

Sensordatenverarbeitung

Das vorige Kapitel ist auf unterschiedliche Sensoren eingegangen, die digitale Daten aus der Umgebung aufnehmen. Dieses Kapitel geht nun auf Möglichkeiten ein, diese Daten zu verarbeiten und sie nutzbar zu machen für Anwendungen späterer Kapitel. Dazu gehören die Filterung der Daten, Extraktion bestimmter Merkmale sowie weiterführende Berechnung von Informationen wie Tiefendaten aus zweidimensionalen Kamerabildern. Während Filter üblicherweise gezielt Daten wegwerfen, beispielsweise zur Entfernung von Messfehlern, aggregieren Merkmale die Daten zu neuartigen Informationen. Ein wichtiges Kriterium der hier vorgestellten Algorithmen ist stets auch ihre Laufzeit: Nur unaufwändige Verfahren sind auf mobilen Robotern mit hochfrequenten Datenströmen und vergleichsweise geringen Rechnerressourcen einsetzbar.

Der erste Teil des Kapitels beschäftigt sich mit elementaren Algorithmen auf Entfernungsdaten, beispielsweise von einem Laserscanner; die restlichen Abschnitte gehen auf unterschiedliche Aspekte der Verarbeitung von Kamerabildern ein.

3.1 Entfernungsdaten

3.1.1 Messfehler filtern

Reale Daten sind fehlerbehaftet. Zu den Fehlern zählen Rauschen in den Messdaten, sowie Ausreißer. Nachfolgend werden wir zwei einfache Möglichkeiten beschreiben, beide Arten von Fehlern in Entfernungsdaten zu reduzieren.

Reduktionsfilter

Ein Reduktionsfilter dient zur Datenreduktion und ersetzt die Messpunkte durch eine (signifikant kleinere) Menge von Datenpunkten, die die ursprüngli-

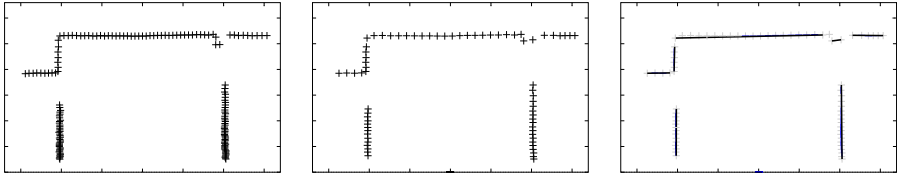


Abbildung 3.1: Reduktions- und Online-Linienfilter. Links: Messpunkte. Mitte: Reduzierte Punkte. Rechts: Gefundene Linien auf den Punkten.

chen Punkte möglichst gut approximieren soll. Dies geht üblicherweise einher mit einer lokalen Mittelung der Messwerte, was stochastisches Rauschen in den Daten reduziert. Abbildung 3.1 (Mitte) verdeutlicht die Anwendung eines solchen Reduktionsfilters.

Algorithmisch werden in Reihenfolge einkommende Punkte $\mathbf{a}_i, \mathbf{a}_{i+1}, \dots, \mathbf{a}_j$ so lange sukzessive akkumuliert, wie der Abstand zwischen dem ersten und dem aktuellen Punkt eine feste Grenze nicht übersteigt, also so lange gilt: $\|\mathbf{a}_i, \mathbf{a}_j\| \leq \delta$. Ist durch den nächsten Punkt \mathbf{a}_{j+1} diese Grenze überschritten, werden die akkumulierten Punkte gelöscht und durch ihren Mittelwert

$$\frac{1}{j-i+1} \sum_{k=i}^j \mathbf{a}_k \quad (3.1)$$

ersetzt. \mathbf{a}_{j+1} dient als Startpunkt für die nächste Akkumulation.

Der Filter verringert nicht nur Sensorrauschens, sondern wirkt sich auch auf die Verteilung der Messpunkte aus: Diese sind in den Originaldaten üblicherweise im Nahbereich des Sensors (z.B. Laserscanners) sehr viel dichter aufgelöst als in der Entfernung. In den gefilterten Daten dagegen sind die Punkte ausgedünnt, aber auch gleichmäßiger verteilt. Wenn die Messdaten sortiert vorliegen – z.B. entgegen dem Uhrzeigersinn, wie es bei Laserscannern typischerweise der Fall ist – so läuft dieser Filter linear in der Anzahl der Punkte.

Medianfilter

Der Zweck eines Medianfilters ist, einzelne grobe Messfehler (Ausreißer) zu entfernen – die Anzahl der Datenpunkte ändert sich dabei jedoch nicht. Dies ist offensichtlich nur approximativ möglich, da im Allgemeinen eine Unterscheidung zwischen Messdatum und Messfehler nicht sicher aus den Daten ersichtlich ist.

Ein Medianfilter ersetzt zu diesem Zweck jeden Punkt durch den Median der k umliegenden Punkte: Der gefilterte Punkt \mathbf{a}'_i zu einem Messpunkt \mathbf{a}_i ergibt sich somit als der $\lceil k/2 \rceil$ -te Punkt der sortierten Folge der Punkte

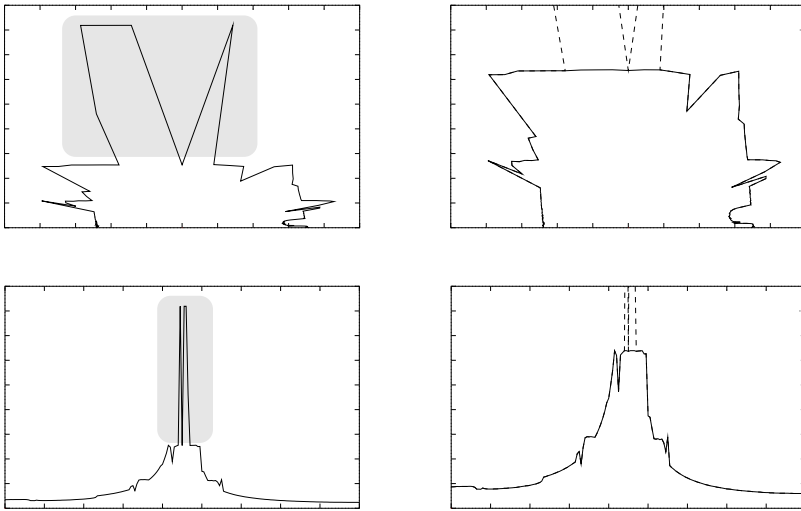


Abbildung 3.2: Medianfilter. Links: Originaldaten, mit grau markierten Ausreißern (oben: Kartesische, unten Polar-Darstellung). Rechts: Der Median-gefilterte Scan (Zoom auf den relevanten Bereich), gestrichelt unterlegt der Originalscan.

$\mathbf{a}_{i-\lfloor \frac{k}{2} \rfloor}, \dots, \mathbf{a}_{i+\lfloor \frac{k}{2} \rfloor}$. Die Sortierung erfolgt beispielsweise nach den Abstandswerten der Punkte in Polardarstellung. Das Ergebnis eines Medianfilters der Größe $k = 7$ ist in Abbildung 3.2 dargestellt. Die Laufzeit ergibt sich bei intuitiver Implementierung zu $\mathcal{O}(nk \log k)$ bei n Datenpunkten.

3.1.2 Linienerkennung

Die im Folgenden beschriebenen Filter haben den Zweck, aus Messdaten eine Menge von Linien zu extrahieren, so dass benachbarte Punkte, die in etwa auf einer Geraden liegen, zu einer Linie zusammengefasst werden. Die gleichen Algorithmen arbeiten selbstverständlich auch auf Bildern, wenn in ihnen zuvor Kanten detektiert und die so gefilterten Bilder binarisiert werden. In Abschnitt 3.2.2 gehen wir näher auf die Kantendetektion ein.

Darüber hinaus sind Linien auch zur Reduzierung von Rauschen einsetzbar: Werden Datenpunkte durch Linien ersetzt und diese Linien ihrerseits durch äquidistant verteilte Punkte approximiert (engl. *subsampling*), werden die Daten an diesen Stellen signifikant geglättet, wie Abbildung 3.3 zeigt.

Online-Linienfinder

Das Hauptmerkmal des in diesem Abschnitt beschriebenen Linienfilters liegt darin, dass die Daten im Gegensatz zu den übrigen hier vorgestellten Verfah-

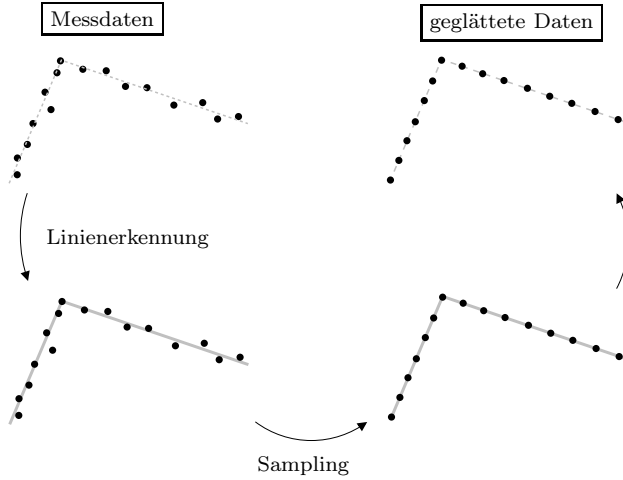


Abbildung 3.3: Verwendung eines Linienfilters zum Glätten der Daten.

ren online verarbeitet werden können. Dies bedeutet, dass für die Berechnungen zu einem Datum lediglich die Vorgänger-Messdaten bekannt sein müssen. Die Messdaten können somit direkt bei Eingang prozessiert werden. Dafür liefern die (i.Allg. langsameren) offline-Verfahren jedoch oftmals bessere Ergebnisse.

Die online-Fähigkeit des Verfahrens wird unterstützt durch eine Sortierung der Daten. Diese ist üblicherweise bei Laserscannern natürlich gegeben, die die Daten in einer festen Reihenfolge, beispielsweise im Gegenuhrzeigersinn, liefern. Die einzelnen Daten eines Scans werden sukzessive durchlaufen. Punkte werden nun so lange zu einer Linie addiert, wie die Abweichung des Punktes zu der durch die Vorgänger definierten Linie einen Schwellwert nicht überschreitet. Andernfalls wird die Vorgängerlinie beendet, und der aktuelle Punkt bildet den Start eines neuen Linien-Kandidaten. Linien gelten als gefunden, wenn mindestens k Punkte auf ihr liegen (z.B. $k = 3$). Um die Auswirkung von Rauschen zu vermindern, kann initial ein Reduktionsfilter auf den Scan angewendet, bzw. direkt in die Online-Liniensuche eingebaut werden (Abbildung 3.1).

Sei $\langle \mathbf{a}_j, \dots, \mathbf{a}_k \rangle$ die bisher aktuell konstruierte Linie. Damit der Punkt \mathbf{a}_{k+1} mit in die Linie aufgenommen wird, müssen folgende Bedingungen erfüllt sein:

1. Mit dem neuem Punkt darf die Streckensumme nur wenig von der Luftlinie abweichen:

$$1 \geq \frac{\|\mathbf{a}_j, \mathbf{a}_{k+1}\|}{\sum_{i=j}^k \|\mathbf{a}_i, \mathbf{a}_{i+1}\|} > 1 - \varepsilon_k . \quad (3.2)$$

Wegen der Dreiecksungleichung ist das Verhältnis Luftlinie zu aufsummierten Einzelstrecken stets kleiner gleich Eins (die linke Ungleichung ist also stets erfüllt), soll aber auch nicht signifikant größer sein. Die zulässige Abweichung soll mit steigender Länge der Linien ebenfalls steigen, d.h. der Parameter ε_k wird mit zunehmendem k größer.

2. Gleiches gilt insbesondere lokal am Ort der Linienerweiterung (typisch: $\varepsilon = 0.2$):

$$1 \geq \frac{\|\mathbf{a}_{k-1}, \mathbf{a}_{k+1}\|}{\|\mathbf{a}_{k-1}, \mathbf{a}_k\| + \|\mathbf{a}_k, \mathbf{a}_{k+1}\|} > 1 - \varepsilon. \quad (3.3)$$

3. Der euklidische Abstand zwischen \mathbf{a}_k und \mathbf{a}_{k+1} darf ein festes Maximum nicht überschreiten – andernfalls würde die Linie durch „leere“ Bereiche laufen, also zwei nicht zusammenhängende Liniensegmente miteinander verbinden.

Die so gefundenen Linien verbinden jeweils den ersten mit dem letzten Punkt der sortierten Menge von Punkten, die als zu einer Linie gehörig erkannt worden sind. Eleganter ist es, für jede dieser Mengen die jeweils optimale Ausgleichslinie zu finden, beispielsweise über einen *least squares*-Ansatz (Methode der kleinsten Quadrate). Alternativ kann an dieser Stelle die Berechnung der Linie gemäß dem nächsten Abschnitt (Formel 3.6) erfolgen.

