Featurebasierte Mustererkennung für markerlose Augmented Reality-Anwendungen

Henning Tjaden

2. März 2012

Inhaltsverzeichnis

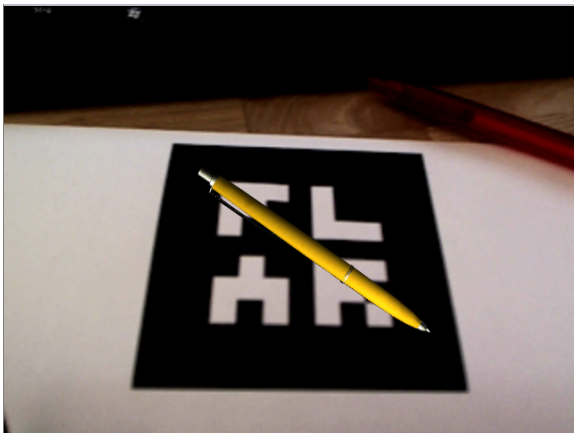
1 Einleitung

Diese Arbeit ist im Rahmen des Masterprojekts im Masterstudiengang Informatik an der Hochschule RheinMain entstanden. Begleitet und unterstützt wurde das Projekt durch Prof. Dr. Ulrich Schwanecke im Wintersemester 2011/2012.

Ziel war es, ein featurebasiertes Augmented Reality (AR) System zu entwickeln. Dabei soll die 3D-Pose eines beliebigen ebenen Musters relativ zur Kamera rein anhand der Homographie zu dessen Abbild im Videobild stabil berechnet werden, ohne klassische *AR-Marken* zu verwenden.

1.1 Motivation

Bei Augmented Reality (dt. „erweiterte Realität“) Anwendungen wird meist eine Aufnahme (Foto oder Video) der realen Welt durch virtuelle Inhalte, die in den realen Kontext eingebunden sind, ergänzt. Bei der populärsten Variante werden virtuelle 3D-Modelle perspektivisch korrekt auf einer realen Ebene im Videobild dargestellt, wie in Abbildung 1(a) zu sehen ist.



(a) Ein virtueller Stift auf einer AR-Marke.

(b) Eine beispielhafte AR-Marke.

Abbildung 1: *Ein Beispiel für eine klassische AR-Anwendung unter Verwendung des FLARToolKit* [1].

Bei den ersten Verfahren dieser Art [KB99] wird die Kamerapose relativ zu einer sogenannten *AR-Marke*, wie zum Beispiel in Abbildung 1(b), berechnet.

[1] http://www.libspark.org/wiki/saqoosha/FLARToolKit/en

Dazu müssen die Marken zunächst in den Bildern gefunden und erkannt werden (auf englisch auch als „Tracking“ bezeichnet). Aus der Kamerapose kann anschließend die Transformation für die virtuellen Objekte bestimmt werden, um diese auf der Marke anzuzeigen. Diese Verfahren haben allerdings den Nachteil, dass die Marken immer vollständig in den Bildern zu sehen sein müssen und nur sehr instabil bei Beleuchtungsveränderungen erkannt werden.

Neuere Verfahren auf diesem Gebiet kommen mittlerweile vollständig ohne Marken oder etwas Vergleichbares aus [KM07]. Dabei werden lokale Merkmalspunkte (sogenannte „Features“) in Videobildern berechnet und mit den Features aus den vorherigen Videobildern abgeglichen, woraus sich wiederum die Kamerapose bestimmen lässt. Daher benötigen diese Verfahren kein vorheriges Wissen über die gefilmte Szene (wie z. B. die Textur einer Marke). Dies führt dazu, dass diese Verfahren gegenüber Perspektiven- und Beleuchtungsänderungen deutlich flexibler und stabiler sind.

Das hier vorgestellte Verfahren kombiniert diese beiden Ansätze in der Form, dass beliebige ebene Muster (z. B. Postkarten, Verpackungen, Poster, etc.) die Rolle der klassischen *AR-Marke* einnehmen können (Abbildung 2). Das Verfahren wird daher auch als „markerlos“ bezeichnet.

(a) Eine virtuelle Trompete auf einer Postkarte.

(b) Die verwendete Postkarte in der Draufsicht.

Abbildung 2: *Ein Beispiel für das Verfahren, in dem eine virtuelle Trompete auf einer Postkarte dargestellt wird, obwohl die Postkarte stark reflektiert, teilweise verdeckt und nicht vollständig im Bild ist.*

Da der Erkennung aber weiterhin ein bekanntes Muster zu Grunde liegt, bleibt der direkte Bezug von virtuellem zu realem Objekt erhalten, was für einige Zwecke sehr wünschenswert ist, beispielsweise wenn eine Figur aus einem Animationsfilm auf dem entsprechenden DVD-Cover animiert wird.

Das Verfahren wird durch die Verwendung von Features allerdings stabiler und vielseitiger bei der Erkennung (dem Tracking).

1.2 Verfahren

Die grundlegende Idee des Verfahrens ist, die 3D-Pose (bestehnd aus Position und Rotation) eines ebenen Musters relativ zur einer zuvor kalibrierten Kamera rein anhand der Homographie zu dessen Abbild im Videobild zu bestimmen. Die Homographie selbst wird mit Hilfe von korrespondierenden Features aus Muster und Videobild berechnet. Dieser Vorgang ist in Abbildung 3 schematisch dargestellt.

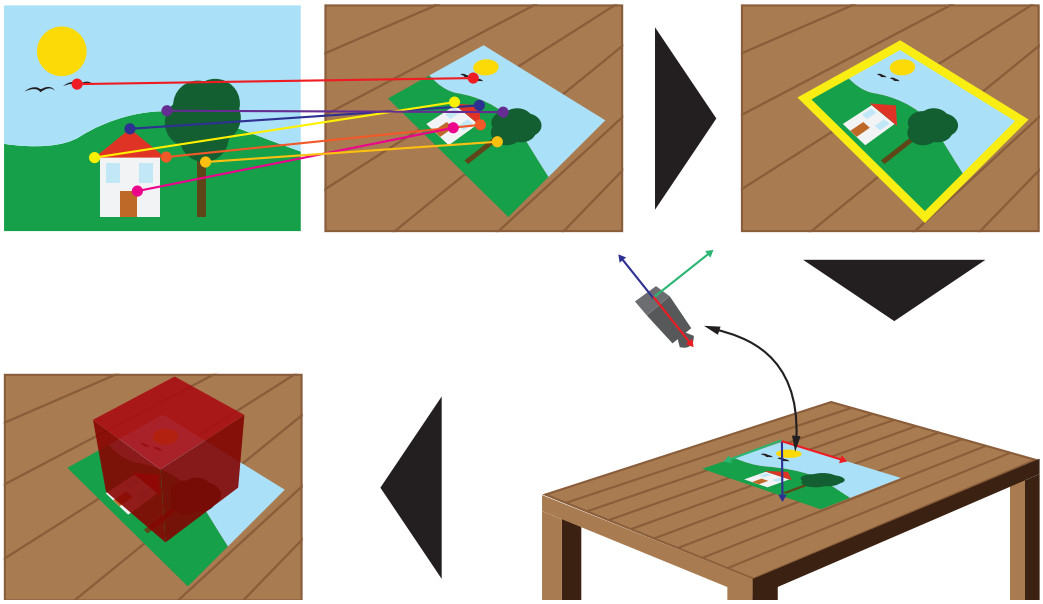


Abbildung 3: *Schematischer Ablauf des Verfahrens: 1. Feature-Korrespondenzen finden. 2. Homographie aus den Korrespondenzen bestimmen. 3. Relative Lage von Muster zu Kamera berechnen. 4. Virtuelle Objekte mit der berechneten Transformation auf dem Muster anzeigen.*

Da die Auswahl einer geeigneten Feature-Erkennung hierbei eine essentielle Rolle spielt, werden die derzeit populärsten Verfahren auf diesem Gebiet in Abschnitt 2 ausführlich vorgestellt. In Abschnitt 3 wird beschrieben, wie aus den Feature-Korrespondenzen die Homographie stabil bestimmt werden kann. Die Berechnung der Pose des Musters aus der Homographie sowie die Kalibrierung der Kamera werden in Abschnitt 4 erläutert.

Im Rahmen dieser Arbeit sind eine kleine AR-Anwendung, sowie eine C++ Klasse entstanden, die das Verfahren implementiert. In Abschnitt 5 wird abschließend erklärt, wie diese Anwendung bedient und die Klasse in eigene Anwendungen integriert werden kann.

2 Feature-Erkennung

Features (lokale Merkmale) in Bildern zu finden und in anderen Bildern (der selben Szene/Objekte) wiederzuerkennen ist ein weit verbreitetes Thema in der computergestützten Bildverarbeitung. In diesem Fall sollen Features in einem statischen, ebenen Muster und dessen perspektivisch verzerrten Abbild in einer Aufnahme gefunden und wiedererkannt werden, um daraus die Homographie zu berechnen. Dabei muss man davon ausgehen, dass das Muster aus sehr unterschiedlichen Perspektiven und bei unterschiedlichen Lichtverhältnissen gefilmt wird (Abbildung 4). Daher benötigt man Verfahren, die sehr robust gegenüber diesen Änderungen sind, um korrespondierende Features zuverlässig erkennen zu können.

