Bachelorarbeit

Hosnia Najem

Realisierung einer Bibliothek für die schnelle Objektverfolgung und Objekterkennung in Bildsequenzen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hosnia Najem

Realisierung einer Bibliothek für die schnelle Objektverfolgung und Objekterkennung in Bildsequenzen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing. Andreas Meisel
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 13. November 2008

Hosnia Najem

Thema der Bachelorarbeit

Realisierung einer Bibliothek für die schnelle Objektverfolgung und Objekterkennung in Bildsequenzen

Stichworte

Merkmale, markante Punkte, Korrespondenz, Tracking, Verfolgung von Objekten, Meanshift-Algorithmus, CamShift-Algorithmus, KLT-Algorithmus, SIFT-Algorithmus

Kurzzusammenfassung

Die vorliegende Arbeit beschreibt und analysiert die Implementierung der drei Bildverarbeitungsalgorithmen zur Merkmalsextraktion von Objekten und deren Verfolgung in Videosequenzen CAMSHIFT, KLT und SIFT in den Bibliothekssystemen LTI-Lib und OpenCV. Diese werden mithilfe der Programmiersprache C++ und C umgesetzt unter der Verwendung der Entwicklungsumgebung Visual Studio .NET 2003 und Impresario.

Hosnia Najem

Title of the paper

Realisation of a library for the fast object tracking and object recognition in picture sequences

Keywords

Featurepoints, Keypoints, SIFT algorithm, CAMSHIFT algorithm, KLT algorithm, Tracking, Correspondence, object tracking, object recognition

Abstract

This work describes and analyzes the implementation of three image processing algorithms for the extraction of objects' attributes and for their tracking in video sequences CAMSHIFT, KLT and SIFT in the library systems LTI-Lib and OpenVC. These are realized using the programming languages C++ and C using the application development systems Visual Studio .NET 2003 and Impresario.

Wer Glaubt, der findet immer einen Weg.

Danksagung

Ich möchte mich an dieser Stelle bei meinem Freund für seine Geduld, während ich an dieser Arbeit saß, bedanken. Des Weiteren möchte ich bei Prof. Dr. Ing. Andreas Meisel für die Begleitung der Arbeit bedanken. Und ein herzlicher Dank geht auch an die Jungs und Deerns von „Oh it's fresh“ für den guten Kaffee und die allmorgendliche Ermunterung. Und nicht vergessen werden sollen auch meine Eltern, die mich immer in meinem Studium bestärkten.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

Die hier vorliegende Arbeit beschäftigt sich mit dem Erkennen und Verfolgen von Objekten in Bildsequenzen. Dieser Vorgang nennt sich in der Fachsprache Objekt-Tracking.

Die Grundlage des Trackings ist die Definition eines zu suchenden Objektes anhand von Referenzbildern. Dieses Objekt wird danach in einer Bildsequenz automatisch identifiziert.

So wird Tracking zum Beispiel bei Fußball spielenden Robotern (s. Abb. 1.1) zum Auffinden des Spielballs eingesetzt. Dabei ist der reale Spielball das in der zweidimensionalen Abbildung einer Kamera zu trackende Objekt und maßgeblich für die Bewegungen der Roboter.

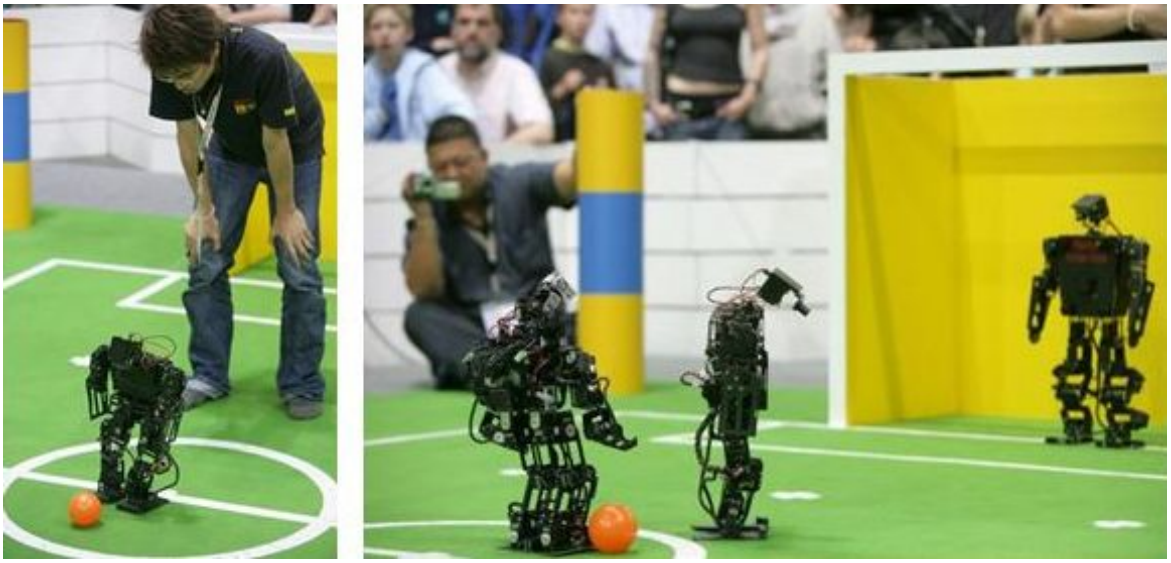

Abbildung 1.1.: Visuelles Tracking bei Fußballrobotern; Quelle: Robocup 2006

Bei computeranimierten Filmen werden menschliche Bewegungen (s. Abb. 1.2) für einen möglichst natürlichen Eindruck auf die Filmfiguren übertragen. Hier kann ein schwarz gekleideter Mensch mit farbigen Markierungen an den Gelenken gefilmt werden, während er entsprechende Bewegungen ausführt. Verfolgt werden dann die Bewegungen der Markierungen, um diese auf die entsprechenden Gelenke der Figur zu übertragen.

Fahrassistenzsysteme unterstützen den Autofahrer beim Halten der Spur. Es werden die seitlichen Fahrbahnmarkierungen erfasst und verfolgt, deren Überschreitung dann dem Fahrer

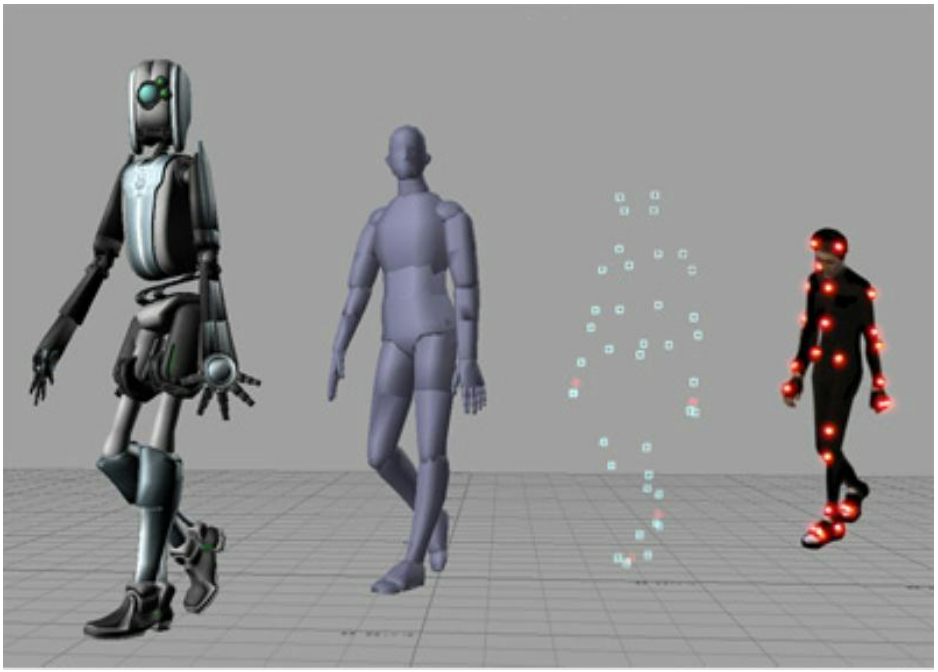

Abbildung 1.2.: Bewegungsanimation; Quelle: Wikipedia Motiontracking

signalisiert wird. Mittlerweile können dem Fahrer mithilfe von Trackingfunktionen sogar schon Informationen von Verkehrsschildern (z. B. bei Geschwindigkeitsbegrenzungen) in Fahrzeug angezeigt werden.

Gemeinsames Problem all dieser Anwendungen ist die robuste Erfassung eines realen Objektes anhand einer zweidimensionalen Abbildung.

Bisher gibt es noch kein universelles Tracking-Verfahren, welches für jedes der genannten Beispiele einsetzbar wäre. Es werden für wechselnde Problemstellungen und Sichtverhältnisse jeweils spezifisch angepasste Algorithmen und unterschiedliche Methoden zur Aufbereitung der eingehenden Bilddaten und zum darauf basierenden Erkennen von Objekten eingesetzt.

In dieser Arbeit sollen verschiedene Methoden und Techniken der Objektverfolgung analysiert und ggf.verfeinert werden.

Die Basis hierfür bilden die Programmbibliotheken LTI-Lib und OpenCV, welche Methoden zur Bildverarbeitung, -erkennung und -verfolgung von Objekten zur Verfügung stellen. Die in diesen Bibliotheken enthaltenen Tracker KLT, CAMSHIFT und SIFT wurden für die Software Impresario zur Bildverarbeitung als Makro adaptiert und auf ihre Verwendbarkeit hin überprüft und getestet.

1.1. Ziel der Arbeit

Die geläufigsten Tracking-Algorithmen und Methoden erwiesen sich als anfällig bei wechselnden äußeren Einflüssen. Typische Störungen, wie z. B. unterschiedliche Lichtverhältnisse, wechselnde Hintergründe oder Verdeckung der zu suchenden Objekte, erschweren deren Erfassung in den Bildsequenzen.

Um eine bessere Stabilität zu gewährleisten, müssen Grundlagen geschaffen werden, die ein Objekt unabhängig von seiner Umgebung eindeutig in einer Bildsequenz identifizieren.

Basierend auf dieser Problematik werden Vergleiche bezüglich Robustheit und Nutzbarkeit aufgestellt. Testszenarien mit Bildsequenzen sollen Aufschluss für die Verwendbarkeit bekannter Tracking-Verfahren bezüglich verschiedener Kriterien wie Helligkeitsschwankungen, Skalierung und Rotation geben.

In Vorarbeiten untersuchte Methoden und Algorithmen zur Verfolgung von Objekten in Echtzeitsystemen lieferten keine fehlerfreien, universal einsetzbaren Lösungen. Aus diesem Grund handelt es sich bei dieser Arbeit um die Umsetzung bekannter Tracking-Lösungen, bei denen unter bestimmten Voraussetzungen die Fehlerrate möglicht niedrig ist.

Die Arbeit soll Aufschluss darüber geben, welches der drei untersuchten Tracking-Verfahren aufgrund bestimmter Objektcharakteristika einer Bildsequenz bzw. eines Videostreams die erfolgversprechendste Lösung darstellt, ohne unbedingt die genauen Funktionsweisen der verschiedenen Methoden und Algorithmen im Detail zu analysieren.

1.2. Aufbau der Arbeit

In Kapitel 2 werden zunächst die theoretischen Grundlagen, die zum Verständnis der Implementierung der verschiedenen Tracker notwendig sind, gegeben. In den darauf folgenden Kapiteln werden die drei Algorithmen CAMSHIFT [Kapitel 3], KLT [Kapitel 4] und SIFT [Kapitel 5] beschrieben, bevor dann in Kapitel 6 deren Implementierung dargestellt wird.

In Kapitel 7 werden schließlich die Ergebnisse zusammengefasst und bewertet. Das Fazit dieser Arbeit findet sich in Kapitel 8.

2. Grundlagen

In diesem Kapitel geht es um die Schaffung von Grundlagen und Definitionen der hier angewandten Bildverarbeitung und Algorithmen.

Die Bildverarbeitung beinhaltet das Verändern der gelieferten Bilder mit Filtern, Masken und Funktionen zur Herausarbeitung und Verstärkung verschiedenster Bildinformationen.

So ist es möglich, ein Farbbild mithilfe von Filtern in einzelne Farbkanäle aufzuteilen oder mithilfe von Masken Bildstörungen oder Rauschen zu entfernen. Über weitere mathematische Funktionen (s. u. Kapitel 2.2) ist ein Hervorheben von Konturen und Kanten bis hin zur Ermittlung verschiedenster Merkmale möglich.

2.1. Farbräume

Die für das Tracking genutzten eingehenden Bilder sind digital. Die sogenannten Images bestehen aus einer Angabe über die Bildgröße in Pixel (z. B. 600 x 480, 800 x 600, 1024 x 768). Diese Größenangabe gibt Auskunft über die Anzahl der farbigen Pixel in x- und y-Richtung. Die Koordinaten der Pixel werden von der linken oberen Ecke ausgehend angegeben.

Für jedes Pixel ist eine Farbinformation hinterlegt, welche sich aus mehreren Werten zusammensetzen kann.

Grauwertbilder besitzen Werte zwischen 0 (schwarz) und 255 (weiß).

Werden die Farbinformationen in Form des RGB-Farbmodells [1] angegeben, bestehen diese Werte aus drei Zahlen zwischen 0 und 255, welche die jeweilige Intensität der Farben Rot, Grün und Blau wiedergeben (s.Abb. 2.1).

Als Drittes wird das HSV-Farbmodell [2] verwendet (s. Abb. 2.2). Auch hier definieren drei Zahlen die Farbe eines Pixels:

[1] R = Rot, G = Grün, B = Blau.
[2] H=hue, S=starvation, V=value

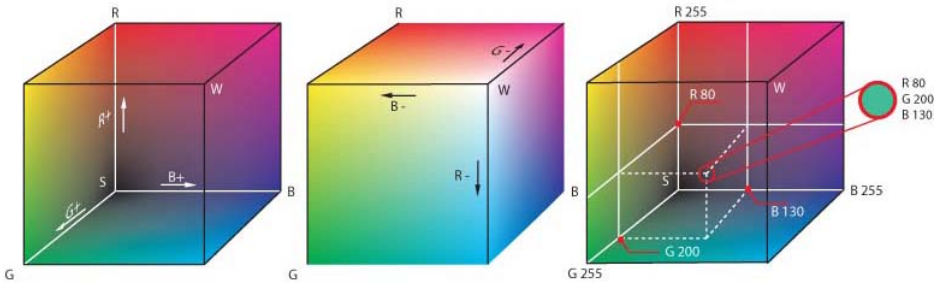

Abbildung 2.1.: RGB-Farbraum; Quelle: Wikipedia RGB-Farbraum

- Farbton H (Winkel zwischen 0° und 360°)
- Sättigung S (prozentuale Angabe 0% bis 100%)
- Helligkeit V (prozentuale Angabe 0% bis 100%)

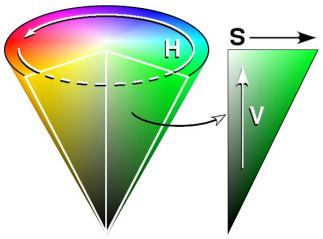

Abbildung 2.2.: HSV-Farbraum; Quelle: Wikipedia HSV-Farbraum

Das HSV-Farbmodell eignet sich besonders für farbbasierte Bildverarbeitung, da sich die Werte für den Farbton und die Sättigung bei Helligkeitsschwankungen in einem Viedeostream nicht verändern.

2.2. Bildfilter und -masken

Dieser Abschnitt vermittelt eine Übersicht über die im späteren Verlauf angewandten Bildfilter und -masken. Die einzelnen Algorithmen werden vorgestellt und ihre Auswirkung anhand der Abbildung 2.3 dargestellt.

Abbildung 2.3.: Originalbild

2.2.1. Gauß-Filter

Der Gauß-Filter wird angewendet zur Glättung oder Weichzeichnung des Bildinhaltes. Dabei gehen kleine Strukturen und Bildrauschen verloren. Anhand einer $n \times m$-Matrix (n;m ist ungerade) und einem Parameter σ als Varianz- oder Gewichtungsfunktion fließt der Betrag der Nachbarpixel in den Grauwert $G(x, y)$ des aktuellen Pixels ein (s. Abb. 2.4).

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2 + y^2}{2\sigma^2}} \tag{2.1}$$

.

Abbildung 2.4.: Beispiele Gauß-Filter

Eine weitere Anwendungsmöglichkeit des Gauß-Filters ist die Minimierung der Auflösung eines Bildes. Hierbei wird eine $n \times n$-Matrix nicht pixelweise verschoben, sondern jeweils um ihre Breite n. Der Farbwert des zu berechnenden Pixels wird genau wie in der die Bildgröße

erhaltenden Version bestimmt, das Ergebnisbild ist somit um den Faktor n kleiner als das Ausgangsbild (s. Abb. 2.5).

Abbildung 2.5.: Gauß-Faltung

2.2.2. Sobel-Filter

Der Sobel-Filter ist ein sehr variabel einsetzbarer Filter zur Detektion von Diskontinuitäten. Da dieser Filter nur mit Grauwerten funktioniert, ist eine Umsetzung von Farbbildern in Grauwerte nötig. Mithilfe von Gradientenoperatoren können die Flanken von Kanten im Grauwertgebirge eines Bildes maximiert werden. Hierbei ist der Gradient ein Vektor, der die Richtung und Stärke einer Flanke im Grauwertgebirge angibt.

Der Sobel-Operator faltet mittels folgender $3x3$-Matrixen das Originalbild und erzeugt dadurch ein Gradientenbild in x- und y-Richtung (s. Abb. 2.6).

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.2}$$

Der richtungsunabhängige Betrag der Gradienten wird durch

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.3}$$

berechnet, ihre Richtung mit

$$\theta = \arctan\left(\frac{G_y}{G_x}\right). \tag{2.4}$$

Dabei entspricht der Wert $\theta = 0$ einer vertikalen Kante und positive Winkel einer Verdrehung gegen den Uhrzeigersinn (s. Abb. 2.7).

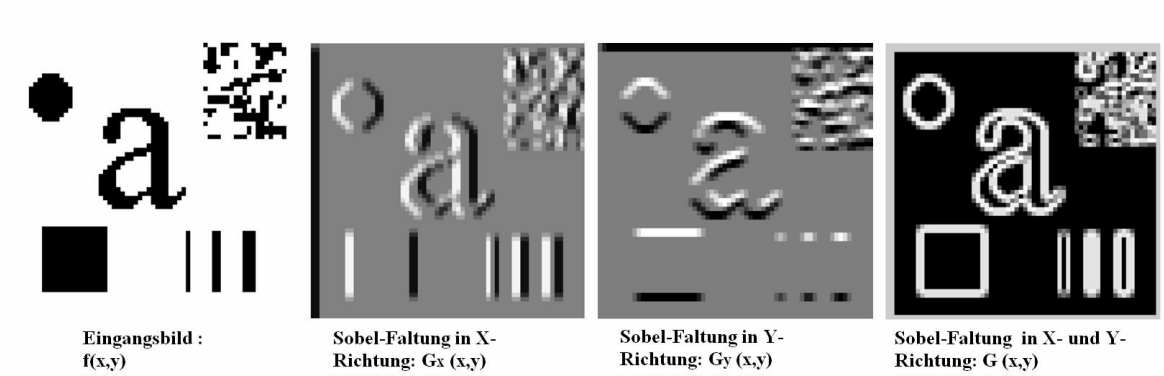

Abbildung 2.6.: Anwendung der Sobel-Maske in x- und y-Richtung; Quelle: Meisel (2006)

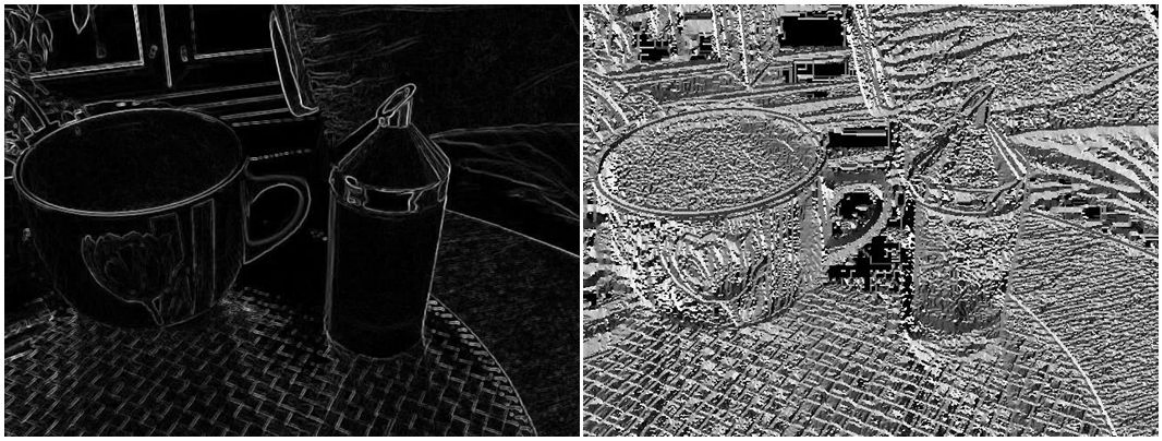

Abbildung 2.7.: Betrag und Richtung des Sobel-Filters

2.2.3. Canny-Filter

Der Cany-Filter basiert auf dem Sobel-Filter und erweitert diesen zur Bestimmung von Kanten innerhalb des Bildes, die genau ein Pixel stark sind. Dabei wird nach der Bestimmung des Gradientenbildes durch den Sobel-Filter jedes Pixel mit seinen acht Nachbarn verglichen. Ist das aktuelle Pixel ein lokales Maximum oder ähnelt sein Wert den Nachbarpixeln in Kantenrichtung, bleibt sein Grauwert erhalten. Andernfalls wird dieser auf null (entspricht der Farbe Schwarz) gesetzt.

Durch Einführung von zwei Schwellwerten $T_1 < T_2$ und Scannen des Bildes nach einem Wert größer als T_2 kann der Richtung einer Kante bis zum Unterschreiten des Wertes T_1 gefolgt und deren Pixel auf 255 (weiß) gesetzt werden. Alle anderen Pixel erhalten den Wert null (s. Abb. 2.8).

T2= 0.20 T1= T2*0.5 T2= 0.15 T1= T2*0.5 T2= 0.04 T1= T2*0.5

Abbildung 2.8.: Canny-Filter

2.3. Markanter Punkt

Markante Punkte werden benötigt zur Bildanalyse. Sie markieren genau ein Pixel innerhalb eines Bildes, das einen Extrempunkt nach unterschiedlichen Gesichtspunkten beschreibt.

Diese markanten Bildpunkte bezeichnen Koordinaten von Kanten, Ecken und Schwerpunkte, welche jeweils zu einem Bild berechnet werden. Für die weitere Verarbeitung ist zusätzlich eine hohe Unterscheidungsfähigkeit zu anderen Punkten notwendig.

Bei der Bestimmung dieser markanten Punkte kommen verschiedene Operatoren zum Einsatz, wobei die berechneten Punkte nicht immer signifikant für das menschliche Auge sind, sondern als rein mathematisches Ergebnis verstanden werden müssen (s. Abb. 2.9).

Für jeden markanten Punkt wird zudem ein Merkmalsvektor bestimmt, der diesen eindeutig von anderen Punkten unterscheidet (s. Abb. 2.9 und 2.10).

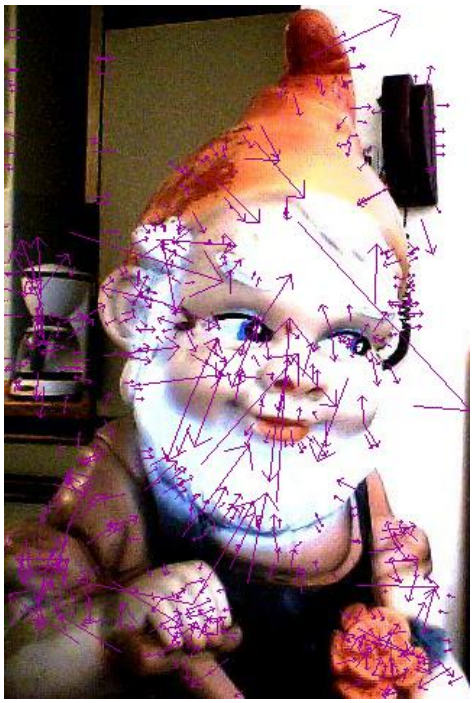

Abbildung 2.9.: Markante Punkte Zwerg

Abbildung 2.10.: Markante Punkte Tasse mit Zucker

2.4. Merkmale markanter Punkte

Die Robustheit von Tracking-Algorithmen zur Verfolgung eines Objektes hängt in erster Linie von der Definition der Merkmale für die markanten Punkte ab.

Das Ergebnis der auf die Erkennung von Merkmalen folgenden Operationen ist direkt von der Güte der Merkmale abhängig. Sind die Merkmale schlecht gewählt, so werden mit den darauf folgenden Operationen keine verwendbaren Ergebnisse zu erzielen sein.

Die hier verwendeten Tracking-Algorithmen basieren auf zwei grundsätzlich unterschiedlichen Ansätzen zur Definition dieser Merkmale:

1. Merkmale, welche das Verhältnis mehrerer markanter Punkte zueinander charakterisieren
2. Merkmale, welche einen einzelnen markanten Punkt charakterisieren

Bei der Verwendung des ersten Ansatzes wird die Menge der identifizierten markanten Punkte eines Bildes auf ein Referenzmuster hin untersucht, welches durch die Anordnung einiger Punkte gebildet wird.

Hier ist es möglich, ein dreidimensionales Referenzmerkmal zu erstellen und bei der Suche nach Korrespondenzen innerhalb der markanten Punkte eines Bildes dieses Referenzmerkmal zu rotieren und zu skalieren, um es in dem zu durchsuchenden Bild auch in verschiedenen Lagen und Entfernungen identifizieren zu können.

Beim zweiten Ansatz wird die Bildumgebung eines markanten Punktes in dessen Definition miteinbezogen. Helligkeitswechsel, Farben, Richtungen und Konturverläufe in der Pixelnachbarschaft von einem jeweiligen markanten Punkt werden als dessen Merkmale identifiziert.

Beiden Definitionen von Merkmalen ist gemeinsam, dass sie in Zahlenwerten gespeichert werden und somit eine Ergänzung der Pixelkoordinaten eines markanten Punktes darstellen. Aus diesem Grund werden die Merkmale in der Bildverarbeitung auch als Zahlenvektoren, Raumwertinformationen oder Deskriptoren bezeichnet.

Optimalerweise sollten Merkmale Invarianzen gegenüber Rotation, Skalierung, Helligkeit, Kontrast und wechselnden Perspektiven aufweisen.

2.5. Korrespondenzen

Werden in unterschiedlichen Bildern einer Sequenz oder Bildern aus unterschiedlicher Perspektive jeweils markante Punkte extrahiert und sollen die von den markanten Punkten markierten Objekte der realen Welt mit deren Hilfe einander zugewiesen werden, werden diese als Korrespondenzen oder korrespondierende Bildpunkte bezeichnet.

Um korrespondierende Punkte eindeutig einander zuweisen zu können, müssen möglichst invariante, dezidierte Merkmale zu jedem markanten Punkt bestimmt werden. Nach diesen Merkmalen kann somit in der Menge der markanten Punkten eines weiteren Bildes gesucht, und die entsprechenden Punkte können einander zugewiesen werden (s. Abb. 2.11 und 2.12).

In unterschiedlichen Kameraansichten eines Objektes sollen dessen markante Punkte ähnliche Merkmale aufweisen, um eine Zuordnung bei geringer Fehlerrate zu ermöglichen.

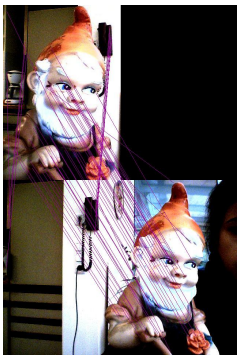

Abbildung 2.11.: Korrespondenzen Zwerg

Abbildung 2.12.: Korrespondenzen Tasse mit Zucker

2.6. Objekte und ihr Abbild

Objekte sind definierbar als linienförmige, flächige oder räumliche Abbilder von Ausschnitten der Welt. Sie können starr wie ein Bauklotz, stückweise starr wie ein Industrieroboter oder formvariabel wie Tuch oder auch ein Mensch sein.

Bei einer perspektivischen Transformation durch die Kamera auf ein zweidimensionales Abbild soll von diesem Objektabbild auf das Objekt selbst geschlossen werden können.

Zu diesem Zweck werden ein oder mehrere Merkmale definiert, die ein Objekt beschreiben und identifizierbar machen.

Typische Merkmale für Objekte werden aus Korrespondenzen unterschiedlicher markanter Punkte erstellt.

Wünschenswert ist eine invariante Erstellung von Merkmalen, welche auch hier bei Rotation, Translation und Skalierung des Objektes eine eindeutige Zuordnung ermöglicht.

In der Praxis wird die Zuordnung des Objektes erschwert durch evtl. Teilabdeckungen durch Gegenstände im Vordergrund oder unberücksichtigte Perspektiven des Objektes bei dessen Abbildung, wie z. B. dessen Rück- oder Unterseite.

Es können nur in der Abbildung sichtbare Merkmale des Objektes identifiziert werden.

2.7. Tracking

Tracking ist die englische Bezeichnung der Verfolgung von Objekten in Bild- oder Videosequenzen und fasst sämtliche Bildbe- und -verarbeitungsschritte, die der Verfolgung von Objekten dienen, in einem Begriff zusammen.

Technischer ausgedrückt, ist Tracking die Extraktion von Zustandsinformationen zur Identifizierung markanter Punkte und deren Merkmale über eine Folge von Bildern zur Erfassung räumlicher Objekte in der Umgebung.

Auf Objekte bezogen ist Tracking die Verfolgung von Objektabbildern in einer Bildsequenz anhand von Objektmerkmalen und evtl. anhand von geschätzten Bewegungsparametern.

Diese extrahierten Zustandsinformationen können Angaben zur Position, Ausrichtung, etwaigen Geschwindigkeit und Beschleunigung des zu trackenden Objektes liefern.

Die Genauigkeit, mit der die Zustandsinformationen extrahiert werden können, hängt von dem verwendeten Tracking-Algorithmus und dessen Methoden der Merkmalsbestimmung ab.

Beim Tracking gibt es unterschiedliche Voraussetzungen, welche vorher festgelegt werden und entscheidend für den Lösungsansatz des speziellen Problems sind:

- Die Kameraperspektive ist konstant, Objekte bewegen sich vor gleichem Hintergrund.
- Die Kamera bewegt sich bezüglich des zu trackenden Objektes.
- Referenzmerkmale werden aus dem ersten Bild erstellt und in weiteren Bildern in dessen lokaler Umgebung gesucht (statisches Tracking).
- Merkmale werden anhand von Unterschieden aufeinander folgender Bilder erstellt (dynamisches Tracking).

