Proseminar Künstliche Intelligenz: Wahrnehmung Sommersemester 2011

Stefan Profanter (profante@in.tum.de)

Technische Universität München, Computer Science Department
Intelligent Autonomous Systems Group
Abgegeben bei: Lars Kunze
Datum: 10.04.2011

Zusammenfassung. Diese Ausarbeitung soll einen Überblick darüber geben, wie Computer, vor allem aber Roboter, die physische Welt in deren Umgebung wahrnehmen können. Der wichtigste Sinn, der Gesichtssinn, wird dabei am Ausführlichsten besprochen. Einbezogen werden auch aktuelle Forschungsergebnisse im Bereich der Wahrnehmung. Als Leitfaden dient das Buch aus [5]. Ein Anwendungsbeispiel für die Wahrnehmung und Navigation wird unter Verwendung von SLAM vorgestellt.

1 Einführung

Unsere Umwelt liefert eine nahezu unendliche Anzahl an Informationen. Damit sich ein Roboter in dieser Umgebung bewegen und Objekte manipulieren kann, muss dieser die Informationen aufnehmen und verarbeiten können.
Was bedeutet aber der Begriff Wahrnehmung?
Damit ein Computer etwas wahrnehmen kann, benötigt dieser Sensoren, die den Zustand der Umgebung messen und den Zustand als digitale oder analoge Werte liefern.

1.1 Sensoren

Die wichtigsten Sensoren, um sich in einer Umgebung zu bewegen, sind Entfernungssensoren. Diese arbeiten meist nach dem selben Prinzip: ein Signal wird ausgestrahlt, von einem Objekt in der Umgebung reflektiert und wieder an den Sensor zurückgestrahlt. Durch Messung der Zeit vom Ausstrahlen bis zum Empfangen des Signals kann die Entfernung zum Objekt bestimmt werden.
Lasersensoren verwenden einen gebündelten Lichstrahl. Dieser Lichtstrahl wird meist mit 1 Grad Abstand in einem Winkel von 180 Grad ausgestrahlt. Dabei können unterschiedliche Mesprinzipien zur Anwendung kommen: kurze Lichtimpulse werden ausgesandt und die Lichtlaufzeit gemessen oder eine Pulsfolge mit fester Frequenz wird ausgestrahlt und die Reflexion gemessen. Zudem gibt die

Phasenverschiebung zwischen ausgesendeter und empfangener Pulsfolge ebenfalls Auskunft über die Entfernung.
Sonarsensoren senden Ultraschallwellen in Form eines Kegels aus und messen dann die reflektierten Wellen. Dies deckt normalerweise einen kleineren Winkel als bei Lasersensoren ab.
Ein weiterer wichtiger Sensor ist die **Kamera**. Diese dient zum Erkennen von Bewegungen und bestimmten Objekten in der Umgebung. Ein Computer mit Entfernungssensor kann z.B. erkennen, dass sich vor Ihm ein Buch befindet. Durch Hinzunahme des Kamerabildes kann aber zusätzlich bestimmt werden, um welches Buch es sich handelt.
Um mit Menschen zu interagieren wird zudem ein Mikrofon benötigt, um z.B. Sprachbefehle zu empfangen und zu interpretieren.
Im folgenden Kapitel wollen wir uns damit beschäftigen, wie ein Programm mit Hilfe der Entfernungsdaten und des Kamerabildes Objekte und Bewegungen erkennen kann.

2 Bildverarbeitung

Licht besteht aus verschiedenen Wellenlängen, wobei sich das sichtbare Licht im Bereich von 380nm (Violett) bis 780nm (Rot) befindet. Jede Wellenlänge besitzt einen Intensitätswert. Aus der Kombination der Wellenlänge und der Intensität ergibt sich somit die Farbe. Experimente von Thomas Young (1801) haben gezeigt, dass jede Kombination von Wellenlängen durch eine Mischung aus nur drei Hauptfarben nachgebildet werden kann [5]. In der Regel verwendet man die Wellenlängen 700nm (Rot), 546nm (Grün) und 436nm (Blau). Dies bedeutet, dass ein Computer einen Farbwert durch Abspeichern dieser drei Intensitätswerte darstellen kann. Üblicherweise wird, für die meist ausreichende Genauigkeit, ein Byte pro Intensitätswert verwendet. Somit sind für die Abspeicherung eines Bildpixels 3 Byte erforderlich.

2.1 Kantenerkennung

Kanten grenzen meist unterschiedliche Objekte in einer Szene ab. Deshalb ist es für die Objekterkennung sehr hilfreich, die Kanten in einem Bild zu finden. Zudem geben Kanten Auskunft darüber, wie das Objekt im Raum liegt.
Kanten entsprechen den Positionen im Bild, wo die Helligkeit eine scharfe Änderung erfährt. Durch Differenzieren können diese Positionen bestimmt werden (siehe Abb. 1). Dabei sollte eine Spitze der Ableitung einer Kante entsprechen. Wie man aber erkennen kann, ist im zweiten Diagramm bei ca. x=75 aufgrund von Bildrauschen eine zweite Spitze. Dieses Rauschen kann reduziert werden, indem man das Bild vorher glättet. Eine einfache Möglichkeit zum Glätten ist, jedem Pixel den Durchschnittswert seiner Nachbarpixel zuzuweisen.
Aber wie viele Nachbarpixel sollen dafür verwendet werden? Die ein Pixel entfernten oder zwei oder mehr? Am Besten kann Gaußsches Rauschen mit dem

