

Eine Systemarchitektur für effiziente videobasierte Fahrerassistenzsysteme

Dissertation

zur Erlangung des Grades eines
Doktor-Ingenieurs
der Fakultät
für Elektrotechnik und Informationstechnik
an der Ruhr-Universität Bochum

Vorgelegt von

Jan Salmen

geboren in Bochum

Datum der mündlichen Prüfung: 05. August 2013

Berichter: Prof. Gregor Schöner und Prof. Christian Igel

Meinen Eltern

Inhaltsverzeichnis

Problemstellung und Motivation	1
I. Grundlagen	7
1. Bildmerkmale	9
1.1. Gradienten und Kanten	9
1.2. Haar-Merkmale	10
1.3. HOG-Merkmale	13
1.4. Census-Transformation	15
1.5. Farbräume	16
1.6. Vergleich von Bildern	18
2. Objekterkennung	21
2.1. Detektion und Klassifikation	21
2.2. Lineare Diskriminanzanalyse	22
2.3. Der Viola-Jones-Detektor	23
3. Stereo-Verarbeitung	29
3.1. Stereogeometrie	29
3.2. Berechnen einer Kostenmatrix	31
3.3. Echtzeitfähige Algorithmen	32
4. Schätzen von optischem Fluss	39
4.1. Dichte Schätzung	40
4.2. Spärliche Schätzung	41
5. Evolutionäre Optimierung	43
5.1. Die $(\mu/\rho + \lambda)$ -ES	43
5.2. Kovarianzmatrix-Adaptation: CMA-ES	45
5.3. Mehrziel-Optimierung	47
6. Naive Architektur für ein Gesamtsystem	51

II. Entwurf einer vereinheitlichenden Architektur	53
7. Vorgeschlagene Systemarchitektur	55
8. Objekterkennung basierend auf Haar-Merkmalen	57
8.1. Detektion von Fußgängern	57
8.1.1. Vorgeschlagenes Vorgehen	58
8.1.2. Evolutionäre Optimierung von Merkmalen	59
8.1.3. Experimente	64
8.1.4. Diskussion	68
8.2. Detektion von Verkehrszeichen	70
8.2.1. Benchmark-Datensatz	70
8.2.2. Experimente	72
8.2.3. Diskussion	79
9. Stereo-Verarbeitung basierend auf Haar-Merkmalen	81
9.1. Kostenberechnung basierend auf Haar-Merkmalen	81
9.2. Evolutionäre Optimierung mittels CMA-ES	82
9.3. Experimente	83
9.3.1. Aufbau	83
9.3.2. Ergebnisse	86
9.4. Diskussion	87
10. Optischer Fluss basierend auf Haar-Merkmalen	89
10.1. Anpassung des PowerFlow-Algorithmus	89
10.2. Evolutionäre Mehrziel-Optimierung	91
10.3. Experimente	92
10.3.1. Aufbau	92
10.3.2. Ergebnisse	95
10.4. Diskussion	98
11. Effizientes Update der Kovarianzmatrix bei iterierter LDA	101
11.1. Problemstellung	102
11.2. Vorgeschlagenes Vorgehen	104
11.3. Experimente	107
11.3.1. Aufbau	107
11.3.2. Ergebnisse	109
11.4. Diskussion	111
12. Bilder von Google Street View als Datenquelle	113
12.1. Problemstellung	113
12.2. Zugriff auf Bilddaten	115

12.3. Experimente	119
12.3.1. Aufbau	120
12.3.2. Ergebnisse	122
12.4. Diskussion	124
13. Zusammenfassung, Bewertung und Ausblick	127
A. Lebenslauf	131
Literaturverzeichnis	133

Problemstellung und Motivation

Die Engländerin Bridget Driscoll war eines der ersten Todesopfer im modernen Straßenverkehr. Sie wurde 1896 in London von einem der damals seltenen Autos angefahren, als sie die Straße überquerte. In der anschließenden Gerichtsverhandlung sagten Zeugen aus, der Fahrer Arthur Edsall sei mit „rücksichtsloser Geschwindigkeit“ gefahren, das Auto so schnell wie „ein gutes Pferd im Galopp“. Die Begutachtung des Fahrzeugs ergab, dass es höchstens 13 km/h gefahren sein konnte. Die 6-stündige Verhandlung endete mit einem Freispruch, der Richter äußerte zum Abschluss „er hoffe, dass so etwas nie wieder passieren werde.“¹

Heute schätzt die Weltgesundheitsorganisation WHO, dass weltweit im Straßenverkehr jedes Jahr mehr als eine Million Menschen getötet werden². Die meisten Verkehrsunfälle lassen sich auf menschliches Fehlverhalten zurückzuführen³. Fehler im alltäglichen Leben können selten so gravierende Auswirkungen haben wie wenn sie im Straßenverkehr passieren. Es gibt daher seit Jahrzehnten vielfältige Bestrebungen, immer fortschrittlichere Technik zu entwickeln und einzusetzen, um den Straßenverkehr sicherer zu machen.

Passive Sicherheitssysteme (wie bspw. Gurte und Airbags) sollen die Auswirkungen von Unfällen verringern, sie dienen typischerweise dem Schutz der Insassen von Kraftfahrzeugen. Passive Sicherheitssysteme haben dazu beigetragen, dass die Anzahl der im Straßenverkehr Getöteten in Deutschland, bezogen auf den Bestand an Kraftfahrzeugen, in den letzten 50 Jahren praktisch kontinuierlich gesunken ist, vgl. Abbildung 1(a).

Während die Autoinsassen selbst immer besser geschützt sind, ist die Zahl der Opfer unter den „schwächeren“ oder „leicht verletzlichen“ Verkehrsteilnehmern (Fußgänger, Radfahrer und Motorradfahrer) vergleichsweise hoch: In Deutschland sind inzwischen mehr als die Hälfte aller Schwerverletzten und Todesopfer im Straßenverkehr selbst keine Autoinsassen, vgl. Abbildung 1(b). Motorradfahrer gehören zu den am stärksten

¹<http://www.bbc.co.uk/news/magazine-10987606>

²<http://www.who.int/research>

³In Deutschland wurden bspw. 2010 bei 17 % aller Unfälle mit Personenschaden „mitverursachende“ Faktoren wie Glätte festgestellt und in weniger als 1 % der Fälle technische Mängel als Ursache (Statistisches Bundesamt, 2011b).

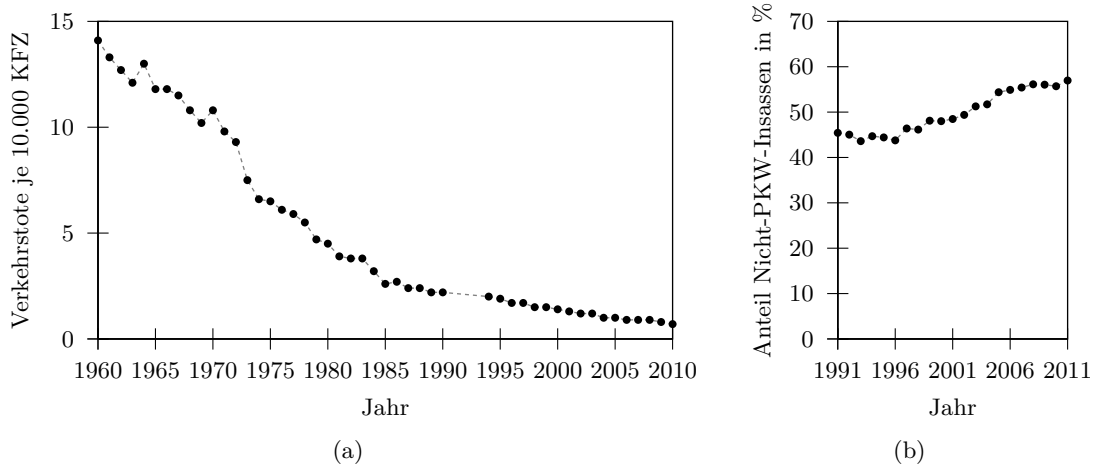


Abbildung 1.: (a) Anzahl der Verkehrstoten, gemessen am Bestand, in Deutschland. Quelle: Daten von (Statistisches Bundesamt, 2011b).
 (b) Anteil von Nicht-PKW-Insassen an Schwerverletzten und Getöteten im Straßenverkehr in Deutschland. Quelle: Daten von (Statistisches Bundesamt, 2011b).

gefährdeten Verkehrsteilnehmern (Statistisches Bundesamt, 2011a). Dabei ist beachtenswert, dass weniger als ein Drittel der Unfälle zwischen einem PKW und einem Motorrad vom jeweils beteiligten Motorradfahrer verursacht wird.

Aktive Sicherheitssysteme sollen zur Vermeidung von Unfällen beitragen, damit haben sie große Bedeutung beim Schutz schwächerer Verkehrsteilnehmer. Viele elektronische aktive Sicherheitssysteme werden unter dem Begriff „fortschrittliche Fahrerassistenzsysteme“ (FAS) zusammengefasst. Beispiele für heute bereits recht verbreitete FAS sind das elektronische Stabilitätsprogramm ESP und die intelligente Abstandsregelung ACC.

Verschiedene FAS greifen auf unterschiedliche Datenquellen zurück: mechanische Sensoren, Radar-, Infrarot- oder Ultraschalltechnik. Jede dieser Sensortechniken impliziert spezifische Einschränkungen. Mechanische Sensoren erlauben praktisch keine Aussagen über die Zukunft, Radar- und Infrarotdaten erfassen nicht das sichtbare Licht, aktuelle Ultraschallsensoren haben eine sehr begrenzte Reichweite.

Die vielversprechendste Alternative für aktuelle und zukünftige Entwicklungen sind Videokameras. Sie kombinieren große Vorausschauweite mit hoher örtlicher und zeitlicher Auflösung. Videobilder enthalten alle Informationen, die zum sicheren, komfortablen und ökonomischen Fahren nötig sind – auch Menschen steuern Fahrzeuge

hauptsächlich basierend auf visueller Information.

Erste videobasierte FAS sind inzwischen in Serienfahrzeugen verfügbar, beispielsweise Fahrspurverlassenswarnung und Verkehrszeichenerkennung. Weitere Applikationen sind Themen der aktuellen Forschung und werden ebenfalls bald Serienreife erlangen. Der Einsatz kamerabasierter Systeme wird teilweise für Neufahrzeuge bereits vorgeschrieben (z. B. vorausschauenden Notbrems- und Spurhalteassistentensysteme für schwere Nutzfahrzeuge in Deutschland ab 2015⁴).

Der verstärkte Einsatz kamerabasierter FAS kann dazu beitragen, die Zahl der Unfallopfer weiter zu reduzieren. Mittelfristig sollen nicht nur automatische Notbremsungen, sondern auch automatische Ausweichmanöver möglich werden. Langfristig werden kamerabasierte Applikationen benötigt, um die Vision von autonom fahrenden Serienfahrzeugen zu verwirklichen.

Bevor neu entwickelte videobasierte FAS in Serienfahrzeugen eingesetzt werden können, gibt es zahlreiche Hürden. Am deutlichsten wird das, wenn FAS aktiv in die Steuerung eingreifen sollen. Das bis heute gültige „Wiener Übereinkommen über den Straßenverkehr“ von 1968 schreibt u. a. vor, dass „jedes Fahrzeug [...] wenn es in Bewegung ist, einen Führer haben muss“ und dass dieser „dauernd sein Fahrzeug beherrschen [...] können muss“⁵. Solche gesetzlichen Vorschriften können natürlich geändert werden, wenn es dafür eine gesellschaftliche Akzeptanz gibt. Entsprechende Fragen im Zusammenhang mit autonom fahrenden Autos müssen öffentlich diskutiert werden. Computer könnten in naher Zukunft Entscheidungen über Leben und Tod treffen, entsprechende Szenarien lassen sich leicht konstruieren (z. B. plötzlich auftauchende Person auf der Fahrbahn und Ausweichen nur in den Gegenverkehr möglich). Solche ethischen Fragen sind jedoch außerhalb des Fokus der vorliegenden Arbeit.

Hier werden die großen technischen Herausforderungen betrachtet, die sich bei der Entwicklung kamerabasierter Assistenzsysteme ergeben. Die betrachteten Systeme leisten auch dann einen sinnvollen Beitrag zur Sicherheit, wenn sie nicht autonom handeln, z. B. indem die Aufmerksamkeit des Fahrers rechtzeitig gelenkt wird. Die zusätzliche Hardware für solche Sicherheitssysteme muss sich sinnvoll und kostengünstig in Serienfahrzeuge integrieren lassen. Dementsprechend muss die eingesetzte Software so wenig Ressourcen wie möglich benötigen. Heute sind videobasierte Systeme auch aufgrund der resultierenden Kosten noch nicht weit verbreitet.

Betrachtet man den aktuellen Stand der Technik, dann fällt auf, dass bisher vorgeschlagenen Algorithmen für typische Applikationen sich voneinander stark unterscheiden. Populäre leistungsfähige Verfahren (bspw. Objekterkennung, Schätzen von optischem

⁴Verordnung (EG) Nr. 661 / 2009 des Europäischen Parlaments und des Rates vom 13. Juli 2009

⁵http://www.admin.ch/ch/d/sr/0_741_10/index.html

Fluss, Stereo-Bildverarbeitung) greifen auf unterschiedliche Bildmerkmale zurück. Solche Merkmale müssen jeweils separat in eigenen Vorverarbeitungsschritten aus den rohen Bilddaten berechnet (extrahiert) werden.

Hier Verbesserungen zu erzielen stellt eine zentrale Herausforderung dar, wie eine vereinfachte Betrachtung zeigt. Dafür wird angenommen, dass auf die Merkmalsberechnung und den folgenden Algorithmus jeweils dieselbe Laufzeit entfällt. Sobald zwei Applikationen auf eine gemeinsame Vorverarbeitung zurückgreifen, sinkt die Rechenzeit insgesamt um 25%. Die Einsparung wird entsprechend größer, je mehr Module integriert werden.

In der vorliegenden Arbeit wird eine neue Systemarchitektur vorgestellt, die kamera-basierte FAS in ein einheitliches *Framework* fasst. Dafür wird eine universelle Vorverarbeitungsstufe vorgeschlagen. Algorithmen für verschiedene relevante Aufgaben im Kontext von FAS werden so angepasst, dass sie im neuen Gesamtsystem einsetzbar sind. Es wird gezeigt, dass alle Algorithmen innerhalb dieser Architektur eine Leistung erreichen können, die mit dem bisherigen Stand der Technik mindestens vergleichbar ist. Das neue Vorverarbeitungsmodul kann zentrale Aufgaben übernehmen, die sonst mehrfach parallel gelöst werden müssten. Insgesamt wird die Komplexität des Gesamtsystems deutlich reduziert. Damit wird ein Beitrag geleistet, um videobasierte FAS einfacher, schneller und kostengünstiger zur Anwendung zu bringen.

Im ersten Teil der vorliegenden Arbeit werden wichtige Grundlagen eingeführt, dabei wird ein Überblick über relevante Literatur gegeben: Bildmerkmale in Kapitel 1, Lernverfahren für Objektdetektion und -klassifikation (Kapitel 2), Verfahren zur Stereo-Verarbeitung (Kapitel 3) und Algorithmen zum Schätzen von optischem Fluss (Kapitel 4). Außerdem wird in Kapitel 5 ein kurzer Überblick über Verfahren zur evolutionären Optimierung gegeben. Aus dem aktuellen Stand der Technik wird schließlich eine mögliche „naive“ Architektur für ein entsprechendes Gesamtsystem hergeleitet (Kapitel 6).

Im zweiten Teil dieser Arbeit (S. 55 ff.) werden eigene Beiträge dargestellt. Dafür wird zunächst in Kapitel 7 ein Überblick über die neu vorgeschlagene effiziente Systemarchitektur gegeben. Danach werden verschiedene Module vorgestellt, die innerhalb dieser Architektur umgesetzt wurden: Ansätze zur Detektion und Klassifikation verschiedener relevanter Objekte (Kapitel 8), zur Stereo-Verarbeitung in Kapitel 9 und zum Schätzen von optischem Fluss in Kapitel 10.

Im Zusammenhang mit den untersuchten Algorithmen werden zwei weitere Verfahren vorgeschlagen, die künftige Entwicklungen positiv beeinflussen können. Das wiederholte Training von Klassifikatoren mittels linearer Diskriminanzanalyse, bspw. erforderlich für Merkmals-Selektion und -Optimierung, lässt sich erheblich beschleunigen, eine neue Methode dazu wird in Kapitel 11 vorgestellt. Das Sammeln von Trainingsdaten

für Lernverfahren zur Objekterkennung ist mit großem Aufwand und entsprechenden Kosten verbunden. Bilder, die über *Google Street View* öffentlich verfügbar sind, können insbesondere in frühen Entwicklungsphasen eine gute Alternative darstellen. Ein entsprechendes Vorgehen wurde entwickelt und untersucht (Kapitel 12).

Die Arbeit endet mit einer Zusammenfassung und einem Ausblick auf mögliche weiterführende Arbeiten in Kapitel 13.

Die hier vorgestellten Beiträge zum Entwurf effizienter videobasierter FAS wurden teilweise bereits veröffentlicht in Salmen u. a. (2007, 2009, 2010, 2011, 2012) oder befinden sich aktuell noch unter Begutachtung (Houben u. a., 2013, Michael u. a., 2013). Weitere Beiträge, z. B. im Bereich FAS für Motorräder in Schlipsing u. a. (2011, 2012) und Benchmarking von Verfahren zur Verkehrszeichen-Klassifikation in Stallkamp u. a. (2011, 2012), werden im Rahmen der vorliegenden Arbeit nicht betrachtet.

Teil I.
Grundlagen

1. Bildmerkmale

Ein Grauwertbild ist eine Funktion auf einem 2-dimensionalen diskreten Definitionsbereich. Die Funktionswerte stellen die Helligkeiten einzelner Bildpunkte dar. Diese Punkte (oder *Pixel*) können über ihre jeweilige Position (x, y) mit $0 \leq x < w$ und $0 \leq y < h$ bestimmt werden, wobei w die Breite des Bildes und h seine Höhe ist. In der digitalen Bildverarbeitung wird ein Koordinatensystem mit Ursprung in der linken oberen Ecke verwendet. Der Wertebereich von Bildern ist typischerweise ganzzahlig, die Helligkeit wird häufig mit einer Auflösung von 8 Bit (Wertebereich von 0 bis 255), teilweise bis zu 16 Bit (0 bis 65.535) pro Bildpunkt dargestellt.

Um Grauwertbilder im Computer verarbeiten zu können, werden aus den rohen Bilddaten in der Regel zuerst höherwertige Merkmale berechnet. Solche Merkmale eignen sich aus verschiedenen Gründen besser zur Weiterverarbeitung, z. B., weil sie über größere Umgebungen integrieren. In diesem Kapitel werden wichtige Bildmerkmale, die auch im Rahmen der vorliegenden Arbeit betrachtet werden, kurz vorgestellt: Gradienten und Kanten (Abschnitt 1.1), Haar-Merkmale (Abschnitt 1.2), HOG-Merkmale (Abschnitt 1.3) und die sog. Census-Transformation (Abschnitt 1.4).

Wenn Farbbilder betrachtet werden, wird typischerweise die Darstellung im RGB-Farbraum verwendet. Jedem Pixel werden dann drei Werte für die drei Farbkanäle rot, grün und blau zugewiesen. Während diese Repräsentation am weitesten verbreitet ist, sind für die digitale Verarbeitung von Bildern andere Farbräume häufig besser geeignet, entsprechende Ansätze werden in Abschnitt 1.5 vorgestellt.

In Abschnitt 1.6 werden schließlich kurz Verfahren vorgestellt, die es erlauben, die Ähnlichkeit von zwei Bildern zu beurteilen. Dies kann nützlich sein, um bekannte starre Objekte in Bildern zu finden.

1.1. Gradienten und Kanten

Gradienten sind einfach zu berechnende lokale Merkmale in Bildern. Der horizontale Gradient g_x an der Position (x, y) im Bild I ist definiert als

$$g_x(x, y) = I(x - 1, y) - I(x + 1, y), \quad (1.1)$$

der vertikale Gradient g_y analog als

$$g_y(x, y) = I(x, y - 1) - I(x, y + 1). \quad (1.2)$$

Aus den Gradienten lassen sich weitere lokale Merkmale bestimmen, insbesondere die Kantenstärke und -orientierung. Die Kantenstärke $E(x, y)$ ergibt sich als Länge der Hypotenuse des Dreiecks, das durch $g_x(x, y)$ und $g_y(x, y)$ aufgespannt wird:

$$E(x, y) = \sqrt{g_x^2(x, y) + g_y^2(x, y)}. \quad (1.3)$$

Die Richtung der Kante $\alpha(x, y)$ kann berechnet werden als

$$\alpha(x, y) = \arctan \frac{g_y(x, y)}{g_x(x, y)}, \quad (1.4)$$

wenn $g_x(x, y) \neq 0$. Abbildung 1.1 zeigt ein Grauwertbild sowie ein aus den Kantenstärken E berechnetes Kantenbild.



Abbildung 1.1.: (a) Kamerabild (8 Bit).
(b) Kantenbild zu (a) – alle Bildpunkte (x, y) mit Kantenstärke $E(x, y) > 100$, siehe Gleichung (1.3).

Anstelle der in Gleichung (1.1) bzw. Gleichung (1.2) angegebenen Formeln werden teilweise auch größere Filter verwendet um die Robustheit gegenüber Rauschen (hohe Frequenzen) zu erhöhen. Gut geeignet zum Berechnen von Kantenbildern ist z. B. der Algorithmus von Canny (1986).

1.2. Haar-Merkmale

Haar-Merkmale sind allgemeine Kantenfilter, die sich besonders effizient berechnen lassen. Sie haben große Popularität erlangt nach einer Veröffentlichung von Viola



Abbildung 1.2.: Grundtypen von Haar-Merkmalen.

und Jones (2001). In Verbindung mit einem entsprechenden Lernverfahren (siehe Abschnitt 2.3) wurden dort herausragende Ergebnisse bei der echtzeitfähigen Detektion von Gesichtern in Videobildern erzielt.

Die Grundlage für die effiziente Berechnung von Haar-Merkmalen bildet das sogenannte Integralbild (engl. *integral image*), das prinzipiell vorher schon unter dem Namen *summed area table* bekannt war (Crow, 1984). Für ein Kamerabild I ist das zugehörige Integralbild II folgendermaßen definiert:

$$II(x, y) = \sum_{i=0, \dots, x} \sum_{j=0, \dots, y} I(i, j). \quad (1.5)$$

Das heißt, im Integralbild steht an jeder Stelle $II(x, y)$ die Summe der Helligkeiten aller Pixel im Eingabebild, die im Rechteck links oberhalb von (x, y) enthalten sind.

Das Integralbild kann bei einem einzigen Durchlauf durch ein gegebenes Kamerabild berechnet werden. Dafür wird auf bereits vorhandene Ergebnisse zurückgegriffen:

$$II(x, y) = II(x - 1, y) + II(x, y - 1) - II(x - 1, y - 1) + I(x, y). \quad (1.6)$$

Dieses rekursive Vorgehen macht eine spezielle Randbehandlung erforderlich, für $x' < 0$ oder $y' < 0$ kann einfach $II(x', y') = 0$ angenommen werden.

Ein Haar-Merkmal wird definiert durch eine Menge von Rechtecken (normalerweise zwei bis vier) im Kamerabild, wobei jedes dieser Rechtecke einer von zwei Klassen zugeteilt wird: „weiß“ oder „schwarz“. Die Antwort des Merkmals ist die Differenz der durchschnittlichen Helligkeiten aller Pixel in „weißen“ Regionen und der durchschnittlichen Helligkeit aller Pixel in „schwarzen“.

Es lassen sich verschiedene Typen von Haar-Merkmalen angeben, die sich darin unterscheiden, für welche Strukturen im Bild (Kanten, Linien, usw.) sie jeweils ihre betragsmäßig größten Antworten erreichen. Abbildung 1.2 zeigt einige Grundtypen von Haar-Merkmalen: Die ersten beiden dargestellten Merkmale erreichen bspw. ihre maximale Antwort, wenn das Eingabebild an der entsprechenden Position eine horizontale bzw. vertikale Kante enthält. Für alle dargestellten Merkmale gilt, dass ihre Antwort 0 ist, wenn sie in einem homogenen Bildbereich (Bereich mit einheitlicher Helligkeit) berechnet werden. In Abbildung 1.3 sind die Antworten der Grundtypen für dasselbe Eingangsbild dargestellt.

Mithilfe des Integralbildes lassen sich die Antworten von Haar-Merkmalen sehr effizient bestimmen: Um die Summe der Helligkeiten innerhalb eines Rechtecks zu berechnen, sind genau vier Zugriffe notwendig. Dabei wird derselbe Rechenweg benutzt wie auch beim Erstellen des Integralbildes, vgl. Gleichung (1.5). Konkret ist für eine rechteckige Fläche mit den Eckpunkten (x_1, y_1) und (x_2, y_2) mit $x_1 \leq x_2$ und $y_1 \leq y_2$ die Summe aller Helligkeiten $S(x_1, y_1, x_2, y_2)$ im Kamerabild über das zugehörige Integralbild II also zu berechnen als

$$S(x_1, y_1, x_2, y_2) = II(x_2, y_2) - II(x_1, y_2) - II(x_2, y_1) + II(x_1, y_1). \quad (1.7)$$

Da Rechtecke in den Grundtypen verschiedene Eckpunkte teilen, sind in den Beispielen aus Abbildung 1.2 für die Berechnung der endgültigen Merkmalsantwort höchstens zehn Zugriffe nötig.

Das Konzept lässt sich vielfältig erweitern, so wurden verschiedene Möglichkeiten vorgeschlagen, gedrehte Merkmale zu berechnen, um bspw. diagonale Kanten im Bild einfacher finden zu können (Lienhart und Maydt, 2002). Von Derpanis u. a. (2007) wurde gezeigt, dass Integralbilder höherer Ordnung dazu geeignet sind, die Berechnung komplexerer Operationen effizient durchzuführen, so z. B. die Berechnung von

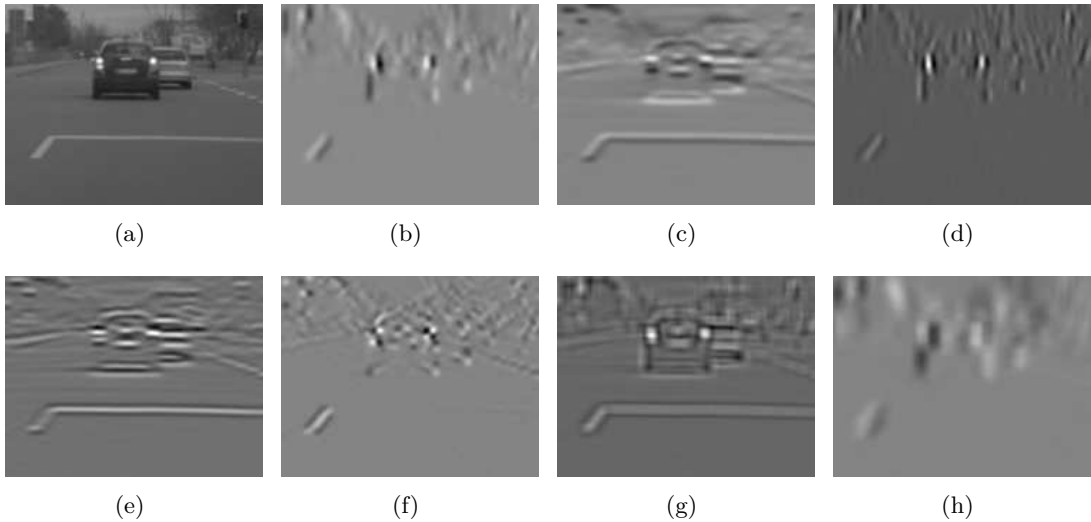


Abbildung 1.3.: (a) Ausschnitt aus einem Kamerabild.
 (b)–(g) Antworten der Grundtypen von Haar-Merkmalen in der Reihenfolge aus Abb. 1.2, alle Merkmale mit einer Größe von 8×8 Pixel.
 (h) Antworten des ersten Merkmals aus (b), hier jedoch mit Größe 16×16 Pixel.

interest points. Die Idee des Integralbildes kann auch genutzt werden, um Histogramme effizient zu berechnen, siehe z. B. Villamizar u. a. (2006).

Da sich Antworten von Haar-Merkmalen mithilfe des Integralbildes bereits sehr einfach und schnell berechnen lassen, gibt es wenig Arbeiten, die sich mit der effizienten Implementierung beschäftigen. Wesierski u. a. (2012) schlagen jedoch ein Verfahren vor, um große Merkmalsätze im selben Bildbereich noch effizienter auszuwerten.

1.3. HOG-Merkmale

Kanten spielen eine zentrale Rolle beim menschlichen Sehen, z. B. für das Erkennen von Objekten. In Zeichnungen (z. B. Karikaturen oder Comics) können wenige Striche ausreichen, um Gegenstände und Personen eindeutig darzustellen. Eine einzelne Kante ist jedoch noch wenig aussagekräftig – es ist die Verteilung von Kanten innerhalb eines Bildausschnitts, d. h., ihre wechselseitige Anordnung, die ein Objekt in einem Bild erkennbar macht.

Das in Abschnitt 1.1 beschriebene Vorgehen der Gradientenberechnung erlaubt es, die Punkte im Bild zu bestimmen, die zu Kanten gehören können. Zusätzlich können diesen Kantenpunkten jeweils auch Richtungen zugeordnet werden. Darauf aufbauend haben Dalal und Triggs (2005) ein Verfahren vorgeschlagen, um diese Merkmale so zu repräsentieren, dass sie besser für die weitere Verarbeitung geeignet sind. Es ist unter der Bezeichnung HOG (*histogram of orientated gradients*) sehr bekannt und wurde für diverse Anwendungen erfolgreich eingesetzt (s. u.).

Ein relevanter Bildausschnitt wird zunächst in nicht überlappende *Zellen* unterteilt. Eine typische Größe für diese Zellen ist etwa 5×5 Pixel. Für jede Zelle wird ein Histogramm über dort vorkommende Kantenrichtungen erstellt. Dafür wird zunächst für jeden Bildpunkt in der Zelle die lokale Kantenrichtung und -stärke berechnet nach Gleichung (1.3) und (1.4). Die Richtungen müssen für das Eintragen in das Histogramm diskretisiert werden, dafür können häufig sehr grob Einteilungen gewählt werden, bspw. acht oder 16 verschiedene Richtungen. Die diskrete Kantenrichtung jeden Bildpunkts wird, gewichtet mit der Kantenstärke, in das Histogramm übernommen. Dabei kann die Position des Pixels innerhalb der Zelle berücksichtigt werden, um eine differenziertere Gewichtung vorzunehmen, vgl. Dalal und Triggs (2005).

Im Anschluss werden Statistiken für sogenannte *Blöcke* berechnet. Ihre Größe ist ein Vielfaches der Zellengröße (häufig 2×2 Zellen). Die Blöcke werden typischerweise im gesamten Bild und überlappend berechnet, d. h., jede Zelle trägt zur Berechnung mehrerer Blöcke bei. Zunächst werden die Histogramme aller Zellen innerhalb eines Blocks konkateniert, danach wird eine Normierung durchzuführen. Häufig wird das

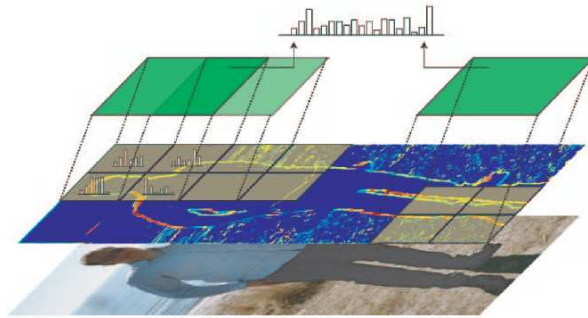


Abbildung 1.4.: Berechnung von HOG-Merkmalen: Aus dem Eingabebild (unten) werden zunächst Kantenrichtungen und -stärken berechnet (zweite Ebene). Das Bild wird in Zellen eingeteilt, in denen jeweils Histogramme für die Verteilung von Kanten berechnet werden. Die Ergebnisse mehrerer benachbarter Zellen werden jeweils in Blöcken zusammengefasst (dritte Ebene, grün dargestellt). Blöcke werden ebenfalls im ganzen Bild berechnet. Die endgültige Histogramm-Darstellung ergibt sich aus den normierten Histogrammen aller betrachteten Blöcke (oben). Für weitere Details siehe Text. Quelle: Abbildung ausENZweiler und Gavrilu (2009).

Block-Histogramm so normiert, dass die Summe aller Einträge 1 entspricht. Es ist bemerkenswert, dass dabei Information über die absoluten Kantenstärken innerhalb der Zellen verworfen wird. Umgekehrt wird so eine wichtige Grundlage dafür geschaffen, dass HOG-Merkmale sehr robust z. B. gegenüber Helligkeits- und Kontrastunterschieden in Bildern sind.

Schließlich werden die Histogramme aller Blöcke konkateniert. Der resultierende Vektor ist das endgültige Ergebnis der Merkmalsberechnung mittels HOG, er stellt eine recht kompakte Beschreibung des Bildes dar. Abbildung 1.4 zeigt das gesamte Vorgehen schematisch.

Über das hier geschilderte Grundprinzip hinaus wurden von Dalal und Triggs (2005) bereits zahlreiche Varianten untersucht, darunter nicht-quadratische Zellen, verschiedene Normierungsfunktionen, usw.

Außer zur Erkennung von Fußgängern konnte mit HOG-Merkmalen im Kontext von FAS auch sehr gute Ergebnisse bei der Erkennung unterschiedlicher anderer Objekte, z. B. Fahrzeuge (Arrospide u. a., 2012) und Verkehrszeichen (Overett und Petersson, 2011, Zaklouta und Stanciulescu, 2011a) erzielt werden.

1.4. Census-Transformation

Die Census-Transformation wurde von Zabih und Woodfill (1994) eingeführt. Das Verfahren stellt eine einfache Möglichkeit dar, für einen Bildausschnitt Merkmale zu berechnen, die invariant gegen Helligkeitsunterschiede sind.

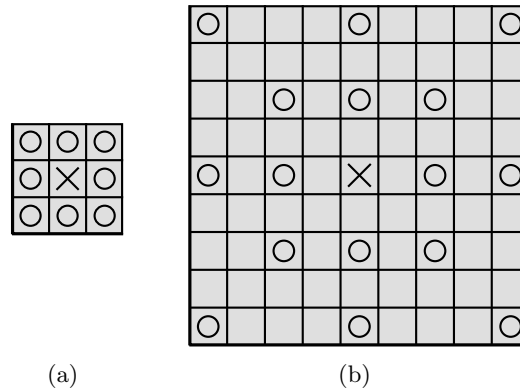


Abbildung 1.5.: (a) Pixel, die bei der ursprünglichen Census-Transformation betrachtet werden: Zentrumpixel P_0 (Kreuz) und P_1, \dots, P_8 aus 8er-Nachbarschaft (Kreise).