Abbildung 2.13.: Geschichtverfolgung innerhalb eines Suchfensters einer Viedeosequenz

Bei allen drei hier untersuchten Tracking-Algorithmen ist folgender Bearbeitungsabblauf identisch:

Die eingehenden Bilddaten werden in Abhängigkeit von dem jeweiligen Algorithmus aufbereitet, um unterschiedliche Eigenschaften der Abbildung zu verstärken und zu segmentieren sowie Bildinformationen auf die benötigten Details zu reduzieren.

Als Nächstes erfolgt die Bestimmung von markanten Punkten und der dazugehörigen Merkmale unter Anwendung algorithmusabhängiger mathematischer Funktionen.

Im letzten Schritt erfolgt die Analyse auf Korrespondenz von markanten Punkten oder Objekten zu vorher definierten Referenzen.

3. CAMSHIFT-Algorithmus

Der CAMSHIFT-Algorithmus (**C**ontinuously **A**daptive **M**ean-Shift) wird in Bradski (1998a) und Bradski (1998b) vorgestellt und basiert auf einer Modifikation des Mean-Shift-Algorithmus (engl.: mean = Schwerpunkt), der von Cheng (1995) beschrieben wird.

Von einem zu trackenden Objekt wird eine Farbreferenztabbelle erstellt, mit welcher die eingehenden Bildsequenzen gefiltert und alle Farbwerte eliminiert werden, die nicht in der Tabelle vorkommen.

Der Mean-Shift-Algorithmus definiert in dem jeweils aktuellen Bild der Sequenz einen Ausschnitt als Suchfenster. Iterativ wird innerhalb dieses Suchfensters die Verteilung der Farbwerte aus der Referenztabelle betrachtet und für diese ein Schwerpunkt berechnet. Es folgt die Verschiebung des Suchfensters in Richtung des ermittelten Schwerpunktes, und der Durchlauf beginnt von Neuem. Abbruchbedingung für die Iteration ist eine Übereinstimmung des Mittelpunktes des Suchfensters mit dem berechneten Schwerpunkt der Farbverteilung. Damit ist eine Konvergenz erreicht (s. Abb. 3.1).

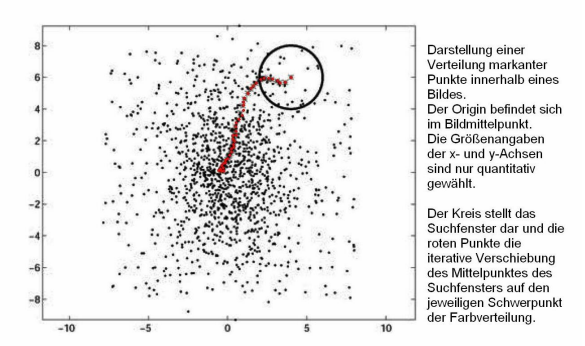

Abbildung 3.1.: Mean-Shift-Verfahren; Quelle: Comaniciu und Meer (1999)

Der Schwachpunkt des Mean-Shift-Algorithmus ist die statische Größe des Suchfensters. Größenänderungen in der Abbildung des zu trackenden Objektes können nicht berücksichtigt werden.

Der CAMSHIFT-Algorithmus erweitert den Mean-Shift-Algorithmus um die Funktionalität der automatischen Größenanpassung des Suchfensters. Nach Konvergenz des Mean-Shift-

Algorithmus werden die Randbereiche des Suchfensters auf vorkommende Farbwerte der Referenztabelle hin betrachtet, und die Suchfenstergröße wird entsprechend angepasst.

Entwickelt wurde der CAMSHIFT-Algorithmus für die Erkennung von Gesichtern. Maßgeblich für deren Identifizierung bei diesem Algorithmus ist eine Farbreferenztabelle im HSV-Farbraum, welche sämtliche möglichen Gesichtsfarben beinhaltet.

Beim Austausch oder bei der Modifizierung der Farbreferenztabelle können jedoch auch beliebige Objekte anderer Farben identifiziert werden.

Mathematischer Ablauf:

Die Formel für die Berechnung der relativen Häufigkeitswerte $H_{rel}(x, y)$ eines Farbwertes aus dem Histogramm der Farbverteilung innerhalb des Suchfensters lautet:

$$H_{rel}\,(x,\ y) = \frac{H\,(h,\ c)}{\sum H} \quad \forall\, x,\ y\ \in\ Suchfenster \tag{3.1}$$

$H\,(h,\ c)$ stellt die Häufigkeit eines Hue-Farbwertes innerhalb des Suchfensters dar. $\sum H$ ist die Summe der Häufigkeitswerte aus dem Suchfenster.

Die Abbildung 3.2 zeigt einen schematischen Ablauf der relativen Farbwerteberechnung aus den Histogrammen.

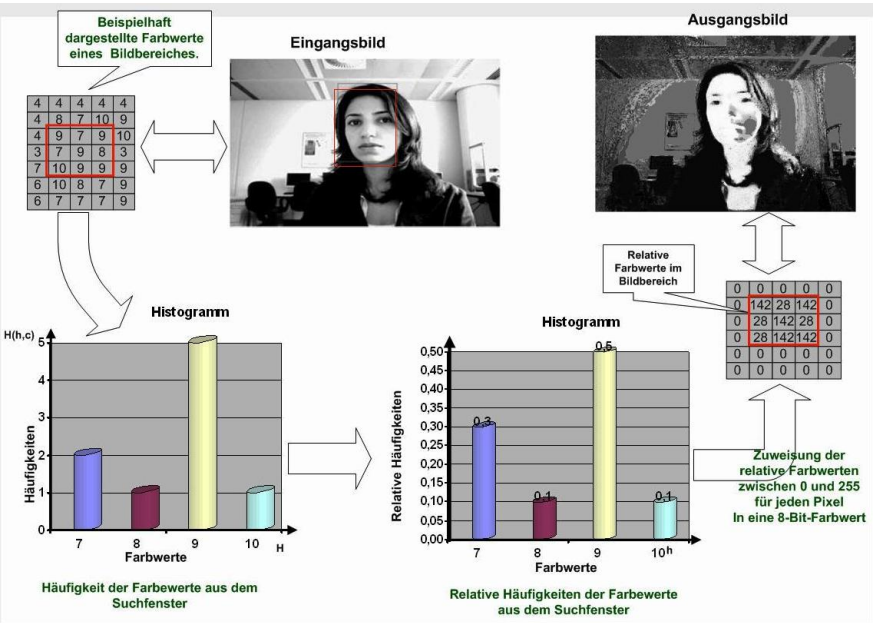

Abbildung 3.2.: Erstellung der Histogramme aus den Farbwerten des Objektes in dem Bildbereich

Mit dem „nullten Moment“ M_{00} wird die Anzahl der zum Objekt gehörenden Pixel innerhalb des Suchfensters berechnet:

$$M_{00} = \sum_x \sum_y H_{rel}(x,\ y) \quad \forall\, x,\ y \in Suchfenster \tag{3.2}$$

Zur Bestimmung der Verschiebungsrichtung des Suchfensters, findet eine Gewichtung der Farbwerte mit den Momenten M_{10} und M_{01} statt.

Aus den Momenten M_{10}, M_{01} und Moment M_{00} lassen sich die Koordinaten des Schwerpunktes $S(x_c,\ y_c)$ der zum Objekt gehörenden Pixel innerhalb des Suchfensters berechnen.

Gewichtung der Farbwerte in x-Richtung.

$$M_{10} = \sum_x \sum_y x\ \ H_{rel}(x,\ y) \tag{3.3}$$

Gewichtung der Farbwerte in y-Richtung.

$$M_{01} = \sum_x \sum_y y\ \ H_{rel}(x,\ y) \tag{3.4}$$

Berechnung der Koordinaten des Schwerpunktes $S(x_c,\ y_c)$ der Pixelverteilung innerhalb des Suchfensters.

$$x_c = \frac{M_{10}}{M_{00}} \qquad y_c = \frac{M_{01}}{M_{00}} \tag{3.5}$$

Der Mittelpunkt des Suchfensters wird auf diesen Schwerpunkt $S(x_c,\ y_c)$ verschoben.

Die Iteration der Verschiebung des Mittelpunkts des Suchfensters findet so lange statt, bis der berechnete Schwerpunkt mit dem Mittelpunkt des Suchfensters konvergiert.

An dieser Stelle wird der CAMSHIFT-Algorithmus in den Mean-Shift-Algorithmus integriert, um die Größe des Suchfensters adaptiv zu verändern.

Die Größenanpassung der Seitenlänge s des Suchfensters wird mit einem relativen Farbwert von 0 bis 255 berechnet mit:

$$s = 2\sqrt{\frac{M_{00}}{256}} \tag{3.6}$$

Die Länge s wird auf $1.2\ *\ s$ gesetzt, um die komplette Abdeckung des Objektes zu erreichen.

Zudem gibt es die Möglichkeit, mithilfe der „zweiten Momente“ die Ausrichtung, Breite und Länge eines Objektes zu berechnen.

Die Formel für die Berechnung der „zweiten Momente“ teilt sich auf in eine Zeilen- und Spaltenoperation:

$$M_{20} = \sum_x \sum_y x^2 \; H_{rel}\,(x,\; y) \qquad M_{02} = \sum_x \sum_y y^2 \; H_{rel}\,(x,\; y) \tag{3.7}$$

Die Ausrichtung der Hauptachsen θ des Objektes innerhalb des Suchfensters kann dann angegeben werden mit:

$$\theta = \frac{arctan\left(\frac{2\left(\frac{M_{11}}{M_{00}} - x_c\; y_c\right)}{\left(\frac{M_{20}}{M_{00}} - x_c^2\right) - \left(\frac{M_{02}}{M_{00}} - y_c^2\right)}\right)}{2} \tag{3.8}$$

Für eine übersichtliche Darstellung der Berechnungen von Länge, Breite und Ausrichtung werden die temporären Variablen a, b und c eingeführt.

$$a = \frac{M_{20}}{M_{00}} - x_c^2 \tag{3.9}$$

$$b = 2\left(\frac{M_{11}}{M_{00}} - x_c\; y_c\right) \tag{3.10}$$

$$c = \frac{M_{02}}{M_{00}} - y_c^2 \tag{3.11}$$

Die Formel für die Ausrichtung der Hauptachsen θ folgt so in verkürzter Form mit:

$$\theta = \frac{1}{2}\, arctan\, \frac{b}{a - c} \tag{3.12}$$

Daraus ergibt sich für die Länge l und die Breite w des Objektes:

$$l = \sqrt{\frac{(a + b) + \sqrt{b^2 + (a - c)^2}}{2}} \qquad w = \sqrt{\frac{(a + b) - \sqrt{b^2 + (a - c)^2}}{2}} \tag{3.13}$$

Mithilfe der zweiten Momente lassen sich die Länge, die Breite und der Winkel einer 2-D-Position (bzw. des verfolgten Objekts) der Verteilungsdichte berechnen.

Beim Einsetzen der CAMSHIFT-Tracker zur Erkennung von Gesichtern liefert die Berechnung von θ die Neigung des Kopfes, l die Länge und w die Breite des Gesichtes in der Bildsequenz.

Anhand der Informationen der „zweiten Momente“ kann die Fläche des Objektes wie folgt berechnet werden:

$$A = \frac{1}{2} l * w * \pi \tag{3.14}$$

Abbildung 3.3.: Camshift Filterung mit Farbreferenztabelle

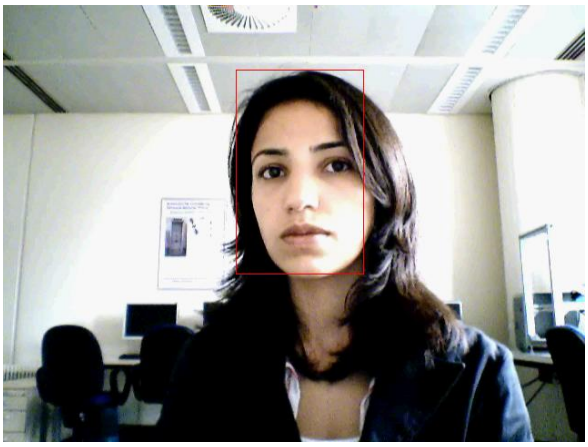

Abbildung 3.4.: CAMSHIFT mit Trackingfenster

4. KLT-Algorithmus

Der Name des KLT-Algorithmus setzt sich aus den Initialen seiner Entwickler Kanade, Lucas und Tomasi zusammen.

Die Theorie zur Entwicklung des Algorithmus wurde von Bruce D. Lucas und Takeo Kanade im Jahre 1981 aufgestellt und beinhaltet die Definition von Merkmalen und deren Verfolgung in einer Bildsequenz Lucas und Kanade (1981). Im Jahr 1991 implementierte T. Kanade mit C. Tomasi eine erste auf dieser Theorie basierende Lösung Tomasi und Kanade (April 1991).

Der hier verwendete KLT-Tracker geht von einem fixen Kamerastandpunkt und sich langsam im Raum bewegenden Objekten aus. Er betrachtet die Veränderungen in der Abfolge der Bilder und berechnet, basierend auf diesen Veränderungen, geeignete markante Punkte für die Verfolgung von Objekten.

Im Jahre 2002 führte Jean-Yves Bouguet (Bouguet (2002)) einen pyramidialen Ansatz zur Findung von markanten Punkten ein. Die Bildpyramide dient hierbei der Beschleunigung des Algorithmus und ermöglicht die Berechnung größerer Verschiebungen von Objekten in der Bildsequenz.

Die Theorie besagt, dass die Differenz von einem zum nächsten Bild durch Translation von Bildobjekten erreicht werden kann, wodurch es möglich ist, den unveränderlichen Hintergrund in den Berechnungen zu ignorieren.

Wird das aktuelle Bild von dem vorherigen Bild subtrahiert, bleiben nur die veränderten Pixel als Umrisse der sich bewegenden Objekte erhalten, für welche sich ein jeweiliger Verschiebungsvektor aus der Abfolge der Bildsequenz berechnen lässt (s. Abb. 4.2).

Die Bewegung zwischen zwei Bildsequenzen wird mit dem sogenannten Optical-Flow-Verfahren beschrieben (siehe Tomasi und Kanade (April 1991)).

Die Auswahl von markanten Punkten erfolgt anhand einer $2x2$-Koeffizienten-Matrix G, welche, basierend auf dem Harris-Corner-Detektor[1], signifikante Ecken in den Umrissen der Bilddifferenzen ermittelt.

[1] Zur genaueren Beschreibung des Harris-Corner-Detektors s. ...

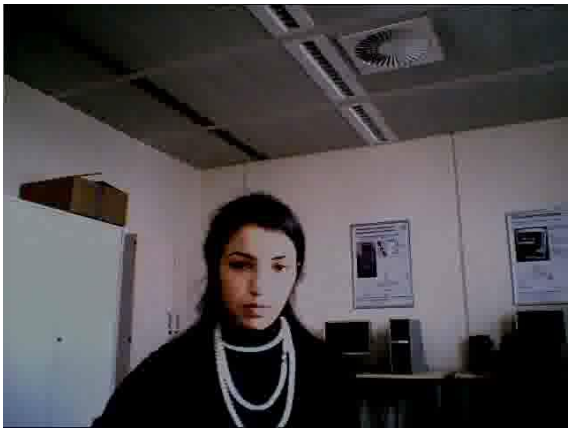

Abbildung 4.1.: KLT-Eingangsbild

Abbildung 4.2.: KLT-Differenzbild

$$G = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{4.1}$$

Markante Punkte werden in Bildregionen mit inhomogenen Texturen extrahiert und dann als signifikant erachtet, wenn der kleinste Eigenwert λ über dem Grenzwert des Bildrauschens liegt und ein Mindestabstand zu weiteren markanten Punkten gegeben ist.

Abbildung 4.3.: Eigenwerte der Koeffizientenmatrix; Quelle: Pitzer (Juni 2004)

Dieser Schwellwert wird durch Messung der Intensität bestimmt (konstante bzw. durchschnittliche Helligkeit). Die untere Schwelle gibt ein Maß aus einer hellen Bildregion an, die obere wird bestimmt durch eine Region, in der Kanten vorhanden sind (s. Abb. 4.4).

Werden diese Eigenwerte in ein Diagramm (s. Abb. 4.5) eingetragen, ist der Bereich für geeignete markante Punkte zu bestimmen.

Sind λ_1 und λ_2 klein, entspricht dies einer homogenen Fläche. Ein unidirektionales, nicht erkennbares Muster spiegelt sich in unterschiedlichen Werten von λ_1 und λ_2 wider. Sind beide Werte λ_1 und λ_2 groß, enthält das Merkmalsfenster eine Ecke.

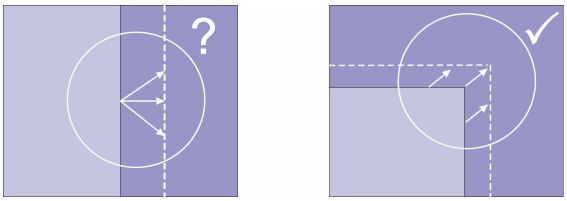

Abbildung 4.4.: Bildregion; Quelle: Pitzer (Juni 2004)

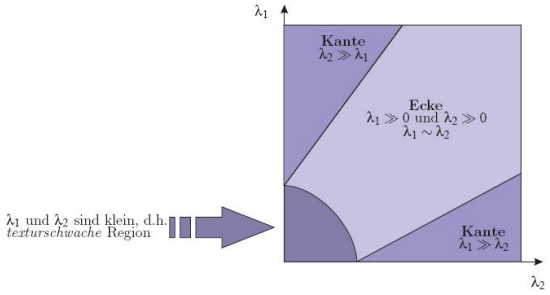

Abbildung 4.5.: Eigenwerte; Quelle: Pitzer (Juni 2004)

Nachdem alle Punkte auf ihre Eignung als markanter Punkt hin untersucht wurden, sind die Merkmale aus den aufeinander folgenden Bildern zu erhalten.

Die Koordinaten der markanten Punkte werden aus dem ersten Bild entnommen und in dem nachfolgenden Bild verfolgt. Die Merkmale aus der ersten Bildregion werden mit denen in der zweiten Bildregion verglichen.

Abbildung 4.6 zeigt die korrespondierenden markanten Punkte mit roten Kreuzen und nicht zugeordnete markante Punkte aus dem Nachfolgebild mit weißen Punkten.

Abbildung 4.6.: KLT-Punkte

Abbildung 4.7.: KLT-Ausgabe

5. SIFT-Algorithmus

SIFT– **S**cale **I**nvariant **F**eature **T**ransformation

Die skalierungsinvariante Merkmalstransformationen ist einer der jüngst entwickelten Algorithmen zur Suche von markanten Punkten in der Bildverarbeitung. Die Grundlage für die Theorie stellte David G. Lowe Lowe (1999) vor und brachte diese in überarbeiteter Version Lowe (2004) im Jahr 2004 auf den Stand der neusten Entwicklung der Bildverarbeitung.

Die Entwicklung des SIFT-Algorithmus wird aufgrund von dessen Robustheit bezüglich der Auffindung von Objekten in Bildsequenzen als ein Durchbruch im Bereich des Trackings betrachtet.

Der SIFT-Algorithmus benutzt, im Unterschied zu den beiden anderen vorgestellten, die Variante der Merkmalsdefinition für einen einzelnen markanten Punkt.

Ein Beispiel für diesen Einsatz des Algorithmus ist die automatische Generierung von Panoramabildern. Die Bilder werden aufgrund ihrer Positionierung im gesamten Panoramabild sortiert und durch erkannte SIFT-Merkmale an den zusammengehörenden Stellen übereinandergeblendet (s. Abb. 5.1 und deren Panoramaabbildung 5.2).

Der SIFT-Algorithmus definiert ein Merkmal als einen markanten Punkt mit Informationen über dessen Umgebung (Pixelnachbarschaft), welche in einem 128-dimensionalen realen Vektor (dem Deskriptor) zusammengefasst sind.

Diese Merkmale sind invariant gegenüber Skalierung, Drehung und Verschiebung des markanten Punktes. Darüber hinaus wird durch den Deskriptor ebenfalls eine Invarianz hinsichtlich unterschiedlicher Beleuchtung eines Objektes erreicht.

Für jeden markanten Punkt wird zudem dessen Orientierung, Skalierung und Position bestimmt und mit den markanten Punkten weiterer Bilder über ihre euklidische Distanz verglichen.

Die extrahierten Merkmale eines markanten Punktes besitzen durch diese Eigenschaften einen hohen Wiedererkennungswert und machen diesen Algorithmus zu einer sehr effizienten Lösung für die Objekterkennung und -verfolgung in Bildsequenzen. Durch die mithilfe eines Computers schnell ausführbaren nötigen Berechnungen ist die Verwendung dieses Trackings nahezu in Echtzeitsystemen möglich.

Abbildung 5.1.: Quellbilder

Abbildung 5.2.: Panorama der Quellbilder

Die Bildverarbeitung des SIFT-Algorithmus basiert auf der mehrfachen Verwendung der Gauß-Funktion.

Zunächst wird eine mit der Gauß-Funktion geglättete Kopie 1 des Eingangsbildes erstellt. Mit dieser Kopie 1 wird unter erneuter Anwendung der Gauß-Funktion eine Kopie 2 erstellt. Dieser Vorgang wird wiederholt, bis das Bild entweder nur noch ein Pixel breit ist oder aber ein zuvor definiertes Abbruchkriterium erreicht wird.

Im nächsten Schritt werden die jeweils benachbarten Bilder voneinander abgezogen und bilden so ein Gauß-Differenzbild (engl: Difference of Gaussian), im Weiteren als DOG-Bild bezeichnet.

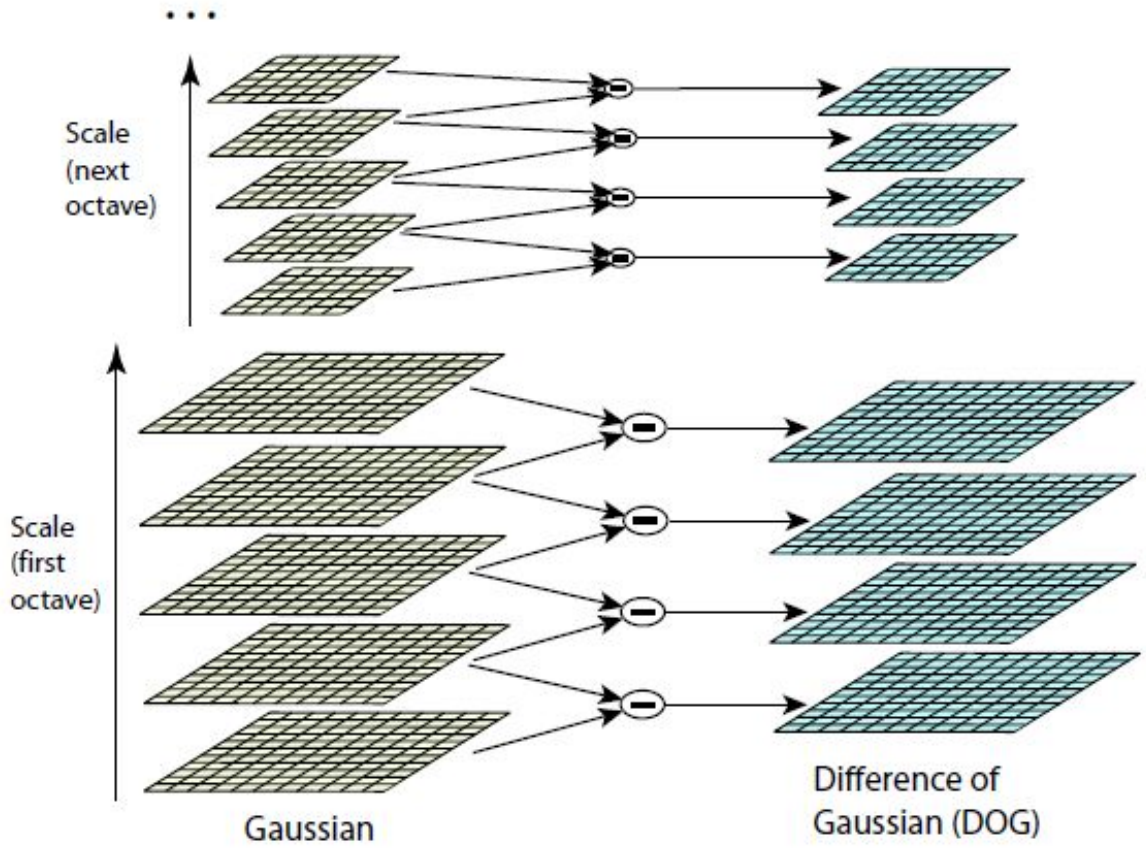

Abbildung 5.3.: Darstellung der SIFT-Oktaven; Quelle: Lowe (2004)

Die bis hierher erhaltenen Kopien abnehmender Schärfe und deren DOG-Bilder werden von Lowe als „Scale“ S_1 bis S_n bezeichnet und bilden gemeinsam die „erste Oktave“.

Das mit der Gauß-Funktion gefaltete und somit in der Auflösung reduzierte Ausgangsbild bildet „Scale“ S_1 der „zweiten Oktave“. Innerhalb dieser „zweiten Oktave“ werden erneut „Scale“ S_1 bis S_n und deren DOG-Bilder erzeugt (s. Abb. 5.3).

Nach dieser Bildverarbeitung werden die DOG-Bilder einer jeweiligen Oktave nach lokalen Extrema durchsucht, welche als markante Punkte infrage kommen könnten.

Zu diesem Zweck wird jedes Pixel des DOG-Bildes der „Scale“ S_n mit seinen angrenzenden acht Nachbarpixeln verglichen sowie mit den jeweils neun angrenzenden Pixeln der DOG-Bilder „Scale“ S_{n-1} und „Scale“ S_{n+1} (s. Abb. 5.4). Bildet das zu untersuchende Pixel im Vergleich zu seinen angrenzenden 26 Nachbarn ein lokales Maximum oder Minimum, ist es ein potenzieller markanter Punkt.

Da die Auflösung der DOG-Bilder in den Oktaven 1 + n niedriger ist als im Ausgangsbild,

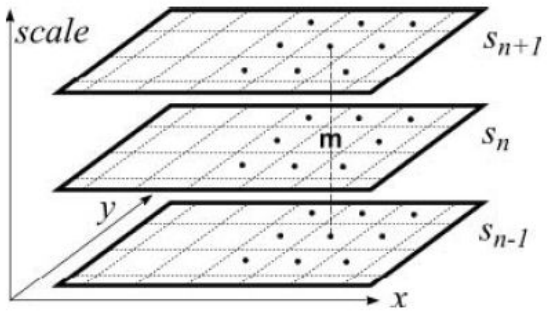

Abbildung 5.4.: Suche nach Extrema in den DOG-Bildern; Quelle:Gremse (2005)

wird die Position eines Extrems aus den DOG-Bildern interpoliert zu einer Koordinate $(x,\ y)$ des Ausgangsbildes.

Basierend auf den Koordinaten des Ausgangsbildes wird nun der Deskriptor für den Vergleich von Merkmalen anhand der umgebenden Pixel bestimmt.

Der Deskriptor setzt sich zusammen aus 128 Vektoren und wird unter Verwendung eines Histogrammes der lokalen Gradientenorientierung um die Koordinaten des Extrems herum berechnet (s. Abb. 5.5).

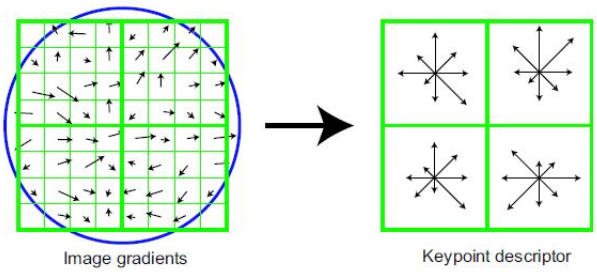

Abbildung 5.5.: Bestimmung des Deskriptors auf Basis von Gradienten; Quelle:Lowe (2004)

Bevor der Deskriptor zu jedem markanten Punkt berechnet werden kann, wird ein potenzieller markanter Punkt, der auf einer Kante liegt, mit der Hesse-Matrix H eliminiert.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{5.1}$$

Dabei werden nicht die Eigenwerte der Hesse-Matrix betrachtet, sondern nur ihre Größenordnung ist entscheidend für das Auffinden von Punkten auf einer Kante (Für die detailliert Berechnungen siehe Lowe (2004)).

Es wird das Verhältnis r zwischen dem größten und dem kleinsten Eigenwert der Gradientenlänge betrachtet.

Die Kantenfilterung wird dann bestimmt mit der Diagonalmatrix $Tr\ (H)$ und ihrer Determinante $Det\ (H)$ von H.

$$\frac{Tr\ (H)^2}{Det\ (H)} < \frac{(r\ +\ 1)^2}{r} \tag{5.2}$$

Dabei ist $\frac{(r\ +\ 1)^2}{r}$ minimal ,wenn beide Eigenwerte gleich sind und der Wert r steigt.

Experimentell hat D. Lowe herausgefunden, dass die Filterung bei $r = 10$ die besten Resultate liefert.

Aus der DOG-Funktion, bei der die Skalierung des markanten Punktes am aussagekräftigsten ist, werden der Gradient θ um jeden Sammelpunkt $L(x, y)$ sowie die Größe $m(x,\ y)$ des Gradienten um diesen Punkt berechnet.

Die Richtung des Gradienten wird berechnet durch:

$$m(x,\ y) = \sqrt{(L\ (x\ +\ 1,\ y)\ -\ L\ (x,\ y\ +\ 1))^2\ +\ (L\ (x,\ y\ +\ 1)\ -\ L\ (x,\ y\ -\ 1))^2} \tag{5.3}$$

Sein richtungsunabhängiger Betrag mit:

$$G_y = L\ (x\ +\ 1,\ y)\ -\ L\ (x,\ y\ +\ 1) \qquad G_x = L\ (x\ +\ 1,\ y)\ -\ L\ (x\ -\ 1,\ y) \tag{5.4}$$

$$\theta(x,\ y) = \arctan \frac{G_y}{G_y} \tag{5.5}$$

Es wird ein Histogramm mit den Gradientenwerten, der Richtung und der Größe jedes Punktes von einer Region um den merkanten Punkt aufgestellt. Das Histogramm besteht aus 36 Bins (Einträgen), die mit jeweils 0° bis 360° angegeben werden.