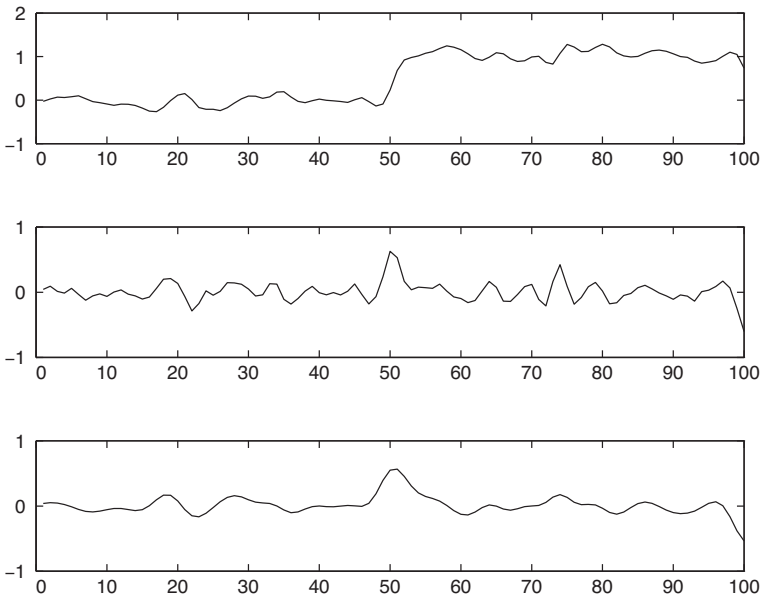


Abb. 1. Erstes Diagramm: Intensitätsprofil entlang eines eindimensionalen Querschnittes. Zweites Diagramm: Ableitung des Intensitätsprofiles. Drittes Diagramm: Ableitung nach Glättung. Quelle: [5]

Gaußschen Filter entfernt werden. Die Gaußsche Funktion in zweiter Dimension sieht wie folgt aus:

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

Die Anwendung eines Gaußschen Filters bedeutet, dass die Intesität I beim Pixel (x_0, y_0) durch die Summe aller (x, y)-Pixel über $I(x, y)G_0(d1, d2)$ ersetzt wird, wobei $d1 = x_0 - x$ und $d2 = y_0 - y$. Diese Art der gewichteten Summe wird auch **Faltung** oder Konvolution genannt:

$$I(x,y) = I(u,v) * G_\sigma(x-u, y-v) := \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} I(u,v) G_\sigma(x-u, y-v). \quad (2)$$

Das geglättete Bild wird also durch Faltung des Bildes mit der Gaußschen Funktion erzeugt. σ ist dabei die Anzahl der Nachbar-Pixel, wobei man bereits mit einem Pixel ein geringes Rauschen entfernen kann. Differenziert man nun das geglättete Bild, erhält man das dritte Diagramm. Aus dem dritten Diagramm kann man nun die Position der Kante (bei $x = 50$) bestimmen.

2.2 3D-Informationen extrahieren

Um Objekte zu manipulieren bzw. in der Umgebung zu navigieren, muss man aus dem 2D-Bild der Kamera dreidimensionale Informationen extrahieren. Somit

kann man die Position und Lage der Objekte im Raum erkennen. Die Objekterkennung besteht aus drei Schritten: Zerlegung der Szene in verschiedene Objekte, Ermittlung der Position und Ausrichtung (Pose) und Ermittlung des Umrisses jedes Objektes.
Es gibt mehrere Hinweise, die uns helfen, die Eigenschaften eines Objektes zu bestimmen:

Bewegung. Kameras nehmen in der Regel 30 Einzelbilder pro Sekunde auf. Bewegt sich die Kamera relativ zur Szene oder ein Objekt in der Szene, entstehen Unterschiede zwischen den Einzelbildern. Diese sichtbare Bewegung wird als **optischer Fluss** bezeichnet. Dabei wird die Richtung und die Geschwindigkeit als Vektor dargestellt (Siehe Abb. 2. [1]).

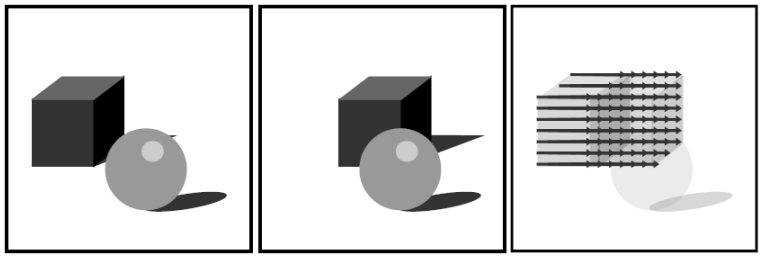


Abb. 2. Frame 1, Frame 2 und der dazugehörige optische Fluss. Quelle: [2]

Einen Flussvektor kann man bestimmen indem man die Tatsache ausnützt, dass Bildbereiche um die entsprechenden Punkte ähnliche Intensitätsmuster aufweisen. Das Pixel p befindet sich zum Zeitpunkt t_0 an der Position (x_0, y_0). Dieses Pixel ergibt mit seinen Nachbar-Pixeln einen Pixelblock. Dieser Pixelblock muss nun im neuen Bild mit den Pixelblöcken um verschiedene Kandidatenpixel q_i an der Stelle $(x_0 + D_x, y_0 + D_y)$ zum Zeitpunkt $t_0 + D_t$ verglichen werden, um die neue Position des Pixels p zu finden. Ein mögliches Ähnlichkeitsmaß ist die Summe der quadrierten Differenzen (SQD, Engl: sum of squared differences):

$$SQD(D_x, D_y) = \sum_{(x,y)} (I(x, y, t) - I(x + D_x, y + D_y, t + D_t))^2. \qquad (3)$$

Die Angabe (x,y) bei der Summe sind dabei die Pixel im Block (x_0, y_0). Hat man das (D_x, D_y) gefunden, welches die SQD minimiert, ist der optische Fluss an der Stelle (x_0, y_0) angegeben durch $(v_x, v_y) = (D_x/D_t, D_y/D_t)$. v_x und v_y sind die Längen der Vektoren in x- und y-Richtung des Vektorfeldes.