Tangentiallinien

Eine weitere Methode, Linien in Punktdaten zu finden, bedient sich Tangentiallinien. Eine solche Tangentiallinie zu einem Punkt \mathbf{P} mit Polarwinkel θ erhält man, indem man eine Regressionsgrade durch seine umgebenden Punkte wie folgt berechnet:

Für einen Punkt \mathbf{P} unter Winkel θ im Laserscan wird eine Regressionsgerade durch $(k-1)/2$ Nachbarpunkte rechts und links bestimmt. Sei r die Normaldistanz zu der Linie, sowie $|\theta - \varphi|$ die Winkeldifferenz zwischen der Linie \overline{OP} und der Normalen der Tangente, wie in Abbildung 3.4 skizziert.

Lemma 1. *Die gesuchte Linie an Punkt \mathbf{P} ist jene, welche den folgenden Fehlerterm über seine k Nachbarpunkte minimiert:*

$$E(\varphi, r) = \sum_{i=1}^k (x_i \cos \varphi + z_i \sin \varphi - r)^2. \quad (3.4)$$

Beweis. Siehe [LM97b].

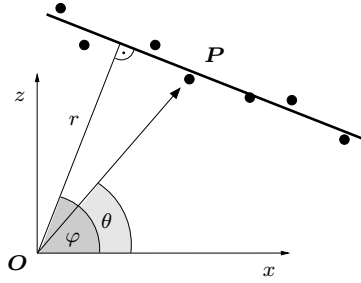


Abbildung 3.4: Liniensuche über Tangentiallinien.

Aus dem vollständigen Beweis ergibt sich, dass zur Minimierung von (3.4) eine geschlossene Lösung existiert, nämlich:

$$\begin{aligned}\varphi &= \frac{1}{2} \arctan \frac{-2S_{xz}}{S_{zz} - S_{xx}} \\ r &= \bar{x} \cos \varphi + \bar{z} \sin \varphi\end{aligned}\tag{3.5}$$

mit

$$\begin{aligned}\bar{x} &= \frac{1}{k} \sum_i x_i & S_{xx} &= \sum_i (x_i - \bar{x})^2 \\ \bar{z} &= \frac{1}{k} \sum_i z_i & S_{zz} &= \sum_i (z_i - \bar{z})^2 \\ & & S_{xz} &= \sum_i (x_i - \bar{x})(z_i - \bar{z})\end{aligned}$$

und es gilt:

$$\min_{r, \varphi} E(r, \varphi) = \frac{1}{2} (S_{xx} + S_{zz} - \sqrt{4S_{xz}^2 + (S_{zz} - S_{xx})^2}) . \tag{3.6}$$

Jedoch sind nicht alle derart berechneten Tangentiallinien sinnvoll zu verwenden. Üblicherweise werden die Linien von Punkten, die nahe an Ecken oder Tiefensprüngen liegen, wieder gelöscht. Kriterien dafür sind zum einen Grenzwerte, die die Winkeldifferenz $|\theta - \varphi|$ sowie die Summe der Distanzquadrate zwischen den k Nachbarpunkten nach oben hin begrenzen. Ein weiterer Anhaltspunkt ist der residuale Wert der minimierten Fehlerfunktion E . Ein großer Wert deutet auf eine niedrige Kollinearität der Nachbarpunkte hin, was in einer schlecht passenden Tangentiallinie resultiert.

Tangentiallinien benachbarter Punkte, die obige Kriterien erfüllen, können nun in Abhängigkeit ihrer Parameter (φ, r) zu längeren Liniensegmenten zusammengefügt werden.

Hough-Transformation

Die Hough-Transformation ist ein Algorithmus zur Detektion von parametrisierten geometrischen Objekten. Im Folgenden gehen wir ausschließlich auf die Erkennung von Linien ein; eine weitere Standardanwendung liegt in der Erkennung von Kreisen oder Ellipsen. In gleicher Weise kann der Algorithmus zur Erkennung von Objekten in höheren Dimensionen eingesetzt werden, beispielsweise von Ebenen in dreidimensionalen Daten.

In der allgemein übliche Parametrisierung von Geraden im \mathbb{R}^2 über $z = mx + b$ mit den Parametern m (Steigung) und b (Achsenabschnitt) können bekanntlich vertikal verlaufende Geraden nicht dargestellt werden ($m = \infty$). Daher benutzt die Hough-Linienerkennung die Parametrisierung über Magnitude und Winkel der Normalen auf die Gerade, die wir eben bereits im Tangentiallinien-Verfahren verwendet haben:

$$r = x \cos \theta + z \sin \theta \quad (3.7)$$

die, sofern $\sin \theta \neq 0$, äquivalent ist zu

$$z = \frac{r}{\sin \theta} - x \cot \theta \quad (3.8)$$

wobei r der Abstand der Geraden zum Ursprung ist und θ der Winkel der Normalen der Geraden zur x -Achse. Eine Gerade im kartesischen \mathbb{R}^2 korrespondiert somit zu einem Punkt im dualen (r, θ) -Raum (auch *Hough-Raum* genannt).

Sei nun eine Menge von Punkten gegeben. Für jeden Punkt (x_i, z_i) werden alle Geraden, die durch diesen Punkt gehen, also Gleichung (3.7) erfüllen, in dem Hough-Raum eingetragen. Jeder Punkt im Ausgangsraum führt so zu einer (sinusförmigen) Kurve im Hough-Raum gemäß Abbildung 3.5. Liegen mehrere Punkte im Ausgangsraum auf einer Geraden, so schneiden sich die Kurven im Hough-Raum in einem Punkt (r_i, θ_i) , der über (3.8) mit eben jener Gerade korrespondiert. Somit reicht es zur Erkennung von Geraden aus, alle Punkte in den dualen Hough-Raum zu transformieren und nach Schnittpunkten zu suchen, in denen sich möglichst viele Hough-Kurven treffen.

Eine praktische, effiziente Implementation verläuft über den Aufbau eines 2D-Akkumulatorarrays $\mathbf{H} = ((r_i, \theta_i))$, d.h. einer Diskretisierung des Hough-Raums in den beiden Parametern. Der Winkel ist naturgemäß beschränkt, die Distanz r ist auf einen relevanten Suchbereich zu beschränken. Für jeden Messpunkt wird nun die duale Kurvengleichung aufgestellt und der Zähler der entsprechenden Zelle (r_i, θ_i) inkrementiert. In dem so resultierenden Histogramm entspricht ein maximaler Peak einer Geraden mit den meisten „Stimmen“, also der größten Anzahl von Punkten, die approximativ auf dieser Geraden liegen. Die korrespondierenden Punkte werden gelöscht und der nächste Maximalwert im Histogramm wird gesucht, bis keine Punkte mehr

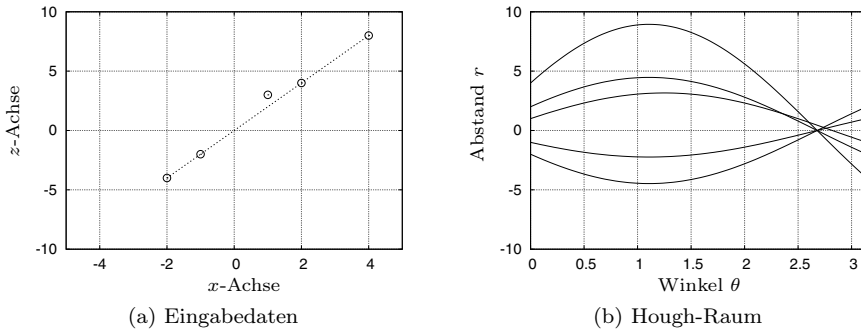


Abbildung 3.5: Abbildung von Punkten zu Kurven im Hough-Raum, schneiden sich, da auf einer Linie liegend. Punkt $(1, 3)$ liegt nicht exakt auf der Linie, und führt somit zu nicht ganz passender oberer Kurve dieses Punktes.

vorhanden sind oder die Höhe der Peaks einen Schwellwert unterschreiten. Abbildung 3.6 skizziert das iterative Vorgehen an einem Beispiel.

Die vergleichsweise grobe Diskretisierung des Hough-Raumes ist nicht nur notwendig, sondern hat auch den positiven Seiteneffekt, dass die Hough-Transformation robust gegen Rauschen in den Messpunkten ist. Andernfalls würden in realen Daten aufgrund von Messungenauigkeiten i.Allg. keine drei Punkte jemals *exakt* auf einer Geraden liegen; ein Histogramm ohne hinreichende Diskretisierung wäre somit vollkommen flach.

Das soweit beschriebene Verfahren liefert eine Liste von erkannten Geraden, geordnet nach ihrer Punktzahl. In einem finalen Schritt müssen nun die Geraden basierend auf den zu ihnen korrespondierenden Punkten zu endlichen Linien begrenzt, sowie ggf. in mehrere Liniensegmente unterteilt werden. Ein exemplarisches Ergebnis ist in Abbildung 3.7 dargestellt.

3.2 Bildmerkmale

Nachfolgend beschreiben wir einige elementare Filter bzw. Merkmalsdetektoren auf Bildern. Die so aggregierten Informationen werden später beispielsweise zur Objekterkennung, Berechnung von Tiefenbildern, Lokalisierung oder Kartierung genutzt. Wenn nicht anders gesagt, setzen wir Grauwertbilder voraus.

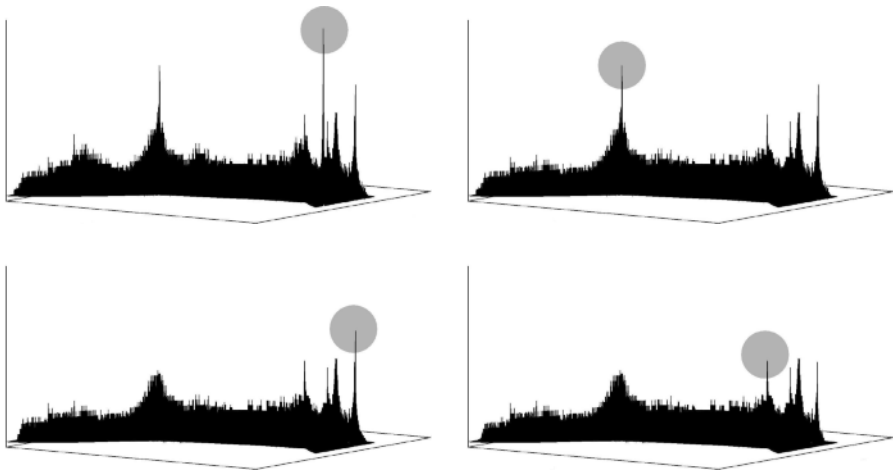


Abbildung 3.6: Hough-Transformation: (r, θ) -Histogramme der ersten vier Schritte der Linienerkennung aus Abbildung 3.7. Die jeweils höchsten Peaks der ersten Linien sind markiert.

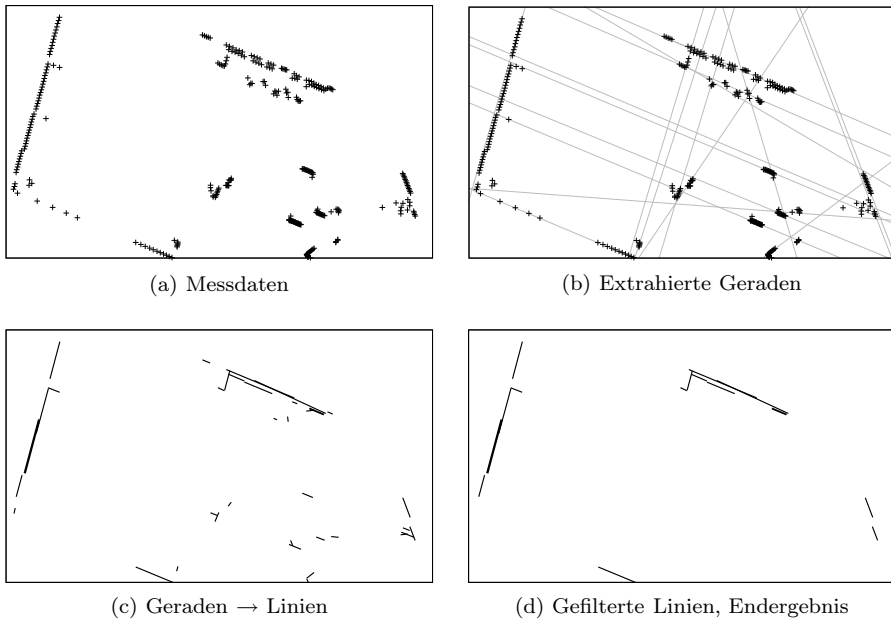


Abbildung 3.7: Hough-Transformation zur Linienerkennung.