In jedem Bin wird dann der stärkste Gradient gesucht und dessen entsprechende Richtung berechnet.

Der 128-dimensionale Vektor (Deskriptor) wird aus den 4x4-Histogramm-Bins, die jeweils acht Orientierungs-Bins haben, erstellt. Jeder Eintrag in diesem Histogramm bildet ein spezifisches Muster, das einem markanten Punkt eine hohen Wiedererkennungswert gibt.

Für die Suche nach den korrespondierenden Punkten wird ein k-diminsionaler Baum (kd-Baum) verwendet,der aus der Datenstruktur des Eingangsbildes gebildet wird.

Es handelt sich hierbei um einen Binär-Suchbaum mit Sortierschlüssel aus eine Menge von mehrdimenisionalen Datenpunkten.

Jeder Baum besitzt eine bestimmte Anzahl an Knoten, die wiederum keinen oder zwei Blätter, d. h. ein linkes und ein rechtes Blatt, besitzen.

Aus den Merkmalen des Suchbildes wird der kd-Baum aufgestellt und mit den gespeicherten Merkmalen verglichen.

Das Verfahren von Beis und Lowe Beis und Lowe (1997) erzeugt einen kd-Baum, bei dem die Verzweigungen stets die gleiche Tiefe annehmen. Somit wird ein optimal balancierter kd-Baum (balanced binary tree) aufgestellt.

Durch diese Optimierung reduziert sich der Zugriff auf ein Blatt in dem Baum von der Ordnung $O\ (log\ N)$ auf die Ordnung $O\ (N)$.

Lowe verwendet den kd-Baum, um in dem 128-dimensionalen Vektor den nächsten Nachbarn mit dem Nearest-Neighbour-Verfahren (NN) zu finden.

Für die Suche wird ein Best-Bin-First-Verfahren angewendet das sich gut für die Suche in höheren Dimensionen eignet.

Lowe vermeidet eine langwierige Suche in einem Baum dadurch, indem er die Zahl der Blattknoten auf einen bestimmten Wert beschränkt, der eine Annäherung über die Entfernung an den nächsten Nachbarn angibt.

Das Best-Bin-First-Verfahren berechnet die minimale Strecke der vom Eingangsknoten aus zu durchlaufenden Knoten im Baum.

Für die Berechnung der Entfernung wird eine priorisierte Warteschlange aufgestellt, indem der nicht begangene Weg mit der gegenwärtigen Baumposition und deren Distanz zu den Eingangsknoten gespeichert wird. Nachdem der Blattknoten überprüft wurde, wird das oberste Element der Warteschlange entnommen und für die weitere Suche verwendet.

6. Implementierung

In diesen Kapitel wird die Implementierung der CAMSHIFT-, KLT- und SIFT-Tracker beschrieben. Dabei wird auf deren Programmierung eingegangen und erläutert, welche Einstellungen für die einzelnen Tracker vorzunehmen sind, um einen optimalen Ablauf zu gewährleisten.

Für den CAMSHIFT- und KLT-Tracker werden die Funktionen der Programmbibliothek LTI-Lib verwendet.

Für die Umsetzung des SIFT-Trackers wird eine Kombination der Bibliotheken LTI-Lib und OpenCV verwendet. Theoretisch bietet die LTI-Lib auch allein alle benötigten Funktionen, aufgrund ihrer mangelnden Dokumentationen wurde jedoch die zweite Programmbibliothek OpenCV zusätzlich in dieses Projekt eingebunden.

Auf diese Weise wurde so zum ersten Mal an der HAW-Hamburg die kombinierte Nutzung der Bibliotheken LTI-Lib und OpenCV in der Programmiersprache C++ und der Entwicklungsumgebung Visual Studio .NET 2003 realisiert.

6.1. LTI-Lib

Die LTI-Lib ist eine Bibliothek mit Ansammlungen von Algorithmen und mathematischen Funktionen mit dem Schwerpunkt Bildbe- und -verarbeitung.

Sie wurde vom **L**ehrstuhl für **T**echnische **I**nformatik (**LTI**) der RWTH Aachen Universität entwickelt.

Ziel der Entwicklung war eine Bibliothek, die explizit in C++ programmiert wurde und auf verschiedenen Betriebssystemen funktionieren sollte. Supportet werden jedoch nur Linux/gcc und MS Windows/MS Visual C++.

LTI-Lib wird von der RWTH Aachen als Open-Source-Paket zur Verfügung gestellt und gibt somit die Möglichkeit, einzelne Funktionen und Algorithmen nach eigenen Wünschen anzupassen.

Installiert wird die LTI-Lib auf dem benutzten Entwicklungsrechner in dem Pfad

`E:\Programme\ltilib`.

Wichtige Verzeichnisse mit den Header- Dateien[1] und den kompilierten „.lib“-Dateien müssen innerhalb der Entwicklungsumgebung „Visual Studio“ hinterlegt werden.

Die Header-Dateien befinden sich in dem Ordner

`E:\Programme\ltilib\lib\headerFiles`

und die „.lib“- und Debug-Dateien in

`E:\Programme\ltilib\lib`.

Die Source-Dateien sind in dem Ordner

`E:\Programme\ltilib\src`

zu finden.

Diese Verzeichnisse müssen bei der Projekterstellung für das C++-Programm in der Entwicklungsumgebung eingetragen und in die Header-Dateien eingebunden werden
(s. Anhang A).

6.2. OpenCV

OpenCV bedeutet Intel® Open Source Computer Vision Library.[2]

In dieser Bibliothek sind die am häufigsten verwendeten und erfolgversprechendsten Bildbearbeitungs- und Bildverabeitungsalgorithmen[3] in der Sprache C und nur einige in C++ implementiert. OpenCV ist ein plattformunabhängiges API[4] und beinhaltet über 300 C-Funktionen und einige C++-Klassen. Es wird von Intel supportet, ist eine lizensierte kostenfreie Software und benutzt intern die Bibliothek Intel® Integrated Performance Primitives (IPP)[5], welche die Verwendung von weiteren Funktionalitäten ermöglicht.

Die OpenCV-Bibliothek kann direkt von der Intel-Webseite oder von Sourceforge [6] heruntergeladen und installiert werden.

Installiert wurde OpenCV in dem Verzeichnis `E:\Programme\OpenCV`.

[1] Header-Dateien haben eine Dateiendung auf „.h“.
[2] http://www.intel.com/technology/computing/opencv/.
[3] Engl.: Image Processing and Computer Vision algorithms.
[4] Engl.: Application Programing Interface.
[5] http://www.intel.com/software/products/ipp/index.htm.
[6] http://www.sourceforge.net/projects/opencvlibrary.

In dem Ordner

`E:\Programme\OpenCV\cxcore\include`

ist die Datei cxcore.h zu finden. Diese Header-Datei ermöglicht die Nutzung von Basisfunktionalitäten zur Bildbearbeitung. Die wichtigsten davon sind:

- Bearbeitung von Bilddaten
- Array-Funktionen
- Mathematische Funktionen zur Bildverarbeitung, wie Addition, Subtraktion, Multiplikation
- Zeichenfunktionen
- Dateifunktion zum Laden und Speichern
- Fehlerbehandlung

Die Dateien in den Ordnern

`E:\Programme\OpenCV\cv` und `E:\Programme\OpenCV\cvaux`

stellen für die Bildverarbeitung u. a. folgende Funktionalitäten zur Verfügung:

- Zusammengesetzte Filterung und Maskierung
- Bewegungsanalyse und Objekt-Tracking
- Objekterkennung
- Funktionen zur Konturbearbeitung
- Kamerakalibrierung und dreidimensionale Rekonstruktion

In dem Verzeichnis `E:\Programme\OpenCV\otherlibs` befinden sich die Funktionalitäten zum Laden und Speichern von Bilddateien und die Funktionen für den Zugriff auf Video-Ein- und -Ausgänge.

Auch diese Verzeichnisse müssen in das C++-Projekt in der Entwicklungsumgebung integriert werden (s. Anhang A).

6.3. Impresario

Impresario ist ein Programm zur schnellen und dynamischen Erstellung von Prozessabläufen. Es wurde für die Verarbeitung von Bildern und Videos entwickelt.

Zur Erstellung eines Prozessablaufs werden sogenannte Module mit einer Funktion zur Bildverarbeitung auf die Projektarbeitsfläche von Impresario gezogen und über ihre vordefinierten Eingangs- und Ausgangsschnittstellen per „Drag and Drop“ miteinander verbunden.

Das Programm Impresario bietet innerhalb eines Prozessablaufes die Möglichkeit, zur Laufzeit das Ergebnis der Bildverarbeitung eines jeden Moduls darzustellen. Die meisten Module besitzen vordefinierte Parameter, deren Änderung zur Laufzeit sofortigen Einfluss auf die Bildausgabe hat.

Um die Funktionalitäten der Bildverarbeitung mit weiteren Modulen zu ergänzen, bietet Impresario detaillierte Anleitungen zu deren Erstellung und Einbindung in C++.

Die Programmierung der Module wird über sogenannte C++-Makros realisiert, deren Aufbau dem Schema der Abbildung 6.1 folgt.

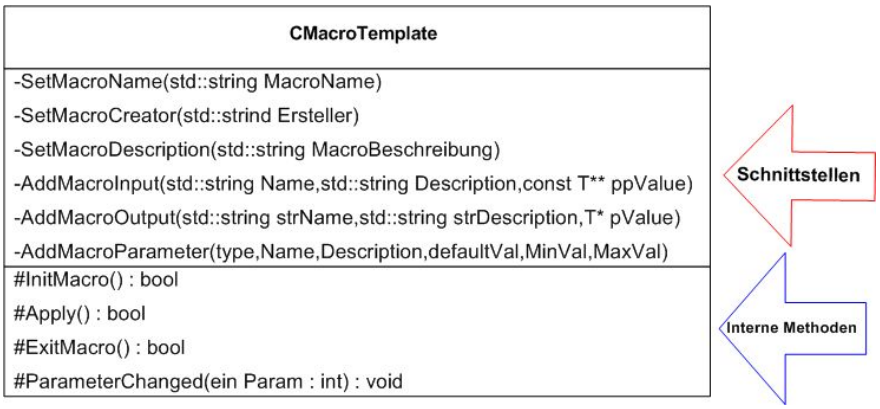

Abbildung 6.1.: Klassendiagramm eines Makros für Impresario

Die einzelnen Methoden des Makros in Visual Studio für Impresario haben dabei folgende Aufgaben:

Das ***InitMacro()*** dient der einmaligen Initialisierung der internen Variablen vor einem Prozessablauf.

Die Methode ***Apply()*** wird bei jeder Änderung des Moduleingangs ausgeführt.

Das ***ExitMacro()*** dient dem „sauberen“ Beenden des Prozessablaufes. Hier können Speicherbereiche freigegeben oder das Abspeichern von Ergebnissen sicher ausgeführt werden.

parameterChanged(int param) wird jedes Mal, wenn die Parameter des Moduls geändert werden, aufgerufen. Diese Methode registriert die Änderungen und setzt die entsprechenden Variablen innerhalb des Makros zur Laufzeit des Moduls um. Zusätzlich zu der in Impresario enthaltenen Dokumentation können weitere Informationen aus Duif (2006) entnommen werden.

6.4. Visual Studio

In diese Arbeit wird für die Erstellung der Makros in der Programmiersprache C++ die Entwicklungsumgebung Microsoft Visual Studio .NET 2003 eingesetzt.

Durch die Nutzung der Software Impresario ist diese Version von Visual Studio Voraussetzung, um Kompatibilität untereinander zu gewährleisten.

Die in der Entwicklungsumgebung Visual Studio benötigten Einstellung für ein erfolgreiches Kompilieren und Einbinden der genutzten Bibliotheken in das Impresario Projekt befinden sich im Anhang dieser Arbeit unter A.

6.5. CAMSHIFT-Tracker

Der CAMSHIFT-Tracker ist in der LTI-Lib vorhanden und wird laut deren Dokumentation implementiert.

Er benötigt als Grundlage für die Berechnung als Eingangsbild eine Proabilitymap, welche die relative Farbhäufigkeitsverteilung der relevanten Farbwerte enthält. Zu diesem Zweck wird das in Impresario vorhandene Modul „ProabilityMap“ zur Bildvorverarbeitung eingesetzt. Es filtert das Eingangsbild anhand von Farbreferenztabellen auf Gesichtsfarbwerte und liefert als Ausgang ein Grauwertebild, bei dessen Darstellung ein Farbwert von 255 der Farbe der Haut entspricht.

Dieses Grauwertebild wird als Eingangsbild des Impresario-Moduls „CamShiftTracker“ genutzt. Abbildung 6.2 zeigt das komplette Impresario-Projekt mit allen benutzten Modulen.

Für den CAMSHIFT-Tracker können folgende Parameter in Impresario eingestellt werden:

Innerhalb des Impresario-Makros erfolgt der Methodenaufruf des CAMSHIFT-Trackers aus der LTI-Lib-Dokumentation mit der „Member Function“:

```
bool lti::camshiftTracker::apply(const channel & src, rectangle & window)
```

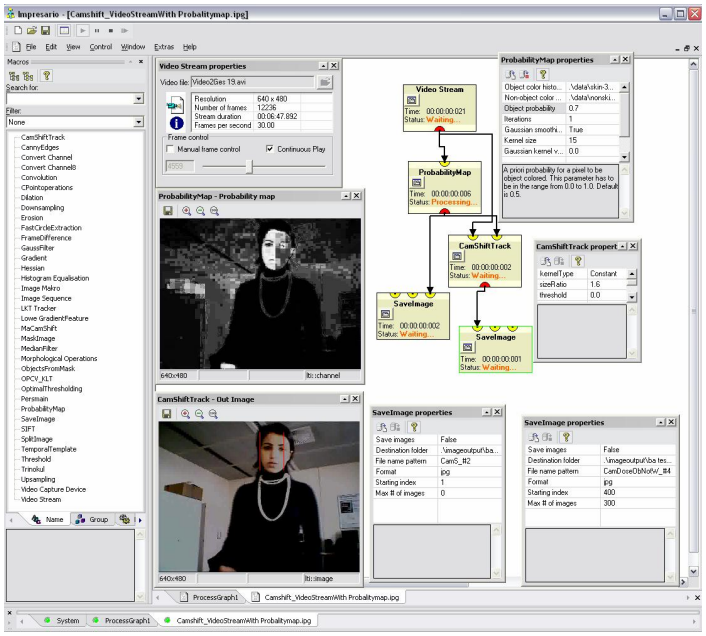

Abbildung 6.2.: Impresario-CAMSHIFT-Projekt

Die entsprechende Anwendung im Programmcode lautet:

```
camTr.apply(*m_pchnInput,outwindow);
```

Dabei ist `camTr` das Klassenobjekt von `lti::camshiftTracker`, und `*m_pchnInput` ist das Grauwertebild der Probabilitymap. Die Methode liefert als Rückgabewert `outwindow` ein „rectangle“ mit den Koordinaten UL und BR (upper left und bottom right). Diese Koordinaten werden benutzt, um ein Rechteck um das gefundene Objekt herum auf dem unbearbeiteten Farbeingangsbild zu zeichnen (s. Abb. 6.3).

Datentyp	Name	Beschreibung
enum eKernelType	kernelType	Kernel-Typen (*Constant*, *Gaussian*) für die Gewichtung der Punkte in dem Suchfenster
double	sizeRatio	Um den Tracker die Suchfenstergröße anpassen zu lassen, muss ein Verhältnis der Höhe zur Breite angeben werden, sonst wird die Fenstergröße vom Tracker nicht verändert: height = sizeRatio * width.
double	threshold	Prozentualer Wert als Schwellwert.

Tabelle 6.1.: Modulparameter für den CAMSHIFT-Tracker

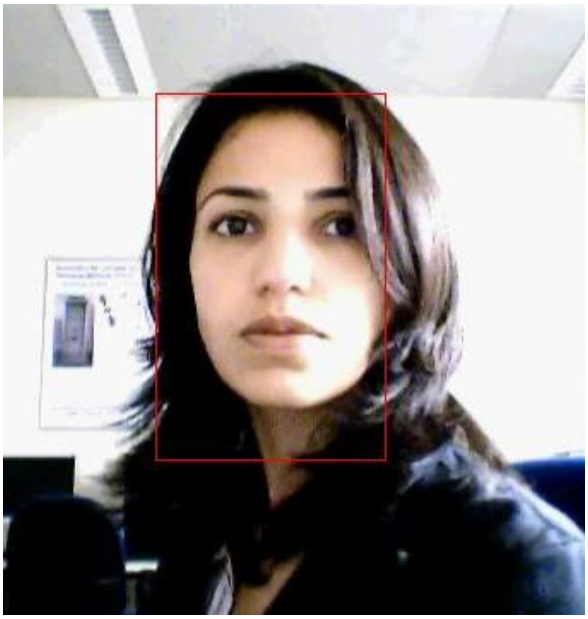

Abbildung 6.3.: Ausgangsbild mit dem Suchfenster

Nach der Initialisierung des Tackers können zusätzliche Informationen über das Objekt innerhalb des Suchfensters mit „get...“-Methoden ermittelt werden:

```
//Koordinaten des Schwerpunkts des Suchfensters
  lti::tpoint <int> param_winCenter = camTr.getCenter();

//Hauptachsen/Orientierung von -Pi/2 bis +Pi/2
  double param_orientation = camTr.getOrientation();

//Breite des Objekts in dem Suchfenster
double param_camWidth = camTr.getWidth();

//Länge des Objekts in dem Suchfenster
double param_laenge = camTr.getLength();
```

6.6. KLT-Tracker

Der Tracker wird mit den Funktionalitäten aus der Header-Datei `#include <ltiLkTracker.h>` der LTI-Lib implementiert. Für die Bildbearbeitung wird zunächst ein Eckendetektor eingesetzt, der markante Punkte aus den Ecken, Kanten und Linien eines Bildes extrahiert. In dieser Tracking-Variante wurde der „Harris-Corner-Detektor“ als Eckendetektor aus der Bibliothek `#include <ltiHarrisCorners.h>` aufgrund der Empfehlung der Dokumentation der LTI-Lib eingesetzt.

Das gesamte Impresario-Projekt mit den benutzten Modulen ist in Abbildung 6.4 ersichtlich.

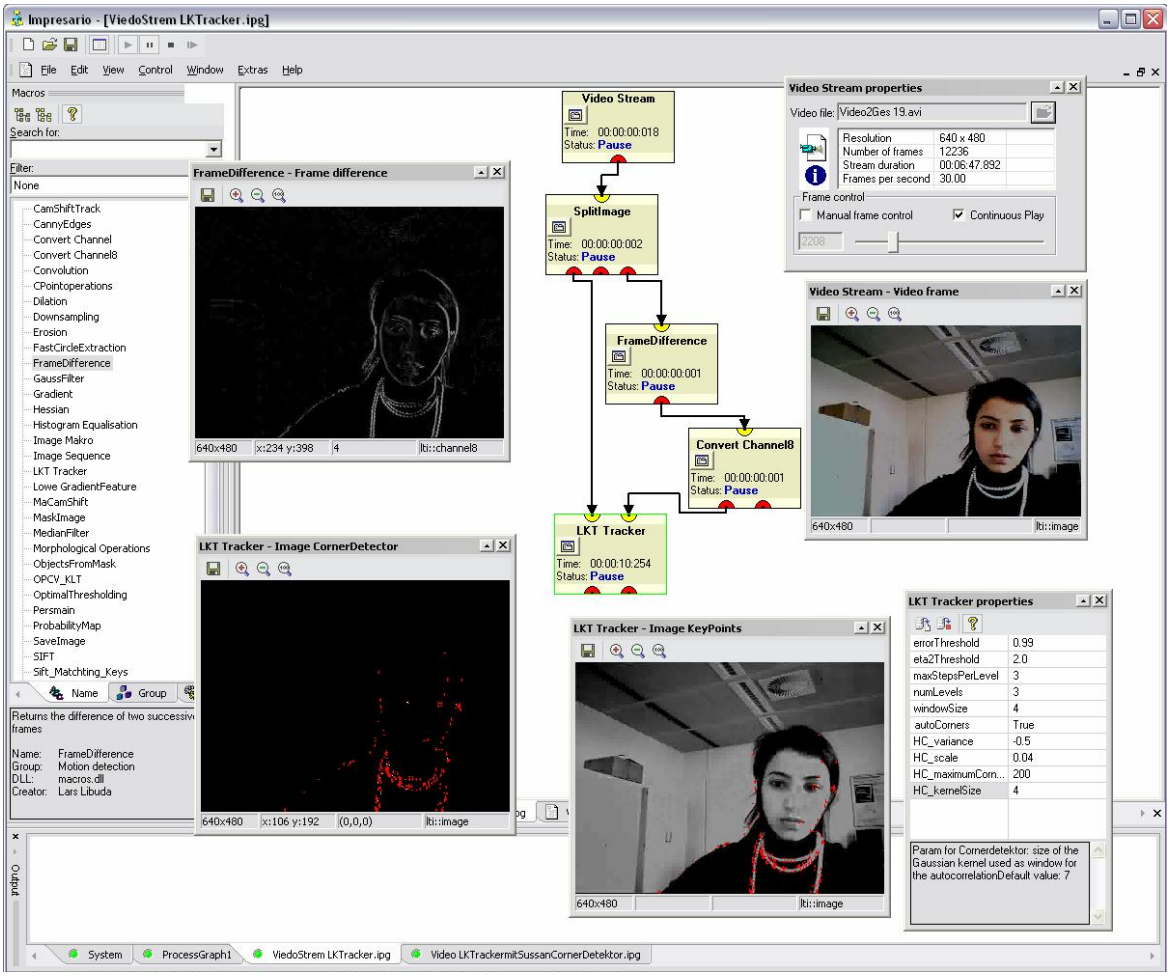

Abbildung 6.4.: Impresario-Projekt KLT-Tracker

Die folgende Tabelle stellt die Parameter für das Modul KLT-Tracker in Impresario dar:

Datentyp	Name	Beschreibung
float	errorThreshold	Wenn die Differenz zwischen den gefundenen Punkten größer ist als dieser Wert, dann wird dieser Punkt als invalid markiert und wird nicht weiter verfolgt
float	eta2Threshold	Oberer Schwellwert für die Erkennung der Pixel auf den Kanten, der erreicht werden muss, bevor der Algorithmus stoppt
int	numLevels	Angabe der Anzahl der Pyramidenstufen
int	maxStepsPerLevel	Maximale Anzahl der Iteration pro Level in der Pyramide
int	windowSize	Angabe der Fenstergröße für den Vergleich der Punkte
boole	autoCorners	Wenn dieser Wert gesetzt ist (true), wird der Harris-Corner-Detektor verwendet
float	HC_variance	Varianz für den Gauß-Filter
int	HC_kernelSize	Größe der Gauß-Maske
float	HC_scale	Faktor für die Autokorrelation im Harris-Corner-Detektor
int	HC_maximumCorners	Maximale Anzahl der detektierten Ecken

Tabelle 6.2.: Modulparameter für den KLT-Tracker

Damit der KLT-Tracker Merkmale aus dem Ergebnisbild des Harris-Corner-Detektors verfolgen kann, müssen einige Parameter aus der Bibliothek `#include <ltiLkTracker.h>` und `#include <ltiHarrisCorners.h>` für den Harris-Corner-Detektor gesetzt werden.

Mit dem Methodenaufruf

```
lk_param.setCornerDetector(lti::harrisCorners());
```

wird für den Tracker ein Harris-Corner-Detektor mit den gesetzten Default- Parameter-Einstellungen initialisiert.

Das Modul wurde so realisiert, dass die Parameter des Harris-Corner-Detektors während der Laufzeit verändert werden können. Aus diesem Grund werden zunächst die eventuell geänderten Parameter mit den Methodenaufruf

```
harrCorn.setParameters(param_harrCorn);
```

gesetzt. Erst danach wird der Harris-Corner-Detektor mit der Methode

```
harrCorn.apply(*lk_chnInput,cD_out_chan);
```

initialisiert.

Diese Methode extrahiert aus dem Eingangsbild `lk_chnInput` die markanten Punkte und zeichnet sie auf dem Ausgangsbild `cD_out_chan` ein.

Die Parameter des KLT-Trackers werden mit dem Methodenaufruf

```
lk_obj.setParameters(lk_param);
```

gesetzt und mit der Methode

```
lk_obj.apply(cD_out_chan,featurePoints)
```

initialisiert. Die Ergebnisse werden in dem Array `featurePoints` gespeichert.

Der Tracker benötigt für seine Aufgabe mindestens zwei aufeinanderfolgende Bilder einer Sequenz, bevor erste Ergebnisse sichtbar werden.

Bei dem ersten Aufruf der `lk_obj.apply(cD_out_chan,featurePoints);`-Methode oder nach dem Aufruf der `lk_obj.reset();`-Methode enthält das Array `featurePoints` die Positionen der verfolgten Punkte aus dem Eingangsbild.

Wenn ein Eckendetektor eingeschaltet ist, werden die gespeicherten Punkte verworfen und die durch den Detektor neu ermittelten Punkte hier gespeichert.

Die Punkte, die während des Verfolgens ungültig werden, werden durch negative Werte in dem Punkt-Array gekennzeichnet.

Die Methode `lk_obj.reset();` initialisiert den Tracker erneut, ohne die gesetzten Parameter zu verändern.

6.7. SIFT-Tracker

Die Implementierung des SIFT-Trackers geschieht unter Verwendung von Funktionalitäten aus drei Quellen. Die Software Impresario benötigt, wie bei den vorhergehenden Implementierungen von CAMSHIFT- und KLT-Tracker, nach wie vor Funktionalitäten der LTI-Lib. Das Open-Source-Programm von D. Lowe[7], geschrieben in der Programmiersprache C, basiert auf Funktionalitäten der Programmbibliothek OpenCV (s. Anhang B.4).

Die Problematik der Implementierung des SIFT-Trackers als ein Modul für Impresario besteht somit in der Kombination dieser einzelnen Komponenten. Umgesetzt wurde der SIFT-Algorithmus in dem Impresario-Modul „SIFTTracker“.

Das Modul besitzt drei Ein- und zwei Ausgänge. Der erste Eingang wird benötigt für ein Referenzbild, welches das zu trackende Objekt enthält. Der zweite Eingang teilt dem Modul einen Dateipfad zur Speicherung der markanten Punkte des Referenzbildes mit. In den dritten Eingang des Modules wird eine Bildsequenz eingegeben, in welcher das Referenzbild verfolgt werden soll.

Abbildung 6.5 zeigt den Prozessablauf des Impresario-Projektes. Im Anhang B.3 ist das Programm dieses Makros zu finden.

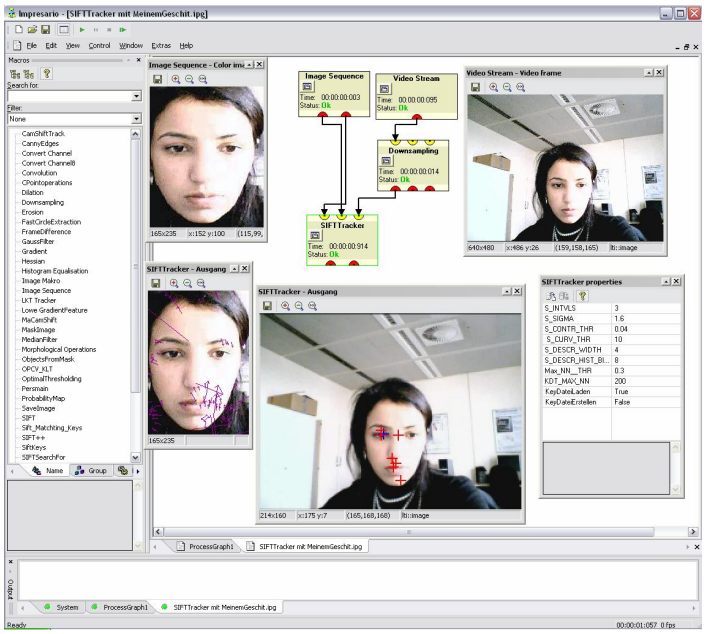

Abbildung 6.5.: Impresario-Projekt SIFT-Tracker

[7] http://www.cs.ubc.ca/ lowe/keypoints/.

Für den SIFT-Tracker können folgende Parameter in Impresario eingestellt werden:

Datentyp	Name	Beschreibung
int	S_INTVLS	Anzahl der Intervalle in einer Oktave
double	S_SIGMA	Gewichtung der Pixel für die Gauß-Faltung
double	S_CONTR_THR	Schwellwert für die Berechnung der Hessischen Matrix
int	S_SIGMA	Maximaler Schwellwert für die Hauptkrümmung
int	S_DESCR_WIDTH	Breite der Histrogrammeinträge
int	S_DESCR_WIDTH	Anzahl der Bins im Histogramm für den Descriptor-Array
int	"Max_NN_THR"	Schwellwert-Faktor für den Abstand zwischen den Nachbarnpunkten
int	KDT_MAX_NN"	Maximale Anzahl der Suche der Nachbarkandidaten mit den Neartst-Neighbors und Best-Bin-First-Verfahren
boole	"KeyDateiLaden"	Extrahierte Punkte aus einer Datei laden
boole	KeyDateiErstellen	Keypoints dymanisch aus dem Objek-Eingangsbild erstellen

Tabelle 6.3.: Modulparameter für den SIFT-Tracker

Die Benutzung der Funktion aus der OpenCV-Bibliothek erfordert zunächst die Konvertierung des Eingangsbildes aus dem LTI-Lib-Format (lti::imge) in das OpenCV-Format (IplImage).

Hier wurde die einfachste, aber auch zeitintensivste Methode für die Konvertierung gewählt und durch pixelweises Kopieren von dem einen in das andere Format realisiert:

```
// Konvertierung LTI-Lib-Image zum IPL-Image
IplImage* ipl_img = create_Ipl_img(m_ImagIn);
```

```
// Konvertierung IPL-Image in LTI-Lib-Image
 lti::image* m_ImageOut = Creat_LTI_Img(ipl_img);
```

Methodenaufrufe des „SIFTTracker“-Makros:

Für die Extraktion der Merkmale aus den Eingangsbildern wird jeweils die Methode

```
int sift_features(IplImage* img, struct feature** feat, int intvls,
  double sigma, double contr_thr, int curv_thr,
  int img_dbl, int descr_width, int descr_hist_bins);
```

aus der Klasse sift.c (vgl. Anhang B.4) aufgerufen.

Für die Extraktion der markanten Punkte und ihre Merkmale aus dem Eingangsbild `ipl_img` mit dem Übergabeparameter von Impresario liefert die Aufrufende Methode

```
 n_Img = sift_features(ipl_img,&img_feat,
  pa_intvals,pa_sigma,
  pa_cont_thr,pa_curv_thr,
  sift_Img_dbl,pa_descr_with,
  pa_descr_hist_bins);
```

als Rückgabeparameter die Anzahl der gefundenen markanten Punkte `n_Img` aus dem Eingangsbild `ipl_img` und speichert die dazugehörigen Merkmale in der Struktur `img_feat`.