[1] Weiteres Beispiel für optischen Fluss:
`http://www.youtube.com/watch?v=g2ZMH-kkvhE`

Betrachtet man ein fixes Objekt und bewegt die Kamera, dann besteht eine Relation zwischen dem Betrag der Flussvektoren und der Entferung eines Punktes zur Kamera. Je näher ein Punkt an der Kamera ist, desto schneller bewegt er sich. Der **Expansionsfokus** (Engl.: Focus of Expansion, **FOE**) ist der zentrale Punkt des optischen Flusses. In ihm scheint der optische Fluss zu entstehen. Aus der Formel für den FOE ([5] und [3]) folgt die Formel für den optischen Fluss:

$$v_x(x,y) = \frac{xT_z - fT_x}{Z(x,y)} \quad , \quad v_y(x,y) = \frac{yT_z - fT_y}{Z(x,y)}. \tag{4}$$

T_x,T_y und T_z bezeichnen die x-Achsen-, y-Achsen- und z-Achsentransformation des optischen Flusses bei (x,y). f ist die Brennweite der Kamera. $Z(x,y)$ ist die z-Koordinate des Punktes in der Szene.
Der Expansionsfokus liegt in diesem Fall am Punkt

$$x_{foe} = f\frac{T_x}{T_z} \;\; und \;\; y_{foe} = f\frac{T_y}{T_z}. \tag{5}$$

da dort die Komponenten $v_x(x_{foe}, y_{foe})$ und $v_y(x_{foe}, y_{foe})$ gleich Null sind. Verwendet man nun (5), kann man (4) umschreiben zu:

$$v_x(x,y) = (x - x_{foe})\frac{T_z}{Z(x,y)} \quad , \quad v_y(x,y) = (y - y_{foe})\frac{T_z}{Z(x,y)}. \tag{6}$$

Stellt man die Formeln aus (6) auf $Z(x,y)$ um, kann man die z-Koordinate bis auf den Skalierungsfaktor T_z bestimmen. Die x- und y-Koordinate in der Szene wird, unter Berücksichtigung der Brennweite, wie folgt bestimmt:

$$X(x,y) = Z(x,y)\frac{x}{f} \quad , \quad Y(x,y) = Z(x,y)\frac{y}{f}. \tag{7}$$

Somit kann man aus dem optischen Fluss die 3D-Szene regenerieren. Eine interessante Anwendung für den optischen Fluss ist in [1] angegeben: Dort wird der optische Fluss verwendet, um mit Hilfe einer Kamera freie Parklücken zu erkennen.

Binokulare Stereopsie. Menschen besitzen zwei Augen, vor allem um die Umgebung in 3D wahrzunehmen. Dieses Prinzip kann man auch auf Computersysteme übertragen. Dabei werden zwei Kameras eingesetzt. Unter normalen Sichtbedingungen fixieren Menschen einen Punkt P_0 mit einer Entfernung Z_0 in der Szene. Dieser Punkt ist auf beiden Bildern an der selben Position. Zudem ist der Abstand b zwischen den Augen (bzw. Kameras) bekannt (ca. 6cm). Nimmt man nun einen anderen Punkt P in der Szene, so befindet sich dieser an unterschiedlichen Positionen der Bilder. Aus der Winkelverschiebung vom Punkt P zum Punkt P_0 kann man die relative Entfernung δZ zum Punkt Z_0 bestimmen. Die Software PhotoModeler der Firma Eos Systems Inc. demonstriert dies eindrucksvoll, indem sie Fotos aus mehreren Perspektiven verwendet, um automatisch ein 3D-Modell zu erstellen.

Oberflächenstruktur-Gradienten. In der Computervision bezieht sich Oberflächenstruktur auf ein wiederholtes Muster auf einer Oberfläche, das visuell abgetastet werden kann. Zum Beispiel die Muster der Fenster in einem Hochhaus, Flecke auf einem Leopard oder auch Kieselsteine am Strand stellen eine Oberflächenstruktur dar. Diese muss aber nicht immer gleichmäßig sein, wie beispielsweise bei den Kieselsteinen ist die Dichte an verschiedenen Stellen des Strandes etwa die selbe, nicht aber auf einem bestimmten Quadratmeter.
Diese Aussagen treffen nur auf die Szene zu. Im Bild hingegen variieren die Größe, Umriss und Abstand aufgrund der perspektivischen Projektion. Durch mathematische Analyse kann man diese sogenannten Oberflächenstruktur Gradienten, wie beispielsweise Fläche, perspektivische Verkürzung und Dichte, bestimmen. Daraus lassen sich dann Aussagen über die Entfernung treffen.

Schattierung. In der Computergrafik ist es das Ziel, die Bildhelligkeit $I(x, y)$ für die Szene zu bestimmen. Die Computervision dagegen versucht, diesen Prozess umzukehren. Das bedeutet, sie will die Geometrie und die Reflexionseigenschaften für die gegebene Bildhelligkeit $I(x, y)$ ermitteln. Dies hat sich als sehr schwierig erwiesen. Das Schwierigste dabei ist, Interreflexionen zu verarbeiten. Normalerweise wird eine Szene nicht nur von einer Lichtquelle, sondern von mehreren beleuchtet. Zusätzlich wird das Licht an Objekten reflektiert die somit als sekundäre Lichtquelle angesehen werden können.
In dieser Ausarbeitung wollen wir auf dieses Thema nicht genauer eingehen. Der interessierte Leser kann sich mit Hilfe des Begriffs **Shape From Shading** näher informieren.