Abbildung 3.8: Faltung eines Bildes. Links: Originalbild. Mitte, rechts: Ergebnisse der Faltung mit einem *center-surround* bzw. einem Gauß-Kernel (jeweils oben links angegeben).

3.2.1 Faltungen von Bildern

Filter auf Bildern, also auf Pixelmatrizen, werden meist als pixelweise Multiplikation oder *Faltung* (engl. *convolution*) eines lokalen Bildausschnitts mit einer gleich großen Wertematrix, dem *Kernel* realisiert. Ein Filter kann mehrere Kernel verwenden. Für das Bild $\mathbf{I} = (I(u, v))$ und den Kernel \mathbf{H} der Größe $\delta_i \times \delta_j$ berechne die Faltung $\mathbf{R} = (R(u, v))$ mit:

$$R(u, v) = \sum_{i=-\lfloor \frac{\delta_i}{2} \rfloor}^{\lfloor \frac{\delta_i}{2} \rfloor} \sum_{j=-\lfloor \frac{\delta_j}{2} \rfloor}^{\lfloor \frac{\delta_j}{2} \rfloor} H(i, j) I(i + u, j + v) \quad \text{kurz: } \mathbf{R} = \mathbf{H} * \mathbf{I}, \quad (3.9)$$

wobei $*$ den Faltungsoperator bezeichnet. Üblicherweise sind δ_i und δ_j ungerade.

Im Beispielfall eines Gaußfilters mit Standardabweichung σ wird als Kernel die Funktion

$$H(i, j) = G_\sigma(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \quad (3.10)$$

verwendet (vergleiche dazu Anhang A.2, insbesondere Abbildung A.3, rechts). In der Bildverarbeitung finden Gaußfilter Verwendung als Tiefpassfilter zur Rauschunterdrückung, in der Computergraphik u.a. als Modell für Unschärfe, um einen Tiefenschärfeneffekt zu erzielen. Die Ergebnisse der Faltung mit einem *center-surround*-Kernel sowie einem Gauß-Kernel sind beispielhaft in Abbildung 3.8 zu sehen.

3.2.2 Kantenerkennung

Kanten in Bildern können folgende vier Ursachen haben, die letztlich aus der 3D-Struktur der Welt, aus Oberflächeneigenschaften und aus den Lichtverhältnissen in der Szene resultieren:

- Tiefensprünge (engl. *jump edges*)
- Oberflächendiskontinuitäten (engl. *crease edges*)
- Reflektionsdiskontinuitäten, hervorgerufen z.B. durch Materialwechsel
- Beleuchtungsunterschiede, beispielsweise Licht und Schatten

Kanten in einem Bild tragen oft wichtige Information, zum Beispiel über den Umriss eines Objekts im Bild. Die Filterung eines Bildes, die nur noch die Kanten enthält, enthielte diese Information dann also in komprimierter Form in einem binären Kantenbild, das anschließend weiter verarbeitet werden könnte – z.B. mit der zuvor beschriebenen Hough-Transformation.

Alle Bildkanten haben die Eigenschaft, dass sich der Grauwert in der Region schnell ändert. Fasst man ein Bild als eine Funktion $I(u, v) \rightarrow \mathbb{R}$ auf, müsste Ableiten nach den Bildachsen auf Kantendetektion herauslaufen, denn die erste Ableitung von Funktionen zeigt bekanntlich hohe Werte, wenn die Funktion sich lokal stark ändert. Beide partielle Ableitungen $\partial I(u, v)/\partial u$ und $\partial I(u, v)/\partial v$ können durch diskrete Faltungen realisiert werden. Der nachfolgende Abschnitt zeigt gebräuchliche Kernel und ihre Wirkungen auf Bilder.

Kantenerkennung mit dem Sobelfilter

Der Sobelfilter ist ein einfacher Kantenerkennungs-Filter, der auf einer oder mehreren Faltungen beruht. Er nutzt 3×3 -Kernel, die aus dem Originalbild ein Gradienten-Bild erzeugen.

Nach Tabelle 3.1 finden vier unterschiedliche Kernel Anwendung. Die ersten beiden heben vertikale Kanten abnehmender (Übergang hell-dunkel) bzw. zunehmender Grauwerte (Übergang dunkel-hell) hervor. Die beiden folgenden entsprechend horizontale Kanten.

Abbildung 3.9 (c) zeigt das Ergebnis einer Kombination beider Richtungen: Hier wurden die beiden Ergebnisbilder der abnehmenden Kernel addiert. Ein vollständiges Kantenbild ergibt sich, wenn die Ergebnisbilder aller vier Kernel addiert werden, wie in Abbildung 3.9 (d).

Weitere Kantenerkennungs-Filter

Das Beispiel zum Sobelfilter zeigt, dass beim Ableiten eines Bildes sehr viele Kanten entstehen. Um ihre Anzahl zu reduzieren, verwendet man statt des Originalbildes ein geglättetes. Zum Glätten ist der Gauß-Filter geeignet (vgl. Abbildung 3.8). Da Faltungen assoziativ sind, kann auch die Ableitung der Gaußglättung berechnet und auf das Eingabebild angewendet werden. Solch ein Filter wird auch *Mexican Hat* genannt (vgl. Abbildung 3.10 rechts: Die Funktion in 3D ähnelt einem Sombrero). Er approximiert im Diskreten den

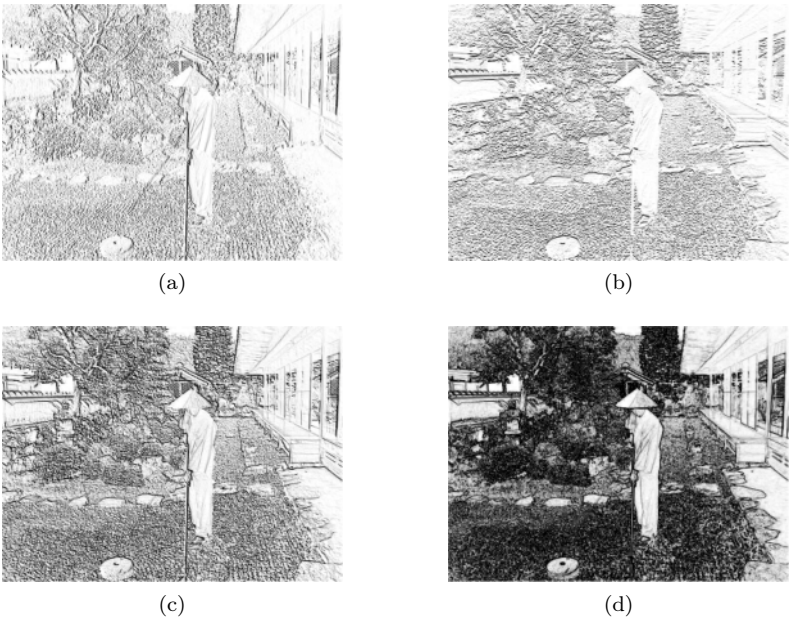


Abbildung 3.9: Anwendung von Sobelfiltern auf ein Beispielbild. (a), (b) Ergebnis des abnehmenden vertikalen resp. horizontalen Kernel. (c) Addition der ersten beiden Bilder. (d) Kombination der Ergebnisbilder aller vier Kernel aus Tabelle 3.1.

Tabelle 3.1: Unterschiedliche Kernel (hier der Größe 3×3) zur Erkennung von Kanten durch einen Sobelfilter.

| H_{SobelV} | H_{SobelV} | H_{SobelH} | H_{SobelH} |
|--|--|--|--|
| $\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$ | $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$ |
| | | | |

Laplace-Operator (Summe der beiden zweiten Ableitungen), der ein weiterer Filter zur Kantendetektion ist:

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} . \tag{3.11}$$

Abbildung 3.11 zeigt einen diskreten Laplace-Kernel und das entsprechende Kantenbild. Abbildung 3.10 zeigt, wie solch ein Laplace-Filter approximiert

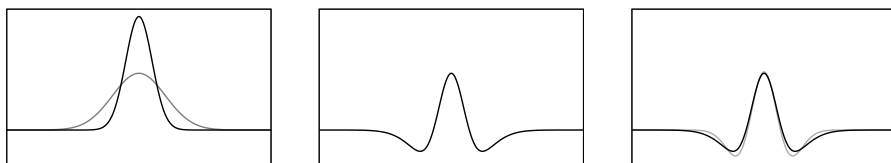


Abbildung 3.10: Links: Zwei Gauß-Verteilungen mit unterschiedlicher Varianz σ_1 (schwarz) und $\sigma_2 > \sigma_1$ (grau). Mitte: Subtraktion der Gauß-Verteilungen, $\mathcal{N}(\mu, \sigma_1) - \mathcal{N}(\mu, \sigma_2)$, ergibt einen DoG-Filter. Rechts: Vergleich des DoG-Filters (schwarz) mit einem Laplace-Filter (grau), der approximiert werden soll.

$$\mathbf{H}_{\text{Laplace}} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

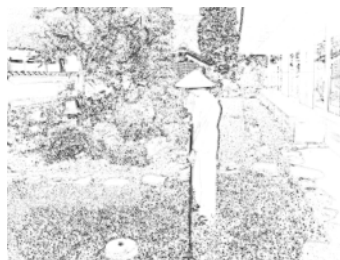


Abbildung 3.11: Ergebnis der Anwendung eines Laplace-Filters. Links: Kernel. Rechts: Gefiltertes Bild. Der Wertebereich des Ergebnisses wurde von $[0, 1, \dots, 254, 255]$ auf $[0, 1, \dots, 254, 255, 254, \dots, 1, 0]$ skaliert, d.h. der mittlere Grauton wird weiß dargestellt.

wird durch Subtraktion zweier Gauß-Verteilungen, bezeichnet als DoG-Filter (engl. *Difference of Gaussians*).

Sobelfilter bzw. *Mexican Hat*-Filter liefern die Grundinformation zur Kantenerkennung in einem Graubild. Wie aus den Bildern in Abbildung 3.9 ersichtlich, lassen sie aber immer noch mehr Kanten übrig, als man zu einer Merkmalerkennung haben möchte. Folglich müssen die resultierenden Kantenbilder nachbearbeitet werden. Die folgenden beiden Schritte sind Beispiele für eine solche Nachbearbeitung:

- Nicht-Maximums-Unterdrückung: Reduziere dünne, aber mehrere Pixel breite Kanten. Ergebnis sind scharfe Kantenlinien mit Breite von einem Pixel.
- Schwellwert und Hysterese: Akzeptiere nur Kantenpixel, die einen Schwellwert t_h übertreffen. Zusätzlich akzeptiere alle Kantenpixel mit niedrigem Schwellwert t_n ($t_n \ll t_h$), wenn diese mit Kantenpixeln verbunden sind, die den hohen Schwellwert übertreffen.

Der *Canny-Algorithmus* ist ein verbreitetes Verfahren, das diese Schritte zur Kantenerkennung benutzt. Abbildung 3.12 zeigt zwei Ergebnisse des Algorithmus nach Canny für das Beispielbild. Die beiden Ergebnisse unterscheiden sich

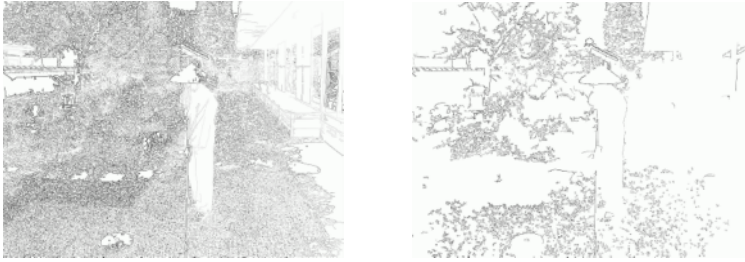


Abbildung 3.12: Ergebnisse einer Canny-Kantenerkennung mit kleinem (links) und großem (rechts) Schwellwert.

durch Verwendung unterschiedlicher Schwellerte t_h, t_n , durch die Einfluss auf die Anzahl der Kanten genommen werden kann.

3.2.3 Eckenerkennung und der Harris Corner Detektor

Wir haben gesehen, dass man mit passenden Kernen Kanten erkennen kann. Eine Kante zeichnet sich dadurch aus, dass sie einen hohen Grauwert-Gradienten in einer Richtung hat; das nutzt man zum Beispiel bei Sobel-Kernen aus. Kanten sind wichtig, um beispielsweise Umrisse von Objekten im Bild zu erkennen. Zur Charakterisierung von Objekten in Bildern sind aber nicht nur glatte Kanten interessant: Wichtiger, weil besonders markant sind Ecken von Objekten, oder generell Regionen im Bild, wo der Grauwert-Gradient in *beiden* Richtungen stark variiert. In solchen Regionen versagen aber reine Kantenfilter kläglich: sie verwaschen Ecken eher, als sie zu markieren.