Für das Referenzbild `ipl_ObjInt` wird die Anzahl der gefundenen markanten Punkte unter `n_obj` gespeichert und die dazugehörigen Merkmale in der Struktur `obj_feat`.

Mit dem Übergabeparameter von Impresario `m_File_name` wird eine neue Datei mit der Endung „.key“ mit folgender Methode erstellt:

```
char *expr = '.key';
char * c_obj_file = replace_extension(m_File_name->c_str(),expr);
```

Die Methode

```
 export_features( c_obj_file, obj_feat, n_obj );
```

speichert die markanten Punkte des Referenzbildes und die Anzahl der gefundenen Keys `n_obj` in die erstellte Datei `c_obj_file`.

Die markanten Punkte des Referenzbildes werden auf dem ersten Ausgangsbild mit der Methode

```
draw_features(ipl_img, obj_feat, n_obj);
```

gezeichnet.

Das Makro bietet weiterhin die Möglichkeit, auf Basis einer zuvor erstellten „.key“-Datei markante Punkte des Referenzbildes mit der Methode

```
n_obj = import_features(c_obj_file,&obj_feat);
```

zu importieren. Sie lädt die gespeicherten markanten Referenzpunkte aus der Datei `c_obj_file` und speichert die Referenzpunkte in der Struktur `struct feature* obj_feat`. Zusätzlich liefert die Methode die Anzahl `n_obj` der gespeicherten Referenzpunkte als Rückgabeparameter.

Für die Suche nach korrespondierenden Punkte wird ein kd-Baum aus dem Array `(img_feat)` des aktuellen Eingangsbildes `ipl_img` erstellt und mit den Referenzmerkmalen `n_obj` verglichen.

Der kd-Baum aus den markanten Punkten `img_feat` des Eingangsbildes wird erstellt mit der aufrufenden Methode

```
struct kd_node* kd_root = kdtree_build(img_feat, n_Img); ,
```

und hat eine Dimension von `n_Img`, was der Anzahl der gefundenen Punkte des aktuellen Bildes entspricht.

Die Suche eines Referenzpunktes `feat` in dem kd-Baum `kd_root` wird umgesetzt durch die Methode

```
k = kdtree_bbf_knn(kd_root,feat,2,&nbrs,KDT_MAX_NN); ,
```

welche in der Klasse kdtree.c implementiert ist. Es werden das „NeartstNeighbours“-Verfahren und die Best-Bin-First-Methode, wie schon in den Grundlagen erwähnt, angewendet.

Die Methode liefert die Anzahl der gefunden Nachbarn k des aktuellen Referenzpunktes $feat$ als Rückgabeparameter und speichert die gefundenen Nachbarn des Referenzpunktes in der Struktur `struct feature** nbrs`.

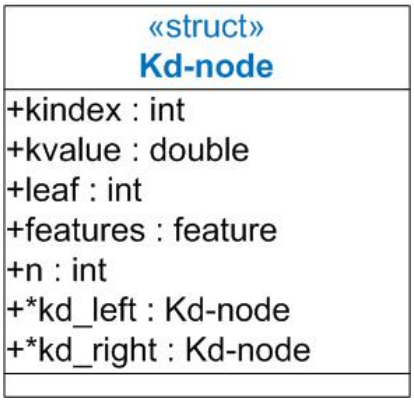

Abbildung 6.6.: kd-Knoten

Die Abbildung 6.6 stellt die Definition eines Knotens innerhalb des Baumes dar.

Wenn zwei Nachbarn gefunden wurden (also $k == 2$), wird der Abstand der beiden Nachbarn zur dem Referenzpunkt `feat` mit quadratischer euklidischer Distanz berechnet.

Ist die Pixel-Entfernung des Referenzpunktes `feat` zum ersten Nachbarn `nbrs[0]` kleiner als die Entfernung zu seinem anderen Nachbarn `nbrs[1]` multipliziert mit dem Faktor des angegebenen Schwellwertes `Max_NN__THR`, dann gehört dieser aktuelle markante Punkt zu dem gesuchten Objekt, ansonsten ist dieser nur ein markanter Punkt der Umgebung. Die korrespondierenden Punkte werden auf dem zweiten Ausgang des Moduls als rote Kreuze dargestellt.

Die Abbildung 6.7 zeigt den Aufbau der Struktur, in der die markanten Punkte mit ihren Merkmalen in Form eines Vektors mit 128 Dimensionen (dem Deskriptor) gespeichert sind.

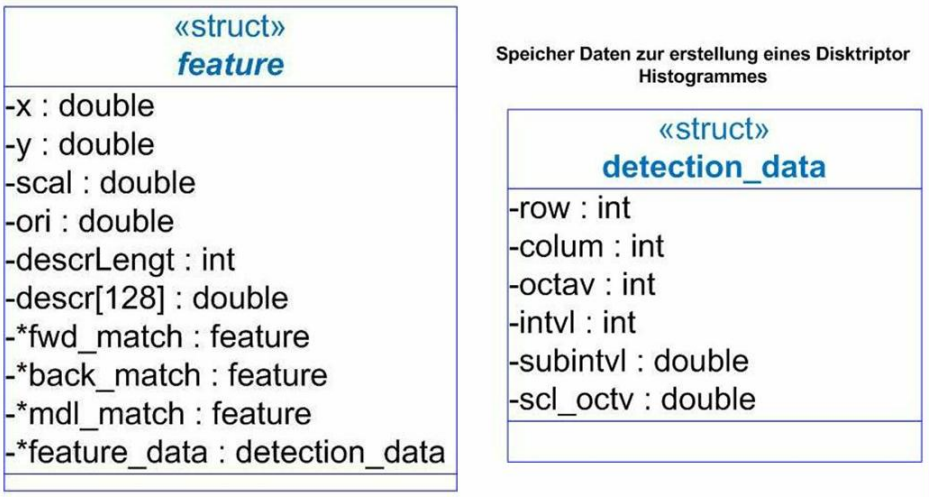

Abbildung 6.7.: Struct-Feature

Folgend ist ein Ausschnitt der Datei „.key“ mit den ersten zwei gespeicherten markanten Punkten und ihren dazugehörigen 128 Deskriptoren aufgeführt:

```
11 128
//Punkt 1
9.197094 18.629528 4.570376 -0.451988
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0
 0 4 7 0 0 0 0 0 0 0 0 0 0 22 0 0 0 0 0 0 6
 70 4 0 0 0 9 48 70 3 1 0 0 10 70 70 14 0 0 0 0
 0 19 2 0 54 2 0 0 0 0 0 6 70 49 1 0 1 2 5 28
 22 19 6 4 26 70 16 5 0 0 0 7 25 36 7 0 7 1 0 0
 0 0 0 0 70 70 12 0 0 0 0 0 13 63 41 9 14 2 0 0
 0 1 3 17 70 3 0 0
//Punkt 2
26.669960 27.257223 3.131284 2.666684
 23 7 0 0 0 0 0 0 71 16 0 0 0 0 5 11 10 1 1 7
 5 25 38 10 3 0 0 0 3 71 42 8 30 2 0 0 0 0 0 1
 71 14 2 0 0 2 7 22 22 5 7 6 4 23 29 15 9 4 10 25
 18 24 8 9 1 0 0 0 0 0 0 0 52 54 29 0 0 0 0 2
 39 64 71 8 3 6 3 5 20 6 5 11 13 42 29 10 0 0 0 0
 0 0 0 0 1 19 6 0 0 0 0 0 50 71 21 0 0 0 0 2
 24 45 71 25 5 6 6 6
```

Aus der ersten Zeile ist ersichtlich, das innerhalb des Referenzbildes insgesamt elf markante Punkte ermittelt wurden und diese mit jeweils 128 Deskriptoren in der Datei abgespeichert sind. In der ersten Zeile eines jeden Punktes sind als Erstes die bis auf Subpixel genauen Koordinaten des markanten Punktes in x- und y-Richtung zu erkennen, gefolgt von einem Skalierungs- und Orientierungsvektor. Die folgenden Zeilen beinhalten die beschreibenden Merkmale (Deskriptoren) zu jedem Punkt.

6.8. Manuelle SIFT-Oktaven

Zur Visualisierung des Aufbaus der SIFT-Oktaven wurden mit Nutzung der vorhandenen Module von Impresario zwei Oktaven mit jeweils drei Scales erstellt. In Abbildung 6.8 ist der Modulaufbau des Impresario Projektes ersichtlich.

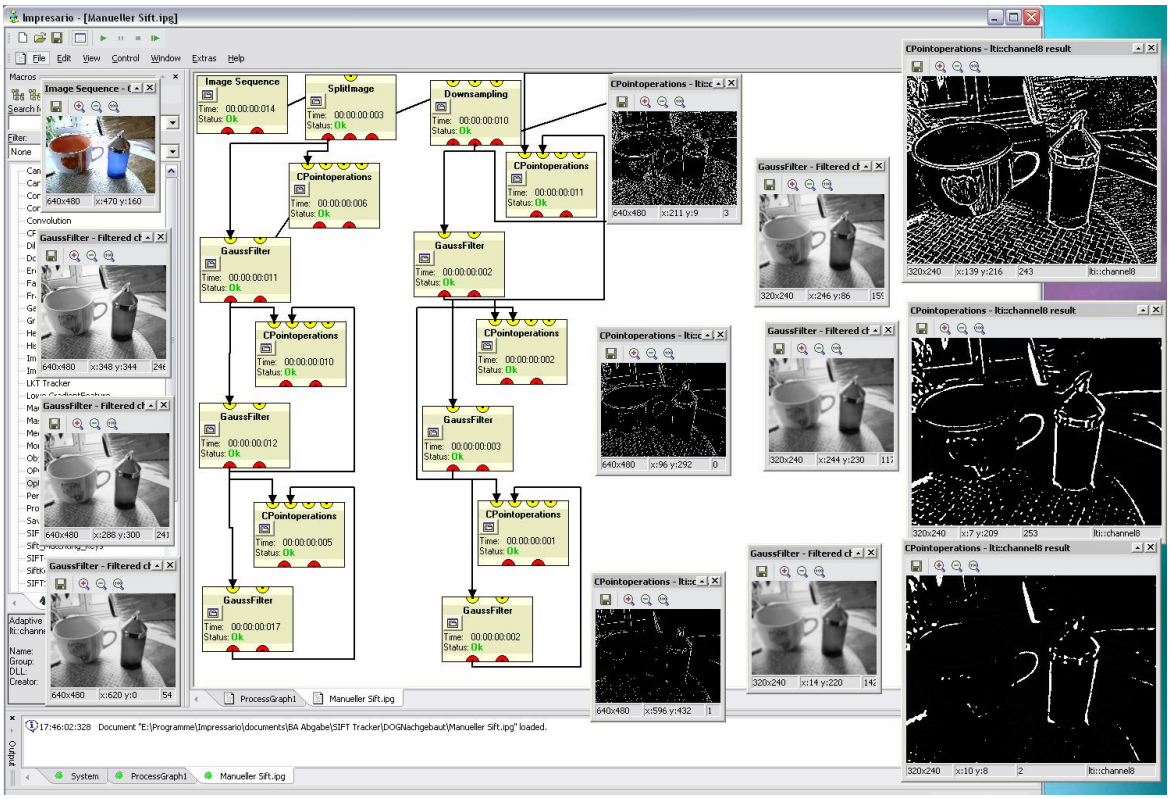

Abbildung 6.8.: Impresario Projekt SIFT-Oktaven

Bemerkenswert sind die Verarbeitungsgeschwindigkeiten zur Erstellung der Gauss- und DOG-Bilder. Bei Austausch des Eingangsmodules „Image Sequence“ gegen „Video Capture Device“ und Nutzung der integrierten USB-Kamera, bleibt die komplette Bildverarbeitung nahe der Echtzeit.

In der Abbildung (s. Abb. 6.8) ist links oben das Eingangsbild zu sehen. Darunter folgen die drei Scale der Gauß-Weichzeichnung, welche mit den dazugehörigen DOG-Bildern der zweiten Spalte die erste Oktave bilden.

Die zweite Oktave wird aus den in der Auflösung reduzierten Bildern der dritten und vierten Spalte gebildet.

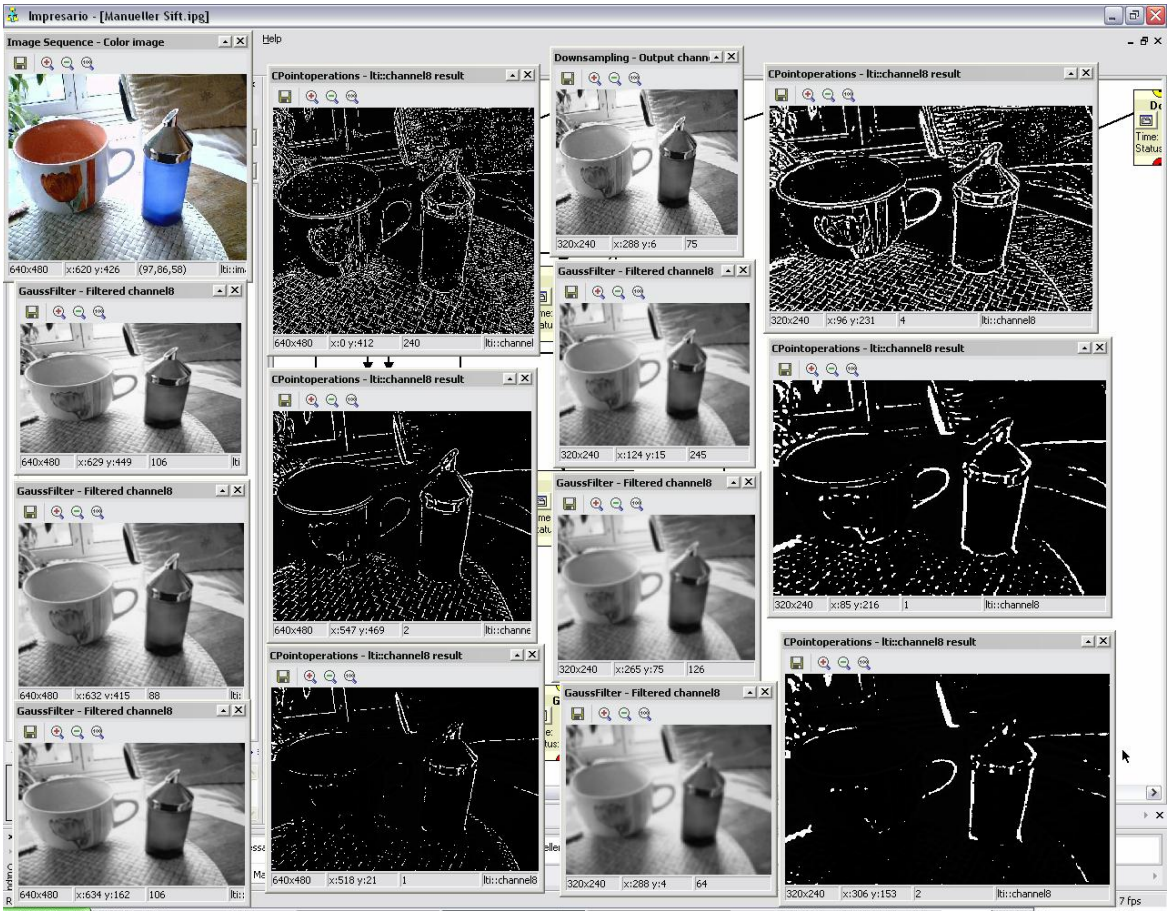

Abbildung 6.9.: Manuelle SIFT-Oktaven

Gut ersichtlich ist die Reduzierung von Bildinformationen und Details in den Stufen der DOG-Bilder von links oben nach rechts unten.

6.9. SIFT-Korrespondenzpunkte

Neben der Fingerübung der manuellen SIFT-Oktaven ist ein weiteres Impresario Projekt zur Demonstration der robusten Zuweisung korrespondierender Punkte mit dem SIFT-Algorithmus in den Anfängen dieser Arbeit entstanden.

In Abbildung 6.10 ist der Impresario Modulaufbau zu erkennen. Die Module „SiftKeys“ errechnen die markanten Siftpunkte und das Modul „Sift_Matching_Keys“ vergleicht diese miteinander.

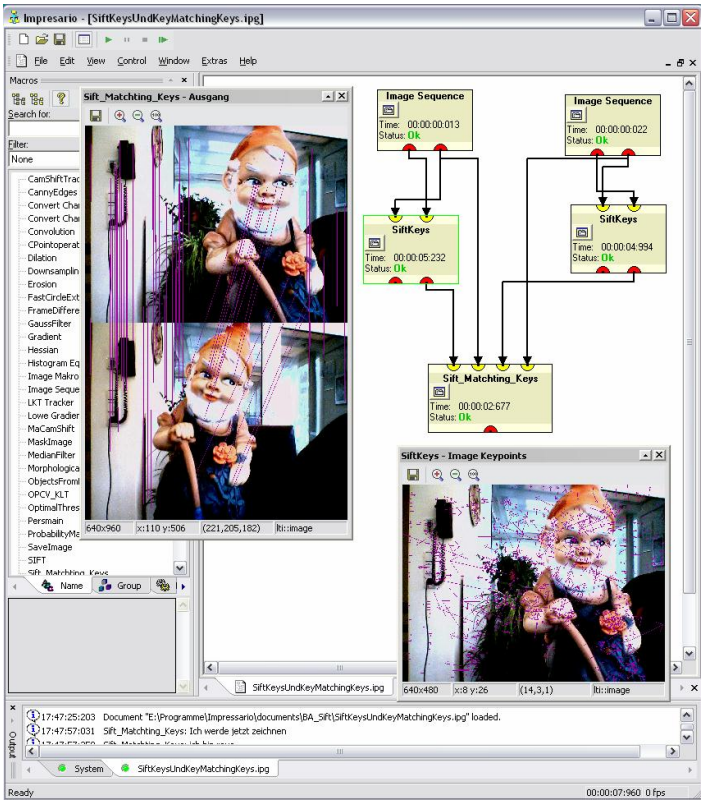

Abbildung 6.10.: Impresario-Projekt SIFT-Matching-Keys

Die Module können jeweils mit einzelnen Bildern oder auch Streams umgehen. So kann als Referenz ein Passfoto benutzt werden um das Gesicht im Stream der Kamera zu suchen.

Anhand der Abbildungen 6.11 und 6.12 sind die durch den SIFT-Algorithmus erstellten, korrespondierenden, markanten Punkte dargestellt.

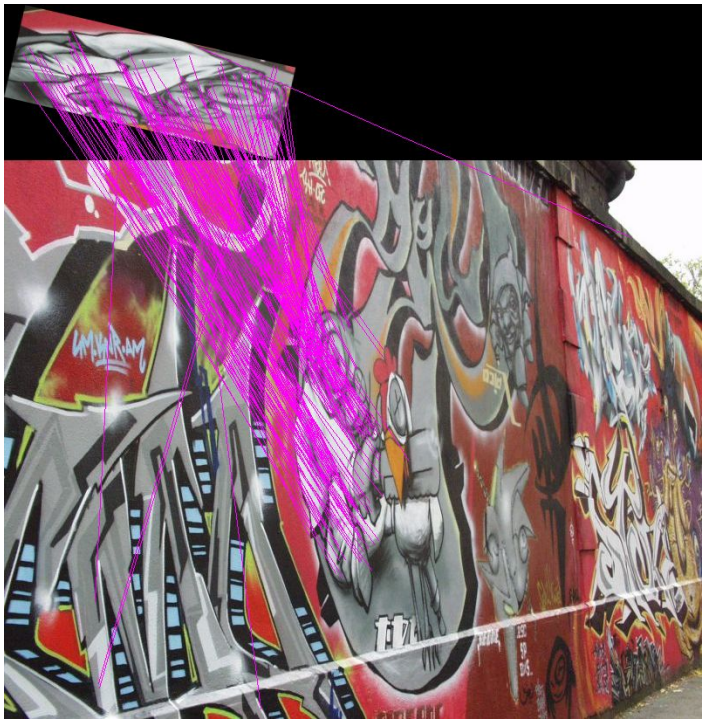

Abbildung 6.11.: SIFT-Korrespondenzen Grafitiwand

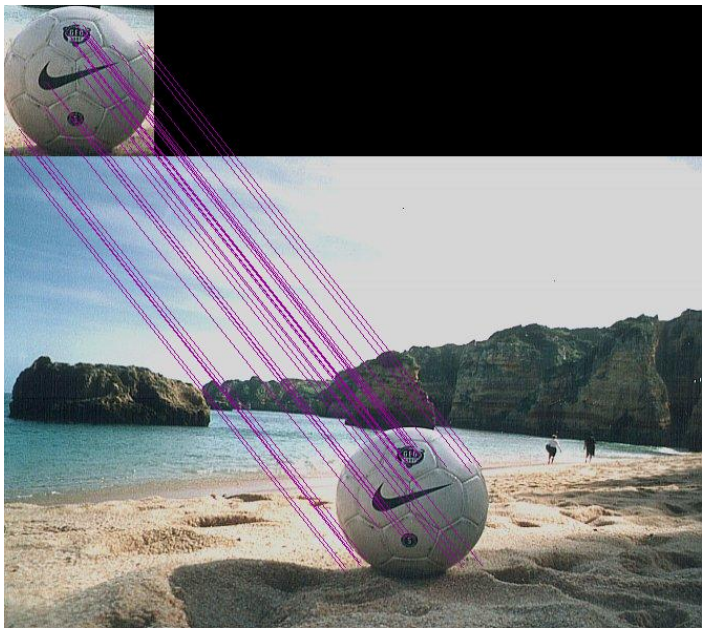

Abbildung 6.12.: SIFT-Korrespondenzen Ball

7. Ergebnisse

Diese Kapitel beschäftigt sich mit den Beobachtungen, die nach der Implementierung der Algorithmen bei deren Anwendung gemacht werden konnten.

Bei der Untersuchung der Tracker hat sich herausgestellt, dass bei dem Vergleich der drei Algorithmen mit einer einzigen Videosequenz keine aussagekräftigen Bewertungen gemacht werden können, da ihnen unterschiedliche Definitionen zur erfolgreichen Objektverfolgung zugrunde liegen. Eine optimal aussagekräftige Bildsequenz müsste Unterschiede in der Beleuchtung, beim Hintergrund, wechselnde Objekte bei wechselnder Objektgeschwindigkeit und unterschiedliche Kameraqualitäten in sich vereinen.

Für die Testszenarien wurde aus Mangel an adäquaten Alternativen eine in den Displayrahmen eines Laptops integrierte Webcam mit 640 x 480 Pixel Auflösung und automatischer Bildanpassung ohne Zoom verwendet. Suboptimale Videoresultate hinsichtlich Helligkeit, Kontrast und Vibration waren die Folge.

Wie oben schon beschrieben, basieren die Algorithmen auf unterschiedlichen Methodiken zur Extraktion der Merkmale. Um die Tracker bezüglich ihrer Verwendbarkeit in der Praxis zu bewerten, werden unterschiedliche Bedingungen und Definitionen benötigt.

Die folgenden Kriterien sollen als Kompromiss für den Vergleich der Algorithmen dienen:

- Bedienbarkeit -> Implementationsaufwand
- Funktionalität -> Einsatzmöglichkeiten, Geschwindigkeit, Genauigkeit, Rotation, Translation, Skalierung und Lichtverhältnisse
- Realisierung -> Was konnte im Zuge dieser Arbeit umgesetzt werden?

Die Implementierung der drei Tracker ergibt somit folgende Resultate:

7.1. CAMSHIFT

Bedienbarkeit

Die Dokumentation der LTI-Lib erwies sich im Laufe der Implementierung als mangelhaft, da ihr nur unzureichende Informationen entnommen werden konnten. Dieser Mangel konnte nur durch erhöhten Arbeitsaufwand vonseiten des Users ausgeglichen werden.

Als der Tracker dann allerdings erst einmal lief, ergaben sich aus der Bedienung keine weiteren Probleme, sodass die Bedienbarkeit als gut beurteilt werden kann.

Funktionalität

Der CAMSHIFT-Tracker ist spezialisiert auf hautfarbene Objekte, was sich als Problem herausstellte, als vor einem rosafarbenen Hintergrund getestet wurde: Das Zielobjekt konnte daher nicht erkannt werden. Auch bei starker Helligkeit traten Probleme auf, das Suchfenster konnte nicht exakt an die Größe und Form des Objektes angepasst werden. Bei der Verfolgung darf das Objekt keine allzu schnellen oder abrupten Bewegungen ausführen, da es sich ansonsten aus dem Bereich des Suchfensters entfernt. In diesem Fall blieb das Suchfenster in der letzten Postion stehen und führte keine Positionskorrektur durch.

Grundsätzlich wird bei der Arbeit mit dem CAMSHIFT-Tracker von einer immobilen Kamera und einem sich bewegenden Objekt ausgegangen. Wenn das Objekt erstmalig erfasst wurde, besteht allerdings auch die Möglichkeit, dass es selbst bei sich bewegender Kamera gut verfolgt wird.

Der CAMSHIFT-Tracker ist gut geeignet für den Einsatz in nicht allzu hellen Räumen und zum Verfolgen von hautfarbenen Gegenständen. Sobald sich ein Objekt nicht sehr schnell bewegt, gewährleistet der Tracker eine kontinuierliche Verfolgung, da er nicht punktbezogen arbeitet, sondern stets auf das Suchfenster fixiert. Es ist auch gut möglich, mit diesem Tracker Objekte über eine relativ große Distanz (bis zu fünf Meter) zu verfolgen.

Realisierung

Vorteile:

- Gute Bedienbarkeit.
- Keine Glättung und Reduktion des Bildes erforderlich.
- Erstellung einer Farbreferenztabelle auch aus dem Ausschnitt eines Bildes möglich.
- Solange das Objekt seine Farbwerte beibehält, ist es egal, wie sich dessen Form ändert.
- Dynamische Anpassung der Größe des Trackingfensters.

Nachteile:

- Es können nur die Objekte verfolgt werden, die vorher durch eine Farbreferenztabelle erfasst wurden.
- Die Erstellung der Farbtabelle ist aufwendig und benötigt zusätzliche Bildbearbeitungssoftware.
- Wird zur Erstellung der Farbreferenztabelle ein Bildausschnitt benutzt, der neben dem Objekt Hintergundinformationen enthält, werden auch diese verfolgt und können gewünschte Ergebnisse verzerren.
- Hebt sich das Suchobjekt nicht deutlich von der Hintergrundfarbe ab, ist kein Tracking möglich.
- Bei einer verdeckten Bewegung des Objektes bleibt das Suchfenster an der letzten Position.

7.2. KLT

Bei der Anwendung des KLT-Trackers auf eine durch die Laptopkamera aufgenommene Videosequenz wurde der statische Hintergrund aus unerfindlichen Gründen mit diversen markanten Punkten identifiziert. Da dieser Tracker laut Algorithmus nur auf Bewegungen anspricht, war hier eine intensive Fehleranalyse des Programmcodes notwendig. Nach eingehender Untersuchung war der Fehler auf erschütterungsbedingte Vibrationen der Laptopkamera zurückzuführen (s. Abb. 7.1 und 7.2). Aus diesem Grund wurden auch unbewegliche Gegenstände identifiziert.

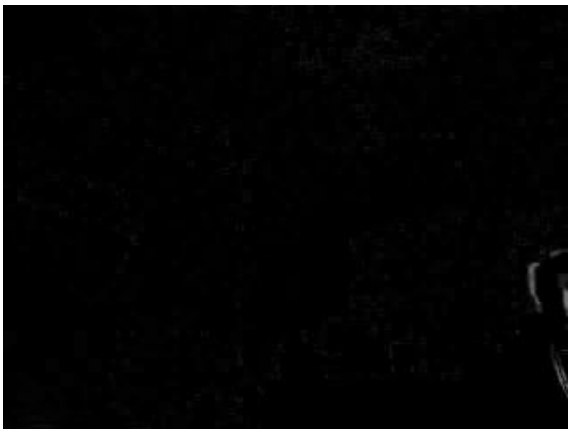

Abbildung 7.1.: KLT-Differenzbild

Abbildung 7.2.: KLT-Ausgangsbild

Bedienbarkeit:

So wie bei dem CAMSHIFT-Tracker traten auch bei der Implementierung des KLT-Trackers mit den Funktionalitäten der LTI-Lib Probleme im Bereich der Dokumentation auf. Schließlich führte ebendieser Mangel zu verstärkten Anstrengungen, die OpenCV-Bilbliothek erfolgreich mit in die Entwicklungsumgebung zu integrieren.

Zusammenfassend ist der Algorithmus in der Modulanwendung mit Impresario so aufwendig, das er durch die langen Laufzeiten des Programmes ungeeignet für einen Einsatz in der Praxis ist. Eventuell könnte eine umfassendere Dokumentation hier Abhilfe schaffen.

Funktionalitätt:

Hinsichtlich Rotation, Translation und Skalierung erwies sich der KLT-Tracker als robust, solange das Objekt langsame Bewegungen ausführt. Bewegt sich das Objekt nicht, werden keine zu trackenden Punkte gefunden. Bei steigender Geschwindigkeit nimmt die Genauigkeit der Objekterfassung ab.

Bei Überbelichtung (s. Abb. 7.3) hat der KLT-Tracker keine Probleme, markante Punkte aus dem Bild zu extrahieren und zu verfolgen. Solange Ecken und Umrisse des Objektes deutlich erkennbar sind, konnten keine negativen Auswirkungen auf das Ergebnis beobachtet werden. Je höher die Kontrastierung der Bilder war, desto besser können Kanten und Konturen des Objektes erkannt werden (s. Abb. 7.4). Somit ist ein hoher Kontrast hilfreich bei der Anwendung des KLT-Trackers.

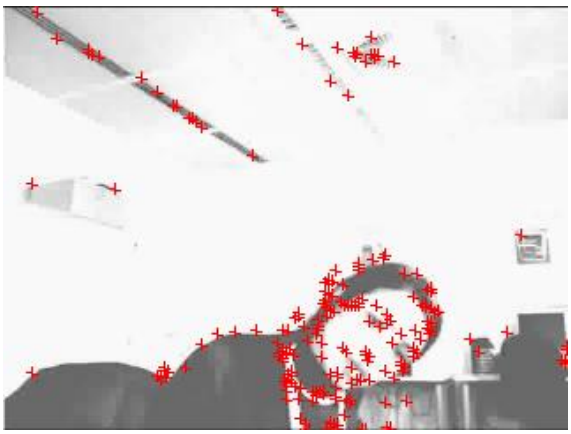

Abbildung 7.3.: KLT mit Überbelichtung

Abbildung 7.4.: KLT: hoher Kontrast

Realisierung

Der Tracker benötigt mindestens zwei aufeinander folgende Bilder für die Berechnung der Bewegung. Es konnte keine Verfolgung eines Objektes in einer Viedeosequenz auf die Praxistauglichkeit hin getestet werden. Ferner konnte nicht realisiert werden, dass ein sich be-

wegendes Objekt während einer ganzen Bildsequenz verfolgt wird. So fällt es schwer zu entscheiden, wie gut der Tracker tatsächlich funktioniert.

