 15(3):130–137, July 1977.

[Whi02] R. T. Whitaker. Isosurfaces and level-set. Technical report, December 06 2002.

[Wil01a] A. Tsai; A. J. Yezzi; A. S. Willsky. Curve evolution implementation of the mumford-shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE Trans. Image Processing*, 10(8):1169–1186, August 2001.

[Wil01b] Y. Chen; S. R. Thiruvenkadam; H. D. Tagare; F. Huang; D. Wilson. On the incorporation of shape priors into geometric active contours. In *Variational and Level Set Methods in Computer Vision*, pages xx–yy, 2001.

[Wre96] A. P. Pentland; T. J. Darrell; A. Azarbayejani; C. R. Wren. Pfinder: Real-time tracking of the human body. In *FG*, pages 51–56, 1996.

[YPBJD06] S.M. Yongsheng Pan; Birdwell JD; Djouadi. Efficient implementation of the chan-vese models without solving pdes. *IEEE 8th Workshop on Multimedia Signal Processing*, pages 350–354, 2006.

[Yu09] J. M. Morel; G. S. Yu. ASIFT: A new framework for fully affine invariant image comparison. *Journal on Imaging Sciences*, 2(2):438–469, 2009.