(b) Pixel, die bei der Census-Transformation von Stein (2004) betrachtet werden: Zentrumpixel und 16 Nachbarn.

Die Helligkeit eines festen Bildpunkts wird mit denen anderer Punkte aus seiner Nachbarschaft verglichen. Die Anzahl der beim Vergleich betrachteten Pixel sowie deren Positionen relativ zum Zentrum sind frei wählbare Parameter. Zwei Varianten sind in Abbildung 1.5 dargestellt.

Die Ergebnisse der Helligkeits-Vergleiche werden bei der ursprünglich vorgeschlagenen Census-Transformation in einer Binärzahl ξ mit N Stellen kodiert. Die Belegung der i -ten Stelle mit $1 \leq i \leq N$ hängt vom Zentrumpixel P_0 und genau einem der Nachbarpixel P_i ab:

$$\xi_i = \begin{cases} 0, & P_i > P_0 \\ 1, & P_i \leq P_0 \end{cases} \quad (1.8)$$

Das Ergebnis der Transformation kann somit als *Hash*-Wert interpretiert werden, also als Wert, der die „Charakteristik“ eines Bildausschnitts komprimiert darstellt. Werden acht Pixel aus der Nachbarschaft zum Vergleich herangezogen, gibt es 8192 mögliche Census-Werte gegenüber mehr als 10^{20} möglichen Bildausschnitten in einem 8-Bit-Grauwertbild. Aus dem Hash-Wert kann die ursprüngliche Information (das Bild) nicht

wiederhergestellt werden. Die Änderungen, die nicht rekonstruiert werden können, z. B. absolute Helligkeiten, sind genau solche, gegen die die Census-Transformation invariant ist. Die Transformation ist außerdem relativ robust gegen lokale Störungen wie Kamera-Rauschen.

Als mögliche Erweiterung der Census-Transformation wurde vorgeschlagen, einen dritten Zustand beim Vergleich zu berücksichtigen, für den Fall, dass die betrachteten Pixel ähnliche Helligkeiten haben, vgl. Stein (2004). Es wird ein neuer Parameter ϵ eingeführt, der als Schwellwert verwendet wird:

$$\xi_i = \begin{cases} 0, & P_0 - P_i > \epsilon \\ 1, & |P_0 - P_i| \leq \epsilon \\ 2, & P_i - P_0 > \epsilon \end{cases} \quad (1.9)$$

Die Census-Transformation eignet sich aufgrund ihrer einfachen Umsetzbarkeit und der geringen Rechenzeit gut für echtzeitfähige Applikationen. Sie wurde u. a. erfolgreich zum Schätzen von optischem Fluss (vgl. Kapitel 4) und zur Stereo-Verarbeitung eingesetzt, bspw. von Humenberger u. a. (2010a,b).

1.5. Farbräume

Farbbilder werden häufig im RGB-Format repräsentiert, wobei die Abkürzung für Rot / Grün / Blau steht. Bereits bei der Aufnahme von Farbbildern werden entweder sog. 3-Chip-Systeme verwendet, die für jeden Bildpunkt separat den Rot-, Grün- und Blauanteil registrieren oder kostengünstigere 1-Chip-Lösungen, bei denen für jeden Punkt nur eine der drei Komponenten aufgenommen wird – die übrigen Werte müssen dann später interpoliert werden. Bei der Anzeige am Bildschirm werden die Farben aus den drei Komponenten rot, grün und blau wieder gemischt.

Die Wahl der drei Bereiche „rot“, „grün“ und „blau“ aus dem Spektrum des für Menschen sichtbaren Lichts ist wohl motiviert durch das menschliche Sehen: In der Netzhaut finden sich zwei verschiedene Typen von Rezeptoren, Stäbchen und Zapfen. Während die Stäbchen lediglich Helligkeit wahrnehmen, sind die Zapfen für das Farbsehen verantwortlich. Es gibt drei verschiedene Arten von Zapfen, die auf unterschiedliche Wellenlängen verschieden stark ansprechen. Ihre jeweiligen maximalen Antworten liegen in den drei genannten Bereichen.

Während sich die RGB-Repräsentation im Laufe der Evolution offensichtlich als optimal erweisen hat, ist die Darstellung im RGB-Farbraum für die digitale Verarbeitung von Farbbildern nicht optimal geeignet: Um beispielsweise eine Aussage über die Helligkeit eines RGB-Werts zu machen, müssen alle drei Kanäle R , G und B gemeinsam

betrachtet werden, sie tragen unterschiedlich zur Intensität Y bei, die sich folgendermaßen berechnen lässt:

$$Y = 0,3 \cdot R + 0,59 \cdot G + 0,11 \cdot B. \quad (1.10)$$

Weitere relevante Eigenschaften, neben der Helligkeit, sind Sättigung und Farbton. Die Sättigung beschreibt intuitiv, wie stark sich ein Farbwert von schwarz / grau / weiß (also „unbunt“) unterscheidet. Der Farbton sollte eine Aussage darüber machen, wo innerhalb des Farbspektrums eine Farbe liegt, also zum Beispiel „im Bereich blau“ oder „zwischen gelb und grün“, usw. Auch diese Information geht aus einem RGB-Tripel nicht direkt hervor.

Es ist naheliegend, andere Farbräume zu definieren, in denen die „interessanten“ Werte (Helligkeit, Sättigung, Farbton, usw.) direkt repräsentiert werden und somit einfacher zugänglich sind. Dies ist insbesondere relevant, wenn Lernverfahren eingesetzt werden sollen, die ggf. darauf angewiesen sind, dass Abstände im Merkmalsraum eine sinnvolle Bedeutung haben, vgl. Abschnitt 1.6.

Im YUV-Farbraum, der in der Fernsehtechnik verwendet wird, ist die Helligkeitsinformation alleine im Y -Kanal repräsentiert, die Farbinformation in der U - V -Ebene. Die Helligkeit Y wird nach Gleichung (1.10) berechnet. Die Werte U und V ergeben sich aus RGB-Werten dann als

$$U = (B - Y) \cdot 0,493 \quad (1.11)$$

und

$$V = (R - Y) \cdot 0,877. \quad (1.12)$$

Der Farbton wird also dargestellt mithilfe der Differenzen des Blau- bzw. Rotanteils zur Helligkeit. Abbildung 1.6(a) zeigt einen Ausschnitt aus dem YUV-Raum.

Im HSV-Farbraum werden zur Repräsentation die folgenden drei Größen verwendet: Farbwinkel (engl. *hue*) H , Sättigung (engl. *saturation*) S , und Helligkeitswert (engl. *value*) V . Der Wert V ist definiert als

$$V = \max(R, G, B), \quad (1.13)$$

die Sättigung S wird, falls $V \neq 0$, bestimmt als

$$S = \frac{V - \min(R, G, B)}{V} \quad (1.14)$$

und 0 sonst. Die Berechnung des Farbwinkels schließlich erfordert mehrere Fallunterscheidungen, der endgültige Wert lässt sich dann aber einfach aus R , G und B berechnen, siehe z. B. Burger und Burge (2006). Abbildung 1.6(b) stellt den HSV-Raum dar.

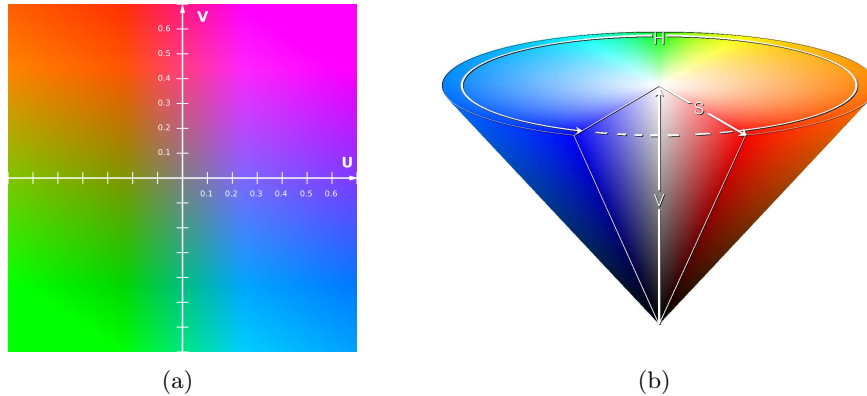


Abbildung 1.6.: (a) Ausschnitt aus dem YUV-Farbraum, U - V -Ebene für $Y = 0,5$.
Quelle: Wikimedia Commons.
(b) Schematische Darstellung des HSV-Farbraums. Quelle: Wikimedia Commons.

1.6. Vergleich von Bildern

In vielen Anwendungsfällen der digitalen Bildverarbeitung sollen bestimmte Objekte in Bildern gefunden werden. Wenn vorab recht genau bekannt, wie diese Objekte aussehen und zudem die erwartete Perspektive eingeschränkt werden kann ist, stellt ein einfacher Vergleich mit einem oder mehreren Prototypen (engl. *templates*) ein geeignetes Verfahren dar.

Die Templates können aus Trainingsdaten gewonnen werden, indem manuell Bildausschnitte gesammelt werden. Die Ausschnitte können alle jeweils einzeln als Template verwendet werden oder zusammengefasst werden (z. B. durch Berechnen von Mittelwerten oder *Clustering*).

Das eigentliche Template-Matching in einem Kamerabild erfolgt dann durch das Verschieben eines Suchfensters (engl. *sliding-window*). Das heißt, für die Templates werden verschiedenen Positionen (und ggf. verschiedenen Skalierungen) im Bild betrachtet, für jedes mögliche Fenster wird die Ähnlichkeit bestimmt. Wenn die Ähnlichkeit einen Schwellwert überschreitet, wird angenommen, dass das gesuchte Objekt gefunden wurde. Um Objekte in verschiedenen Größen zu finden, kann das Suchfenster in seiner Größe variiert werden, für praktische Anwendungen kann es jedoch günstiger sein, das Kamerabild schrittweise zu verkleinern (sog. Bild- oder Gauß-Pyramide).

Um zwei Bilder I_1 und I_2 , beide mit Breite w und Höhe h zu vergleichen, können verschiedene Abstandsmaße verwendet werden. Für Grauwertbilder stellt die Summe

der absoluten Differenzen (engl. *sum of absolute differences*, *SAD*) eine besonders einfache Möglichkeit dar:

$$sad(I_1, I_2) = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} |I_1(x, y) - I_2(x, y)|. \quad (1.15)$$

Das Abstandsmaß *sad* ist nicht robust gegenüber Helligkeits-Unterschieden in den beiden Bildern, was für die praktische Anwendung problematisch sein kann. Eine populäre Alternative stellt die normierte Kreuzkorrelation (engl. *normalized cross-correlation*, *NCC*) dar:

$$ncc(I_1, I_2) = \frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (I_1(x, y) - \bar{I}_1)(I_2(x, y) - \bar{I}_2)}{\sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (I_1(x, y) - \bar{I}_1)^2} \cdot \sqrt{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} (I_2(x, y) - \bar{I}_2)^2}}, \quad (1.16)$$

dabei bezeichnet \bar{I} die durchschnittliche Helligkeit von I .

Die oben genannten Verfahren lassen sich zum Teil auf Farbbilder verallgemeinern. Wenn andere Farbräume als RGB berücksichtigt werden (vgl. Abschnitt 1.5), ergeben sich aber noch weitere Möglichkeiten. Zum Beispiel kann im HSV- oder YUV-Farbraum der Helligkeits-Kanal ignoriert werden und für die Abstandsberechnung nur der Farbton berücksichtigt werden (z. B. Euklidische Distanz in der U - V -Ebene).

2. Objekterkennung

Viele wichtige Applikationen im Bereich videobasierter FAS lassen sich unter dem Oberbegriff Objekterkennung zusammenfassen: Das Erkennen von Fußgängern, Fahrzeugen, Verkehrszeichen, Ampeln, usw. Die genannten Aufgaben werden zwar teilweise in der Praxis recht unterschiedlich gelöst, dies spiegelt sich aber eher in der Extraktion geeigneter Bildmerkmale wieder, weniger jedoch in den eigentlichen Algorithmen für die Erkennung.

In diesem Abschnitt wird zunächst der Begriff der „Erkennung“ genauer definiert, dafür werden die Begriffe *Detektion* und *Klassifikation* eingeführt (Abschnitt 2.1). Danach werden zwei Algorithmen zur Objekterkennung vorgestellt: lineare Diskriminanzanalyse in Abschnitt 2.2 und der Viola-Jones-Detektor in Abschnitt 2.3. Beide Verfahren gehören zu den populärsten wenn Detektion und / oder Klassifikation unter Echtzeitanforderungen durchgeführt werden soll.

2.1. Detektion und Klassifikation

Beim Erkennen von Objekten in Videobildern werden häufig zwei Schritte unterschieden: *Detektion* und *Klassifikation*.

Die Aufgabe der Detektion ist, im gesamten Kamerabild alle Ausschnitte zu finden, die das gesuchte Objekt enthalten. Wenn dies mit hoher Genauigkeit gelingt, ist die „Erkennung“ schon erfolgreich geleistet falls keine weitere Unterscheidung erforderlich ist (z. B. bei Fußgängern und Fahrzeugen).

Für andere Objekte (wie Verkehrszeichen und Ampeln) lassen sich verschiedene Klassen unterscheiden (z. B. Geschwindigkeitsbegrenzungen mit unterschiedlichen Zahlen oder Ampeln in verschiedenen Phasen). Dann kann es trotzdem sinnvoll sein, bei der Detektion allgemein zu suchen (also bspw. alle runden Schilder mit rotem Rand) und erst anschließend die gefundenen Bildausschnitte in die Klassen zu trennen. Dieser Schritt wird dann als Klassifikation bezeichnet.

Die Unterscheidung zwischen Detektion und Klassifikation ist zwar in der Beschreibung entsprechender Systeme gebräuchlich, bei genauer Betrachtung aber in der Regel so nicht notwendig. Zur Detektion werden sehr häufig Suchfenster-Ansätze verwendet

(vgl. Abschnitt 1.6), dabei werden nacheinander verschiedene Bildausschnitte im Kamerabild betrachtet und jeweils entschieden, ob ein relevantes Objekt vorliegt. Damit wird hier eine (binäre) Klassifikation durchgeführt. In diesem Schritt werden tatsächlich auch häufig typische Klassifikationsverfahren eingesetzt (so z. B. lineare Klassifikatoren, vgl. Abschnitt 2.2).

Schließlich ist es möglich, auch die Detektion von Objekten, die nicht in Klassen aufgeteilt werden müssen (z. B. Fußgänger), schrittweise durchzuführen. Dafür kann zuerst ein Detektionsverfahren eingesetzt werden, das das ganze Bild besonders schnell verarbeitet und dabei wenig *Falsch-Negativ*-Ergebnisse aber verhältnismäßig viele *Falsch-Positiv*-Ergebnisse erzielt. Anschließend werden diese Ergebnisse mithilfe eines zweiten Verfahrens endgültig bewertet. Der Viola-Jones-Detektor, der hier in Abschnitt 2.3 vorgestellt wird, verfeinert dieses Vorgehen.

2.2. Lineare Diskriminanzanalyse

Bei der linearen Diskriminanzanalyse (LDA) wird angenommen, dass die Merkmale aller Klassen multivariat Gauß-verteilt sind und insbesondere, dass sie eine gemeinsame Kovarianzmatrix haben. Basierend auf diesen Annahmen können mittels LDA recht einfach trennende Hyperebenen zwischen den Klassen bestimmt werden. Diese Trennungen sind optimal unter allen linearen Klassifikatoren. Für eine detaillierte Beschreibung wird auf Hastie u. a. (2001) verwiesen. Im Folgenden wird das Verfahren kurz am Beispiel von zwei Klassen erklärt.

Das Klassifikationsproblem ist gegeben durch zwei Zufallsvariablen X und Y mit gemeinsamer Wahrscheinlichkeitsverteilung in $\mathbb{R}^n \times \{-1, +1\}$, wobei \mathbb{R}^n der Eingaberaum ist und -1 und $+1$ die *Label* der beiden Klassen (negative und positive Beispiele). Sei $P(Y = c | X = x)$ die Wahrscheinlichkeit, dass der Vektor x zur Klasse c gehört. Ein Beispiel x wird Klasse $+1$ oder -1 zugeordnet, je nachdem, ob der Quotient

$$\ln \frac{P(Y = +1 | X = x)}{P(Y = -1 | X = x)} = \ln \frac{p(X = x | Y = +1)}{p(X = x | Y = -1)} + \ln \frac{P(Y = +1)}{P(Y = -1)} \quad (2.1)$$

positiv oder negativ ist (d. h., je nachdem, welche der beiden Wahrscheinlichkeiten größer ist).

Bei der LDA wird die Annahme getroffen, dass beide betrachteten Verteilungen dieselbe Kovarianzmatrix Σ teilen (s. o.), also $p(X | Y = c) \sim \mathcal{N}(\mu_c, \Sigma)$. Unter dieser Annahme lässt sich Gleichung (2.1) folgendermaßen darstellen

$$\underbrace{\ln \frac{P(Y = -1)}{P(Y = +1)} - \frac{1}{2}(\mu_{(-)} + \mu_{(+)})^\top \Sigma^{-1}(\mu_{(-)} - \mu_{(+)})}_{b} + x^\top \underbrace{\Sigma^{-1}(\mu_{(-)} - \mu_{(+)})}_{w} \quad (2.2)$$

und führt damit zu einer einfachen Klassifikationsregel, einem linearen Klassifikator:

$$h(x) = \text{sgn}(b + x^\top w). \quad (2.3)$$

Das Trainieren solch eines Klassifikators mittels LDA erfordert Trainingsbeispiele $S = \{(x_1, y_1), \dots, (x_\ell, y_\ell)\}$ mit $(x_i, y_i) \in \mathbb{R}^n \times \{-1, +1\}$. Es sei $S^{(c)} = \{(x, y) \mid (x, y) \in S \wedge y = c\}$ und $\ell_c = |S^{(c)}|$. Die *a priori* Wahrscheinlichkeiten für beide Klassen können aus S geschätzt werden mit $P(Y = c) = \ell_c/\ell$, die Klassenzentren als

$$\mu_c = \frac{1}{\ell_c} \sum_{(x,y) \in S^{(c)}} x \quad (2.4)$$

und die Kovarianzmatrix

$$\Sigma = \frac{1}{\ell - 2} \left(\sum_{(x,y) \in S^{(-1)}} (x - \mu_{(-1)})(x - \mu_{(-1)})^\top + \sum_{(x,y) \in S^{(+1)}} (x - \mu_{(+1)})(x - \mu_{(+1)})^\top \right). \quad (2.5)$$

Die LDA zugrunde liegenden Annahmen sind in praktischen Anwendungen zwar selten (vollständig) erfüllt, trotzdem können mit dem Verfahren häufig sehr gute Ergebnisse erzielt werden (Hastie u. a., 2001). Da das Lernverfahren selbst relativ einfach ist, spielt jedoch die Repräsentation der Eingabedaten eine entscheidende Rolle. Bei Anwendungsbeispielen aus dem Bereich Bildverarbeitung heißt das konkret, dass geeignete Merkmale aus Bildern extrahiert werden müssen. Entsprechende Ansätze werden z. B. in Abschnitt 8.1 dieser Arbeit vorgestellt.

Das Training eines linearen Klassifikators nach Gleichung (2.2) erfordert insbesondere das Invertieren der Kovarianzmatrix, dies ist tatsächlich der einzige aufwendige Schritt mit Rechenzeit-Komplexität $O(n^3)$. Für den Fall, dass die LDA häufiger mit ähnlichen Daten wiederholt wird (z. B. im Zusammenhang mit Merkmalsoptimierung oder -selektion), wird hier in Kapitel 11 ein effizientes Verfahren vorgestellt.

2.3. Der Viola-Jones-Detektor

Viola und Jones (2001) haben das Verfahren vorgeschlagen, das inzwischen unter ihrem Namen zu den bekanntesten Ansätzen für echtzeitfähige Bildverarbeitung zählt. Das vorgeschlagene Detektionsverfahren verwendet Haar-Merkmale (vgl. Abschnitt 1.2) die daraufhin ebenfalls an Popularität gewannen. Der Algorithmus erlaubt es, Kamerabilder unter Echtzeitanforderungen zu verarbeiten und dabei sehr gute Ergebnisse

zu erzielen, in (Viola und Jones, 2001) beispielhaft für die Detektion von Gesichtern gezeigt.

Um den Detektor zu trainieren, wird ein Lernverfahren vorgeschlagen, das sich großteils am *AdaBoost*-Algorithmus von Freund und Schapire (1997) orientiert. Für weiterführende Details zu diesem Algorithmus wird auf Mohri u. a. (2012) verwiesen. Ziel beim *Boosting* ist es, mehrere sog. schwache Klassifikatoren aus einem *Pool* auszusuchen und so zu kombinieren, dass sie zusammen einen besseren (starken) Klassifikator bilden.

Bei der binären Klassifikation eines Beispiels x kann jeder von N verwendeten schwache Klassifikatoren h_i mit $i = 1, \dots, N$ entweder -1 oder $+1$ antworten. Im *Boosting*-Algorithmus werden diese N Antworten für die endgültige Entscheidung H gewichtet, damit ergibt sich ein linearer Klassifikator, vgl. Abschnitt 2.2:

$$H(x) = \operatorname{sgn} \left(b + \sum_{i=1}^N \alpha_i h_i(x) \right). \quad (2.6)$$

Wenn eine Menge mit Trainingsdaten $S = \{(x_1, y_1), \dots, (x_l, y_l)\}$ und ein *Pool* \mathcal{H} mit schwachen Klassifikatoren zur Verfügung stehen, löst der *Boosting*-Algorithmus beim Training drei Aufgaben: Er wählt eine geeignete Teilmenge h_1, \dots, h_N aus \mathcal{H} aus, bestimmt die Gewichte $\alpha_1, \dots, \alpha_N$ für die gewählten Klassifikatoren und setzt den *Bias* b . Abbildung 2.1 veranschaulicht das Vorgehen eines *Boosting*-Algorithmus. Das beim Training des Viola-Jones-Detektors eingesetzte Vorgehen ist in Pseudo-Code in Algorithmus 1 dargestellt.

Im Vektor w_i werden Gewichte für die Trainingsbeispiele verwaltet. Wenn in Iteration t ein neu hinzugenommener schwacher Klassifikator das Beispiel x_i korrekt klassifiziert, wird dessen Gewicht reduziert, sonst erhöht (Zeilen 17 und 18). In jeder Iteration werden die schwachen Klassifikatoren neu trainiert (Zeile 11), dabei werden jeweils die aktuellen Gewichte berücksichtigt (s. u.). Es wird dann der beste schwache Klassifikator gewählt und der Menge der verwendeten hinzugefügt (Zeilen 13 und 14). Sein Gewicht α_t (Zeile 15) hängt von seinem gewichteten Fehler ϵ_k ab – in der Praxis fällt es häufig von einer Iteration zur nächsten, das ist aber nicht notwendig.

Der Algorithmus unterscheidet sich vom klassischen *AdaBoost*-Algorithmus an verschiedenen Stellen, dazu gehört das Neu-Trainieren der schwachen Klassifikatoren in jeder Iteration sowie die Tatsache, dass jeder schwache Klassifikator auf einem einzigen Haar-Merkmal (d. h., ein Merkmalstyp an einer festen Position innerhalb des betrachteten Bildausschnitts) basiert. Aus der Antwort des Merkmals $r_i(x)$ für das

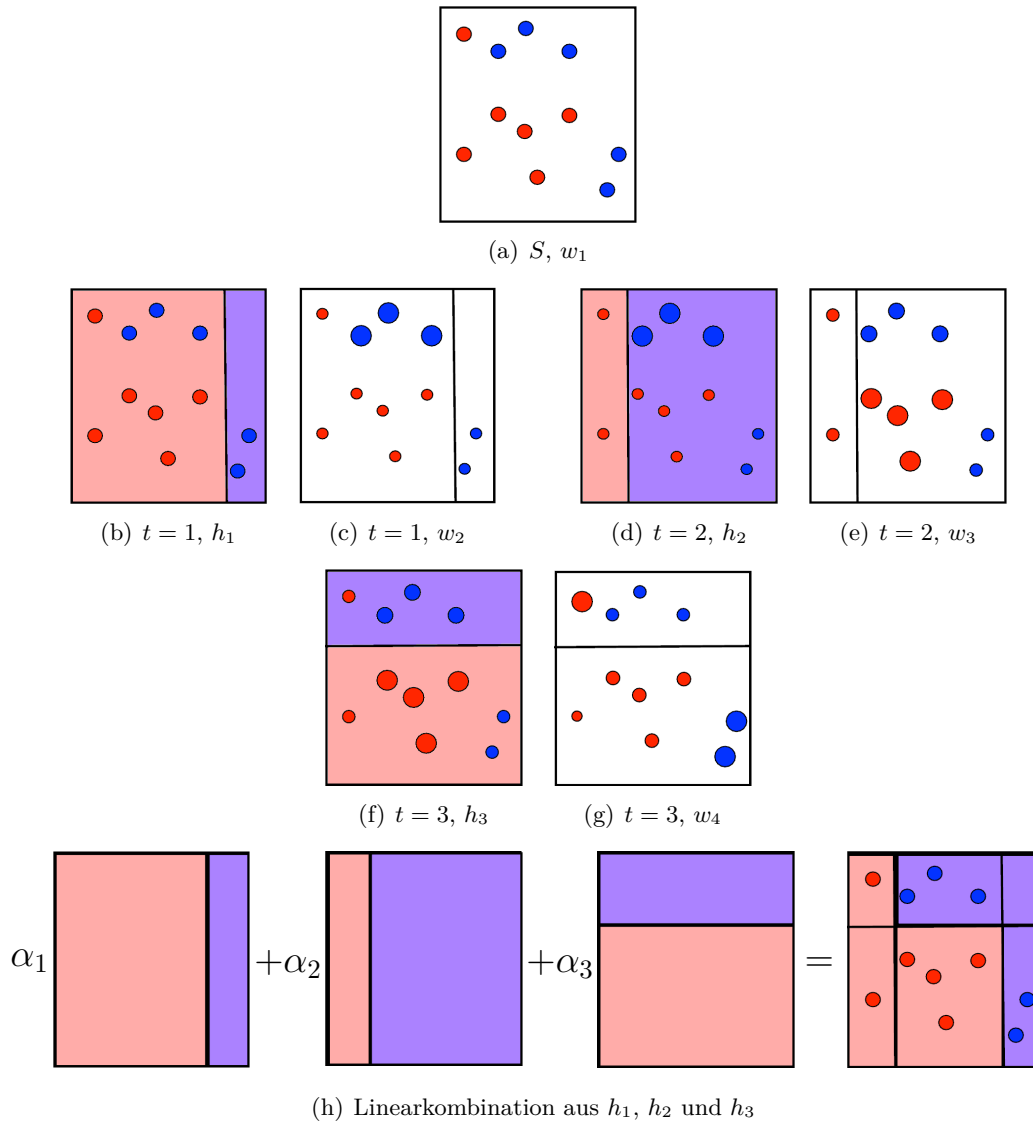


Abbildung 2.1.: Training mit *Boosting*-Algorithmus im \mathbb{R}^2 . Schwache Klassifikatoren können parallel zu den Achsen trennen. (a) Verteilung der 11 Trainingsbeispiele aus zwei Klassen (blau und rot). Die Größen der Kreise stellen die Gewichte w_i dar. (b) Die in der ersten Iteration ausgewählte Hypothese h_1 trennt 8/11 der Beispiele korrekt. (c) Neue Gewichte w_2 für die nächste Iteration. Aktuell falsch klassifizierte Beispiele erhalten höheres Gewicht. (d)–(g) Iterationen $t = 2$ und $t = 3$. (h) Endgültiger Klassifikator trennt alle Beispiele korrekt. Quelle: Abbildung verändert aus Mohri u. a. (2012).

Algorithmus 1: *Boosting-Algorithmus*, der beim Training des Viola-Jones-Detektors verwendet wird.

Eingabe :

- Trainingsdaten $S = \{(x_1, y_1), \dots, (x_l, y_l)\}$ mit $(x_i, y_i) \in \mathcal{X} \times \{-1, +1\}$
- $n = S_{(-1)}$, die Anzahl der negativen Beispiele in S
- $m = S_{(+1)}$, die Anzahl der positiven Beispiele in S
- *Pool* \mathcal{F} von schwachen Klassifikatoren $f_i : \mathcal{X} \mapsto \{-1, +1\}$, jeweils basierend auf einem Haar-Merkmal
- Gewünschte Anzahl schwacher Klassifikatoren N (oder anderes Abbruchkriterium)

Ausgabe :

- Liste der gewählten schwachen Klassifikatoren h_i mit $i = 1, \dots, N$
 - Zugehörige Gewichte α_i mit $i = 1, \dots, N$
 - *Bias* b
-

```

1 for  $i \leftarrow 1$  to  $l$  do
2   if  $y_i = -1$  then
3      $w_1(i) \leftarrow 1/n$ ;           // Gewichtung des Beispiels (Klasse -1) initial
4   else
5      $w_1(i) \leftarrow 1/m$ ;           // Gewichtung des Beispiels (Klasse +1) initial
6 for  $t \leftarrow 1$  to  $N$  do
7    $z \leftarrow \sum_{j=1}^l w_t(j)$ ;
8   for  $i \leftarrow 1$  to  $l$  do
9      $w_t(i) \leftarrow w_t(i)/z$ ;           // Gewichtung der Beispiele normieren
10  for  $i \leftarrow 1$  to  $|\mathcal{F}|$  do
11    train( $f_i, S, w_t$ );           // Training des schwachen Klassifikators
12     $\epsilon_i \leftarrow \sum_{j=1}^l w_t(j) |f_i(x_j) - y_j|$ ; // Fehler des schwachen Klassifikators
13   $k \leftarrow \arg \min_{1 \leq i \leq N} \epsilon_i$ ;
14   $h_t \leftarrow f_k$ ;           // Bester der schwachen Klassifikatoren...
15   $\alpha_t \leftarrow \log((1 - \epsilon_k)/\epsilon_k)$ ;           // ... und zugehöriges Gewicht
16  for  $i \leftarrow 1$  to  $l$  do
17     $e \leftarrow 1 - |h_t(x_i) - y_i|$ ;
18     $w_{t+1}(i) \leftarrow w_t(i)(\epsilon_k/(1 - \epsilon_k))^e$ ; // Gewichtung der Beispiele anpassen
19  $b = -\frac{1}{2} \sum_{i=1}^N \alpha_i$ 

```

Trainingsbeispiel x wird die Antwort $f_i(x)$ des schwachen Klassifikators folgendermaßen berechnet:

$$f_i(x) = \begin{cases} 1 & \text{wenn } p_i r_i(x) < p_i \theta_i \\ -1 & \text{sonst} \end{cases}. \quad (2.7)$$

Dabei ist θ_i ein Schwellwert und p_i die Parität, über die festgelegt wird, auf welchen Seiten des Schwellwerts die beiden Klassen liegen. Beim Trainieren des schwachen Klassifikators f_i in Iteration t ändern sich die Merkmalsantworten r_i nicht, da die Trainingsmenge konstant ist. Es können sich jedoch die Gewichte der Trainingsbeispiele w_t in jeder Iteration ändern. Beim Training von f_i in Iteration t werden θ_i und p_i so gewählt, dass der Fehler ϵ_i (Zeile 12) minimal wird. Damit ergeben sich im Allgemeinen in jeder Iteration unterschiedliche schwache Klassifikatoren.

Da der Ansatz von Viola und Jones sich für Echtzeitanwendungen eignen sollte, wurde nicht ein einziger starker Klassifikator mit dem oben beschriebenen *Boosting*-Algorithmus trainiert, sondern mehrere, die für jeden Bildausschnitt nacheinander ausgewertet werden. Dabei ergibt sich eine *Kaskade*: Wenn die erste Stufe (der erste starke Klassifikator) den betrachteten Bildausschnitt verwirft, also der negativen Klasse zuordnet, kann die Klassifikation sofort abgebrochen werden. Im anderen Fall wird die zweite Stufe betrachtet, die einen zusätzlichen starken Klassifikator enthält. Dieser wird bereits beim Training nur mit Beispielen trainiert, die die erste Stufe passieren. Nur wenn alle vorhandenen starken Klassifikatoren den betrachteten Bildausschnitt akzeptieren, wird er endgültig der positiven Klasse zugeordnet. Für Details zu diesem Vorgehen siehe Viola und Jones (2001).

Das Verfahren wurde erfolgreich für verschiedenste Anwendungen eingesetzt, im Rahmen der vorliegenden Arbeit wird der Viola-Jones-Detektor zum Erkennen von Verkehrszeichen verwendet, vgl. Abschnitt 8.2 und Kapitel 12.

3. Stereo-Verarbeitung

Im menschlichen Auge wird das Bild auf die Netzhaut projiziert, dabei geht die Tiefeninformation verloren. Um trotzdem Entfernungen abschätzen zu können, berücksichtigt das Gehirn (unter anderem) die Unterschiede zwischen den beiden Bildern, die von den zwei Augen gleichzeitig aus unterschiedlichen Positionen erfasst werden. Das skizzierte Vorgehen kann technisch mit einem Stereo-Kamerasystem nachgeahmt werden. Auch hier stehen in jedem Zeitschritt zwei Bilder zur Verfügung, die verglichen werden können.

Im folgenden Abschnitt 3.1 wird die Problemstellung aus technischer Sicht genauer erläutert. Danach, in Abschnitt 3.2, werden Ansätze zum Berechnen einer Kostenmatrix für zwei Eingabebilder vorgestellt. In Abschnitt 3.3 werden schließlich einige populäre Algorithmen beschrieben, die unter Echtzeitbedingungen Entfernungen aus Stereo-Bildern schätzen können.

3.1. Stereogeometrie

Mithilfe eines Stereo-Kamerasystems können Tiefen in der Welt geschätzt werden. Die beiden Kameras werden typischerweise parallel nebeneinander installiert, der Abstand zwischen ihnen wird dann als Basisbreite bezeichnet. Für praktische Anwendungen, etwa in der Robotik oder im Automobilbereich, sind Basisbreiten von 20 cm oder mehr üblich.