Vor Inbetriebnahme des KLT-Trackers wurde ein zusätzliches Modul („Frame Difference“) vorgeschaltet, welches zwei aufeinander folgende voneinander subtrahiert und somit nur die tatsächliche Bewegung des Objektes als Konturen anzeigt (s. Abb. ...). Auf diese Weise konnten langsame Bewegungen verfolgt werden.

Vorteile:

- Unempfindlich gegenüber Helligkeits- und Kontrastschwankungen.
- Objekterkennung durch Extraktion bewegter Bildregionen.
- Der Algorithmus ist farbunabhänig.

Nachteile

- Zu lange Laufzeiten für einen Einsatz nahe der Echtzeit.
- Einmalige Verdeckung des Objektes führt zum Verlust der markanten Punkte.
- Schnelle Bewegungen führen zum Absturz des Trackers.
- Der Harris-Corner-Deketor ist zu langsam, um Ecken und Konturen erkennen zu können.
- Die Verfolgung von Objekten wird gestört durch Kanten und Linien in der Umgebung des Objektes.

7.3. SIFT

Bedienbarkeit

Die erste Hürde bei der Implementierung des SIFT-Trackers waren das zusätzliche Verwenden der OpenCV-Bibliothek sowie die benötigten Projekteinstellungen in Visual Studio. Dank guter Dokumentation von OpenCV, Visual Studio und einiger Experimente konnten die Eintragung der richtigen .lib- und Header-Datein in den Projekteigenschaften ermittelt werden.

Die zweite Hürde bestand dann darin, den in C geschriebenen Sourcecode an C++ anzupassen und anzugleichen.

Als Nächstes mussten die LTI-Formate in OpenCV-Formate konvertiert werden.

Schließlich ergab sich noch das Problem, dass die mit Visual Studio erstellte DLL-Datei unter Verwendung der OpenCV-Bibliothek nicht mit Impresario kompatibel waren. Es galt also herauszufinden, welche zusätzlichen DLL-Dateien für Impresario benötigt werden, damit das Modul diese erkennt. Für die Lösung wurde der Software Dependencie Walker eingesetzt. Das Tool stellt die Abhängigkeiten der DLLs und deren Reihenfolge der Aufruhe untereinander dar.

Funktionalität

Es findet Punktkorrespondenz auch dann statt, wenn das Referenzobjeket rotiert, gespiegelt wird oder eine andere Skalierung als das Objekt in dem Suchbild hat. Der Tracker ist robust und invariant gegenüber der Beleuchtung eines Objektes in Viedeo- bzw. Bildsequenzen und erkennt Objekte in Bildsequenzen auch in weiter Entfernung.

Realesierung

Durch die Definition der markanten Punkte nach Lowe mit dem 128-dimensionalen Vektor der näheren Umgebung scheint diesbezüglich ein Durchbruch gelungen zu sein.

Vorteile

- Der Tracker ist robust und arbeitet relativ unabhängig von der Umgebung.
- Es kann in der Regel immer eine Korrespondenz von markanten Punkte eines Objektes ausfindig gemacht werden.

Nachteile

- Die Erstellung einer einzigen „.key“-Datei zur Speicherung markanter Punkte und ihren Merkmalen von verschiedenen Objekten konnte aus Zeitgründen nicht umgesetzt werden, da es bislang keine standardisierte Methode gibt, um diese aus mehreren Dateien zu erstellen.
- Wenn der Schwellwert für die Berechnung der quadratischen euklidischen Differenz zu groß gewählt wird, werden Punkte erkannt, die nicht zum Referenzbild, sondern zur Umgebung gehören (s. Abb. 7.7).
- Wenn ein zweites Objekt im Bild eine ähnliche Struktur wie das zu verfolgende besitzt, finden falsche Zuordnungen statt.

Markante Referenzpunkte Korrespondenz der markanten Punkte

Abbildung 7.5.: Rotationinvariante Zuordnungen der markanten Referenzpunkte

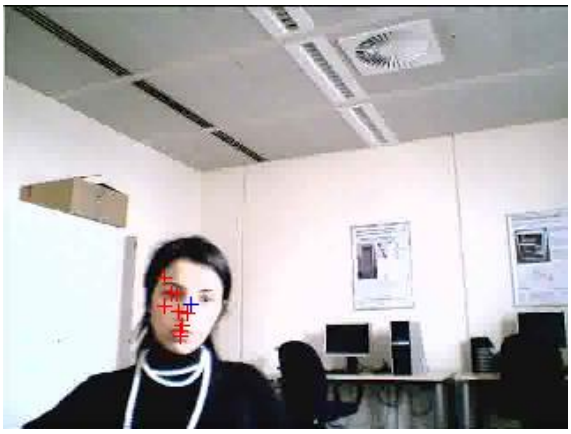

Abbildung 7.6.: Normale Lichtverhältnisse

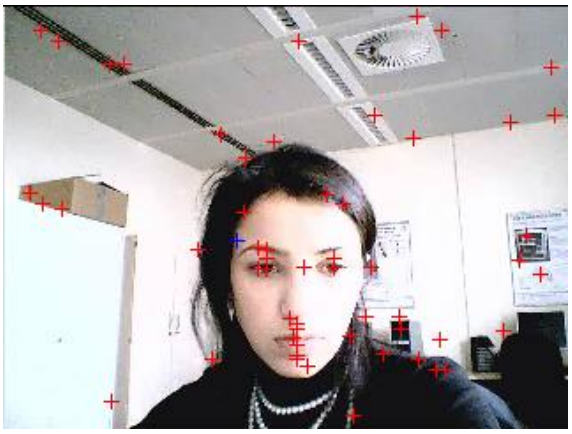

Abbildung 7.7.: Korrespondierende Punkte mit einem Schwellwert > 0.5

Abbildung 7.8.: Dunkelheit und Entfernung

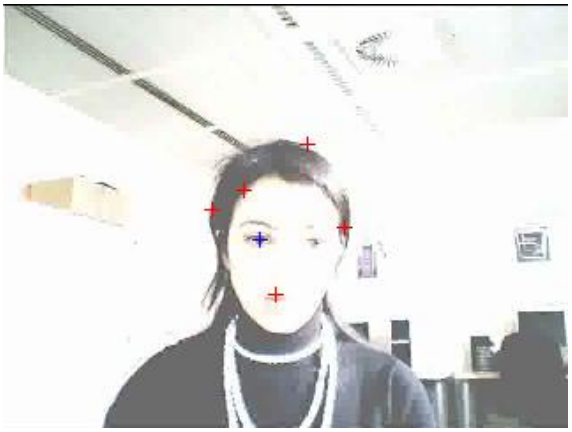

Abbildung 7.9.: Überbelichtung

7.4. Zusammenfassung

Es hat sich herausgestellt, dass das CAMSHIFT-Verfahren als einziges in der Lage ist, wirklich ein Objekt in einer Bildsequenz zu tracken. Die beiden anderen Verfahren identifizieren „lediglich“ markante Punkte aufgrund ihrer Merkmale und verfolgen deren Korrespondenzen innerhalb einer Bildsequenz. Der SIFT-Tracker weist dabei in der Praxis durchaus eine gute Funktionalität auf, während der KLT-Tracker sich als zu empfindlich gegenüber Bewegung herausgestellt hat.

Die folgende Tabelle zeigt eine zusammenfassende Auflistung der Eigenschaften der untersuchten Tracker. Die Einschätzungen zu den aufgeführten Kriterien sind die Ergebnisse unterschiedlicher Tests und basieren auf den jeweils optimalen Parametereinstellungen der Tracker in den Impresario Modulen.

Kriterien	**CAMSHIFT**	**KLT**	**SIFT**
Rotation eines Objketes	+	+	+
Translation eines Objketes	+	+	++
Skalierung eines Objketes	+	+	++
Objektverdeckung	-	–	++
Objekterkennung bei optimalen Lichtverhältnissen	+	-	++
Objekt Überbelichtung	-	+	-
Kontraständerung	++	+	+
Verfolgung eines Objektes	+	-	++
Verfolgung mehrerer Objekte	–	+	+
Statische Objekte	++	–	++
Langsame Objekte	++	+	++
Schnelle Objekte	-	-	++
Farbige Objekte	++	–	+
Spiegelnde Objekte	–	+	–
Bewegte Kamera	+	–	++
Statische Kamera	+	-	++
Algorithmusgeschwindigkeit	++	–	+

Tabelle 7.1.: Ergebnis Tabelle

8. Schlussbetrachtung

Bei einem Vergleich der drei getesteten Tracker schneidet der KLT deutlich am schlechtesten ab. Seine Bewegungsempfindlichkeit gegenüber äußeren Einflüssen führt durchweg zu unbefriedigenden Ergebnissen bei der Verfolgung von Objekten. Hinzu kommt, dass er nur bewegte Objekte identifiziert, sodass bei Bewegungsänderungen falsche Korrespondenzen entstehen. Dadurch ist der KLT-Tracker nicht in der Lage, genau ein Objekt permanent zu verfolgen.

CAMSHIFT ist hier mit seinem grundsätzlich anderen Ansatz, ein Objekt aufgrund von Farbwerten zu identifizieren, deutlich besser für die Praxis geeignet. Auch der SIFT-Tracker erweist sich als recht praxistauglich, was vor allem auf seine große Robustheit zurückzuführen ist. Beide Systeme bieten jedoch auch noch hinreichend Raum für Verbesserungen. Die Farberkennung anhand von vordefinierten Referenztabellen beim CAMSHIFT-Tracker schränkt seine Nutzbarkeit ein. Hier wäre es wünschenswert, während der Laufzeit die Farbwerte jedes Objektes dynamisch zu errechnen und das Trackingfenster dann daran anzupassen.

Beim SIFT-Tracker ist es nicht ohne Weiteres möglich, markante Punkte verschiedenster Objekte in einer einzigen Datei zu speichern. So kann jeweils nur ein Objekt pro Bildzyklus erkannt werden. Auch wäre es wünschenswert, die markanten Punkte nicht nur auf dem gesamten Bild bestimmen zu können, sondern lediglich in einem mit der Maus definierten Ausschnitt, da somit viele Fehltreffer aus der Umgebung des Objektes ausgeschlossen werden könnten. Auf diese Weise könnten dann auch die Referenzpunkte in anderen Bildabschnittsregionen wiedererkannt werden.

Als Resultat der Arbeit kann gesagt werden, dass der SIFT-Tracker für mobile Anwendungen geeigneter ist, da beim CAMSHIFT das Objekt nach Möglichkeit für eine konstante Verfolgung nicht aus dem Trackingfenster verschwinden sollte. CAMSHIFT hat seine Stärken im Erkennen von Menschen, Gesichtern und Gesten, da hier unabhängig von Formen aufgrund der Farben verfolgt wird.

Denkbar wäre auch eine Kombination der verschiedenen Tracker. Eine Erstellung der markanten Punkte nach dem SIFT-Verfahren in Verbindung mit dem Tracking innerhalb des Suchfensters von CAMSHIFT könnte durchaus vielversprechende Ergebnisse liefern.

Literaturverzeichnis

[Beis und Lowe 1997] BEIS, Jeffrey S. ; LOWE, David G.: Shape indexing using approximate nearest-neighbour search in high-dimensional spaces, In Proc. IEEE Conf. Comp. Vision Patt. Recog, 1997, S. 1000–1006. – URL http://www.cs.ubc.ca/~lowe/papers/cvpr97.pdfStand:Nov.2008

[Bouguet 2002] BOUGUET, Jean-Yves: Pyramidal Implementation of the Lucas Kanade Feature Tracker, URL http://robots.stanford.edu/cs223b04/algo_tracking.pdf, 2002

[Bradski 1998a] BRADSKI, Gary R.: Computer Vision Face Tracking For Use in a Perceptual User Interface, Intel Technology Journal Q2 '98, Intel Corporation, 1998. – URL ftp://download.intel.com/technology/itj/q21998/pdf/camshift.pdf

[Bradski 1998b] BRADSKI, Gary R.: Real Time Face and Object Tracking as a Component of a Perceptual User Interface, IEEE WACV, 1998, S. 214–219. – URL http://ieeexplore.ieee.org/iel4/5940/15812/00732882.pdf?tp=&arnumber=732882&isnumber=15812

[Cheng 1995] CHENG, Yizong: Mean Shift, Mode Seeking, and Clustering. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 17 (1995), Nr. 8, S. 790–799. – ISSN 0162-8828

[Comaniciu und Meer 1999] COMANICIU, Dorin ; MEER, Peter: Mean Shift Analysis and Applications, 1999, S. 1197–1203

[Duif 2006] DUIF, Thomas: Entwicklung von Bildverarbeitungsalgorithmen mit Impresario, Lehrstuhl für Technische Informatik - RWTH Aachen, 2006. – URL http://www.rz.rwth-aachen.de/global/show_document.asp?id=aaaaaaaaaaavhxq

[Gremse 2005] GREMSE, Felix: Skaleninvariante Merkmalstransformation - SIFT Merkmale, "Hauptseminar Medizinische Bildverarbeitung 2005 RWTH-Aachen. ", 2005, S. 149–164. – URL http://phobos.imib.rwth-aachen.de/lehmann/seminare/bv_2005.pdfStand:Nov.2008

[Lowe 2004] LOWE, D.: Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision, 60, 2004, S. 91–110. – URL http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdfStand:Nov2008

[Lowe 1999] LOWE, David G.: Object Recognition from Local Scale-Invariant Features, Proc. of the International Conference on Computer Vision ICCV, Corfu, 1999, S. 1150–1157. – URL http://www.cs.ubc.ca/~lowe/papers/iccv99.pdfStand: Juli.2008

[Lucas und Kanade 1981] LUCAS, Bruce D. ; KANADE, Takeo: An Iterative Image Registration Technique with an Application to Stereo Vision, International Joint Conference on Artificial Intelligence, 1981, S. 674–679. – URL http://www.ces.clemson.edu/~stb/klt/lucas_bruce_d_1981_1.pdf

[Meisel 2006] MEISEL, Andreas: Vorlesung Robot Vision, Hochschule für Angewandte Wissenschaften Hamburg, 2006. – URL http://www.informatik.haw-hamburg.de/wp_robot_vision.html

[Pitzer Juni 2004] PITZER, Benjamin: Untersuchung von Algorithmen zur Findung von Punktkorrespondenzen in Videosequenzen, RWTH Aachen, Juni 2004, S. Kapitel 3

[Tomasi und Kanade April 1991] TOMASI, Carlo ; KANADE, Takeo: Detection and Tracking of Point Features., Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991, S. 674–679. – URL http://www.ces.clemson.edu/~stb/klt/tomasi-kanade-techreport-1991.pdf

A. Settings

Settings von Visual Studio

Für die Erstellung der kompilierten DLL-Dateien als Makro für Impresario sind folgende Eintragungen in den Projekteigenschaften der Entwicklungsumgebung von Visual-Studio-Projekten nötig:

Einstellung der C/C++ Eigenschaften

Header Pfadeintragung zur Benutzung der LTI-Lib- und OpenCV-Bibliotheken in einem Projekt.

Allgemein —>Zusätzliche Includverzeichnisse

```
E:\Programme\Libs\LTILib\lib\headerFiles
E:\Programme\OpenCV\cvaux\include
E:\Programme\OpenCV\cv\include
E:\Programme\OpenCV\cxcore\include
E:\Programme\OpenCV\otherlibs\highgui
E:\Programme\OpenCV\otherlibs\cvcam\include
```

Linker-Einstellung für das Projekt

Eingabe der Pfade zusätzlicher Bibliotheksverzeichnisse für den Linker (s. Abb.A.2).

In dem Linker —>*Allgemein —> Zusätzliche Bibliotheksverzeichnisse*

```
E:\Programme\Libs\LTILib\lib
E:\Programme\OpenCV\otherlibs\_graphics\lib
E:\Programme\OpenCV\lib
```

Eingaben der „.lib“-Dateien der LTI-Lib und OpenCV für den Linker (s. Abb.A.2).

Eingabe —> Zusätzliche Abhängigkeiten

```
ltilib7_dmfcd.lib ,cvd.lib,highguid.lib,
cvauxd.lib,cxcored.lib,cvcam.lib
```

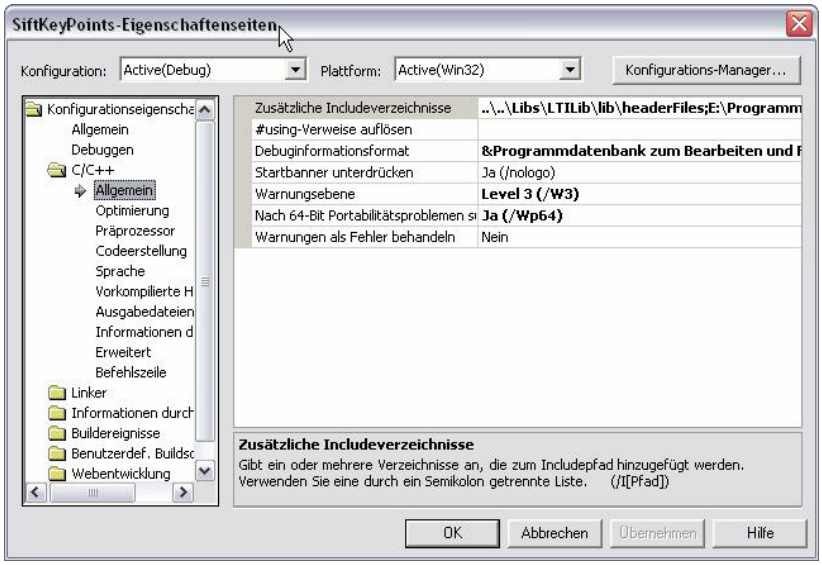

Abbildung A.1.: C/C++ allgemeine Einstellungen

Einstellung des Pfades für die Ausgabedatei des Makros als DLL-Datei in den *Konfigurationseigenschaften* des Projektes:

Für die Einbindung der OpenCV-Bibliothek in das Impresario-Projekt werden noch zusätzlich folgende DLL-Dateien aus dem Pfad

`E:\Programme\Impressario\libs\macros`

benötigt:

- cxore100.dll
- highgui100d.dll
- cv100d.dll

Ohne diese Dateien wird ein Makro mit den Funktionalitäten aus OpenCV-Bibliotheken unter Impresario nicht als Modul erkannt.

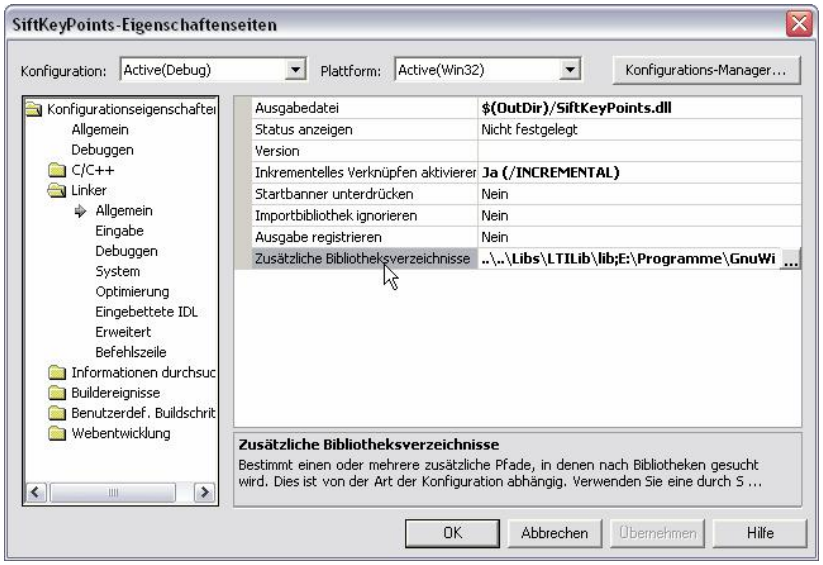

Abbildung A.2.: Linker Allgemein

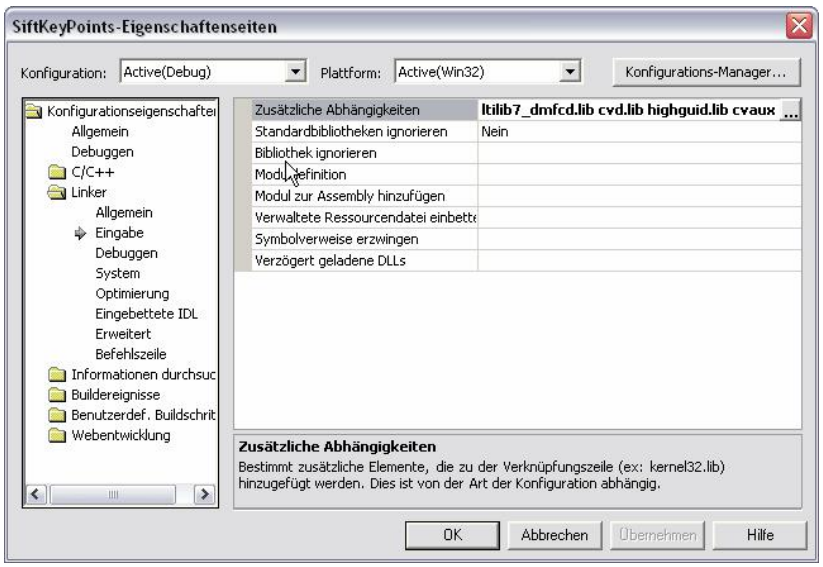

Abbildung A.3.: Linker-Eingabe

B. Quellcode

B.1. Quellcode CAMShift Tracker

```
1
2  #ifndef CamShiftTrack_h
3  #define CamShiftTrack_h
4  #include "macrotemplate.h"
5  //LTI-Lib Funktionen
6  #include <ltiCamshiftTracker.h>
7  #include <ltiDraw.h>
8  #include <sstream>
9  #include <iostream>
10
11 class MACRO_API CCamShiftTrack : public CMacroTemplate
12 {
13 public:
14     CCamShiftTrack(void);
15     virtual ~CCamShiftTrack(void);
16     virtual bool InitMacro();
17     virtual bool Apply();
18     virtual bool ExitMacro();
19     virtual void ParameterChanged(int nParameter);
20     virtual CMacroTemplate* Clone() { return new CCamShiftTrack(); }
21 private:
22     //input date
23     const lti::image *       m_InputImage;
24     const lti::channel *     m_pchnInput;
25     //output date
26     lti::image           m_ImageOut;
27     //Klassen instanzen aus der LTI-Lib
28     lti::camshiftTracker::parameters param_camShift;
29     lti::camshiftTracker camTr;
30
31     //verwendete interne Variablen
32     // Position des Trackingfensters auf dem Ausgabebild
33     lti::rectangle  outwindow;
34     std::string      ekernel;
35     int         columnSize;
36     int         rowSize;
```

```
37 };#endif

1
2 #include ".\CamShiftTrack.h"
3 /*
4 * CAMShift Tracker bassieren auf der LTI-Lib
5 * @ Hosnia Najem
6 *
7 */
8 CCamShiftTrack::CCamShiftTrack(void)
9 {
10  SetMacroName("CamShiftTrack");
11  SetMacroCreator("Hosnia Najem");
12  SetMacroGroup("Tracker");
13  SetMacroDescription("Verfolgung von Gesichter in einer Videosequenz");
14
15   //Eingangsbilder von den Impressario
16   AddMacroInput("Eingang orgina Image",
17   "Originalbild für das Zeichnen des Suchfensters ",&m_InputImage);
18   AddMacroInput("Eingang channel ProbMap",
19   "Häufigkeitsverteilung der Farbwerte",&m_pchnInput);
20   //Ausgangsbild für den Impressario
21   AddMacroOutput("Out Image",
22   "Das Ausgangsbild mit dem Suchfenster.",&m_ImageOut);
23   //Makroparametern
24   AddMacroParameter(tList,"kernelType",
25    "This indicates, how the data inside the search window should be weighted."
26    "constant: All data gets equal weight (default)."
27    "Gaussian: data in the center of search window gets more weight."
28    "Default is Constant.",
29    std::string("Gaussian"),std::string("Constant,Gaussian"));
30
31     //index 1
32    AddMacroParameter(tDouble,"sizeRatio",
33     "Size of the the window for the tracker "
34     "In order to let the tracker adapt the window size,
35     that has to be kept constant "
36     "Default is 0.0.",0.5,0.0,10.0);
37    //index 2
38    AddMacroParameter(tDouble,"threshold",
39     "This is meant for dealing with missing or insufficient
40     data within the search window set"
41     "this to a value between 0.0 and 1.0 for 0",0.0,0.0,1.0);
42 }
43
44 CCamShiftTrack::~CCamShiftTrack(void){
45 }
46
```

```
47 bool CCamShiftTrack::InitMacro(){
48     outwindow.ul.set(m_pchnInput->columns()/2,m_pchnInput->rows()/2);
49     outwindow.br.set(m_pchnInput->columns()/2,m_pchnInput->rows()/2);
50     return true;
51 }
52
53 bool CCamShiftTrack::Apply(){
54     //interne Variablen
55     bool bSuccess =true;
56     lti::rectangle null;
57     null.ul.set(0,0);
58     null.br.set(0,0);
59
60     if (m_pchnInput!= NULL){
61      columnSize= m_InputImage->columns();
62      rowSize= m_InputImage->rows();
63      m_ImageOut.resize(rowSize, columnSize);
64      //Eingsngsbild Daten auf die Ausgangsbild kopieren
65      m_ImageOut=*m_InputImage;
66     }
67     else{
68      m_ImageOut.clear();
69      return true;
70     }
71     //Setzen der Anfangsposition des Suchfensters auf die Bildmitte
72     if(outwindow.operator ==(null)){
73       outwindow.ul.set(m_pchnInput->columns()/2,m_pchnInput->rows()/2);
74       outwindow.br.set(m_pchnInput->columns()/2,m_pchnInput->rows()/2);
75     }
76     // Parameter-Initialisierung
77     std::string kernelTyp = GetParameterValue<std::string >(0);
78     if (kernelTyp=="Gaussian"){
79         param_camShift.kernelType=lti::camshiftTracker::parameters::Gaussian;
80     }
81     else{
82         param_camShift.kernelType=lti::camshiftTracker::parameters::Constant;
83     }
84
85     param_camShift.sizeRatio=GetParameterValue<double>(1);
86     param_camShift.threshold=GetParameterValue<double>(2);
87     camTr.setParameters(param_camShift);
88
89     //Aufruf des CAMSHIFT-Trackers aus der LTI-Lib
90     //outwindow: Ergebnis des rechteckigen Suchfensters
91     bSuccess= camTr.apply(*m_pchnInput,outwindow);
92     if(!bSuccess){
93        std::string strError(camTr.getStatusString());
94        std::string strMsg = "Falscher Aufruf von Camshift Tracker. ";
```

```
95          AddStatusMsg(strMsg + strError,icoError);
96      }
97      // Camshift->get-Methoden
98      //Koordinaten des Schwerpunktes des Suchfensters
99      Iti::tpoint <int> param_winCenter= camTr.getCenter();
100     //Orientierung in den Werte von (-Pi/2 to +Pi/2)
101     double param_orientation = camTr.getOrientation();
102     // Breite der Objekts in dem Suchfenster
103     double param_camWidth= camTr.getWidth();
104     //Länge des Objektes in dem Suchfenster
105     double param_laenge = camTr.getLength();
106
107     // Objektinitialisierung für Zeichen des Rechtecks im Ausgangsbild
108     Iti::draw<Iti::image::value_type> drawRed;
109     drawRed.use(m_ImageOut);
110     drawRed.setColor(Iti::Red);
111
112     // Information über das Rechteckfenster mit dem Objekt
113     Iti::tpoint<int> winUl =outwindow.ul;
114     Iti::tpoint<int> winBr=outwindow.br;
115     //Korrektur des Rechtecks auf die Größe des Bildes
116     if(winUl.y==0){
117         winUl.y = winUl.y+6;
118     }
119     if(winBr.y >= m_pchnInput->rows()-5){
120         winBr.y =winBr.y-10;
121     }
122     //Zeichnen des Rechtecks um das zu trackende Objekt
123     drawRed.rectangle(outwindow.ul,outwindow.br,false);
124
125     // Anfang: Ausgabe Variablen für Impressario!
126     Iti::tpoint<int> CenterOutWin=outwindow.getCenter();
127     //Fläche des Objektes in dem Suchfenster
128     int area = outwindow.getArea();
129
130     std::stringstream str_outwindow, str_rectCenter,str_area;
131     std::stringstream str_center,str_laenge;
132     std::stringstream str_cam_orientation, str_camWidth, str_winy;
133
134     //Wertezuordnung für die Ausgabe
135     str_area<<area;
136     str_rectCenter<< CenterOutWin;
137     str_outwindow << outwindow;
138     str_cam_orientation <<param_orientation;
139     str_laenge<<param_laenge;
140     str_camWidth<<param_camWidth;
141
142     // Ausgabe für Impressareio
```

```
143     std::string strBase =" outwindow: "+str_outwindow.str()+
144         " Mitte des Fensters  :" +str_rectCenter.str()+
145         " Area:" +str_area.str()+
146         " Orientierung: " + str_cam_orientation.str()+
147         " Laenge:  "+str_laenge.str()+
148         " Breite: "+str_camWidth.str();
149     AddStatusMsg( strBase,icoInfo);
150
151     // Ende: Ausgabe Variablen für Impressario!
152     return true;
153 }
154
155 bool CCamShiftTrack::ExitMacro()
156 {   outwindow.ul.set(m_pchnInput->columns()/2,m_pchnInput->rows()/2);
157     outwindow.br.set(m_pchnInput->columns()/2,m_pchnInput->rows()/2);
158     return true;
159 }
160
161 void CCamShiftTrack::ParameterChanged(int nParameter)
162 {   std::string Constant,Gaussian;
163     switch(nParameter)
164     {
165     case 0:
166       ekernel= GetParameterValue<std::string>(nParameter);
167     if(ekernel== "Constant"){
168        param_camShift.kernelType=Iti::camshiftTracker::parameters::Constant;
169        camTr.setParameters(param_camShift);
170     }
171     if(ekernel=="Gaussian" ){
172        param_camShift.kernelType=Iti::camshiftTracker::parameters::Gaussian;
173        camTr.setParameters(param_camShift);
174     }
175     break;
176     case 1:
177       param_camShift.sizeRatio=GetParameterValue<double>(nParameter);
178       camTr.setParameters(param_camShift);
179       break;
180     case 2:
181       param_camShift.threshold=GetParameterValue<double>(nParameter);
182       camTr.setParameters(param_camShift);
183       break;
184     }
185 }
```