Der Gedanke, den lokalen Grauwertgradienten in beiden Richtungen zur Bestimmung von „interessanten“ Bildpunkten zu verwenden, kann aber direkt umgesetzt werden. Wenn wir einen solchen Operator auf alle Regionen eines Bilds lokal anwenden, dann würden wir als Ergebnis erwarten:

- Regionen homogener Textur werden nicht als interessant erkannt;
- für alle Punkte auf Grauwertkanten erkennt man Gradienten in einer Richtung, nämlich der Normalen auf die Kante;
- für Punkte in „Ecken“, aber auch für isolierte Punkte mit stark aus ihrer Umgebung herausstehendem Grauwert, erkennt man Gradienten in beide Richtungen.

Ecken werden dabei oft mit interessanten Punkten gleichgesetzt, weil sie besonders charakteristische Merkmale in Bildern darstellen.

Formal kann man diese Idee folgendermaßen fassen. In einem Grauwertbild I betrachten wir eine Region der Größe (u, v) und verschieben sie um (x, y) .

Die gewichtete Summe der quadratischen Grauwertunterschiede dieser beiden Regionen wird mit S bezeichnet und ist durch

$$S_I(x, y) = \sum_u \sum_v w(u, v) (I(u, v) - I(u - x, v - y))^2 \quad (3.12)$$

gegeben. Die Harris-Matrix \mathbf{A} wird durch eine Taylorreihen-Approximation von S gefunden:

$$S_I(x, y) \approx S_I(0, 0) + (x, y) \nabla S_I + \frac{1}{2} (x, y) \mathbf{A} \begin{pmatrix} x \\ y \end{pmatrix}, \quad (3.13)$$

wobei ∇S der Gradientenvektor und die Matrix \mathbf{A} die Hessematrix der zweiten Ableitungen von S_I ist. S_I wird an der Stelle $(x, y) = (0, 0)$ berechnet. Nach der Definition von S_I verschwinden $S_I(0, 0)$ und ∇S_I und es ergibt sich für kleine x und y :

$$S_I(x, y) \approx \frac{1}{2} (x, y) \mathbf{A} \begin{pmatrix} x \\ y \end{pmatrix}. \quad (3.14)$$

S_I ist eine Funktion der Grauwerte von \mathbf{I} . Mit den Ableitungen $\mathbf{I}_x := \partial \mathbf{I} / \partial x$ und $\mathbf{I}_y := \partial \mathbf{I} / \partial y$ schreibt sich die Matrix \mathbf{A} als

$$\mathbf{A} = \sum_u \sum_v w(u, v) \begin{pmatrix} \mathbf{I}_x^2 & \mathbf{I}_x \mathbf{I}_y \\ \mathbf{I}_x \mathbf{I}_y & \mathbf{I}_y^2 \end{pmatrix}. \quad (3.15)$$

Eine Ecke, bzw. ein interessanter Punkt, wird durch eine große Variation von S_I in *beide* Richtungen des Vektors (x, y) charakterisiert. Dies lässt sich durch die Analyse der Eigenwerte λ_1, λ_2 von \mathbf{A} bestimmen. Die Eckenerkennung nach Harris verlangt, wie eingangs intuitiv erklärt, folgende Eigenschaften:

1. Wenn $\lambda_1 \approx 0$ und $\lambda_2 \approx 0$, liegt kein interessanter Punkt vor.
2. Eine Kante liegt vor, wenn $\lambda_1 \approx 0$ und $\lambda_2 = c_1$ und $c_1 \gg 0$ ist.
3. Eine Ecke liegt vor, wenn $\lambda_1 = c_1$, $\lambda_2 = c_2$ und $c_1 \neq c_2$, sowie $c_1 \gg 0$ und $c_2 \gg 0$.

Abbildung 3.13 zeigt ein Ergebnis der Erkennung.

3.2.4 SIFT-Merkmale

SIFT (*Scale-invariant feature transform*) bezeichnet einen Algorithmus zur Detektierung lokaler Merkmale, die größtenteils invariant gegenüber Rotation und Skalierung sind. Diese Eigenschaft lassen die Merkmale in der Robotik

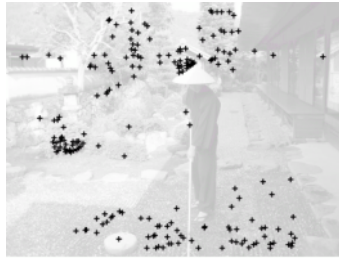


Abbildung 3.13: Ergebnisse einer Harris-Ecken-Detektion.

vielfältige Verwendung finden, beispielsweise zur Objekterkennung, der Berechnung von Tiefenbildern mit Kameras, der Lokalisierung und Kartierung.

Das grundlegende Vorgehen zur Berechnung der Merkmale ist leicht erklärt: In einer Pyramide von gaußverrauschten Versionen des Originalbildes werden Extrema-Pixel detektiert und besonders stabile als Schlüsselpunkte definiert. An allen Schlüsselpunkten werden lokale Intensitätsgradienten (Stärke, Orientierung) berechnet und relativ zu den Gradienten normierte Deskriptoren der Schlüsselpunkt-Umgebungen erzeugt. Dies sind die SIFT-Merkmale. Gaußfaltung und Normierung realisieren eine gewisse Robustheit gegen Variation der Umgebung, wie wechselnde Lichtverhältnisse.

Schließlich sei noch angemerkt, dass die zur Codierung der SIFT-Schlüsselpunkte verwendeten Deskriptoren sich auch als Container für andere Merkmalspunkte anstatt der erwähnten Schlüsselpunkte eignen und entsprechend verwendet werden.

Berechnung der SIFT-Merkmale

Den oben skizzierten vierstufigen Prozess zur Generierung von SIFT-Merkmalen wollen wir nun im Detail beschreiben:

1. **Generierung von Schlüsselpunkten:** In der ersten Stufe werden „interessante“ Stellen, genannt Schlüsselpunkte (engl. *keypoints*) identifiziert. Dazu werden k Kopien des Bildes, jeweils mit einem Gaußfilter fester Größe aber steigender Varianz, also $\sigma_i > \sigma_{i-1}$, gefiltert – alternativ führt eine iterative Faltung mit stets demselben Gaußfilter, jeweils auf das Vorgängerbild angewendet, zu dem gleichen Ergebnis. Alsdann werden adjazente Bilder der gefilterten Folge voneinander subtrahiert. Auf diese Weise wird ein effizienter DOG-Filter implementiert, der nach Abbildung 3.10 einen Laplace-Filter (LOG, engl. *Laplacian of Gaussian*) approximiert und Veränderungen im Gradientenverlauf aufzeigt.

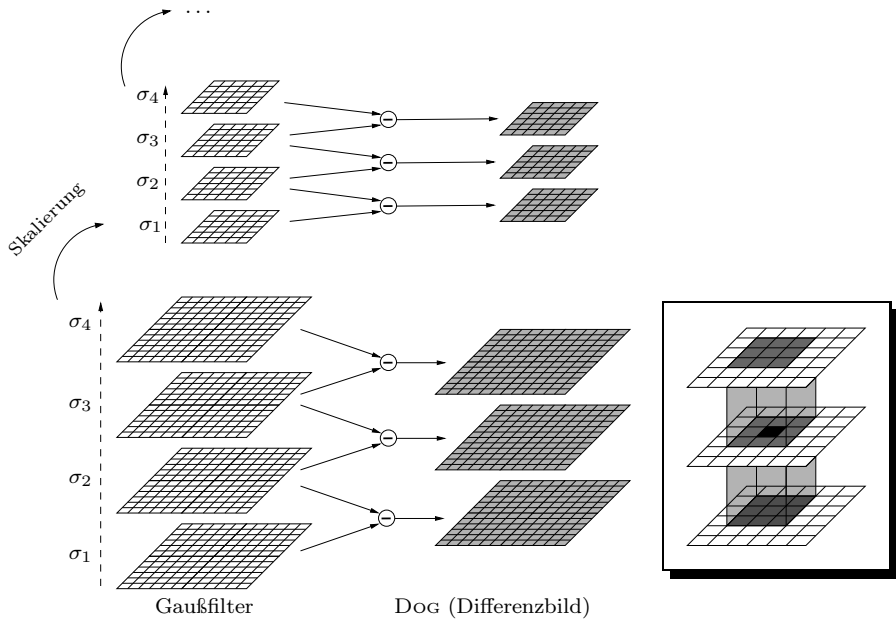


Abbildung 3.14: Links: SIFT-Pyramide zur Schlüsselpunkt-Erkennung. Rechts: Untersuchung der 26-er Nachbarschaft (grau) innerhalb der DOG-Bilderfolge eines Pixels (schwarz). Pixel gelten als extrem, wenn sie maximal/minimal in dieser Nachbarschaft sind. Abbildung erstellt nach [Low04].

Die Serie der gaußgefilterten Bilder sowie der zugehörigen Differenzbilder bilden eine *Epoche*, auch als *Oktave* bezeichnet. Die k Bilder werden nun um den Faktor 2 verkleinert und neue DOG-Differenzbilder generiert. Die auf diese Weise nach mehreren Iterationen gebildete Pyramide (Abbildung 3.14) stellt die Grundlage für die Erkennung von Schlüsselpunkten dar: Dies sind Extrempunkte, also solche Pixel, die in ihrem Wert unter ihren maximal 26 ($= 1 \times 8 + 2 \times 9$) Nachbarpixeln innerhalb der DOG-Pyramide maximal oder minimal sind, vgl. dazu Abbildung 3.14 (rechts).

2. **Selektion der Schlüsselpunkte:** Nur solche Extrema aus Schritt (1) werden weiter als Schlüsselpunkte betrachtet, die a) sich hinreichend stark von ihren Nachbar-DOG-Pixeln unterscheiden, und b) im Bild nicht auf Kontrastkanten liegen. Auf diese Weise wird die Anzahl reduziert, und solche Punkte werden gelöscht, die wenig stabil sind. Das erste Kriterium wird über einen Schwellwert realisiert, den der betragsmäßige Wert des Extremums überschreiten muss; die Überprüfung des zweiten Kriteriums erfolgt über einen direkten Vergleich der umliegenden Pixel.
3. **Berechnung der Schlüsselorientierung:** Zu dem gaußverrauschten Bild $R = G_{\sigma_i} * I$, aus dem ein Schlüsselpunkt stammt, wird ein *Gra-*

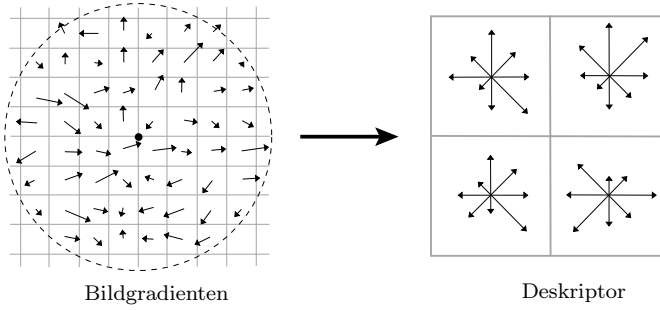


Abbildung 3.15: Erstellung eines SIFT-Deskriptors, hier am Beispiel einer 8×8 -Umgebung um den Schlüsselpunkt, und Generierung eines $2 \times 2 \times 8$ -Deskriptors. Der Kreis symbolisiert die Gaußgewichtung der Gradienteninformationen, die Längen der Pfeile entsprechen den Magnituden der Gradienten in den entsprechenden Richtungen (links), bzw. den summierten Werten der Histogramm-Bins (rechts). Abbildung erstellt nach [Low04].