Wenn sich beide Kameras auf derselben Höhe befinden, exakt dieselbe Ausrichtung haben und senkrecht zu ihrem Verbindungsvektor orientiert sind, dann gilt, dass jeder Punkt in der betrachteten Szene in beiden Bildern in die gleiche Zeile projiziert wird. Im Allgemeinen unterscheidet sich die Position innerhalb der Zeile im linken und rechten Kamerabild, x_l bzw. x_r . Die Differenz der Positionen, die sogenannte Disparität $d = x_l - x_r$, erlaubt es dann, die Entfernung in der Welt z zu berechnen. Es gilt

$$z = \frac{b \cdot f}{d}, \quad (3.1)$$

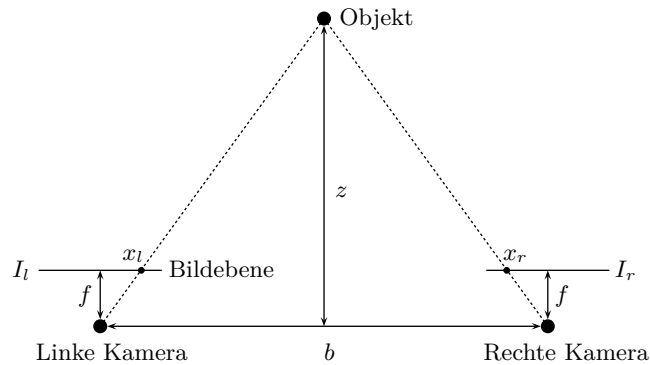


Abbildung 3.1.: Stereogeometrie: Die Disparität ergibt sich als $d = x_l - x_r$, wobei (x_l, y) und (x_r, y) der auf die linke bzw. rechte Bildebene projizierte Punkt in derselben Bildzeile y ist. Aus d , der Basisbreite b und der Fokallänge f kann die Entfernung z nach Gleichung (3.1) berechnet werden.

wobei b die Basisbreite des Stereosystems ist und f die Fokallänge der Kameras in Pixel. Abbildung 3.1 stellt diese Zusammenhänge dar.

Disparitäten in Kamerabildern können aufgrund der diskreten Auflösung zunächst nur mit Pixel-Genauigkeit geschätzt werden. Daraus ergeben sich Grenzen für die Abstandsschätzung: Für typische Größen ($f = 1000$ Pixel und $b = 20$ cm) beträgt für $d = 2$ Pixel der zugehörige Abstand $z = 100$ m, für $d = 1$ Pixel schon 200 m. Für Objekte, die sehr weit entfernt sind ($d = 0$ Pixel), kann mit Gleichung (3.1) keine Distanz berechnet werden. Umgekehrt werden für nahe Objekte die zugehörigen Disparitäten schnell sehr groß, für $z = 1$ m ergibt sich bspw. schon $d = 250$ Pixel. Die *maximale* Disparität d_{\max} , die ein Algorithmus beim Schätzen von Disparitäten berücksichtigt, bestimmt also die *minimal* erkennbare Entfernung. Weil das Vergrößern des Parameters d_{\max} i. d. R. einen negativen Einfluss auf die Laufzeit von Stereo-Algorithmen hat, ist hier sorgfältiges Abwägen erforderlich.

Abbildung 3.2 zeigt zwei Kamerabilder eines im Auto installierten Stereosystems, sie entstammen einem öffentlich verfügbaren Datensatz¹, vgl. Vaudrey u. a. (2008). Als Beispiel ist auch ein geschätztes Disparitätsbild abgebildet. Zwar sind einige Fehler deutlich zu erkennen, doch treten sie besonders in Bereichen auf, die für Anwendungen weniger relevant sind (Motorhaube, Himmel). Die Streifen-Effekte sind typisch für das hier eingesetzte Verfahren, dynamische Programmierung, dazu siehe Abschnitt 3.3.

In der Praxis ist es im Allgemeinen nicht möglich, die Kameras perfekt zu installieren und auszurichten wie oben beschrieben. Für das Stereo-System kann jedoch durch eine

¹http://www.mi.auckland.ac.nz/index.php?option=com_content&view=article&id=44



Abbildung 3.2.: (a) und (b) Linkes und rechtes Kamerabild, rektifiziert.
 (c) Disparitätsbild erzeugt mit dynamischer Programmierung, siehe Abschnitt 3.3.

einfache Kalibrierung eine Transformation ermittelt werden, die es erlaubt, die Bilder so umzurechnen, dass Gleichung (3.1) gilt. Diesen Vorgang nennt man *Rektifizierung*, für weitere Details siehe z. B. Hartley und Zisserman (2004). Die Rektifizierung kann auch *online* durchgeführt werden, siehe z. B. Hansen u. a. (2012). Im Folgenden wird davon ausgegangen, dass Stereo-Bilder rektifiziert vorliegen.

3.2. Berechnen einer Kostenmatrix

Die Grundlage für die meisten Stereo-Algorithmen bildet eine 3-dimensionale Kostenmatrix C , deren Einträge $C(x, y, d)$ aussagen, welche Kosten entstehen, wenn dem Pixel an Position (x, y) im Bild der linken Kamera I_l die Disparität d zugewiesen wird, er also dem Pixel an Position $(x - d, y)$ im rechten Bild I_r (Korrespondenzbild) zugeordnet wird.

Wenn Grauwertbilder betrachtet werden, wird die Kostenmatrix häufig basierend auf den Helligkeitsunterschieden der korrespondierenden Pixel berechnet. Im einfachsten Fall also

$$C(x, y, d) = |I_l(x, y) - I_r(x - d, y)|, \quad (3.2)$$

wobei der letzte Term hier eine Randbehandlung notwendig macht, bspw. indem für entsprechende Einträge mit $x - d < 0$ die zugehörigen Kosten auf einen festen ausreichend großen Wert gesetzt werden.

Häufig kann es sinnvoll sein, bei der Kostenberechnung nicht nur einzelne Pixel, sondern größere Nachbarschaften zu betrachten. Es kann beispielsweise die Summe der

Intensitätsdifferenzen in einem Fenster der Größe $s_x \times s_y$ um jeden Pixel herum betrachtet werden:

$$C(x, y, d) = \sum_{i=-s_x/2}^{s_x/2} \sum_{j=-s_y/2}^{s_y/2} |I_l(x+i, y+j) - I_r(x+i-d, y+j)|, \quad (3.3)$$

wobei dies wiederum eine geeignete Randbehandlung erfordert.

Methoden, die zum Schätzen der Disparität eines Pixels einen kleinen Ausschnitt um diesen Pixel herum betrachten können auch als *block matching* bezeichnet werden (Brown u. a., 2003). Die in Gleichung (3.3) angegebene Art der Kostenberechnung wird in diesem Zusammenhang als *SAD* (*sum of absolute differences*) bezeichnet. Alternative Kostenfunktionen basieren auf der Census-Transformation (siehe Abschnitt 1.4) oder der normierten Kreuzkorrelation (vgl. Abschnitt 1.6).

In der Praxis sind die genannte Kostenfunktionen teilweise nicht ausreichend: Die beiden Bilder können sich in ihren Helligkeiten recht stark unterscheiden (z. B. aufgrund unterschiedlicher Belichtungssteuerung der beiden Kameras, reflektierender Oberflächen, Schatten, etc.). Hirschmüller und Scharstein (2007) sowie Morales u. a. (2009) untersuchen die Auswirkungen auf die Ergebnisse verschiedener Algorithmen. Je nach Art der angenommenen Störung sind unterschiedliche Kostenfunktionen zu bevorzugen. Sobald es z. B. große Helligkeitsunterschiede in den beiden Bildern gibt, können *mutual information* (Hirschmüller, 2005) oder *residual images* (Vaudrey u. a., 2011) gut geeignet sein, vgl. auch Kapitel 9.

Wenn Farbbilder zur Verfügung stehen, kann bei der Kostenberechnung die zusätzliche Information aus den entsprechenden Kanälen berücksichtigt werden. Beispiele wären das Berechnen der Euklidischen Distanz in einem Farbraum wie RGB oder anspruchsvollere Abstandsfunktionen in einem anderen Farbraum wie HSV.

3.3. Echtzeitfähige Algorithmen

Das Schätzen von Tiefeninformation aus Stereo-Bildern ist seit Jahrzehnten ein lebhaftes Thema in der Literatur. Für einen guten Überblick über Stereo-Algorithmen kann bspw. auf Scharstein und Szeliski (2002, 2003) oder auf Klette u. a. (2011) verwiesen werden.

Es existieren öffentliche Benchmark-Datensätze, mit deren Hilfe Algorithmen in Bezug auf ihre Leistungsfähigkeit verglichen werden können, z. B. der populäre Middlebury-Benchmark² und Sequenzen aus dem *Enpeda*-Projekt³ von Vaudrey u. a. (2008). Die

²<http://vision.middlebury.edu/stereo>

³http://www.mi.auckland.ac.nz/index.php?option=com_content&view=article&id=44

Verfahren, die in solchen Benchmarks die besten Ergebnisse erzielen, sind häufig nicht für echtzeitfähige Anwendungen geeignet.

Im Folgenden werden kurz drei unterschiedliche Ansätze vorgestellt, die häufig für praktische Anwendungen eingesetzt werden, wenn Echtzeitanforderungen zu erfüllen sind.

Winner-takes-all Die einfachste Möglichkeit, ein Disparitätsbild D zu schätzen, besteht darin, jedem Pixel (x, y) die Disparität d zuzuordnen, die die Kosten $C(x, y, d)$ minimiert, also

$$D(x, y) = \arg \min_d C(x, y, d), \forall (x, y) \in D. \quad (3.4)$$

Das skizzierte Verfahren wird als *Winner-Takes-All (WTA)* bezeichnet. Es kann nur dann zu befriedigenden Ergebnissen führen, wenn die einzelnen Einträge der Kostenmatrix C sehr aussagekräftig sind. In Verbindung mit pixelweisen Kosten nach Gleichung (3.2) sind z. B. keine sinnvollen Ergebnisse zu erwarten.

Der WTA-Algorithmus eignet sich sehr gut für die effiziente Implementierungen auch auf einfacher Hardware. Aus diesem Grund wird die Verwendung von WTA im Kontext von Fahrerassistenzsystemen ausführlich von van der Mark und Gavrilu (2006) untersucht. In der Arbeit konnte gezeigt werden, dass die WTA-Ergebnisse für praktische Anwendungen ausreichen können, wenn einerseits mehr Aufwand bei der Kostenberechnung betrieben wird (bspw. größere Fenster im Rahmen der Kostenfunktion aus Gleichung (3.3) betrachtet werden) und andererseits Nachbearbeitungsschritte auf dem Disparitätsbild durchgeführt werden.

Dynamische Programmierung Beim hier vorgestellten Stereo-Algorithmus basierend auf dynamischer Programmierung (DP) werden die Bildzeilen unabhängig voneinander betrachtet. Es wird dann jeweils eine Lösung gesucht, die für die ganze Zeile optimal ist. Das heißt, einzelnen Punkten können – im Gegensatz zum WTA-Algorithmus – Disparität zugewiesen werden, für die diese Kosten lokal nicht minimal sind. Stattdessen versucht der DP-Algorithmus, einen glatteren Verlauf der Disparitäten zu erreichen.

Es wurden zahlreiche DP-Varianten zur Stereo-Berechnung vorgeschlagen. Die ersten Ansätze verwendeten dabei nur Kanteninformation (Ohta und Kanade, 1985), erst später wurden Kostenfunktionen basierend auf pixelweise Helligkeitsunterschieden berücksichtigt, bspw. von Geiger u. a. (1992) und Birchfield und Tomasi (1999). Der hier vorgestellte Ansatz orientiert sich an Bobick und Intille (1999).

Der DP-Algorithmus berechnet das Disparitätsbild D , das die folgende Kostenfunktion minimiert:

$$\arg \min_D R(D) = \sum_{y=0}^{h-1} \underbrace{C_M(D, y)}_{\text{data term}} + \lambda \underbrace{C_P(D, y)}_{\text{penalty term}} \quad (3.5)$$

mit

$$C_M(D, y) = \sum_{x=0}^{w-1} C(x, y, D(x, y)) \quad (3.6)$$

und

$$C_P(D, y) = \sum_{x=0}^{w-2} \lambda \cdot |D(x, y) - D(x+1, y)|. \quad (3.7)$$

Dabei ist h die Höhe des Eingabebildes und w seine Breite. Der erste Term in Gleichung (3.5), C_M , beschreibt die Gesamtkosten die sich laut Kostenmatrix für das Zuordnen der entsprechenden Pixel ergeben (*data term*). Der zweite Term, C_P , bestraft Disparitätsänderungen zwischen Nachbarpixeln (*penalty term*), dabei können prinzipiell zwei Fälle unterschieden werden (s. u.). Die Gewichtung der beiden Terme lässt sich über den Parameter λ steuern, für $\lambda = 0$ ergibt sich WTA als Spezialfall.

Bei der Optimierung können alle Zeilen unabhängig voneinander betrachtet werden – mit Blick auf eine effiziente Implementierung ist das sehr vorteilhaft, weil dadurch z. B. geringerer Speicherplatzbedarf resultiert und Möglichkeiten zur parallelen Verarbeitung ausgenutzt werden können. Bei der Minimierung von $R(D)$ ist jeweils nur die entsprechende x - d -Ebene der Kostenmatrix C relevant. Das Zuordnen der Disparitäten kann interpretiert werden als Suche eines Pfades durch diese Ebene. In der in Abbildung 3.3 gewählten Darstellung muss der Pfad vom linken zum rechten Rand verlaufen. Zusätzlich lassen sich weitere Randbedingungen formulieren. Zum Beispiel kann man annehmen, dass die Sortierung von Bildpunkten im Referenzbild und im Korrespondenzbild identisch ist (*ordering constraint*). Daraus folgt, dass für alle Paare von korrekt zugeordneten Disparitäten (x_1, d_1) und (x_2, d_2) mit $x_1 < x_2$ gelten muss $x_1 - d_1 \leq x_2 - d_2$.

Durch die Randbedingungen lässt sich die Suche nach Pfaden vereinfachen: Erlaubte Pfade in der x - d -Ebene können sich nur aus Teilen mit drei unterschiedlichen Richtungen zusammensetzen. Trotzdem ist die Anzahl aller möglichen Pfade noch exponentiell in der Zeilenlänge w , d. h., eine naive Suche führt zu Rechenzeit-Komplexität $O(d_{\max}^w)$. Der DP-Algorithmus leistet dieselbe Optimierung in $O(d_{\max} \cdot w)$.

Diese effiziente Verarbeitung wird möglich, weil es auf dem Pfad von links nach rechts für jeden Punkt (x, d) nur drei mögliche Vorgänger gibt: $(x-1, d)$, $(x-1, d-1)$ oder

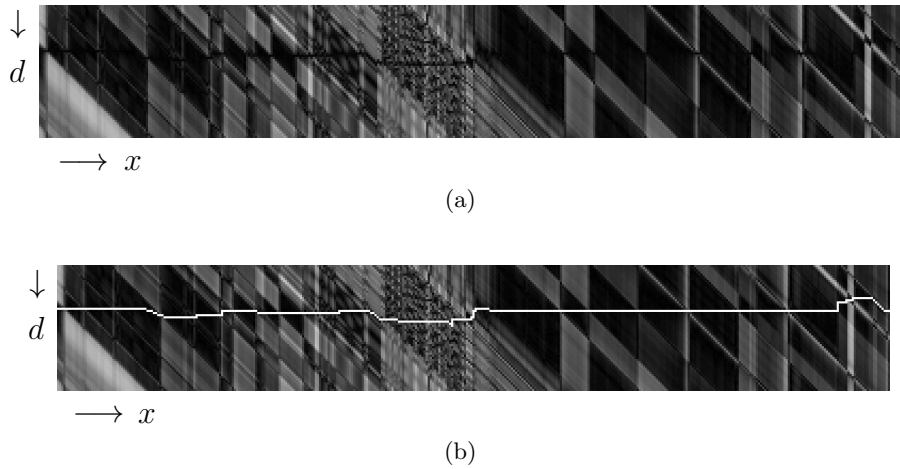


Abbildung 3.3.: (a) x - d -Ebene der Kostenmatrix C , höhere Kosten sind heller dargestellt.
 (b) Durch DP gefundener Pfad mit minimalen Gesamtkosten.

$(x, d + 1)$. Der DP-Algorithmus berechnet zuerst die Kosten der besten Pfade, die in diesen drei Punkten enden. Danach können die Kosten, um (x, d) zu erreichen, durch eine einfache Fallunterscheidung bestimmt werden: Zu den Pfadkosten des gewählten Vorgängers kommen in jedem Fall die Kosten $C(x, y, d)$ und ggf. Strafkosten, wenn sich die Disparität vom Vorgänger unterscheidet (also in zwei der drei Fälle). Der Algorithmus wählt die günstigste dieser drei Möglichkeiten.

Das eben beschriebene Vorgehen wird für alle Punkte der x - d -Ebene *vorwärts* durchgeführt, d. h., nacheinander für $x = 0, \dots, w - 1$ und für jedes x dann nacheinander für $d = 0, \dots, d_{\max}$. Danach kann im sog. *Backtracking*-Schritt der günstigste Pfad der bei $x = w - 1$ endet *rückwärts* zurückverfolgt werden. Für weitere Details wird auf Bobick und Intille (1999) verwiesen.

Bereiche, in denen der Pfad durch die betrachtete Ebene der Kostenmatrix nicht horizontal verläuft, korrespondieren zu Verdeckungen in einem der beiden betrachteten Kamerabilder. Es ist ein Vorteil des vorgestellten DP-Algorithmus, dass diese Verdeckungen erkannt werden können. Die beiden Fälle unterscheiden sich darin, ob die Disparität schrittweise geändert wird (diagonaler Pfad in der Darstellung aus Abbildung 3.3) oder Disparitätssprünge möglich sind (vertikaler Pfad in der Darstellung aus Abbildung 3.3). Daher können unterschiedliche Strafkosten P_1 und P_2 anstatt der einheitlichen Strafe λ verwendet werden, siehe auch Salmen u. a. (2009).

Dass der DP-Algorithmus alle Zeilen unabhängig voneinander behandelt, führt zu typischen Streifenwirkungen, vgl. Abbildung 3.2(c). Es gibt verschiedene Vorschläge, wie

diese verhindert werden können, z. B. DP-Varianten, die Baum-Strukturen betrachten (Veksler, 2005) und Verfahren, die einen zusätzlichen Optimierungsschritt in vertikaler Richtung durchführen (Salmen u. a., 2009). Bereits Bobick und Intille (1999) schlagen verschiedene Erweiterungen vor, z. B. sog. *ground control points*, für Details siehe dort.

Semi-Global Matching Die hier in letzten beiden Abschnitten vorgestellten Verfahren werden als lokale Algorithmen bezeichnet, weil sie zum Schätzen der Disparität eines Pixels nur jeweils eine recht begrenzte Anzahl an Einträgen aus der Kostenmatrix C berücksichtigen. Dies kann zu Störungen im Disparitätsbild führen wie bspw. den Streifeneffekten bei DP. Im Gegensatz dazu berücksichtigen globale Algorithmen beide verfügbaren Kamerabilder vollständig. Entsprechende Ansätze, etwa *graph cuts* (Zhao, 2000) oder *belief propagation* (Sun u. a., 2003), erreichen in praxisnahen Benchmarks sehr gute Ergebnisse (Klette u. a., 2011). Sie sind aber auch aufwendiger in Bezug auf die Rechenzeit und für praktische Anwendungen mit Echtzeitanforderungen daher in der Regel nicht geeignet.

Der Algorithmus *Semi-Global-Matching (SGM)*, von Hirschmüller (2005) vorgeschlagen, stellt einen guten Kompromiss zwischen lokalen und globalen Verfahren dar. Er erzeugt Disparitätsbilder ohne auffällige Störungen während er sich prinzipiell für Echtzeitsysteme eignen kann – SGM wird in heutigen Serienfahrzeugen eingesetzt.

Ziel des Algorithmus ist die Minimierung der Energiefunktion $E(D)$, die einem Disparitätsbild D Kosten folgendermaßen zuordnet:

$$E(D) = \sum_{(x,y) \in D} \left[\underbrace{C(x, y, D(x, y))}_{\text{data term}} + \underbrace{\sum_{(\hat{x}, \hat{y}) \in N(x, y)} \left(P_1 T[|D(x, y) - D(\hat{x}, \hat{y})| = 1] + P_2 T[|D(x, y) - D(\hat{x}, \hat{y})| > 1] \right)}_{\text{smoothness term}} \right] \quad (3.8)$$

Dabei ist $T[\cdot] = 1$ wenn das Argument wahr ist und sonst 0. $N(x, y)$ bezeichnet die Nachbarschaft von (x, y) . Die Funktion aus Gleichung (3.8) summiert die rohen Kosten $C(d)$ für alle durch D gewählten Disparitäten. Zusätzlich werden ggf. Strafen P_1 und P_2 addiert – dies hängt von den Disparitäten der benachbarten Pixel ab: Wenn die Disparität zweier Nachbarpixel sich um 1 unterscheidet, wird die kleinere Strafe P_1 addiert, für größere Differenzen, also Sprünge im Disparitätsbild, die größere P_2 .

Die Funktion $E(D)$ berücksichtigt bei der Summierung mit $C(x, y, d)$ einen Term, der von den Bilddaten abhängt (*data term*). Die anderen Terme bewerten die Glattheit

der Umgebung (*smoothness term*). Dieses Vorgehen ist auch in verwandten Problemstellungen erfolgreich, z. B. dem Schätzen von optischem Fluss, vgl. Kapitel 4.

Um ein bestmögliches Disparitätsbild D gemäß $E(D)$ zu erhalten, könnte eine entsprechende Minimierung durchgeführt werden. Dieses Vorgehen ist aufgrund der Rechenzeitanforderungen jedoch für Echtzeitanwendungen nicht geeignet. Der SGM-Algorithmus approximiert die Lösung, indem mehrere 1-dimensionale Pfade mit verschiedenen Orientierungen und Richtungen für jeden Pixel betrachtet werden. Die Optimierung entlang der Pfade ist dabei wenig aufwendig, sie ähnelt dem Vorgehen des im letzten Abschnitt vorgestellten DP-Algorithmus.

Gleichung (3.9) beschreibt das Vorgehen am Beispiel eines horizontalen Pfades, der in einer beliebigen Zeile y von links nach rechts durchlaufen wird:

$$E_{\text{lr}}(x, y, d) = C(x, y, d) + \min [E_{\text{lr}}(x - 1, y, d), \\ E_{\text{lr}}(x - 1, y, d - 1) + P_1, \\ E_{\text{lr}}(x - 1, y, d + 1) + P_1, \\ \min_i (E_{\text{lr}}(x - 1, y, i) + P_2)] \quad (3.9)$$

Die rekursive Funktion $E_{\text{lr}}(x, y, d)$ weist jedem Pixel an Position (x, y) auf dem horizontalen Pfad Kosten für jede mögliche Disparität d zu. In jeder Zeile des Referenzbildes ist also ein Durchlauf von $x = 0$ bis $x = w - 1$ nötig, wobei w die Breite des Bildes ist. Nach demselben Prinzip werden auch horizontale Pfade von rechts nach links durchlaufen sowie vertikale, diagonale Pfade, usw. Laut Hirschmüller (2005) sollten mindestens vier Orientierungen berücksichtigt werden.

Die endgültige, geglättete Kostenmatrix S ergibt sich durch einfaches Zusammenfassen der Kosten $E_{\mathbf{r}}(x, y, d)$ aller betrachteten Pfadrichtungen \mathbf{r} :

$$S(x, y, d) = \sum_{\mathbf{r}} E_{\mathbf{r}}(x, y, d) \quad (3.10)$$

Im letzten Schritt wird, wie beim WTA-Algorithmus, jedem Pixel (x, y) die Disparität zugeordnet, für die $S(x, y, d)$ minimal ist. Da bei der Berechnung von S recht große Bildausschnitte mit einfließen, wird eine globale Lösung approximiert – die Ergebnisse des SGM-Algorithmus nach diesem Schritt sind recht exakt. Es können aber lokale Fehler auftreten, daher wurde vorgeschlagen, eine Nachbearbeitung z. B. mit einem Median-Filter und / oder einer Links-Rechts-Konsistenzprüfung durchzuführen, für Details siehe Hirschmüller (2005).

Es wurden verschiedene Änderungen und Erweiterungen des SGM-Algorithmus vorgeschlagen. In Hirschmüller (2006) wird ein zusätzlicher Segmentierungs-Schritt eingeführt, der Verbesserungen in gut strukturierten Szenen erlaubt. Während SGM sich

3. Stereo-Verarbeitung

gut für die Implementierung auf spezieller Hardware (GPU, FPGA) eignet (Ernst und Hirschmüller, 2008, Buder, 2012), ist es schwierig, eine echtzeitfähige CPU-Implementierung zu erhalten. Zu Diskussionen bzgl. CPU-Implementierungen siehe Gehrig und Rabe (2010) und Hermann u. a. (2011).

4. Schätzen von optischem Fluss

Als optischen Fluss bezeichnet man die auf die Bildebene projizierten Vektoren, die sich aus der Bewegung von Objekten in der Szene und / oder der Bewegung der Kamera ergeben. Zwischen zwei Zeitschritten (z. B. zwei aufeinanderfolgenden Bildern) kann also theoretisch jedem Bildpunkt ein Flussvektor zugeordnet werden. In Abbildung 4.1 sind diese Vektoren für eine synthetische Sequenz veranschaulicht.

Das Ziel der in diesem Kapitel vorgestellten Algorithmen ist, Fluss-Vektoren aus Bildfolgen zu schätzen. Dabei ist beachtenswert, dass das Problem inkorrekt gestellt ist, da sich bei der Formulierung von Lösungsansätzen unterbestimmte Gleichungssysteme ergeben müssen. Dies lässt sich durch die Projektion der 3-dimensionalen Szene auf die Bildebene erklären, vgl. z. B. Bruhn und Weickert (2005).

Algorithmen zum Schätzen von optischem Fluss lassen sich grundsätzlich in zwei Kategorien einteilen: dichte Schätzung und spärliche Schätzung. Verfahren der erstgenannten Kategorie berechnen 2-dimensionale Flussvektoren für jeden einzelnen Bildpunkt, ein kurzer Überblick wird in Abschnitt 4.1 gegeben. Ansätze zur spärlichen Schätzung

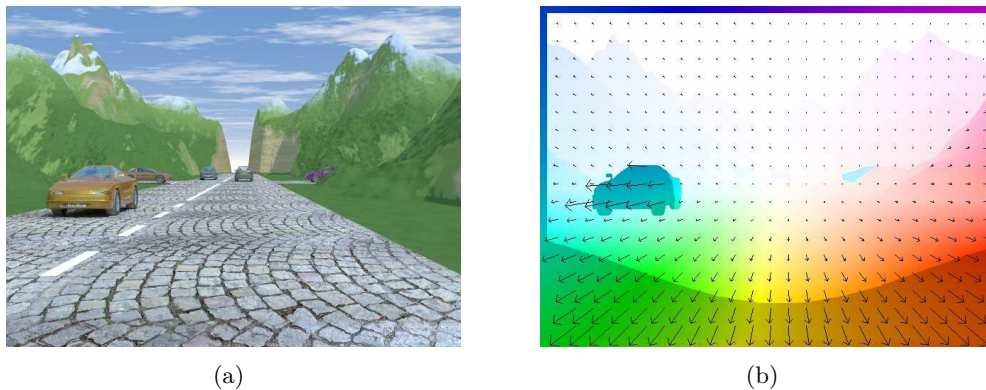


Abbildung 4.1.: (a) Beispiel-Kamerabild aus einer simulierten Sequenz, siehe Vaudrey u. a. (2008).

(b) Zugehöriger optischer Fluss, farbcodiert. Die eingezeichneten Vektoren veranschaulichen Richtung und den Betrag des optischen Fluss an den jeweiligen Punkten.

ordnen nur ausgewählten Bildpunkten Flussvektoren zu, in Abschnitt 4.2 wird mit dem sog. *PowerFlow*-Algorithmus insbesondere ein echtzeitfähiger Algorithmus vorgestellt, der in heutigen Serienfahrzeugen eingesetzt wird.

4.1. Dichte Schätzung

Hier werden kurz Algorithmen vorgestellt, die für jeden einzelnen Bildpunkt optischen Fluss schätzen, also eine sog. dichte Schätzung leisten. Das Ergebnis der Schätzung kann in zwei 2-dimensionalen Bildern u und v repräsentiert werden, die jeweils nur eine der beiden Richtungskomponente jedes Flussvektoren enthalten. Der Flussvektor zu (x, y) ergibt sich also als $(u(x, y), v(x, y))$.

Ein typischer Ansatz zum Schätzen von optischem Fluss besteht darin, eine globale Energiefunktion E zu minimieren, die z. B. nach Horn und Schunck (1981) folgendermaßen definiert werden kann:

$$\arg \min_{u,v} E(u, v) = \iint \left[\underbrace{(I_x u + I_y v + I_t)^2}_{\text{data term}} + \alpha^2 \underbrace{(\|\nabla u\|^2 + \|\nabla v\|^2)}_{\text{smoothness term}} \right] dx dy. \quad (4.1)$$

Dabei sind I_x und I_y die Ableitungen des Kamerabildes I in horizontaler bzw. vertikaler Richtung und I_t die zeitliche Ableitung.

Der erste Term in Gleichung (4.1), $(I_x u + I_y v + I_t)^2$, ergibt sich unter der Annahme, dass die Helligkeit eines Pixels sich innerhalb des betrachteten Zeitschritts nicht ändert. Er bewertet die Güte der Lösung (u, v) bzgl. der Bilddaten (*data term*). Der zweite Term bewertet die Glattheit (*smoothness term*) der Lösung, er kann mit dem Parameter α unterschiedlich stark gewichtet werden.

Der hier präsentiert Ansatz über die Energiefunktion mit zwei unterschiedlichen Teilen weist Ähnlichkeiten mit in Abschnitt 3.3 vorgestellten Ansätzen zur Stereo-Berechnung auf, siehe Gleichung (3.5) und Gleichung (3.8). Das lässt sich darauf zurückführen, dass die Probleme verwandt sind und daher ähnliche Ansätze zu erfolgreichen Lösungen führen. Interessanterweise können die beiden Probleme (Schätzen von optischem Fluss und Schätzen von Disparitäten) auch in einem gemeinsamen Ansatz betrachtet werden, für solche sog. *SceneFlow*-Algorithmen wird auf Huguet und Deverna (2007) und Wedel u. a. (2011) verwiesen.

Die aktuell erfolgreichsten Algorithmen zum Schätzen von optischem Fluss gehen auf die oben skizzierte Grundidee zurück. Diese Algorithmen sind allerdings in vielen Anwendungsfällen nicht echtzeitfähig. Es gibt außerdem typische Probleme, die bei dieser Art der Schätzung auftreten: Aufgrund der bevorzugten Glattheit wird die Bewegung kleiner Objekte häufig nicht gut erkannt. Aus ähnlichen Gründen können teilweise

verhältnismäßig große Bewegungen schwer erkannt werden. Viele Arbeiten haben zum Ziel, die Schätzung für genau solche Fälle zu verbessern, siehe z. B. Bruhn u. a. (2006), Steinbruecker u. a. (2009), Brox u. a. (2009) und Xu u. a. (2012).

4.2. Spärliche Schätzung

Algorithmen, die spärlichen optischen Fluss schätzen, bestimmen Flussvektoren nur für ausgewählte Pixel. Typischerweise werden dafür besonders „markante“ Punkte gewählt, z. B., Ecken, die sich recht gut verfolgen lassen (im Gegensatz zu Kanten, bei denen das sog. Blendenproblem (engl. *aperture problem*) auftritt). Entsprechende Ansätze verfolgen Bildpunkte aufgrund ihrer Merkmale, daher die englische Bezeichnung *feature point tracking*. Ein populäres Beispiel ist das Verfahren von Shi und Tomasi (1994).

Im Folgenden wird beispielhaft ein effizienter Algorithmus zur spärlichen Schätzung vorgestellt, der in aktuellen Serienfahrzeugen eingesetzt wird. Der sogenannte *Power-Flow*-Algorithmus, von Stein (2004) vorgeschlagen, verwendet die Census-Transformation (siehe Abschnitt 1.4) als Grundlage zum Verfolgen von Punkten.

Im ersten Verarbeitungsschritt wird für jeden Pixel im Kamerabild eine Signatur basierend auf der Census-Transformation berechnet. Stein schlägt vor, eine erweiterte Variante zu verwenden sowie größere Nachbarschaften zu betrachten, vgl. Abschnitt 1.4, Abbildung 1.5(b) und Gleichung (1.9). In seinen Experimenten betrachtet er Signaturen mit bis zu 20 Stellen, d. h., es werden nicht mehr nur direkte Nachbarpixel berücksichtigt.

Die Signaturen können wie ein *Hash*-Wert verwendet werden. Die Positionen (x, y) aller Pixel mit Signatur s im Kamerabild zum Zeitpunkt t werden in einer Tabelle $T_t(s)$ gespeichert.

Im zweiten Schritt findet der Algorithmus Hypothesen für mögliche Flussvektoren zwischen den Zeitpunkten $t-1$ und t , indem alle möglichen Paare aus $T_{t-1}(s)$ und $T_t(s)$ für alle relevanten Signaturen s (s. u.) betrachtet werden. In diesem Schritt werden zwei Einschränkungen berücksichtigt: Die Länge von Flussvektoren kann nach oben durch einen Parameter beschränkt werden. Außerdem werden nur Hypothesen berücksichtigt, bei denen die relative Helligkeitsdifferenz der beiden Zentrumspixel kleiner als ein Schwellwert ist. Dadurch werden effektiv viele falsche Hypothesen unterdrückt. Alle Hypothesen zum Zeitpunkt t werden in der Liste H_t gespeichert.

Um die Geschwindigkeit und die Robustheit des Algorithmus zu erhöhen, sollten nicht alle möglichen Signaturen bei der Verarbeitung betrachtet werden. Stein schlägt vor,

eine sog. schwarze Liste (*black-list*) zu verwenden, mithilfe derer alle Signaturen ausgeschlossen werden, die zu unbrauchbaren Bildausschnitten gehören (z. B., homogene Bereiche oder Ausschnitte, bei denen das Blendenproblem auftritt). Zusätzlich werden nur Signaturen s in der Tabelle $T_t(s)$ berücksichtigt, für die höchstens mdp (*maximum discriminative power*) Einträge vorliegen. Auch diese Heuristik kommt der Laufzeit zugute, da die Anzahl der möglichen Fluss-Hypothesen exponentiell in der Zahl der Einträge wächst.

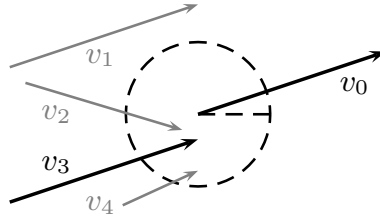


Abbildung 4.2.: Zeitliche Integration in Zeitschritt t . v_1, \dots, v_4 aus H_{t-1} sind mögliche Vorgänge für v_0 aus H_t . Der Vektor v_1 ist außerhalb des Suchradius, v_2 hat eine zu unterschiedliche Orientierung und v_4 eine zu große Längendifferenz. v_3 ist der einzige zulässige Vorgänger.