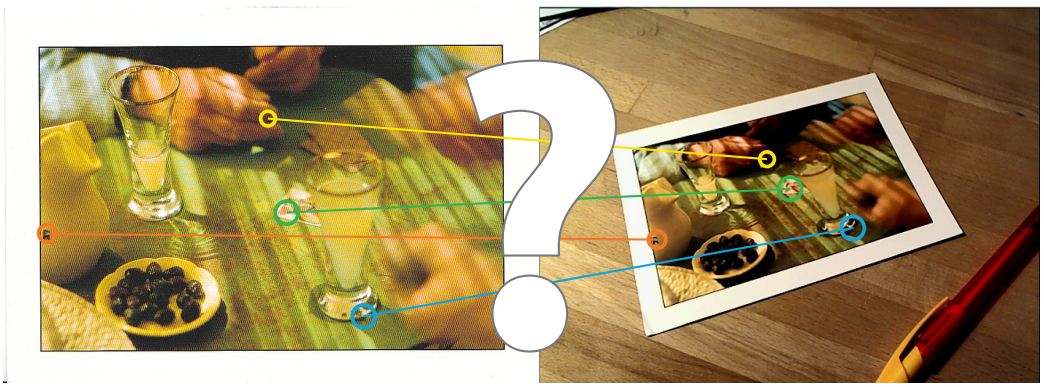


Abbildung 4: *Beispiel zur Problematik der Wiedererkennung von Bildmerkmalen. Hier dient eine Postkarte als Muster, das in der rechten Ansicht perspektivisch verzerrt im Kamerabild zu sehen ist. Dabei sind korrespondierende Punktepaare über farbige Linien verbunden.*

Um dieses Problem zu lösen, gibt es verschiedene Ansätze und Verfahren. Im Folgenden werden die drei Algorithmen SIFT [Low99], SURF [BTG06] und FAST [RD05] vorgestellt. Dabei ist zu beachten, dass die Extraktion von Features in Bildern immer in zwei Hauptschritte eingeteilt ist. Zuerst werden die Positionen der Features ermittelt und anschließend so beschrieben, dass sie möglichst eindeutig und unter großen Veränderungen (wie Skalierung, Rotation, Helligkeit) wiedererkannt werden können. Bei FAST entfällt der zweite Schritt, da hierbei ausschließlich das Auffinden der Features behandelt wird. Die drei hier präsentierten Algorithmen gehören aktuell zu den populärsten und stabilsten Verfahren auf dem Gebiet der Feature-Erkennung.

2.1 SIFT

Der SIFT (Scale Invariant Feature Transform) Algorithmus lässt sich in die folgenden groben Schritte eingeteilen:

1. **Erzeugung des Maßstabsraums.** Dabei wird das Bild auf verschiedene Maßstäbe skaliert, um die Features möglichst unabhängig von der Skalierung zu finden.

2. **Erzeugung von DoG-Bildern.** In LoG-Bildern (Laplacian of Gaussian) findet man leicht markante Punkte in Bildern. Da deren Erzeugung sehr rechenintensiv ist, wird in diesem Schritt eine Näherung mit Hilfe des Maßstabsraums berechnet.

3. **Bestimmung von Merkmalspunkten.** In diesem Schritt werden die Positionen der Features bestimmt. Dazu werden Maxima und Minima in den DoG-Bildern ermittelt und auf Subpixelebene berechnet. Außerdem werden ungeeignete Merkmalspunkte aussortiert.

4. **Orientierung der Merkmalspunkte bestimmen.** Um eine Beschreibung der Merkmalspunkte zu finden, die möglichst unabhängig von Rotationen ist, wird zu jedem Merkmalspunkt eine Orientierung berechnet.

5. **Beschreibung der SIFT-Features.** In diesem Schritt werden die gefundenen Merkmalspunkte durch einen 128-dimensionalen Vektor so beschrieben, dass sie möglichst stabil wiedererkannt werden können.

Diese werden in den folgenden Abschnitten im Detail erklärt.

Erzeugung des Maßstabsraums

Dieser Schritt dient zur Vorbereitung des Verfahrens. Im Hinblick auf die Skalierungsinvarianz wird das Originalbild zunächst auf verschiedene Größen skaliert (die sogenannten Oktaven). Der Algorithmus erzielt die besten Ergebnisse bei vier Oktaven (200%, 100%, 50% und 25%). Anschließend werden pro Oktave fünf Bilder erzeugt (Abbildung 5). Jedes dieser Bilder lässt immer weniger Details erkennen, ähnlich dem Betrachten der Szene aus immer größerer Distanz, d. h. in einem anderen Maßstab.

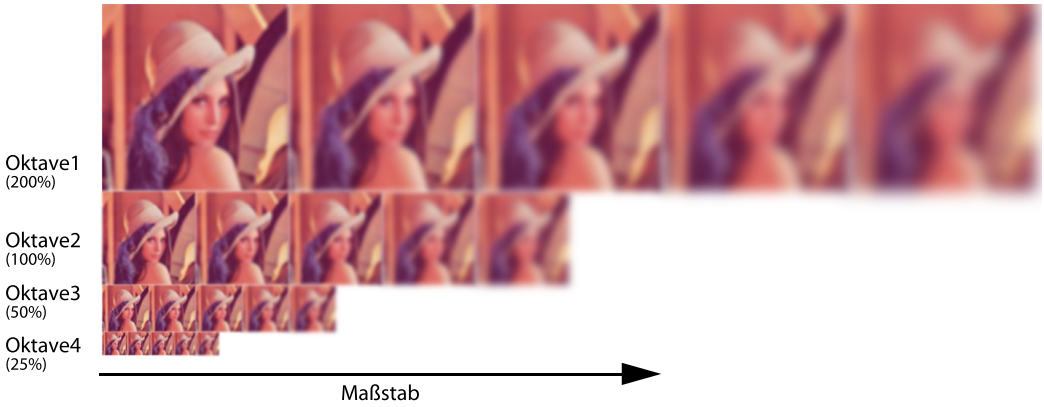


Abbildung 5: *Beispiel eines Maßstabsraums.*

Dazu wird das Ursprungsbild per Gaußfilter G mit ansteigender Stärke σ geglättet

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

um die verschiedenen Detailstufen zu erhalten. Da die Kantenlänge des Bildes von Oktave zu Oktave halbiert wird, muss auch σ halbiert werden, um die Filtergröße an die Bildgröße anzupassen. Angenommen die Filtergröße eines Bildes sei σ, dann berechnet sich die Filtergröße für das nächste, stärker geglättete Bild als $k \cdot \sigma$. Wobei k eine wählbare Konstante ist. Die folgende Tabelle zeigt mögliche Sigma-Werte für $k = \sqrt{2}$:

Oktave	Maßstab (σ)				
1	5,656854	8,000000	11,313708	16,000000	22,627417
2	2,828427	4,000000	5,656854	8,000000	11,313708
3	1,414214	2,000000	2,828427	4,000000	5,656854
4	0,707107	1,000000	1,414214	2,000000	2,828427

Dadurch entsteht der sogenannte Maßstabsraum. Darin können in den folgenden Schritten die Merkmalspunkte weitestgehend unabhängig von ihrer Skalierung, in der sie in den Bildern auftauchen, bestimmt werden. Außerdem fallen Bildstörungen wie Rauschen oder Artefakte in gröberen Maßstäben nicht mehr ins Gewicht.

Erzeugung von DoG-Bildern

Ecken und Kanten sind gut geeignet, um markante Punkte in Bildern zu finden. Eine effiziente Art, um Kanten aus Bildern zu extrahieren, ist das

LoG-Verfahren (Laplacian of Gaussian). Dabei wird ein Kantenbild entsprechend der 2. Ableitung des Bildes berechnet. Diese Operation ist allerdings sehr rechenintensiv. Mit Hilfe des Maßstabsraums können wesentlich schneller ähnliche Ergebnisse wie bei LoG-Bildern erzielt werden. Dazu berechnet man Differenzbilder aus zwei aufeinanderfolgenden Maßstabsbildern innerhalb einer Oktave (Abbildung 6).

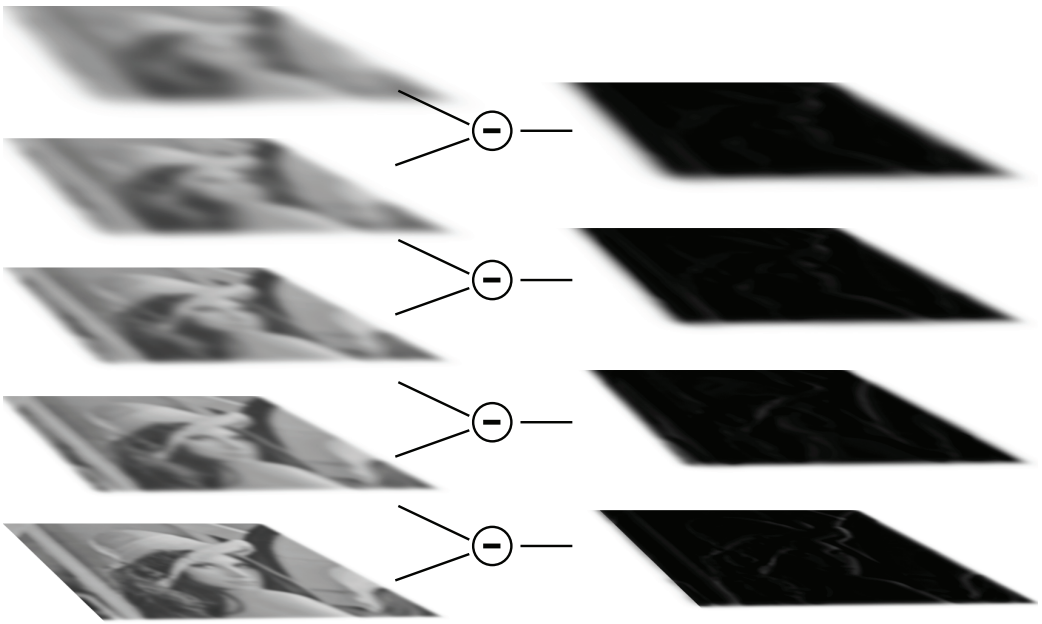


Abbildung 6: *Erzeugung der DoG-Bilder innerhalb einer Oktave.*

Die dabei entstandenen DoG-Bilder (Difference of Gaussian) sind eine für diesen Zweck ausreichende Näherung der LoG-Bilder.

Bestimmung von Merkmalspunkten

Die Differenzbilder aus dem vorherigen Schritt werden nun auf lokale Maxima und Minima untersucht. Hierfür wird eine Non-Maximum/Non-Minimum Unterdrückung durchgeführt. Dabei werden immer drei aufeinanderfolgende DoG-Bilder zusammen betrachtet. Für jedes Pixel aus dem mittleren DoG-Bild wird geprüft, ob es das Maximum oder Minimum seiner 26 Nachbarn ist (Abbildung 7). Ist dies der Fall, so wird das Pixel als Merkmalspunkt gespeichert. Dadurch entstehen aus den vier DoG-Bildern zwei neue Bilder pro Oktave, in denen nur noch die möglichen Merkmalspunkte übrigbleiben.