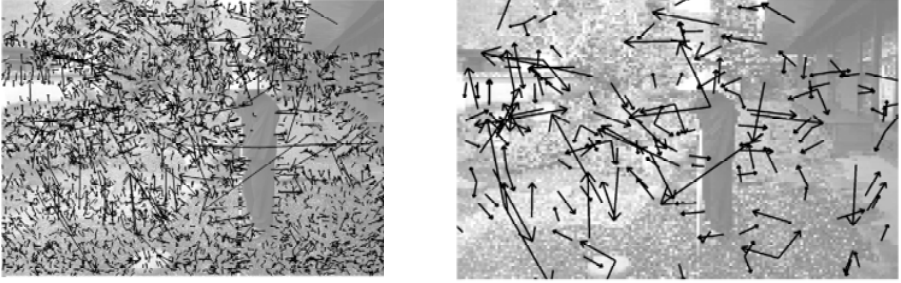


Abbildung 3.16: Beispiel-Ergebnis: SIFT-Deskriptoren auf zwei Beispielbildern. Pfeile geben die Position, Orientierung und Skalierung (= Länge) an. Links: 4805 Schlüsselpunkte in einem Bild der Größe (800×600) . Rechts: Das gleiche Bild, reduziert auf (162×121) Pixel, führt zu 174 Schlüsselpunkten.

dientenbild mit Magnitude Γ und Orientierung Θ berechnet:

$$\Gamma(x, y) = \sqrt{(R(x+1, y) - R(x-1, y))^2 + (R(x, y+1) - R(x, y-1))^2}$$

$$\Theta(x, y) = \text{atan2} \left(\frac{R(x, y+1) - R(x, y-1)}{R(x+1, y) - R(x-1, y)} \right). \quad (3.16)$$

Dabei ist es hinreichend, den Gradienten nur lokal für Punkte innerhalb einer Nachbarschaftsregion der Schlüsselpunkte zu generieren. Alle in dieser Region auf diese Weise berechneten Gradientenpixel werden in einem Winkelhistogramm mit einer Diskretisierung von 10° (also 36 Balken, engl. *bin* oder *bucket*) eingetragen, gewichtet über eine kreisförmige Gaußverteilung – mit einer Varianz abhängig von der Varianz des gaußverrauschten Bildes, aus dem der Schlüsselpunkt stammt, d.h. $\sigma = 1.5\sigma_i$. Diese Ge-

wichtung führt dazu, dass Gradienteninformationen von zu dem Schlüsselpunkt weiter weg liegenden Pixeln weniger Einfluss haben als von näher liegenden Nachbarpixeln.

Peaks in dem so berechneten Winkelhistogramm entsprechen nun Hauptorientierungen des Punktes. Existiert ein eindeutiges Maximum, so wird dieses dem Schlüsselpunkt als Schlüsselorientierung zugeordnet. Im Falle von mehreren, numerisch recht ähnlichen Maxima wird der Schlüsselpunkt vervielfacht und mit den Orientierungen belegt.

4. **Generierung des Deskriptors:** Die bisherigen Schritte haben Informationen über charakteristische Punkte an festen Position im Bild ergeben, in definierter Skalierung und mit zugeordneter Orientierung. Das führt zu Invarianz gegenüber Translation, Skalierung und Rotation. Der letzte Schritt generiert nun möglichst unterschiedliche Deskriptoren und soll darüber hinaus zudem zu Invarianz gegen Beleuchtungsunterschieden und dergleichen führen.

Die zuvor berechnete Schlüsselorientierung definiert für jeden Schlüsselpunkt ein lokales Bezugssystem. In diesem Koordinatensystem werden nun die Gradienten der Umgebungspixel des Schlüsselpunktes berechnet und quadrantenweise gaußgewichtet zu Histogrammen mit acht vergrößerten Gradientenrichtungen zusammengefasst. Das Vorgehen ähnelt somit Schritt (3), die Gradienteninformationen berechnen sich wie oben. Unterschiede liegen in der Diskretisierung in diesmal acht Orientierungen und in der Berechnung von vier getrennten, quadrantenweise aufgestellten Histogrammen. Als Informationen für die Histogramme wird üblicherweise eine Umgebung von 16×16 um den Schlüsselpunkt zu Grunde gelegt, wiederum gaußgewichtet, und pro Quadrant in 2×2 Histogramme (also jeweils nochmals unterteilt) umgerechnet. Somit ergeben sich für jeden Schlüsselpunkt 4×4 Histogramme à 8 Bins, d.h. 128 Parameter. Daneben finden auch andere Konfigurationen wie beispielsweise $2 \times 2 \times 8$ -Histogramme Verwendung. Diese Parameter liefern nun einen z.B. 128-dimensionalen Vektor, den SIFT-Deskriptor des Schlüsselpunktes, der letztlich normalisiert wird, um die Invarianz gegen Beleuchtung zu verbessern. Abbildung 3.15 verdeutlicht diesen Schritt.

Abbildung 3.16 zeigt das Ergebnis der Schlüsselpunkt-Suche am gewohnten Beispiel. In Abbildung 5.19 auf Seite 192 werden wir SIFT-Merkmale zum Matching von Bildern benutzen.

3.2.5 SURF-Merkmale

Eine weitere Methode zur Berechnung von Merkmalen in Bildern, die sich eng an SIFT-Merkmalen orientiert, stellen SURF-Merkmale (engl. *Speeded Up*

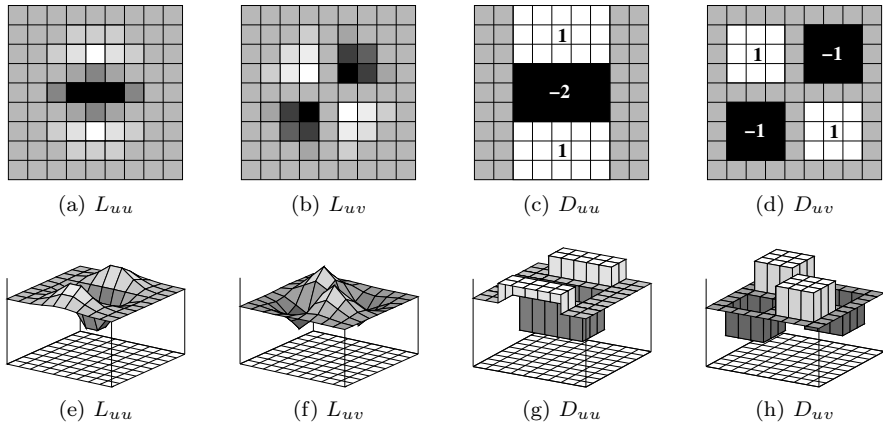


Abbildung 3.17: Gauß- und Box-Kernel, hier für $\sigma = 1.2$. (a) und (b): Diskretisierte Gaußverteilungen zweiter Ordnung in vertikaler bzw. diagonalen Richtung (L_{hh} und L_{hv}). (c) und (d) zeigen die Approximationen über Kastenfilter, D_{vv} und D_{hv} . Abbildungen erstellt nach [BTG06]. (e)–(h): Korrespondierende 3D-Darstellungen.

Robust Features) dar. Eines der Kennzeichen, die Geschwindigkeit in der Berechnung, wird erreicht, indem der DOG-Filter der SIFT-Merkmale wiederum approximiert wird. Die recht grobe Approximation geschieht durch einen Kasten-Filter (DOB, engl. *Difference of Boxes*) DOB. Abbildung 3.17 skizziert, wie für eine feste Varianz σ die DOB-Filter realisiert werden.

Die SURF-Schlüsselpunkte werden berechnet über die Auswertung der Hessematrix an einem Punkt, unter variierender Varianz:

$$\mathcal{H}_\sigma(x, y) = \begin{pmatrix} L_{hh,\sigma}(x, y) & L_{hv,\sigma}(x, y) \\ L_{hv,\sigma}(x, y) & L_{vv,\sigma}(x, y) \end{pmatrix} \quad (3.17)$$

Dabei ist L_{hv} die Filterung des Bildes mit einem LOG-Filter $\frac{\partial^2}{\partial hv} \mathbf{G}$; L_{hh} , L_{vv} sind entsprechend definiert. Wie bereits erwähnt, wird der LOG- durch einen DOB-Filter angenähert, der die Filtermasken D_{hh} , D_{hv} , D_{vv} liefert. Dies führt zu einer approximierten Hessematrix $\mathcal{H}^{\text{apprx}}$, die im Folgenden Grundlage der Schlüsselpunkt-Detektion sein wird.

SURF-Deskriptoren werden analog SIFT-Deskriptoren wie im vorigen Abschnitt berechnet:

1. **Generierung von Schlüsselpunkten:** Potenziell interessante Punkte, also Schlüsselpunkte, werden durch Maxima der Determinante der Hessematrix bestimmt. Diese wird approximiert über

$$\det(\mathcal{H}^{\text{apprx}}) = \mathbf{D}_{hh}\mathbf{D}_{vv} - (0.9 \cdot \mathbf{D}_{hv})^2. \quad (3.18)$$

Die Suche erfolgt nun analog zu der Untersuchung der $3 \times 3 \times 3$ -Nachbarschaft eines Pixels innerhalb der DOG-Pyramide (Abbildung 3.14, rechts). Das Vorgehen unterscheidet sich lediglich dadurch, dass eine Nicht-Maximums-Unterdrückung innerhalb der Pyramide durchgeführt wird, weshalb nachfolgend die tatsächliche Position des Schlüsselpunktes innerhalb eines Bildes und einer Skalierungsstufe interpoliert werden müssen. Details dazu sind in [BL02] beschrieben.

2. **Selektion der Schlüsselpunkte:** Eine explizite Selektion von „guten“ Schlüsselpunkten findet nicht statt, da ein vergleichbarer Schritt implizit aus der Bestimmung der Schlüsselpunkte in (1) durch die Nicht-Maximums-Unterdrückung und Interpolation resultiert.
3. **Berechnung der Schlüsselorientierung:** Um invariant gegen Rotation zu sein, werden den Schlüsselpunkten reproduzierbare Orientierungen zugewiesen. Dazu werden innerhalb eines festen Radius um den Punkt Haar-Merkmale (vgl. Abschnitt 3.2.6) in horizontaler und vertikaler Richtung, also entlang der x - bzw. y -Achse, berechnet. Diese werden dann gaußgewichtet und als 2D-Vektoren in einem Koordinatensystem betrachtet, das aufgespannt wird, indem das Ergebnis des jeweiligen horizontalen Haar-Merkmals, d_h , gegen das des vertikalen (d_v) aufgetragen wird. Das System wird nun diskretisiert und Vektoren, die durch die Diskretisierung zusammen fallen, aufsummiert; der längste so berechnete Vektor definiert nun die Hauptorientierung.
4. **Generierung des Deskriptors:** Zur Berechnung des Deskriptors wird ein rechteckiger Bereich um den Schlüsselpunkt an dessen Hauptorientierung ausgerichtet (sofern Rotationsinvarianz gewünscht), dann in 4×4 Unterregionen aufgeteilt. Diese Unterteilung soll helfen, räumliche Charakteristiken in der Umgebung des Schlüsselpunktes widerzuspiegeln: In jeder Unterregion werden an ausgewählten Stichproben (engl. *samples*) wiederum die Haar-Merkmale (d_h, d_v) berechnet, in Abhängigkeit ihrer Entfernung vom Schlüsselpunkt gaußgewichtet und pro Region zu einem Merkmalsvektor \mathbf{V} gespeichert:

$$\mathbf{V} = \left(\sum d_h, \sum d_v, \sum |d_h|, \sum |d_v| \right)^T. \quad (3.19)$$

Die absoluten Einträge liefern dabei Informationen über die Polarität der Änderungen in der Helligkeit.

Für jede der 4×4 Regionen wird solch ein 4-Vektor berechnet, konkateniert zu einem Deskriptor der Länge 64. Invarianz gegen Beleuchtung entsteht durch die verwendeten Haar-Merkmale, gegen Skalierung durch Normierung des Vektors.

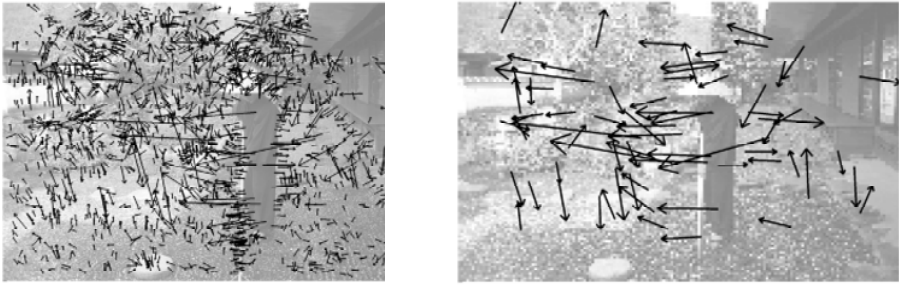


Abbildung 3.18: SURF-Deskriptoren auf dem Beispielbild. Pfeile geben die Position, Orientierung und Skalierung (= Länge) an. Links: 1693 Schlüsselpunkte in einem Bild der Größe (800×600) . Rechts: Das gleiche Bild, reduziert auf (162×121) Pixel, führt zu 71 Schlüsselpunkten.

Das Ergebnis der Berechnung ist am Beispiel in Abbildung 3.18 dargestellt. Abbildung 5.20 (Seite 192) zeigt die Benutzung von SURF-Merkmalen zum Matching von Bildern.