Die Hypothesen, die zwischen zwei Bildern berechnet werden, sind im Allgemeinen nicht eindeutig und nicht zuverlässig genug. Um zum Zeitpunkt t endgültige Flussvektoren zu berechnen, wird eine zeitliche Integration mithilfe von H_t, H_{t-1}, \dots durchgeführt. Für jede Hypothese h zum Zeitschritt t wird überprüft, ob bereits zum Zeitschritt $t - 1$ eine unterstützende Hypothese vorlag. Dafür wird konkret überprüft, ob eine Hypothese mit ähnlicher Länge und ähnlicher Orientierung ungefähr da endet, wo h beginnt. Abbildung 4.2 veranschaulicht dieses Vorgehen. Schließlich ergeben sich die endgültigen Flussvektoren aus den Hypothesen, die mindestens p_{\min} Vorgänger haben. Für Details siehe Stein (2004).

5. Evolutionäre Optimierung

Die natürliche Evolution hat sehr gute Lösungen für unterschiedlichste Probleme hervorgebracht (z. B., Wahrnehmung, Informationsverarbeitung, Fortbewegung, usw.). Ein Ergebnis der Evolution ist das menschliche Gehirn, das selbst dazu in der Lage ist, vielfältige Herausforderungen zu bewältigen. Trotzdem ist es in vielen Bereichen mit den heutigen technischen Lösungen noch nicht möglich, dieselben Leistungen zu erzielen, die die Evolution ermöglicht hat (s. o.).

Evolutionäre Algorithmen (EA) bilden Prinzipien der natürlichen Evolution nach, um so Optimierungsprobleme im Rechner iterativ zu lösen. Sie werden bspw. dann eingesetzt, wenn keine analytische Beschreibung vorliegt und keine anderen Verfahren (wie etwa Gradientenabstieg) verwendet werden können.

Die Verfahren, die EA zugerechnet werden, weisen zum Teil große Ähnlichkeiten auf und können nicht immer eindeutig voneinander abgegrenzt werden. Die sog. Evolutionsstrategien (ES) bilden, neben z. B. evolutionärer Programmierung und genetischen Algorithmen, eine wichtige Teilmenge der EA.

Im folgenden Abschnitt 5.1 werden die klassischen $(\mu/\rho \dagger \lambda)$ -ES beschrieben. Danach wird in Abschnitt 5.2 die Kovarianzmatrix-Adaptation vorgestellt, die eine moderne ES darstellt. In Abschnitt 5.3 wird schließlich erläutert, wie die Selektion von Lösungen im Rahmen evolutionärer Algorithmen angepasst werden kann, wenn mehrere Ziele gleichzeitig optimiert werden sollen.

5.1. Die $(\mu/\rho \dagger \lambda)$ -ES

Die hier vorgestellten Ansätze wurden von Rechenberg und Schwefel in den 70er Jahren entwickelt (Rechenberg, 1973, Schwefel, 1977). Evolutionsstrategien sind vielseitig einsetzbar und gleichzeitig verhältnismäßig einfach zu implementieren.

Sei $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ein Funktion ohne Nebenbedingungen, die maximiert werden soll. Gesucht ist also $x^* = \arg \max_{x \in \mathbb{R}^n} f(x)$. Evolutionäre Algorithmen werden typischerweise dann angewendet, wenn f nicht analytisch beschrieben ist, aber Auswertungen an einzelnen Stellen durchgeführt werden können (sog. *Black-Box*-Optimierung).

Evolutionstrategien beruhen auf der Betrachtung von Populationen mit einem oder mehreren Individuen. Ein Individuum repräsentiert eine mögliche Lösung für das betrachtete Problem, also einen Vektor $x \in \mathbb{R}^n$. Jedem Individuum x kann sein sog. Fitnesswert $f(x) \in \mathbb{R}$ zugeordnet werden. Die gesamte Population in Iteration t wird als eine Generation bezeichnet.

Die ES beruhen darauf, nach Auswertung der Fitness aller Individuen in Generation t neue Individuen zu generieren und dann aus der Menge der Eltern und/oder Nachkommen Individuen für die nächste Generation $t + 1$ auszuwählen. Die Selektion in ES erfolgt deterministisch, d. h., es wird zuerst das Individuum mit der höchsten Fitness gewählt, dann das Individuum mit der zweit-höchsten Fitness, usw. Damit unterscheiden sich ES von anderen EA, z. B. genetischen Algorithmen, bei denen die Selektion randomisiert erfolgt und die Wahrscheinlichkeit, ein bestimmtes Individuum zu selektieren, bspw. proportional zu seiner Fitness sein kann.

Beim Erzeugen von Nachkommen werden vorhandene Lösungen zufällig verändert (Mutationen). Um eine Lösung x zu mutieren, werden bei der ES alle Komponenten x_i unabhängig voneinander verändert. Dies geschieht durch komponentenweise Addition von normalverteilten Zufallsvariablen mit Erwartungswert 0 und Standardabweichung σ_i :

$$x^{(t+1)} = x^{(t)} + z, \quad z_i \sim \mathcal{N}(0, \sigma_i^2) \quad \text{für } i = 1, \dots, n \quad (5.1)$$

Optional kann bei der Reproduktion auch die Rekombination von zwei Individuen x und y berücksichtigt werden (s. u.). Diese kann entweder diskret oder intermediär erfolgen. Im ersten Fall wird für jede Komponente der neu zu erzeugenden Lösung z zufällig entschieden, welcher der beiden möglichen Werte übernommen wird (also entweder $z_i = x_i$ oder $z_i = y_i$). Im zweiten Fall wird für jede Komponente die durchschnittliche Lösung ermittelt (also $z_i = (x_i + y_i)/2$). Dieses Vorgehen lässt sich leicht für mehr als zwei Eltern verallgemeinern. Die Wahrscheinlichkeit, ein bestimmtes Individuum als Elter zu berücksichtigen, ist in den klassischen ES für alle Individuen gleich, also insbesondere nicht abhängig von seiner Fitness.

Die von ES durchgeführte Optimierung erfolgt iterativ. Zu Beginn werden μ Individuen zufällig erzeugt, sie bilden die erste Eltern-Generation. Danach werden λ Nachkommen erzeugt, dazu wird jeweils zuerst aus ρ Individuen der Eltern-Generation rekombiniert und anschließend mutiert. Danach erfolgt die Selektion der μ Individuen, die die nächste Eltern-Generation bilden. Dafür können entweder nur die letzten λ Nachkommen berücksichtigt werden oder die Vereinigung der beiden Mengen (letzte Eltern und Nachkommen). Nach Selektion der neuen Eltern kann der eben beschriebene Prozess mit der Erzeugung der nächsten Nachkommen erneut durchgeführt werden (nächste

Generation). Der Algorithmus stoppt, sobald ein Kriterium erfüllt (z. B. ausreichend gute Lösung oder maximale Anzahl Generationen).

Alle oben geschilderten Varianten von ES lassen sich in der Notation $(\mu/\rho \ddagger \lambda)$ darstellen. Dabei ist μ die Anzahl der Eltern-Individuen in jeder Generation (Populationsgröße) und λ die Anzahl der Nachkommen in jeder Generation. Der Parameter ρ gibt an, wie viele Eltern beim Erzeugen eines Nachkommens berücksichtigt werden. Wenn hier kein Wert angegeben wird gilt $\rho = 1$. Schließlich gibt entweder das Plus-Zeichen oder das Komma die Strategie an, die zur Auswahl der in jeder Generation überlebenden Individuen verwendet wird: Bei der Plus-Strategie werden alle Individuen betrachtet, bei der Komma-Strategie nur die Nachkommen. Für weitere Details siehe bspw. Beyer und Schwefel (2002).

Eine wichtige Ergänzung der oben beschriebenen ES sind Möglichkeiten zur Strategieadaptation. Bei der in Gleichung (5.1) angegebenen Mutation ist die Standardabweichung σ_i für jede Dimension fest vorgegeben. Dagegen wird bei der sog. σ -Adaptation, die von Rechenberg für den (1+1)-ES entwickelt wurde, eine (für alle Dimensionen gleiche) Mutationsstärke σ während der Optimierung dynamisch angepasst. Dafür kann z. B. die bekannte 1/5-Regel angewendet werden: Diese Heuristik besagt, dass σ vergrößert werden sollte, wenn der Anteil der erfolgreichen Mutationen an den insgesamt durchgeführten in einem festen Zeitabschnitt größer als 1/5 ist. Liegt der Anteil der erfolgreichen Mutationen dagegen unter 1/5, sollte σ verkleinert werden. Der Wert 1/5 ergibt aus der Betrachtung zweier Testfunktionen (Korridor- und Kugelmodell), bei denen er zur größtmöglichen Konvergenzgeschwindigkeit führt (Schwefel, 1977).

Es wurden zahlreiche weitere Möglichkeiten zur Mutationsweitensteuerung vorgeschlagen, darunter auch solche, wo die sog. Strategieparameter in jedem Individuum als zusätzliche Parameter mit optimiert werden. Die im folgenden Abschnitt vorgestellte CMA-ES verwendet eine weiterentwickelte Art der Anpassung.

5.2. Kovarianzmatrix-Adaptation: CMA-ES

Die Kovarianzmatrix-Adaptation (engl. *Covariance Matrix Adaption, CMA*) wurde von Hansen und Ostermeier (1996) vorgeschlagen und danach stetig weiterentwickelt. Sie stellt eine moderne ES und gleichzeitig den heutigen Stand der Technik in diesem Bereich dar¹. Die folgenden Erklärungen orientieren sich an Hansen und Ostermeier (2001).

¹ „CMA-ESs represent the state-of-the-art in evolutionary optimization in real-valued \mathbb{R}^n search spaces.“ (Beyer, 2007)

Die λ Nachkommen in Zeitschritt $t + 1$, $x^{(t+1)}$, werden in der CMA-ES basierend auf der Population aus Zeitschritt t folgendermaßen erzeugt:

$$x_k^{(t+1)} \sim \mathcal{N} \left(\langle x \rangle_w^{(t)}, \sigma^{(t)^2} C^{(t)} \right), k = 1, \dots, \lambda. \quad (5.2)$$

Dabei bezeichnet $\langle x \rangle_w^{(t)}$ den gewichteten Schwerpunkt der Generation t , der berechnet wird als

$$\langle x \rangle_w^{(t)} = \sum_{i=1}^{\mu} w_i x_i^{(t)}, \quad (5.3)$$

wobei $x_i^{(t)}$ das i -te beste Individuum der Generation t ist und die Gewichte w_i feste Parameter mit Summe 1 sind (z. B. $w_i = 1/\mu$ für $i = 1, \dots, \mu$).

Die Selektion zur Reproduktion wird also realisiert, indem bei der Schwerpunktsberechnung nur die besten μ von λ Individuen berücksichtigt werden. Aus diesen Individuen wird durch Rekombination das einzige Eltern-Individuum erzeugt, aus dem alle Nachkommen generiert werden. Der klassischen ES-Notation folgend (siehe Abschnitt 5.1) kann die Strategie also als $(\mu/\mu, \lambda)$ -CMA-ES bezeichnet werden.

Die Kovarianzmatrix C und der Wert σ in Gleichung (5.2) bestimmen die Richtungen und Weite der Mutationen. Eine besondere Eigenschaft der CMA-ES ist, dass diese Werte laufend angepasst werden. Dadurch sollen erfolgreiche Mutationen verstärkt werden. Der Prozess zur Anpassung der Kovarianzmatrix C wird im Folgenden skizziert.

Betrachtet man die gewichteten Schwerpunkte in zwei aufeinanderfolgenden Generationen, beschreibt der Vektor

$$v^{(t)} = \langle x \rangle_w^{(t+1)} - \langle x \rangle_w^{(t)} \quad (5.4)$$

die Mutationen, die erfolgreiche Nachkommen hervorgebracht haben. Der CMA-Algorithmus verwaltet, basierend auf diesen Schritten, einen sog. Evolutionspfad p_c , der durch

$$p_c^{(t+1)} = (1 - c_c) \cdot p_c^{(t)} + \sqrt{c_c(2 - c_c)} \cdot \frac{\sigma^{(g)}}{\|w\|} \cdot v^{(t)} \quad (5.5)$$

in jeder Generation angepasst wird (initial gilt $p_c^{(0)} = 0$). Die Faktoren dienen dabei der Normierung der Varianz von p_c , außerdem wird der Pfad zeitlich geglättet. Die Kovarianzmatrix wird unter Berücksichtigung des Evolutionspfades in jeder Generation folgendermaßen angepasst

$$C^{(t+1)} = (1 - c_{cov}) \cdot C^{(t)} + c_{cov} \cdot p_c^{(t+1)} \left(p_c^{(t+1)} \right)^{\top}, \quad (5.6)$$

zu Beginn wird für C die Einheitsmatrix verwendet.

Die Kovarianzmatrix C gibt die erwarteten Richtungen der Mutationen vor. Um die erwartete Weite der Mutationen zu steuern, wird der Wert σ verwendet, der mit Hilfe eines eigenen Mutationspfades p_σ so adaptiert wird, dass die erwartete Schrittweite der entspricht, die vorher zu erfolgreichen Mutationen geführt hat. Für die genau Darstellung dieser Adaption wird auf Hansen und Ostermeier (2001) verwiesen. Eine wichtige Weiterentwicklung der Adaption wurde in Hansen u. a. (2003) vorgeschlagen.

Die CMA-ES stellt einen robusten und schnellen Optimierungsalgorithmus dar (Hansen und Kern, 2004), beim bekannten *Black-Box Optimization Benchmarking* gehören CMA-Varianten zu den besten Verfahren in vielen Teilbereichen, siehe z. B. Hansen u. a. (2010). Die CMA-ES wurde auch für die im nächsten Abschnitt vorgestellte Mehrziel-Optimierung angepasst (Igel u. a., 2007).

5.3. Mehrziel-Optimierung

Ein Sonderfall der (evolutionären) Optimierung ist das gleichzeitige Betrachten mehrerer Ziele. Typischerweise sind diese konkurrierend (engl. *conflicting goals*), d. h., dass nicht in allen betrachteten Zielen *gleichzeitig* die Lösungen erreicht werden können, die in den *einzelnen* Ziel jeweils möglich wäre. Statt eines einzelnen Fitnesswerts wird einem Individuum x ein Vektor $\Phi(x) = (\Phi_1(x), \dots, \Phi_m(x))$ mit m Fitnesswerten zugeordnet.

Bei der Mehrziel-Optimierung soll eine möglichst diverse Menge von Lösungen gefunden werden, die die Menge der *Pareto*-optimalen Lösungen approximiert. Eine Lösung ist Pareto-optimal genau dann wenn sie von keiner anderen Lösung *dominiert* wird. Eine Lösung dominiert eine andere Lösung (schwach), wenn sie in mindestens einem Kriterium besser und in keinem schlechter ist.

Wenn für die Mehrziel-Optimierung evolutionäre Algorithmen betrachtet werden, können Mutations- und Rekombinations-Operatoren genauso verwendet werden wie bei klassischen evolutionären Algorithmen. Ein relevanter Unterschied ergibt sich jedoch bei der Selektion von Individuen. Das Ziel der Optimierung ist, die Pareto-Front möglichst gut zu approximieren (s. o.). Dazu muss also regelmäßig sichergestellt werden, dass die Individuen einer Population möglichst breit und gleichmäßig entlang dieser Front verteilt sind. Die Durchführung der Selektion ist ein kritischer Punkt, weil dieses Ziel dabei berücksichtigt werden muss. Im Folgenden wird ein geeignetes Verfahren zum Sortieren von Lösungen vorgestellt, das sich an Igel u. a. (2007) orientiert.

Basierend auf dem bereits erwähnten Konzept der Dominanz kann jeder Lösung x_i in einer Menge $M = \{x_0, \dots, x_n\}$ ein Rang $r(x_i)$ zugewiesen werden, der intuitiv

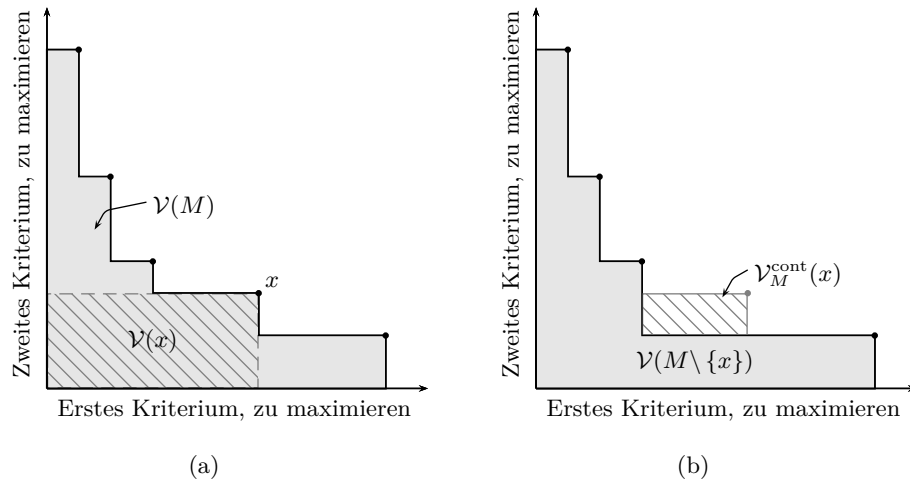


Abbildung 5.1.: Bewerten von Lösungen aufgrund von Hypervolumen.

(a) $\mathcal{V}(x)$ ist das von Individuum x dominierte Hypervolumen, $\mathcal{V}(M)$ das von der Menge M dominierte Hypervolumen.

(b) Beitragendes Hypervolumen $\mathcal{V}_M^{\text{cont}}(x)$ von $x \in M$ und von der Menge $M \setminus \{x\}$ dominiertes Hypervolumen $\mathcal{V}(M \setminus \{x\})$.

beschreibt, wie sehr sie andere Lösungen dominiert und / oder selbst dominiert wird. Die Menge M wird so in Teilmengen $M_1, M_2, \dots, M_{n_{\text{levels}}}$ partitioniert: Lösungen in M_1 werden von keiner anderen Lösung aus M dominiert, Lösungen in M_2 werden nur von Elementen aus M_1 dominiert, Lösungen in M_3 nur von Elementen aus $M_1 \cup M_2$, usw. Der Rang $r(x)$ einer Lösung x ist nun definiert als der Index der Teilmenge, in der sie enthalten ist.

Der Rang bietet eine erste Möglichkeit, Individuen bzgl. ihrer Fitness zu vergleichen. Jedoch wird ein weiteres Konzept benötigt, um Lösungen zu sortieren, die den gleichen Rang haben. Hierfür kann beispielsweise das *beitragende Hypervolumen* (engl. *contributing hypervolume*) betrachtet werden, wie in Igel u. a. (2007), Beume u. a. (2007) und Suttrop u. a. (2009) vorgeschlagen. Dafür wird zunächst das Konzept des *dominierten Hypervolumen* (engl. *dominated hypervolume*) benötigt.

Das dominierte Hypervolumen $\mathcal{V}(x)$ einer Lösung x ist definiert als Volumen aller Punkte im Zielraum, die durch $\Phi(x)$ dominiert werden. Das dominierte Hypervolumen einer Menge M ist definiert als Vereinigung der dominierten Hypervolumen aller Elemente in M . Damit das Hypervolumen berechnet werden kann, muss ein Referenzpunkt angegeben werden, das ist in der Praxis i. d. R. aber möglich (untere Grenze für die Fitness-Werte Φ_1, \dots, Φ_m). Abbildung 5.1(a) veranschaulicht das Konzept des dominierten Hypervolumens.

Das beitragende Hypervolumen $\mathcal{V}_{M_i}^{\text{cont}}(x)$ einer Lösung $x \in M_i$ ist gegeben als Anteil des dominierten Hypervolumens, der verloren geht, wenn die Lösung aus der Menge entfernt wird: $\mathcal{V}_{M_i}^{\text{cont}}(x) = |\mathcal{V}(M_i) - \mathcal{V}(M_i \setminus \{x\})|$. Abbildung 5.1(b) illustriert dieses Konzept.

Eine Menge M^* , die ausschließlich nicht-dominierte Lösungen enthält, wird Pareto-Menge genannt, die zugehörigen Vektoren $\{\Phi(x) \mid x \in M^*\}$ Pareto-Front. Nach der Optimierung repräsentiert M^* verschiedene mögliche *Trade-offs* für das betrachtete Problem. Die hier vorgestellten Konzepte erlauben es also, Algorithmen zur evolutionären Optimierung (z. B. die in den letzten beiden Abschnitten 5.1 und 5.2 beschriebenen Verfahren) einzusetzen, um mehrere Ziele gleichzeitig zu optimieren.

6. Naive Architektur für ein Gesamtsystem

In aktuellen und zukünftigen Serienfahrzeugen wird eine wachsende Zahl verschiedener kamerabasierter FAS parallel eingesetzt, relevant sind insbesondere Objekterkennung (Fahrzeuge, Fußgänger, Verkehrszeichen, usw.), Schätzen von optischem Fluss und Stereo-Verarbeitung.

Aus Kostengründen sollten entsprechende Algorithmen auf einfachen digitalen Signalprozessoren (DSP) ausführbar sein, also keine spezielle Hardware (bspw. *Field Programmable Gate Array, FPGA*) benötigen. Zu dieser Einschränkung bezüglich der Hardware-Ressourcen kommen die hohen Anforderungen an die Rechenzeit (Echtzeitanforderungen) und gleichzeitig an die Güte der Ergebnisse. Aufgrund dieser Kombination stellt der Entwurf einer geeigneten Software-Architektur im betrachteten Kontext eine große Herausforderung dar.

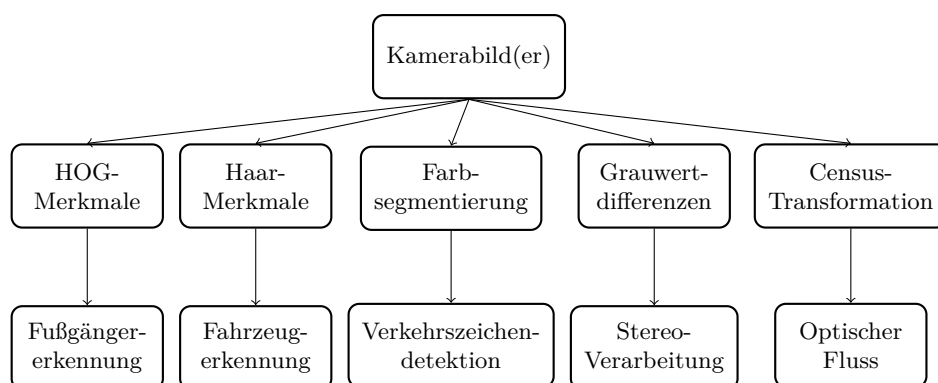


Abbildung 6.1.: Systemarchitektur für videobasierte FAS nach heutigem Stand der Technik. Angegeben ist jeweils die Merkmalsextraktion, die hauptsächlich genutzt wird (erste Stufe) und der Algorithmus (zweite Stufe). Für Details siehe Text.

In den Kapiteln 1–4 dieser Arbeit wurden verschiedene Verfahren angesprochen, die aus aktueller Sicht für die oben genannten Anwendungen für videobasierte FAS in Betracht kommen und teilweise bereits in heutigen Serienfahrzeugen eingesetzt werden. Ein einfaches („naives“) Gesamtsystem könnte also entworfen werden, indem diese

Algorithmen kombiniert werden. Es ergibt sich dann die eine Architektur wie in Abbildung 6.1 skizziert.

Die kamerabasierten Applikationen Fußgängererkennung (Dalal und Triggs, 2005, Enzweiler und Gavrilu, 2009, Dollár u. a., 2009), Fahrzeugerkennung (Khammari u. a., 2005, Alefs, 2006, Zheng und Liang, 2009), Verkehrszeichendetektion (Houben, 2011), Schätzen von Optischem Fluss (Stein, 2004) und Stereo-Verarbeitung (Hirschmüller, 2005, 2008) erfordern jeweils eigene spezielle Vorverarbeitungsschritte, dies ist eine Schwachstelle der naiven Architektur. Bei der praktischen Umsetzung resultiert daraus ein großer Hardware-Bedarf und somit höhere Kosten.

Das naive Gesamtsystem eignet sich so kaum für eine Umsetzung in Serienfahrzeugen. Neben den hohen Anforderungen erschwert die Komplexität des Gesamtsystems auch die Wartbarkeit, da Anpassungen immer an allen Modulen unabhängig vorgenommen werden müssen (z. B. wenn eine Kamera mit anderem Öffnungswinkel und / oder anderer Auflösung verwendet werden soll). Dieses Argument gilt auch im Betrieb, da typischerweise Anpassungen (z. B. an äußere Bedingungen wie Helligkeit) notwendig sind.

Teil II.

Entwurf einer vereinheitlichenden Architektur

7. Vorgeschlagene Systemarchitektur

In der vorliegenden Arbeit wird eine neue Architektur vorgeschlagen, die verglichen mit einem naivem Ansatz (vgl. Kapitel 6) deutlich besser geeignet ist, wenn mehrere verschiedene videobasierte FAS parallel eingesetzt werden.

Die neue Architektur zeichnet sich dadurch aus, dass ausschließlich Haar-Merkmale (siehe Abschnitt 1.2) in einer universellen Vorverarbeitung genutzt werden. Haar-Merkmale stellen einfache Kanten-Detektoren dar, sie sind – gerade auch in einfacher Hardware – sehr effizient zu berechnen. Die neu vorgeschlagene Architektur ist in Abbildung 7.1 dargestellt.

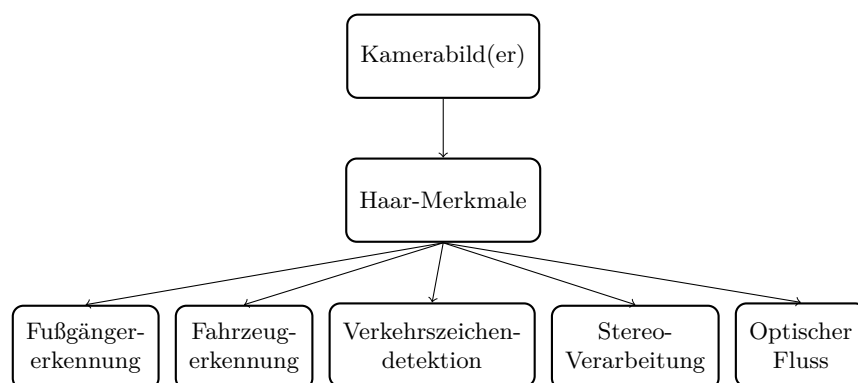


Abbildung 7.1.: Systemarchitektur für videobasierte FAS mit universeller Vorverarbeitung. Die Applikationen Fußgängererkennung (siehe Abschnitt 8.1), Fahrzeugerkennung (siehe Alefs (2006), Zheng und Liang (2009), Yong u. a. (2011)), Verkehrszeichendetektion (siehe Abschnitt 8.2), Stereo-Verarbeitung (siehe Kapitel 9) und Schätzen von Optischem Fluss (siehe Kapitel 10) greifen alle auf Haar-Merkmale zurück.

Durch den Einsatz einer gemeinsamen Bildvorverarbeitung ergeben sich große Einsparungen. Wie groß genau diese sind, hängt von der Anzahl der integrierten Module und ihrer konkreten Funktionalität ab. Als vereinfachtes Modell kann z. B. angenommen werden, dass bei einem typischen Algorithmus zur Bildverarbeitung auf die Merkmalsberechnung und auf die eigentliche Verarbeitung jeweils gleich viel Rechenzeit entfällt.

Vergleicht man nun zwei alternative Systeme (Abbildung 6.1 und Abbildung 7.1) in denen jeweils fünf Module eingesetzt werden, erfordert die Variante mit gemeinsamer Vorverarbeitung insgesamt 40 % weniger Rechenzeit.

Haar-Merkmale stellen vielseitig parametrisierbare Kantenfilter dar. Es ist daher gut vorstellbar, dass die als Ersatz für andere Merkmale verwendet werden können, die im Wesentlichen Kanten im Bild repräsentieren (z. B. HOG-Merkmale oder Census-Transformation). Für einige relevante Anwendungen scheint es jedoch nötig, auf Farbinformation zurückzugreifen, etwa für das Erkennen von Verkehrszeichen. Besonders interessant ist daher die Frage, wie gut Aufgaben die normalerweise mithilfe von Farbinformation gelöst werden in der neu vorgeschlagenen Architektur funktionieren können.

Durch die Verwendung von Haar-Merkmalen ergibt sich ein wesentlicher Unterschied zu alternativen Verfahren. Da typischerweise ein Satz von Haar-Merkmalen betrachtet wird (für die Erkennung von Objekten werden 100 oder mehr verwendet), hat der betrachtete Algorithmus eine hohe Zahl Parameter: Für jedes einzelne Merkmal müssen z. B. dessen Typ und Größe (vgl. Abschnitt 1.2) festgelegt werden. Es ist nicht sinnvoll und kaum möglich, diese Parameter manuell zu wählen. Damit wird es erforderlich, Strategien anzugeben, wie dieser Prozess automatisiert werden kann.

Die neue Architektur verspricht große Einsparungen mit Blick auf Rechenzeit. Dabei sollten jedoch alle o. g. wichtigen Module eine Leistungsfähigkeit erreichen, die mindestens dem bisherigen Stand der Technik entspricht. In den folgenden Kapiteln 8–10 werden Algorithmen vorgestellt, die diese Bedingungen erfüllen.

8. Objekterkennung basierend auf Haar-Merkmalen

In diesem Kapitel werden Algorithmen vorgestellt, die innerhalb der vorgeschlagenen Architektur verwendet werden können, um verschiedene relevante Objekte zu erkennen. In Abschnitt 8.1 wird ein neues effizientes Verfahren zur Detektion von Fußgängern vorgeschlagen. In Abschnitt 8.2 werden unterschiedliche Algorithmen zur Erkennung von Verkehrsschildern betrachtet und auf einem neuen Benchmark-Datensatz verglichen.

Die hier betrachteten Aufgaben sind sehr unterschiedlicher Natur: Fußgängererkennung ist ein 2-Klassen-Problem, während es sich bei der Verkehrszeichenerkennung um ein Multi-Klassen-Problem handelt. Fußgänger weisen eine große Variabilität in ihrer Erscheinung auf, während Verkehrszeichen starre Objekte sind, die so entworfen wurden, dass sie gut erkennbar sind. Trotzdem kann das computerbasierte Erkennen selbst dieser Objekte noch nicht als gelöst betrachtet werden.

Aufgrund der genannten Unterschiede zwischen den Problemen werden in den folgenden Abschnitten jeweils problemspezifische Verfahren betrachtet. Es soll die Frage beantwortet werden, welche Leistungsfähigkeit erreicht werden kann, wenn alleine Haar-Merkmale für die Bildvorverarbeitung eingesetzt werden, so wie für die neue Architektur vorgeschlagen.

8.1. Detektion von Fußgängern

Das Erkennen von Menschen in Videobildern ist die Grundlage für besonders wichtige FAS-Applikationen – Fußgänger gehören zu den leicht verletzlichen Verkehrsteilnehmern. Um Kollisionen zu vermeiden, kann der Fahrer rechtzeitig gewarnt werden, genauso sind (halb-)automatische Brems- oder Ausweichmanöver vorstellbar.

In jedem Fall sind leistungsfähige Algorithmen erforderlich, die Echtzeitanforderungen erfüllen wenn sie in einfacher Hardware realisiert werden. In diesem Kapitel wird ein entsprechendes Verfahren vorgeschlagen, die Grundlagen werden in Abschnitt 8.1.1 skizziert. Einen wichtigen Beitrag leistet die evolutionäre Optimierung von Haar-Merkmalen, die in Abschnitt 8.1.2 beschrieben wird. Die durchgeführten Experimente werden in Abschnitt 8.1.3 beschrieben und in Abschnitt 8.1.4 diskutiert.

8.1.1. Vorgeschlagenes Vorgehen

Munder und Gavrilu (2006) haben zahlreiche Verfahren zur Fußgängererkennung systematisch auf einem umfangreichen Benchmark-Datensatz verglichen. Dabei zeigten Support-Vektor-Maschinen (SVM) als Klassifikatoren generell besonders gute Leistungen, auch, wenn sie basierend auf einem großen Satz von Haar-Merkmalen eingesetzt wurden. Die besten Merkmale in den Experimenten waren lokale rezeptive Felder (Fukushima u. a., 1983), die allerdings selbst für die Daten trainiert wurden. Auch in Enzweiler und Gavrilu (2009) wurden bei der Detektion von Fußgängern sehr gute Ergebnisse mit linearen SVMs erzielt, hier in Kombination mit HOG-Merkmalen (siehe Abschnitt 1.3).

Damit ein Verfahren zur Detektion von Fußgängern in der Praxis eingesetzt werden kann, muss es Echtzeitanforderungen erfüllen. Je nach Aufbau des Gesamtsystems (Vorsteuerung, initiale Detektion, usw.) müssen pro Kamerabild mehrere tausend Bildausschnitte verarbeitet werden. Für diese Aufgabe sind die oben genannten Ansätze aufgrund ihrer Rechenzeitanforderungen nicht geeignet. Der in Bezug auf die Erkennungsleistung beste Ansatz in (Enzweiler und Gavrilu, 2009) benötigte in der praktischen Anwendung Rechenzeiten von mehr als 2 Sekunden pro Bild. Als sinnvolle Alternative wird dort ein Viola-Jones-Detektor basierend auf Haar-Merkmalen genannt.

Hier wird vorgeschlagen¹, einen einfachen linearen Klassifikator basierend auf einem möglichst kleinen Satz von Haar-Merkmalen zu verwenden. Diese Kombination ist für Systeme mit Echtzeitanforderungen sehr gut geeignet.

Binäre Klassifikation mit einem linearen Klassifikator basierend auf Haar-Merkmalen wird konkret folgendermaßen durchgeführt: Für einen gegebenen Bildausschnitt werden die Antworten von N Haar-Merkmalen berechnet, die Ergebnisse im Merkmalsvektor v zusammengefasst, wobei die i -te Komponente v_i die Antwort des i -ten Merkmals (mit $i = 1, \dots, N$) enthält. Die lineare Klassifikation erfolgt dann aufgrund der Vorschrift $0 < b + w^\top \cdot v$ wobei der Gewichtsvektor w und der *Bias* b beim Training des Klassifikators (z. B. mittels LDA, siehe Abschnitt 2.2) festgelegt werden.

Wenn N stark beschränkt wird, ist die Wahl dieser N Merkmale von besonders großer Bedeutung. Es gibt eine immense Zahl von Haar-Merkmalen (Kombinationen von Grundtyp, Größen, Position im Bild und ggf. weitere Parameter, vgl. Abschnitt 1.2). Dementsprechend gibt es praktisch unendlich viele Möglichkeiten, einen Merkmalsatz zusammenzustellen. Eine vollständige Suche ist damit unmöglich (Jain u. a., 2000), es

¹Teile der in diesem Abschnitt vorgestellten Arbeiten und Ergebnisse wurden bereits in Salmen u. a. (2007) sowie Salmen u. a. (2010) veröffentlicht. Die Arbeiten wurden teilweise durch Drittmittelprojekte mit der BMW AG, der Robert Bosch GmbH und der Continental AG finanziert.

muss stattdessen auf geeignete Heuristiken zurückgegriffen werden, um einen möglichst guten Merkmalsatz zu bestimmen.

