Ausarbeitung
Masterprojekt Bildverarbeitung Teil1

Scale Invariant Feature Transform

Hauke Martens, Matr.-Nr.: 6222858
Jören Carstens, Matr.-Nr.: 6242743
M.Sc. Informatik

Eingereicht am: 14. Sep. 2014

Betreuer: Dr. Benjamin Seppke

Diese Arbeit wurde in der Arbeitsgruppe Szenenanalyse und Visualisierung des Fachbereichs Informatik an der Universität Hamburg erstellt

Inhaltsverzeichnis

1 Einleitung und Motivation

1.1 Motivation

Ein großes Problem in der Bildverarbeitung stellt das Erkennen von Objekten und die 3D-Rekonstruktion von Objekten auf Bildern dar. Objekterkennung wird beispielsweise in komplizierten Fertigungprozessen genutzt, in welchen die Position und die Lage von Objekten auf einem Fließband mit einer großen Genauigkeit bekannt sein muss, um dort zum Beispiel andere Objekte anzubringen.
Komplizierter ist es bei der Identifizierung und Verfolgung von sich bewegenden Objekten, wenn beispielsweise ein Roboter eine bestimmte Aufgabe erledigen soll. In diesem Fall muss die Objeterkennung auch unabhängig von der Entfernung, und somit der Größe des Objekts in der Abbildung, und der Orientierung des Objekts funktionieren.
Weitere Anwendungsbeispiele in denen Objekterkennung genutzt wird sind Sicherheitskameras an Flughäfen, um verbotene Gegenstände, wie zum Beispiel Waffen, zu erkennen, Analyseprogramme in der Medizin, welche automatisch Tumore auf Computertomografiebildern erkennen, oder Programme in der Automobilbranche, um das selbstfahrende Auto zu entwickeln.

1.2 Objekterkennungsalgorithmen

Eine häufige Herangehensweise an das Problem der Objekterkennung beginnt mit dem Sammeln von Daten anhand von Bildern, sei es in Form von Fotos oder einzelnen Frames eines Videos. Daraufhin erfolgt die Datenverarbeitung, welche zunächst aus der Filterung der Informationen besteht um nur noch die prägnanten Merkmale (features) betrachten zu müssen. Nachdem die Daten in ein festgelegtes Format gebracht wurden, können diese mit anderen Informationen verglichen werden und es kann entschieden werden, ob eine große Ähnlichkeit vorliegt, oder nicht.
Ein Beispielalgorithmus zum Auffinden von Schlüsselpunkten, in diesem Fall Ecken, ist der Harris-Algorithmus. Dieser findet mithilfe der vertikalen und der horizontalen Gradientenstärken und den Eigenwerten Ecken in einem Bild. Durch die Nutzung der Eigenwerte ist dieser Algorihmus rotationsinvariant. Wenn jedoch gleiche Objekte mit unterschiedlicher Größe verglichen werden sollen, kann dies zu Schwierigkeiten führen, weil einige hochskalierte vermeintliche Ecken eventuell nur noch Rundungen sind. Diese Problem der nicht vorhandenen Skalierungsinvarianz wird in Abb. 1 [4] dargestellt. Das SIFT-Verfahren (kurz für Scale Invariant Feature Transform) nach Lowe[3] kann mit eben diesem Problem umgehen.

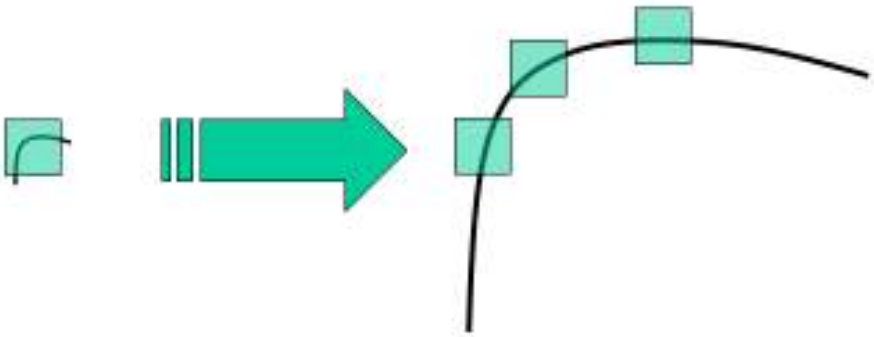

Abbildung 1: Beispiel für eine Ecke, welche vergrößert keine ist [4]

1.3 Implementation des SIFT-Verfahrens im Rahmen des Bildverarbeitungsprojektes

Im Rahmen des Bildverarbeitungsprojektes im Masterstudiengang Informatik der Universität Hamburg in der Betreuung von Dr. benjmain Seppke war es die Aufgabe des SIFT-Verfahren nach Lowe[3] bis einschließlich der Definition der lokalen Deskriptoren zu implementieren. Weitere Rahmenbedingungen waren, dass das Programm in der Programmiersprache C++ geschrieben wird und das die Vigra-Bildverarbeitungs-Bibliothek[2] von Ullrich Köthe verwendet wird.

2 Beschreibung des SIFT-Verfahrens anhand des Ausgangspapers: Distinctive Image Features from Scale-Invariant Keypoints von David G. Lowe

Im Folgenden wird das SIFT-Verfahren anhand des Papers 'Distinctive Image Features from Scale-Invariant Keypoints' von David G. Lowe[3] beschrieben. Der SIFT-Algorithmus beschreibt ein Verfahren, welches aus einem Bild vergleichbare rotations- und skalierungsinvariante Schlüsselpunkte findet um diese in einem weiteren Verfahren auf anderen Bildern wiederzufinden um somit Objekte mit bekannten Merkmalen auf anderen Bildern zu finden.