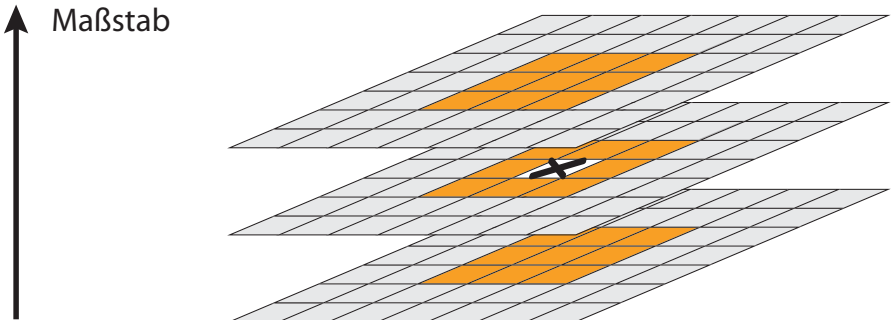


Abbildung 7: *Nachbarschaftsprüfung bei der Non-Maximum/Non-Minimum Unterdrückung. Das weiße, mit einem Kreuz markierte Pixel wird dabei mit seinen 8 orangenen Nachbarn aus dem eigenen und den jeweils 9 Nachbarn aus dem vorherigen und nachfolgenden DoG-Bild verglichen.*

Da die erkannten Merkmalspunkte (die Maxima und Minima der DoG-Bilder) in den meisten Fällen nicht direkt an der Position eines Pixels liegen, müssen die genauen Positionen nun anschließend, mit Hilfe einer Taylorentwicklung, auf Subpixelebene bestimmt werden. Dadurch wird die Bildfunktion um den Merkmalspunkt wie folgt definiert

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x.$$

Die genaue Position des Merkmalspunkts $\hat{x}$ befindet sich also an dem Extremwert (Nullstelle der ersten Ableitung) dieser Funktion. Der ermittelte Extremwert $D(\hat{x})$ der Taylorentwicklung lässt sich wiederum dafür nutzen, um Punkte mit zu geringer Intensität (unterhalb eines festen Schwellwerts) auszusortieren, da diese sich nicht ausreichend von ihrer Umgebung unterscheiden. Punkte auf Kanten sind für die Wiedererkennung ungeeignet und müssen daher ebenfalls ausgefiltert werden, so, dass nur noch Punkte auf Ecken erhalten bleiben. Dazu wird die 2×2 Hesse-Matrix

$$H(D) = \begin{pmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial xy} \\ \frac{\partial^2 D}{\partial yx} & \frac{\partial^2 D}{\partial y^2} \end{pmatrix}$$

im Merkmalspunkt berechnet. Ob der Punkt auf einer Kante oder einer Ecke liegt, wird dabei ähnlich wie bei dem Harris-Ecken-Detektor [HS88] über die Eigenwerte der Hesse-Matrix bestimmt.

Orientierung der Merkmalspunkte bestimmen

Die Orientierung eines Merkmalspunkts wird im Hinblick auf die Rotationsinavrianz der Erkennung ermittelt. So können alle anschließenden Berechnungen unter Berücksichtigung der Orientierung des Merkmals durchgeführt werden. Dazu wird eine Gradientenkarte einer Region um den Merkmalspunkt erstellt (Abbildung 8).

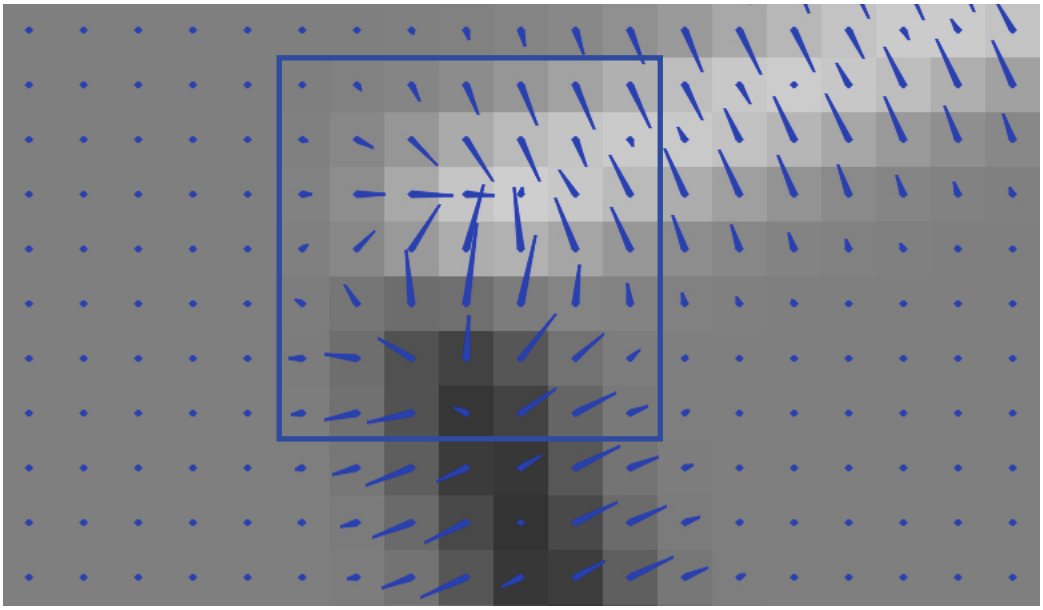


Abbildung 8: *Beispiel einer Gradientenkarte. Der blaue Rahmen umfasst die Region, aus der die Gradienten in die Berechnung der Orientierung mit einbezogen werden.*

Die berechneten Winkel der Gradienten werden mit einer Genauigkeit von 10° diskretisiert und mit der Gradientenlänge und der Glättung gewichtet in einem Winkelhistogramm aufgetragen. Die Größe der Gradientenkarte richtet sich nach der Glättung, also dem aktuellen Maßstab. Hier gilt: je stärker die Glättung, desto größer die Region. Der zugehörige Winkel des höchsten Wertes im Histogramm wird als Orientierung des Punktes gespeichert. Es kann vorkommen, dass in einer Region mehrere markante Orientierungen auftauchen. Dies ist der Fall, wenn mehrere Orientierungen ähnlich häufig vorkommen wie die das Maximum des Winkelhistogramms (oberhalb von 80% des höchsten Wertes). Bei mehreren Hauptorientierungen werden die Punkte erneut mit den jeweiligen Orientierungen gespeichert, um nicht mehrere Orientierungen mit demselben Punkt zu assoziieren.

Beschreibung der SIFT-Features

Zuletzt muss eine Beschreibung der lokalisierten Merkmalspunkte gefunden werden. Dazu wird zu jedem Punkt eine Art Fingerabdruck (der sogenannte Feature-Vektor) gespeichert, über den dieser möglichst eindeutig identifiziert werden können. Die grundlegende Idee besteht darin, die benachbarten Pixel der Merkmalspunkte mit in dessen Beschreibung einzubeziehen. Dazu wird wiederum eine Gradientenkarte in einer 16×16 Pixel großen Region um den Merkmalspunkt berechnet (Abbildung 9).

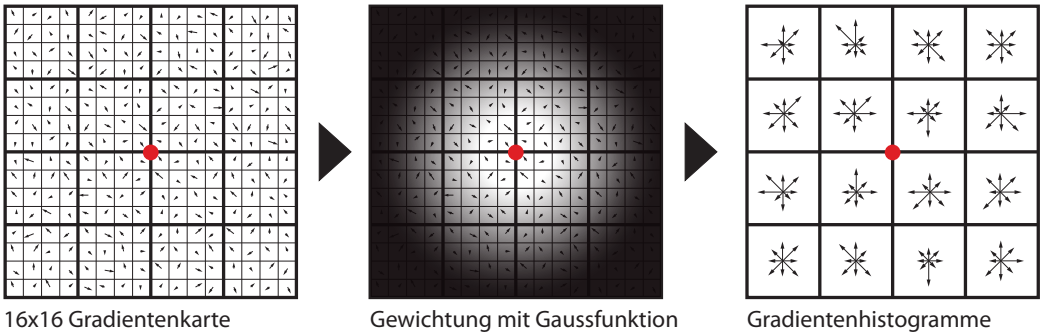


Abbildung 9: *Ablauf der Berechnung des Feature-Vektors. Im ersten Bild sieht man die Gradientenkarte, eingeteilt in* 4×4 *Unterregionen (Merkmalspunkt rot markiert). Im zweiten Bild ist die Gewichtung anhand der Gaußfunktion grafisch verdeutlicht (je dunkler, desto geringer das Gewicht). Das dritte Bild veranschaulicht die Gradientenhistogramme. Je länger ein Pfeil, desto größer der Wert im Histogramm.*

Anschließend wird die Region in 16 Unterregionen der Größe 4×4 eingeteilt. Für jede dieser Unterregionen wird ein Histogramm der Gradientenwinkel mit jeweils acht Einträgen (Winkelgenauigkeit von 45°) erzeugt. Im Gegensatz zum vorherigen Schritt spielt hierbei die Entfernung der Unterregionen zum Merkmalspunkt eine Rolle. Gradienten, die weiter entfernt sind, werden schwächer gewichtet in die Histogramme eingetragen, um ihren Einfluss zu verringern. Für diese Gewichtung wird eine Gaußfunktion, ausgehend von der Position des Merkmalspunkts, verwendet. So erhält man für alle 16 Unterregionen ein Histogramm mit je 8 Werten. Zusammengefasst ergibt sich daraus ein 128-dimensionaler Featurevektor, der den Merkmalspunkt eindeutig beschreibt. Abschließend wird der Vektor noch normiert, wodurch die Beschreibung beleuchtungsinvariant wird.

2.2 SURF

Der SURF (Speeded Up Robust Features) Algorithmus wurde zeitlich nach SIFT entwickelt und basiert in Teilen darauf. Allerdings wurde bei SURF speziell darauf geachtet, das Verfahren deutlich zu beschleunigen. Dazu wurden einige Berechnungen stark vereinfacht oder durch simplere ersetzt. Der grundsätzliche Ablauf ist auch hierbei in die zwei Hauptschritte Bestimmung und Beschreibung der Merkmalspunkte unterteilt.

Bestimmung von Merkmalspunkten

Als Vorbereitung wird zunächst ein Integralbild des Originalbildes berechnet. In einem Integralbild hat jedes Pixel den Wert der Summe der Intensitäten aller Pixel links oberhalb von sich aus dem Originalbild.