Eine naheliegende und populäre Möglichkeit besteht darin, einen festen *Pool* möglicher Merkmale vorzugeben (der z. B. verschiedene Merkmalstypen in unterschiedlichen Größen enthält) und daraus die beste Teilmenge zu wählen. Ein Überblick über entsprechende Algorithmen zur Merkmalsselektion findet sich bspw. bei Jain u. a. (2000). Ein Nachteil bei diesem Vorgehen besteht in der Abhängigkeit vom Merkmals-*Pool*, der immer noch fest vorgegeben, d. h., manuell gewählt werden muss. Das eigentliche Problem – möglichst gut geeignete Merkmale vorzugeben – wird somit lediglich verschoben.

Eine sinnvolle Alternative zu oben beschriebenen Selektionsverfahren stellt das automatische Optimieren von Merkmalen dar. Im folgenden Abschnitt wird eine entsprechende Methode für das betrachtete Problem vorgestellt.

8.1.2. Evolutionäre Optimierung von Merkmalen

Das Ziel der Optimierung ist, einen Merkmalsatz, der für eine gegebene Aufgabe möglichst gut geeignet ist, in einem iterativen Prozess automatisch zu konstruieren. Hier wird vorgeschlagen, dafür eine Evolutionsstrategie zu verwenden, bei der die einzelnen Individuen jeweils einen Merkmalsatz darstellen. Wie in Kapitel 5 beschrieben, wird es so möglich, neue Lösungen durch Mutation und Rekombination zu erhalten. Die Optimierungsschleife für die konkrete Anwendung hat den folgenden Aufbau:

1. Erzeuge λ zufällige Merkmalsätze mit jeweils N Haar-Merkmalen.
2. Die Merkmalsätze bilden die Individuen der ersten Eltern-Generation. Evaluiere die Güte aller Individuen der Eltern-Generation.
3. Erzeuge μ Merkmalsätze durch Rekombination von Individuen aus der Eltern-Generation und anschließende Mutationen.
4. Die Merkmalsätze bilden die Individuen der Nachkommen-Generation. Evaluiere die Güte aller Individuen der Nachkommen-Generation.
5. Wähle die besten λ Individuen aus der Vereinigung der Eltern- und Nachkommen-Generation. Sie bilden die nächste Eltern-Generation.
6. Wenn Stopp-Kriterium noch nicht erfüllt, gehe zu Schritt 3.

In den folgenden Abschnitten wird die Umsetzung der einzelnen Schritte beim hier betrachteten Beispiel genauer beschrieben: die gewählte Repräsentation von Haar-Merkmalen, das Erzeugen von Merkmalen, die verwendeten Mutations-Operatoren und die Anpassung der Operator-Wahrscheinlichkeiten.

Merkmal-Repräsentation

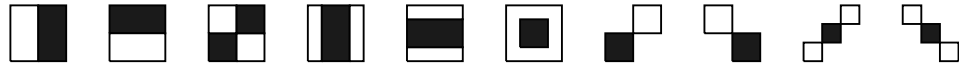


Abbildung 8.1.: Grundtypen von Haar-Merkmalen, die in den Experimenten zur Fußgängererkennung berücksichtigt wurden.

Für die Experimente wird eine erweiterte Repräsentation von Haar-Merkmalen verwendet. Zunächst werden die 10 Grundtypen berücksichtigt, die in Abbildung 8.1 dargestellt sind. Während der Optimierung können Merkmale sich jedoch von diesen Grundtypen entfernen, d.h., die einzelnen Regionen können sich unabhängig voneinander bewegen.

Die Antwort r eines Merkmals kann optional durch eine nicht-lineare Transformation verändert werden. Dafür werden zwei Parameter eingeführt: eine untere Schranke x_t^F (Schwellwert) und eine obere Schranke x_s^F (Sättigung). Die sog. Aktivierungsfunktion f_1 wird dann folgendermaßen berechnet

$$f_1(r, x_t, x_s) = \begin{cases} -g_{\max} & r \leq -x_s, \\ (r + x_t) \cdot \frac{g_{\max}}{x_t - x_s} & -x_s < r \leq -x_t, \\ 0 & -x_t < r < x_t, \\ (r - x_t) \cdot \frac{g_{\max}}{x_t - x_s} & x_t \leq r < x_s, \\ g_{\max} & x_s \leq r \end{cases},$$

dabei ist g_{\max} der maximal mögliche Grauwert und damit auch die größtmögliche Antwort eines Haar-Merkmals in diesem Bild. Die Auswirkungen der beiden Schwellwerte sind in Abbildung 8.2 dargestellt. Die untere Schranke stellt die minimale notwendige Antwort dar, die obere Schranke bewirkt ein Sättigungs-Verhalten. Für den Sonderfall $x_t = 0$ und $x_s = g_{\max}$ wird die ursprüngliche Merkmalsantwort nicht verändert.

Je nach Anwendungsfall kann es bei Haar-Merkmalen sinnvoll sein, den Betrag der Merkmalsantwort zu betrachten. Daher wird eine zweite, alternative Aktivierungsfunktion $f_2(r, x_t, x_s) = |f_1(r, x_t, x_s)|$ definiert. Für jedes verwendete Merkmal wird während der Optimierung ein Parameter a^F mitgeführt, der festlegt, ob f_1 oder f_2 verwendet wird.

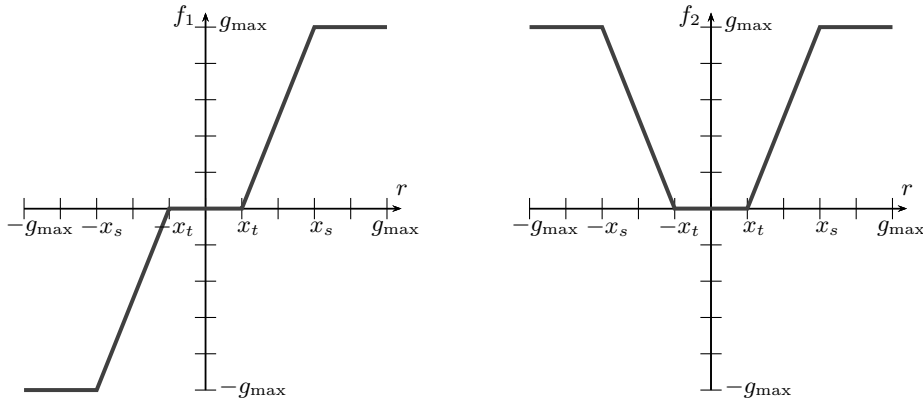


Abbildung 8.2.: Nicht-lineare Aktivierungsfunktionen, die zur Transformation von Merkmals-Antworten verwendet werden: f_1 (links) und f_2 (rechts).

Erzeugen von Merkmalen

Zu Beginn der Optimierungsschleife und im Rahmen von Mutationen (s. u.) sollen neue Merkmale zufällig erzeugt werden. Für die hier vorgestellten Experimente wurde das Erzeugen folgendermaßen durchgeführt. Zunächst wird der Typ des neuen Merkmals zufällig gemäß einer festen Verteilung bestimmt. In den Untersuchungen hier wurde dafür immer eine Gleichverteilung gewählt. Die Breite w^F des neu erzeugten Haar-Merkmals muss jeweils fest vorgegeben werden (s. u.). Die Höhe h^F wird dann in Abhängigkeit von dieser Breite zufällig bestimmt. Hier wurde immer $h^F = w^F \cdot 1/2 \cdot z$ mit $z \sim \mathcal{N}$ gewählt, der Wert z ist also normalverteilt. Je nach Parametrierung dieser Normalverteilung ergeben sich unterschiedliche Merkmale. Wenn bspw. als Erwartungswert 1 gewählt wird, werden bevorzugt quadratische Merkmale erzeugt.

Die beiden Schwellwert x_t^F und x_s^F wurden zufällig normalverteilt gewählt mit Mittelwert 5 bzw. 150. Der Wert a^F wurde mit Wahrscheinlichkeit 0,6 als 2 gewählt (d. h., f_2 wird verwendet), sonst als 1. Die Werte wurden empirisch ermittelt und führten in vorausgehenden Experimenten zu guten Ergebnissen.

Mutations-Operatoren

Das Mutieren eines Individuums (hier also eines Merkmalsatzes) wird durch sukzessive Anwendung von m Mutations-Operatoren implementiert. Der Wert m wird dabei in jeder Generation t mit $0 \leq t \leq t_{\max}$ neu bestimmt gemäß einer Poisson-Verteilung, deren Mittelwert λ von der aktuellen Generation t , der maximalen Anzahl Generationen t_{\max} und der Zahl zu optimierender Merkmale N abhängt. Die erwartete Anzahl

an Mutationen pro Generation sinkt kontinuierlich – in frühen Generationen kann der Suchraum so schneller betrachtet werden (Exploration), während später feinere Annäherungen an lokale Maxima möglich werden (engl. *exploitation*).

Die Mutations-Operatoren, die in jeder Generation angewendet wurden, wurden zufällig bestimmt. Für jeden Operator wurde eines der Merkmale aus dem Merkmalssatz gleichverteilt gezogen, auf das der Operator angewendet wird. Somit kann ein einzelnes Merkmal in einem Zeitschritt auch durch mehrere aufeinanderfolgende Mutationen verändert werden, z. B. erst Mutation die Größe betreffend und dann den Ort.

Es werden 13 verschiedene Mutations-Operatoren definiert:

- *Verschieben*: Die x - und y -Koordinaten aller Regionen des Merkmals werden um Δ_x bzw. Δ_y verschoben
- *Vergrößern*: Das Merkmal wird um den Faktor s vergrößert.
- *Verkleinern*: Das Merkmal wird um den Faktor $1/s$ verkleinert.
- *Umschalten von a^F* : Der Parameter a^F wird geändert, von 1 auf 2 oder umgekehrt.
- *Erhöhen von x_t^F* : Der Schwellwert x_t^F wird um den Faktor s erhöht.
- *Reduzieren von x_t^F* : Der Schwellwert x_t^F wird um den Faktor $1/s$ reduziert.
- *Erhöhen von x_s^F* : Der Schwellwert x_s^F wird um den Faktor s erhöht.
- *Reduzieren von x_s^F* : Der Schwellwert x_s^F wird um den Faktor $1/s$ reduziert.
- *Mutiere Merkmal*: Der Mittelpunkt, die Höhe und die Breite aller rechteckigen Bereiche im Merkmal werden unabhängig voneinander mutiert. Nach dieser Operation entspricht das Merkmal i. d. R. keinem der Grundtypen mehr.
- *Neu-Initialisierung „klein“*: Das Merkmal wird durch ein neues, zufällig erzeugtes (s. o.) ersetzt. Für die Breite, die dafür fest vorgegeben werden muss, wird hier ein zufälliger *kleiner* Wert gewählt.
- *Neu-Initialisierung „mittel“*: So wie oben, jedoch mit einem etwas größeren Erwartungswert für die Breite des neuen Merkmals.
- *Neu-Initialisierung „groß“*: So wie oben, jedoch mit einem großen Erwartungswert für die Breite des neuen Merkmals.
- *Keine Operation*: Das Merkmal wird nicht verändert.

Nach dem Mutieren eines Merkmals wird sichergestellt, dass alle Werte in sinnvollen Bereichen liegen.

Anpassung der Operator-Wahrscheinlichkeiten

Die hier vorgeschlagene Suchstrategie wird hauptsächlich durch die Mutations-Operatoren bzw. die Wahrscheinlichkeiten, mit denen sie jeweils angewendet werden, bestimmt. Da die optimalen Wahrscheinlichkeiten für ein gegebenes Problem einerseits nicht vorab bekannt sind und sich andererseits im Laufe der Evolution ändern können, ist es vielversprechend, sie selbst automatisch anzupassen. Igel und Kreutz (2003) schlagen eine Methode vor, um solch eine Anpassung zu erreichen. Das Verfahren basiert auf der Annahme, dass Operatoren die in den vergangenen Generationen zu relativ großen Verbesserungen geführt haben, auch in den folgenden Generationen nützlich sein können und deshalb bevorzugt werden sollten.

Sei Ω die Menge der Mutations-Operatoren und $P_o^{(t)}$ die Wahrscheinlichkeit $o \in \Omega$ in Generation t zu wählen. Weiterhin sei $O_o^{(t)}$ die Menge aller Nachkommen, die in Generation t erzeugt wurden und an deren Entstehung der Operator o beteiligt war. Im Fall, dass mehrere Operatoren zur Entstehung des Individuums beigetragen haben, wird es so behandelt, als wäre es mehrfach erzeugt worden, also jeweils einmal durch jeden der beteiligten Operatoren.

Die Wahrscheinlichkeiten, die einzelnen Mutations-Operatoren aus Ω zu verwenden, werden im vorgestellten Algorithmus alle τ Generationen angepasst. Die durchschnittliche Leistung $q_o^{(t,\tau)}$ jedes Operators $o \in \Omega$ innerhalb der letzten τ Generationen wird in Generation $t \geq \tau$ definiert als

$$q_o^{(t,\tau)} = \frac{\sum_{i=0}^{\tau-1} \sum_{g \in O_o^{(t-i)}} \max \left\{ 0, \Phi(g) - \max_{g' \in \text{Par}(g)} \Phi(g') \right\}}{\sum_{i=0}^{\tau-1} |O_o^{(t-i)}|}, \quad (8.1)$$

wobei die Menge $\text{Par}(g)$ alle Eltern des Individuums g enthält. Das Ergebnis $q_o^{(t,\tau)}$ entspricht also der durchschnittlichen Verbesserung, die nach Anwendung des Operators erzielt wurde. Es sei

$$q_{\text{all}}^{(t,\tau)} = \sum_{o' \in \Omega} q_{o'}^{(t,\tau)} \quad (8.2)$$

die Summe aller Leistungen. Für das Anpassen der Wahrscheinlichkeiten werden zunächst die neuen Gewichtungen

$$s_o^{(t+1)} \begin{cases} c_\Omega q_o^{(t,\tau)} / q_{\text{all}}^{(t,\tau)} + (1 - c_\Omega) s_o^{(t)} & \text{wenn } q_{\text{all}}^{(t,\tau)} > 0 \\ c_\Omega / |\Omega| + (1 - c_\Omega) s_o^{(t)} & \text{sonst} \end{cases} \quad (8.3)$$

bestimmt, dabei ist $c_\Omega \in (0,1]$ eine Lernrate, mit der eine zeitliche Glättung erreicht werden kann. Im Fall $q_{\text{all}}^{(t,\tau)} \leq 0$ (kein Operator konnte in den letzten τ Generationen eine Verbesserung erzielen) sollen die Wahrscheinlichkeiten sich einer Gleichverteilung nähern. Sonst (im Fall $q_{\text{all}}^{(t,\tau)} > 0$) sollen die Operatoren bevorzugt werden, die überdurchschnittliche Verbesserungen ermöglicht haben.

Die neuen Wahrscheinlichkeiten $P_o^{(t+1)}$ werden danach folgendermaßen berechnet

$$P_o^{(t+1)} = P_{\min} + (1 - |\Omega|P_{\min})s_o^{(t+1)} / \sum_{o' \in \Omega} s_{o'}^{(t+1)}. \quad (8.4)$$

Initial haben alle verfügbaren Mutations-Operatoren die gleiche Wahrscheinlichkeit, also $s_o^{(0)} = P_o^{(0)} = 1/|\Omega|$. Die Wahrscheinlichkeiten $P_o^{(t)}$ können beschränkt werden durch eine Konstante P_{\min} mit $0 < P_{\min} < 1/|\Omega|$ um sicherzustellen, dass kein Operator dauerhaft verworfen wird.

8.1.3. Experimente

In diesem Abschnitt werden der Aufbau und die Ergebnisse der Experimente beschrieben, die durchgeführt wurden, um das neu vorgeschlagene Vorgehen zu untersuchen und seine Leistungsfähigkeit auf einem Benchmark-Datensatz zu vergleichen.

Aufbau

In Munder und Gavrilu (2006) werden verschiedene Verfahren zur Fußgängererkennung systematisch in Bezug auf ihre Erkennungsleistung verglichen. In diesem Zusammenhang wurde auch ein großer Benchmark-Datensatz veröffentlicht². Er umfasst 14.400 manuell gewählte Bildausschnitte, die Fußgänger zeigen, und 15.000 automatisch generierte Ausschnitte, die keine Fußgänger zeigen, jedoch ähnliche Bildstrukturen enthalten. Die Ausschnitte sind alle auf eine einheitliche, verhältnismäßig kleine Größe von 18×36 Pixel skaliert. Zusätzlich zu diesen Ausschnitten stehen 1.200 komplette Kamerabilder zur Verfügung, die keine Fußgänger enthalten. Abbildung 8.3 zeigt beispielhaft einige Bilder aus dem Datensatz. Die Einzelbilder aus dem Datensatz sind in drei Trainings- und zwei Testmengen unterteilt, sie wurden so für die Experimente hier verwendet.

In den Experimenten wurden verschiedene Größen des Merkmalsatzes berücksichtigt, für $N \in \{50, 100, 150, 200\}$ wurden jeweils neun unabhängige Optimierungsläufe durchgeführt. In jedem Optimierungslauf wurde ein evolutionärer Algorithmus

²http://www.gavrila.net/Research/Pedestrian_Detection/Daimler_Pedestrian_Benchmark_D/Daimler_Mono_Ped_Class_Bench/daimler_mono_ped_class_bench.html



Abbildung 8.3.: (a) Bilder aus dem verwendeten Benchmark-Datensatz von Munder und Gavrila (2006). Positive (obere Reihe) und negative (untere Reihe) Beispiele, Größe jeweils 18×36 Pixel.
 (b) Beispiel für zusätzlich verfügbare Kamerabilder ohne Fußgänger, Größe dieses Bildes 360×288 Pixel.

durchgeführt mit Populationsgrößen $\lambda = 25$ und $\mu = 25$ und 2000 Generationen (vgl. Kapitel 5). Es wurden immer zwei Eltern für das Erzeugen eines Nachkommen berücksichtigt.

In jeder Generation t wurden $N \cdot 10^{-1 - \frac{t}{2000}}$ Mutationen durchgeführt, wobei N die Anzahl der zu optimierenden Merkmale ist. Die Lernrate c_{Ω} für das Anpassen der Wahrscheinlichkeiten der Mutations-Operatoren wurde auf 0,3 gesetzt.

Die besten Nachkommen aus jedem der neun Optimierungsläufe wurden für die endgültige Auswertung berücksichtigt. Die Güte dieser Lösungen auf dem Testdatensatz (der während der Optimierung nicht verwendet wurde) wurde gemäß dem von Munder und Gavrila (2006) vorgeschlagenen Vorgehen ermittelt: Drei unterschiedliche Klassifikatoren wurden auf jeweils einem der drei Trainings-Datensätze trainiert. Dann wurde jeder der drei Klassifikatoren auf jedem der zwei Testdatensätze getestet. Damit ergeben sich zunächst sechs verschiedene *ROC*-Kurven – aus diesen ergibt sich eine endgültige *ROC*-Kurve als Durchschnitt der Einzel-Experimente. Eine *ROC*-Kurve stellt verschiedene mögliche *Trade-offs* zwischen der Falschalarmrate FO und der Detektionsrate R (Richtig-Positiv-Rate, *recall*) dar. Sei tp die Anzahl der korrekt erkannten positiven Beispiele (*true positive*), tn die Anzahl der korrekt erkannten negativen Beispiele (*true negative*), fp die Anzahl der Fehldetektionen (*false positive*) und fn die Anzahl der nicht erkannten positiven Beispiele (*false negative*). Dann berechnen sich FO und R folgendermaßen:

$$FO = \frac{fp}{fp + tn} \quad (8.5)$$

und

$$R = \frac{tp}{tp + fn}. \quad (8.6)$$

Ergebnisse

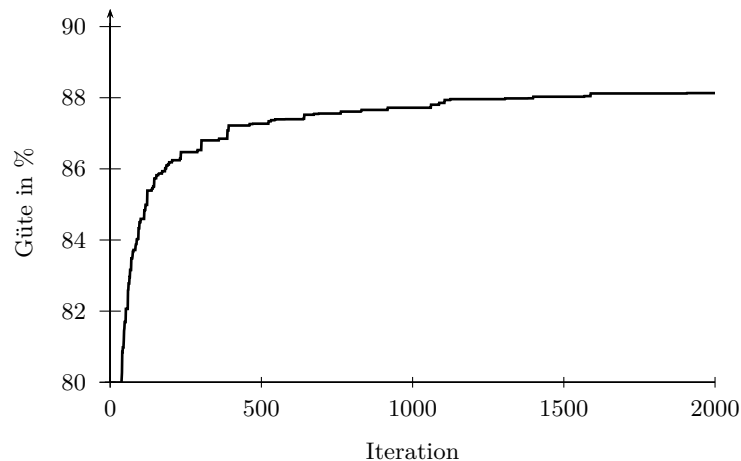


Abbildung 8.4.: Entwicklung der Fitness während der evolutionären Optimierung, hier beispielhaft für $N = 50$. Für jeden der neun Läufe wurde die beste Lösung in jeder Iteration bestimmt, dargestellt ist der Median dieser Werte.

Abbildung 8.4 veranschaulicht die durchschnittliche Fitness der jeweils besten Individuen in den neun durchgeführten Optimierungsläufen in jeder Generation für das Beispiel $N = 50$.

Tabelle 8.1.: Güte der jeweils besten gefundenen Lösungen für verschiedene Größen des Merkmalsatzes.

Anzahl Merkmale N	Höchste Fitness
50	89,23 %
100	90,75 %
150	91,68 %
200	92,69 %

In Tabelle 8.1 sind die Ergebnisse des jeweils besten Individuums aus allen Läufen für verschiedene Größen des Merkmalsatzes angegeben. Der beste Merkmalsatz für $N = 100$ ist in Abbildung 8.5 dargestellt. Viele der evolvierten Merkmale unterscheiden

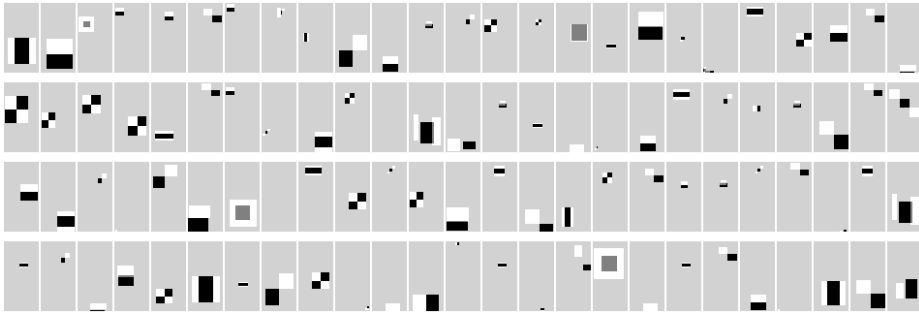


Abbildung 8.5.: Optimierter Merkmalsatz der Größe $N = 100$.

sich von den Grundtypen und damit von typischen manuell vorgegebenen Merkmalen. Der optimale Merkmalsatz hätte ohne evolutionäre Algorithmen mit Sicherheit nicht vorgegeben werden können.

Die ROC-Kurven der besten optimierten Klassifikatoren mit unterschiedlicher Anzahl Merkmalen werden in Abbildung 8.6(a) miteinander verglichen. Als *Baseline* sind auch die Ergebnisse eines linearen Klassifikators auf dem Standard-Merkmalssatz aus Munder und Gavrilu (2006) mit berücksichtigt. Die Ergebnisse zeigen deutlich die Vorteile von Merkmalen, die dem betrachteten Problem angepasst sind.

Da die unabhängigen Optimierungsläufe zu unterschiedlichen Merkmalsätzen führten, wurde schließlich ein *Ensemble*-Klassifikator untersucht. Dafür wurden für feste N jeweils die besten Lösungen aus allen Optimierungsläufen zusammengefasst. Die endgültige Klassifikation im *Ensemble*-Klassifikator ergibt sich als Mehrheitsentscheidung der einzeln optimierten Klassifikatoren. In Abbildung 8.6(b) werden die *ROC*-Kurven der resultierenden Klassifikatoren mit den Ergebnissen der quadratischen SVM aus Munder und Gavrilu (2006) auf einem Standard-Merkmalssatz verglichen. Diese SVM entspricht dem besten Klassifikator basierend auf Haar-Merkmalen aus Munder und Gavrilu (2006).

Die Klassifikatoren, die evolutionär optimiert wurden, erreichen nicht nur sehr gute Ergebnisse bzgl. der Erkennung, sondern sind auch um mehrere Größenordnungen schneller als die SVM: Die lineare Klassifikation besteht im Berechnen eines einzigen Skalarprodukts während die quadratische SVM ein Skalarprodukt pro Support-Vektor berechnet, was in diesem Fall mehrere tausend sind. Das Klassifizieren von 1.000 Bildausschnitten basierend auf $N = 100$ Merkmalen dauerte 0,76 ms auf einem Intel Pentium M Prozessor mit 1,8 GHz.

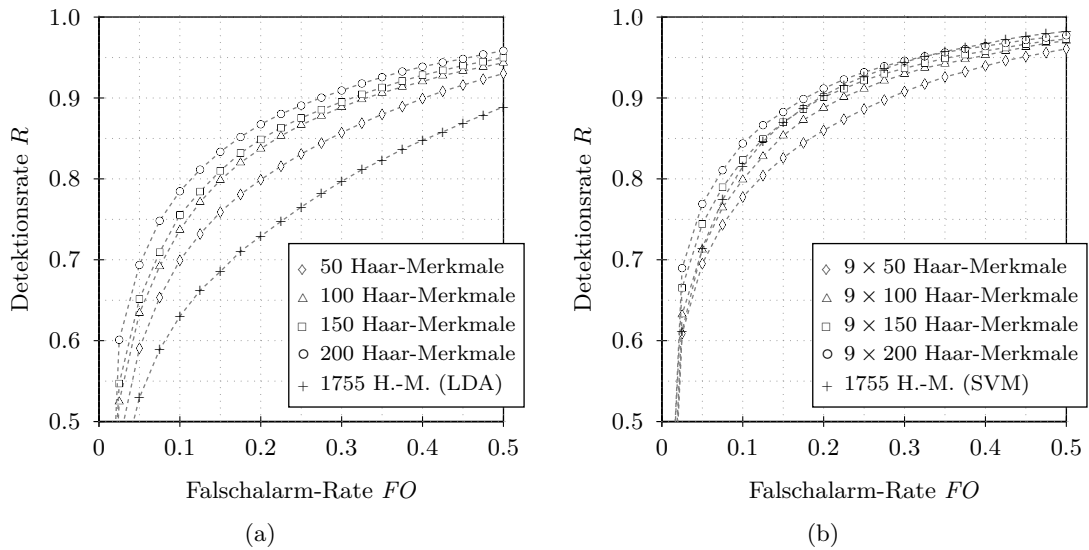


Abbildung 8.6.: (a) *ROC*-Kurven der besten Klassifikatoren basierend auf unterschiedlich vielen Haar-Merkmalen verglichen mit *ROC*-Kurve eines linearen Klassifikators auf Standard-Merkmalssatz der Größe 1755 aus Munder und Gavril (2006).

(b) *ROC*-Kurven von *Ensemble*-Klassifikatoren aus jeweils neun unabhängig voneinander optimierten Klassifikatoren basierend auf Haar-Merkmalen verglichen mit der besten SVM basierend auf Haar-Merkmalen aus Munder und Gavril (2006).

8.1.4. Diskussion

Die mit den optimierten Haar-Merkmalen erzielten Ergebnisse sind vergleichbar mit den Ergebnissen, die in Munder und Gavril (2006) mit einer SVM auf einem deutlich größeren Satz von Haar-Merkmalen erzielt wurden. Das hier vorgeschlagene Verfahren ist jedoch für Echtzeit-Anwendungen deutlich besser geeignet, da die Rechenzeit mehrere Größenordnungen geringer ist.

Ein weiterer Vorteil besteht in der praktisch vollständigen Unabhängigkeit von manuellen Vorgaben. Es müssen keine relevanten Parameter durch den Nutzer vorgegeben werden, daher ist weder Vorwissen über die vorliegenden Daten noch über die eingesetzten Verfahren notwendig.

Ein mögliches Risiko bei der automatischen Optimierung besteht in der Überanpassung an die Trainingsdaten. Im hier vorgeschlagenen System wurde dieses Risiko minimiert, indem der *cross-validation*-Fehler als Gütemaß betrachtet wurde. Dieser Wert gibt den

Fehler auf „ungesehenen“ Daten wieder, wenn er lediglich ein einziges Mal betrachtet wird. In der hier vorgeschlagenen Optimierung wird er jedoch mehrfach, nämlich in jeder Generation, neu evaluiert und dann zur Modell-Selektion verwendet. Daher ist er mit zunehmender Anzahl an Generationen eine zunehmend schlechte Schätzung für die tatsächliche Generalisierungsfähigkeit.

Aufgrund der vorgeschlagenen Optimierung von Haar-Merkmalen erlaubt bereits die einfache lineare Klassifikation, ausreichend gute Ergebnisse für die praktische Anwendung in der Fußgängererkennung zu erzielen. Die Erkennung anderer relevanter Objekte, z. B. Fahrzeuge, ist wegen ihrer weniger variablen Erscheinung nicht schwieriger. Mit dem vorgeschlagenen Vorgehen konnte auch für diese Anwendung sehr gute Ergebnisse erzielt werden.

Die in den Experimenten ermittelten ROC-Kurven geben einen guten Überblick über die Leistungsfähigkeit verschiedener Ansätze, insbesondere stellen sie für jedes Verfahren jeweils verschiedene mögliche Trade-Offs dar. Für die praktische Anwendung kommen jedoch nur Lösungen in Frage, die eine verhältnismäßig geringe Falschalarm-Rate ($< 10\%$) erreichen. In diesen Bereichen sind die Lösungen, die mit dem hier vorgeschlagenen Vorgehen erreicht wurden, signifikant besser als alternative Ansätze.

Für die evolutionäre Optimierung von Haar-Merkmalen wurden nur die 29.400 einzelnen Bildausschnitte verwendet, die im betrachteten Benchmark-Datensatz zur Verfügung stehen. Die zusätzlichen 1.200 Kamerabilder aus dem Datensatz wurden nicht verwendet um Vergleichbarkeit mit anderen veröffentlichten Ergebnissen zu gewährleisten. Da aus den Kamerabildern viele weitere Gegenbeispiele extrahiert werden können (mehr, als ursprünglich im Datensatz vorhanden), ließe sich, durch eine entsprechende Erweiterung des Trainings, die Leistungsfähigkeit der hier vorgeschlagenen Verfahren noch weiter steigern.

Es stellt eine sehr große Herausforderung dar, Fußgänger in Videobildern einer Monokamera zu erkennen. Aktuelle Arbeiten beschäftigen sich daher mit der Einbeziehung zusätzlicher Merkmale: Wojek und Schiele (2008) untersuchen Kombinationen verschiedener Bildmerkmale (darunter Haar- und HOG-Merkmale) und zeigen, dass alle untersuchten Klassifikatoren profitieren können. Walk u. a. (2010b,a) zeigen, dass das Einbeziehen von Tiefen- bzw. Fluss-Information helfen kann, die Ergebnisse zu verbessern. Neue öffentliche Benchmarks zur Fußgängererkennung enthalten zusätzlich zu den rohen Bilddaten auch Tiefen- und / oder Fluss-Information, siehe z. B. Enzweiler u. a. (2010) und Keller u. a. (2011).

8.2. Detektion von Verkehrszeichen

Verkehrsschilder sind, genauso wie Fahrspurmarkierungen und Ampeln, wichtige fest installierte Informationsquellen für Autofahrer. Das Missachten kann Unfälle begünstigen (z. B. Geschwindigkeitsbegrenzungen oder Schilder, die Vorfahrt regeln). Prinzipiell sind Verkehrsschilder aus den genannten Gründen so entworfen, dass sie von Menschen sehr gut wahrgenommen und erkannt werden können. Trotzdem ist das rechnerbasierte Erkennen von Verkehrszeichen in Videobildern noch nicht vollständig gelöst.

Algorithmisch wird die Verkehrszeichenerkennung typischerweise in zwei Schritten durchgeführt: Zuerst werden in einem Kamerabild Ausschnitte gesucht, die Verkehrszeichen enthalten könnten (sog. initiale Detektion). Dafür werden schnelle Algorithmen eingesetzt, die sich dadurch auszeichnen, dass sie sehr geringe *false negative*- und *false positive*-Raten bei gleichzeitig möglichst hoher *true positive*-Rate erreichen. Danach werden die Bildregionen aus der Detektion mit einem aufwendigeren aber noch zuverlässigerem Verfahren endgültig klassifiziert (vgl. Abschnitt 2.1).

Ein aktueller Benchmark von Stallkamp u. a. (2011, 2012) hat gezeigt, dass bei der Klassifikation maschinelle Verfahren mindestens genauso gute Ergebnisse erzielen wie Menschen (unter der Einschränkung, dass ausschließlich Bilder betrachtet werden, die tatsächlich Verkehrsschilder zeigen).

Hier wird ein neuer Benchmark-Datensatz vorgestellt, der eine systematische Evaluation von Verfahren zur Detektion von Verkehrszeichen ermöglicht (Abschnitt 8.2.1). Mit Hilfe dieses Datensatzes wurden umfangreiche Experimente durchgeführt. Das wichtigste Ziel dabei war, die Leistungsfähigkeit von Verfahren basierend auf Haar-Merkmalen mit anderen populären Ansätzen (die teilweise auf Farbinformation zurückgreifen) zu vergleichen. Die Experimente sind in Abschnitt 8.2.2 dokumentiert, die Ergebnisse werden in Abschnitt 8.2.3 diskutiert.

8.2.1. Benchmark-Datensatz

Die Kamerabilder für den Benchmark wurden im Frühjahr und Herbst 2010 bei mehreren Fahrten im Ruhrgebiet aufgenommen. Sie decken zahlreiche Szenarien (Innenstadt, Landstraße, Autobahn) und Wetterbedingungen ab und wurden tagsüber (einschließlich Dämmerung) aufgenommen. Abbildung 8.7 zeigt einige Beispiele. Die Bilder für den Klassifikations-Benchmark von Stallkamp u. a. (2011, 2012) entstammen denselben Sequenzen.











Abbildung 8.7.: Beispiel-Bilder aus dem neuen Benchmark-Datensatz zur Verkehrszeichen-Detektion mit Innenstadt- (erste Reihe), Landstraßen- (zweite Reihe) und Autobahn-Szenarien (dritte Reihe).