2.1 Das Auffinden von Extrempunkten

Um Schlüsselpunkte zu finden, die unabhängig von der Skalierung der Objekts sind, müssen diese auf unterschiedlichen Skalierungsebenen gefunden werden. Hierfür wird in der Theorie der sogenannte Laplacian of Gaussian (LoG) genutzt, welcher abhängig von dem Skalierungsparameter σ sogenannte „blobs“ findet. Als Ergebnis erhalten wir eine Liste von Punkten in verschiedenen Skalierungsebenen (x, y, σ).
Da LoG jedoch sehr kostenspielig ist, wird stattdessen eine Annäherung des LoGs, das Difference of Gaussian-Verfahren, benutzt. Um dieses Anzuwenden müssen zuerst weitere kleiner Schritte abgearbeitet werden. Da die Skalierungsinvarianz gegeben sein soll, wird das Bild je in kleiner werdenden Skalierungen durchsucht. Für je eine Skalierung gibt es eine bestimmte Anzahl x_{lay} Ebenen, an welchen abhängig vom Skalierungsparameter σ der gaußsche Weichzeichner angewandt wurden. Je x_{lay} solcher gleich großen Ebenen gehören zu einer Oktave. Bei jedem Oktavenübergang wird nun das Bild herunterskaliert, sodass es nur noch ein Viertel der vorherigen Größe besitzt.
Aus jeder Oktave können $x_{lay} - 1$ Difference of Gaussian-Bilder errechnet werden, indem die Differenz der Grauwerte zweier benachbarter Ebenen gebildet wird.

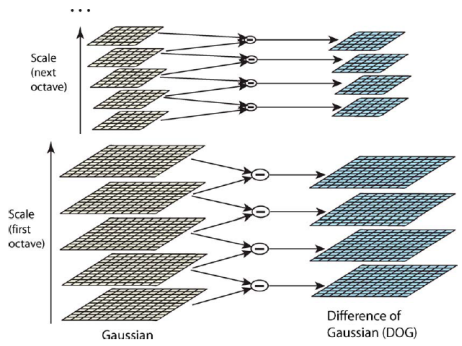

Abbildung 2: Aufteilung in Oktaven und deren Ebenen und Berechnung der DoGs [3]

Auf den mittleren beiden DoGs einer Oktave kann nun „im Raum“ nach Extrema gesucht werden. Hierfür wird für jeden Punkt auf diesen DoGs verglichen, ob er den kleinsten/größten Grauwert innerhalb seiner acht Nachbarpixel auf der selben Ebene und je innerhalb seiner neun Nachbarpixel auf der vorherigen und der nächsten Ebene besitzt (Abb. 3 [3]).

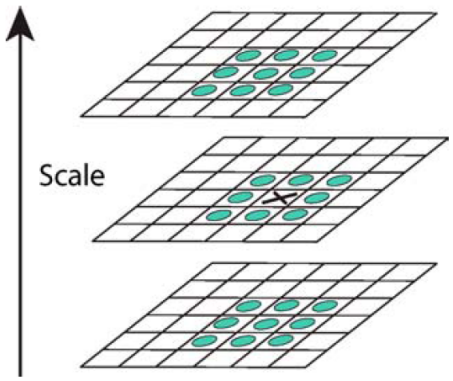

Abbildung 3: Ein auf Extremum zu überprüfender Punkt und seine Nachbarpixel [3]

2.2 Die präzise Ortsbestimmung für die Extrema

Da die Extrempunkte für die herunterskalierten DoGs „nur“ pixelgenau lokalisiert wurden, würden die Pixelkoordinaten bei einer Hochskalierung verfälscht werden. Um dies zu vermeiden wird für jedes gefundene Extremum mit Hilfe der Taylorreihen-Expansion ein Offset berechnet, welcher es einem ermöglicht die Position des Extremums subpixelgenau zu bestimmen. Lowe schlägt vor, dass alle Extrema mit einem Offset größer als 0.5 Pixel verschoben werden, da das Extremum näher an einem anderen Pixel liegt, als

an dem Errechneten.
Nachdem nun die Ortsbestimmung der Punkte auf subpixelebene erfolgt ist, müssen viele der gefundenen Punkte herausgefiltert werden. Zunächst werden die Punkte entfernt, deren Kontrast zu niedrig ist, d.h. deren Funktionswert $D(\hat{x})$ den von Lowe vorgeschlagenen Grenzwert von 3% nicht überschreiten.
Daraufhin müssen alle gefundenen Extrema herausgefiltert werden, die auf einer Kante liegen, da diese in verschiedenen Skalierungen an unterschiedlichen Orten gefunden werden können. Hierfür wird mithilfe der Determinante und der „Trace“ die Summe der „Eigenvalues“ berechnet. Ist diese größer als $(r+1)^2/r$, wobei Lowe für den Grenzwert $r = 10$ vorschlägt, so liegt das Extremum auf einer Kante.

2.3 Bestimmung der Orientierung eines Schlüsselpunkts

Um die Rotationsinvarianz zu gewährleisten muss nun zu jedem Schlüsselpunkt eine Orientierung bestimmt werden. Hierfür wird für jedes Pixel in einem bestimmten Bereich um den Schlüsselpunkt die Gradientenrichtung in Form eines Winkels bestimmt (Abb. 4). Diese wird gewichtet mit der Gauß-Funktion abhängig von der Entfernung zu dem Schlüsselpunkt in ein 36-Bin Histogramm (1 Bin pro 10°) eingetragen. Nachdem dies für jedes Pixel innerhalb des vorbestimmten Bereichs getan wurde, kann aus dem Histogramm das Maximum und somit die Orientierung des Schlüsselpunkts abgelesen werden. Nach Lowe kann ebenfalls noch eine weitere Orientierung vorhanden sein, wenn ein anderes lokales Maximum mindestens 80% des absoluten Maximums entspricht (Abb. 5).

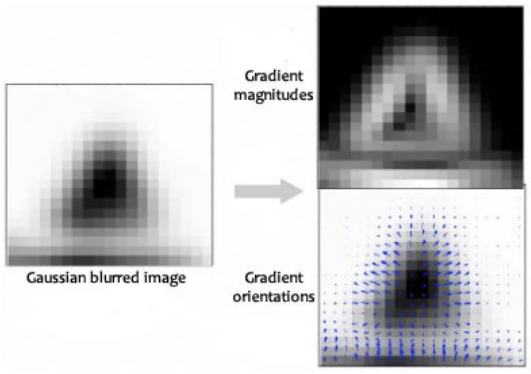

Abbildung 4: Die einzelnen Gradienten innerhalb eines bestimmten Bereichs um einen Schlüsselpunkt [5]