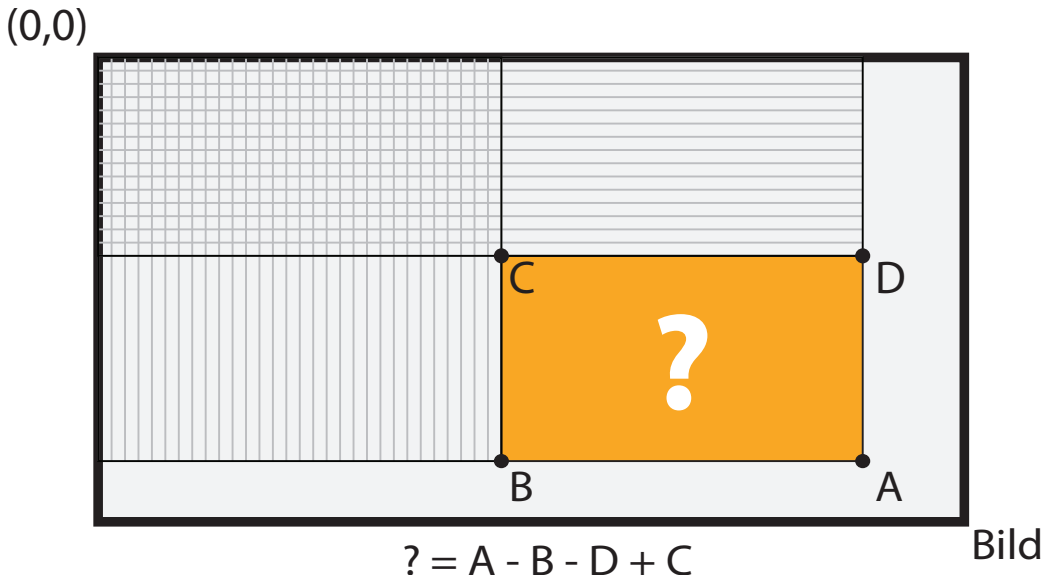


Abbildung 10: *Beispiel zu der Verwendung eines Integralbildes. Hier wird die Summe der Intensitäten des mit Orange markierten Rechtecks aus dem Originalbild berechnet. Die vier Werte A, B, C und D sind die Einträge an den entsprechenden Stellen im Integralbild. Sie gleichen den Intensitätssummen der Rechtecke links oberhalb von ihren Koordinaten (Kreuzschraffur, vertikale Schraffur, horizontale Schraffur und ohne Schraffur).*

Daher kann mit Hilfe eines Integralbildes die Summe der Intensitäten in einem beliebigen rechteckigen Ausschnitt des Originalbildes in konstanter Zeit

berechnet werden (Abbildung 10). Diese Eigenschaft wird sich im Folgenden genutzt, um Rechenzeit einzusparen. Wie bereits bei SIFT wird zur Bestimmung der Merkmalspunkte zunächst ein Kantenbild durch die 2. Ableitungen des Bildes erzeugt (LoG). Bei SIFT wurde dieser Schritt durch DoG-Bilder approximiert. Bei SURF wird dieser Schritt noch weiter vereinfacht, und anstelle des LoG-Operators werden Box-Filter verwendet, welche die 2. Ableitungen in x-, y- und yx-Richtung für jedes Pixel noch stärker approximieren.

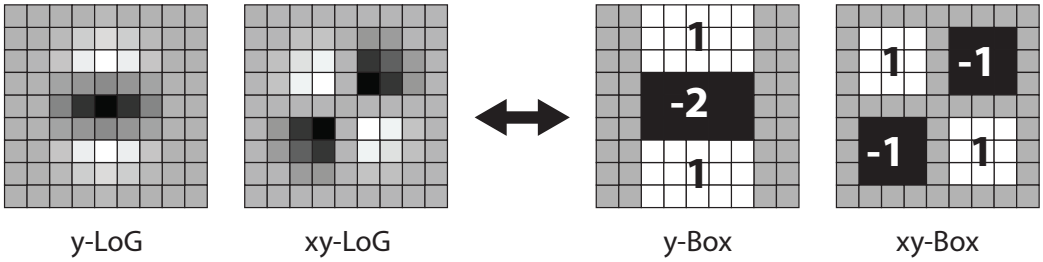


Abbildung 11: *Links sind die* 9×9 *Filtermasken des LoG-Filters in y- und xy-Richtung abgebildet. Rechts daneben sieht man die jeweiligen Filtermasken als Boxfilter. Die grauen Regionen werden mit 0 gewichtet.*

Wie Abbildung 11 zeigt, können die einheitlichen Teilflächen der Boxfiltermasken bei einer Faltung mit Hilfe des Integralbildes in konstanter Zeit berechnet werden. Dies hat weiterhin den Vorteil, dass die Filtergröße keine Auswirkung auf die Berechnungsgeschwindigkeit hat. Das Filter kann also beliebig skaliert werden. Auch SURF verwendet im Hinblick auf Skalierungsinvarianz einen Maßstabsraum.

Anders als bei SIFT muss für die Oktaven hier nicht das Bild auf verschiedene Größen skaliert werden, um dadurch die Filtergröße und somit den Rechenaufwand gering zu halten. Er wird allein über die Größe der Filtermaske erzeugt. Eine Filtermaske der Größe 9×9 Pixeln entspricht einem LoG-Filter mit einer Varianz von $\sigma = 1,2$. Da die Boxfilter diskretisiert sind und ein Zentrumspixel (und dadurch eine ungerade Kantenlänge) haben müssen, können sie nicht beliebig skaliert werden. Die möglichen Maskengrößen hängen zusätzlich von den Teilflächen des Filters ab. Dabei gilt, dass die kürzere Kante einer Teilfläche stets ein Drittel der Kantenlänge der Filtermaske betragen muss. Im Fall des y-Boxfilters gibt es 3 dieser Teilflächen. Damit diese auch weiterhin zu dem Zentrumspixel zentriert sind, müssen sie um mindestens 2 Pixel und die Filtermaske insgesamt um 6 Pixel vergrößert werden. Daher werden die Kantenlängen der Filter der ersten Oktave ausgehend von 9 Pixeln jeweils um 6 Pixel vergrößert (9px, 15px, 21px, 27px usw.). Für

jede weitere Oktave wird der Vergrößerungsfaktor jeweils verdoppelt. Daraus ergeben sich die Filtergrößen, die in Abbildung 12 zu sehen sind.

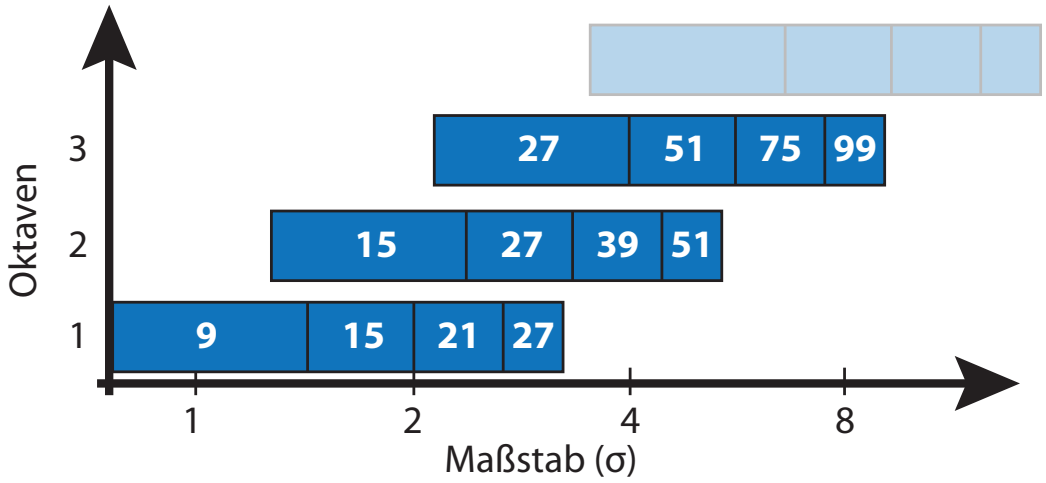


Abbildung 12: *Darstellung der unterschiedlichen Filtergrößen pro Oktave bei der Erstellung des Maßstabsraums für SURF.*

Bei der Bestimmung der Merkmalspunkte wird die 2×2 Hesse-Matrix mit den Werten der drei Boxfilter in x-, y- und xy-Richtung verwendet. Ob sich in einem Bild an einer Position möglicherweise ein Merkmalspunkt befindet, hängt davon ab, ob die Determinante der Hesse-Matrix an dieser Stelle ein lokales Maximum erreicht. Dies ist der Fall, wenn die Determinante positiv ist. Ist sie negativ, so handelt es sich um einen Sattelpunkt. Außerdem kann anhand der Spur der Matrix der Kontrast des Merkmalspunkts zu seiner direkten Umgebung abgelesen werden, der wiederum zum Aussortieren verwendet werden kann.

In den verbliebenen Punkten werden nun, wie bei SIFT, durch eine Non-Maximum/Non-Minimum-Unterdrückung mit den 26 Nachbarn innerhalb des Maßstabraums die eigentlichen Merkmalspunkte ermittelt. Ihre Position muss anschließend ebenfalls wieder auf Subpixelebene bestimmt werden.

Beschreibung der SURF-Features

Nachdem die Merkmalspunkte lokalisiert wurden, müssen sie auch bei SURF eindeutig beschrieben werden. Die Beschreibung ähnelt der der SIFT-Features, wurde jedoch in ihrer Komplexität weiter reduziert. Zunächst wird im Hinblick auf die Rotationsinvarianz die Orientierung der Merkmalspunkte bestimmt.