Für die Aufnahmen wurde eine *Prosilica GC1380CH* Kamera mit automatischer Belichtungssteuerung verwendet. Die einzelnen Videobilder wurden zunächst im *Bayer-pattern*-Format (Bayer, 1975) mit Größen von 1360×1024 Pixel gespeichert. Für den endgültigen Datensatz wurde nur der obere Teil der Bilder (1360×800 Pixel) betrachtet, da der untere Teil für die Anwendung nicht relevant ist. Die Bilder wurden *offline* vom *Bayer*-Format in *RGB* konvertiert, dafür wurde ein sehr genaues, adaptives Verfahren eingesetzt (Gunturk u. a., 2005, Ramanath u. a., 2002). Schließlich wurden alle Verkehrsschilder in den Videobildern durch Rechtecke manuell *gelabelt*.

Der endgültige Datensatz umfasst 900 Bilder die insgesamt 1206 Verkehrszeichen enthalten. Die markierten Verkehrszeichen haben Größen zwischen 16 und 128 *Pixel* (bezogen auf die jeweils längere Seite des entsprechenden Rechtecks). Der Datensatz wurde zufällig in Trainings- (600 Bilder, 846 Verkehrszeichen) und Test-Menge (300 Bilder, 360 Verkehrszeichen) geteilt. Dabei wurde sichergestellt, dass Bilder die dasselbe Schild zeigen im selben Datensatz enthalten sind. Dieser Fall ist allerdings selten, die meisten Schilder kommen im Datensatz nur einmal vor. Daher kann der Datensatz auch problemlos weiter unterteilt werden (z. B., um *cross-validation* durchzuführen). Die markierten Schilder lassen sich für den Vergleich von Detektions-Verfahren in acht

Kategorien unterteilen, die in Tabelle 8.2 dargestellt sind.

Tabelle 8.2.: Alle Verkehrsschilder aus dem Datensatz lassen sich einer dieser acht Kategorien zuordnen.

1.		Vorschriftzeichen rot	5.		Verbot der Einfahrt
2.		Weisungszeichen blau	6.		Stop
3.		Gefahrzeichen	7.		Vorfahrt gewähren
4.		Aufhebungszeichen	8.		Vorfahrtstraße

Die Daten zur Evaluierung von Detektions-Verfahren sind im Rahmen eines neuen Benchmark öffentlich verfügbar³. Aus der Webseite ist es möglich, die Ergebnisse verschiedener Algorithmen direkt zu veröffentlichen und zu vergleichen.

8.2.2. Experimente

In den Experimenten werden drei der acht Kategorien (vgl. Tabelle 8.2) betrachtet: Gebotsschilder (rund, roter Rand oder ganz blau) und Gefahrzeichen (dreieckig, roter Rand).

Für diese drei Kategorien werden jeweils vier verschiedene Verfahren trainiert: ein *Template*-basiertes Verfahren (siehe Abschnitt 1.6), ein Viola-Jones-Detektor (siehe Abschnitt 2.3), ein linearer Klassifikator basierend auf HOG-Merkmalen (siehe Abschnitt 1.3) sowie ein modellbasiertes Verfahren (s. u.). Diese Verfahren eignen sich prinzipiell für den Einsatz unter Echtzeit-Anforderungen, sie repräsentieren die populärsten Verfahren zur Verkehrszeichenerkennung.

Der genaue Aufbau der durchgeführten Experimente sowie ihre Ergebnisse (Seite 77) werden in den folgenden Abschnitten beschrieben.

Aufbau

Hier werden zunächst die vier Verfahren beschrieben, die untersucht wurden, danach werden die Methoden zur Auswertung erklärt.



Abbildung 8.8.: Durchschnitts-Bilder (die als *Templates* verwendet werden) für die drei betrachteten Kategorien.

Template-matching Aus den verfügbaren Trainingsdaten können sehr einfach *Templates* für jede Kategorie erstellt werden. Dazu wurden die Bilder jeweils auf eine einheitliche Größe skaliert und dann für jeden Pixel und jeden Farbkanal unabhängig der durchschnittliche Wert bestimmt. Die resultierenden *Templates* sind in Abbildung 8.8 dargestellt.

Die *Templates* können als „Vorlage“ genutzt werden, um ähnliche Bildausschnitte zu finden (vgl. Abschnitt 1.6). Einige Pixel sind jedoch für die Berechnung der Ähnlichkeit wichtiger als andere: Bildpunkte im Hintergrund können sich häufig ändern und sollten daher beim Vergleichen weniger (oder gar nicht) berücksichtigt werden. Dasselbe gilt ggf. für Strukturen innerhalb der Verkehrszeichen (z. B. Piktogramme oder Zahlen), die sich innerhalb der Klassen einer Kategorie unterscheiden.

Um dies möglichst einfach abzubilden, wurde beim Erstellen der *Templates* nicht nur der Mittelwert für jeden Pixel, sondern auch die Varianz bestimmt. Für die Berechnung der Ähnlichkeit wurden dann nur die 50 % der Bildpunkte berücksichtigt, die die geringste Varianz bzgl. der Helligkeit auswiesen. Für die Bestimmung der Ähnlichkeit wurde die normierte Kreuz-Korrelation verwendet, siehe Abschnitt 1.6.

Die Detektion der Schilder in einem Kamerabild kann schließlich einfach durchgeführt werden, indem die verfügbaren *Templates* auf verschiedene Größen (hier 20×20 bis 135×135 Pixel mit einer Schrittweite von 10 %) skaliert und an verschiedenen Positionen im Bild (Schrittweite jeweils 5 % der *Template*-Größe) die Ähnlichkeit bestimmt wird. Alle Bildausschnitte, für die die Ähnlichkeit einen festen Schwellwert überschreitet und gleichzeitig lokale Maxima sind, werden als Ergebnis ausgegeben.

Detektion basierend auf HOG-Merkmalen Die in Abschnitt 1.3 vorgestellten HOG-Merkmale erlauben z. B. sehr gute Ergebnisse bei der Detektion von Fußgängern in Kamerabildern, aber auch bei der Detektion und Klassifikation von Verkehrsschildern, siehe z. B. Zaklouta und Stanculescu (2011b).

³<http://benchmark.ini.rub.de>

Die Detektion im Kamerabild erfolgt, ähnlich wie beim oben beschriebenen *Template-matching*, in einem Suchfenster-basierten Ansatz. Das heißt, für verschiedene Bildausschnitte werden die Merkmale berechnet und dann eine lineare Klassifikation durchgeführt. Der entsprechende Klassifikator wird auf dem verfügbaren Trainingsdatensatz trainiert. Im Unterschied zum ursprünglich vorgeschlagenen Vorgehen von Dalal und Triggs (2005) wurde für das Training LDA (vgl. Abschnitt 2.2) statt einer linearen SVM verwendet. Während beide Verfahren prinzipiell die gleichen Freiheitsgrade haben, ist für LDA keine Modell-Selektion erforderlich (d. h., das Lernverfahren selbst hat keine kritischen Parameter).

Für das Training werden positive und negative Beispiele benötigt. Bildausschnitt die Verkehrszeichen zeigen (positive Beispiele) können aus den manuell markierten Regionen leicht gewonnen werden. Um angemessene Gegenbeispiele zu finden (d. h., für den Klassifikator "schwierige" Beispiele), wird das Training iterativ durchgeführt. Für die erste Iteration werden P negative Bildausschnitte zufällig aus den Trainings-Kamerabildern gesammelt, wobei P die Anzahl der positiven Beispiele ist. Nach dem ersten Training durch LDA werden P weitere negative Beispiele zur Trainingsmenge hinzugefügt, wobei diese nun zufällig aus der Menge der Bilder gezogen werden, die der Detektor in den Bildern fälschlicherweise als positiv klassifiziert. Dieser Prozess wurde 10-mal wiederholt.

Für die Merkmalsberechnung wurde die Implementierung der Original-Autoren verwendet⁴. Es wurden Zellen der Größe 5×5 Pixel und Block-Größen von 2×2 Zellen verwendet. In jedem Suchfenster wurden 6×6 Zellen betrachtet (also Bildausschnitte der Größe 30×30 Pixel). Um verschiedene Größenstufen abzudecken, wurde das Kamerabild in einer sog. Gauß-Pyramide in verschiedenen Skalierungen verarbeitet. Dafür wurden die Skalierungsfaktoren $\sqrt{2}^i$ mit $i = -5, -4, \dots, 0$ betrachtet. Es wurde die Implementierung der LDA aus der quelloffenen Bibliothek *Shark* verwendet (Igel u. a., 2008), die frei verfügbar ist⁵.

Modellbasierte Ansatz Dieser Ansatz repräsentiert verschiedene populäre Verfahren zur Verkehrszeichendetektion, z. B. (Broggi u. a., 2007, Barnes u. a., 2008, Belaroussi und Tarel, 2009). Solche modellbasierten Ansätze berücksichtigen zwei wichtige Merkmale: Farbe und Form. Dabei kann Farbe ein gutes Kriterium sein, um viele Bildbereiche von der Suche auszuschließen, während Kanteninformation dazu dient, Schilder von anderen Objekten ähnlicher Farbe zu unterscheiden.

Für die Experimente hier wurde das in Houben (2011) beschriebene Verfahren implementiert. Zuerst wird für jeden Punkt des Eingabebildes die Wahrscheinlichkeit

⁴<http://www.navneetdalal.com/software>

⁵<http://shark-project.sourceforge.net>

bestimmt, dass dessen Farbe einer relevanten Farbe (also der eines bestimmten Verkehrszeichen-Typs) entspricht. In diesem Bild werden dann Kanten berechnet mithilfe des Algorithmus von Canny (1986) berechnet, vgl. Abschnitt 1.1. Die stärksten Kanten werden für die Suche nach Dreiecken und Kreisen mithilfe einer angepassten Version des bekannten Hough-Algorithmus (Duda und Hart, 1972).

Für das Erkennen der eben genannten Formen werden in jedem Schritt jeweils genau so viele Kanten betrachtet, wie zur exakten Bestimmung von Position und Größe nötig. Die Kantenstärke wird bei der eingesetzten Hough-Variante als Gewicht verwendet. Insgesamt erzeugt dieses Verfahren Ergebnisse, die das Auffinden von Maxima erleichtern, für weitere Details siehe Houben (2011).

Zur Modellierung der Schilder-Farben wurde der YUV-Farbraum verwendet (siehe Abschnitt 1.5). Alle relevanten farbigen Pixel wurden in den Trainingsbildern manuell markiert. Zur Schätzung der endgültigen Verteilung wurden die Median-Werte aus allen Einzelbildern berücksichtigt.

Viola-Jones-Detektor Der Viola-Jones-Detektor ist ein für echtzeitfähige Anwendungen sehr populäres Verfahren (vgl. Abschnitt 2.3). Im Rahmen der vorliegenden Arbeit wird der Algorithmus auch in anderen Experimenten berücksichtigt (siehe Kapitel 12).

Für die Experimente hier wurde der Detektor prinzipiell so trainiert, wie ursprünglich vorgeschlagen. Um die Robustheit weiter zu erhöhen wurden jedoch (ähnlich wie bei den anderen hier getesteten Verfahren) negative Beispiele dem Training gezielt hinzugefügt. Dafür wurden nach dem Training jeder Kaskaden-Stufe zufällig Gegenbeispiele aus der Menge derjenigen hinzugefügt, die in den Trainingsbildern noch falsch erkannt wurden. Zusätzlich wurden nach dem Training der letzten Stufe einmalig alle noch falsch erkannten Bilder ins Training aufgenommen um dann den finalen Detektor zu trainieren.

Es wurde ein fester Satz von 5445 Haar-Merkmalen verwendet: Fünf Grundtypen (die ersten fünf aus Abbildung 1.2 auf Seite 11) jeweils in neun verschiedenen Größen (1, 1/2 und 1/4 der ROI-Größe jeweils für Breite und Höhe unabhängig voneinander) an jeder möglichen Position innerhalb des kleinsten Suchfensters. Die Größe des Suchfensters selbst wurde beginnend mit 24×24 Pixel skaliert jeweils mit einem Faktor von 1,25, insgesamt wurden 6 Größenstufen betrachtet. Bei der Suche im Kamerabild wurde das Fenster in beide Richtungen jeweils um 1/12 seiner Breite verschoben (vgl. auch Abschnitt 2.1).

Auswertung Bei der Auswertung hier werden die drei größten Kategorien betrachtet (vgl. Tabelle 8.2), dabei werden die Ergebnisse sowohl separat als auch insgesamt verglichen.

Die Ausgabe eines Detektions-Algorithmus für ein Bild besteht in einer Liste von als interessant erachteten Bereichen (*regions of interest, ROI*). Jede dieser ROIs wurde für die Auswertung mit jeder vorhandenen korrekten ROI G verglichen. Dafür wurde das Jaccard-Ähnlichkeitsmaß herangezogen

$$J(S, G) = \frac{|S \cap G|}{|S \cup G|} \in [0,1], \quad (8.7)$$

das die gegenseitige Überdeckung zweier ROIs bewertet. Um Ergebnisse mit verschiedenen Trade-offs zwischen Detektionsrate und Fehlalarm-Rate zu erhalten, wurden unterschiedliche Schwellwert berücksichtigt: Ein Verkehrszeichen in einem Kamerabild (GT-Roi G) als genau dann als erkannt gewertet, wenn für mindestens eine detektierte ROI S das Maß $J(S, G)$ den jeweiligen Schwellwert überschreitet.

Für jedes GT-Verkehrszeichen wurde nur die Detektor-ROI berücksichtigt, die den maximalen Jaccard-Wert erreicht. Mögliche weitere überlappende ROIs wurden ignoriert, also weder als zusätzliche Treffer noch als falsch gezählt.

Bei Detektions-Aufgaben ist das Zählen von Richtig-Negativ-Ergebnissen (*true negative, TN*) schwer handhabbar: Jeder mögliche Ausschnitt in einem Kamerabild, die vom Detektor nicht gefunden wird, gehörte zu dieser Kategorie. Da aber Detektoren in der Praxis nicht *jeden möglichen* Ausschnitt betrachten, sondern nur ausgewählte (da z. B. diskrete Größenstufen und feste Positionen betrachtet werden), führt diese Betrachtungsweise zu irreführenden Ergebnissen.

In solchen Fällen, wo die TN -Rate kaum sinnvoll zu ermitteln ist, werden statt einer ROC-Kurve (vgl. Abschnitt 8.1.3 und Abschnitt 11.3) sog. *Precision-Recall*-Kurven verwendet, die eng verwandt sind. Sie veranschaulichen verschiedene mögliche *Trade-offs* zwischen dem *zuverlässigen* Finden von Schildern (*precision*) und dem Anteil der korrekt gefundenen Schilder (*recall*). Konkret werden *precision* P und *recall* R folgendermaßen berechnet

$$P = \frac{tp}{tp + fp} \quad (8.8)$$

und

$$R = \frac{tp}{tp + fn}, \quad (8.9)$$

wobei tp die Anzahl der korrekt erkannten positiven Beispiele (*true positive*) ist, fp die Anzahl der Fehldetektionen (*false positive*) und fn die Anzahl der nicht erkannten positiven Beispiele (*false negative*).

Um verschiedene mögliche Trade-offs der verschiedenen Verfahren zu erhalten, wurden beim Template-basierten Verfahren unterschiedliche Schwellwerte für den NCC-Wert betrachtet, beim HOG-basierten Verfahren unterschiedliche Schwellwerte für das Ergebnis der linearen Klassifikation, beim modellbasierten Verfahren verschiedene Grenzen für die Zahl der Hough-Stimmen und die Ergebnisse verschiedener Kaskadenstufen beim Viola-Jones-Detektor.

Ergebnisse

In Abbildung 8.9 sind typische Beispiele für Fehldetektionen dargestellt – häufig lassen sich diese Fehler auf eine große Ähnlichkeit mit Verkehrsschildern zurückführen, das gilt insbesondere für die Form.

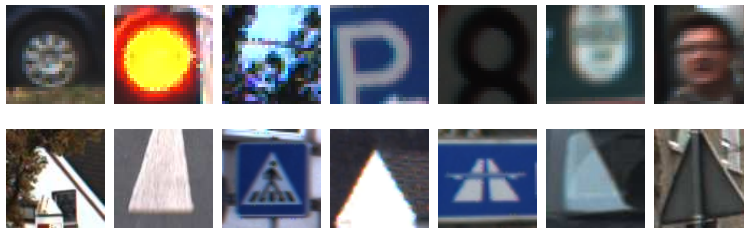


Abbildung 8.9.: Beispiele für falsche Detektionen, Verbotsschilder (oben) und Gefahrzeichen (unten).

In Abbildung 8.10 sind für jede der drei betrachteten Kategorien jeweils die wichtigsten fünf Haar-Merkmale dargestellt, d. h. die Merkmale, die beim Training als erste für die erste Kaskadenstufe gewählt wurden.



Abbildung 8.10.: Die fünf Haar-Merkmale die beim Training der Detektoren für die drei Kategorien jeweils zuerst ausgesucht wurden.

8. Objekterkennung basierend auf Haar-Merkmalen

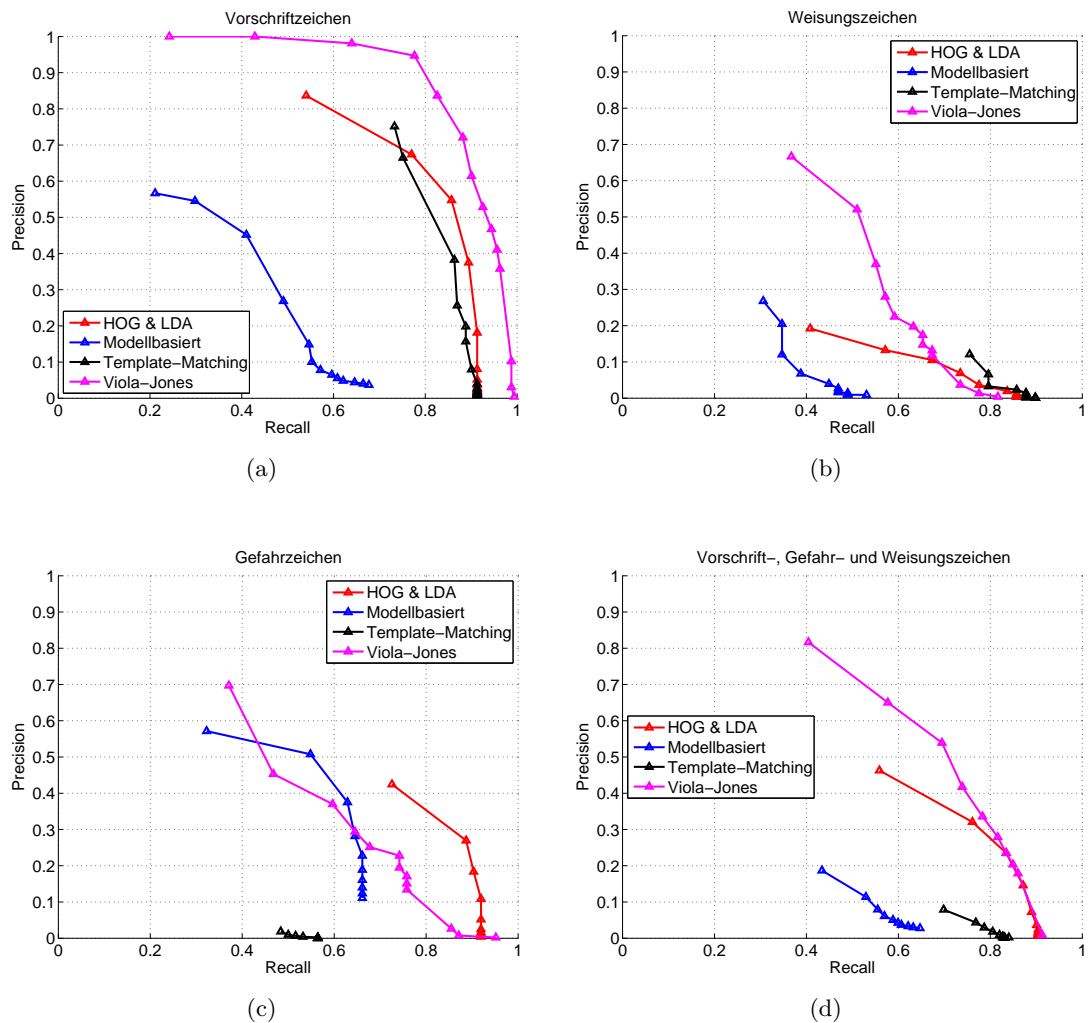


Abbildung 8.11.: (a)–(c) Ergebnisse der vier untersuchten Algorithmen auf den drei betrachteten Verkehrszeichen-Kategorien. (d) Ergebnisse der Algorithmen zusammengefasst über alle Kategorien.

In Abbildung 8.11 sind die *Precision-Recall*-Kurven der untersuchten Verfahren dargestellt. Der Viola-Jones-Detektor basierend auf Haar-Merkmalen erreicht auf allen hier betrachteten Kategorien die höchsten Detektionsraten. Der Detektor basierend auf HOG-Merkmalen erzielte ebenfalls gute Ergebnisse, insbesondere bei den dreieckigen Gefahrzeichen. Insgesamt waren die Ergebnisse sowohl für die blauen Weisungszeichen

als auch die Gefahrzeichen deutlich schlechter als die Ergebnisse für die roten runden Vorschriftzeichen.

8.2.3. Diskussion

In diesem Abschnitt wurde ein neuer umfangreicher Datensatz vorgestellt, der es ermöglicht, Verfahren zur Detektion von Verkehrszeichen zu vergleichen. Der Datensatz wird öffentlich verfügbar gemacht.

Hier wurden vier Ansätze evaluiert, die für die betrachtete Aufgabe häufig eingesetzt werden: Ein Template-basiertes Verfahren, ein modellbasierter Algorithmus, Klassifikation basierend auf HOG-Merkmalen und ein Viola-Jones-Detektor. Die beiden zuerst genannten Ansätze können auch Farbinformation berücksichtigen, die anderen beiden nur Kanteninformation. Der Viola-Jones-Detektor greift auf Haar-Merkmale zurück, die im Rahmen dieser Arbeit für eine universelle Vorverarbeitung vorgeschlagen werden.

Für die Experimente wurden die drei wichtigsten Kategorien von Verkehrszeichen berücksichtigt, die sich sowohl in ihrer Form (rund, dreieckig) als auch in ihrer Farbe (rot, blau) unterscheiden.

Während die Klassifikation von Verkehrszeichen im Rechner sehr gute Ergebnisse erzielen kann, stützen die Ergebnisse hier die vorherrschende Meinung, dass die Verkehrszeichen-Detektion noch nicht zufriedenstellend gelöst ist. Es gibt entsprechende Anwendungen zwar schon in Serienfahrzeugen, hier sind jedoch Zusatzinformationen (z. B. Kartendaten) verfügbar. Falsche Detektionen können durch eine gute funktionierende Klassifikation ausgeglichen werden.

Zusammenfassend lässt sich festhalten, dass mit dem Viola-Jones-Detektor ein allgemeines Verfahren zur Objekterkennung die vielversprechendsten Ergebnisse erzielt hat. Der Detektor verwendet Haar-Merkmale, bezieht also keine Farbinformation ein. Trotzdem waren die Ergebnisse besser als die anderer Ansätze, insbesondere die eines populären modellbasierten Verfahrens.

9. Stereo-Verarbeitung basierend auf Haar-Merkmalen

In Abschnitt 3.3 wurden drei Stereo-Algorithmen vorgestellt, die häufig in Anwendungen mit Echtzeitanforderungen eingesetzt werden. Die Verfahren bestimmen das nach ihrer jeweiligen Definition optimale Disparitätsbild für eine gegebene Kostenmatrix. Die Berechnung dieser Matrix ist unabhängig vom eigentlichen Stereo-Algorithmus. Sie muss für alle sinnvoll möglichen Zuordnungen von Pixeln aus dem linken und rechten Kamerabild die dafür entstehenden Kosten enthalten. Für die Berechnung werden normalerweise Grauwert- oder Farbdifferenzen berücksichtigt (vgl. Abschnitt 3.2).

In der in dieser Arbeit vorgeschlagenen Architektur wird davon ausgegangen, dass verschiedene Haar-Merkmale im gesamten Bild berechnet werden. Sie sind als Grundlage für die Erkennung von Objekten (vgl. Kapitel 8) und das Schätzen von optischem Fluss (vgl. Kapitel 10) sehr gut geeignet. Es ist naheliegend, ebenfalls zu untersuchen, inwiefern Haar-Merkmale sich für die Kostenberechnung von Stereo-Algorithmen eignen.

Die vorgeschlagene Kostenberechnung mit Haar-Merkmalen wird in Abschnitt 9.1 erklärt. In Abschnitt 9.2 wird beschrieben, wie alle Parameter der betrachteten Algorithmen automatisch optimiert werden können. So wird u. a. ein objektiver Vergleich gewährleistet. Die durchgeführten Experimente werden in Abschnitt 9.3 dokumentiert und die Ergebnisse in Abschnitt 9.4 diskutiert.

9.1. Kostenberechnung basierend auf Haar-Merkmalen

Sei R_l ein Vektor mit Antworten von N Haar-Merkmalen an einem Pixel (x, y) im linken Kamerabild und R_r ein Vektor mit Antworten derselben Merkmale an einem Punkt $(x - d, y)$ im rechten Bild. Um aus diesen Werten den entsprechenden Eintrag $C(x, y, d)$ für die Kostenmatrix zu berechnen, gibt es zahlreiche Möglichkeiten. In den Experimenten wird eine recht einfache Art der Kostenberechnung betrachtet, die gewichtete Summe der absoluten Differenzen

$$C(x, y, d) = \sum_{i=1}^N w_i \cdot |R_l(i) - R_r(i)|. \quad (9.1)$$

Für weiterführende Untersuchungen bieten sich z. B. die folgenden, aufwendigeren Kostenfunktionen an: Die Euklidische Distanz

$$C(x, y, d) = \sum_{i=1}^N (R_l(i) - R_r(i))^2, \quad (9.2)$$

die Maximumsnorm

$$C(x, y, d) = \max_{i=1, \dots, N} |R_l(i) - R_r(i)|, \quad (9.3)$$

oder die normierte Kreuzkorrelation (vgl. Abschnitt 1.6)

$$C(x, y, d) = \frac{1}{N} \sum_{i=1}^N \frac{(R_l(i) - \bar{R}_l)(R_r(i) - \bar{R}_r)}{\sigma_{R_l} \sigma_{R_r}}, \quad (9.4)$$

wobei \bar{R} den Mittelwert und σ_R die Standardabweichung von R bezeichnet.

9.2. Evolutionäre Optimierung mittels CMA-ES

Die in Abschnitt 3.3 vorgestellten Stereo-Algorithmen lassen sich relativ gut konfigurieren: WTA, DB und SGM haben jeweils nur zwei Parameter (Fensterbreite sx und -höhe sy bei WTA und jeweils zwei Strafkosten P_1 und P_2 bei DP und SGM). Um diese Parameter für eine konkrete Anwendung einzustellen, müssen einige (Trainings-)Bilder betrachtet werden, die den in der Anwendung erwarteten (Test-)Bildern möglichst ähneln. Typischerweise werden die Parameter dann manuell – in einem mehr oder weniger systematischen Prozess – iterativ optimiert.

Die Anzahl der Parameter kann sich leicht erhöhen, z. B., wenn Haar-Merkmale für die Kostenberechnung betrachtet werden oder wenn bewusst mehr Freiheitsgrade eingeführt werden (Salmen u. a., 2009). In den hier durchgeführten Experimenten (siehe den folgenden Abschnitt 9.3) mussten bis zu 17 Parameter optimiert werden. Diese Werte manuell gut einzustellen stellt auch für erfahrene Entwickler eine große Herausforderung dar, insbesondere weil das komplexe Zusammenwirken schwer handhabbar ist.

Wenn eine objektive Bewertung der Güte verschiedener Lösungen möglich ist (z. B. Stereo-Verarbeitung synthetischer Sequenzen), gibt es keinen Grund, den Prozess der Parameter-Optimierung manuell durchzuführen. Die Vorteile etablierter Verfahren wie etwa evolutionärer Algorithmen (siehe Kapitel 5) sind vielfältig: Sie erlauben das Verarbeiten großer Datenmengen, ermöglichen Zeitersparnis, gewährleisten Objektivität, usw.

Die CMA-ES stellt für evolutionäre Optimierung im \mathbb{R}^n den Stand der Technik dar, vgl. Abschnitt 5.2. Der Algorithmus kann mit einer großen Anzahl Parametern und insbesondere deren komplexem Zusammenwirken gut umgehen. Eine Anforderung für die Anwendung der CMA-ES ist, dass alle in der Optimierung betrachteten Variablen reelwertig sind. Die Parameter von Haar-Merkmalen (Größe und Position im Bild) sind normalerweise ganzzahlig, daher wurden in anderen Experimenten im Rahmen der vorliegenden Arbeit entsprechend angepasste evolutionäre Algorithmen verwendet (vgl. Abschnitt 8.1 und Abschnitt 10.3).

Für die Experimente zur Stereo-Verarbeitung wurden alle relevanten Module mit CUDA¹ für Grafikkhardware programmiert. Das führt einerseits zu einer Steigerung der Verarbeitungsgeschwindigkeit, andererseits bietet die Grafikkhardware eine automatische Interpolierung beim Zugriff auf nicht ganzzahlige Koordinaten. Damit wird es sofort sehr einfach möglich, die o. g. Parameter der Haar-Merkmale während der Optimierung als reelwertige Größen zu behandeln.

9.3. Experimente

Hier werden Aufbau (Abschnitt 9.3.1) und Ergebnisse (Abschnitt 9.3.2) der durchgeführten Experimente beschrieben.

9.3.1. Aufbau

In den Experimenten wurden zwei verschiedene Datensätze verwendet: Einerseits der sehr populäre Middlebury-Stereo-Benchmark² der in Scharstein und Szeliski (2002) vorgestellt wurde (Abbildung 9.1 zeigt Beispielbilder). Andererseits eine synthetische Sequenz von Vaudrey u. a. (2008) aus dem *Enpeda*-Benchmark³ (Abbildung 9.2 zeigt Beispiele).

Im Middlebury-Benchmark wurden bereits Ergebnisse von weit über 100 Stereo Algorithmen verglichen (Stand November 2011). Der Benchmark besteht aus lediglich vier Bildpaaren, die alle mit identischen Algorithmus-Parametern verarbeitet werden sollen. Es ist also keine Trennung zwischen Trainings- und Testdaten vorgesehen. Damit lassen die Ergebnisse des Middlebury-Benchmark wenig Rückschlüsse auf die *Generalisierungsfähigkeit* der jeweiligen Verfahren zu.

¹<http://www.nvidia.com/cuda>

²<http://vision.middlebury.edu/stereo>

³http://www.mi.auckland.ac.nz/index.php?option=com_content&view=article&id=44

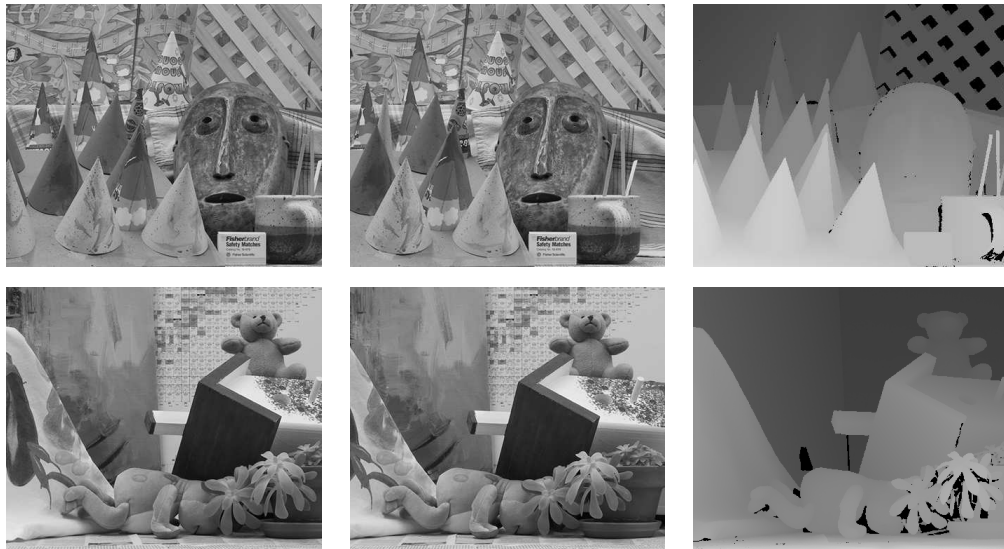


Abbildung 9.1.: Zwei der vier Bildpaare aus dem Middlebury Stereo-Benchmark: Bildpaar „Cones“ mit zugehörigen Disparitäten (obere Reihe) und Bildpaar „Teddy“ mit zugehörigen Disparitäten (untere Reihe).



Abbildung 9.2.: Bilder aus der künstlichen *Enpeda*-Sequenz, die in den Experimenten verwendet wurden.

Die *Enpeda*-Sequenz ist mit einem Simulator generiert, sie bildet ein recht realistisches Szenario nach mit verschiedenen Herausforderungen für bildverarbeitende Algorithmen (z. B. Schatten und komplexe Texturen). Solche synthetischen Sequenzen stellen aktuell die beste Möglichkeit dar, Verfahren, die in Fahrzeugen eingesetzt werden sollen, zu evaluieren, siehe auch van der Mark und Gavrilu (2006). Für reale Sequenzen *ground-truth*-Daten (korrekte Entfernungen in der Welt und damit korrekte Disparitäten) zu erhalten ist sehr aufwendig und nicht fehlerfrei möglich (Vaudrey u. a., 2008), vgl. auch die Diskussion in Abschnitt 9.4.

In den Experimenten hier wurden drei Stereo-Algorithmen betrachtet: WTA, DP und SGM (siehe Abschnitt 3.3). Die Parameter der drei untersuchten Verfahren wurden jeweils mittels CMA-ES für beide Datensätze optimiert. Für jedes betrachtete Szenario wurden fünf unabhängige Optimierungsläufe mit jeweils 300 Iterationen durchgeführt.

Beim Middlebury-Benchmark erfolgt diese Optimierung direkt auf den vier Bildpaaren, auf denen auch die Auswertung durchgeführt wird (s. o.). Für die *Enpeda*-Sequenz wurden hier die ersten 100 Bilder zum Training und die übrigen 290 zum Testen verwendet, d. h., beim Vergleich ist auch eine Aussage über die Generalisierungsfähigkeit der erhaltenen Lösungen möglich. Da aufeinanderfolgende Bilder innerhalb der Sequenz sich jeweils nur wenig voneinander unterscheiden wurde sowohl beim Training als auch beim Testen nur jedes zehnte Bild betrachtet. Damit umfasste der resultierende Trainingsdatensatz 10 Bilder und der Testdatensatz 29 Bilder.

Als Fehlermaß (sowohl bei der Optimierung als auch bei der abschließenden Auswertung) wurde die Anzahl der falschen Disparitäten (*bad pixels*) verwendet wie von Scharstein und Szeliski (2002) vorgeschlagen. Somit lassen sich die Ergebnisse auf dem Middlebury-Datensatz mit den öffentlich verfügbaren Ergebnissen zahlreicher anderer Lösungen vergleichen (s. o.).