Zu dem oben beschriebenen Vorgehen existieren einige Varianten, zum Beispiel:

SURF-128 gebildet aus Unterregions-Vektoren der Länge 8, indem nach dem Vorzeichen der Haar-Merkmale unterschieden wird: So liefert der Eintrag $\sum d_h$ des 4-Vektors \mathbf{V} nun zwei Einträge, $\sum_{d_v < 0} d_h$, $\sum_{d_v \geq 0} d_h$; für die anderen drei Einträge entsprechend.

U-SURF ist schneller zu berechnen, da die Bestimmung der Hauptorientierung weggelassen wird. Diese Merkmale liefern unter Umständen sogar bessere Ergebnisse, sind allerdings nicht mehr rotationsinvariant.

Ein weiterer Unterschied zu Standard-SIFT-Merkmalen besteht in der Möglichkeit, zu jedem Schlüsselpunkt neben der Determinante der approximierten Hessematrix auch das Vorzeichen der Spur (also der Summe der Diagonalelemente) der Matrix zu speichern. Dies ermöglicht einen performanteren Vergleich von Merkmalen, da nur jene in Betracht gezogen werden müssen, die das gleiche Vorzeichen aufweisen. Dieses Vorgehen ist intuitiv zu motivieren, da die Unterscheidung von hellen Blobs (also Strukturen, die mit den Schlüsselpunkten koinzidieren) auf dunklem Hintergrund, versus dem invertierten Fall, mit einem positiven respektive negativen Vorzeichen korreliert. Ferner wird ein weiterer Geschwindigkeitsvorteil erreicht durch Benutzung von Integralbildern, die im nachfolgenden Abschnitt näher beschrieben werden. Da jedes Pixel eines Integralbilds die Summe der Pixel des Originalbildes innerhalb eines rechteckigen Bereiches bis zu dieser Stelle darstellt, ist die Anwendung des Kasten-Filters sehr effizient und ohne iterative Verkleinerung und Filterung des Bildes implementierbar und somit auch leicht parallelisierbar.

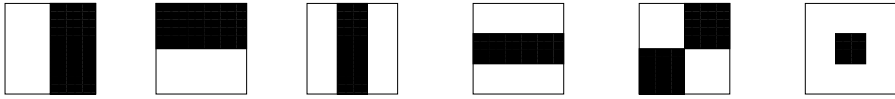


Abbildung 3.19: Haar-Merkmale zur Objekterkennung. Kanten-, Linien-, Diagonal-, und center-surround-Merkmale.

3.2.6 Haar-Merkmale

Die nun vorgestellten Merkmale haben die gleiche Struktur wie die Haar-Funktionen, mit denen Wavelets dargestellt werden können: Gegeben sei $f_{\text{Haar}}: \mathbb{R} \rightarrow [-1, 1]$, mit

$$f(x) = \begin{cases} -1 & 0 \leq x \leq 1/2 \\ 1 & 1/2 \leq x \leq 1 \\ 0 & \text{sonst} . \end{cases} \quad (3.20)$$

Diese Haar-Funktionen sind Schrittfunktionen und werden auch in verwendet. Abbildung 3.19 zeigt die 6 Basisfunktionen, d.h. die Kanten-, Linien-, Diagonal-, und center-surround-Merkmale. In einem Basisdetektor der Größe von z.B. 30×30 Pixel werden alle möglichen Merkmale generiert. Für einen 30×30 Detektor ergeben sich mehr als 180000 Merkmale.

Die Haar-Merkmale lassen sich effizient auswerten, indem Integralbilder verwendet werden (engl. *integral images* oder *summed area tables*). Ein Integralbild \mathcal{I} eines Bildes I ist eine Zwischenrepräsentation für das Bild mit einer Breite x und Höhe y und enthält die Summe der Pixelwerte aus N :

$$\mathcal{I}(x, y) = \sum_{x' \leq x} \sum_{y' \leq y} I(x', y') . \quad (3.21)$$

Das Integralbild wird rekursiv durch folgende Formel bestimmt:

$$\mathcal{I}(x, y) = \mathcal{I}(x, y - 1) + \mathcal{I}(x - 1, y) - \mathcal{I}(x - 1, y - 1) \quad (3.22)$$

mit $\mathcal{I}(-1, y) = \mathcal{I}(x, -1) = \mathcal{I}(-1, -1) = 0$. Demnach benötigt die Berechnung von \mathcal{I} nur einen einzigen Zugriff auf die Eingabedaten. Das Zwischenergebnis Integralbild erlaubt die Berechnung einzelner Rechteckmerkmale der Breite (h, w) an Pixel (x, y) durch vier Referenzen auf das Integralbild (vgl. Abbildung 3.20):

$$\begin{aligned} F(x, y, h, w) &= \mathcal{I}(x, y) + \mathcal{I}(x + w, y + h) \\ &\quad - \mathcal{I}(x, y + h) - \mathcal{I}(x + w, y) . \end{aligned} \quad (3.23)$$

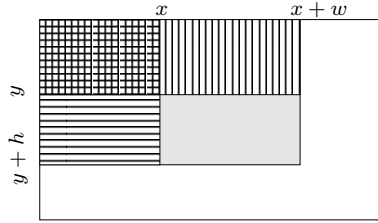


Abbildung 3.20: Die Berechnung der Haar-Merkmalwerte f in der schattierten Region basiert auf den Integralwerten der 4 linken oberen Rechtecke.

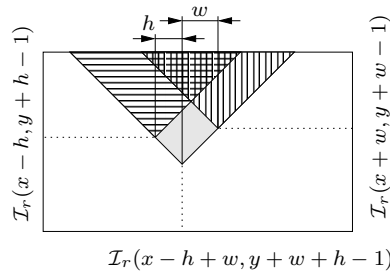


Abbildung 3.21: Die Fläche von rotierten Merkmalen basiert auf den Integralwerten der 4 schraffierten Rechtecke.

Zur Bestimmung von um 45° rotierten Merkmalen wurden rotierte Integralbilder eingeführt. Diese enthalten die Summen über die Pixel eines Rechtecks, das um 45° gedreht wurde. Die untere Spitze befindet sich bei (x, y) (vgl. Abbildung 3.21, rechts):

$$\mathcal{I}_r(x, y) = \sum_{x'=0}^x \sum_{y'=0}^{x-|x'-y|} I(x', y') . \quad (3.24)$$

Da alle Merkmale aus Rechtecken zusammengesetzt sind, lassen sie sich durch das Referenzieren des Integralbildes und gewichtete Subtraktionen bestimmen. Die Gewichtung geschieht dabei proportional zu den Flächen der weißen und schwarzen Bereiche.

Um ein Merkmal f_i zu detektieren, benötigt man einen Schwellwert. Ist die Auswertung eines Merkmals über einen Bildbereich größer als der Schwellwert, gilt das Merkmal als erkannt.

3.3 Stereobildverarbeitung

Abschnitt 2.6.2 hat bereits Stereokameras als Sensoren für Entfernungsinformationen vorgestellt und die beiden zu lösenden Aufgaben erwähnt, nämlich die Kamerakalibrierung und das Lösen des Korrespondenzproblems. In diesem Kapitel beschäftigen wir uns zunächst mit dem Rektifizieren der Kamerabilder, einer speziellen Kalibrierungsmethode, um zu so genannten Orthophotos zu kommen. In solchen Bildpaaren lässt sich das Korrespondenzproblem durch eine Suche entlang einer horizontalen Linie lösen.

Seien zunächst zwei Kameras wie in Abbildung 2.39 gegeben. Die 2D-Bildpunkte der Bilder haben 3D-Koordinaten, also die Raumkoordinaten der CCD-Zellen. Für zwei korrespondierende Punkte $\hat{\mathbf{p}}_l, \hat{\mathbf{p}}_r \in \mathbb{R}^3$ seien die 3D-Koordinaten $\mathbf{p}_l, \mathbf{p}_r \in \mathbb{R}^3$ der Bildpunkte bekannt. Eine Transformation der beiden Kamerakoordinatensysteme ergibt folgende Gleichung:

$$\hat{\mathbf{p}}_r = \mathbf{R}(\hat{\mathbf{p}}_l - \mathbf{t}) , \quad (3.25)$$

wobei die Transformation von einem Kamerakoordinatensystem in das andere mit einer Rotation \mathbf{R} und einer Translation \mathbf{t} beschrieben ist. Die Kamerazentren $\mathbf{o}_l, \mathbf{o}_r$, der Punkt $\hat{\mathbf{p}}$, sowie deren Projektionen liegen in einer Ebene. Daher sind die Vektoren $\mathbf{t}, \hat{\mathbf{p}}_l, \hat{\mathbf{p}}_l - \mathbf{t}$ koplanar und es gilt

$$0 = (\hat{\mathbf{p}}_l - \mathbf{t})^T (\mathbf{t} \times \hat{\mathbf{p}}_l) . \quad (3.26)$$

Einsetzen von (3.25) in (3.26) ergibt

$$\begin{aligned} 0 &= (\mathbf{R}^T \hat{\mathbf{p}}_r)^T (\mathbf{t} \times \hat{\mathbf{p}}_l) \quad \text{bzw.} \\ &= (\mathbf{R}^T \hat{\mathbf{p}}_r)^T \mathbf{T} \hat{\mathbf{p}}_l \\ &= \hat{\mathbf{p}}_r^T \mathbf{R} \mathbf{T} \hat{\mathbf{p}}_l . \end{aligned} \quad (3.27)$$

Dabei wurde das Kreuzprodukt als Matrix-Vektor-Multiplikation geschrieben:

$$\mathbf{t} \times \hat{\mathbf{p}}_l = \mathbf{T} \hat{\mathbf{p}}_l \quad \text{mit} \quad \mathbf{T} = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix} . \quad (3.28)$$

Die Multiplikation $\mathbf{E} = \mathbf{R} \mathbf{T}$ wird *essentielle Matrix* genannt. Sie beinhaltet die gesuchte Rotation und Translation. Für die Bildpunkte \mathbf{p}_l und \mathbf{p}_r kann analog geschlussfolgert werden:

$$\hat{\mathbf{p}}_r^T \mathbf{E} \hat{\mathbf{p}}_l = 0 \quad \Rightarrow \quad \mathbf{p}_r^T \mathbf{E} \mathbf{p}_l = 0 . \quad (3.29)$$

Die essentielle Matrix \mathbf{E} hat spezielle Eigenschaften, die in den folgenden Sätzen ausgedrückt werden können. Die Definition von \mathbf{E} indiziert eine Bedingung in die Bildkoordinaten für einen Punkt \mathbf{x} .

Lemma 2. \mathbf{t}_0 ist der Eigenvektor von \mathbf{E}^T , der mit dem 0-Eigenwert korrespondiert.

Beweis. Sei \mathbf{x} ein beliebiger Punkt. Es gilt für $\hat{\mathbf{x}}_r$ und $\hat{\mathbf{x}}_r$:

$$\begin{aligned} \hat{\mathbf{x}}_r^T \mathbf{E} \hat{\mathbf{x}}_r &= (\hat{\mathbf{x}}_r - \mathbf{t})^T \mathbf{R}^T \mathbf{E}^T \hat{\mathbf{x}}_r && (\text{Nach Definition (3.25)}) \\ &= (\hat{\mathbf{x}}_r - \mathbf{t})^T \mathbf{R}^T \mathbf{R} \mathbf{T} \hat{\mathbf{x}}_r && (\text{Definition der essentiellen Matrix}) \\ &= (\hat{\mathbf{x}}_r - \mathbf{t})^T \mathbf{T} \hat{\mathbf{x}}_r && (\text{für Rotationsmatrizen gilt: } \mathbf{R}^T \mathbf{R} = \mathbf{I}) \\ &= 0 && (\text{für } \mathbf{t} = \mathbf{t}_0) \end{aligned}$$

□

Lemma 3. Die Singulärwertzerlegung von \mathbf{E} ergibt $\mathbf{E} = \mathbf{U} \mathbf{S} \mathbf{V}$ mit den Singulärwerten $\sigma_0 = 1$, $\sigma_1 = 1$ und $\sigma_2 = 0$.

Beweis. Siehe [HF89].

□

Die Singulärwertzerlegung (engl. *Singular Value Decomposition*, *SVD*, vgl. Kapitel A.3) $\mathbf{E} = \mathbf{U} \mathbf{S} \mathbf{V}$ ergibt die orthonormalen Matrizen \mathbf{U} und \mathbf{V} . \mathbf{S} ist eine Diagonalmatrix, die die Singulärwerte σ_i enthält:

$$\mathbf{S} = \begin{pmatrix} \sigma_0 & 0 & 0 \\ 0 & \sigma_1 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix}. \quad (3.30)$$

Lemma 4. Es gilt: $\|\mathbf{E}\|^2 = \text{Spur } \mathbf{E}^T \mathbf{E} = 2$.

Beweis. Siehe [HF89].