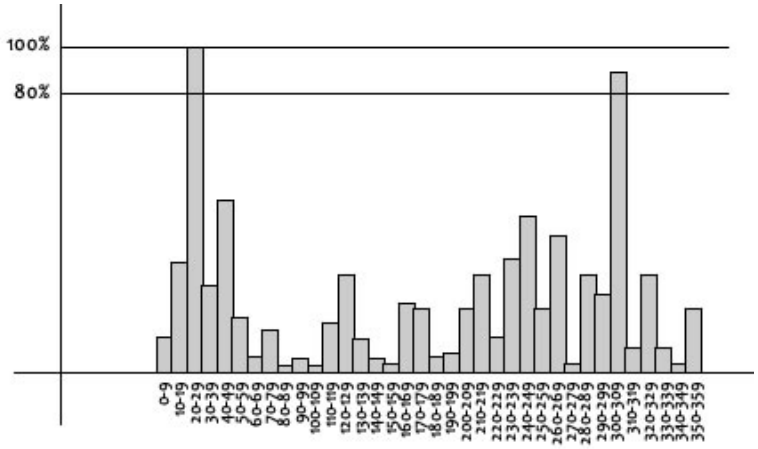

Abbildung 5: Histogramm eines Schlüsselpunkts [5]

2.4 Der Schlüsselpunkt Descriptor

Es wurde dafür gesorgt, dass die Schlüsselpunkte invariant gegenüber Rotation und Skalierung sind. Im letzten Schritt wird jedem Schlüsselpunkt ein Fingerabdruck gegeben um ihn später mit anderen Schlüsselpunkten vergleichen zu können.
Um dies zu erreichen werden innerhalb eines Gitters entlang der Orientierung des Schlüsselpunkts weitere 16 Histogramme berechnet, welche sich jeweils aus je 16 Pixeln errechnen lassen. Jedes dieser 16 Histogramme hat 8 Bins. Diese Daten werden als „Local Image Descritor“ in Form eines „Feature Vektors“ gespeichert. Jeder „Local Image Descritor“ enthält also 128 Daten.

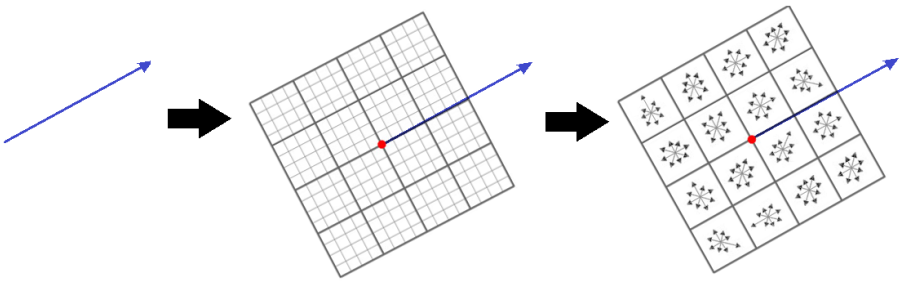

Abbildung 6: Histogramme für den Fingerabdruck

2.5 Ergebnisse

Lowe zeigt in seinem Paper, dass die SIFT-Schlüsselpunkte durch ihre Eindeutigkeit sehr nützlich sind, wodurch für diese auch in großen Schlüsselpunktdatenbanken ein Partner gefunden werden kann. Außerdem zeigt Lowe, dass die Schlüsselpunkte sowohl rotations- und skalierungsinvariant, als auch robust gegenüber Verzerrung sind. Außerdem zeigt er anhand von Beispielen, dass die Punkte sehr gut zur Objekterkennung dienen (Abb. 7 [3]).

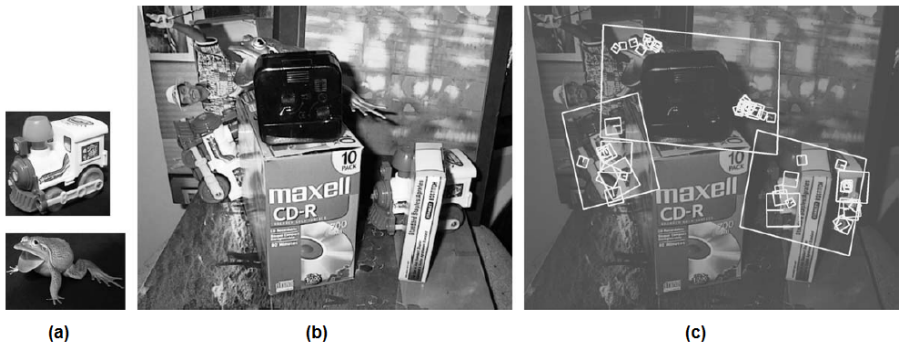

Abbildung 7: (a) Trainings Bilder (b) Das Testbild mit Überdeckung (c) Das Ergebniss der Objekterkennung [3]

3 Implementation und Unterschiede zum Verfahren im Paper von Lowe

Dieser Abschnitt beschreibt, wie wir anhand des Papers von Lowe [3] das SIFT-Verfahren implementiert haben. Außerdem wird hier auf Unterschiede eingegangen, die bei der Implementation gemacht wurden und warum diese gemacht wurden.

3.1 Das Auffinden von Extrempunkten

In diesem Abschnitt wird das Auffinden von Extrempunkten in detaillierten Schritten beschrieben.

3.1.1 Erstellen des „Scalespace“

Nach dem Einlesen des Bildes definieren wir zunächst die Variablen $max\_octave = 2$, welche die Anzahl an Oktaven bestimmt, $scale\_space\_size = 5$, welche die Anzahl der Ebenen innerhalb einer Oktave bestimmt, $max\_sigma = 1.6$, unser maximales σ (wie vorgeschlagen bei Lowe) und die Konstante $k = 2^{\left(\frac{1}{scale\_space\_size-3}\right)}$. Außerdem erstellen wir ein zweidimensionales Array $levels[max\_octave][scale\_space\_size]$, in welchem wir die einzelnen Bilddaten halten.
Daraufhin werden die einzelnen Oktaven inklusive ihrer einzelnen Ebenen erstellt. Dabei wird jedes Bild mit einer wachsenden Standardabweichung, die sich aus der „Tiefe“ im Skalenraum und max_sigma ergibt, Gauss-geglättet. Nach jeder letzten Ebene einer Oktave wird die Größe verringert, sodass die Bildgröße einer Oktave einem Viertel der Größe der vorherigen Oktave entspricht. Die einzelnen Ebenen exportieren wir zu Vergleichszwecken im PNG-Bildformat (Abb. 8).