Die Parameter der drei Stereo-Algorithmen wurden jeweils getrennt optimiert für die zwei betrachteten Fälle: Kostenberechnung basierend auf Grauwertdifferenzen oder basierend auf Haar-Merkmalen. Als Grundlage für die Berechnung der Kosten im



Abbildung 9.3.: Typen von Haar-Merkmalen, die als Basis für die Kostenberechnung verwendet wurden.

zweiten Fall wurden an jedem Punkt in beiden Kamerabildern die Antworten von sieben Haar-Merkmale berechnet – die verwendeten Typen sind in Abbildung 9.3 dargestellt. Das letzte Merkmal berechnet lediglich den durchschnittlichen Grauwert im betrachteten Bildausschnitt. Es wurde hier hinzugefügt, damit die Original-Lösung (Grauwert-Differenz für einzelne Pixel) im Suchraum enthalten ist: Dafür muss das Gewicht des letzten Merkmals auf 1 und alle anderen auf 0 gesetzt werden (s. u.) und die Größe des letzten Merkmals auf 1×1 Pixel.

Höhe und Breite der einzelnen Merkmale sind Parameter, die optimiert werden. Zum Vergleich zweier Bildpunkte wurde die in Gleichung (9.1) angegebene Funktion verwendet. Die Gewichte w_i für die einzelnen Merkmalstypen wurden ebenfalls optimiert. Da die betrachteten Stereo-Algorithmen DP und SGM selbst jeweils zwei Parameter

haben, ergaben sich in diesen Fällen 23 freie Parameter (für WTA nur 21, da die Fenstergröße nicht mehr relevant ist und der Algorithmus selbst keine Parameter hat).

Es wurde eine Implementierung der CMA-ES aus der Bibliothek *Shark* von Igel u. a. (2008) verwendet, die quelloffen ist⁴.

9.3.2. Ergebnisse

Tabelle 9.1.: Ergebnisse der untersuchten Stereo-Algorithmen für beide Datensätze.

Algorithmus	Kostenfunktion	Fehler Middlebury in %	Fehler <i>Enpeda</i> in %
WTA	Grauwertdifferenzen	12,96	7,34
WTA	Haar-Merkmale	10,46	6,37
DP	Grauwertdifferenzen	10,00	7,26
DP	Haar-Merkmale	8,45	5,92
SGM	Grauwertdifferenzen	6,26	4,59
SGM	Haar-Merkmale	5,04	3,87

In Tabelle 9.1 sind die Ergebnisse der besten Lösungen für die sechs untersuchten Algorithmen auf beiden Datensätzen angegeben. Auf beiden Datensätzen konnten alle untersuchten Algorithmen jeweils bessere Ergebnisse erzielen, wenn für die Kostenberechnung Haar-Merkmale statt Grauwertdifferenzen verwendet wurden. Die Zahl der falschen Disparitäten konnte um ca. 15 bis ca. 20 % reduziert werden.

Die Größe der Haar-Merkmale, die zu den jeweils besten Lösungen geführt haben, war bei den drei betrachteten Algorithmen sehr unterschiedlich. Für den Middlebury-Benchmark ergaben sich Kantenlängen zwischen 1 und 17,8 Pixel für WTA, zwischen 1,2 und 14,2 Pixel für DP und zwischen 1,1 und 6,6 Pixel für SGM. Für den *Enpeda*-Benchmark ergaben sich Kantenlängen zwischen 1 und 15 Pixel für WTA, zwischen 1 und 12,7 Pixel für DP und zwischen 1 und 6,1 Pixel für SGM.

Die Gewichtung der Merkmale variierte ebenfalls zwischen den Lösungen, die beste SGM-Variante für den Middlebury-Benchmark gewichtete bspw. die ersten beiden Merkmalstypen aus Abbildung 9.3 am stärksten (jeweils mit 0,26) und den vierten Grundtyp mit 0,19. Die Gewichte der anderen Merkmale lagen zwischen 0,06 und 0,09.

Abbildung 9.4 stellt beispielhaft Ergebnisse für ein Bild der *Enpeda*-Sequenz dar. Es wurden jeweils die Parameter verwendet, die bei der Optimierung zu den besten Ergebnissen geführt haben. Bei allen Algorithmen treten Fehler am häufigsten an Kanten auf, die wiederum teilweise mit Disparitätssprüngen einhergehen.

⁴<http://shark-project.sourceforge.net>

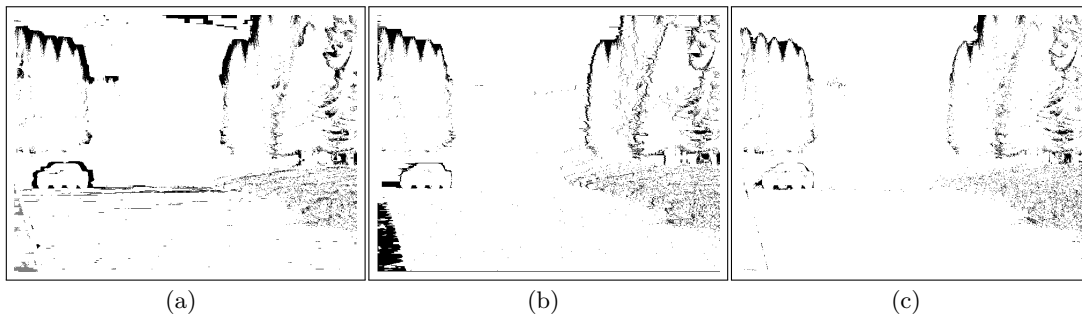


Abbildung 9.4.: Ergebnisse für ein Bild der *Enpeda*-Sequenz. Falsche Pixel sind schwarz dargestellt.

- (a) WTA basierend auf Grauwertdifferenzen.
- (b) DP basierend auf Haar-Merkmalen.
- (c) SGM basierend auf Haar-Merkmalen.

9.4. Diskussion

Echtzeitfähige Stereo-Algorithmen sind relevant für viele verschiedene Anwendungsgebiete, bspw. Robotik, Überwachung und Fahrerassistenzsysteme. Es gibt mehrere populäre Verfahren für solche Aufgaben, die je nach verfügbarer Hardware und notwendiger Güte der Ergebnisse eingesetzt werden. All diese Algorithmen erfordern als Eingabe eine Kostenmatrix, die Aussagen darüber macht, wie ähnlich jeweils Pixel aus den beiden Kamerabildern sind. Diese Kostenmatrix wird typischerweise basierend auf Helligkeits- oder Farbunterschieden einzelner Pixel berechnet.

Hier wurde untersucht, wie gut die Kostenberechnung basierend auf Haar-Merkmalen geeignet ist, um darauf basierend Disparitäten zu schätzen. In den Experimenten wurden drei echtzeitfähige Stereo-Algorithmen betrachtet, die unterschiedlich komplexe Ansätze verfolgen: WTA betrachtet Pixel unabhängig voneinander, DP optimiert Lösungen zeilenweise und SGM approximiert eine global optimale Lösung. Es wurden zwei verschiedene Datensätze betrachtet, der populäre Middlebury-Benchmark, der aus vier Bildpaaren besteht und eine lange synthetische Sequenz.

Für das Testen von Stereo-Algorithmen können leider kaum reale Bilder verwendet werden. Eine interessante Alternative, um die Leistungsfähigkeit von Verfahren auf Videodaten zu messen, wurde von Morales und Klette (2009) vorgeschlagen. Das Verfahren verlässt sich bei der Fehler-Berechnung allerdings auf den Vergleich von zwei Kamerabildern (einem tatsächlichen und einem geschätzten), was bei typischen Straßen-Szenen mit verhältnismäßig vielen homogenen Flächen zu Ungenauigkeiten bei der Auswertung führt. Es ist erstrebenswert, solche Ansätze weiter zu verbessern.

Die Parameter aller hier betrachteten Verfahren wurden durch automatische Optimierung mittels CMA-ES bestimmt. Unabhängig davon, dass dieses Vorgehen generell vorteilhaft und empfehlenswert ist, wurde auf diese Weise eine objektive Auswertung sichergestellt.

Die Ergebnisse zeigen, dass alle betrachteten Stereo-Verfahren von Kostenberechnung basierend auf Haar-Merkmalen profitieren, unabhängig von der Komplexität der Algorithmen selbst. Die Fehler für beiden Datensätze konnten um bis zu 20 % reduziert werden.

Die Ursachen für diese Erkenntnis sind sicherlich vielfältig, doch ist die Glättung, die durch Haar-Merkmale erreicht wird, eine naheliegende Erklärung: Die optimierten Haar-Merkmale berücksichtigen Fenster, die mehrere Pixel breit und hoch sind, damit wird die Nachbarschaft von Pixeln bei der Kostenberechnung mit einbezogen. Trotzdem können die Verbesserungen nicht allein auf diesen Glättungs-Effekte zurückgeführt werden: Der WTA-Algorithmus, der zum Vergleich betrachtet wurde, verwendet für die Kostenberechnung große Fenster, der DP- und der SGM-Algorithmus berücksichtigen selbst Abhängigkeiten innerhalb einer Zeile bzw. innerhalb der Nachbarschaft.

Eine weitere Erklärung ist daher die folgende. Die Antworten verschiedener Haar-Merkmalen stellen eine gute (und kompakte) Beschreibung eines Bildausschnitts dar, sie kodieren bestimmte Strukturen, z. B. Kanten und Linien. Die darauf basierenden Vergleiche, deren Ergebnisse in die Kostenmatrix eingehen, sind aussagekräftiger als Helligkeitsunterschiede. In den hier beschriebenen Experimenten wurde zunächst eine relativ einfache Kostenfunktion verwendet, mögliche Alternativen wurden jedoch genannt. Es kann für zukünftige Arbeiten lohnend sein, diese und weitere Kostenfunktion zu betrachten, um die vorhandenen Merkmalsantworten noch besser für die Kostenberechnung zu nutzen.

10. Optischer Fluss basierend auf Haar-Merkmalen

Das Schätzen von optischem Fluss ist ein wichtiger Verarbeitungsschritt für intelligente Fahrzeuge, da die Ergebnisse Basis für vielfältige Applikationen sind: Schätzen der eigenen Bewegung, Schätzen von 3D-Information aus Bewegung (*structure from motion*), insbesondere Kollisionsvermeidung und Freiraum-Erkennung.

Im Folgenden wird beschrieben, wie der in Kapitel 4 vorgestellte *PowerFlow*-Algorithmus so angepasst werden kann, dass er in der neu vorgeschlagenen Architektur mit Haar-Merkmalen als universelle Vorverarbeitungsstufe einsetzbar wird¹.

Die Anpassung des *PowerFlow*-Algorithmus wird in Abschnitt 10.1 erklärt, die vorgeschlagene Mehrziel-Optimierung von Merkmalen und anderen Parametern wird in Abschnitt 10.2 beschrieben. Anschließend werden die durchgeführten Experimente in Abschnitt 10.3 dokumentiert. Das Kapitel endet mit einer Diskussion in Abschnitt 10.4.

10.1. Anpassung des PowerFlow-Algorithmus

Der sog. *PowerFlow*-Algorithmus aus Stein (2004) wird bereits in heutigen Serienfahrzeugen eingesetzt. Der Algorithmus wurde so entworfen, dass er sich besonders für die Umsetzung in eingebetteter Hardware eignet. Er bietet eine ausreichend gute Genauigkeit bei gleichzeitig verhältnismäßig hoher Anzahl geschätzter Fluss-Vektoren, vgl. Abschnitt 4.2.

Damit der Algorithmus in der hier vorgeschlagenen Architektur eingesetzt werden kann, sind einige Anpassungen erforderlich. Im ursprünglichen Algorithmus wird die Census-Transformation genutzt um für jeden Bildpunkt einen *Hash*-Wert zu bestimmen (vgl. Abschnitt 1.4). Hier sollen Haar-Merkmale für das Berechnen dieser Signaturen verwendet werden. Um aus den Antworten mehrerer Haar-Merkmale R eine

¹Teile der in diesem Abschnitt vorgestellten Arbeiten und Ergebnisse wurden bereits in Salmen u. a. (2011) veröffentlicht. Die Arbeiten wurden teilweise durch Drittmittelprojekte mit der Continental AG und durch eine Förderung des BMWi finanziert.

Signatur (einen *Hash*-Wert) $\xi_i(R)$ zu gewinnen wird die folgende Berechnung durchgeführt

$$\xi_i(R) = \begin{cases} 0, & R_i < -\epsilon_i \\ 1, & |R_i| \leq \epsilon_i \\ 2, & R_i > \epsilon_i \end{cases}, \quad (10.1)$$

wobei ϵ_i ein Schwellwert für das i -te Merkmal ist. Genauso wie im Original-Algorithmus dienen diese Schwellwerte der Diskretisierung der kontinuierlichen Merkmalsantworten.

Während bei der zuerst vorgeschlagenen Census-Transformation lediglich ein Schwellwert verwendet wird, verwendet der *PowerFlow*-Algorithmus zwei. Das Vorgehen wird hier verallgemeinert, indem mehrerer Schwellwerte pro Merkmal erlaubt werden. Mit zwei Schwellwerten $\epsilon_i^{(0)}$ und $\epsilon_i^{(1)}$ wird die Diskretisierung beispielsweise folgendermaßen durchgeführt

$$\xi_i(R) = \begin{cases} 0, & |R_i| \leq \epsilon_i^{(0)} \\ 1, & \epsilon_i^{(1)} > R_i > \epsilon_i^{(0)} \\ 2, & R_i > \epsilon_i^{(1)} \\ 3, & -\epsilon_i^{(1)} < R_i < -\epsilon_i^{(0)} \\ 4, & R_i < -\epsilon_i^{(1)} \end{cases}. \quad (10.2)$$

Es ist beachtenswert, dass diese Formulierung die ursprünglich vorgeschlagene Census-Transformation als einen Spezialfall enthält. Die Definition von Haar-Merkmalen (vgl. Abschnitt 1.2) erlaubt es prinzipiell, mit 1 Pixel großen Regionen zu arbeiten. Deshalb kann die Census-Transformation durch eine Menge geeigneter Haar-Merkmale „emuliert“ werden. In diesem Fall stellt natürlich die Berechnung des Integralbildes keinen Vorteil dar.

Mit den oben beschriebenen Anpassungen kann der *PowerFlow*-Algorithmus basierend auf Haar-Merkmalen verwendet werden. Die weiteren Verarbeitungsschritte werden übernommen, damit bleiben auch die Parameter für die Zuordnung (*matching*) über mehrere Bilder und die zeitliche Integration erhalten.

Die Verwendung von Haar-Merkmalen erlaubt es, das Schätzen von optischem Fluss mit Sub-Pixel-Genauigkeit durchzuführen. Dafür wird ein optionaler Verfeinerungsschritt vorgeschlagen. Für jeden Flussvektor der in Bild t endet, wird eine Optimierung der Position des Endpunkts durchgeführt. Dabei wird diese Position so verschoben, dass die Merkmalsantworten denen am zugehörigen Startpunkt in Bild $t - 1$ möglichst stark ähneln. Zum Vergleich der Merkmalsvektoren R und S mit jeweils n

Komponenten kann die Summe der komponentenweisen absoluten Differenzen, also $c(R, S) = |R_1 - S_1| + |R_2 - S_2| + \dots + |R_n - S_n|$, herangezogen werden. Um Merkmalsantworten zu interpolieren, kann bilineare Interpolation verwendet werden.

10.2. Evolutionäre Mehrziel-Optimierung

Der hier vorgestellte modifizierte *PowerFlow*-Algorithmus zum Schätzen von optischem Fluss hat, wie der Original-Algorithmus, sieben Parameter. Zusätzlich müssen die Parameter der zu verwendenden Haar-Merkmale gewählt werden. Es bietet sich an, hier eine automatische Optimierung durchzuführen.

Ein Algorithmus zur spärlichen Schätzung von optischem Fluss, wie der hier betrachtete, muss zwei sich entgegengesetzte Ziele erfüllen. Einerseits sollen pro Bild möglichst viele Fluss-Vektoren geschätzt werden. Andererseits soll die durchschnittliche Güte der Ergebnisse möglichst hoch sein. Durch geeignete Wahl entsprechender Parameter können diese beiden Ziele typischerweise gegeneinander „eingetauscht werden“ (*trade-off*). Verschiedene Lösungen können also nicht eindeutig sortiert werden.

Aus diesem Grund wird bei der Optimierung der Haar-Merkmale und Parameter für den neu vorgeschlagenen Algorithmus eine Mehrziel-Optimierung durchgeführt (vgl. Abschnitt 5.3). Ergebnis der Optimierung sind dann mehrere mögliche Lösungen, die eine Pareto-Front approximieren. Das heißt, jede der Lösungen ist optimal in dem Sinn, dass jeweils keines der beiden Ziele (möglichst große Anzahl an Fluss-Vektoren und möglichst kleiner mittlerer Fehler) weiter verbessert werden kann, ohne dass das andere schlechter wird.

Fitnessberechnung Für das betrachtete Problem werden zwei Ziele betrachtet. Die Fitness eines Individuums x ist gegeben durch einen Vektor $\Phi(g) = (n, 1/e)$, wobei n die durchschnittliche Anzahl von Fluss-Vektoren ist und $1/e$ deren Güte: Als Fehlermaß e wird der durchschnittliche Endpunkt-Fehler verwendet wie von Baker u. a. (2007) vorgeschlagen. Beide Ziele Φ_1 und Φ_2 sollen maximiert werden.

Mutation Während der Optimierung können einerseits die an jedem Bildpunkt berechneten Merkmale verändert werden, andererseits können die Parameter des eigentlichen Fluss-Algorithmus variieren. Im Folgenden werden die Mutations-Operatoren erklärt, die für den Haar-Merkmal-basierten Algorithmus angewendet wurden.

Beim Mutieren einer Lösung wird zunächst zufällig gleichverteilt entschieden, ob der Merkmalsatz oder die Algorithmus-Parameter (siehe Tabelle 10.1) mutiert werden.

Die letztgenannten werden mit Standard-Operatoren mutiert, wobei für jeden Parameter festgelegte Grenzen berücksichtigt wurden (s. u.).

Beim Mutieren der Merkmale wird mit Wahrscheinlichkeit $p_{\text{size}} = 0,20$ deren Anzahl verändert (d. h., dann wurde mit jeweils gleicher Wahrscheinlichkeit ein Merkmal entfernt oder ein zufälliges neues hinzugefügt). Mit Wahrscheinlichkeit $1 - p_{\text{size}} = 0,80$ wurden bestehende Merkmale verändert. Jedes Haar-Merkmal hat sowohl reelwertige, ganzzahlige als auch nominale Parameter. Daher wurden unterschiedliche Operatoren (ähnlich wie in den Experimenten in Abschnitt 8.1.3 bzw. 11.3) definiert: *changeSize*, *changeType* und *mutateEpsilons* wurden auf jedes Merkmal jeweils mit Wahrscheinlichkeit $p_{\text{mut}} = 0,50$ angewendet. Der Operator *changeSize* ändert Breite und Höhe unabhängig voneinander um $+1$ oder -1 . Der Operator *changeType* ändert den Typ des Merkmals zufällig. Der Operator *mutateEpsilons* verändert die Schwellwerte $\epsilon_i^{(0)}, \epsilon_i^{(1)}, \dots$ zufällig. Dabei kann auch deren Anzahl zufällig verringert und vergrößert werden. Maximal wurden jedoch drei dieser Schwellwerte pro Merkmal verwendet werden.

Reparatur-Mechanismen Im Rahmen der Optimierung sollen nur Lösungen betrachtet werden, die sinnvolle und zulässige Parameterkombinationen darstellen. Zum Beispiel sind teilweise negative Werte nicht sinnvoll (d_{int} und r) und Haar-Merkmale sollten nicht kleiner als $2 \times 2 \text{ Pixel}$ sein. Daher wurde für jeden Parameter ein zulässiger Wertebereich definiert und nach dem Mutieren sichergestellt, dass Werte innerhalb dieser Grenzen liegen.

10.3. Experimente

In diesem Abschnitt werden die durchgeführten Experimente und deren Ergebnisse beschrieben. Ziel der Experimente ist einerseits, den neu vorgeschlagenen Algorithmus (basierend auf Haar-Merkmalen) und das Original-Verfahren (basierend auf Census-Transformation) in Bezug auf Ihre Leistungsfähigkeit zu vergleichen. Andererseits sollen diese beiden verwandten Verfahren mit einem anderen populären Algorithmus zum Schätzen von optischem Fluss verglichen werden.

10.3.1. Aufbau

Als *Baseline* wird eine Implementierung des Lucas-Kanade-Ansatzes betrachtet, die zusätzlich Bildpyramiden verwendet (Bouguet, 2000). Es wurde auf eine öffentlich

verfügbare Implementierung aus der *OpenCV*-Bibliothek² zurückgegriffen.

Es ist kaum möglich, die Qualität von Algorithmen zur Schätzung von optischem Fluss auf realen Videodaten zu evaluieren. Dazu müssten die Bewegungen in der Bildebene während einer Videosequenz für alle Bildpunkte manuell erfasst werden, was bei typischen Auflösungen (Größenordnung 10^5 bis 10^6 Pixel) immensen Aufwand erfordern würde. Es ist daher üblich, zur Evaluierung synthetische Daten zu benutzen, die Ergebnisse übertragen sich ausreichend gut auf reale Daten (Vaudrey u. a., 2008).



Abbildung 10.1.: Bilder aus den *Middlebury*-Datensätzen, die in den Experimenten verwendet wurden.

Für den Vergleich der drei Verfahren werden zwei unterschiedliche Datensätze betrachtet. Der öffentlich verfügbare *Middlebury* Benchmark-Datensatz³ besteht zwar nur aus wenigen Einzelbildern, ist aber in der Literatur sehr populär. Die zugehörige Studie von Baker u. a. (2007) hatte zum Ziel, Algorithmen zu vergleichen, die *dichten* optischen Fluss schätzen. Aus diesem Grund können nicht alle Datensätze hier für die Evaluierung *spärlicher* Algorithmen verwendet werden. Es bleiben sechs Datensätze (*RubberWhale*, *Hydrangea*, *Grove2*, *Grove3*, *Urban2* und *Urban3*), die jeweils aus acht Bildern bestehen. Abbildung 10.1 zeigt beispielhaft drei Bilder aus diesen Datensätzen. Die Auswertung erfolgte jeweils auf dem letzten Bild der Folge. Dabei wurden bei jedem Algorithmus für alle Datensätze dieselben Parameter verwendet.

Als zweiter Datensatz wird eine lange synthetische Videosequenzen aus dem *Enpeda*-Benchmark⁴ berücksichtigt. Dieser Benchmark wurde von Vaudrey u. a. (2008) veröffentlicht und zur Evaluation verschiedener Algorithmen im Kontext von FAS verwendet (vgl. Abschnitt 9.3). Für die Optimierung von Parametern wurden hier die Bilder 80 bis 180 verwendet. Die folgenden Bilder (mehr als 200) dienen zur Auswertung, d. h., insbesondere zum Messen der Generalisierungsfähigkeit. Abbildung 9.2 auf Seite 84 zeigt beispielhaft drei Kamerabilder aus der Sequenz.

²<http://sourceforge.net/projects/opencvlibrary>

³<http://vision.middlebury.edu/flow>

⁴http://www.mi.auckland.ac.nz/index.php?option=com_content&view=article&id=44

Für alle sechs Kombinationen von Algorithmus und Datensatz wird jeweils eine Mehrziel-Optimierung durchgeführt, bei der alle Parameter des jeweiligen Verfahrens berücksichtigt werden. Der *OpenCV* Algorithmus hat acht Parameter. Sowohl für den Census-basierten *PowerFlow* als auch für die Variante basierend auf Haar-Merkmalen werden die Algorithmus-Parameter (siehe Tabelle 10.1) und die verwendeten Merkmale optimiert. Es wurden Census-Werte mit 12 Stellen berücksichtigt, die entsprechenden Positionen der betrachteten Pixel und die zugehörigen Schwellwerte wurden im Rahmen der Optimierung bestimmt. Für den neuen Ansatz basierend auf Haar-Merkmalen wurden die Merkmale selbst (Grundtyp und Größen) sowie Schwellwerte $\epsilon_i^{(0)}, \epsilon_i^{(1)}, \dots$ optimiert.

Tabelle 10.1.: Parameter des betrachteten Algorithmus.

Beschreibung	Einheit	
Maximale <i>discriminative power</i>	mdp	
Maximale Helligkeits-Differenz	d_{int}	%
Maximale Vektor-Länge	l_{max}	Pixel
Radius für Vorgänger-Suche	r	Pixel
Maximale Winkel-Differenz	d_{ang}	°
Maximale Längen-Differenz	d_l	%
Minimale Anzahl Vorgänger	p_{min}	

In den Experimenten wurde keine Ausschluss-Liste (*blacklist*) verwendet, wie von Stein ursprünglich vorgeschlagen. Solch eine Liste könnte erst nach der Parameterwahl (also nach der Optimierung) sinnvoll erstellt werden und würde dann die Ergebnisse höchstens weiter verbessern. Da auf diese Liste bei beiden Ansätzen verzichtet wurde, wird die Aussagekraft des Vergleichs nicht beeinflusst.

Für die Implementierung der Mehrziel-Optimierung wurde auf die frei verfügbare⁵ Bibliothek *Shark* zurückgegriffen (Igel u. a., 2008). Sie bietet insbesondere effiziente Algorithmen für die Hyper-Volumen-Berechnung.

Für jedes einzelne Experiment wurden fünf unabhängige Optimierungsläufe durchgeführt. Dabei wurden jeweils die Ergebnisse nach 1.000 Generationen betrachtet. Es wurde eine $(\mu + \lambda)$ -Selektion durchgeführt (vgl. Kapitel 5). Dabei wurden in jeder Generation $\lambda = 15$ neue Lösungen durch Rekombination und Mutation aus $\mu = 15$ Eltern-Lösungen erzeugt. Es wurde sichergestellt, dass alle Parameter in einem jeweils sinnvollen Wertebereich lagen (siehe Abschnitt 10.2).

Insbesondere wurden während der Optimierung Lösungen verworfen, die zu Fitness-Werten führten, die nicht relevant mit Blick auf die betrachtete Anwendung waren.

⁵<http://shark-project.sourceforge.net>

Hier wurden durchschnittlich mindestens 500 Flussvektoren pro Bild und höchstens 5 Pixel durchschnittlicher Endpunkt-Fehler gefordert. Für den endgültigen Vergleich wurden die besten Lösungen aus allen Optimierungsläufen gewählt.

10.3.2. Ergebnisse

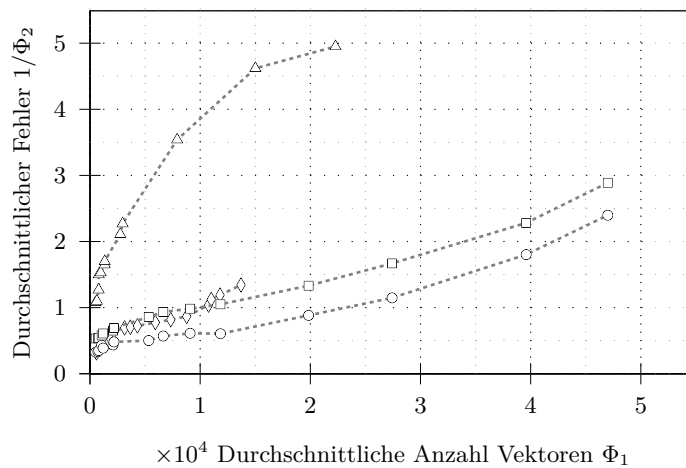


Abbildung 10.2.: Ergebnisse der Optimierungsläufe für den Middlebury-Datensatz: Pareto-optimale Lösungen für den Census-basierten *PowerFlow* Algorithmus (Dreiecke), den neuen Haar-Merkmal-basierten Ansatz mit Pixel- (Quadrate) und Sub-Pixel-Genauigkeit (Kreise) und den *OpenCV*-Algorithmus (Rauten).

Die Menge der optimierten Lösungen für die Middlebury-Datensätze sind in Abbildung 10.2 dargestellt. Sie zeigen für jeden Algorithmus mögliche *Trade-offs* zwischen Φ_1 , der mittleren Anzahl an Fluss-Vektoren pro Bild und $1/\Phi_2$, dem zugehörigen mittleren Fehler.

Mit dem Census-basierten *PowerFlow*-Algorithmus können im Testdatensatz bis zu 22.286 Fluss-Vektoren pro Bild geschätzt werden, der mittlere Fehler beträgt dann 4,95 Pixel. Der neue Haar-Merkmal-basierte Ansatz ermöglicht bis zu 46.999 Fluss-Vektoren pro Bild bei einem mittleren Fehler von 2,89 Pixel. Das optionale Verfeinern mit Sub-Pixel-Auflösung erhöht die Genauigkeit weiter. Mit dem Lucas-Kanade-Algorithmus aus der *OpenCV*-Bibliothek können maximal 13.669 Fluss-Vektoren pro Bild geschätzt werden bei einem Fehler von 1,46 Pixel. Abbildung 10.3 stellt die Ergebnisse nach Optimierung auf der *Enpeda*-Sequenz dar. Der *OpenCV* Algorithmus

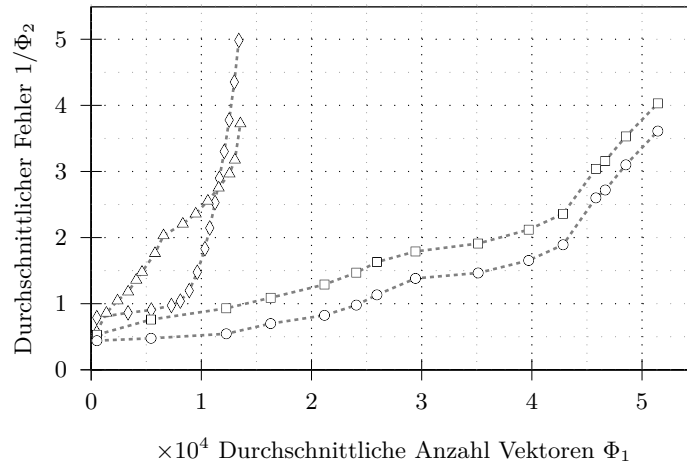


Abbildung 10.3.: Ergebnisse der Optimierungsläufe für Enpeda-Sequenzen: Pareto-optimale Lösungen für den Census-basierten Algorithmus (Dreiecke), den neuen Haar-Merkmal-basierten Ansatz mit Pixel- (Quadrate) und Sub-Pixel-Genauigkeit (Kreise) und den *OpenCV*-Algorithmus (Rauten).

erlaubt hier im Durchschnitt bis zu 12.415 Fluss-Vektoren pro Bild. Für den Census-basierten Ansatz erhält man bis zu 13.545 und für den neu vorgeschlagenen Algorithmus bis zu 51.434 Vektoren. Durch das vorgeschlagene Verfeinern können die Fehler des neuen Algorithmus weiter reduziert werden, im Schnitt um 0,4 Pixel.



Abbildung 10.4.: Optimierte Merkmalssätze für zwei unterschiedliche Lösungen. Die Merkmale haben die Größen (a) 11×11 , 5×11 , 12×12 , 12×10 , 11×12 und 7×7 . (b) 12×11 , 3×11 , 13×13 , 12×10 , 13×13 und 11×11 .

In Abbildung 10.4 sind optimierte Haar-Merkmale dargestellt, Tabelle 10.2 zeigt optimale Parameter für verschiedene Lösungen der beiden *PowerFlow*-Varianten.

Die Ergebnisse aller Verfahren auf der Testsequenz sind in Abbildung 10.5 dargestellt. Sowohl der Census-basierte Ansatz als auch der *OpenCV*-Algorithmus erreichen hier deutlich bessere Ergebnisse als auf der Trainingssequenz. Das bedeutet, dass es in diesem der Videosequenz „einfacher“ ist, optischen Fluss zu schätzen. Der neue Ansatz basierend auf Haar-Merkmalen erzielt ebenfalls leicht bessere Ergebnisse für Lösungen

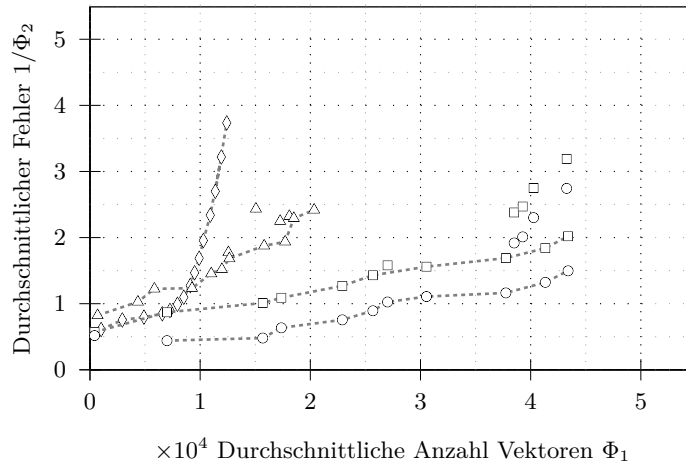


Abbildung 10.5.: Generalisierung der optimierten Lösungen auf dem zweiten Teil der Enpeda-Sequenz: Census-basierter Algorithmus (Dreiecke), Haar-Merkmal-basierter Ansatz mit Pixel- (Quadrate) und Sub-Pixel-Genauigkeit (Kreise) und *OpenCV*-Algorithmus (Rauten).

mit bis zu 4×10^4 Fluss-Vektoren, jedoch etwas schlechtere Ergebnisse für Lösungen mit mehr Vektoren.

Tabelle 10.2.: Optimierte Parameter unterschiedlicher Lösungen beider Ansätze für die Enpeda-Sequenz.

Φ_1	$1/\Phi_2$	mdp	d_{int}	l_{max}	r	d_{ang}	d_l	p_{min}
Census-Transformation-basiert								
502	0,57	33	1,2	43	0,8	0,32	0,11	2
6.569	2,03	70	0,5	40	5	0,05	0,1	1
13.545	3,72	70	1	45	5	0,23	0,1	1
Haar-Merkmal-basiert								
521	0,53	33	0,6	80	0,6	0,23	0,1	3
12.264	0,93	70	1,6	40	1,6	0,05	0,15	3
25.940	1,63	70	2	40	5	0,05	0,1	2
39.709	2,12	70	2	40	3,3	0,05	0,1	1
51.435	4,03	69	2	46	4,6	0,15	0,74	1

Die optimierten Lösungen für den *Census-PowerFlow* führen zu Laufzeiten zwischen 200 ms und 400 ms auf einem normalen Desktop-PC mit einem 2,2 GHz-Prozessor. Für den *Haar-PowerFlow* ergeben sich Laufzeiten zwischen 250 ms und 600 ms. Da beide Verfahren die gleiche Zuordnung (zeitliche Integration) zurückgreifen, ist die Ähnlich-

keit der Laufzeiten nicht überraschend. Die Implementierung des dritten Algorithmus aus der *OpenCV*-Bibliothek hat bei den optimierten Lösungen Laufzeiten zwischen 200 *ms* und 1.100 *ms*.