Abbildung 13: *Ablauf der Orientierungsbestimmung bei SURF. Links sind die, für die Transformation verwendeten, Haar-Wavelets abgebildet. In der Mitte sieht man die resultierenden Vektoren, gewichtet mit einer Gaußfunktion (Merkmalspunkt rot markiert). Rechts ist ein beispielhaftes Ergebnis der aufsummierten Vektoren in 60°-Intervallen dargestellt.*

Dazu wird als Erstes eine kreisförmige Region um den Merkmalspunkt mit Haar-Wavelets in x- und y-Richtung transformiert (Abbildung 13). Die Ergebnisse dieser Transformation werden im Folgenden mit d_x und d_y abgekürzt. Der Radius dieser Region beträgt $6s$, wobei s der Maßstab ist, bei dem der Merkmalspunkt lokalisiert wurde. Da dieser Schritt allgemein maßstabsabhängig ist, werden auch die Haar-Wavelets entsprechend skaliert. Ihre Kantenlänge beträgt $4s$. Daraus folgt, dass bei großen Maßstäben auch die Wavelets sehr groß werden. An dieser Stelle kann das Verfahren wieder durch die Verwendung des Integralbildes beschleunigt werden. Egal wie groß der Maßstab ist, kann die Wavelet-Transformation mit nur 6 Additionen und 4 Speicherzugriffen (jeweils 3 Additionen und 4 Speicherzugriffe pro Teilfläche des Filters) pro Richtung durchgeführt werden (Abbildung 10). Die Ergebnisse dieser Transformation werden, gewichtet mit einer Gaußfunktion ($\sigma = 2,5s$, Zentrum im Merkmalspunkt), als zweidimensionale Vektoren (d_x, d_y) aufgefasst. Um die Orientierung der Region und damit auch des Merkmalspunktes zu bestimmen, werden diese Vektoren in Intervalle von 60° eingeteilt und pro Intervall aufsummiert. Der längste der dabei entstandenen sechs Vektoren bestimmt schließlich die Orientierung des Merkmalspunktes.

Die eigentliche Beschreibung wird aus einer quadratischen Region (Kantenlänge $20s$) um die Merkmalspunkte berechnet. Diese Region ist anhand der zuvor ermittelten Orientierung gedreht (Abbildung 14).

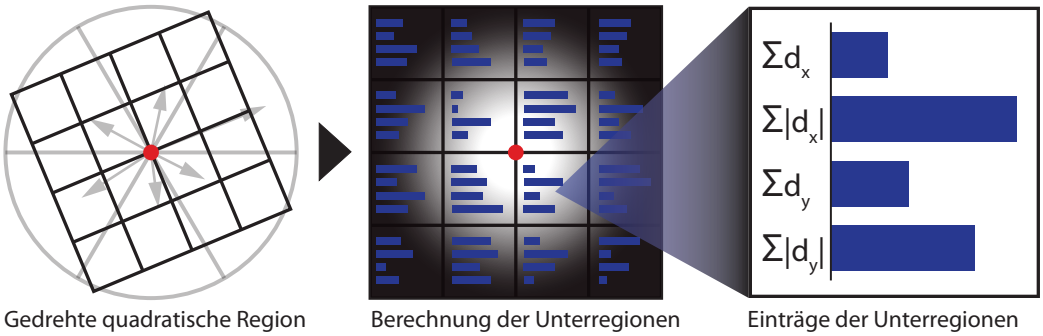


Abbildung 14: *Veranschaulichung der Beschreibung bei SURF. Links sieht man die orientierte quadratische Region um den Merkmalspunkt. In der Mitte sind die 16 gewichteten Unterregionen schematisch dargestellt. Rechts ist die Detailansicht mit den Einträgen einer dieser Regionen abgebildet.*

Diese Region wird wiederum mit Haar-Wavelets in gedrehter x- und y-Richtung gefiltert (Filtergröße $2s$). Sie werden mit Hilfe einer Gaußfunktion ($\sigma = 3,3s$, Zentrum im Merkmalspunkt) gewichtet und aufgeteilt in 16 quadratische Unterregionen aufsummiert. Diese Summen bilden die ersten Einträge des Feature-Vektors. Um zusätzlich die Polarität (die Stärke) der Intensitätsänderungen mit einzubeziehen, werden die Beträge $|d_x|$ und $|d_y|$ für jede Unterregion aufsummiert. So entsteht für jede der 16 Unterregion ein vierdimensionaler Vektor ($v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$). Zusammen bilden sie den 64-dimensionalen Feature-Vektor zu einem Merkmalspunkt.

2.3 FAST

Bei FAST (Features from Accelerated Segment Test) hendelt es sich um ein weiteres Verfahren, um Features in Bildern zu lokalisieren. Allerdings fehlt hierbei der Schritt der Beschreibung der Merkmalspunkte. Dieses Verfahren ist vom Rechenaufwand her das simpelste, gleichzeitig aber auch das instabilste der drei hier vorgestellten. Es unterscheidet sich von Ansatz her deutlich von SIFT und SURF, wie in den folgenden Abschnitten verdeutlicht wird.

Bestimmung von Merkmalspunkten

Auch bei FAST wird wieder bevorzugt an Ecken in den Bildern nach Features gesucht. Hierbei wird anhand einer kleinen Region um ein Pixel entschieden, ob diese so „aussieht“ wie eine Ecke oder nicht.

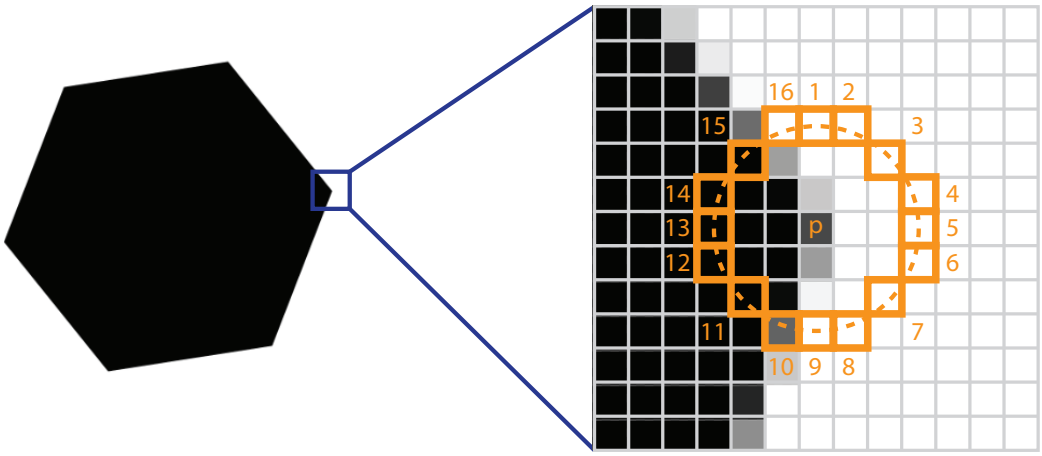


Abbildung 15: *Beispiel für einen Bresenham-Kreis mit Radius $r = 3$ um ein Pixel p. Die Pixel auf dem Kreisrand sind orange markiert. Die gestrichelte Linie zeigt den eigentlichen Kreis an, auf dem die Pixel liegen. In diesem Fall hätte der längste Bogen die Länge $n = 10$, da Pixel 16, 1, 2, ..., 9 eine höhere Intensität als p haben. Um diese Ecke zu erkennen, müsste der Schwellwert für den Bogenwinkel also unter $\frac{5}{4}\pi$ liegen.*

Dabei wird eine intuitive Definition einer Ecke herangezogen. Danach ist eine Ecke ein Punkt, in dem zwei Strecken oder drei Flächen zusammenstoßen. Um diese Bedingung zu prüfen, betrachtet der Algorithmus alle Pixel auf dem Rand eines Bresenham-Kreises, um ein Pixel p (Abbildung 15). Dabei wird er längste Bogen gesucht, auf dem die Intensitäten von benachbarten Pixeln auf dem Kreisrand alle höher oder niedriger sind als die Intensität von p (I_p). Die Intensitäten müssen sich hierbei mindestens um einen Schwellwert t von I_p unterscheiden. Ein Pixel gilt als Ecke, wenn der Winkel θ des längsten gefundenen Bogens, über einem vorgegebenen Schwellwert θ_t liegt. Dadurch wird eine ganze Familie von Eckendetektoren beschrieben, da sowohl die Mindestlänge des Bogens n, als auch der Kreisradius r variiert werden können. Wird der Radius beispielsweise als $r = 3$ gewählt und die Mindestbogenlänge $n = 16$ ($\theta_t = 2\pi$), so würden dunkle Punkte in heller Umgebung und umgekehrt als Merkmalspunkte erkannt werden. Da der Bogen ein beliebiger Ausschnitt auf dem Kreisrand sein kann, ist das Verfahren an dieser Stelle rotationsinvariant.

Ein Hauptvorteil dieses Algorithmus ist, dass die Überprüfung, ob es sich um einen Merkmalspunkt handelt oder nicht, stark vereinfacht werden kann. Ist $r = 3$ und $n = 12$ und p kein Merkmalspunkt ist, so kann dies mit sehr wenigen Überprüfungen ermittelt werden. Im einfachsten Fall betrachtet man nur zwei gegenüberliegende Pixel auf dem Kreisrand. Sind die Intensitäten

dieser Pixel beide nah an I_p, also unterhalb von t, so kann der längste Bogen sieben benachbarte Pixel lang sein. Somit kann es sich bei p nicht um einen relevanten Merkmalspunkt handeln.

Da dieses Verfahren häufig direkt benachbarte Pixel als Merkmalspunkte erkennt, müssen diese anschließend per Non-Maximum Unterdrückung in z. B. einer 3x3 Nachbarschaft gefiltert werden. Hierbei muss ein Maß für die Punkte gefunden werden, nach dem entschieden werden kann, welcher von ihnen das Maximum für einen Merkmalspunkt ist. Dazu wir die Summe der Beträge der Differenzen aller Intensitäten der Pixel auf dem Bogen zu der Intensität von p berechnet. Als Merkmalspunkt wird also das Pixel gespeichert, für den diese Summe in seiner Umgebung maximal ist.

2.4 Auswahl des Verfahrens

Für die Feature-Erkennung wird bei diesem Verfahren SURF verwendet. Da SURF eine Weiterentwicklung von SIFT ist, dadurch schneller berechnet werden kann und noch robuster bei der Wiedererkennung ist, ist es SIFT vorzuziehen. Zwar können FAST Features deutlich schneller als SURF berechnet werden. Sie sind allerdings wie beschrieben das instabilste Verfahren. Besonders die Skalierungsinvarianz spielt für das hier vorgestellte Verfahren eine große Rolle, daher wäre FAST in seiner Urform ungeeignet.

3 Bestimmung der Homographie

Das Ergebnis der Feature-Erkennung liefert die Koordinaten paarweise korrespondierender Features in dem Muster und dessen perspektivisch verzerrtem Abbild in einer Aufnahme (z. B. dem aktuellen Videobild). Da es sich um eine ebenes Muster handelt, ist dessen Abbild in der Aufnahme eine projektive Abbildung in die Bildebene der Kamera - also von einer Ebene in eine andere Ebene (Abbildung 16).