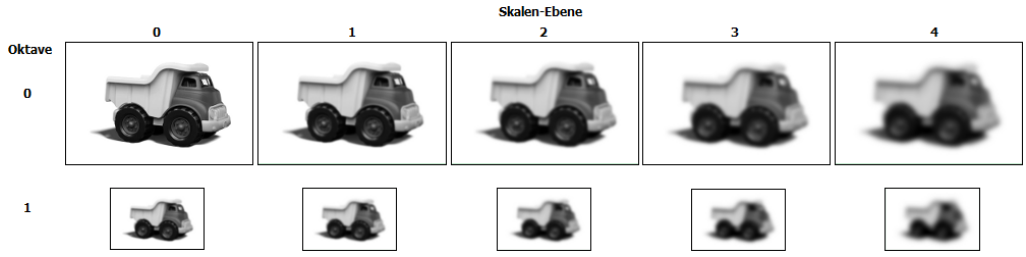

Abbildung 8: Die exportierten Bilder der Oktaven und deren Ebenen

3.1.2 Erstellung der „Differences of Gaussian“

Als nächstes werden aus den Bilddaten im Array *levels* die Differences of Gaussian (DoG) erstellt. Hierfür wurde ebenfalls ein Array $dogs[max\_octave][scale\_space\_size-1]$ erstellt, in dem die DoG gehalten werden. Dieses wird mit Bildern gefüllt, deren Grauwerte die Differenzen der Gauss-geglätteten Bildern aus *levels* sind. Da zur Erzeugung eines DoG jeweils zwei Bilder benötigt werden, resultiert dieser Schritt in $scale\_space\_size - 1$ DoG pro Oktave, in unserem Fall vier. Auch die Differences of Gaussian exportieren wir zu Vergleichszwecken als Bilder, wobei die Grauwerte auf den Bereich $[0, 255]$ skaliert sind, um diese besser erkennbar zu machen (Abb. 9).

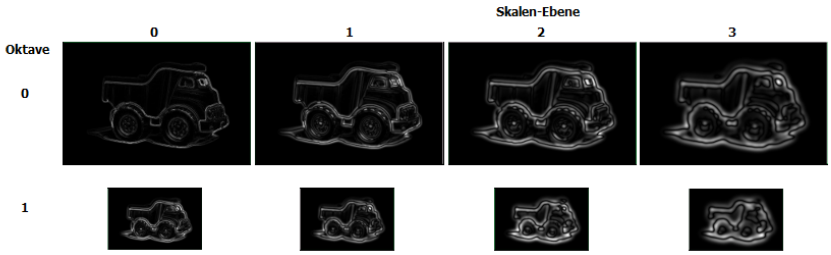

Abbildung 9: Die exportierten Bilder der DoGs

3.1.3 Auffinden von Extrempunkten

Für jedes DoG, welches in seiner Oktave einen Vorgänger und einen Nachfolger im Skalenraum hat, wird für jedes Pixel innerhalb der 26er-Nachbarschaft geprüft, ob es sich bei diesem um ein lokales Minimum, bzw. Maximum handelt. Die 26er-Nachbarschaft beschreibt dabei die acht umliegenden Pixel auf der Ebene des zu prüfenden Pixels selbst, sowie die neun nächsten Pixel auf der Ebene darüber und darunter (Abb. 3).
Alle Pixel die in diesem Schritt als Extremum gefunden wurden sind potentielle Kandidaten für Features und werden im weiteren Programmablauf behandelt.

3.2 Die präzise Ortsbestimmung der Extrema

Für alle gefundenen Kandidaten wird in diesem Schritt die Position subpixelgenau bestimmt.

Dies geschieht durch das Approximieren einer dreidimensionalen quadratischen Funktion über eine Taylorreihenentwicklung nach Brown [1]. Zu diesem Zweck werden die

partiellen Ableitungen über die Werte benachbarter Pixel angenähert. Auch die Hesse-Matrix wird für diesen Schritt benötigt. Das entstehende lineare Gleichungssystem wird mittels der Funktion „linearSolve“ aus Vigra gelöst und so ein Offset, um den der ursprünglich gefundene Punkt vorschoben werden muss, gefunden. Hat dieser Vektor einen Betrag größer 0.5 in wenigstens eine Richtung, liegt der Punkt näher an einem anderen “Ursprung„ und muss diesem entsprechend zugeordnet werden.

3.2.1 Qualitatives Filtern

Die im vorherigen Schritt gebildeten partiellen Ableitungen und die Hesse-Matrix werden in diesem Schritt erneut verwendet, um Punkte zu Filtern, die den qualitativen Ansprüchen nicht genügen.

Zum Einen sollen nur Punkte betrachtet werden, deren Kontrast größer 3% ist. In unserer Umsetzung jedoch werden nur Punkte aussortiert, deren Kontrast kleiner 0.1% ist. Dies beruht darauf, dass durch einen uns unbekannten Fehler ansonsten (durchschnittlich) $90\% - 95\%$ der gefundenen Punkte verworfen werden.

Des Weiteren werden auch sogenannte „Edge-responses“ eliminiert. Dies soll dafür sorgen, dass Punkte die auf einer Kante liegen nicht weiter betrachtet werden, da sich die genaue Position auf einer Kante schlecht feststellen lässt und dementsprechend anfällig, auch für kleine Änderungen, ist. Zu diesem Zweck wird die Hauptkrümmung mittels der Hesse-Matrix und der Spur am entsprechenden Punkt berechnet. Wir verwenden hier den selben Grenzwert ($r = 10$) wie Lowe.

3.3 Orientierung

Alle verbliebenen Punkte sind Features, die nun weiter verarbeitet und schließlich in den Deskriptor aufgenommen werden. Jedem Feature wird eine Orientierung zugeordnet und alle weiteren Operationen ausschließlich unter Betrachtung dieser ausgeführt. Dadurch ergibt sich eine Resistenz gegenüber einer Rotationen des Bildes.

Für alle Features, die für die folgenden Berechnungen weit genug entfernt vom Bildrand sind, werden in einer 16x16 Umgebung die Orientierungen und die Beträge der Gradienten der Pixel gebildet und aufgeteilt. Je nach (lokaler) Orientierung werden die Beträge in ein Histogramm mit 36 „Bins“ eingetragen. Die Werte sind dabei Gewichtet durch einen Gauss mit der Standardabweichung 1.5, in Abhängigkeit der Entfernung

zum eigentlich „Ursprung“ des Features. Daraus ergibt sich auch die 16x16 Umgebung, da bei einer Entfernung von 8 Pixeln der Faktor zur Gewichtung bereits $1.77 * 10^{-7}$ beträgt und weiter entfernte Pixel zu vernachlässigen sind.