□

Leider sind die 3D-Bildkoordinaten $\mathbf{p}_l, \mathbf{p}_r \in \mathbb{R}^3$ in der Regel unbekannt. Nur die 2D-Bildkoordinaten liegen im Bildkoordinatensystem vor: $\bar{\mathbf{p}}_l = (u_l, v_l, 1)^T \in \mathbb{R}^3$ und $\bar{\mathbf{p}}_r = (u_r, v_r, 1)^T \in \mathbb{R}^3$. Der Übergang von der essentiellen Matrix \mathbf{E} zur *fundamentalen Matrix* \mathbf{F} trägt dem Rechnung:

$$\mathbf{p}_r^T \mathbf{E} \mathbf{p}_l = 0 \quad \Rightarrow \quad \bar{\mathbf{p}}_r^T \mathbf{F} \bar{\mathbf{p}}_l = 0. \quad (3.31)$$

Die fundamentale Matrix enthält die Abbildungsfunktion der Kameras (vgl. Formel (2.14)):

$$\mathbf{F} = \mathbf{M}_r^{-1} \mathbf{E} \mathbf{M}_l^{-1} \quad \text{mit} \quad \mathbf{M} = \begin{pmatrix} \alpha_x & 0 & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (3.32)$$

Für die Berechnung der Matrix \mathbf{F} wurde der 8-Punkte-Algorithmus entwickelt. Eingabe sind $n \geq 8$ Punktkorrespondenzen, die in der Minimierung von

$$\operatorname{argmin}_{\mathbf{F}} (\bar{\mathbf{p}}_r \mathbf{F} \bar{\mathbf{p}}_l)^2 \quad (3.33)$$

münden. Gesucht ist hier die Matrix \mathbf{F} , die die Punktabstände minimiert. Das Problem wird auf ein Problem der kleinsten Quadrate zurückgeführt und folgender Term minimiert:

$$\min_{\mathbf{F}} \|\mathbf{A} \mathbf{f}\|. \quad (3.34)$$

Dabei wird die 3×3 Matrix \mathbf{F} als Vektor \mathbf{f} mit 9 Einträgen dargestellt und das Matrix-Vektor-Produkt ausgeschrieben:

$$\begin{aligned} u_{l,1}u_{r,1}f_{1,1} + u_{l,1}v_{r,1}f_{1,2} + u_{l,1}f_{1,3} + v_{l,1}u_{r,1}f_{2,1} + v_{l,1}v_{r,1}f_{2,2} + \\ v_{l,1}f_{2,3} + u_{r,1}f_{3,1} + v_{r,1}f_{3,2} + f_{3,3} = 0. \end{aligned} \quad (3.35)$$

Die Matrix \mathbf{A} ist mit Hilfe aller Punktkorrespondenzen in (3.34) definiert als

$$\mathbf{A} = \begin{pmatrix} u_{l,1}u_{r,1} & u_{l,1}v_{r,1} & u_{l,1} & v_{l,1}u_{r,1} & v_{l,1}v_{r,1} & v_{l,1} & u_{r,1} & v_{r,1} & 1 \\ u_{l,2}u_{r,2} & u_{l,2}v_{r,2} & u_{l,2} & v_{l,2}u_{r,2} & v_{l,2}v_{r,2} & v_{l,2} & u_{r,2} & v_{r,2} & 1 \\ & & & \vdots & & & & & \\ u_{l,n}u_{r,n} & u_{l,n}v_{r,n} & u_{l,n} & v_{l,n}u_{r,n} & v_{l,n}v_{r,n} & v_{l,n} & u_{r,n} & v_{r,n} & 1 \end{pmatrix}. \quad (3.36)$$

Jedes Punktpaar ergibt also eine Gleichung. Das Minimierungsproblem wird mit Hilfe der Singulärwertzerlegung gelöst: $\mathbf{A} = \mathbf{U} \mathbf{S} \mathbf{V}$, wobei \mathbf{U} und \mathbf{V} orthonormale Matrizen sind. \mathbf{S} ist eine Diagonalmatrix, die die Singulärwerte σ_i enthält:

$$\mathbf{S} = \begin{pmatrix} \sigma_0 & 0 & \cdots & 0 \\ 0 & \sigma_1 & \cdots & 0 \\ & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \sigma_9 \end{pmatrix}. \quad (3.37)$$

Die Singulärwerte sind sogar der Größe nach sortiert: $\sigma_0 > \sigma_1 > \cdots > \sigma_9$. Verwendet man genau 8 Punkte, ist σ_9 immer Null, da 8 Gleichungen mit 9

Unbekannten vorliegen. Die mit der SVD bestimmte initiale Schätzung sei die Matrix $\mathbf{F}_{\text{est}} = \mathbf{V}$, die nun nachbearbeitet werden muss, um den Rang 2 zu garantieren (vgl. Lemma 3). Dazu wird wiederum die Singulärwertzerlegung angewandt, diesmal auf die Matrix \mathbf{F}_{est} .

$$\mathbf{F}_{\text{est}} = \mathbf{U}\mathbf{S}\mathbf{V} = \mathbf{U} \begin{pmatrix} \sigma_0 & 0 & 0 \\ 0 & \sigma_1 & 0 \\ 0 & 0 & \sigma_2 \end{pmatrix} \mathbf{V}. \quad (3.38)$$

Der kleinste der drei Singulärwerte wird Null gesetzt ($\sigma_2 = 0$) und die resultierende Matrix als \mathbf{S}' bezeichnet. Letztendlich ergibt sich die fundamentale Matrix als:

$$\mathbf{F} = \mathbf{U}\mathbf{S}'\mathbf{V}. \quad (3.39)$$

Mit Hilfe der Matrix \mathbf{F} lassen sich die Bilder rektifizieren. Abbildungen 2.40 und 3.22 zeigen rektifizierte Bildpaare. In rektifizierten Bildern befinden sich die Korrespondenzen auf den horizontalen Scanlinien. Leider sind so die Korrespondenzen nicht eindeutig bestimmbar. Zu einem Pixel im linken Bild gibt es viele im rechten, die den gleichen oder einen ähnlichen Grauwert aufweisen. Da die Kameras die Szene auch noch aus unterschiedlichen Perspektiven wahrnehmen, sind die Helligkeiten in beiden Bildern unterschiedlich. Aus diesen Gründen werden keine Pixelkorrespondenzen bestimmt, sondern Bildbereiche verglichen (vgl. Abbildung 3.22). Für den Vergleich solcher Bildbereiche wurden unterschiedliche Ähnlichkeitsfunktionen entwickelt, beispielsweise

$$\sum_{[i,j] \in R} (f(i,j) - g(i,j))^2 \quad (3.40)$$

oder

$$\sum_{[i,j] \in R} f(i,j)g(i,j). \quad (3.41)$$

Hier wurden die Bilder mit f und g bezeichnet. Der Term (3.40) ist die Summe der quadratischen Abstände (vgl. auch (3.12)) und muss minimiert werden. Sind die Bildregionen identisch, ist der numerische Wert 0. Der Term (3.41) wird als Kreuzkorrelation bezeichnet und für gleiche Bildregionen zeigt sie ein Maximum, wobei helle Regionen bevorzugt werden.

Abbildung 3.23 zeigt zwei Ergebnisse der Disparitätsbestimmung. Der Vergleich von Bildregionen zur Bestimmung von Disparitäten ist lediglich eine Approximation, da unter Umständen Welpunkte nur von einer Kamera aufgenommen werden und keinen korrespondierenden Punkt im zweiten Bild haben.



Abbildung 3.22: Rektifizierte Stereobilder erlauben es, korrespondierende Punkte entlang einer horizontalen Linie zu finden. Da viele Pixelwerte auf der Linie (a) gleich sind, verwendet man einen Ausschnittsvergleich (b).

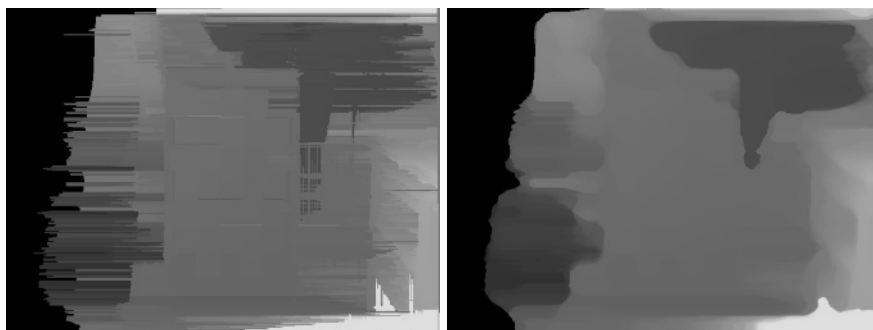


Abbildung 3.23: Disparitätsbilder gewonnen aus den rektifizierten Bildern der Abbildung 3.22: Links: Ergebnisse bei der Verwendung eines kleinen Bildausschnitts. Rechts: Großes Vergleichsfenster.

Neben dem Fenstervergleich gibt es noch merkmalsbasierte Verfahren, die korrespondierende Merkmale finden, beispielsweise SIFT-Merkmale (siehe Abschnitt 3.2.4). Hier lässt sich allerdings die Tiefe, bzw. die 3D-Information nur an den Merkmalspunkten berechnen und es entstehen nicht-dichte Sensoreindrücke.

3.4 Optischer Fluss und Struktur aus Bewegung

3.4.1 Optischer Fluss

Als optischen Fluss (engl. *optical flow*) bezeichnet man in der Bildverarbeitung ein Vektorfeld, das die 2D-Bewegungsrichtung und -Geschwindigkeit für jeden Bildpunkt einer Bildsequenz angibt. Der optische Fluss wird von den auf die Bildebene projizierten Geschwindigkeitsvektoren von sichtbaren Objekten



Abbildung 3.24: Optischer Fluss, wie er sich bei einer Autofahrt darstellt. An die Bildpunkte, von denen der Fluss berechnet werden konnte, wurde ein Pfeil gezeichnet, der die Bewegungsrichtung zwischen zwei Bildern angibt. Die Kamera bewegt sich in der Szene nach links. Die dargestellten Autos bewegen sich nach vorne rechts.

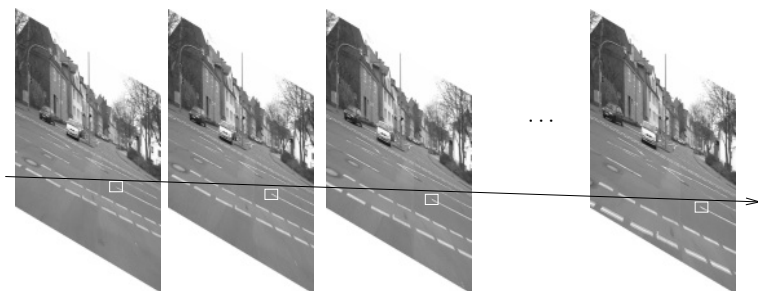


Abbildung 3.25: Ein Punkt weist in einer Sequenz von Bildern Helligkeitskonstanz, räumliche Kohärenz und zeitliche Persistenz auf.

gebildet. Abbildung 3.24 gibt ein Beispiel. Die Berechnung des optischen Flusses macht folgende drei Annahmen (vgl. Abbildung 3.25):

Helligkeitskonstanz besagt, dass die durch die Kamera gemessene Helligkeit einer Region zwischen zwei Bildern in etwa konstant bleibt.

Räumliche Kohärenz bedeutet, dass benachbarte Punkte in einer Szene typischerweise zu der gleichen Oberfläche gehören und daher ähnliche räumliche Bewegungen zeigen. Dies spiegelt sich auch in den Bildpunkten wider.

Zeitliche Persistenz ist die Annahme, dass sich die Bewegung eines Bildpunktes nur allmählich über die Zeit ändert.

Seien $I(x, y, t)$ die Bildpunkte des Bildes I zum Zeitpunkt t , und u die horizontale, bzw. v die vertikale Geschwindigkeit eines Bildpunktes. Unter der Helligkeitskonstanzannahme gilt:

$$I(x + u, y + v, t + 1) = I(x, y, t) . \quad (3.42)$$

Da sich die Helligkeit nicht verändert, ist die Ableitung des Terms nach der Zeit gleich Null

$$\left. \frac{\partial I}{\partial x} \right|_t \left(\frac{\partial x}{\partial t} \right) + \left. \frac{\partial I}{\partial y} \right|_t \left(\frac{\partial y}{\partial t} \right) + \left. \frac{\partial I}{\partial t} \right|_{x(t)} = 0 \quad (3.43)$$

und somit

$$\left. \frac{\partial I}{\partial x} \right|_t u + \left. \frac{\partial I}{\partial y} \right|_t v + \left. \frac{\partial I}{\partial t} \right|_{x(t)} = 0 \quad (3.44)$$

$$\nabla I^T \mathbf{u} = -\mathbf{I}_t \quad (3.45)$$

mit $\mathbf{u} = \begin{pmatrix} u \\ v \end{pmatrix}$ und $\nabla I = \begin{pmatrix} I_x \\ I_y \end{pmatrix}$.