Konturen. Linien in einer Strichzeichnung haben unterschiedliche Bedeutungen. Dabei verwenden wir ein vereinfachtes Modell der Szene, wo die Objekte keine Oberflächenmarkierungen haben und Linien von z.B. Schatten oder Reflexionen bereits entfernt wurden (siehe Abb. 3) , sodass nur mehr Konturen der Objekte zu sehen sind. Die Auswertung einer Strichzeichnung erfolgt durch Linienbeschriftung (Huffman-Clowes-Labeling). Eine Linie kann entweder als Projektion einer **Verästelung** (Punkte wo Sichtlinie eine Tangente zur Oberfläche bildet) oder als **Kante** klassifiziert werden. Um darzustellen, welche der beiden Oberflächen, die an die Linie der Strichzeichnung grenzen, näher in der Szene liegt, gibt es sechs verschiedene Zahlenbeschriftungen (siehe Abb. 4):

- $+$ und $-$ stellen **konvexe** bzw. **konkave** Kanten dar. Konvex bedeutet dabei, dass der Winkel der Kante größer 180° beträgt, konkav das Gegenteil.
- $\leftarrow$ und $\rightarrow$ stellen eine **verdeckende** konvexe Kante dar. Wenn man sich in Pfeilrichtung bewegt, befindet sich die verdeckende Oberfläche rechts.
- $\twoheadleftarrow$ und $\twoheadrightarrow$ stellen eine **Verästelung** dar. Hier krümmt sich die Oberfläche, um sich selbst zu verdecken. In Richtung der Pfeile liegt die Oberfläche rechts.

Von den 6^n möglichen Linienbeschriftungen für n Linien ist nur eine kleine Anzahl physisch möglich. Welche Beschriftungszuordnungen das sind ist das

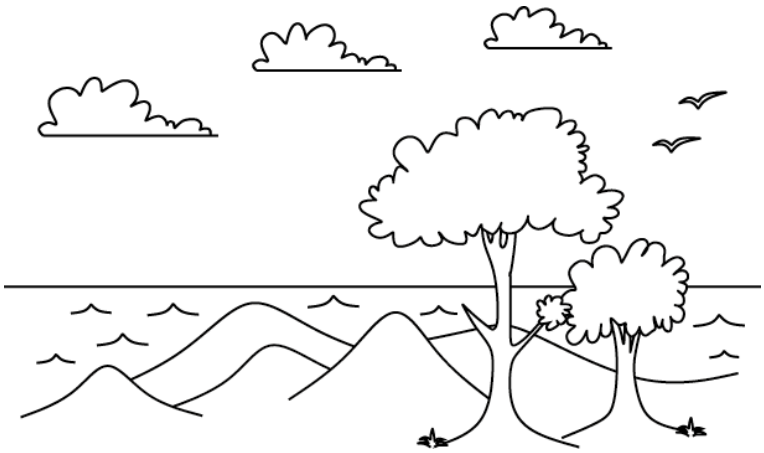


Abb. 3. Strichzeichnung ohne Schatten und Oberflächenmarkierungen. Quelle: [5]

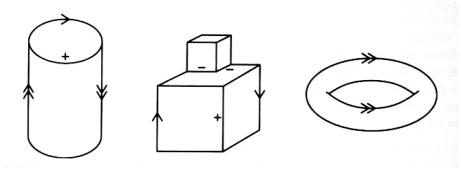


Abb. 4. Verschiedene Arten von Linienbeschriftungen. Quelle: [5]

Problem der Linienbeschriftung. Huffman (1971) und Clowes (1971) haben unabhängig voneinander den ersten Ansatz für die Szenenanalyse erarbeitet. Sie haben dabei ihre Analyse auf undurchsichtige trihedrale Festkörper beschränkt - Objekte, für die an jeder Ecke genau drei Oberflächen zusammentreffen. Sie haben zudem Objektausrichtungen ausgeschlossen, die die trihedrale Bedingung verletzen. Sie haben anschließend eine erschöpfende Auflistung aller möglichen Ecken erstellt.
Waltz (1975) schlug einen ersten Algorithmus zur automatischen Kantenbeschriftung vor. Durch die Kantenbeschriftung ist es möglich, Aussagen über das Verhältnis der Oberflächen zueinander zu treffen und einfache 3D Eigenschaften zu bestimmen.