Im nächsten Schritt wird der „größte Bin“ gesucht und über ihn und seine beiden Nachbarn eine Parabel gebildet, wobei die x-Koordinate deren Ursprungs ($*10$) die Orientierung des Features darstellt.

3.4 Erzeugung des Deskriptors

Der letzte Schritt besteht darin, den eigentlichen Deskriptor zu erzeugen.

Zu diesem Zweck werden die 4x4 Nachbarschaften der Größe 4x4 Pixel um jedes Feature betrachtet, ausgerichtet nach der im vorherigen Schritt errechneten Orientierung. Für jedes (innere) 4x4 Feld wird wiederum ein Histogramm erzeugt. Diese haben 8 „Bins“, in welche die Beträge der Gradienten, sortiert nach ihrer Orientierung, Gauss-gewichtet eingetragen werden. Um Rundungsfehler zu vermeiden werden die Nachbarn nicht auf Pixelgenauigkeit errechnet, sonder unter Hilfe eines „SplineImageViews“ aus Vigra die benötigten Werte für die partiellen Ableitungen direkt aus den „rotierten Koordinaten“ approximiert. Durch die Interpolation ist es möglich, Grauwerte an Positionen zu erhalten, die nicht genau auf einem Bildpunkt liegen. Diese 4x4x8 Werte stellen den Deskriptor eines Features dar und werden schlussendlich zeilenweise in eine Textdatei exportiert.

3.5 Erzeugung der Grafiken

Wie in vorherigen Kapitel bereits erwähnt, werden während des Programmablaufs an mehreren Stellen Bilder erzeugt, die unterschiedlich Charakteristiken gefundener Punkte oder Features visualisieren sollen. Diese Bilder erzeugen wir mittels Gnuplot [6] und während der Laufzeit dynamisch erzeugten Skripten. Die dynamische Erzeugung nimmt uns beim Wechsel des Bildes die manuelle Anpassung z.B. der Bildgröße, oder das Ändern des Dateinames ab. Zu zeichnende Elemente, wie Punkte, Linien, Kreise oder Quadrate, werden in eine Textdatei exportiert und von Gnuplot eingelesen. Gnuplot selbst wird über eine .bat Datei direkt aus dem Programm heraus aufgerufen.
Alle Bilder, die nicht „erweitert“ werden, werden direkt aus Programm über Vigra exportiert.

4 Ergebnisse und Evaluierung

Dieses Kapitel befasst sich mit den Ergebnissen der verschiedenen Arbeitsschritte unserer SIFT-Implementation und deren Einschätzung der Güte.

4.1 Das Finden und Filtern von Extrempunkten

In Abb.10(a) ist sehr gut dargestellt, dass die meisten noch ungefilterten Extrema an erwünschten Koordinaten gefunden werden. Das Originalbild des Spielzeug LKWs (Abb. 8) ist immer noch zu erkennen.
Abb.10(b) zeigt alle Extrema nach dem ersten Filterschritt, bei welchem alle Punkte entfernt werden, deren Subpixelgenauigkeits-Offset größer als 0.5 Pixel ist. Wie erwartet, werden bei diesem Schritt nicht viele Punkte entfernt. Bei der Filterung der sogenannten „low-contrast-points“, also die Punkte deren Kontrast nicht mehr als 3% aufweist, werden jedoch, wie bereits in Kapitel 3.2.1 beschrieben, ca. $90\% - 95\%$ aller Punkte entfernt. Diese Tatsache ist in Abb.10(c) dargestellt. In Abb.10(d) wurden dann wiederrum alle Punkte entfernt, welche auf einer Kante lagen.
Die Tabelle 1 zeigt die genauen Zahlen der gefundenen Punkte insgesamt nach den verschiedenen Filterschritten, sowie auf den einzelnen Ebenen, auf denen sie gefunden wurden. Aufällig ist, wie bereits erwähnt, dass allein auf der ersten Ebene der nullten Oktave nach dem Kontrast-Filterschritt von 1043 Extrema nur noch zwei übrig sind. Aus diesem Grund haben wir den Grenzwert für den Kontrast auf 0.1% gesetzt, statt der vorgeschlagenen 3%.
Genau wie in Abb.10(a) und (b) hat man in Abb.10(a) und (b) die gleiche Anzahl Extrema. Bei der Filterung der „low-contrast-points“, wurde der Grenzwert von 3% auf 0.1% verringert. Dadurch bleiben deutlich mehr Punkte erhalten, wie im Vergleich von Abb.10(c) und Abb.11(c) zu sehen ist. In Abb.11(d) wurden dann wiederrum alle Punkte entfernt, welche auf einer Kante lagen.
Ebenso wie die Tabelle 1 zeigt die Tabelle 2 die Anzahl der Extrema auf den verschiedenen Ebenen. Hier ist allerdings zu erkennen, dass durch Verringerung des Grenzwerts von 3% auf 0.1% fast die zehnfache Anzahl an Schlüsselpunkten übrig bleibt.

Filterschritt	Oktave 0 Ebene 1	Oktave 0 Ebene 2	Oktave 1 Ebene 1	Oktave 1 Ebene 2	Gesamtanzahl Punkte
alle Extrema:	1043	781	234	18	2240
Offset $< 0,5$:	643	618	190	154	1605
Kontrast $> 3\%$:	2	5	5	3	15
nicht-Kante:	2	4	5	3	14

Tabelle 1: Die einzelnen Zahlen der Extrema nach verschiedenen Filterungsschritten

Filterschritt	Oktave 0 Ebene 1	Oktave 0 Ebene 2	Oktave 1 Ebene 1	Oktave 1 Ebene 2	Gesamtanzahl Punkte
alle Extrema:	1043	781	234	18	2240
Offset $< 0,5$:	643	618	190	154	1605
Kontrast $> 0.1\%$:	44	48	37	16	145
nicht-Kante:	39	41	33	14	127

Tabelle 2: Die einzelnen Zahlen der Extrema nach verschiedenen Filterungsschritten mit dem Kontrast-Grenzwert von 0.1%