B.2. Quellcode KLT Tracker

LKTracker.h

```
1
2  #ifndef LKTracker_h
3  #define LKTracker_h
4  #include "macrotemplate.h"
5  #include <ltiLkTracker.h>
6  #include <ltiPointList.h>
7  #include <ltiHarrisCorners.h>
8  #include <ltiDraw.h>
9  #include <sstream>
10 #include <iostream>
11 using std::cout;
12 using std::endl;
13 class MACRO_API CLKTracker : public CMacroTemplate
14 {
15   public:
16   CLKTracker(void);
17   virtual ~CLKTracker(void);
18   virtual bool InitMacro();
19   virtual bool Apply();
20   virtual bool ExitMacro();
21   virtual void ParameterChanged(int nParameter);
22   virtual CMacroTemplate Clone(){return new CLKTracker();
23  }
24    private:
25   //Eingangsbilder
26    const lti::channel8 *  lk_chn8Input;
27    const  lti::channel *  lk_chnInput;
28
29    //Ausgangsbild mit den gefunden Merkmalen des KLT-Trackers
30    lti::image          lk_ImageOut;
31
32    //Ergebnis-Channel vom Harris-Corner-Detektor
33    lti::channel cD_out_chan;
34     //Ausgabebild mit den Ergebnissen des Harris-Corner-Detektors
35    lti::image hkTemp_ImageOut;
36
37   //Interne Variablen
38    int rowSize, columnSize;
39    int xx , yy ;
40   // Klassenobjekt des KLT-Trackers
41    lti::lkTracker lk_obj;
42   //Parameterobjekt der Klasse KLT-Tracker
43    lti::lkTracker::parameters lk_param;
44
```

```
45    ///Speicher für die gefunden Merkmale des KLT-Trackers
46    lti::tpointList<float> featurePoints;
47    //Iterator für die Liste mit den Merkmalen
48    lti::tpointList<float>::iterator it;
49
50    //Klassen- und Parameterobjekt des Harris-Corner-Detektors
51    lti::harrisCorners harrCorn;
52    lti::harrisCorners::parameters param_harrCorn;
53 };#endif
```

LKTracker.cpp

```
1
2  #include ".\LKTracker.h"
3
4  CLKTracker::CLKTracker(void)
5  {
6    // Aufruf vom LK-Tracker,
7    //  1. Parameter: ein Channel übergeben.
8    //  Der Kanal mit den Punkten, die verfolgt werden.
9    //  Der erste Aufruf wird für Initialisierung verwendet.
10   //  Die folgenden werden verwendet,
11   //  um den ersten gegebenen Punkt-Satz zu verfolgen,
12   //  und die Ergebnisse werden in featurePoints gespeichert.
13   //  2. Parameter: enthält die Postion der verfolgten Punkte
14   //  von dem Channel, aber erst nach
15   //  dem reset().
16   //  Wenn autoCorners = true, dann enthält es die Kantenposition,
17   //  die der Autocorner gefunden hat.
18   //
19   SetMacroName("LKT Tracker");
20   SetMacroCreator("Hosnia Najem");
21   SetMacroGroup("Tracker");
22   SetMacroDescription("von Bruce D. Lucas&Takeo Kanade&Carlo Tomasi");
23
24   //Eingangsbilder Channel8 und Channel von Impresario
25   AddMacroInput("Eingang channel8","Orginal Bild Für die Ausgabe",
26   &lk_chn8Input);
27   AddMacroInput("Eingang channel","Orginal Bild Eingabe.",
28   &lk_chnInput);
29
30   // Ausgangsild für Impressario
31   AddMacroOutput("Image KeyPoints ","Bild mit den featurePoints.",
32   &lk_ImageOut);
33   AddMacroOutput("Image CornerDetector ","CornerDetector Bild .",
34   &hkTemp_ImageOut);
35
36   //Parametereinstellung für den LK-Tracker
37   // define the paramters for this class
38   AddMacroParameter(tFloat,"errorThreshold",
39   "if the difference between both matched windows"
40   "is greater than this threshold,"
41   "the point will be marked as invalid,"
42   "and it will not be tracked any more."
43   "Default value: 0.99f ",0.99f,0.00f,10.0f);
44
45   AddMacroParameter(tFloat,"eta2Threshold",
46   "This is the square of the computed pixel residual"
47   "which must be reached before the algorithm stops"
```

```
48    "Default value: 0.09f, eingebener Wert wird mit 100 devidiert;",
49     0.09f,0.0f,00.0f);
50
51    AddMacroParameter(tInt,"maxStepsPerLevel",
52    "Choose the maximal number of iteration steps per pyramid level."
53    "Default value: 10; ",10,1,100);
54
55    AddMacroParameter(tInt,"numLevels",
56    "Choose the Nummber oft the levels in the Pyramid."
57    "Default value is 4 ",4,1,10);
58
59    AddMacroParameter(tInt,"windowSize",
60    "WindowSize gives the size of the window used"
61    "for comparsion of the features."
62    "Default value is 4 ",4,0,0);
63
64    AddMacroParameter(tBool,"autoCorners",
65    "Einstellung der Harris Corner Dedektor  "
66    "if true, dann wird einen Corner detector eingesetzt.",
67    true,(true,false));
68
69    //Parameter für den Harris-Corner-Detektor
70    AddMacroParameter(tFloat,"HC_variance",
71    "Param for Cornerdetektor:Variance used for the Gaussian kernel
72    "used as window for the autocorrelation."
73    "Default value: -1f ",-1.0f,0.0f,0.0f);
74
75    AddMacroParameter(tFloat,"HC_scale",
76    "Scale factor in Harris auto-correlation."
77    "Default value: 0.04f ",0.04f,0.00f,0.00f);
78
79    AddMacroParameter(tInt,"HC_maximumCorners",
80    "Param for Cornerdetektor:
81    "Maximum number of corners to be detected."
82    "Default value: 300",300,10,600);
83
84    AddMacroParameter(tInt,"HC_kernelSize",
85    "Param for Cornerdetektor:
86    "`size of the Gaussian kernel"
87    "Default value:7",7,0,20);
88  }
```

```
1
2   CLKTracker::~CLKTracker(void){}
3   bool CLKTracker::InitMacro()
4   {   return true;}
5   bool CLKTracker::Apply()
6   {
```

```
7   //Parameter für den LK-Tracker aus Impresario setzten
8   lk_param.errorThreshold=GetParameterValue<float>(0);
9   lk_param.eta2Threshold=GetParameterValue<float>(1)/100;
10  lk_param.maxStepsPerLevel=GetParameterValue<int>(2);
11  lk_param.numLevels= GetParameterValue<int>(3);
12  lk_param.windowSize=GetParameterValue<int>(4);
13  lk_param.autoCorners =GetParameterValue<bool>(5);
14  //Herris-Corner-Detektor mit eigenen Parametern aufrufen
15    if (GetParameterValue<bool>(5) ==true){
16    //HC-Detektor-Parameter aus Impresario nehmen
17    param_harrCorn.variance =GetParameterValue<float>(6);
18    param_harrCorn.scale =GetParameterValue<float>(7);
19    param_harrCorn.maximumCorners =GetParameterValue<int>(8);
20    param_harrCorn.kernelSize =GetParameterValue<int>(9);
21    param_harrCorn.cornerValue =255;
22    param_harrCorn.noCornerValue = 0;
23    //HC-Detektor-Parameter setzen und berechnen lassen
24    harrCorn.setParameters(param\_harrCorn);
25    harrCorn.apply(lk_chnInput,cD_out_chan);
26
27    //Für den LK-Tracker den HC-Detektor aktivieren
28      lk_param.setCornerDetector(harrCorn);
29  }
30  else{
31    cD_out_chan=lk_chnInput;
32  }
33  //Parametereinstellung den LK-Tracker übergeben
34  lk_obj.setParameters(lk_param);
35
36  //Initialisierung des LK-Trackers mit dem Ergebnisbild von HC-Detektor
37  lk_obj.apply(cD_out_chan,featurePoints);
38
39  bool resTrue= lk_obj.reset();
40  if (!resTrue){
41    std::string reset ="Der Reste hat nicht geklappt in";
42    AddStatusMsg( reset+lk_obj.getStatusString(),icoInfo);
43  }
44  //Alle negative Punkte herausfiltern
45  bool pointNeg =lk_obj.reset(featurePoints);
46  if(!pointNeg){
47  std::string reset ="Reste mit den Punkten hat nicht geklappt in";
48  AddStatusMsg( reset+lk_obj.getStatusString(),icoInfo);
49  }
50  //Zeichnen auf das Bild
51  lk_ImageOut.resize(lk_chn8Input->rows(), lk_chn8Input->columns(),false,false);
52  lk_ImageOut.castFrom(lk_chn8Input);
53  hkTemp_ImageOut.castFrom(cD_out_chan);
54
```

```
55    lti::draw<lti::image::value_type> drawerKlt,drawerHC;
56    drawerKlt.use(lk_ImageOut);
57    drawerHC.use(hkTemp_ImageOut);
58    // Zugriff auf einzelne markante Punkte
59    for (it=featurePoints.begin();it!=featurePoints.end();it++) {
60      xx = (it).x;
61      yy= (it).y;
62      drawerKlt.setColor(lti::Red);
63      drawerKlt.line(xx-3,yy,xx+3,yy+1);
64      drawerKlt.line(xx,yy-3,xx,yy+3);
65      drawerHC.setColor(lti::Red);
66      drawerHC.line(xx-3,yy,xx+3,yy+1);
67      drawerHC.line(xx,yy-3,xx,yy+3);
68     }
69    return true;
70  }
71  bool CLKTracker::ExitMacro()
72  { return true;}
73
74  void CLKTracker::ParameterChanged(int nParameter)
75  {
76    switch(nParameter)
77    {
78    case 0:
79      lk_param.errorThreshold=GetParameterValue<float>(nParameter);
80      lk_obj.setParameters(lk_param);
81    break;
82    case 1:
83      lk_param.eta2Threshold=GetParameterValue<float>(nParameter);
84      lk_obj.setParameters(lk_param);
85    break;
86    case 2:
87      lk_param.maxStepsPerLevel=GetParameterValue<int>(nParameter);
88      lk_obj.setParameters(lk_param);
89    break;
90    case 3:
91      lk_param.numLevels= GetParameterValue<int>(nParameter);
92      lk_obj.setParameters(lk_param);
93    break;
94    case 4:
95      lk_param.windowSize=GetParameterValue<int>(nParameter);
96      lk_obj.setParameters(lk_param);
97    break;
98    case 5:
99      lk_param.autoCorners=GetParameterValue<bool>(nParameter);
100     lk_obj.setParameters(lk_param);
101   break;   } }
```

B.3. Quellcode SIFT Tracker

Das SiftKeys-Makro

Das SiftKeys.h

```
1
2  #ifndef SiftKeys_h
3  #define SiftKeys_h
4  #include "macrotemplate.h"
5  //LLI-Headers
6  #include <ltiImage.h>
7  #include <ltiDraw.h>
8  //OpenCV Header
9  #include <cxcore.h>
10 #include <cv.h>
11 #include <highgui.h>
12 //Interne Header
13 //sift.c: Header-Dateien  zum Erstellen von Keypoints
14 #include "sift.h"
15 #include "imgfeatures.h"
16 #include "utils.h"
17
18
19 class MACRO_API CSiftKeys :public CMacroTemplate{
20   public:
21   CSiftKeys(void);
22   virtual ~CSiftKeys(void);
23   virtual bool InitMacro();
24   virtual bool Apply();
25   virtual bool ExitMacro();
26   virtual void ParameterChanged(int nParameter);
27   virtual CMacroTemplate* Clone() { return new CSiftKeys(); }
28   private:
29   //Eingangsbilder
30   const lti::image*    m_ImagIn;
31   const lti::image*    m_ImagIn_Obj;
32   const std::basic_string
33   <char,std::char_traits<char>,class std::allocator<char> >* m_File_name;
34
35   //Ausgangsbilder
36   lti::image      m_ImageOut;
37   lti::image      m_ImageOut2;
38   //OpenCV-Bild
39   IplImage* ipl_img;
40   //Strukturen für die gefundenen Merkmale aus den Eingangsbild
41   struct feature* features;
42   struct feature* feat;
```

```
43    //Endung der Datei
44    std::string    m_strExt;
45    //Parametervariablen für Makroparameter
46    //Anzahl der Intervalle in  einer Oktave
47    int pa_intvals;
48    //Gewichtung der Pixel für  die Gauß-Faltung
49    double pa_sigma;
50    //Schwellwert für die Berechnung der Hessischen Matrix  |D(x)|
51    double pa_cont_thr;
52    //Maximaler Schwellwert für die Hauptkrümmung
53    int pa_curv_thr;
54    //Breite der Histrogrammeinträge
55    int pa_descr_with;
56    //Anzahl der Bins im Histogramm
57    int pa_descr_hist_bins;
58 };#endif
```

Das SiftKeys.ccp

```
1
2  #include ".\SiftKeys.h"
3  #include "sift.h"
4
5  CSiftKeys::CSiftKeys(void){
6    SetMacroName("SiftKeys");
7    SetMacroCreator("Hosnia Najem ");
8    SetMacroGroup("SIFT ");
9    SetMacroDescription("Find Keypoint in an Image and safe this in .Key Data!");
10   //Bildeingänge
11   AddMacroInput("Objekt File ","Objekt, File für das Ergebnis..",
12   &m_File_name);
13   AddMacroInput("Image Objekt ","Objekt, was gesucht werden soll.",
14   &m_ImagIn);
15   //Bildausgänge
16   AddMacroOutput("Image Keypoints","Image mit den gefunden Keypoints",
17   &m_ImageOut);
18   AddMacroOutput("Out for SIFTTracker ",
19   "Eingang für den SIFTTracker.",&m_ImageOut2);
20
21   // Parametereinstellung im Makro
22   //Anzahl der Intervalle in einer Oktave
23   AddMacroParameter(tInt,"S_INTVLS",
24   "Anzahl der Intervalle in einer Oktave"
25   "Default value:3 ",
26   3,1,10);
27   //Gewichtung der Pixel für die Gauß-Faltung
28   /*Benutzung in
29   *create_init_img( )--> cvSmooth(sigma)
30   *build_gauss_pyr ()
```

```
31    *calc_feature_scales()*/
32    AddMacroParameter(tDouble,"S_SIGMA",
33    "Gewichtung der Pixel für die Gauß-Faltung"
34    "Default value:1.6 ",
35    1.6,0.5,20.0);
36    /*Benutzung in scale_space_extrema(..)*/
37    AddMacroParameter(tDouble,"S_CONTR_THR",
38    "Schwellwert für die Berechnung der Hessischen Matrix |D(x)|"
39    "Default-Wert: 0.04",
40    0.04,0.00,0.00);
41    /*Benutzung in scale_space_extrema
42    *Hauptkrümmung Eigenwert der Hessischen Matrix.*/
43    AddMacroParameter(tInt," S_CURV_THR",
44    "Maximaler Schwellwert für die Hauptkrümmung"
45    "Default-Wert: 10",
46    10,1,30);
47    //Benutzung in compute_descriptors(..)*/
48    AddMacroParameter(tInt,"S_DESCR_WIDTH",
49    "Breite der Histrogrammeinträge"
50    "Default-Wert: 4 ",
51    4,1,32);
52    /*Benutzung in compute_descriptors(..)*/
53    AddMacroParameter(tInt,"S_DESCR_HIST_BINS ",
54    "Anzahl der Bins im Histogramm für den Descriptor-Array "
55    "Default Wert: 8",
56    8,1,16);
57  }
58
59  //Konstruktur
60  CSiftKeys::~CSiftKeys(void){}
61  bool CSiftKeys::InitMacro(){return true;}
62  //Main-Methode
63  bool CSiftKeys::Apply(){
64    //Anfang der Parameternabfrage
65    //Anfang: Parameter zum Aufruf von sift_features(..)
66    pa_intvals=GetParameterValue<int>(0);
67    double pa_sigma=GetParameterValue<double>(1);
68    double pa_cont_thr=GetParameterValue<double>(2);
69    pa_curv_thr=GetParameterValue<int>(3);
70    pa_descr_with=GetParameterValue<int>(4);
71    pa_descr_hist_bins=GetParameterValue<int>(5);
72    //Ende Parameter zum Aufruf von sift_features(..)
73
74    //Mit dem Übergabeparameter: m_File_name wird eine neue
75    //Datei mit der Endung .key erstellt:
76    m_strExt="key"; // endung angeben
77    char *expr =  new char[m_strExt.length()];
78    strcpy(expr,m_strExt.c_str());
```

```
79    // Änderung der Filnamen von .jpg zur .key
80    char * c_neuFile = replace_extension(m_File_name->c_str(),expr);
81
82    // Image initialisieren
83    int columnSize= m_ImagIn->columns();
84    int rowSize= m_ImagIn->rows();
85    m_ImageOut.clear();
86    m_ImageOut.resize(rowSize,columnSize,false, false);
87
88    /*Konvertierung das Eingangsbild von LTI-Lib-Format(lti::imge)
89    *zum penCV-Format (IplImage* ipl_img)
90    */
91    ipl_img= create_Ipl_img(m_ImagIn);
92    /*Aufruf der Klasse sift.c mit den Eingangsvariablen
93    *Extraktion der markanten Punkte und ihrer Merkmale aus dem Eingangsbild
94    *Rückgabeparameter: Anzahl der gefundenen Merkmale
95    */
96    int anzahl_feat=sift_features(ipl_img,&features,pa_intvals,
97              pa_sigma,pa_cont_thr,pa_curv_thr,1,pa_descr_with,pa_descr_hist_bins);
98
99    //Die markanten Punkte werden auf das Bild gezeichnet
100   draw_features( ipl_img, features, anzahl_feat);
101
102   /*Konvertierung der ipl_img von OpenCV-Format zum LTI-Lib-Format*/
103   m_ImageOut=Creat_LTI_Img(ipl_img);
104
105   //Merkmalspunkte speichern
106   if(c_neuFile != NULL )
107   export_features( c_neuFile, features, anzahl_feat );
108
109   //Eingangsbild für den Aufruf von SIFTTracker in Impressario
110   m_ImageOut2.clear();
111   m_ImageOut2.resize(rowSize,columnSize,false,false);
112   m_ImageOut2=*m_ImagIn;
113   return true;
114 } //Ende Appy()
115
116 bool CSiftKeys::ExitMacro(){return true;}
117 void CSiftKeys::ParameterChanged(int nParameter)
118 { switch(nParameter)
119   case 0:break;
120 }
```

Das SIFTTracker.h

```
1
2  #ifndef SIFTTracker_h
3  #define SIFTTracker_h
4
5  #include "macrotemplate.h"
6  //LTI-Lib Headers
7  #include <ltiImage.h>
8  #include <string.h>
9  #include <ltiDraw.h>
10 #include <ltiPoint.h>
11 //OpenCV-Lib Headers
12 #include <cxcore.h>
13 #include <cv.h>
14 #include <highgui.h>
15 //sift.c Header-Dateien für zum Erstellen von der Keypoints
16 #include "sift.h"
17 #include "imgfeatures.h"
18 #include "utils.h"
19 //Header-Dateien für den KT-Baum
20 #include "kdtree.h"
21 #include "utils.h"
22 class MACRO_API CSIFTTracker : public CMacroTemplate
23 {
24 public:
25   CSIFTTracker(void);
26   virtual ~CSIFTTracker(void);
27   virtual bool InitMacro();
28   virtual bool Apply();
29   virtual bool ExitMacro();
30   virtual void ParameterChanged(int nParameter);
31   virtual CMacroTemplate* Clone() { return new CSIFTTracker(); }
32
33 private:
34   //LTI-Lib:Eingangsbilder
35   const lti::image*    m_Img_ObjInt;
36   const lti::image*    m_ImgInput;
37   //Datei Pfade der Eingangsbilder
38   const std::basic_string
39   <char,std::char_traits<char>,class std::allocator<char> >* m_str_name;
40   const std::basic_string
41    <char,std::char_traits<char>,class std::allocator<char> >* m_File_obj;
42   //LTI-Lib:Ausgangsbilder
43   lti::image      m_ImgOut;
44   lti::image      m_ImgOut_obj;
45   lti::image   m_OPVImget;
46   //OpenCV-Bild
47   IplImage* ipl_img;
```

```
48    //SIFT Variabeln
49    //Endung der Datei und File-Namen
50    std::string    m_strExt;
51    char* c_obj_file;
52    //Strukturen für die gefundenen markanten Punkte aus den Eingangsbildern
53    struct feature* obj_feat;//für das Objket-Image
54    struct feature* img_feat; //für die aktuellen Bildsequenz
55    //Anzahl der gefundenen markanten Punkte
56    int n_obj;
57    int n_Img;
58    //Variablen für den kd-Baum
59    struct kd_node* kd_root;
60    struct feature* feat;
61    struct feature** nbrs;
62    int k, i, m;
63    double d0, d1;
64    CvPoint pt1, pt2;
65    bool KeyDateiLaden,KeyDateiErstellen;
66 };#endif
```

Das SIFTTracker.ccp

```
1
2  #include ".\SIFTTracker.h"
3
4  CSIFTTracker::CSIFTTracker(void)
5  {
6    /*Notiz::
7    *Den Pfad kann ich mir speichern. Das Bild(.jpg) muss in dem
8    *gleichen Verzeichnis liegen wie die erstelle Datei (.key)
9    */
10   SetMacroName("SIFTTracker");
11   SetMacroCreator("Hosnia Najem ");
12   SetMacroGroup("SIFT ");
13   SetMacroDescription("Sucht die  ShiftKeyPoint in einer Viedeosequenz");
14   //Macro: Eingänge
15   AddMacroInput("Objekt File ","Objekt, Pfad für das Ergebnis.",&m_File_obj);
16   AddMacroInput("Image Objket","Objekt, was gesucht werden soll.",&m_Img_ObjInt);
17   AddMacroInput("Image Sequenz","Sequenz von dem gesuchten Objekt",&m_ImgInput);
18   //Macro:Ausgänge
19   AddMacroOutput("Ausgang Sequenz","Image mit den korrespondierenden Punkten",
20   &m_ImgOut);
21   AddMacroOutput("Ausgang Objekt","Objekt mit den korrespondierenden Punkten",
22   &m_ImgOut_obj);
23
24
25   //Makro:Parametereinstellung
26   //Anzahl der Intervalle in einer Oktave
27   AddMacroParameter(tInt,"S_INTVLS",
```

```
28      "Anzahl der Intervalle in einer Oktave"
29      "Default value:3 ", 3,1,25);
30
31
32    //Gewichtung der Pixel für die Gauß-Faltung
33    AddMacroParameter(tDouble,"S_SIGMA",
34      "Gewichtung der Pixel für die Gauß-Faltung"
35      "Default value:1.6 ",
36      1.6,0.5,20.0);
37
38    // index 2
39    AddMacroParameter(tDouble,"S_CONTR_THR",
40      "Schwellwert für die Berechnung der Hesse-Matrix |D(x)|"
41      "Default Wert : 0.04",0.04,0.00,0.00);
42
43    // index 3
44    AddMacroParameter(tInt," S_CURV_THR",
45      "Maximaler Schwellwert für die Hauptkrümmung"
46      "Default Wert : 10",10,1,30);
47
48    // index 4
49    AddMacroParameter(tInt,"S_DESCR_WIDTH",
50      "Breite der Histrogrammeinträge"
51      "Default Wert: 4 ",4,1,32);
52
53    // index 5
54    AddMacroParameter(tInt,"S_DESCR_HIST_BINS ",
55      "Anzahl der Bins im Histogrammf für den Descriptor-Array "
56      "Default Wert: 8",8,1,16);
57
58    //index 6 NN_SQ_DIST_RATIO_THR
59      AddMacroParameter(tDouble,"Max_NN__THR",
60      "Faktor als Schwellwert für den Abstand zwischen
61      den Nachbarnpunkt1 und den Nachbarpunkt2"
62      "Default value:0.49 ",0.49,0.00,0.00);
63
64    //index 7
65      //#define KDTREE_BBF_MAX_NN_CHKS 200
66    AddMacroParameter(tInt,"KDT_MAX_NN ",
67      "Maximale Anzahl für die Suche der Nachbarkandidaten
68      für Neartst-Neighbors und Best Bin First Verfahren."
69      "Default Wert: 200",200,1,400);
70
71    // index 8:true:Die Features aus der.key Datei laden
72    AddMacroParameter(tBool,"KeyDateiLaden",
73      "Extrahierte Punkte aus einer Datei laden"
74      "Default:true ",true,true,false);
75
```

```
76    // index 9:ture:Erstelle eine .Key Datei für die Features
77    AddMacroParameter(tBool,"KeyDateiErstellen",
78      "Keypoints dymanisch aus dem Objek-Eingangsbild erstellen"
79      "Default:false ",false,true,false);
80  }
81  CSIFTTracker::~CSIFTTracker(void)
82  {}
83  bool CSIFTTracker::InitMacro()
84  {    return true;
85  }
86  bool CSIFTTracker::Apply()
87  {
88    //Anfgang der Parametern Abfrage
89    //Anfgang:Parametern zum Aufruf von sift_features(..)
90    int pa_intvals=GetParameterValue<int>(0);
91    double pa_sigma=GetParameterValue<double>(1);
92    double pa_cont_thr=GetParameterValue<double>(2);
93    int pa_curv_thr=GetParameterValue<int>(3);
94    int pa_descr_with=GetParameterValue<int>(4);
95    int pa_descr_hist_bins=GetParameterValue<int>(5);
96    double Max_NN__THR =GetParameterValue<double>(6);
97    int KDT_MAX_NN =GetParameterValue<int>(7);
98    bool KeyDateiLaden=GetParameterValue<bool> (8);
99    bool KeyDateiErstellen=GetParameterValue<bool> (9);
100
101   //Ende Parametern zum Aufruf von sift_features(..)
102   if(m_ImgInput!=NULL){
103     // Initialesierung der Ausgangsimage
104     m_ImgOut.clear();
105     m_ImgOut.resize(m_ImgInput->rows(),m_ImgInput->columns(),false,false);
106
107     // LTI Image Eingang zur IPL-Imge convertieren
108     IplImage* ipl_img= create_Ipl_img(m_ImgInput);  //
109     IplImage* ipl_ObjInt=create_Ipl_img(m_Img_ObjInt);
110
111     //Änderung der Filenamen Endungen von .jpg zur .key
112     m_strExt="key"; // endung angeben
113     char *expr =  new char[m_strExt.length()];
114
115     //Endung von sdt::string zu char*konvertiern
116     strcpy(expr,m_strExt.c_str());
117
118     // neue Datei erstellen
119     c_obj_file = replace_extension(m_File_obj->c_str(),expr);
120
121     //******  Objekt Merkmale laden und Bild Merkmale extrahieren!! ********/
122     //Lade aus der.key-Datei dien extrahiert markanten Punkte ihre descriptoren
123     //(Featurpoints/KeyPoints),falls diese bereits gespeichert ist.
```

```
124     if (KeyDateiLaden == true){
125         //Die markante Punkte und ihren Descriptor aus der .key Datei laden
126         //und in den struct feat speichern
127         n_obj = import_features(c_obj_file,&obj_feat );
128     }
129     //Erstelle aus den Objekt Eingangsbild die Keypoints und speichere dies ab
130     if(KeyDateiErstellen==true){
131
132       // Keypoints aus dem Objekt Image erstellen und in den obj_feat speichern
133       //Aufruf der er Klasse sift.c mit den Eingangs variablen
134       //Extraktion die markante Punkte und Ihre Merkmale aus dem Eingangsbild
135       //Rückgabeparameter: Anzahl der gefundenen Merkmale
136       n_obj=sift_features(ipl_ObjInt,&obj_feat,pa_intvals,pa_sigma,
137       pa_cont_thr,pa_curv_thr,1,pa_descr_with,pa_descr_hist_bins);
138
139       /*Konvertierung das Eingangsbild von LTI-Lib-Format(lti::imge)*/
140       /*zum penCV-Format (IplImage* ipl_img)*/
141       ipl_img= create_Ipl_img(m_ImgInput);
142
143       //Keypoints Merkmalspunkte und Descriptoren in .key Datei speichern
144       if( c_obj_file != NULL && n_obj!=0 ){
145          export_features( c_obj_file, obj_feat, n_obj );
146       }
147     }
148     //Nachdem markante Punkte und Descriptor aus Objektbild bestimmt
149     //sind berechne dies auch für das aktuelle Bild der Bildsequenz.
150     if(n_obj!=0 && ipl_img!=NULL){
151         n_Img= sift_features(ipl_img,&img_feat,pa_intvals,pa_sigma,
152                 pa_cont_thr,pa_curv_thr,1,pa_descr_with,pa_descr_hist_bins);
153       }
154
155     /********K-Tree Baum Erstellung und Korrespondenzen**********/
156     lti::draw<lti::image::value_type> drawImg;
157     drawImg.use(m_ImgOut);
158     drawImg.setColor(lti::Blue);
159     lti::tpoint<int> pt;
160     kd_root = kdtree_build( img_feat, n_Img );
161     m_ImgOut=*m_ImgInput;
162     m_OPVImget.resize(m_ImgInput->rows(),m_ImgInput->columns(),false,false);
163     //Ausgangsbild mit dem Eingangsbild zum Zeichen der Punkte belegen.
164     m_OPVImget=*m_ImgInput;
165     for( i = 0; i < n_obj; i++ )
166     {
167       feat = obj_feat + i;
168       k = kdtree_bbf_knn( kd_root, feat, 2, &nbrs, KDT_MAX_NN );
169       int  opx= cvRound(obj_feat->x);
170       int  opy=cvRound(obj_feat->y);
171       draw_x(ipl_ObjInt,pt1,4,4,CV_RGB(225,0,0));
```

```
172         if( k == 2 )
173         { d0 = descr_dist_sq( feat, nbrs[0] );
174           d1 = descr_dist_sq( feat, nbrs[1] );
175           if( d0 < d1 * Max_NN_THR )
176           {
177             int X=cvRound( nbrs[0]->x );
178             int Y=cvRound(nbrs[0]->y);
179             //Zeichne einen Kreuz auf den Ausgangsbild
180             drawImg.line(X-4,Y,X+4,Y);
181             drawImg.line(X,Y-4,X,Y+4);
182             drawImg.setColor(Iti::Red);
183           }
184         }
185         free( nbrs );
186       }
187     }
188     return true;
189 }//End Apply-Methode
190 bool CSIFTTracker::ExitMacro(){ return true;}
191 void CSIFTTracker::ParameterChanged(int nParameter)
192 { switch(nParameter){ case 0:break;}
193 }
```