3 Objekterkennung

Im vorheringen Abschnitt haben wir gesehen, wie man aus 2D-Bildern 3D Informationen extrahieren kann. Oft ist es aber auch wünschenswert, dass ein Computer erkennt, um welches Hindernis oder Gegenstand es sich handelt. Manchmal ist es aber auch nur erforderlich, ohne 3D Informationen z.B. ein Gesicht oder Handschriften zu erkennen.
Die Vision wird nicht nur zur Erkennung von Objekten sondern auch von Aktivitäten verwendet. Beispielsweise kann man Gesichtsausdrücke, Gesten, Aktionen, usw. erkennen. Die Forschung zur Aktivitätserkennung steckt aber immer noch in den Kinderschuhen. Wobei dieser Bereich in den letzten Monaten einen starken Aufschwung erfuhr: Microsoft präsentierte die Kinect, wo basierend auf ein Open-Source Framework jeder Hobbyprogrammierer seine eigene Aktivitäts- und Gestenerkennung programmieren kann. Daraus sind bereits sehr interessante Projekte entstanden (kinecthacks.net). Wir werden uns aber mehr auf die Objekterkennung konzentrieren.
Ein Computer sollte in der Lage sein, ein Objekt in seinen unterschiedlichen Variationen zu erkennen. Dies stellt sich als relativ schwierig heraus: Ein Gesicht kann unterschiedlich beleuchtet sein, eine andere Position im Hinblick zur Kamera sowie diverse Gesichtsausdrücke haben. Für eine Kategorie-Erkennung wie etwa „Auto“ ist es zudem notwendig, Objekte mit vollkommen unterschiedlichen Umrissen zu erkennen und einzuordnen.
Allgemein gibt es zwei Anzätze: die **helligkeitsbasierte Erkennung**, wobei Pixel-Helligkeitswerte verwendet werden, und die **merkmalbasierte Erkennung**, wobei die räumliche Anordnung der extrahierten Merkmale verwendet wird, wie beispielsweise Kanten oder Schlüsselpunkte. Ist ein Objekt erkannt, gilt es noch, dessen **Pose**, also die Ausrichtung und Position zu bestimmen. Diese drei Aspekte werden wir in den folgenden Abschnitten genauer untersuchen. Im letzten Abschnitt wenden wir uns der Objekterkennung in einer **3D-Punktewolke** zu, wie sie beispielsweise von 3D-Lasersensoren generiert wird.

3.1 Helligkeitsbasierte Erkennung

Wir definieren die Merkmale eines Objektes, beispielsweise ein Gesicht, anhand seiner Pixelhelligkeitswerte. Man könnte dazu das Bild mit diversen linearen Filtern falten (siehe 2.1) und die Pixelwerte des Ergebnisses als Merkmale behandeln. Dieser Ansatz ist sehr erfolgreich für Aufgaben wie beispielsweise die Erkennung von Handschriften, wo es mehr auf die Konturen anstatt dem Helligkeitsprofil ankommt.
Ein weiterer Ansatz ist die Verwendung von statischen Methoden wie neuronale Netzwerke mit reinen Pixeleingaben, Entscheidungsbäume mit Merkmalen, die durch verschiedene Balken- und Kantenfilter definiert wurden, sowie Bayessche Modelle mit Wavelet-Merkmalen.
Bei der Verwendung der Pixelhelligkeit als Merkmal sollte man vor allem darauf achten, die große Redundanz der unterschiedlichen Merkmale zu reduzieren: vergleicht man z.B. die Backe von zwei Gesichtern, sind die benachbarten Pixel sehr ähnlich. Diese Redundanz kann man mit Datenreduktionstechniken verringern, um den Merkmalsvektor zu verkleinern und dadurch die Objekterkennung effizienter zu gestalten.

3.2 Merkmalbasierte Erkennung

Für die Objekterkennung können auch Kanten oder Bereiche (siehe 2.1) verwendet werden. Dies hat gegenüber der Pixelhelligkeit zwei große Vorteile: Zum Einen sind weniger Daten für die Abspeicherung der Merkmale nötig, es gibt sehr viel weniger Kanten als Pixel im Bild, zum Anderen sind Kanten beleuchtungsinvariant: Innerhalb eines geeigneten Kontrastbereiches werden die Kanten, unabhängig von der Beleuchtungskonfiguration, an etwa denselben Positionen erkannt.
Die Anordnung der Kanten ist charakteristisch für ein Objekt. Das ist der Grund, warum wir Strichzeichnungen einfach interpretieren können (Abb. 3). Man bestimmt die Distanz der Kanten des aktuellen Objektes mit den Kantenmerkmalen bereits bekannter Objekte und findet so den nächsten Nachbarn, welcher die kleinste Distanz besitzt. Um die Distanz zu berechnen verwenden wir die Idee der deformierbaren Übereinstimmung: D'Arcy Thompson (1917) hat beobachtet, dass verwandte, aber nicht identische Umrisse häufig unter Verwendung einfacher Koordinatentransformationen in die gleiche Ausrichtung deformiert werden können. Dieses Konzept beruht auf einen dreistufigen Prozess: (1) Lösung des Entsprechungsproblemes zwischen den beiden Umrissen (finden von identischen Punkten), (2) Verwendung der Entsprechungen, um eine Ausrichtungstransformation abzuschätzen, (3) Berechnung der Distanz zwischen beiden Umrissen.
Wir stellen einen Umriss durch eine diskrete Menge von Punkten dar (siehe Abb. 5.a und 5.b). Jetzt betrachten wir einen Beispielpunkt p_i zusammen mit den Vektoren, die von diesem Punkt zu allen anderen Punkten des Umrisses ausgehen. Diese Vektoren werden unter Verwendung der mittleren Distanz zwischen Punktepaaren normalisiert, um eine Skalierungsinvarianz zu erreichen. Dieses

Konzept drückt den Umriss relativ zum Punkt p_i aus. Aus den Vektoren erstellt man einen Umrisskontext, indem ein grobes räumliches Histogramm h_i für den Punkt p_i erstellt wird. Dazu wird die Anzahl der Punkte in Richtung des jeweiligen Winkels in ein logarithmisches Polarkoordinatensystem (siehe Abb. 5.c) eingetragen. Logarithmisch deswegen, um den Umrisskontext sensibler für Differenzen bei naheliegenden Pixel zu machen.
Umrisskontexte erlauben, das Entsprechungsproblem zwischen zwei ähnlichen, aber nicht identischen Umrissen zu lösen, da diese für entsprechende Punkte sehr ähnlich ausfallen. Vergleichen Sie dazu die markierten Punkte $\circ$ und $\triangleleft$ in Abbildung 5.a und 5.b. Diese haben einen ähnlichen Umrisskontext: 5.d und 5.e . Der Punkt $\Diamond$ hingegen hat einen vollkommen anderen Umrisskontext (Abb. 5.f).
Programmatisch kann man zwei Histogramme h_i und h_j vergleichen, indem man die χ^2-Distanz verwendet:

$$C_{ij} = \frac{1}{2} \sum_{k=1}^{K} \frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)} \tag{8}$$