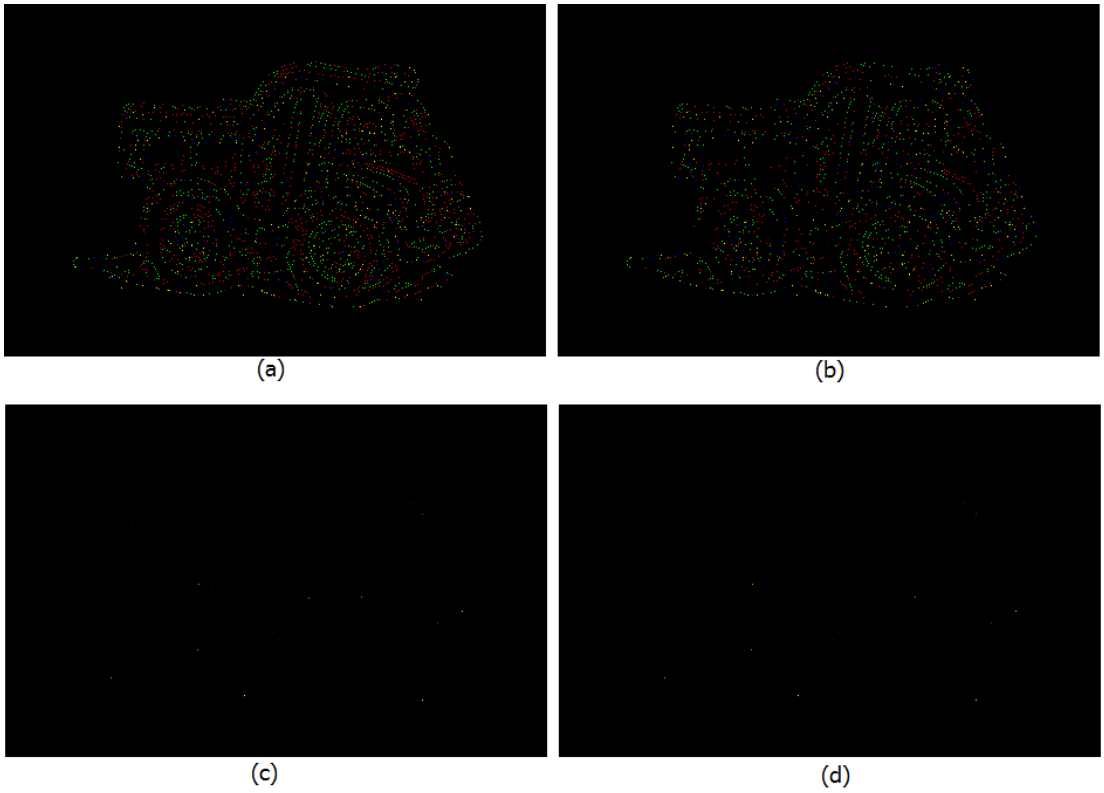

Abbildung 10: (a) alle gefundenen Extrema (b) nach dem Filtern aller Punkte mit einem $Offset > 0.5$ (c) nach dem Filtern aller „low-contrast-Points“($< 3\%$) (d) nach dem Filtern der auf Kanten liegenden Punkte; rot = Oktave 0 & Ebene 1, grün = Oktave 0 & Ebene 2, blau = Oktave 1 & Ebene 1, gelb = Oktave 1 & Ebene 2

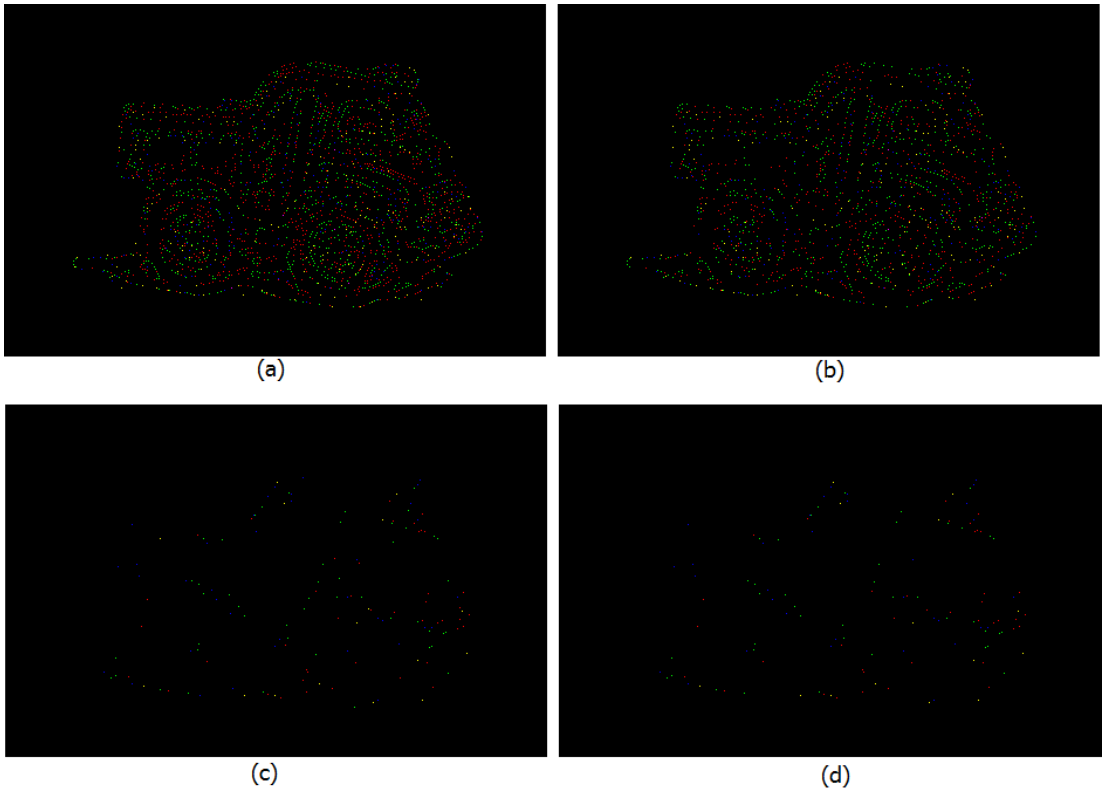

Abbildung 11: (a) alle gefundenen Extrema (b) nach dem Filtern aller Punkte mit einem $Offset > 0.5$ (c) nach dem Filtern aller „low-contrast-Points“($< 0.1\%$) (d) nach dem Filtern der auf Kanten liegenden Punkte; rot = Oktave 0 & Ebene 1, grün = Oktave 0 & Ebene 2, blau = Oktave 1 & Ebene 1, gelb = Oktave 1 & Ebene 2