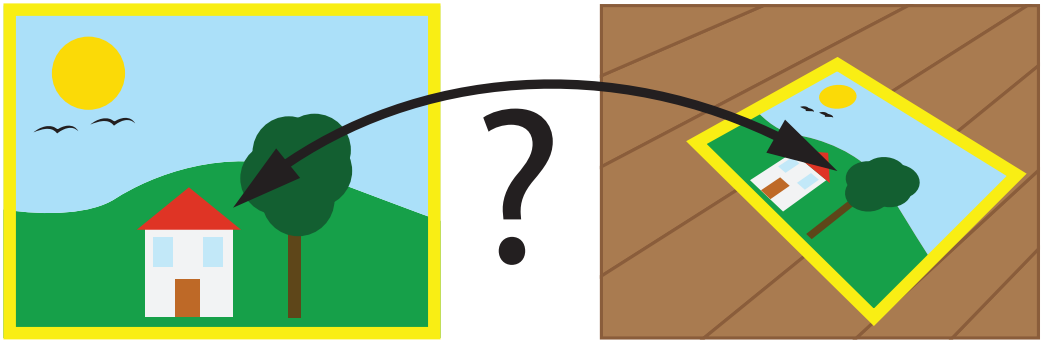


Abbildung 16: *Schematische Darstellung einer perspektivischen Projektion von einer Ebene in einer andere.*

Dieser perspektivische Zusammenhang kann mathematisch über eine Homographie beschrieben werden [HZ04].

3.1 Mathematische Zusammenhänge

Eine Homographie (oder auch Ebenen-Homographie) ist eine invertierbare, reguläre 3×3 Matrix, die Punkte einer Ebene perspektivisch in eine andere Ebene abbildet:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Punkte in homogener Darstellung $\tilde{x}$ werden durch $\tilde{x}' = H\tilde{x}$ wie folgt abgebildet:

$$x'_1 = \frac{x'_1}{x'_3} = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}}{h_{31}x_1 + h_{32}x_2 + h_{33}}, \quad x'_2 = \frac{x'_2}{x'_3} = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}}{h_{31}x_1 + h_{32}x_2 + h_{33}}$$

Es gelten demnach 2 Bedingungen pro Punktepaar. Da H 8 Freiheitsgrade (DOF) hat, kann sie folglich über 4 Punktkorrespondenzen eindeutig bestimmt werden.

Hierfür kann das Ergebnis der Feature-Erkennung verwendet werden, da dabei genau diese Korrespondenzen für das Muster und dessen Abbild bestimmt werden.

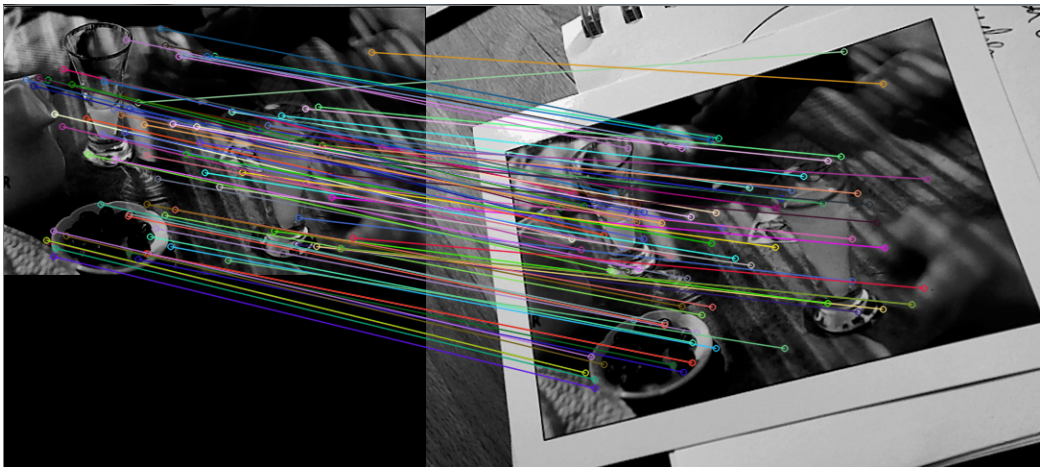


Abbildung 17: *Korrespondierende Features von einem Muster und dessen perspektivisch verzerrtem Abbild im Videobild. Gleiche Features sind dabei über farbige Linien miteinander verbunden.*

Wie in Abbildung 17 zu erkennen ist, werden in den meisten Fällen zum Einen deutlich mehr als die 4 benötigten Korrespondenzen bestimmt und zum Anderen auch Features einander falsch zugeordnet. Daher wird für die Berechnung der Homographie ein RANSAC Verfahren verwendet.

3.2 Berechnung mit RANSAC

Der RANSAC Algorithmus (RANdom Sample And Consensus) [FB81] ist eine Methode, um die Parameter eines vorgegeben Modells aus einer Datenmenge zu bestimmen, die viele Ausreißer enthält. In diesem Fall handelt es sich bei dem Modell um die passende Homographie zu den ermittelten Feature-Korrespondenzen und die Ausreißer sind die Fehlzuordnungen.

Das Verfahren ist in 2 Hauptschritte unterteilt. Zunächst werden im ersten Schritt willkürlich 4 Korrespondenzen aus der Gesamtmenge ausgewählt, um daraus eine hypothetische Homographie zu bestimmen. Im zweiten Schritt

wird für alle anderen Korrespondenzen $(x, \tilde{x})$ geprüft, ob sie die Homographiegleichung $\tilde{x}' = H\tilde{x}$ (innerhalb einer geringen Abweichung) erfüllen und damit die Hypothese bestätigen. Ist dies der Fall, werden sie zur sogenannten „Konsensmenge“ dieser Homographie hinzugefügt. Die beiden Schritte werden solange wiederholt, bis die Wahrscheinlichkeit, eine Homographie mit einer größeren Konsensmenge zu finden, unterhalb einer bestimmten Grenze ist. Das Ergebnis des Verfahrens ermittelt demnach die Homographie zu den 4 Korrespondenzen mit der größten Konsensmenge, wodurch Ausreißer (bis zu einem Anteil von ca. 50% an der Korrespondenzenmenge) automatisch keinen Einfluss haben.

4 Berechnung der Pose

In den folgenden Abschnitten wird beschrieben, wie die Gesamttransformation für die 3D-Objekte berechnet wird, damit diese perspektivisch korrekt auf dem Muster dargestellt werden. Diese Gesamttransformation ist in eine projektive Kameratransformation von Kamera- in Bildkoordinaten und eine euklidische Ansichtstransformation von Muster- in Kamera-/Weltkoordinaten aufgeteilt.

4.1 Projektive Transformation der Kamera

Im Folgenden wird davon ausgegangen, dass die Kalibriermatrix K bekannt ist. Sie kann beispielsweise mit Hilfe einer Offline-Kalibrierung mit einem Schachbrettmuster ermittelt worden sein. Bei K handelt es sich um das Lochkameramodell der kalibrierten Kamera:

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Diese 3×3 Matrix enthält die intrinsischen Parameter der Kamera bezüglich der Pixelgröße. Sie definiert die lineare Transformation von 3D-Weltkoordinaten zu homogenen 2D-Koordinaten im Kamerabild.

In Anlehnung an die meisten Grafik-APIs wird der Sichtbereich der Kamera bei diesem Verfahren als Pyramidenstumpf zwischen einer sogenannten Nah- und einer Fernebene betrachtet (Abbildung 18).

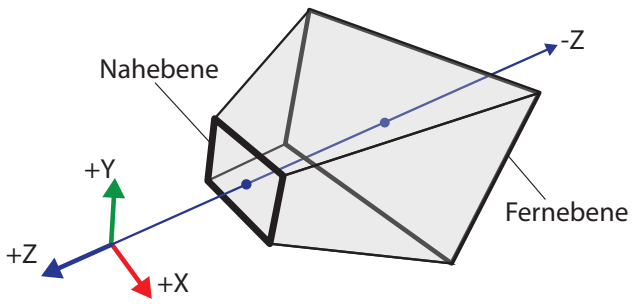


Abbildung 18: *Veranschaulichung des Koordinatensystems und Sichtbereichs der Kamera.*

Die Kamera ist in diesem Fall durch ein rechtshändiges Koordinatensystem definiert.

Dieses Sichtvolumen kann unter Berücksichtigung von K als die folgende 4×4 Matrix P_{pers} definiert werden:

$$P_{pers} = \begin{pmatrix} \frac{2 \cdot f_x}{w} & 0 & 1 - \frac{2 \cdot c_x}{w} & 0 \\ 0 & \frac{2 \cdot f_y}{h} & -1 + \frac{2 \cdot c_y + 2}{h} & 0 \\ 0 & 0 & \frac{zFar + zNear}{zNear - zFar} & \frac{2 \cdot zFar \cdot zNear}{zNear - zFar} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Da K bezüglich des Bildkoordinatensystems bestimmt wurde, muss hier eine Normierung vorgenommen werden, wobei w die Breite und h die Höhe des Kamerabildes sind. Die Entfernung der Nah- ($zNear$) und Fernebene ($zFar$) sind ebenfalls in der Matrix enthalten. Sie beschreibt die homogene Transformation von 3D-Kamerakoordinaten in 2D-Bildkoordinaten.

4.2 3D-Pose des Musters aus der Homographie

In diesem Schritt wird aus der Homographie die 3D-Pose des Musters im Kamerakoordinatensystem (Abbildung 19) berechnet. Die Pose besteht aus der $t = (m_x, m_y, m_z)^T$ und der Rotation R um die 3 Achsen und hat somit 6 Freiheitsgrade.