Dabei bezeichnen $h_i(k)$ und $h_j(k)$ den k-ten Wert der normalisierten Histogramme an den Punkten p_i und p_j. Ziel ist es, die Summe aller Kosten C_{ij} zwischen allen Punkten i auf dem ersten und j auf dem zweiten Umriss zu minimieren. Mit dem gewichteten Bipartite-Matchingproblem findet man den Gegenpunkt im zweiten Umriss.
Mit den somit erhaltenen Entsprechungen kann man eine Ausrichtungstransformation abschätzen, die einen Umriss auf einem anderen abbildet (siehe Abb. 5.g). Die Distanz zwischen zwei Umrissen kann nun definiert werden als gewichtete Summe der Umrisskontextdistanzen C_{ij} der entsprechenden Punkte gewichtet mit den Transformationsvektoren.

3.3 Posenabschätzung

Für die Manipulation eines Objektes ist es zusätzlich zur Erkennung desselben notwendig, die Pose des Objektes zu bestimmen. Ein Roboterarm kann ein Objekt nur aufnehmen, wenn dessen Pose bekannt ist. Bei Festkörperobjekten, egal ob zwei- oder dreidimensional, verwendet man dazu die **Ausrichtungsmethode**, welche im Folgenden behandelt wird.
Das Objekt wird durch M Merkmale (z.B. unterschiedliche Punkte) m_1,m_2,...,m_M im dreidimensionalen Raum dargestellt. Dies können beispielsweise die Ecken eines polyhedralen Objektes sein. Diese Punkte werden einer dreidimensionalen Drehung $\mathbf{R}$ unterzogen, gefolgt von einer Übersetzung in einen unbekannten Betrag $\mathbf{t}$ sowie eine Projektion, um die Bildmerkmalpunkte p_1,p_2,...,p_N auf der Bildebene entstehen zu lassen (3D zu 2D). Im Allgemeinen gilt $N \neq M$, da einige Modellpunkte verdeckt sein können oder einige Merkmale bei der Detektion übersehen wurden. Dies können wir ausdrücken als:

$$p_i = \Pi(\mathbf{R}m_i + \mathbf{t}) = Q(m_i) \tag{9}$$

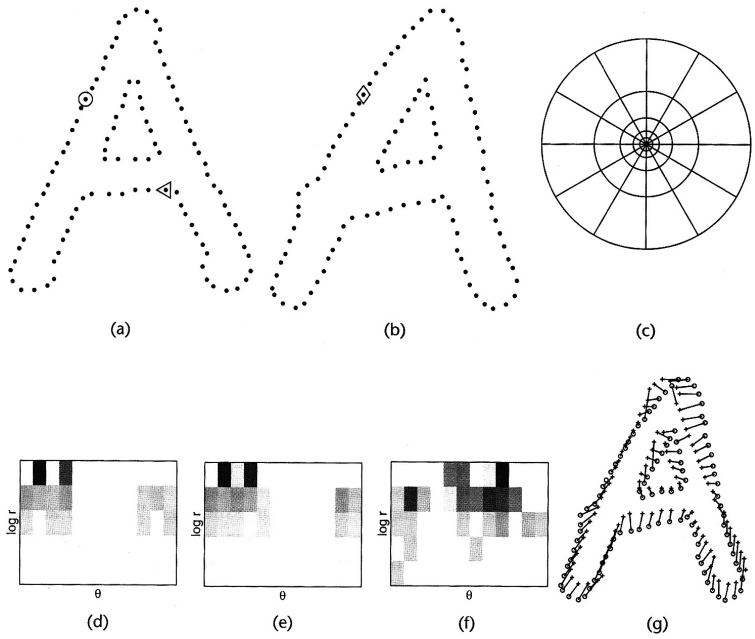


Abb. 5. (a,b) Zwei verschiedene Umrisse eines gleichen Objektes, (c) Polarkoordinatensystem mit logarithmischer Skalierung (wir verwenden fünf Werte für *logr* und zwölf Werte für den Winkel θ, (d,e,f) Umrisskontexte (Polarhistogramme) für die Punkte $\circ$, $\Diamond$, $\triangleleft$. Dunkle Werte stellen mehr Punkte in dieser Richtung dar. Quelle: [5]

hier ist $\mathbf{R}$ die Rotationsmatrix, $\mathbf{t}$ eine Übersetzung und Π gibt die perspektivisches Projektion an. Das Endergebnis ist die Transformation Q, die den Modellpunkt m_i im Hinblick auf den Punkt p_i ausrichtet. Dabei wissen wir, dass Q für alle Modellpunkte gleich sein muss.
Man kann nach Q auflösen, wenn man die dreidimensionalen Koordinaten der Modellpunkte und ihre zweidimensionale Projektion kennt. Das heißt, man kann Gleichungen aufschreiben, die Koordinaten von p_i mit denen von m_i in Verbindung bringen. Die Parameter in diesen Gleichungen entsprechen dabei den Parametern der Rotationsmatrix $\mathbf{R}$ und des Übersetzungsvektors $\mathbf{t}$. Wenn man ausreichend viele Gleichungen hat, sollte man in der Lage sein, nach Q aufzulösen. Q gibt somit die Pose des Objektes an.