Abb. 12 zeigt neben dem Originalbild 12(a), den gefundenen Schlüsselpunkten nach der Filterung 12(b) auch ein Bild mit erwarteten, bzw. nicht erwarteten Punkten. Grün steht in der Abb. 12(c) für Punkte, die wir eigentlich erwartet hatten, orange für den Bereich, in welchem wir keine Punkte erwartet hatten, da diese durch den Kontrastfilter herausgefiltert werden sollten. Mit einem 3% Grenzwert, wären diese wahrscheinlich nicht mehr in dem Bild. Die blauen Formen zeigen den Bereich in dem wir erwartet hätten, dass die Punkte dort durch den Filter ausgeschlossen werden, der Punkte auf Kanten erkennt.

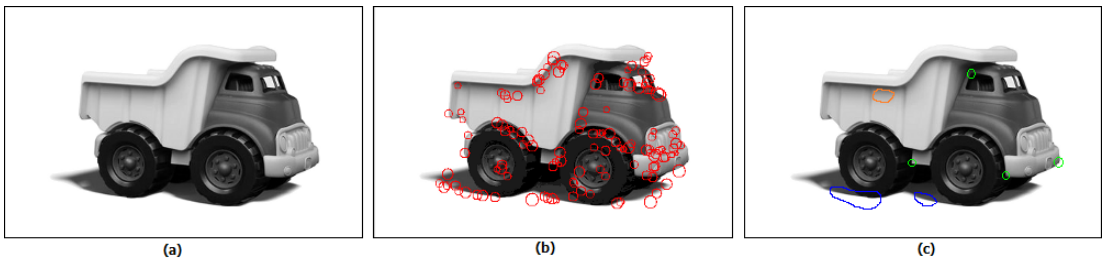

Abbildung 12: (a) Das Originalbild (b) Die gefundenen Schlüsselpunkte (c) Erwartungen: grün = fehlende, erwartete Schlüsselpunkte; orange = erwartete Filterung durch Kontrast-Filter; blau = erwartete Filterung da Kantenpunkte

4.2 Rotations-Invarianz

In Abb. 13 ist zu erkennen, dass die Schlüsselpunkte trotz der Drehungen um -7°, bzw. um 45° an sehr ähnlichen Stellen im Bild gefunden werden. Es wird außerdem deutlich, dass die Punkte auf Kanten vorwiegend an horizontalen, bzw. vertikalen Kanten entfernt werden und weniger an diagonalen Kanten, was ebenfalls erwünscht gewesen wäre.
Die Gradientenrichtung, angezeigt durch die Pfeile in Abb. 13(a)1,(b)1 und (c)1 entprechen in etwa der Richtung, die wir erwartet hatten. Auch die jeweilige Anzahl der gefundenen Schlüsselpunkte ((a) = 127, (b) = 135, (c) = 143) unterscheidet sich nicht allzu stark von den anderen.

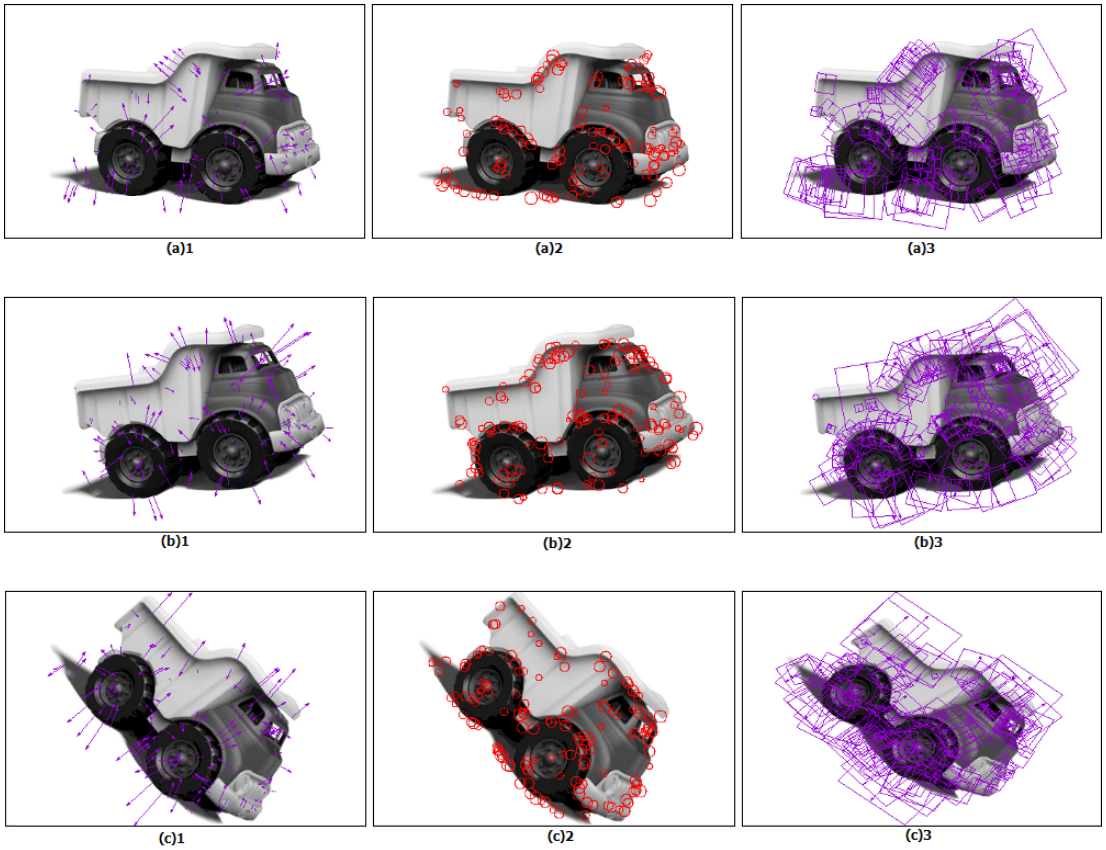

Abbildung 13: Der Spielzeug LKW (a) in seiner Ursprungsposition (b) um -7° gedreht (c) um 45° gedreht

4.3 Skalierungs-Invarianz

Auch nach dem Herunterskalieren des Spielzeug-LKWs befinden sich die Schlüssenpunkte an annähernd der selben Position, wie im Originalbild. Außerdem entsprechen die Gradientenrichtungen denen im Originalbild. Erwarteterweise werden weniger Schlüsselpunkte gefunden, je kleiner das Bild skaliert wird. In Abb. 14(a) sind es 127 Punkte, in (b) 74 und in (c) sind es 60 Schlüsselpunkte.