B.4. Quellcode Lowe Sift-Klasse

Sift.h

```
/**@file Functions for detecting SIFT image features.
For more information, refer to:
Lowe, D. Distinctive image features from scale-invariant keypoints.
<EM>International Journal of Computer Vision, 60</EM>, 2 (2004),
pp.91--110.
Note: The SIFT algorithm is patented in the United States and cannot be
used in commercial products without a license from the University of
British Columbia. For more information, refer to the file LICENSE.ubc
that accompanied this distribution.
*/
#ifndef SIFT_H
#define SIFT_H
#include "cxcore.h"
#include <ltiImage.h>

/******************struct feature*****************/
/** holds feature data relevant to detection */
struct detection_data
{
  int r;
  int c;
  int octv;
  int intvl;
  double subintvl;
  double scl_octv;
};
struct feature;

/******************Defines************************/
/* 8 width of border in which to ignore keypoints */
#define SIFT_IMG_BORDER 5
//scale_space_extrema(..) ruft auf
/* 9 maximum steps of keypoint interpolation before failure */
#define SIFT_MAX_INTERP_STEPS 5
/* 10 default number of bins in histogram for orientation assignment */
#define SIFT_ORI_HIST_BINS 36 // OK
/* 11 determines gaussian sigma for orientation assignment */
#define SIFT_ORI_SIG_FCTR 1.5 //OK
/* 12 determines the radius of the region used in orientation assignment */
#define SIFT_ORI_RADIUS 3.0 * SIFT_ORI_SIG_FCTR
/* 13 number of passes of orientation histogram smoothing */
#define SIFT_ORI_SMOOTH_PASSES 2
/* 14 orientation magnitude relative to max that results in new feature */
#define SIFT_ORI_PEAK_RATIO 0.8    //calc_feature_oris()
```

```
45 /* 15 determines the size of a single descriptor orientation histogram */
46 #define SIFT_DESCR_SCL_FCTR 3.0
47 /* 16 threshold on magnitude(== Krümmung) of elements of descriptor vector */
48 #define SIFT_DESCR_MAG_THR 0.2
49 /* factor used to convert floating-point descriptor to unsigned char */
50 #define SIFT_INT_DESCR_FCTR 256.0
51 /* returns a feature's detection data */
52 #define feat_detection_data(f) ( (struct detection_data*)(f->feature_data) )
53
54 /******************Function Prototypes ************************/
55 // IplImge -> LTI::Imge Konvertiert
56 //Param_input: Initialisierten IplImge
57 // Ergebniss: Einen LTI::image als rückgabeparameter.
58 extern lti::image Creat_LTI_Img(void* img );
59
60 //LTI::Imge --> IplImge Konvertierung
61 // Param_input: lti::imge als Bildeingang übergeben
62 // Ergibniss: einen IPL Image wird zurückgegeben
63 extern IplImage* create_Ipl_img(const lti::image* img);
64 /*
65 Finda SIFT features in an image using user-specified parameter values.  All
66 detected features are stored in the array pointed to by \a feat.
67 @param img the image in which to detect features
68 @param feat a pointer to an array in which to store detected features
69 @param intvls the number of intervals sampled per octave of scale space
70 @param sigma the amount of Gaussian smoothing applied to each image level
71 before building the scale space representation for an octave
72 @param contr_thr a threshold on the value of the scale space function
73 \f$\left|D(\hat{x})\right|\f$, where \f$\hat{x}\f$ is a vector specifying
74 feature location and scale, used to reject unstable features;  assumes
75 pixel values in the range [0, 1]
76 @param curv_thr threshold on a feature's ratio of principle curvatures
77 used to reject features that are too edge-like
78 @param img_dbl should be 1 if image doubling prior to scale space
79 construction is desired or 0 if not
80 @param descr_width the width, \f$n\f$, of the \f$n \times n\f$ array of
81 orientation histograms used to compute a feature's descriptor
82 @param descr_hist_bins the number of orientations in each of the
83 histograms in the array used to compute a feature's descriptor
84
85 @return Returns the number of keypoints stored in \a feat or -1 on failure
86 @see sift_features()
87 */
88 extern int sift_features( IplImage* img, struct feature** feat, int intvls,
89             double sigma, double contr_thr, int curv_thr,
90             int img_dbl, int descr_width, int descr_hist_bins );
91
92 /** Für den Zugriff der Funktion von anderen Klassen***/
```

```
93 /** Damit der Zugriff auf die Methoden von anderen Klassen möglich ist,*/
94 /** müssen die Mehtoden mit "extern"gekenntzeichnet sein.*/
95
96 extern IplImage* create_init_img( IplImage*, int, double );
97 extern IplImage* convert_to_gray32( IplImage* );
98 extern IplImage*** build_gauss_pyr( IplImage*, int, int, double );
99 extern IplImage* downsample( IplImage* );
100 extern IplImage*** build_dog_pyr( IplImage***, int, int );
101 extern CvSeq* scale_space_extrema(IplImage***,int,int,double,int,CvMemStorage*);
102 extern int is_extremum( IplImage***, int, int, int, int );
103 extern struct feature* interp_extremum(IplImage***,int,int,int,int,int,double);
104 extern void interp_step(IplImage***,int,int,int,int,double*,double*,double*);
105 extern CvMat* deriv_3D( IplImage***, int, int, int, int );
106 extern CvMat* hessian_3D( IplImage***, int, int, int, int );
107 extern double interp_contr(IplImage***,int,int,int,int,double,double,double);
108 extern struct feature* new_feature( void );
109 extern int is_too_edge_like( IplImage*, int, int, int );
110 extern void calc_feature_scales( CvSeq*, double, int );
111 extern void adjust_for_img_dbl( CvSeq* );
112 extern void calc_feature_oris( CvSeq*, IplImage*** );
113 extern double* ori_hist( IplImage*, int, int, int, int, double );
114 extern int calc_grad_mag_ori( IplImage*, int, int, double*, double* );
115 extern void smooth_ori_hist( double*, int );
116 extern double dominant_ori( double*, int );
117 extern void add_good_ori_features(CvSeq*,double*,int,double,struct feature*);
118 extern struct feature* clone_feature( struct feature* );
119 extern void compute_descriptors( CvSeq*, IplImage***, int, int );
120 extern double*** descr_hist( IplImage*, int, int, double, double, int, int );
121 extern void interp_hist_entry(double***,double,double,double,double,int,int);
122 extern void hist_to_descr( double***, int, int, struct feature* );
123 extern void normalize_descr( struct feature* );
124 extern int feature_cmp( void*, void*, void* );
125 extern void release_descr_hist( double****, int );
126 extern void release_pyr( IplImage****, int, int );
127 #endif
```

Sift.c

```
1 /*
2 Functions for detecting SIFT image features.
3 For more information, refer to:
4 Lowe, D. Distinctive image features from scale-invariant keypoints.
5 <EM>International Journal of Computer Vision, 60</EM>, 2 (2004),
6 pp.91--110.
7 Note: The SIFT algorithm is patented in the United States and cannot be
8 used in commercial products without a license from the University of
9 British Columbia. For more information, refer to the file LICENSE.ubc
10 that accompanied this distribution.
11 */
```

```
12 #include "sift.h"
13 #include "imgfeatures.h"
14 #include "utils.h"
15 //OpenCV Libs
16 #include <cxcore.h>
17 #include <cv.h>
18 #include <math.h>
19 //****** Umwandlung von den Bildern von LTI--> OpenCV und OpenCV --> LTI****
20 //Code by Hosnia Najem
21 //2.VARIANTE einer IplImge -> LTI::Imge Konvertiert
22 //Param_input: Initialisierten IplImge
23 // Ergebniss: Einen LTI::image als rückgabeparameter.
24 lti::image Creat_LTI_Img(void* img ){
25
26   IplImage* tmpImg = reinterpret_cast<IplImage*>(img);
27   lti::image ImgOut;
28
29   int nl= tmpImg->height;
30   int nc= tmpImg->width;
31   ImgOut.clear();
32   ImgOut.resize(nl,nc,false,false);
33   CvScalar scal;
34   int r=0,g=0,b=0;
35   for (int i=0; i<nl; i++) {
36     for (int j=0; j<nc; j++) {
37       // scal bauen für von den ImpelImage !!
38       scal=cvGetAt(tmpImg, i, j);
39       b = static_cast<int> (scal.val[0]);
40       g=static_cast<int> (scal.val[1]);
41       r=static_cast<int> (scal.val[2]);
42       // Implimage in den Lti Image kopieren !!
43       ImgOut.at(i,j).set(r,g,b);
44     }
45   }
46   return ImgOut;
47 }
48 ////Code by Hosnia Najem
49 //  LTI::Imge --> IplImge Konvertierung
50 // Param_input: lti::imge als Bildeingang übergeben
51 // Ergibniss: einen IPL Image wird zurückgegeben
52 IplImage* create_Ipl_img(const lti::image* img){
53
54   const lti::image* litimge = reinterpret_cast<const lti::image*>(img);
55   IplImage* iplImage;
56   int columS= litimge->columns();
57   int rowS= litimge->rows();
58   CvScalar salar;
59   iplImage = cvCreateImage(cvSize(columS,rowS),IPL_DEPTH_8U,3);
```

```
60
61    for (int y=0; y<iplImage->height;y++){
62      for(int x=0; x<iplImage->width;x++){
63
64        salar.val[2] =litimge->at(y,x).getRed();
65        salar.val[1]= litimge->at(y,x).getGreen();
66        salar.val[0]= litimge->at(y,x).getBlue();
67        // OpenCV Bild beschreiben !!!
68        cvSetAt(iplImage,salar,y,x);
69      }
70    }
71
72    return iplImage;
73  }
74  /*****************Functions prototyped in sift.h ********************/
75
76  /**
77  Finds SIFT features in an image using user-specified parameter values.  All
78  detected features are stored in the array pointed to by \a feat.
79
80  @param img the image in which to detect features
81  @param fea a pointer to an array in which to store detected features
82  @param intvls the number of intervals sampled per octave of scale space
83  @param sigma the amount of Gaussian smoothing applied to each image level
84  before building the scale space representation for an octave
85  @param cont_thr a threshold on the value of the scale space function
86  \f$\left|D(\hat{x})\right|\f$, where \f$\hat{x}\f$ is a vector specifying
87  feature location and scale, used to reject unstable features;  assumes
88  pixel values in the range [0, 1]
89  @param curv_thr threshold on a feature's ratio of principle curvatures
90  used to reject features that are too edge-like
91  @param img_dbl should be 1 if image doubling prior to scale space
92  construction is desired or 0 if not
93  @param descr_width the width, \f$n\f$, of the \f$n \times n\f$ array of
94  orientation histograms used to compute a feature's descriptor
95  @param descr_hist_bins the number of orientations in each of the
96  histograms in the array used to compute a feature's descriptor
97
98  @return Returns the number of keypoints stored in \a feat or -1 on failure
99  @see sift_keypoints()
100 */
101 int sift_features( IplImage* img, struct feature** feat, int intvls,
102           double sigma, double contr_thr, int curv_thr,
103           int img_dbl, int descr_width, int descr_hist_bins )
104 {
105   CvSize imageSize = cvSize( img->width,img->height);
106   IplImage* init_img = cvCreateImage( imageSize, IPL_DEPTH_8U, 3);
107   IplImage*** gauss_pyr, *** dog_pyr;
```

```
108   CvMemStorage* storage;
109   CvSeq* features;
110   int octvs, i, n = 0;
111
112   /* check arguments */
113   if( ! img )
114     fatal_error( "NULL pointer error, %s, line %d", __FILE__, __LINE__ );
115
116   if( ! feat )
117     fatal_error( "NULL pointer error, %s, line %d", __FILE__, __LINE__ );
118   double min = static_cast<double> (MIN( init_img->width, init_img->height ));
119   /* build scale space pyramid; smallest dimension of top level is ~4 pixels */
120   init_img = create_init_img( img, img_dbl, sigma );
121   octvs = static_cast<int>(log( min) / log(2.0) - 2);
122   gauss_pyr = build_gauss_pyr( init_img, octvs, intvls, sigma );
123   dog_pyr = build_dog_pyr( gauss_pyr, octvs, intvls );
124
125   storage = cvCreateMemStorage(256000 );
126   features=scale_space_extrema(dog_pyr,octvs,intvls,contr_thr,curv_thr,storage);
127   calc_feature_scales( features, sigma, intvls );
128   if( img_dbl )
129     adjust_for_img_dbl( features );
130   calc_feature_oris( features, gauss_pyr );
131   compute_descriptors( features, gauss_pyr, descr_width, descr_hist_bins );
132
133   /* sort features by decreasing scale and move from CvSeq to array */
134   cvSeqSort( features, (CvCmpFunc)feature_cmp, NULL );
135   n = features->total;
136   *feat = (struct feature*)calloc( n, sizeof(struct feature) );
137   *feat = (struct feature*)cvCvtSeqToArray( features, *feat, CV_WHOLE_SEQ );
138   for( i = 0; i < n; i++ )
139   {
140     free( (*feat)[i].feature_data );
141     (*feat)[i].feature_data = NULL;
142   }
143
144   cvReleaseMemStorage( &storage );
145   cvReleaseImage( &init_img );
146   release_pyr( &gauss_pyr, octvs, intvls + 3 );
147   release_pyr( &dog_pyr, octvs, intvls + 2 );
148   return n;
149 }
150 /*****************Functions prototyped here *************************/
151 /*Converts an image to 8-bit grayscale and Gaussian-smooths it.  The image is
152 optionally doubled in size prior to smoothing.
153
154 @param img input image
155 @param img_dbl if true, image is doubled in size prior to smoothing
```

```
156 @param sigma total std of Gaussian smoothing
157 */
158 IplImage* create_init_img( IplImage* img, int img_dbl, double sigma )
159 {
160   IplImage* gray, * dbl;
161   float sig_diff;
162
163   gray = convert_to_gray32( img );
164   if( img_dbl )
165   {
166     sig_diff =sqrtf(static_cast <float>(sigma * sigma - 0.5f * 0.5f * 4));
167     dbl = cvCreateImage(cvSize(img->width*2, img->height*2 ),IPL_DEPTH_32F,1);
168     cvResize(gray, dbl, CV_INTER_CUBIC );
169     cvSmooth(dbl, dbl, CV_GAUSSIAN, 0, 0, sig_diff, sig_diff );
170     cvReleaseImage(&gray);
171     return dbl;
172   }
173   else
174   {
175     sig_diff = sqrt(static_cast <float>(sigma * sigma - 0.5f * 0.5f) );
176     cvSmooth( gray, gray, CV_GAUSSIAN, 0, 0, sig_diff, sig_diff );
177     return gray;
178   }
179 }
180 /*
181 Converts an image to 32-bit grayscale
182
183 @param img a 3-channel 8-bit color (BGR) or 8-bit gray image
184
185 @return Returns a 32-bit grayscale image
186 */
187 IplImage* convert_to_gray32( IplImage* img )
188 {
189   IplImage* gray8, * gray32;
190
191   gray8 = cvCreateImage( cvGetSize(img), IPL_DEPTH_8U, 1 );
192   gray32 = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
193
194   if( img->nChannels == 1 )
195     gray8 = (IplImage* )cvClone( img );
196   else
197     cvCvtColor( img, gray8, CV_BGR2GRAY );
198   cvConvertScale( gray8, gray32, 1.0 / 255.0, 0 );
199
200   cvReleaseImage( &gray8 );
201   return gray32;
202 }
203 /*
```

```
204 Builds Gaussian scale space pyramid from an image
205
206 @param base base image of the pyramid
207 @param octvs number of octaves of scale space
208 @param intvls number of intervals per octave
209 @param sigma amount of Gaussian smoothing per octave
210
211 @return Returns a Gaussian scale space pyramid as
212 an octvs x (intvls + 3) array*/
213
214 IplImage*** build_gauss_pyr( int intvls, double sigma )
215 {
216   IplImage*** gauss_pyr;
217   double* sig = (double*)calloc( intvls + 3, sizeof(double));
218   double sig_total, sig_prev, k;
219   int i, o;
220
221   gauss_pyr =(IplImage***)calloc( octvs, sizeof( IplImage** ) );
222   for( i = 0; i < octvs; i++ )
223     gauss_pyr[i] = (IplImage**)calloc( intvls + 3, sizeof( IplImage* ) );
224
225   /*precompute Gaussian sigmas using the following formula:
226   \sigma_{total}^2 = \sigma_{i}^2 + \sigma_{i-1}^2
227   */
228   sig[0] = sigma;
229   k = pow( 2.0, 1.0 / intvls );
230   for( i = 1; i < intvls + 3; i++ )
231   {
232     sig_prev = pow( k, i - 1 ) * sigma;
233     sig_total = sig_prev * k;
234     sig[i] = sqrt( sig_total * sig_total - sig_prev * sig_prev );
235   }
236
237   for( o = 0; o < octvs; o++ )
238     for( i = 0; i < intvls + 3; i++ )
239     {
240       if( o == 0  &&  i == 0 )
241         gauss_pyr[o][i] = cvCloneImage(base);
242
243       /* base of new octvave is halved image from end of previous octave */
244       else if( i == 0 )
245         gauss_pyr[o][i] = downsample( gauss_pyr[o-1][intvls] );
246
247       /* blur the current octave's last image to create the next one */
248       else
249       {
250         gauss_pyr[o][i] = cvCreateImage( cvGetSize(gauss_pyr[o][i-1]),
251           IPL_DEPTH_32F, 1 );
```

```
252           cvSmooth( gauss_pyr[o][i-1], gauss_pyr[o][i],
253             CV_GAUSSIAN, 0, 0, sig[i], sig[i] );
254         }
255       }
256
257       free( sig );
258       return gauss_pyr;
259   }
260   /*
261   Downsamples an image to a quarter of its size (half in each dimension)
262   using nearest-neighbor interpolation
263
264   @param img an image
265
266   @return Returns an image whose dimensions are half those of img
267   */
268   IplImage* downsample( IplImage* img )
269   {
270     IplImage* smaller = cvCreateImage( cvSize(img->width / 2, img->height / 2),
271       img->depth, img->nChannels );
272     cvResize( img, smaller, CV_INTER_NN );
273
274     return smaller;
275   }
276   /*
277   Builds a difference of Gaussians scale space pyramid by subtracting adjacent
278   intervals of a Gaussian pyramid
279
280   @param gauss_pyr Gaussian scale-space pyramid
281   @param octvs number of octaves of scale space
282   @param intvls number of intervals per octave
283
284   @return Returns a difference of Gaussians scale space pyramid as an
285   octvs x (intvls + 2) array
286   */
287   IplImage*** build_dog_pyr( IplImage*** gauss_pyr, int octvs, int intvls )
288   {
289     IplImage*** dog_pyr;
290     int i, o;
291
292     dog_pyr = (IplImage*** )calloc( octvs, sizeof( IplImage** ) );
293     for( i = 0; i < octvs; i++ )
294       dog_pyr[i] =(IplImage**) calloc( intvls + 2, sizeof(IplImage*) );
295
296     for( o = 0; o < octvs; o++ )
297       for( i = 0; i < intvls + 2; i++ )
298       {
299         dog_pyr[o][i]=cvCreateImage(cvGetSize(gauss_pyr[o][i]),IPL_DEPTH_32F,1);
```

```
300         cvSub( gauss_pyr[o][i+1], gauss_pyr[o][i], dog_pyr[o][i], NULL );
301       }
302
303       return dog_pyr;
304   }
305   /*
306   Detects features at extrema in DoG scale space.  Bad features are discarded
307   based on contrast and ratio of principal curvatures.
308
309   @param dog_pyr DoG scale space pyramid
310   @param octvs octaves of scale space represented by dog_pyr
311   @param intvls intervals per octave
312   @param contr_thr low threshold on feature contrast
313   @param curv_thr high threshold on feature ratio of principal curvatures
314   @param storage memory storage in which to store detected features
315
316   @return Returns an array of detected features whose scales, orientations,
317   and descriptors are yet to be determined.
318   */
319   CvSeq* scale_space_extrema( IplImage*** dog_pyr, int octvs, int intvls,
320                               double contr_thr, int curv_thr,
321                               CvMemStorage* storage )
322   {
323     CvSeq* features;
324     double prelim_contr_thr = 0.5 * contr_thr / intvls;
325     struct feature* feat;
326     struct detection_data* ddata;
327     int o, i, r, c;
328
329     features = cvCreateSeq( 0, sizeof(CvSeq), sizeof(struct feature), storage );
330     for( o = 0; o < octvs; o++ )
331       for( i = 1; i <= intvls; i++ )
332         for(r = SIFT_IMG_BORDER; r < dog_pyr[o][0]->height-SIFT_IMG_BORDER; r++)
333           for(c = SIFT_IMG_BORDER; c < dog_pyr[o][0]->width-SIFT_IMG_BORDER; c++)
334             /* perform preliminary check on contrast */
335             if( ABS( pixval32f( dog_pyr[o][i], r, c ) ) > prelim_contr_thr )
336               if( is_extremum( dog_pyr, o, i, r, c ) )
337               {
338                 feat = interp_extremum(dog_pyr, o, i, r, c, intvls, contr_thr);
339                 if( feat )
340                 {
341                   ddata = feat_detection_data( feat );
342                   if( ! is_too_edge_like( dog_pyr[ddata->octv][ddata->intvl],
343                     ddata->r, ddata->c, curv_thr ) )
344                   {
345                     cvSeqPush( features, feat );
346                   }
347                   else
```

```
348                     free( ddata );
349                   free( feat );
350                 }
351               }
352
353               return features;
354 }
355 /*
356 Determines whether a pixel is a scale-space extremum by comparing it to it's
357 3x3x3 pixel neighborhood.
358
359 @param dog_pyr DoG scale space pyramid
360 @param octv pixel's scale space octave
361 @param intvl pixel's within-octave interval
362 @param r pixel's image row
363 @param c pixel's image col
364
365 @return Returns 1 if the specified pixel is an extremum (max or min) among
366 it's 3x3x3 pixel neighborhood.
367 */
368 int is_extremum( IplImage*** dog_pyr, int octv, int intvl, int r, int c )
369 {
370   float val = pixval32f( dog_pyr[octv][intvl], r, c );
371   int i, j, k;
372
373   /* check for maximum */
374   if( val > 0 )
375   {
376     for( i = -1; i <= 1; i++ )
377       for( j = -1; j <= 1; j++ )
378         for( k = -1; k <= 1; k++ )
379           if( val < pixval32f( dog_pyr[octv][intvl+i], r + j, c + k ) )
380             return 0;
381   }
382
383   /* check for minimum */
384   else
385   {
386     for( i = -1; i <= 1; i++ )
387       for( j = -1; j <= 1; j++ )
388         for( k = -1; k <= 1; k++ )
389           if( val > pixval32f( dog_pyr[octv][intvl+i], r + j, c + k ) )
390             return 0;
391   }
392
393   return 1;
394 }
395 /*
```

```
396 Interpolates a scale-space extremum's location and scale to subpixel
397 accuracy to form an image feature.  Rejects features with low contrast.
398 Based on Section 4 of Lowe's paper.
399
400 @param dog_pyr DoG scale space pyramid
401 @param octv feature's octave of scale space
402 @param intvl feature's within-octave interval
403 @param r feature's image row
404 @param c feature's image column
405 @param intvls total intervals per octave
406 @param contr_thr threshold on feature contrast
407
408 @return Returns the feature resulting from interpolation of the given
409 parameters or NULL if the given location could not be interpolated or
410 if contrast at the interpolated loation was too low.  If a feature is
411 returned, its scale, orientation, and descriptor are yet to be determined.
412 */
413 struct feature* interp_extremum( IplImage*** dog_pyr, int octv, int intvl,
414   int r, int c, int intvls, double contr_thr )
415 {
416   struct feature* feat;
417   struct detection_data* ddata;
418   double xi, xr, xc, contr;
419   int i = 0;
420
421   while( i < SIFT_MAX_INTERP_STEPS )
422   {
423     interp_step( dog_pyr, octv, intvl, r, c, &xi, &xr, &xc );
424     if( ABS( xi ) < 0.5  &&  ABS( xr ) < 0.5  &&  ABS( xc ) < 0.5 )
425       break;
426
427     c += cvRound( xc );
428     r += cvRound( xr );
429     intvl += cvRound( xi );
430
431     if( intvl < 1  ||
432       intvl > intvls  ||
433       c < SIFT_IMG_BORDER  ||
434       r < SIFT_IMG_BORDER  ||
435       c >= dog_pyr[octv][0]->width - SIFT_IMG_BORDER  ||
436       r >= dog_pyr[octv][0]->height - SIFT_IMG_BORDER )
437     {
438       return NULL;
439     }
440
441     i++;
442   }
443
```

```
444   /* ensure convergence of interpolation */
445   if( i >= SIFT_MAX_INTERP_STEPS )
446     return NULL;
447
448   contr = interp_contr( dog_pyr, octv, intvl, r, c, xi, xr, xc );
449   if( ABS( contr ) < contr_thr / intvls )
450     return NULL;
451
452   feat = new_feature();
453   ddata = feat_detection_data( feat );
454   feat->img_pt.x = feat->x = ( c + xc ) * pow( 2.0, octv );
455   feat->img_pt.y = feat->y = ( r + xr ) * pow( 2.0, octv );
456   ddata->r = r;
457   ddata->c = c;
458   ddata->octv = octv;
459   ddata->intvl = intvl;
460   ddata->subintvl = xi;
461
462   return feat;
463 }
464 /*
465 Performs one step of extremum interpolation.  Based on Eqn. (3) in Lowe's
466 paper.
467
468 @param dog_pyr difference of Gaussians scale space pyramid
469 @param octv octave of scale space
470 @param intvl interval being interpolated
471 @param r row being interpolated
472 @param c column being interpolated
473 @param xi output as interpolated subpixel increment to interval
474 @param xr output as interpolated subpixel increment to row
475 @param xc output as interpolated subpixel increment to col
476 */
477
478 void interp_step( IplImage*** dog_pyr, int octv, int intvl, int r, int c,
479           double* xi, double* xr, double* xc )
480 {
481   CvMat* dD, * H, * H_inv, X;
482   double x[3] = { 0 };
483
484   dD = deriv_3D( dog_pyr, octv, intvl, r, c );
485   H = hessian_3D( dog_pyr, octv, intvl, r, c );
486   H_inv = cvCreateMat( 3, 3, CV_64FC1 );
487   cvInvert( H, H_inv, CV_SVD );
488   cvInitMatHeader( &X, 3, 1, CV_64FC1, x, CV_AUTOSTEP );
489   cvGEMM( H_inv, dD, -1, NULL, 0, &X, 0 );
490
491   cvReleaseMat( &dD );
```

```
492   cvReleaseMat( &H );
493   cvReleaseMat( &H_inv );
494
495   *xi = x[2];
496   *xr = x[1];
497   *xc = x[0];
498 }
499 /*
500 Computes the partial derivatives in x, y, and scale of a pixel in the DoG
501 scale space pyramid.
502
503 @param dog_pyr DoG scale space pyramid
504 @param octv pixel's octave in dog_pyr
505 @param intvl pixel's interval in octv
506 @param r pixel's image row
507 @param c pixel's image col
508
509 @return Returns the vector of partial derivatives for pixel I
510 { dI/dx, dI/dy, dI/ds }^T as a CvMat*
511 */
512 CvMat* deriv_3D( IplImage*** dog_pyr, int octv, int intvl, int r, int c )
513 {
514   CvMat* dI;
515   double dx, dy, ds;
516
517   dx = ( pixval32f( dog_pyr[octv][intvl], r, c+1 ) -
518     pixval32f( dog_pyr[octv][intvl], r, c-1 ) ) / 2.0;
519   dy = ( pixval32f( dog_pyr[octv][intvl], r+1, c ) -
520     pixval32f( dog_pyr[octv][intvl], r-1, c ) ) / 2.0;
521   ds = ( pixval32f( dog_pyr[octv][intvl+1], r, c ) -
522     pixval32f( dog_pyr[octv][intvl-1], r, c ) ) / 2.0;
523
524   dI = cvCreateMat( 3, 1, CV_64FC1 );
525   cvmSet( dI, 0, 0, dx );
526   cvmSet( dI, 1, 0, dy );
527   cvmSet( dI, 2, 0, ds );
528
529   return dI;
530 }
531 /*
532 Computes the 3D Hessian matrix for a pixel in the DoG scale space pyramid.
533
534 @param dog_pyr DoG scale space pyramid
535 @param octv pixel's octave in dog_pyr
536 @param intvl pixel's interval in octv
537 @param r pixel's image row
538 @param c pixel's image col
539
```

```
540 @return Returns the Hessian matrix (below) for pixel I as a CvMat*
541
542 / Ixx  Ixy  Ixs \ <BR>
543 | Ixy  Iyy  Iys | <BR>
544 \ Ixs  Iys  Iss /
545 */
546 CvMat* hessian_3D( IplImage*** dog_pyr, int octv, int intvl, int r, int c )
547 {
548   CvMat* H;
549   double v, dxx, dyy, dss, dxy, dxs, dys;
550
551   v = pixval32f( dog_pyr[octv][intvl], r, c );
552   dxx = ( pixval32f( dog_pyr[octv][intvl], r, c+1 ) +
553     pixval32f( dog_pyr[octv][intvl], r, c-1 ) - 2 * v );
554   dyy = ( pixval32f( dog_pyr[octv][intvl], r+1, c ) +
555     pixval32f( dog_pyr[octv][intvl], r-1, c ) - 2 * v );
556   dss = ( pixval32f( dog_pyr[octv][intvl+1], r, c ) +
557     pixval32f( dog_pyr[octv][intvl-1], r, c ) - 2 * v );
558   dxy = ( pixval32f( dog_pyr[octv][intvl], r+1, c+1 ) -
559     pixval32f( dog_pyr[octv][intvl], r+1, c-1 ) -
560     pixval32f( dog_pyr[octv][intvl], r-1, c+1 ) +
561     pixval32f( dog_pyr[octv][intvl], r-1, c-1 ) ) / 4.0;
562   dxs = ( pixval32f( dog_pyr[octv][intvl+1], r, c+1 ) -
563     pixval32f( dog_pyr[octv][intvl+1], r, c-1 ) -
564     pixval32f( dog_pyr[octv][intvl-1], r, c+1 ) +
565     pixval32f( dog_pyr[octv][intvl-1], r, c-1 ) ) / 4.0;
566   dys = ( pixval32f( dog_pyr[octv][intvl+1], r+1, c ) -
567     pixval32f( dog_pyr[octv][intvl+1], r-1, c ) -
568     pixval32f( dog_pyr[octv][intvl-1], r+1, c ) +
569     pixval32f( dog_pyr[octv][intvl-1], r-1, c ) ) / 4.0;
570
571   H = cvCreateMat( 3, 3, CV_64FC1 );
572   cvmSet( H, 0, 0, dxx );
573   cvmSet( H, 0, 1, dxy );
574   cvmSet( H, 0, 2, dxs );
575   cvmSet( H, 1, 0, dxy );
576   cvmSet( H, 1, 1, dyy );
577   cvmSet( H, 1, 2, dys );
578   cvmSet( H, 2, 0, dxs );
579   cvmSet( H, 2, 1, dys );
580   cvmSet( H, 2, 2, dss );
581
582   return H;
583 }
584 /*
585 Calculates interpolated pixel contrast.  Based on Eqn. (3) in Lowe's paper.
586
587 @param dog_pyr difference of Gaussians scale space pyramid
```

```
588 @param octv octave of scale space
589 @param intvl within-octave interval
590 @param r pixel row
591 @param c pixel column
592 @param xi interpolated subpixel increment to interval
593 @param xr interpolated subpixel increment to row
594 @param xc interpolated subpixel increment to col
595
596 @param Returns interpolated contrast.
597 */
598 double interp_contr( IplImage*** dog_pyr, int octv, int intvl, int r,
599           int c, double xi, double xr, double xc )
600 {
601   CvMat* dD, X, T;
602   double t[1], x[3] = { xc, xr, xi };
603
604   cvInitMatHeader( &X, 3, 1, CV_64FC1, x, CV_AUTOSTEP );
605   cvInitMatHeader( &T, 1, 1, CV_64FC1, t, CV_AUTOSTEP );
606   dD = deriv_3D( dog_pyr, octv, intvl, r, c );
607   cvGEMM( dD, &X, 1, NULL, 0, &T,  CV_GEMM_A_T );
608   cvReleaseMat( &dD );
609
610   return pixval32f( dog_pyr[octv][intvl], r, c ) + t[0] * 0.5;
611 }
612 /*
613 Allocates and initializes a new feature
614
615 @return Returns a pointer to the new feature
616 */
617 struct feature* new_feature( void )
618 {
619   struct feature* feat;
620   struct detection_data* ddata;
621
622   feat =( struct feature *)malloc( sizeof( struct feature ) );
623   memset( feat, 0, sizeof( struct feature ) );
624   ddata =(struct detection_data*) malloc( sizeof( struct detection_data ) );
625   memset( ddata, 0, sizeof( struct detection_data ) );
626   feat->feature_data = ddata;
627   return feat;
628 }
629 /*
630 Determines whether a feature is too edge like to be stable by computing the
631 ratio of principal curvatures at that feature.  Based on Section 4.1 of
632 Lowe's paper.
633
634 @param dog_img image from the DoG pyramid in which feature was detected
635 @param r feature row
```

```
636 @param c feature col
637 @param curv_thr high threshold on ratio of principal curvatures
638
639 @return Returns 0 if the feature at (r,c) in dog_img is sufficiently
640 corner-like or 1 otherwise.
641 */
642 int is_too_edge_like( IplImage* dog_img, int r, int c, int curv_thr )
643 {
644   double d, dxx, dyy, dxy, tr, det;
645
646   /* principal curvatures are computed using the trace and det of Hessian */
647   d = pixval32f(dog_img, r, c);
648   dxx = pixval32f( dog_img, r, c+1 ) + pixval32f( dog_img, r, c-1 ) - 2 * d;
649   dyy = pixval32f( dog_img, r+1, c ) + pixval32f( dog_img, r-1, c ) - 2 * d;
650   dxy = ( pixval32f(dog_img, r+1, c+1) - pixval32f(dog_img, r+1, c-1) -
651     pixval32f(dog_img, r-1, c+1) + pixval32f(dog_img, r-1, c-1) ) / 4.0;
652   tr = dxx + dyy;
653   det = dxx * dyy - dxy * dxy;
654
655   /* negative determinant -> curvatures have different signs; reject feature */
656   if( det <= 0 )
657     return 1;
658
659   if( tr * tr / det < ( curv_thr + 1.0 )*( curv_thr + 1.0 ) / curv_thr )
660     return 0;
661   return 1;
662 }
663 /*
664 Calculates characteristic scale for each feature in an array.
665
666 @param features array of features
667 @param sigma amount of Gaussian smoothing per octave of scale space
668 @param intvls intervals per octave of scale space
669 */
670 void calc_feature_scales( CvSeq* features, double sigma, int intvls )
671 {
672   struct feature* feat;
673   struct detection_data* ddata;
674   double intvl;
675   int i, n;
676
677   n = features->total;
678   for( i = 0; i < n; i++ )
679   {
680     feat = CV_GET_SEQ_ELEM( struct feature, features, i );
681     ddata = feat_detection_data( feat );
682     intvl = ddata->intvl + ddata->subintvl;
683     feat->scl = sigma * pow( 2.0, ddata->octv + intvl / intvls );
```

```
684       ddata->scl_octv = sigma * pow( 2.0, intvl / intvls );
685     }
686   }
687   /*
688   Halves feature coordinates and scale in case the input image was doubled
689   prior to scale space construction.
690
691   @param features array of features
692   */
693   void adjust_for_img_dbl( CvSeq* features )
694   {
695     struct feature* feat;
696     int i, n;
697
698     n = features->total;
699     for( i = 0; i < n; i++ )
700     {
701       feat = CV_GET_SEQ_ELEM( struct feature, features, i );
702       feat->x /= 2.0;
703       feat->y /= 2.0;
704       feat->scl /= 2.0;
705       feat->img_pt.x /= 2.0;
706       feat->img_pt.y /= 2.0;
707     }
708   }
709   /*
710   Computes a canonical orientation for each image feature in an array.  Based
711   on Section 5 of Lowe's paper.  This function adds features to the array when
712   there is more than one dominant orientation at a given feature location.
713
714   @param features an array of image features
715   @param gauss_pyr Gaussian scale space pyramid
716   */
717   void calc_feature_oris( CvSeq* features, IplImage*** gauss_pyr )
718   {
719     struct feature* feat;
720     struct detection_data* ddata;
721     double* hist;
722     double omax;
723     int i, j, n = features->total;
724
725     for( i = 0; i < n; i++ )
726     {
727       feat = (struct feature*) malloc( sizeof( struct feature ) );
728       cvSeqPopFront( features, feat );
729       ddata = feat_detection_data( feat );
730       hist = ori_hist( gauss_pyr[ddata->octv][ddata->intvl],
731         ddata->r, ddata->c, SIFT_ORI_HIST_BINS,
```

```
732         cvRound( SIFT_ORI_RADIUS * ddata->scl_octv ),
733         SIFT_ORI_SIG_FCTR * ddata->scl_octv );
734       for( j = 0; j < SIFT_ORI_SMOOTH_PASSES; j++ )
735         smooth_ori_hist( hist, SIFT_ORI_HIST_BINS );
736       omax = dominant_ori( hist, SIFT_ORI_HIST_BINS );
737       add_good_ori_features( features, hist, SIFT_ORI_HIST_BINS,
738         omax * SIFT_ORI_PEAK_RATIO, feat );
739       free( ddata );
740       free( feat );
741       free( hist );
742     }
743   }
744   /*
745   Computes a gradient orientation histogram at a specified pixel.
746   @param img image
747   @param r pixel row
748   @param c pixel col
749   @param n number of histogram bins
750   @param rad radius of region over which histogram is computed
751   @param sigma std for Gaussian weighting of histogram entries
752
753   @return Returns an n-element array containing an orientation histogram
754   representing orientations between 0 and 2 PI.
755   */
756   double* ori_hist( IplImage* img, int r, int c, int n, int rad, double sigma)
757   {
758     double* hist;
759     double mag, ori, w, exp_denom, PI2 = CV_PI * 2.0;
760     int bin, i, j;
761
762     hist = (double*)calloc( n, sizeof( double ) );
763     exp_denom = 2.0 * sigma * sigma;
764     for( i = -rad; i <= rad; i++ )
765       for( j = -rad; j <= rad; j++ )
766         if( calc_grad_mag_ori( img, r + i, c + j, &mag, &ori ) )
767         {
768           w = exp( -( i*i + j*j ) / exp_denom );
769           bin = cvRound( n * ( ori + CV_PI ) / PI2 );
770           bin = ( bin < n )? bin : 0;
771           hist[bin] += w * mag;
772         }
773
774         return hist;
775   }
776   /*
777   Calculates the gradient magnitude and orientation at a given pixel.
778
779   @param img image
```

```
780 @param r pixel row
781 @param c pixel col
782 @param mag output as gradient magnitude at pixel (r,c)
783 @param ori output as gradient orientation at pixel (r,c)
784
785 @return Returns 1 if the specified pixel is a valid one and sets mag and
786 ori accordingly; otherwise returns 0
787 */
788 int calc_grad_mag_ori( IplImage* img, int r, int c, double* mag, double* ori )
789 {
790   double dx, dy;
791
792   if( r > 0  &&  r < img->height - 1  &&  c > 0  &&  c < img->width - 1 )
793   {
794     dx = pixval32f( img, r, c+1 ) - pixval32f( img, r, c-1 );
795     dy = pixval32f( img, r-1, c ) - pixval32f( img, r+1, c );
796     *mag = sqrt( dx*dx + dy*dy );
797     *ori = atan2( dy, dx );
798     return 1;
799   }
800
801   else
802     return 0;
803 }
804 /*
805 Gaussian smooths an orientation histogram.
806
807 @param hist an orientation histogram
808 @param n number of bins
809 */
810 void smooth_ori_hist( double* hist, int n )
811 {
812   double prev, tmp, h0 = hist[0];
813   int i;
814
815   prev = hist[n-1];
816   for( i = 0; i < n; i++ )
817   {
818     tmp = hist[i];
819     hist[i] = 0.25 * prev + 0.5 * hist[i] +
820       0.25 * ( ( i+1 == n )? h0 : hist[i+1] );
821     prev = tmp;
822   }
823 }
824 /*
825 Finds the magnitude of the dominant orientation in a histogram
826
827 @param hist an orientation histogram
```

```
828 @param n number of bins
829
830 @return Returns the value of the largest bin in hist
831 */
832 double dominant_ori( double* hist, int n )
833 {
834   double omax;
835   int maxbin, i;
836
837   omax = hist[0];
838   maxbin = 0;
839   for( i = 1; i < n; i++ )
840     if( hist[i] > omax )
841     {
842       omax = hist[i];
843       maxbin = i;
844     }
845     return omax;
846 }
847 /*
848 Interpolates a histogram peak from left, center, and right values
849 */
850 #define interp_hist_peak( l, c, r ) ( 0.5 * ((l)-(r)) / ((l) - 2.0*(c) + (r)) )
851
852 /*
853 Adds features to an array for every orientation in a histogram greater than
854 a specified threshold.
855
856 @param features new features are added to the end of this array
857 @param hist orientation histogram
858 @param n number of bins in hist
859 @param mag_thr new features are added for entries in hist greater than this
860 @param feat new features are clones of this with different orientations
861 */
862 void add_good_ori_features( CvSeq* features, double* hist, int n,
863                             double mag_thr, struct feature* feat )
864 {
865   struct feature* new_feat;
866   double bin, PI2 = CV_PI * 2.0;
867   int l, r, i;
868
869   for( i = 0; i < n; i++ )
870   {
871     l = ( i == 0 )? n - 1 : i-1;
872     r = ( i + 1 ) % n;
873
874     if( hist[i] > hist[l]  &&  hist[i] > hist[r]  &&  hist[i] >= mag_thr )
875     {
```

```
876         bin = i + interp_hist_peak( hist[l], hist[i], hist[r] );
877         bin = ( bin < 0 )? n + bin : ( bin >= n )? bin - n : bin;
878         new_feat = clone_feature( feat );
879         new_feat->ori = ( ( PI2 * bin ) / n ) - CV_PI;
880         cvSeqPush( features, new_feat );
881         free( new_feat );
882       }
883     }
884 }
885 /*
886 Makes a deep copy of a feature
887
888 @param feat feature to be cloned
889
890 @return Returns a deep copy of feat
891 */
892 struct feature* clone_feature( struct feature* feat )
893 {
894   struct feature* new_feat;
895   struct detection_data* ddata;
896
897   new_feat = new_feature();
898   ddata = feat_detection_data( new_feat );
899   memcpy( new_feat, feat, sizeof( struct feature ) );
900   memcpy( ddata, feat_detection_data(feat), sizeof( struct detection_data ) );
901   new_feat->feature_data = ddata;
902
903   return new_feat;
904 }
905 /*
906 Computes feature descriptors for features in an array.  Based on Section 6
907 of Lowe's paper.
908
909 @param features array of features
910 @param gauss_pyr Gaussian scale space pyramid
911 @param d width of 2D array of orientation histograms
912 @param n number of bins per orientation histogram
913 */
914 void compute_descriptors( CvSeq* features, IplImage*** gauss_pyr, int d, int n)
915 {
916   struct feature* feat;
917   struct detection_data* ddata;
918   double*** hist;
919   int i, k = features->total;
920
921   for( i = 0; i < k; i++ )
922   {
923     feat = CV_GET_SEQ_ELEM( struct feature, features, i );
```

```
924       ddata = feat_detection_data( feat );
925       hist = descr_hist( gauss_pyr[ddata->octv][ddata->intvl], ddata->r,
926         ddata->c, feat->ori, ddata->scl_octv, d, n );
927       hist_to_descr( hist, d, n, feat );
928       release_descr_hist( &hist, d );
929     }
930   }
931   /*
932   Computes the 2D array of orientation histograms that form the feature
933   descriptor.  Based on Section 6.1 of Lowe's paper.
934
935   @param img image used in descriptor computation
936   @param r row coord of center of orientation histogram array
937   @param c column coord of center of orientation histogram array
938   @param ori canonical orientation of feature whose descr is being computed
939   @param scl scale relative to img of feature whose descr is being computed
940   @param d width of 2d array of orientation histograms
941   @param n bins per orientation histogram
942
943   @return Returns a d x d array of n-bin orientation histograms.
944   */
945   double*** descr_hist( IplImage* img, int r, int c, double ori,
946             double scl, int d, int n )
947   {
948     double*** hist;
949     double cos_t, sin_t, hist_width, exp_denom, r_rot, c_rot, grad_mag,
950       grad_ori, w, rbin, cbin, obin, bins_per_rad, PI2 = 2.0 * CV_PI;
951     int radius, i, j;
952
953     hist = (double***)calloc( d, sizeof( double** ) );
954     for( i = 0; i < d; i++ )
955     {
956       hist[i] = (double**)calloc( d, sizeof( double* ) );
957       for( j = 0; j < d; j++ )
958         hist[i][j] = (double*)calloc( n, sizeof( double ) );
959     }
960
961     cos_t = cos( ori );
962     sin_t = sin( ori );
963     bins_per_rad = n / PI2;
964     exp_denom = d * d * 0.5;
965     hist_width = SIFT_DESCR_SCL_FCTR * scl;
966     radius = static_cast<int>(hist_width*(sqrt(2.0))*(d+1.0)*0.5+0.5);
967     for( i = -radius; i <= radius; i++ )
968       for( j = -radius; j <= radius; j++ )
969       {
970         /*
971         Calculate sample's histogram array coords rotated relative to ori.
```

```
972         Subtract 0.5 so samples that fall e.g. in the center of row 1 (i.e.
973         r_rot = 1.5) have full weight placed in row 1 after interpolation.
974         */
975         c_rot = ( j * cos_t - i * sin_t ) / hist_width;
976         r_rot = ( j * sin_t + i * cos_t ) / hist_width;
977         rbin = r_rot + d / 2 - 0.5;
978         cbin = c_rot + d / 2 - 0.5;
979 
980         if( rbin > -1.0  &&  rbin < d  &&  cbin > -1.0  &&  cbin < d )
981           if( calc_grad_mag_ori( img, r + i, c + j, &grad_mag, &grad_ori ))
982           {
983             grad_ori -= ori;
984             while( grad_ori < 0.0 )
985               grad_ori += PI2;
986             while( grad_ori >= PI2 )
987               grad_ori -= PI2;
988 
989             obin = grad_ori * bins_per_rad;
990             w = exp( -(c_rot * c_rot + r_rot * r_rot) / exp_denom );
991             interp_hist_entry( hist, rbin, cbin, obin, grad_mag * w, d, n );
992           }
993       }
994 
995       return hist;
996 }
997 /*
998 Interpolates an entry into the array of orientation histograms that form
999 the feature descriptor.
1000 
1001 @param hist 2D array of orientation histograms
1002 @param rbin sub-bin row coordinate of entry
1003 @param cbin sub-bin column coordinate of entry
1004 @param obin sub-bin orientation coordinate of entry
1005 @param mag size of entry
1006 @param d width of 2D array of orientation histograms
1007 @param n number of bins per orientation histogram
1008 */
1009 void interp_hist_entry( double*** hist, double rbin, double cbin,
1010                         double obin, double mag, int d, int n )
1011 {
1012   double d_r, d_c, d_o, v_r, v_c, v_o;
1013   double** row, * h;
1014   int r0, c0, o0, rb, cb, ob, r, c, o;
1015 
1016   r0 = cvFloor( rbin );
1017   c0 = cvFloor( cbin );
1018   o0 = cvFloor( obin );
1019   d_r = rbin - r0;
```

```
1020   d_c = cbin - c0;
1021   d_o = obin - o0;
1022
1023   /*
1024   The entry is distributed into up to 8 bins.  Each entry into a bin
1025   is multiplied by a weight of 1 - d for each dimension, where d is the
1026   distance from the center value of the bin measured in bin units.
1027   */
1028   for( r = 0; r <= 1; r++ )
1029   {
1030     rb = r0 + r;
1031     if( rb >= 0  &&  rb < d )
1032     {
1033       v_r = mag * ( ( r == 0 )? 1.0 - d_r : d_r );
1034       row = hist[rb];
1035       for( c = 0; c <= 1; c++ )
1036       {
1037         cb = c0 + c;
1038         if( cb >= 0  &&  cb < d )
1039         {
1040           v_c = v_r * ( ( c == 0 )? 1.0 - d_c : d_c );
1041           h = row[cb];
1042           for( o = 0; o <= 1; o++ )
1043           {
1044             ob = ( o0 + o ) % n;
1045             v_o = v_c * ( ( o == 0 )? 1.0 - d_o : d_o );
1046             h[ob] += v_o;
1047           }
1048         }
1049       }
1050     }
1051   }
1052 }
1053 /*
1054 Converts the 2D array of orientation histograms into a feature's descriptor
1055 vector.
1056
1057 @param hist 2D array of orientation histograms
1058 @param d width of hist
1059 @param n bins per histogram
1060 @param feat feature into which to store descriptor
1061 */
1062 void hist_to_descr( double*** hist, int d, int n, struct feature* feat )
1063 {
1064   int int_val, i, r, c, o, k = 0;
1065
1066   for( r = 0; r < d; r++ )
1067     for( c = 0; c < d; c++ )
```

```
1068        for( o = 0; o < n; o++ )
1069          feat->descr[k++] = hist[r][c][o];
1070
1071    feat->d = k;
1072    normalize_descr( feat );
1073    for( i = 0; i < k; i++ )
1074      if( feat->descr[i] > SIFT_DESCR_MAG_THR )
1075        feat->descr[i] = SIFT_DESCR_MAG_THR;
1076    normalize_descr( feat );
1077
1078    /* convert floating-point descriptor to integer valued descriptor */
1079    for( i = 0; i < k; i++ )
1080    {
1081      int_val = static_cast <int> (SIFT_INT_DESCR_FCTR * feat->descr[i]);
1082      feat->descr[i] = MIN( 255, int_val );
1083    }
1084  }
1085  /*
1086  Normalizes a feature's descriptor vector to unitl length
1087  @param feat feature
1088  */
1089  void normalize_descr( struct feature* feat )
1090  {
1091    double cur, len_inv, len_sq = 0.0;
1092    int i, d = feat->d;
1093
1094    for( i = 0; i < d; i++ )
1095    {
1096      cur = feat->descr[i];
1097      len_sq += cur*cur;
1098    }
1099    len_inv = 1.0 / sqrt( len_sq );
1100    for( i = 0; i < d; i++ )
1101      feat->descr[i] *= len_inv;
1102  }
1103  /*
1104  Compares features for a decreasing-scale ordering.  Intended for use with
1105  CvSeqSort
1106  @param feat1 first feature
1107  @param feat2 second feature
1108  @param param unused
1109  @return Returns 1 if feat1's scale is greater than feat2's, -1 if vice versa,
1110  and 0 if their scales are equal
1111  */
1112  int feature_cmp( void* feat1, void* feat2, void* param )
1113  {
1114    struct feature* f1 = (struct feature*) feat1;
1115    struct feature* f2 = (struct feature*) feat2;
```

```
1116
1117   if( f1->scl < f2->scl )
1118     return 1;
1119   if( f1->scl > f2->scl )
1120     return -1;
1121   return 0;
1122 }
1123 /*
1124 De-allocates memory held by a descriptor histogram
1125 @param hist pointer to a 2D array of orientation histograms
1126 @param d width of hist
1127 */
1128 void release_descr_hist( double**** hist, int d )
1129 {
1130   int i, j;
1131
1132   for( i = 0; i < d; i++)
1133   {
1134     for( j = 0; j < d; j++ )
1135       free( (*hist)[i][j] );
1136     free( (*hist)[i] );
1137   }
1138   free( *hist );
1139   *hist = NULL;
1140 }
1141 /*
1142 De-allocates memory held by a scale space pyramid
1143
1144 @param pyr scale space pyramid
1145 @param octvs number of octaves of scale space
1146 @param n number of images per octave
1147 */
1148 void release_pyr( IplImage**** pyr, int octvs, int n )
1149 {
1150   int i, j;
1151   for( i = 0; i < octvs; i++ )
1152   {
1153     for( j = 0; j < n; j++ )
1154       cvReleaseImage( &(*pyr)[i][j] );
1155     free( (*pyr)[i] );
1156   }
1157   free( *pyr );
1158   *pyr = NULL;
1159 }
```