3.4 Erkennung in 3D Punktewolke

In den vorherigen Abschnitten haben wir hauptsächlich die Verarbeitung von Daten aus Kamerabildern besprochen. In der Einleitung haben wir gesehen, dass ein weiterer wichtiger Sensor existiert: der 3D-Laser Sensor. Dieser liefert eine 3D-Punktewolke aus der Umgebung. Das reicht bereits aus, um Hindernissen auszuweichen. Um diese 3D-Daten aber sinnvoll für z.B. einen Serviceroboter zu nutzen, werden spezifischere Informationen über Objekttyp, Pose und Position benötigt. Hierzu ist es notwendig, Korrespondenzen zwischen erfassten Punkten und Objektpunkten in einer Datenbank zu finden. Roboter sind bereits in der Lage, Objekte zu lokalisieren, von denen Punktewolken-Darstellungen zur Verfügung stehen. Die Objekte aber zu identifizieren, ob es sich um eine Tasse, eine Flasche, etc. handelt, ist noch aktives Forschungsgebiet, welches unter anderem auf dem Lehrstuhl „Intelligent Autonomous Systems Group“ der Technischen Universität München erforscht wird (siehe Open Thesis).
Im Juni 2010 veröffentlichte PhD Bastian Steder auf Willow Garage ein Verfahren zum Erkennen von Objekten in 3D-Scans [2]. Dieser Algorithmus ist bereits in in die PCL (Point Cloud Lybrary) des ROS (Robot Operating System) eingeflossen. Aufgrund der immer mehr verbreiteten 3D-Sensoren (z.B. Kinect) wurde für die PCL eine eigene Homepage (`http://pointclouds.org/`) gegründet, um die Informationen und Projekte zu bündeln.

4 Navigation

Neben der Objektmanipulation ist die Navigation ein weiterer wichtiger Askpekt der Wahrnehmung: Für sie ist es notwendig, gewisse Informationen aus Bildern zu extrahieren. Dabei sollte man aber beachten, dass nicht immer alle zur Verfügung stehenden Informationen ausgewertet werden müssen. Beispielsweise bei einem autonom fahrenden Auto sind fogende Punkte wichtig:

[2] 3D Point Cloud Based Object Recognition System, `http://www.willowgarage.com/blog/2010/10/06/3d-point-cloud-based-object-recognition-system`

1. Seitensteuerung: sicherstellen, dass das Fahrzeug in der Spur bleibt. Dies kann mit Kantenerkennung zwischen Asphalt und der Straßenmarkierung bewerkstelligt werden.
2. Längssteuerung und Vermeidung von Hindernissen: sicherstellen, dass ein sicherer Abstand zum Vordermann eingehalten wird und dass Hindernissen sicher ausgewichen werden kann. Dazu verwendet man binokulare Stereopsie oder den optischen Fluss, um Objekte zu erkennen.

Dieses Beispiel verdeutlicht, dass nicht alle Informationen erkannt werden müssen, die im Prinzip aus einem Bild erkannt werden könnten. Man muss also nicht den genauen Umriss jedes Fahrzeugs erkennen oder die Oberflächenstruktur des Grases am Straßenrand bestimmen.
Im folgenden letzten Abschnitt der Ausarbeitung behandeln wir ein weiteres Beispiel für die Verwendung von Wahrnehmung für die Navigation.

4.1 SLAM

Simultaneous **L**ocalization **a**nd **M**apping fordert von einem Roboter, welcher in einem unbekannten Ort in einer unbekannten Umgebung platziert wurde, schrittweise eine einheitliche Karte der Umgebung mit gleichzeitiger Bestimmung der eigenen Position zu erstellen. Der Vorteil von dynamisch generierten Karten wie bei SLAM gegenüber statischen Karten ist, dass auf zwischenzeitlichen Veränderungen der Umgebung reagiert und gehandelt werden kann. Zudem muss bei SLAM nicht die gesamte Umgebung abgespeichert und behandelt werden, sondern nur jene um der aktuellen Position.
Bei der gleichzeitigen Erstellung von Karten und bestimmen der eigenen Position eines Roboters handelt es sich um das bekannte Henne-Ei Problem: Um die aktuelle Position zu bestimmen, ist es notwendig, Kenntnisse über die Umgebung zu haben. Um eine Karte zu erstellen, ist es notwendig, die aktuelle Position zu kennen.

SLAM besteht aus mehreren Schritten (siehe auch Abb. 6 und 7) : Die Daten eines Lasersensors werden ausgewertet und Orientierungspunkte gesucht. Der Roboter bewegt sich um eine gewisse Entfernung. Diese Entfernung wird mit einem Wegmesser gemessen und einem Extended Kalman Filter (EKF) übergeben. Anschließend misst der Roboter erneut die Umgebung und bestimmt Orientierungspunkte. Diese Punkte werden mit bereits bekannten verglichen. Bereits bekannte Orientierungspunkte werden dann vom EKF verwendet, um die Position des Roboter in der Karte zu aktualisieren. Noch nicht bekannte Orientierungspunkte werden dem EKF für spätere Wiedererkennung hinzugefügt. Dieser Vorgang wiederholt sich dann immer wieder.
Ein Kalman Filter entfernt dabei im Allgemeinen Störungen bei Messungen, also die Ungenauigkeit der aktuellen Position und der Ungenauigkeit der gefundenen Orientierungspunkten.
Folgende Abbildungen, entnommen aus [4] sollen das Prinzip von SLAM anschaulicher darstellen:

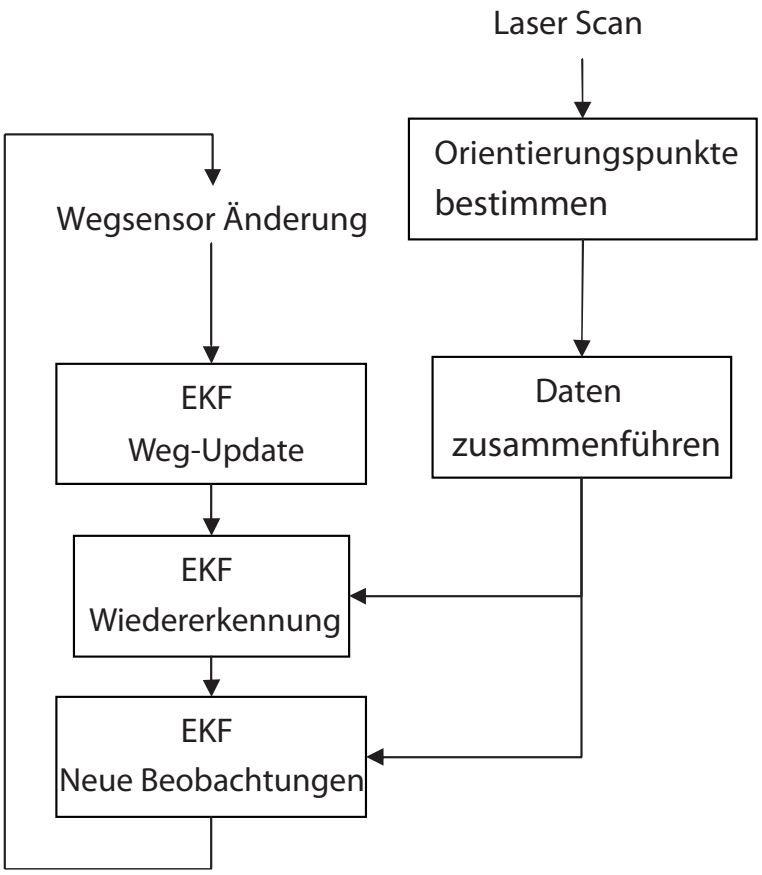
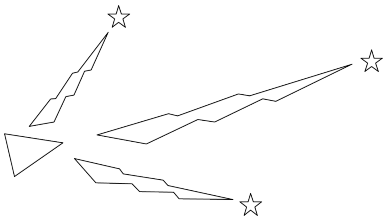
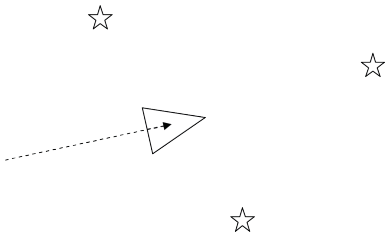


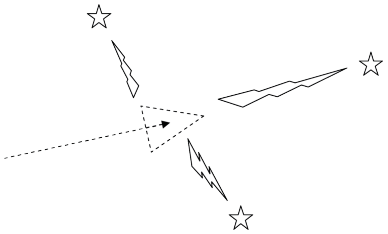
Abb. 6. Die verschiedenen Phasen bei SLAM. Quelle: [4]



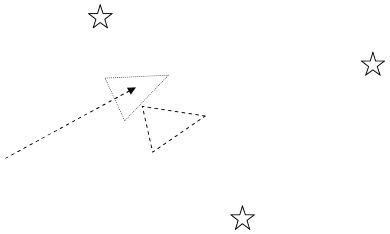
(a) Der Roboter wird dargestellt als Dreieck. Die Sterne sind Orientierungspunkte. Der Roboter bestimmt als Erstes die Positionen der Orientierungspunkte.



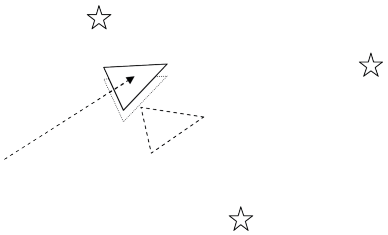
(b) Der Roboter bewegt sich nach vorne. Die Entfernung ist gegeben durch den Wegmesser des Roboters. Somit denkt der Roboter er befindet sich an dieser Position.



(c) Der Roboter misst erneut die Umgebung und findet heraus, dass er sich nicht an der Position befindet, wo er denkt, dass er ist.



(d) Aus den neuen Sensordaten wird nun die Position des Roboters auf der Karte (gepunktet) bestimmt. Die vorher gedachte Position ist gestrichelt dargestellt.



(e) In der Realität befindet sich der Roboter hier (durchgezogenes Dreieck). Die Abweichung zur vorher bestimmten Position (gepunktet) beruht auf Messfehlern.

Abb. 7. SLAM erklärt an einem Beispiel. Quelle: [4]

Literatur

1. John Alberts, *Seminararbeit bildverarbeitung für das projekt faust optischer fluss focus of expansion*, http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07/alberts/report.pdf, 15.02.2007.
2. Iris Christadler, *Mehrgitterverfahren für die berechnung des optischen flusses mit nicht-standardregularisierungen*, http://www.lrz.de/~christadler/da/ausarbeitung.pdf, 01.12.2004.
3. Suhr Jae Kyu, Bae Kwanghyuk, Kim Jaihie, and Jung Ho Gi, *Free parking space detection using optical flow-based euclidean 3d reconstruction*, http://b2.cvl.iis.u-tokyo.ac.jp/mva/proceedings/2007CD/papers/15-01.pdf, 06.04.2007.
4. Søren Riisgaard and Morten Rufus Blas, *Slam for dummies*, http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf, 23.10.2005.
5. Stuart Russell and John F. Canny, *Künstliche intelligenz: Ein moderner ansatz*, 1 ed., Pearson Studium, München ;, Boston [u.a.], 2004.