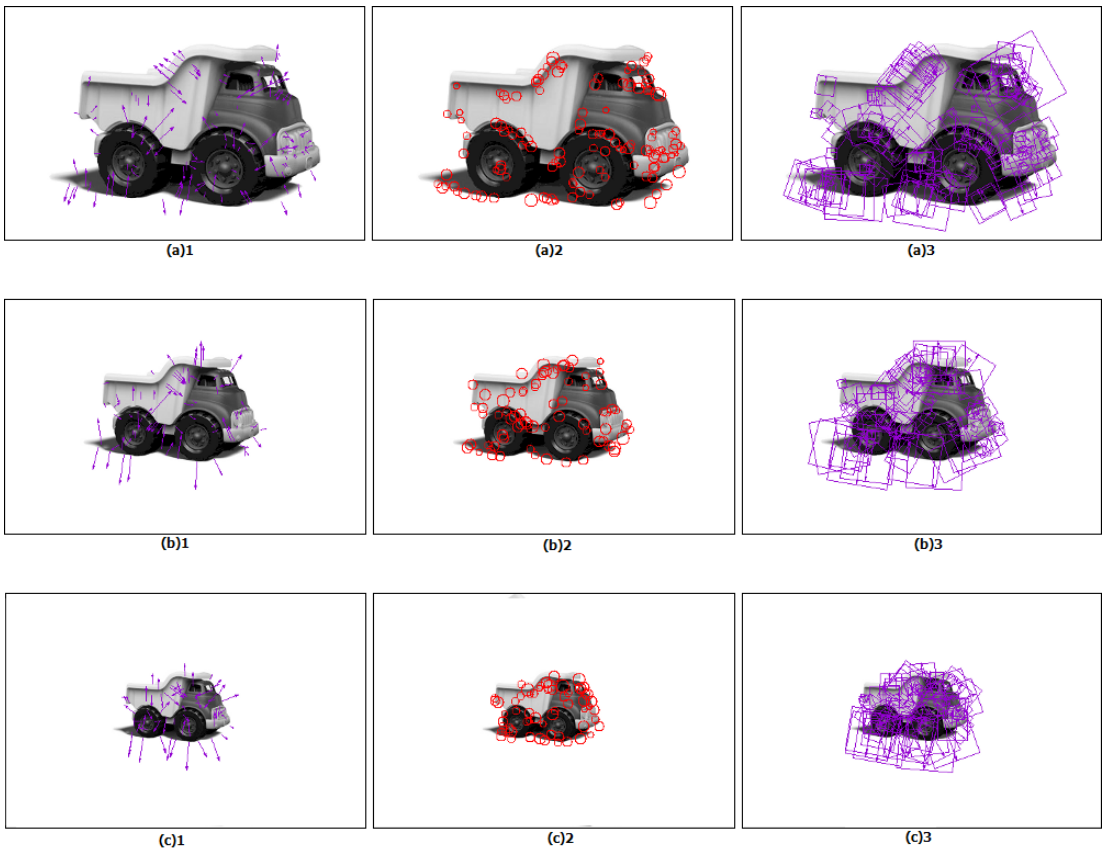

Abbildung 14: Der Spielzeug LKW (a) in seiner Ursprungsgröße (b) mit 2/3 bzw. (c) 1/3 seiner Ursprungsgröße

5 Fazit

Im Rahmen des ersten Teils des Masterprojektes Bildverarbeitung haben wir das SIFT-Verfahren nach Lowe [3] bis zum Export der Feature-Deskriptoren implementiert. Wir haben ein interessantes Verfahren zum Thema Objekterkennung kennen gelernt und konzeptuell verstanden.

Obwohl wir Ergebnisse erhalten, die nach subjektiver Einschätzung akzeptabel sind, verbleiben einige Unklarheiten, die wahrscheinlich auf Fehlern im unserer Umsetzung beruhen. Zum Einen führt eine Filterung der Features bezüglich des Kontrastes mit dem von Lowe vorgeschlagenen Parameter bei uns zur Eliminierung eines Großteils gefundener Punkte und zum Anderen werden bei uns insgesamt weniger Punkte gefunden, als wir, begründet auf der Arbeit Lowes vermuteten. Des Weiteren ist unsere Umsetzung insofern eingeschränkt, als dass sie nur Grauwertbilder verarbeiten kann.

Die Entwicklung des Programms hat insgesamt mehr Zeit in Anspruch genommen als anfangs gedacht. Dies beruht einerseits darauf, dass, trotz teilweise sehr ausführlicher Veranschaulichungen, Teilaspekte des Verfahren aus der Arbeit Lowes für uns schwierig nach zu vollziehen waren, andererseits darauf, dass die Entwicklung in C++ für uns neu war und daher besonders anfangs sehr zeitintensiv.

Literatur

[1] Matthew Brown und David G Lowe. 2002. Invariant Features from Interest Point Groups. In *BMVC*, number s 1.

[2] Ullrich Köthe. Generic Programming for Computer Vision - The VIGRA Computer Vision Library Version 1.10.0. `http://ukoethe.github.io/vigra/`, [Abgerufen: September 2015].

[3] Lowe, David G. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision 60(2)*, S. 91–110.

[4] Alexander Mordvintsev. Introduction to SIFT (Scale-Invariant Feature Transform). `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html`, [Abgerufen: September 2015].

[5] Utkarsh Sinha. SIFT: Keypoint orientations. `http://www.aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/`, [Abgerufen: September 2015].

[6] Thomas Williams, Colin Kelley, Russell Lang, Dave Kotz, John Campbell, Gershon Elber, Alexander Woo, und viele Andere. Gnuplot. `http://www.gnuplot.info/`, [Abgerufen: September 2015].

Abbildungsverzeichnis