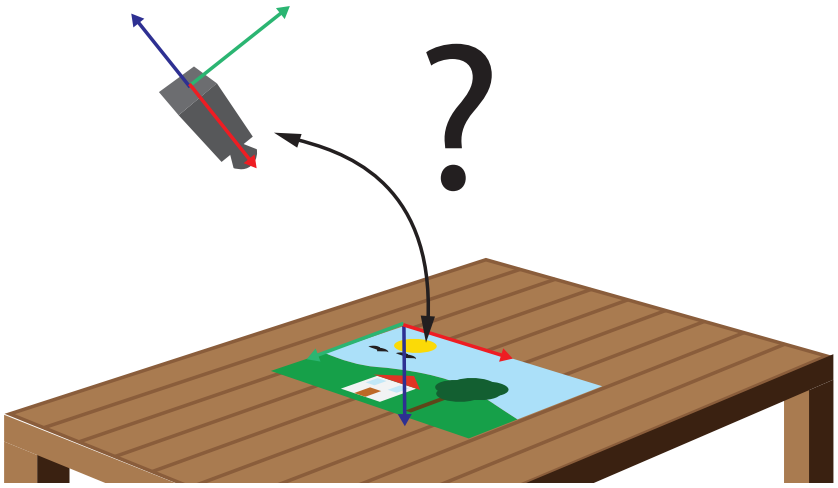


Abbildung 19: *Beispielhafte Lagenbeziehung von Kamera- und Musterkoordinatensystem.*

Allgemein lässt sich der Zusammenhang von Bildkoordinaten $(x'_1, x'_2, 1)$ und Weltkoordinaten (x_1, x_2, x_3) durch eine projektive Matrix $P_{3\times4} = K_{3\times3} \cdot R_{3\times3} \cdot [I|t]_{3\times4}$ beschreiben.

Dabei gilt

$$\begin{pmatrix} x_1' \\ x_2' \\ 1 \end{pmatrix} = P \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} = K \cdot [r_1, r_2, r_3, t] \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}.$$

Wobei $R_{3\times 3} = [r_1, r_2, r_3]$ die Rotation und $t_{3\times 1}$ die Position der Pose darstellen. Die Homographie H bildet ebenfalls die Koordinaten der xy-Ebene des Musters $\tilde{m}$ auf Koordinaten $\tilde{m}'$ in der xy-Bildebene der Kamera ab. Daher ist sie ein Spezialfall der projektiven Abbildung, bei der gilt

$$\begin{pmatrix} \tilde{m}_x' \\ \tilde{m}_y' \\ 1 \end{pmatrix} = H \cdot \begin{pmatrix} \tilde{m}_x \\ \tilde{m}_y \\ 0 \\ 1 \end{pmatrix} = K \cdot [r_1, r_2, 0, t] \cdot \begin{pmatrix} \tilde{m}_x \\ \tilde{m}_y \\ 0 \\ 1 \end{pmatrix}$$

und demnach

$$H = K \cdot [r_1, r_2, t].$$

Da H und K bekannt sind, kann die Pose des Musters aus $H' = K^{-1} \cdot H$ bestimmt werden. Außerdem handelt es sich bei R um eine Rotationsmatrix, weswegen folgende zusätzliche Bedingungen gelten müssen:

$$||r_1|| = ||r_2|| = ||r_3|| = 1, \quad r_1 \perp r_2, \quad r_1 \perp r_3, \quad r_2 \perp r_3$$

Daher lässt sich R aus H' folgendermaßen berechnen [XKM09]:

$$r_1 = \frac{h_1'}{||h_1'||}, \quad r_2 = \frac{h_2'}{||h_2'||}, \quad r_3 = r_1 \times r_2$$

Wobei h_1' und h_1' der erste und zweite Spaltenvektor von H' sind. Da r_1 und r_2 in der Praxis meist nicht exakt senkrecht aufeinander stehen, wird dies nachträglich durch $r_1 = r_2 \times r_3$ korrigiert.

Die Position t des Musters berechnet sich als

$$t = \frac{2 \cdot h_3'}{||h_1'|| + ||h_2'||}.$$

Passend zu der perspektivischen Projektionsmatrix kann nun die 4×4 Matrix T aufgestellt werden, die die Transformation von Musterkoordinaten in Kamera-/Weltkoordinaten beschreibt:

$$T = \begin{pmatrix} [r_1, r_2, r_3] & t \\ 0 & 1 \end{pmatrix}$$

Als letzter Schritt ist noch zu beachten, dass im Musterkoordinatensystem die y-Achse nach unten und die z-Achse in das Muster hinein zeigt, wie in Abbildung 19 zu erkennen ist. Wie zuvor erwähnt, ist das im Kamerakoordinatensystems jedoch genau umgekehrt. Daher muss T zuletzt noch an der xz-Ebene, sowie an der xy-Ebene gespiegelt werden:

$$T' = S \cdot T, \quad S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Somit wird das Muster- in das Kamerakoordinatensystem transformiert.

4.3 Stabilisierung mittels Kalman-Filter

Auch ohne Bewegung der Kamera oder des Musters werden die Features auf Grund von Bildrauschen nicht immer an exakt denselben Positionen erkannt. Dieses Rauschen macht sich folglich auch in der Homographie bemerkbar. Bei der anschließenden Bestimmung der Kamerapose führt dies dazu, dass die 3D-Objekte nicht ruhig auf dem Muster stehen, sondern sich ständig in einem kleinen Bereich bewegen (zittern).

Aus diesem Grund werden sowohl der Positionsvektor t als auch die Rotationsmatrix R mittels Kalman-Filtern [Kal60] geglättet, um das Messrauschen zu reduzieren. Es kann davon ausgegangen werden, dass sowohl die Kamera als auch das Muster in den meisten Fällen von Menschen bewegt werden. Die Stärke der Glättung im Fall der Position ist deutlich geringer als bei der Rotation. Wäre dies nicht der Fall, würde die Erkennung in bestimmten Situationen träge erscheinen, da Positionsänderungen der Kamera bzw. des Musters deutlich abrupter und schlagartiger vollzogen werden können als Rotationen.

5 Implementierung

Im Rahmen dieses Projekts wurde eine kleine Augmented Reality-Anwendung namens *SurfAR* geschrieben, die dazu verwendet werden kann, 3D-Modelle auf beliebigen ebenen Mustern (z. B. Postkarten, Verpackungen oder Poster) in Echtzeit im Videobild einer RGB-Kamera darzustellen.

Die Anwendung ist in C++ geschrieben und verwendet OpenCV 2.3.1 [2] für Bildverarbeitungsfunktionen, VTK 5.8.0 [3] für die Darstellung des Videobildes und der 3D-Modelle sowie Qt 4.7.4 [4] als GUI-Framework.

Entwickelt wurde das Programm auf einem Microsoft Windows 7-Rechner mit einer Intel Core i7 Quadcore (2.67GHz) CPU mit einer Nvidia 260 GTX Grafikkarte. Bei der verwendeten Kamera handelt es sich um eine Logitech HD Pro C910 USB Webcam, die mit einer Auflösung von 640×480px verwendet wurde.

Den Kern der Anwendung bildet die C++ Klasse *MLARTracker*, die eine einfache Schnittstelle zur Berechnung der Kamerapose bietet und daher auch ohne großen Aufwand in anderen C++ Anwendugen wieder verwendet werden kann. In dieser Klasse ist das in Abschnitt 4 beschriebene Verfahren implementiert. Für die Feature-Erkennung wird die GPU-Implementierung für SURF von OpenCV verwendet, durch die die Berechnung in Echtzeit möglich wird. Für die Kalibrierung der intrinsischen Kameramatrix wurde die GML Camera Calibration Toolbox 0.4 [5] verwendet.

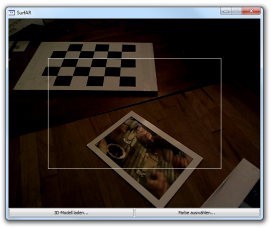
Wie die bestehende Software bedient und wie die *MLARTracker*-Klasse in eigene Programme eingebunden werden kann, wird in den folgenden Abschnitten erläutert.

5.1 Verwendung der Software

Nachdem das Programm gestartet wurde, muss der Benutzer zunächst eine XML-Datei auswählen, die die Kalibrierparameter der verwendeten Kamera enthält. Wie diese Datei im Detail aussehen muss, wird in den folgenden Abschnitten erklärt. Daraufhin wird, sofern eine unterstützte Kamera angeschlossen ist, das Videobild angezeigt (Abbildung 20(a)). Außerdem sieht man einen weißen Rahmen, der dabei helfen soll, einen geeigneten Bildausschnitt als Muster für das Tracking auszuwählen. Um dies zu tun, sollte das

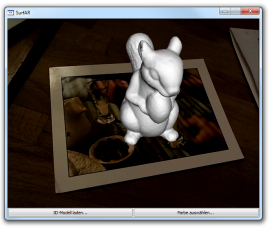
[2] http://opencv.willowgarage.com/wiki/
[3] http://www.vtk.org/
[4] http://qt.nokia.com/
[5] http://graphics.cs.msu.ru/ru/science/research/calibration/cpp

Muster so gefilmt werden, dass der gewünschte Ausschnitt möglichst Achsenparallel in dem weißen Rahmen ist (Abbildung 20(b)).



(a) Videobild mit weißem Hilfsrahmen.

(b) Auswahl des Musters.



(c) 3D-Modell auf dem Muster.

Abbildung 20: *Ablauf beim Aufnehmen eines Musters, auf dem ein 3D-Modell dargestellt werden soll.*

Durch Drücken der Leertaste kann das Muster für das Tracking ausgewählt werden, woraufhin das aktuell geladene 3D-Modell auf im Videobild auf dem Muster angezeigt wird (Abbildung 20(c)). Über den Knopf *3D-Modell laden...* können 3D-Modelle im Wavefront OBJ-Format geladen und auf dem Muster dargestellt werden. Außerdem kann die Farbe dieser Modelle über den Knopf *Farbe auswählen...* verändert werden. Durch Drücken der r-Taste kann das Muster gelöscht und damit das Tracking abgebrochen werden. Daraufhin befindet sich die Anwendung wieder im Ausgangszustand, und ein neues Muster kann, wie zuvor beschrieben, ausgewählt werden.

5.2 Einbinden in OpenGL

Zur Anzeige des mit dem 3D-Objekt überlagerten Videobildes kann in OpenGL der Z-Buffer verwendet werden. Dazu wird das Videobild als Textur auf ein Rechteck bei Orthogonalprojektion und ausgeschaltetem Z-Buffer vor der 3D-Szene gezeichnet. Hierfür kann ein Bild im OpenCV-Format wie folgt als OpenGL Textur verpackt werden:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, frame.cols, frame.rows, \\
                0, GL_BGR_EXT, GL_UNSIGNED_BYTE, frame.data);
```

Für das Tracking und die Bestimmung der Musterpose wird lediglich die C++ Klasse *MLARTracker* benötigt. Eine Tracker-Instanz bekommt im Konstruktor lediglich den Pfad zu einer XML-Datei, in der sowohl die Kalibriermatrix als auch die Verzerrungskoeffizienten der Kamera als OpenCV-Matrix stehen:

```
MLARTracker* mlarTracker = new MLARTracker("pfadzur.xml");
```

Die XML-Datei muss dem Format aus Listing 1 entsprechen. Die Einträge *fx, fy, cx, cy* und *dist1- dist4* sind dabei nur Platzhalter für die bei der Kalibrierung ermittelten konkreten Werte.