B.5. Quellcode Utils-Klasse

Utils.h

```
1  /**@file Miscellaneous utility functions. */
2  #ifndef UTILS_H
3  #define UTILS_H
4  #include "cxcore.h"
5  #include <stdio.h>
6  /* absolute value */
7  #ifndef ABS
8  #define ABS(x)((x < 0 )? -x : x )
9  #endif
10 /****************** Inline Functions ***************************/
11 /**
12 A function to get a pixel value from an 8-bit unsigned image.
13
14 @param img an image
15 @param r row
16 @param c column
17 @return Returns the value of the pixel at (\a r, \a c) in \a img
18 */
19 static __inline int pixval8( IplImage* img, int r, int c )
20 {
21   return (int)( ( (uchar*)(img->imageData + img->widthStep*r) )[c] );
22 }
23 /**
24 A function to set a pixel value in an 8-bit unsigned image.
25
26 @param img an image
27 @param r row
28 @param c column
29 @param val pixel value
30 */
31 static __inline void setpix8( IplImage* img, int r, int c, uchar val)
32 {
33   ( (uchar*)(img->imageData + img->widthStep*r) )[c] = val;
34 }
35 /**
36 A function to get a pixel value from a 32-bit floating-point image.
37
38 @param img an image
39 @param r row
40 @param c column
41 @return Returns the value of the pixel at (\a r, \a c) in \a img
42 */
43 static __inline float pixval32f( IplImage* img, int r, int c )
44 {
```

```
45   return ( (float*)(img->imageData + img->widthStep*r) )[c];
46 }
47 /**
48 A function to set a pixel value in a 32-bit floating-point image.
49 @param img an image
50 @param r row
51 @param c column
52 @param val pixel value
53 */
54 static __inline void setpix32f( IplImage* img, int r, int c, float val )
55 {
56   ( (float*)(img->imageData + img->widthStep*r) )[c] = val;
57 }
58
59 /**
60 A function to get a pixel value from a 64-bit floating-point image.
61
62 @param img an image
63 @param r row
64 @param c column
65 @return Returns the value of the pixel at (\a r, \a c) in \a img
66 */
67 static __inline double pixval64f( IplImage* img, int r, int c )
68 {
69   return (double)( ( (double*)(img->imageData + img->widthStep*r) )[c] );
70 }
71
72 /**
73 A function to set a pixel value in a 64-bit floating-point image.
74 @param img an image
75 @param r row
76 @param c column
77 @param val pixel value
78 */
79 static __inline void setpix64f( IplImage* img, int r, int c, double val )
80 {
81   ( (double*)(img->imageData + img->widthStep*r) )[c] = val;
82 }
83 /****************************** Function Prototypes ******************************/
84 /**
85 Prints an error message and aborts the program.  The error message is
86 of the form "Error: ...", where the ... is specified by the \a format
87 argument
88 @param format an error message format string (as with \c printf(3)).
89 */
90 extern void fatal_error( char* format, ... );
91 /**
92 Replaces a file's extension, which is assumed to be everything after the
```

```
93  last dot ('.') character.
94  @param file the name of a file
95  @param extn a new extension for \a file; should not include a dot (i.e.
96  \c "jpg", not \c ".jpg") unless the new file extension should contain
97  two dots.
98  @return Returns a new string formed as described above.  If \a file does
99  not have an extension, this function simply adds one.
100 */
101 extern char* replace_extension( const char* file, const char* extn );
102 /**
103 Doubles the size of an array with error checking
104 @param array pointer to an array whose size is to be doubled
105 @param n number of elements allocated for \a array
106 @param size size in bytes of elements in \a array
107
108 @return Returns the new number of elements allocated for \a array.  If no
109   memory is available, returns 0 and frees array.
110 */
111 extern int array_double( void** array, int n, int size );
112 /**
113 Calculates the squared distance between two points.
114 @param p1 a point
115 @param p2 another point
116 */
117 extern double dist_sq_2D( CvPoint2D64f p1, CvPoint2D64f p2 );
118 /**
119 Draws an x on an image.
120 @param img an image
121 @param pt the center point of the x
122 @param r the x's radius
123 @param w the x's line weight
124 @param color the color of the x
125 */
126 extern void draw_x( IplImage* img, CvPoint pt, int r, int w, CvScalar color );
127 /**
128 Combines two images by scacking one on top of the other
129 @param img1 top image
130 @param img2 bottom image
131 @return Returns the image resulting from stacking \a img1 on top if \a img2
132 */
133 extern IplImage* stack_imgs( IplImage* img1, IplImage* img2 );
134 #endif
```

Utils.c

```
#include "utils.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
/************************** Function Definitions ***************************/
/*
Prints an error message and aborts the program.  The error message is
of the form "Error: ...", where the ... is specified by the \a format
argument
@param format an error message format string (as with \c printf(3)).
*/
void fatal_error(char* format, ...)
{
    va_list ap;

    fprintf( stderr, "Error: ");

    va_start( ap, format );
    vfprintf( stderr, format, ap );
    va_end( ap );
    fprintf( stderr, "\n" );
    abort();
}
/*
Replaces a file's extension, which is assumed to be everything after the
last dot ('.') character.
@param file the name of a file
@param extn a new extension for \a file; should not include a dot (i.e.
\c "jpg", not \c ".jpg") unless the new file extension should contain
two dots.
@return Returns a new string formed as described above.  If \a file does
not have an extension, this function simply adds one.
*/
char* replace_extension( const char* file, const char* extn )
{
 char* new_file, * lastdot;

 new_file =(char*)calloc(strlen(file)+strlen(extn)+2,sizeof(char));
 strcpy( new_file, file );
 lastdot = strrchr(new_file, '.' );
 if( lastdot )
     *(lastdot + 1) = '\0';
 else
  strcat( new_file, "." );
```

```
48     strcat( new_file, extn );
49
50   return new_file;
51 }
52 /*
53 Doubles the size of an array with error checking
54 @param array pointer to an array whose size is to be doubled
55 @param n number of elements allocated for \a array
56 @param size size in bytes of elements in \a array
57 @return Returns the new number of elements allocated for \a array.  If no
58 memory is available, returns 0 and frees array.
59 */
60 int array_double( void** array, int n, int size )
61 {
62   void* tmp;
63
64   tmp = realloc( *array, 2 * n * size );
65   if( ! tmp )
66   {
67     fprintf( stderr, "Warning: unable to allocate memory in array_double(),"
68       " %s line %d\n", __FILE__, __LINE__ );
69      if( *array )
70         free( *array );
71       *array = NULL;
72       return 0;
73     }
74    *array = tmp;
75   return n*2;
76 }
77 /*
78 Calculates the squared distance between two points.
79 @param p1 a point
80 @param p2 another point
81 */
82 double dist_sq_2D( CvPoint2D64f p1, CvPoint2D64f p2 )
83 {
84   double x_diff = p1.x - p2.x;
85   double y_diff = p1.y - p2.y;
86   return x_diff * x_diff + y_diff * y_diff;
87 }
88 /*
89 Draws an x on an image.
90 @param img an image
91 @param pt the center point of the x
92 @param r the x's radius
93 @param w the x's line weight
94 @param color the color of the x
95 */
```

```
96 void draw_x( IplImage* img, CvPoint pt, int r, int w, CvScalar color )
97 {
98  cvLine(img, pt, cvPoint( pt.x + r, pt.y + r), color, w, 8, 0 );
99  cvLine( img, pt, cvPoint( pt.x - r, pt.y + r), color, w, 8, 0 );
100  cvLine( img, pt, cvPoint( pt.x + r, pt.y - r), color, w, 8, 0 );
101  cvLine( img, pt, cvPoint( pt.x - r, pt.y - r), color, w, 8, 0 );
102 }
103 /*
104 Combines two images by scacking one on top of the other
105 @param img1 top image
106 @param img2 bottom image
107 @return Returns the image resulting from stacking \a img1 on top if \a img2
108 */
109 extern IplImage* stack_imgs( IplImage* img1, IplImage* img2 )
110 {
111  IplImage* stacked = cvCreateImage( cvSize( MAX(img1->width, img2->width),
112        img1->height + img2->height ), IPL_DEPTH_8U, 3 );
113  cvZero( stacked );
114  cvSetImageROI( stacked, cvRect( 0, 0, img1->width, img1->height ) );
115  cvAdd( img1, stacked, stacked, NULL );
116  cvSetImageROI( stacked, cvRect(0, img1->height, img2->width, img2->height) );
117  cvAdd( img2, stacked, stacked, NULL );
118  cvResetImageROI( stacked );
119  return stacked; }
```

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 13. November 2008	
Ort, Datum	Unterschrift