Gleichung (3.44) bzw. (3.45) wurde für den Bildpunkt (x, y) aufgestellt und enthält zwei unbekannte Variablen (u, v) . Da dadurch keine eindeutige Lösung bestimmt ist, bedient man sich eines Tricks: Statt nur einen Bildpunkt zu betrachten, wird eine Region betrachtet. Die Annahme von räumlicher Kohärenz rechtfertigt dieses Vorgehen. Betrachtet man beispielsweise eine Bildregion der Größe 5×5 mit den Pixeln $\mathbf{p}_1, \dots, \mathbf{p}_{25}$, ergeben sich 25 Gleichungen:

$$\nabla I^T(\mathbf{p}_i) \mathbf{u} = -\mathbf{I}_t(\mathbf{p}_i) . \quad (3.46)$$

Umschreiben ergibt

$$\underbrace{\begin{pmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} u \\ v \end{pmatrix}}_{\mathbf{u}} = - \underbrace{\begin{pmatrix} I_t(\mathbf{p}_1) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{pmatrix}}_{\mathbf{b}} . \quad (3.47)$$

Nun haben wir mehr Gleichungen als Unbekannte und können (3.47) als Optimierungsproblem auffassen und $\|\mathbf{A}\mathbf{u} - \mathbf{b}\|^2$ minimieren. Die Lösung geschieht mit Hilfe der Methode der kleinsten Quadrate (vgl. Anhang A.3):

$$\begin{aligned} (\mathbf{A}^T \mathbf{A}) \mathbf{u} &= \mathbf{A}^T \mathbf{b} \\ \left(\begin{pmatrix} \sum_{i=1}^{25} I_x(\mathbf{p}_i) I_x(\mathbf{p}_i) & \sum_{i=1}^{25} I_x(\mathbf{p}_i) I_y(\mathbf{p}_i) \\ \sum_{i=1}^{25} I_y(\mathbf{p}_i) I_x(\mathbf{p}_i) & \sum_{i=1}^{25} I_y(\mathbf{p}_i) I_y(\mathbf{p}_i) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \right) &= - \begin{pmatrix} \sum_{i=1}^{25} I_x(\mathbf{p}_i) I_t(\mathbf{p}_i) \\ \sum_{i=1}^{25} I_y(\mathbf{p}_i) I_t(\mathbf{p}_i) \end{pmatrix} . \end{aligned} \quad (3.48)$$

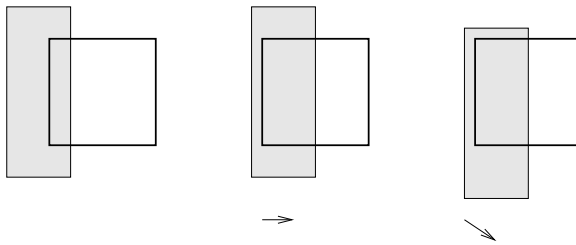


Abbildung 3.26: Das Apertur-Problem. Links: Ausgangsbild. Mitte: Bewegung des grauen Streifens nach rechts. Rechts: Bewegung diagonal von links-oben nach rechts-unten. Lokal sind beide Bewegungen nicht zu unterscheiden.

Die Matrix auf der linken Seite hat die gleiche Form wie die Matrix, mit der Ecken nach Harris berechnet wurden (vgl. Gleichung (3.15)). Demnach können wir Bildregionen analysieren und feststellen, ob sie sich für die Berechnung des optischen Flusses eignen. Der Lucas/Kanade-Algorithmus definiert eine Reihe von Kriterien, die an $\mathbf{A}^T \mathbf{A}$ gestellt werden, um gute Bereiche zu identifizieren. Ecken eignen sich besonders gut für die Berechnung des optischen Flusses (vgl. Abbildung 3.24). Interessanterweise benötigt diese Analyse nur ein Bild der Sequenz.

Durch Verwenden von Bildregionen statt einzelner Pixel tritt das *Apertur-Problem* zu Tage. Es ist eine Konsequenz aus der Mehrdeutigkeit der eindimensionalen Bewegung eines einfachen Musters in einer Region. In Abbildung 3.26 ist ein solcher Fall dargestellt. Anhand der Bildregion ist nicht klar, ob eine horizontale Bewegung oder eine Diagonalebewegung stattgefunden hat.

Der optische Fluss nach Lucas/Kanade wird auf eine Bildpyramide angewendet, um sicherzustellen, dass die Bewegung zwischen den Bildern der Sequenz klein genug ist. Dies garantiert bessere Ergebnisse. Neben dem Lucas/Kanade-Fluss existieren weitere Methoden, den optischen Fluss zu berechnen. Die bekannteste weitere Methode ist der Horn/Schunk-Algorithmus, der ein dichtes Feld von Geschwindigkeitsvektoren zu berechnen versucht.

3.4.2 Struktur aus Bewegung

Ist eine Sequenz von Bildern gegeben, lassen sich in der Sequenz Merkmale extrahieren und verfolgen, bzw. lässt sich der optische Fluss ausrechnen. Aus korrespondierenden Merkmalen sollten sich sowohl die Sensorbewegung als auch die 3D-Struktur der Szene, d.h. die 3D-Koordinaten der Punkte, ausrechnen lassen. Liegen zwei Bilder vor, kann analog des Abschnitts 3.3 (Stereobildverarbeitung) gerechnet werden. Die zweite Kamera ist nicht fest mit der ersten verbunden, sondern wird durch nachfolgende Bilder einer ein-

zigen Kamera ersetzt. Die *Baseline* ist die Verschiebung der Kamera während der Bewegung.

Um die Kamerabewegung aus den Merkmalen zu schätzen, kommt der in Abschnitt 3.3 vorgestellte 8-Punkte-Algorithmus zum Einsatz. Algorithmen, die die Struktur aus Bewegung (engl. *structure from motion*) berechnen, nehmen mehr als zwei Bilder auf, in denen jeweils die Merkmale extrahiert und zugeordnet werden. Angenommen, m Bilder werden aufgenommen, aus denen n Merkmale extrahiert wurden. Somit müssen $6m + 3n$ Unbekannte bestimmt werden: 3 für die Rotation und 3 für die Translation jeder Kamera, die das Bild erzeugt hat, sowie 3 für jeden Merkmalspunkt. Durch einfache Überlegungen lässt sich die Anzahl der Unbekannten reduzieren, jedoch vereinfacht sich das Problem nicht wesentlich. Auch kann angemerkt werden, dass es zur Zeit keine geschlossene Lösung für das Problem „Struktur aus Bewegung“ gibt. Zwar lassen sich Fehlerfunktionen aufstellen und mit Gradientenabstiegsverfahren lösen, doch oftmals konvergieren diese Ansätze nicht oder zu einem falschen Minimum hin.

Auch wurde vorgeschlagen, die perspektivische Abbildung an der Kamera näherungsweise durch Parallelprojektion zu ersetzen. Zusätzlich lässt sich die euklidische Geometrie durch affine Geometrie ersetzen. Damit kann man das „Struktur aus Bewegung“-Problem näherungsweise geschlossen lösen. Diese Lösung wird anschließend als Startwert für einen Gradientenabstieg einer Fehlerfunktion verwendet, die die eigentliche Abbildungsvorschrift beinhaltet. Dies führt häufig zur korrekten Lösung des Problems.

Bemerkungen zur Literatur

Die Linienerkennung über Tangentiallinien wurde von Lu/Milios in [LM97b] eingeführt. Einen Überblick über die Hough-Transformation sowie unterschiedliche Erweiterungen liefert die Arbeit von Kälviäinen [Käl94]. Das Java-Applet unter [Tec] demonstriert recht intuitiv die Vorgehensweise.

Als weiterführende Lektüre zum Thema Bildverarbeitung im Allgemeinen sei [FP02] empfohlen. Dort finden sich auch ausführliche Darstellungen der üblichen Bildfilter.

Ausführlichere Darstellung zu SIFT-Merkmalen sind in [Low04,SLL02,Low99] zu finden. Details der Berechnung von SURF-Merkmalen sind in [BTG06] beschrieben. [VL07] liefert qualitative und quantitative Vergleiche der beiden Algorithmen sowie unterschiedlicher Parameterkonfigurationen.

Haar-Merkmale basieren auf Funktionen, die von Alfred Haar im Jahr 1910 eingeführt wurden [Haa10]. In der Bildverarbeitung wurden diese Schritt-

funktionen beispielsweise von [LM02, POP98, VJ04] erfolgreich eingesetzt. Dabei dienen sie zur Erzeugung von Haar-Merkmalen. [LM02] gibt Details zur Anzahlberechnung von Haar-Merkmalen innerhalb eines Detektors. Haar-Merkmale eignen sich besonders im Zusammenspiel mit Integralbildern [VJ01, VJ04], so dass sogar rotierte Integralbilder eingeführt wurden [LM02].

Das vorgestellte Verfahren zur Berechnung des optischen Flusses geht auf Lucas und Kanade zurück, die die angegebenen Gleichungen 1981 aufstellten [LK81]. Zeitgleich wurde auch der Horn/Schunk-Algorithmus entwickelt [HS81].

Das „Struktur aus Bewegung“-Problem ist sehr ähnlich zu GraphSLAM, das wir in Abschnitt 6.4.1 behandeln. In der Bildverarbeitung wird es auch häufig Bündelblockausgleichung (engl. *bundle adjustment*) genannt. Die zu Grunde liegenden Optimierungslösungen werden vollständig in [TMHF00] angegeben. Das „Struktur aus Bewegung“-Problem erweitert die Bündelblockausgleichung dahingehend, dass die Bildmerkmale und Korrespondenzen automatisch bestimmt werden. Um eine gute Initialschätzung für das „Struktur aus Bewegung“-Problem zu erzeugen, haben Tomasi und Kanade vorgeschlagen, die perspektivische Abbildung an der Kamera näherungsweise durch Parallelprojektion zu ersetzen [TK92]. Aktuelle Software arbeitet vollständig automatisch [SSS06, SSS08].

Aufgaben

Für die folgenden Aufgaben benötigen Sie Sensordaten. Diese können natürlich mit eigener Hardware aufgenommen werden. Alternativ stehen auf <http://www.mobile-roboter-dasbuch.de> Daten zur Bearbeitung der Aufgaben bereit. Darüber hinaus finden Sie dort auch eine Anleitung, um eine eigene Simulationsumgebung aufzubauen.

Die Programme der Übungen 3.1–3.3 sollen als Eingabe Laserscan-Daten, also diskrete Messpunkte, erwarten. Die weiteren Aufgaben arbeiten auf Kamera-Daten.

Übung 3.1. Implementieren Sie einen Median-Filter der Größe k auf Laserscan-Daten. Der Wert von k soll variabel und spezifizierbar sein. Sie werden (insbesondere bei größerem k) dabei bemerken, dass sich die Daten auch an Stellen verändern, die keine offensichtlichen Messfehler darstellen. Welche Möglichkeiten gibt es, dies zu unterbinden?

Übung 3.2. Implementieren Sie einen Reduktionsfilter und wenden ihn auf Laserscan-Daten an.

Übung 3.3. Implementieren Sie den Online-Algorithmus zur Erkennung von Linien. Welche Verbesserungsmöglichkeiten sehen Sie? Erweitern Sie Ihr Programm entsprechend und diskutieren die Ergebnisse.

Übung 3.4. Schreiben Sie ein Programm, das ein Eingabebild mit einem 2D-Kernel filtert. Das Programm soll als Eingabe das Ausgangsbild, den Namen des gefilterten Bildes, sowie eine Datei erhalten, in der der Kernel – mit variabler Größe – spezifiziert wird.

Übung 3.5. Von welchen der in Abbildung 3.19 präsentierten Haar-Merkmalen kann es um 45° rotierte Haar-Merkmale geben?

Übung 3.6. Verwenden Sie Ihr in Aufgabe 2.8 kalibriertes Stereokamerasystem, um Disparitätsbilder zu erstellen. Tipp: OPENCV stellt dafür ebenfalls Funktionen zur Verfügung.

Übung 3.7. Schreiben Sie ein OPENCV-Programm, das in einem Datenstrom gute Merkmale für den optischen Fluss berechnet. Zeichnen Sie den Fluss in die Bildsequenz ein und zeigen Sie ihn auf dem Bildschirm an.

Mobile Roboter

Eine Einführung aus Sicht der Informatik

Hertzberg, J.; Lingemann, K.; Nüchter, A.

2012, X, 390 S. 189 Abb., Softcover

ISBN: 978-3-642-01725-4