```
<?xml version="1.0"?>
<opencv_storage>
<CalibrationMatrix type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
        fx  0  cx  0  fy  cy  0  0  1
   </data>
</CalibrationMatrix>
<Distortion type_id="opencv-matrix">
  <rows>4</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
        dist1  dist2  dist3  dist4
  </data>
</Distortion>
</opencv_storage>
```

Listing 1: *Formatvorgabe für die XML-Datei mit den intrinsischen Kameraparametern.*

Um das Tracking eines Musters zu starten, muss der Tracker noch initialisiert werden. Hierzu muss die Funktion *init* aufgerufen werden:

```
mlarTracker->init(sample, width, height, nearPlane, farPlane);
```

Dieser wird als erster Parameter der Bildausschnitt des Musters als OpenCV-Matrix übergeben. Zusätzlich müssen noch Breite und Höhe des Anzeigefensters sowie die Nah- und Fernebene für die perspektivische Projektionsmatrix angegeben werden. Die Funktion hat einen booleschen Rückgabewert, der angibt, ob in dem übergebenen Bildauschnitt genügend (mindestens 4) Features erkannt wurden, um damit eine Homografie zu berechnen.

Nach einer erfolgreichen Initialisierung kann zunächst die 4×4 perspektivische Projektionsmatrix mit der Funktion *getProjectionMatrix* vom Tracker geholt und in OpenGL gesetzt werden:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
Mat projectionMat;
transpose(mlarTracker->getProjectionMatrix(), projectionMat);
glMultMatrixd(projectionMat.ptr<double>());
```

Hierbei ist zu beachten, dass die Matrix transponiert werden muss, da in OpenGL alle Matrizen spaltenweise angegeben werden.

Um zu einem Bild des Musters die Transformationsmatrix für die 3D-Objekte zu berechnen, muss zunächst das jeweilige Bild an die Funktion *process* übergeben werden:

```
mlarTracker->process(currentFrame);
```

Dieser Aufruf aktualisiert die interne Transformationsmatrix des Trackers anhand des erkannten Musters. Die aktualisierte Matrix kann anschließend mit der Funktion *getModelViewMatrix* vom Tracker geholt und an OpenGL für die Darstellung der 3D-Objekte übergeben werden:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
Mat modelViewMat;
transpose(mlarTracker->getModelViewMatrix(), modelViewMat);
glMultMatrixd(modelViewMat.ptr<double>());

// 3D-Objekte zeichnen...
```

Der Tracker kann jederzeit mit einem neuen Muster per *init* wieder initialisiert werden.

5.3 Einbinden in VTK

Die Kombination aus Videobild und 3D-Szene kann in VTK (Visualization Toolkit) über *Render-Ebenen* erzielt werden:

```
vtkRenderWindow* renderWindow = new vtkRenderWindow();
renderWindow->SetNumberOfLayers(2);
// Renderer fuer das Videobild
vtkRenderer* videoRenderer = new vtkRenderer();
videoRenderer->SetLayer(0);
renderWindow->AddRenderer(videoRenderer);
// Renderer fuer die 3D-Szene
vtkRenderer* sceneRenderer = vtkRenderer();
vtkRenderer->SetLayer(1);
renderWindow->AddRenderer(sceneRenderer);
```

Hierfür wird ein *vtkRenderer* für das Videobild und einer für die 3D-Szene erstellt und dem *vtkRenderWindow* auf unterschiedlichen Ebenen hinzugefügt. Dadurch wird das Videobild immer hinter der 3D-Szene dargestellt, ohne explizit den Z-Buffer zu verwenden.

Das Einbinden des *MLARTrackers* in VTK funktioniert im Prinzip genauso wie das Einbinden in OpenGL. Jedoch müssen die perspektivische Projektionsmatrix (*getProjectionMatrix*) und die Transformationsmatrix (*getModelViewMatrix*) in VTK anders verwendet werden, da hier nicht direkt auf den Matrix-Stack zugegriffen werden kann.

Die Projektionsmatrix muss in VTK parameterweise an die *vtkCamera* des *sceneRenderers* übergeben werden. Da die Projektionsmatrix folgendermaßen aufgebaut ist

$$P = \begin{pmatrix} \frac{f}{aspect} & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2 \cdot zFar \cdot zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

und

$$f = cot(fovy/2),$$

können die einzelnen Einträge genutzt werden, um damit das optische Zentrum und den Öffungswinkel einer *vtkCamera* einzustellen:

```
vtkCamera* cam =  sceneRenderer->GetActiveCamera();
Mat P = mlarTracker->getProjectionMatrix();
double cx = P.at<double>(0, 2);
double cy = P.at<double>(1, 2);
double f = P.at<double>(1, 1);
// Berechnung des vertikalen Oeffungswinkels
double fovy = ((3.1415926/2.0 - atan(f))*2) ;
// Setzen der Kameraparameter
cam->SetClippingRange(nearPlane, farPlane);
cam->SetPosition(cx, cy, 0.0);
cam->SetFocalPoint(cx, cy, -1.0);
cam->SetViewAngle(fovy * 180 / 3.1415926);
```

Die Transformationsmatrix muss in VTK den 3D-Objekten, die auf das Muster transformiert werden sollen, als *UserMatrix* gesetzt werden:

```
vtkMatrix4x4* modelViewMatrix = new vtkMatrix4x4();
modelViewMatrix->DeepCopy(tracker->getModelViewMatrix().ptr<double>());
sceneActor->SetUserMatrix(modelViewMatrix);
sceneActor->Modified();
```

Dazu muss die Matrix des Trackers lediglich in eine *vtkMatrix4x4* verpackt werden.

6 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Verfahren entwickelt, um die Pose von beliebigen ebenen Mustern in Echtzeit stabil in Videobildern zu berechnen. Dadurch bietet es eine robuste Grundlage für verschiedene Augmented Reality-Anwendungen. Um dies zu demonstrieren, wurde außerdem eine kleine AR-Anwendung entwickelt, mit der zur Laufzeit Muster ausgewählt werden können, auf denen anschließend 3D-Modelle im Videobild angezeigt werden. Das Verfahren ist, besonders im Vergleich zu Marker-basierten Alternativen, sehr stabil gegenüber starken Beleuchtungs- und Perspektivenänderungen. Da die Mustererkennung rein Feature-basiert ist, kann die Pose des Musters auch stabil bestimmt werden, obwohl große Teile des Musters verdeckt oder außerhalb des Sichtbereichs der Kamera sind.

Auf dieser Grundlage könnten nun auch komplexere AR-Anwendungen entwickelt werden, da sich das entwickelte Modul problemlos in OpenGL und VTK Anwendungen integrieren lässt.

Außerdem ist es denkbar, dass die hierbei gewonnenen Erkenntnisse über Feature-Erkennung als Grundlage für sogenannte „Structure from Motion“ Verfahren dienen.

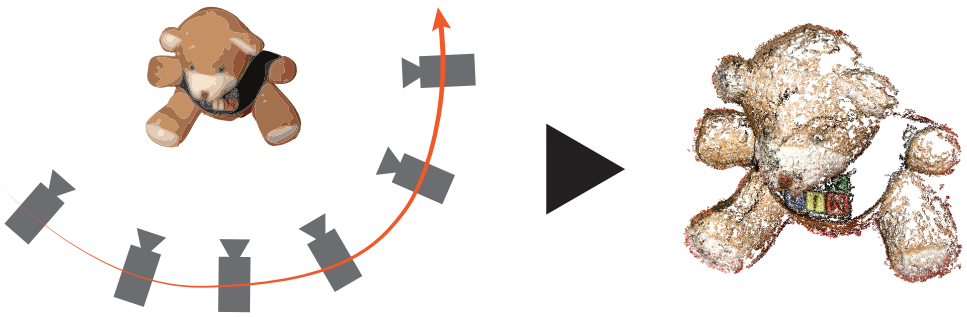


Abbildung 21: *Veranschaulichung von Structure from Motion Verfahren.*

Dabei wird die Kamerapose aus Punktkorrespondenzen aus aufeinander folgenden Bildern bestimmt, um damit eine 3D-Rekonstruktion der gefilmten Szene zu erstellen (Abbildung 21). Daher wäre ein solches Verfahren im Kern sehr ähnlich zu dem hier vorgestellten und könnte einige der Konzepte wiederverwenden.

Literatur

[BTG06] Bay, Herbert, Tinne Tuytelaars und Luc Van Gool: *Surf: Speeded up robust features.* In: *In ECCV*, Seiten 404–417, 2006.

[FB81] Fischler, Martin A. und Robert C. Bolles: *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography.* Commun. ACM, 24(6):381–395, Juni 1981.

[HS88] Harris, C. und M. Stephens: *A Combined Corner and Edge Detection.* In: *Proceedings of The Fourth Alvey Vision Conference*, Seiten 147–151, 1988.

[HZ04] Hartley, R. I. und A. Zisserman: *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, Second Auflage, 2004.

[Kal60] Kalman, R. E.: *A New Approach to Linear Filtering and Prediction Problems.* Transactions of the ASME - Journal of Basic Engineering, (82 (Series D)):35–45, 1960.

[KB99] Kato, Hirokazu und Mark Billinghurst: *Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System.* In: *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, Seiten 85–, Washington, DC, USA, 1999. IEEE Computer Society.

[KM07] Klein, Georg und David Murray: *Parallel Tracking and Mapping for Small AR Workspaces.* In: *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[Low99] Lowe, David: *Object Recognition from Local Scale-Invariant Features.* Seiten 1150–1157, 1999.

[RD05] Rosten, Edward und Tom Drummond: *Fusing points and lines for high performance tracking.* In: *IEEE International Conference on Computer Vision*, Band 2, Seiten 1508–1511, 2005.

[XKM09] Xu, Changhai, Benjamin Kuipers und Aniket Murarka: *3D pose estimation for planes.* In: *ICCV Workshop on 3D Representation for Recognition (3dRR-09)*, 2009.