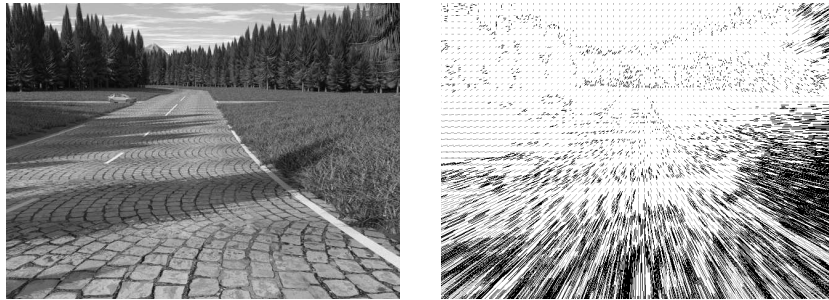


Abbildung 10.6.: Beispiel für ein Kamerabild aus der Enpeda-Sequenz (links) und Ergebnis des neuen Haar-Merkmal-basierten Algorithmus (rechts), geschätzte Flussvektoren sind in schwarz dargestellt.

Typische Ergebnisse des neu vorgeschlagenen Algorithmus sind in Abbildung 10.6 dargestellt. Hier wurde eine der optimierten Lösungen verwendet, die sehr kleinen Fehler mit einer ausreichend großen Anzahl geschätzter Vektoren vereint. Diese Lösung eignet sich gut für Anwendungen im Bereich von FAS. Mithilfe der geschätzten Fluss-Vektoren können weitere Applikationen angewendet werden, zum Beispiel Objektsegmentierung und -tracking oder Rekonstruktion von 3D-Information.

10.4. Diskussion

In den im letzten Abschnitt beschriebenen Experimenten wurden die untersuchten Verfahren auf denselben Trainingsdaten mit dem gleichen Verfahren optimiert. Da insbesondere der *Enpeda*-Datensatz sehr umfangreich ist, lassen sich einige zuverlässige Aussagen ableiten. Zunächst konnte gezeigt werden, dass die betrachteten Varianten des *PowerFlow*-Algorithmus dem klassischen Verfahren praktisch in allen Bereichen (d. h. für verschiedene Anforderungen an Anzahl und Güte der geschätzten Fluss-Vektoren) überlegen ist.

Der *Census-PowerFlow* und der *Haar-PowerFlow* teilen die meisten Parameter, da sie auf dieselbe zeitliche Verarbeitung zurückgreifen. Daher lassen sich die beobachteten signifikanten Unterschiede in der Leistungsfähigkeit dieser beiden Algorithmen eindeutig auf die verwendeten Merkmale zurückführen. Die neue Methode, die Haar-Merkmale nutzt, hat für jede mögliche Anzahl an Fluss-Vektoren einen kleineren Fehler. Diese

Unterschiede werden noch deutlicher, wenn die Sub-Pixel-Verfeinerung durchgeführt wird, die das neue Verfahren optional unterstützt.

Die neue Methode ermöglicht es insbesondere, sehr viele Fluss-Vektoren zu schätzen. In den *Enpeda*-Sequenzen konnte die Anzahl der möglichen Ergebnisse fast um den Faktor 5 gesteigert werden. Die Abdeckung des Kamerabildes (d. h., der Anteil der Bildpunkte für die Fluss-Vektoren geschätzt werden können) ist mit über 15 % für einen spärlichen Fluss-Algorithmus durchaus bemerkenswert.

Es gibt einen möglichen Einwand gegen die Ergebnisse. Der neue Haar-Merkmal-basierte Ansatz hat mehr freie Parameter da die Merkmale selbst mehr Freiheitsgrade haben als die Census-Transformation. Daher kann dieser Ansatz potentiell mehr von einer automatischen Parameter-Optimierung profitieren als der Census-basierte Ansatz. Umgekehrt lässt sich jedoch argumentieren, dass es genau diese erhöhte Flexibilität ist, die für den neuen Algorithmus spricht. Außerdem wurde festgestellt, dass die meisten der optimierten Lösungen gute Generalisierungsfähigkeit gezeigt haben, d. h., keine Überanpassung (*over-fitting*) stattgefunden hat. Dies wäre bspw. zu erwarten, wenn der Algorithmus „zu viele“ freie Parameter hätte.

Die Laufzeit der Algorithmen war kein Kriterium für die durchgeführten Optimierungen. Die gefundenen Lösungen führen daher zu relativ hohen Rechenzeiten. Dieser Umstand wird am Beispiel des Haar-*PowerFlow* deutlich: Alle optimierten Lösungen verwenden sechs Haar-Merkmale, was dem Maximum bei der durchgeführten Optimierung entspricht. Die Laufzeit könnte als zusätzliches Kriterium bei der Optimierung betrachtet werden.

In den optimierten Lösungen für den Haar-*PowerFlow* lassen sich auffällige Gemeinsamkeiten erkennen. Sie lassen Rückschlüsse darauf zu, wie die Parameter für den vorgeschlagenen Algorithmus gewählt werden sollten.

Die meisten optimierten Haar-Merkmale waren verhältnismäßig groß, 85 % der Merkmale der endgültig besten Lösungen waren bspw. in beiden Dimensionen größer als 10 Pixel. Es scheint also im betrachteten Algorithmus vorteilhaft zu sein, recht große Nachbarschaften einzubeziehen. Wenn Haar-Merkmale verwendet werden, ist dies ohne Nachteile möglich, da die Rechenzeit (im Gegensatz zu anderen Filtern) unabhängig von der Größe ist.

Bei den Algorithmus-Parametern lässt sich an vielen Stellen erkennen, wie diese mit dem *Trade-off* zwischen Anzahl an Vektoren und Güte korreliert sind. Trotzdem sind die Abhängigkeiten vielfältig und das Zusammenspiel der Parameter ist komplex. Auch deshalb ist die automatische Optimierung dem manuellen Einstellen vorzuziehen. Schließlich muss beachtet werden, dass „optimale Parameter“ immer vom konkret betrachteten Problem abhängig sind.

11. Effizientes Update der Kovarianzmatrix bei iterierter LDA

Für viele praktische Anwendungen digitaler Bildverarbeitung (Automobilbereich, Robotik, Sportanalyse, etc.) werden Klassifikatoren benötigt, die in geringer Rechenzeit entscheiden können. Um solche (Echtzeit-)anforderungen einzuhalten, werden häufig einfache Klassifikatoren eingesetzt, die trotzdem gute Ergebnisse erzielen. Lineare Diskriminanzanalyse (LDA) zum Trainieren von linearen Klassifikatoren (siehe Abschnitt 2.2) eignet sich in solchen Szenarien hervorragend: Trotz der Einfachheit des Ansatzes lassen sich in der Praxis häufig sehr gute Ergebnisse erzielen (Hastie u. a., 2001). Auch in der vorliegenden Arbeit wurden in den Abschnitten 8.1 bzw. 8.2 entsprechende Ergebnisse dokumentiert.

Wenn verhältnismäßig einfache Klassifikatoren eingesetzt werden sollen, spielt die Repräsentation der Eingabedaten eine entscheidende Rolle. Während komplexere maschinelle Lernverfahren, bspw. Support-Vektor-Maschinen (Vapnik, 1998), in der Lage sind, beim Training geeignete nicht-lineare Transformationen selbst zu bestimmen, gilt genau dies nicht für lineare Klassifikatoren.

Um einen linearen Klassifikator mit hoher Erkennungsrate zu erhalten, ist es also notwendig, eine Menge von Merkmalen zu finden, die die „dominanten Nicht-Linearitäten“ („dominant non-linearities“, siehe Bertsekas und Tsitsiklis (1996)) beinhaltet. Bei Anwendungen, die auf Bilddaten basieren, steht eine immense Zahl möglicher Merkmale zur Verfügung. Eine vollständige Suche, also Evaluierung aller möglichen Merkmalskombinationen, ist jedoch praktisch unmöglich (Jain u. a., 2000), vgl. Abschnitt 8.1. Als Ersatz für eine vollständige Suche bieten sich in der Praxis Heuristiken an, um möglichst gute Merkmalsätze automatisch zu bestimmen. Da solche Heuristiken typischerweise iterativ arbeiten, müssen Klassifikatoren häufig wiederholt trainiert werden.

Im ersten Abschnitt dieses Kapitels wird zunächst die hier betrachtete Problemstellung definiert und von verwandten Problemen abgegrenzt. In Abschnitt 11.2 wird ein neues effizientes Vorgehen beschrieben¹. Die Leistungsfähigkeit des neuen Verfahrens und

¹Teile der in diesem Abschnitt vorgestellten Arbeiten und Ergebnisse wurden bereits in Salmen u. a. (2010) veröffentlicht.

dessen Genauigkeit wurde in Experimenten untersucht (Abschnitt 11.3), die Ergebnisse werden schließlich in Abschnitt 11.4 diskutiert.

11.1. Problemstellung

Um eine Menge möglichst gut geeigneter Merkmale für einfache Klassifikatoren zu finden, muss auf passende Heuristiken zurückgegriffen werden. Dabei gibt es zwei prinzipiell unterschiedliche Möglichkeiten: Entweder das Bestimmen einer guten Auswahl aus einem vorgegebenen *Pool* (Merkmalsselektion) oder automatische Konstruktion möglichst guter Merkmale (Merkmalsoptimierung). Für beide Zwecke wurden verschiedene Verfahren vorgeschlagen und erfolgreich angewendet, beispielsweise evolutionäre Optimierung (Beyer, 2007), simuliertes Abkühlen (Kirkpatrick u. a., 1983, Cerný, 1985), Tabu-Suche (Glover und Laguna, 1997), siehe auch Jain u. a. (2000).

Die entsprechenden Verfahren haben alle gemeinsam, dass die Lösung *iterativ* bestimmt wird und dabei einzelne Merkmale fortwährend ersetzt und / oder geändert werden. Nach diesen Änderungen ist jeweils das Trainieren eines neuen Klassifikators nötig. Die kanonische Schleife für Merkmalsselektion und -optimierung hat die Form

1. Erzeuge einen möglichen Merkmalsatz basierend auf einer oder mehreren vorhandenen Lösungen (oder, wenn noch keine Lösung vorhanden, zufällige Initialisierung).
2. Trainiere einen Klassifikator basierend auf dem neuen Merkmalsatz.
3. Teste den Klassifikator.
4. Springe zu Schritt 1, wenn Abbruchkriterium nicht erreicht.

Während das Ändern des Merkmalsatzes (Schritt 1) typischerweise wenig Aufwand bedeutet, ist das Trainieren selbst bei einfachen Verfahren deutlich aufwendiger. Das Training mit LDA beinhaltet das Invertieren der Kovarianzmatrix, was konkret bei n Merkmalen $\Omega(n^2)$ Operationen erfordert.

Lineare Diskriminanzanalyse bestimmt, basierend auf einer Menge von Trainingsdaten S , eine Klassifikations-Vorschrift. Für Klassifikationsaufgaben im Kontext von Bildverarbeitung besteht S typischerweise aus Merkmalsvektoren $x_1, \dots, x_l \in \mathbb{R}^n$ die durch eine Funktion $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ aus „rohen“ Trainingsdaten, einer Menge von Bildern $R = \{r_1, \dots, r_l\}$, extrahiert wurden (zur einfacheren Darstellung wird hier angenommen $r_i \in \mathbb{R}^m$ für $i = 1, \dots, l$).

In Anwendungen von LDA in der Praxis können die verfügbaren Daten R sich über die Zeit ändern. Dies gilt bspw. für sog. *Online*-Klassifikations-Szenarien, bei denen ein

erster Klassifikator auf initial verfügbaren Daten S_0 (berechnet aus R_0) trainiert wird. Danach können zur Laufzeit neue Daten gesammelt werden, so dass neue Trainingsmengen R_1, R_2, \dots verfügbar sind. Es ist dann eine wichtige Aufgabe, den Klassifikator entsprechend anzupassen.

Aber auch in *Offline*-Szenarien ist es wichtig, auf veränderte Trainingsdaten effizient reagieren zu können. Ein relevantes Beispiel stellen Meta-Heuristiken dar, die während des Trainings eingesetzt werden (z. B. Merkmals-Selektion und -Optimierung, *cross-validation*). Sie erfordern ein iteriertes Training auf unterschiedlichen Trainingsmengen S_0, S_1, \dots , die nicht unabhängig voneinander sind. Für reale Daten ist die naive Lösung, nämlich für jeden Datensatz den Klassifikator neu zu trainieren, oft kaum praktikabel.

Aus den oben geschilderten Gründen wurden diverse Lösungen, insbesondere effiziente Update-Gleichungen für LDA, vorgeschlagen. Dabei ist entscheidend, welche Art von Änderungen an den Trainingsdaten, also von R_i zu R_{i+1} und damit von S_i zu S_{i+1} , tatsächlich erwartet werden. Außerdem können Anforderungen bzgl. der Funktion f zur Merkmalsextraktion relevant sein.

Generell können Änderungen an den Trainingsdaten die Anzahl der Trainingsdaten $|R_i|$, die Anzahl der Klassen oder die Repräsentation von Daten in S_i betreffen. Änderungen an S_i können dabei auch durch Änderungen von f bedingt sein.

Für den Spezialfall $R_0 \subseteq R_1 \subseteq \dots$ und $S_0 \subseteq S_1 \subseteq \dots$, also eine kontinuierlich wachsende Menge an Trainingsdaten mit fester Merkmalsextraktion, wurden verschiedene effiziente Lösungen (darunter Approximationen) vorgeschlagen, siehe z. B. Pang u. a. (2005), Kim u. a. (2007) und Huang u. a. (2009).

Tapp und Kemsley (2008) schlagen Formeln für ein effizientes LDA-Update vor für den Fall, dass *cross-validation* durchgeführt wird und dabei in jedem Schritt nur genau ein einziges Trainingsbeispiel ausgetauscht wird, also $|R_i \cap R_{i+1}| = |R_i| - 1$, während die Merkmalsextraktion f nicht verändert wird.

In der vorliegenden Arbeit wird ein weiteres, unterschiedliches Szenario betrachtet. Dafür wird vorausgesetzt, dass sich die Trainingsbeispiele selbst nicht ändern, also $R_i = R_{i+1}$ für alle i , aber ihre Repräsentation S_i sich aufgrund geänderter Merkmalsberechnung f ändern kann. Eine Methode, um genau diesen Fall effizient zu behandeln, wird benötigt, wenn z. B. Merkmals-Optimierung oder -Selektion durchgeführt werden sollen.

Im Folgenden soll die betrachtete Aufgabe etwas formaler definiert werden: Trainiere einen Klassifikator basierend auf LDA in Schritt i möglichst effizient, wenn Folgendes gegeben ist:

- Ein Klassifikator aus dem letzten Zeitschritt $i - 1$, trainiert mittels LDA
- Die zugehörigen Trainingsdaten S_{i-1}
- Neue Trainingsdaten S_i
- $f_i \approx f_{i-1}$

Die Notation $f_i \approx f_{i-1}$ soll dabei beschreiben, dass sich die Repräsentation der Trainingsdaten (aus der festen Menge R) durch Änderungen nur in wenigen Dimensionen von S_{i-1} zu S_i ändern.

11.2. Vorgeschlagenes Vorgehen

Im Folgenden wird eine Möglichkeit vorgestellt, das wiederholte Training mit LDA in den Szenarien deutlich effizienter durchzuführen, die am Ende des vorangegangenen Abschnitts definiert wurden. Dabei wird die Inverse der Kovarianzmatrix nicht in jedem Schritt (nach Änderung des Merkmalsatzes) neu berechnet, sondern inkrementell geändert. Das Vorgehen resultiert lediglich in $\Theta(n^2)$ Operationen für jedes einzelne Merkmal das geändert wird. Diese Zahl entspricht der asymptotisch unteren Grenze.

Gegeben sei eine Menge von Trainingsdaten $S = \{(x_1, y_1), \dots, (x_\ell, y_\ell)\} \in (\mathbb{R}^n \times \{c_1, \dots\})^\ell$. Nach Änderung eines einzelnen Merkmals sei $\hat{S} = \{(\hat{x}_1, y_1), \dots, (\hat{x}_\ell, y_\ell)\} \in (\mathbb{R}^n \times \{c_1, \dots\})^\ell$ der resultierende neue Trainingsdatensatz. Aufgrund der getroffenen Annahmen unterscheiden die Merkmalsvektoren x_k und \hat{x}_k für $k = 1, \dots, l$ sich höchstens in der Komponente i während alle anderen Einträge identisch sind.

Die Kovarianzmatrix, die bei der LDA im Training auf S verwendet wird, bezeichnen wir mit Σ . Die Kovarianzmatrix, die beim Training auf \hat{S} verwendet wird, mit $\hat{\Sigma}$. Wenn wir, wie oben beschrieben, davon ausgehen, dass beim iterierten Training in jedem Schritt LDA durchgeführt wird, dann muss die invertierte Kovarianzmatrix Σ^{-1} bekannt sein, wenn $\hat{\Sigma}^{-1}$ berechnet wird.

Das Bestimmen von $\hat{\Sigma}^{-1}$ kann nun – verglichen mit Neuberechnung – effizienter durchgeführt werden, wenn die *Woodbury-Matrix-Identität*

$$\hat{A}^{-1} = (A + UCV)^{-1} = A^{-1} - A^{-1}U \left(C^{-1} + VA^{-1}U \right)^{-1} VA^{-1} \quad (11.1)$$

anwendbar ist. Die Terme $A^{-1}U$ und VA^{-1} sind einfache Matrix-Produkte. Statt die Inverse \hat{A}^{-1} selbst zu berechnen, müssen hier die Inversen C^{-1} und $(C^{-1} + VA^{-1}U)^{-1}$ berechnet werden. Um von der Formel zu profitieren, ist es also entscheidend, dass die Matrix C möglichst geringe Dimensionalität hat.

Für die Matrizen im hier betrachteten Fall gilt

$$\hat{\Sigma}^{-1} = (\Sigma + G)^{-1} \quad (11.2)$$

wobei

$$G = \hat{\Sigma} - \Sigma = \begin{pmatrix} 0 & \cdots & g_{1i} & \cdots & 0 \\ 0 & \cdots & \vdots & \cdots & 0 \\ g_{i1} & \cdots & g_{ii} & \cdots & g_{in} \\ 0 & \cdots & \vdots & \cdots & 0 \\ 0 & \cdots & g_{ni} & \cdots & 0 \end{pmatrix} \quad (11.3)$$

eine Matrix mit Rang 2 ist.

Damit die Formel (11.1) sinnvoll anwendbar ist, wird G folgendermaßen in die Form $G = UCV$ zerlegt mit

$$U = \begin{pmatrix} g_{1i} & 0 \\ g_{2i} & 0 \\ \vdots & \vdots \\ g_{(i-1)i} & 0 \\ g_{ii} & 1 \\ g_{(i+1)i} & 0 \\ \vdots & \vdots \\ g_{ni} & 0 \end{pmatrix}, \quad (11.4)$$

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (11.5)$$

und

$$V = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ g_{1i} & g_{2i} & \cdots & g_{(i-1)i} & 0 & g_{(i+1)i} & \cdots & g_{ni} \end{pmatrix}. \quad (11.6)$$

Damit kann $\hat{\Sigma}^{-1}$, gegeben Σ^{-1} , folgendermaßen berechnet werden:

$$\hat{\Sigma}^{-1} = (\Sigma + UCV)^{-1} = \Sigma^{-1} - \Sigma^{-1}U \left(C^{-1} + V\Sigma^{-1}U \right)^{-1} V\Sigma^{-1}. \quad (11.7)$$

Für die beiden Matrixprodukte in (11.7) müssen jeweils $2n$ Elemente betrachtet werden. Es ist hier $C^{-1} = C$, damit verursacht diese Invertierung keinen Rechenaufwand.

Die einzige Matrix die tatsächlich noch invertiert werden muss, $(C^{-1} + V\Sigma^{-1}U)$, hat die Größe 2×2 und kann daher folgendermaßen einfach invertiert werden

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}. \quad (11.8)$$

Die resultierenden Matrizen auf der rechten Seite von (11.7) zu multiplizieren und zusammenzufassen hat Komplexität $\Theta(n^2)$. Damit ergibt sich für das vorgeschlagene inkrementelle Update insgesamt eine Komplexität von $\Theta(n^2)$.

Da der Update-Schritt mehrmals nacheinander ausgeführt werden kann, ist die resultierende Gesamtkomplexität $\Theta(Nn^2)$ wenn Änderungen in N Dimensionen zu behandeln sind.

Von dem neuen Update innerhalb einer typischen Schleife zur Merkmals-Selektion oder -Optimierung (vgl. Abschn. 11.1) zu profitieren, ist unkompliziert: Die Kovarianzmatrix für die LDA muss nur einmalig direkt invertiert werden, nämlich zur Initialisierung des Algorithmus. Alle folgenden Änderungen am Merkmalsatz können danach mithilfe des hier vorgestellten inkrementellen Updates umgesetzt werden.

Die Änderung am Merkmal mit Index i führen dazu, dass sich (im Allgemeinen) in den Merkmalsvektoren aller Beispiele neue Werte für Komponente i ergeben. Um nun einen neuen Klassifikator zu erhalten, müssen konkret die folgenden Schritte durchgeführt werden:

1. Berechne neue Klassen-Zentren $\hat{\mu}_{ki}$
2. Berechne die geänderten Kovarianzmatrix-Einträge $\hat{c}_{1i}, \dots, \hat{c}_{ni}$. Da die Matrix symmetrisch ist, damit gleichzeitig auch $\hat{c}_{i1}, \dots, \hat{c}_{in}$
3. Berechne die Werte g_{1i}, \dots, g_{ni} und g_{i1}, \dots, g_{in} aus (11.3)
4. Berechne U und V nach Gleichung (11.4) bzw. (11.6)
5. Berechne $\hat{\Sigma}^{-1}$ mit der Update-Gleichung (11.7)

Diese Schritte werden wiederholt, wenn mehr als ein Merkmal geändert wird.

Um basierend auf der LDA einen Klassifikator zu erhalten, müssen die neu berechneten Klassen-Zentren $\hat{\mu}_k$ und die angepasste Kovarianzmatrix $\hat{\Sigma}^{-1}$ berücksichtigt werden. Der neue Gewichtsvektor (vgl. Abschnitt 2.2) ergibt sich als

$$\hat{w} = \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_2) \quad (11.9)$$

und die additive Konstante (*bias*)

$$\hat{b} = -\frac{1}{2}(\hat{\mu}_1 + \hat{\mu}_2)^\top \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_2). \quad (11.10)$$

11.3. Experimente

Ein Ziel der Experimente war, die *numerische Stabilität* der neuen Gleichung zu untersuchen. Als Kriterium für die Genauigkeit einer Kovarianzmatrix $\hat{\Sigma}^{-1}$ können die Terme $E_l = \hat{\Sigma}^{-1}\hat{\Sigma}$ und $E_r = \hat{\Sigma}\hat{\Sigma}^{-1}$ herangezogen werden. In beiden Fällen sollte sich für E_l bzw. E_r die Einheitsmatrix I ergeben. Die *Max-Norm* $\|A\|_\infty$ einer Matrix A ist gegeben durch

$$\|A\|_\infty = \max \{|a_{ij}|\}, \quad (11.11)$$

sie kann verwendet werden, um den Fehler e am Ende des Trainingsprozesses folgendermaßen zu bestimmen

$$e = \max \{\|I - E_l\|_\infty, \|I - E_r\|_\infty\}. \quad (11.12)$$

Außerdem wurden Laufzeiten gemessen um zu überprüfen ob und wie weit die theoretischen Betrachtungen sich in der praktischen Anwendung bestätigen.

Als Anwendungsbeispiele in den Experimenten wurde sowohl ein künstliches Merkmals-Selektion-Problem als auch ein reales Merkmals-Optimierungs-Problem betrachtet. In beiden Fällen wurden ein entsprechender Algorithmus, der in jedem Schritt ein neues Training durchführt, mit einer Variante verglichen, die das hier vorgestellte *Update* nutzt.

11.3.1. Aufbau

Für erste Untersuchungen wurde das klassische Beispiel von Trunk (1979) betrachtet, vgl. auch Jain u. a. (2000). In diesem künstlichen 2-Klassen-Problem müssen n -dimensionale Merkmalsvektoren $x \in \mathbb{R}^n$ klassifiziert werden. Beide Klassen haben *a priori* die gleiche Wahrscheinlichkeit. Die Wahrscheinlichkeit, den Wert $x \in \mathbb{R}^n$ für eine feste Klasse $y \in \{\pm 1\}$ zu beobachten ist normalverteilt mit

$$p(x | y) = \frac{1}{\sqrt{2\pi}} e^{-\|x - ym\|^2/2}, \quad (11.13)$$

wobei ym der Erwartungswert ist und die Kovarianzmatrix die Einheitsmatrix. Die Komponenten m_i des Vektors m werden gewählt als $m_i = \sqrt{1/i}$ mit $i = 1, \dots, n$.

Ein Bayes-optimaler Klassifikator (Hastie u. a., 2001) muss nun eine Eingabe x der Klasse 1 zuordnen wenn $x^\top m > 0$ und sonst der Klasse -1 . Der Bayes-Fehler ergibt sich dann als

$$P_e \left(\sqrt{\sum_{i=1}^n 1/i} \right) \quad (11.14)$$

mit der Hilfsfunktion

$$P_e(r) = \int_r^\infty \frac{1}{\sqrt{2\pi}} e^{-z^2/2} dz. \quad (11.15)$$

Da $\sum_{i=1}^n 1/i$ für steigende n divergiert, kann man leicht sehen, dass der Fehler des Bayes-optimalen Klassifikators fällt und gegen 0 geht, wenn die Anzahl der Merkmale n erhöht wird (Trunk, 1979).

Ein Klassifikator, der aus einer festen Trainingsmenge $S = \{(x_1, y_1), \dots, (x_\ell, y_\ell)\}$ lediglich die Parameter der klassenweisen Normalverteilungen schätzt und dann darauf basierend optimal trennt (*plug-in classifier*), konvergiert mit $\ell \rightarrow \infty$ für beliebig große n gegen den Bayes-optimalen Klassifikator. Allerdings hat Trunk gezeigt, dass für festes ℓ und $n \rightarrow \infty$ die Fehler-Wahrscheinlichkeit gegen die Wahrscheinlichkeit konvergiert, die mit bloßem Raten erzielt werden kann (*chance level*).

Für die Experimente hier wurde das folgende Szenario betrachtet. Aus einer Menge \mathcal{P} mit 2.000 Merkmalen sollen die besten 100 ausgewählt werden. Es wurden zufällig $\ell = 6.000$ Trainingsbeispiele (jeweils 3.000 für beide Klassen) gemäß (11.13) erzeugt. Die Beispiele wurden in drei Mengen geteilt, um *cross-validation* zu ermöglichen. In jeder Untermenge waren also 1.000 Beispiele pro Klasse.

Die Untersuchungen sollten mit einem möglichst einfachen Algorithmus durchgeführt werden. Daher wurde die einfachste Evolutionsstrategie verwendet: Eine Lösung \mathcal{C} (hier die Menge der selektierten Merkmale) wird iterativ verbessert indem zufällige Änderungen vorgenommen und die Auswirkungen evaluiert werden. In jeder Iteration wurde der *cross-validation*-Fehler des Klassifikators auf dem veränderten Merkmalsatz bestimmt. Alle Änderungen wurden verworfen, wenn der Fehler gestiegen war und sonst beibehalten (*stochastic hill-climbing*), siehe Kapitel 5.

Die Lösung \mathcal{C} wurde zu Beginn mit $n = 100$ zufälligen Merkmalen aus \mathcal{P} initialisiert. In jeder der $t_{\max} = 10^6$ Generationen wurden dann zufällige Merkmale aus \mathcal{C} durch zufällige andere aus $\mathcal{P} \setminus \mathcal{C}$ ersetzt. Die Anzahl der auf diese Weise ausgetauschten Merkmale N wurde in jeder Iteration t gemäß einer Poisson-Verteilung mit Erwartungswert $\lambda = |\mathcal{C}| \cdot 10^{-1 - \frac{t}{t_{\max}}}$ bestimmt. Insgesamt wurden 40 unabhängige Läufe dieses Experiments durchgeführt, dabei wurde \mathcal{P} jedes Mal neu zufällig initialisiert.

Als zweites Experiment wurde ein praktisches Beispiel für Merkmals-Optimierung berücksichtigt, dabei wurden teilweise die in Abschnitt 8.1 beschriebenen Experimente nachgebildet. Für den Benchmark-Datensatz zur Fußgängererkennung von Munder und Gavrilu (2006) sollte eine Menge von Haar-Merkmalen gefunden werden, die möglichst gut geeignet ist, um die beiden Klassen linear zu trennen. Im Vergleich zu den in Abschnitt 8.1 beschriebenen Experimenten wurde hier ein etwas vereinfachter Aufbau betrachtet, da das Ziel die Evaluierung des neuen Verfahrens war und nicht die

Optimierung von Merkmalen für das betrachtete Problem. Es wurden lediglich Merkmalsätze mit der Größe 7 betrachtet, die ausgewählten Merkmale wurden aber jeweils an jedem zweiten Pixel berechnet. Bei der Bildgröße von 18×36 ergeben sich so Merkmalsvektoren der Länge $7 \cdot 136 = 952$.

Für die Optimierung wurden 5 verschiedene Typen von Haar-Merkmalen berücksichtigt, die Größen wurden auf 3×3 bis 16×16 Pixel eingeschränkt. Damit ergeben sich bereits mehr als 10^{20} mögliche Merkmalsätze und eine vollständige Suche ist praktisch unmöglich.

Zur Optimierung wurde wieder eine möglichst einfache Evolutionsstrategie verwendet: In jeder Iteration wurde eines der sieben Merkmale zufällig bestimmt und entweder mutiert (Änderung von Größe und / oder Typ) oder durch ein zufälliges neues Merkmal ersetzt. Je nachdem, ob der *cross-validation*-Fehler durch diese Änderung kleiner wurde oder nicht, wurden die Änderungen beibehalten oder verworfen. Für die Merkmalsoptimierung wurden 5 Läufe mit jeweils 500 Generationen betrachtet.

Die *C++* Implementierung für alle beschriebenen Experimente basierten auf der frei verfügbaren Software-Bibliothek *Shark* (Igel u. a., 2008). Außerdem wurde *OpenMP*² verwendet um von der Parallelisierung auf dem Testrechner (Intel Core 2 Duo CPU E6300 mit 3 GB RAM) profitieren zu können.

11.3.2. Ergebnisse

Im Folgenden werden die Ergebnisse der durchgeführten Experimente dargestellt.

Bei der Merkmals-Selektion für das künstliche Beispiel war der größte Fehler bzgl. der tatsächlichen Inverse nach Gleichung (11.12) am Ende eines Optimierungslaufes $9 \cdot 10^{-15}$. Der durchschnittliche Fehler über alle Läufe betrug $6 \cdot 10^{-15}$. Abweichungen in dieser Größenordnung haben praktisch keine Auswirkungen auf die Güte des resultierenden Klassifikators. Die durchschnittliche Laufzeit für die Merkmals-Selektion betrug 63 min, während die Referenz-Implementierung ohne inkrementelles *Update* etwa 40-Mal so lange brauchte.

Das betrachtete Testproblem erlaubt auch, die Konvergenz der einfachen Meta-Heuristik zu beurteilen, die hier gewählt wurde. Das Histogramm in Abbildung 11.1 zeigt, wie oft jedes der ersten 200 Merkmale in den durchgeführten Optimierungsläufen für den endgültigen Merkmalsatz gewählt wurde. Es ist beachtenswert, dass selbst das einfache Verfahren hier die wichtigsten Merkmale zuverlässig gefunden hat.

²<http://openmp.org>

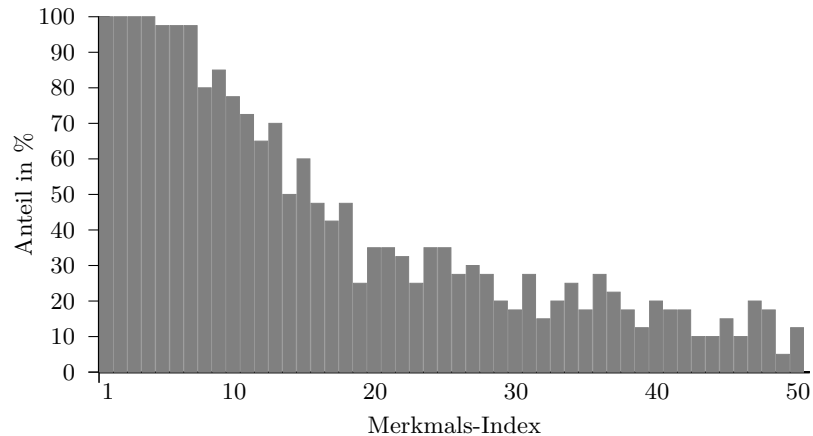


Abbildung 11.1.: Relative Häufigkeit mit der Merkmal $i = 1, \dots, 50$ für den finalen Merkmalsatz gewählt wurde. Je kleiner i , desto diskriminanter ist das Merkmal.

Im betrachteten Merkmals-Optimierungs-Problem wurden die folgenden Ergebnisse beobachtet. Der größte numerische Fehler nach einem einzelnen Lauf betrug $5,2 \cdot 10^{-5}$, der durchschnittliche Fehler $7,1 \cdot 10^{-6}$. Diese Abweichungen haben in der Praxis keine Auswirkung.

Die durchschnittliche Laufzeit für die Optimierung der Haar-Merkmale für die Fußgängererkennung war $22,4 h$. Die Implementierung die nicht das inkrementelle *Update* verwendet benötigte für dieselbe Optimierung etwa die 2,5-fache Rechenzeit. Der gemessene Geschwindigkeits-Gewinn war damit nicht so hoch wie im ersten Experiment. Das lässt sich folgendermaßen erklären: Das Ändern eines Merkmals hat Auswirkungen auf ein Siebtel der Einträge des Merkmalsvektors, da insgesamt sieben Merkmale verwendet werden und jedes im ganzen Bild berechnet wird. Der Geschwindigkeits-Gewinn ist trotzdem immer noch signifikant für reale Anwendungen.



Abbildung 11.2.: Optimierte Merkmale für Fußgängererkennung. Die Merkmale sind 16×8 , 3×3 , 6×5 , 3×9 , 4×3 , 4×4 und 3×5 Pixel groß.

Der beste Merkmalsatz für das Erkennen von Fußgängern ist in Abbildung 11.2 dargestellt. Die Ergebnisse auf dem Testdatensatz wurden nach der von Munder und Gavrilu (2006) vorgeschlagenen Methode gemessen, vgl. auch Abschnitt 8.1.3. Abbildung 11.3 zeigt die resultierende *ROC*-Kurve, die alle möglichen *Trade-offs* zwischen Detektionsrate und Fehlalarm-Rate beschreibt.

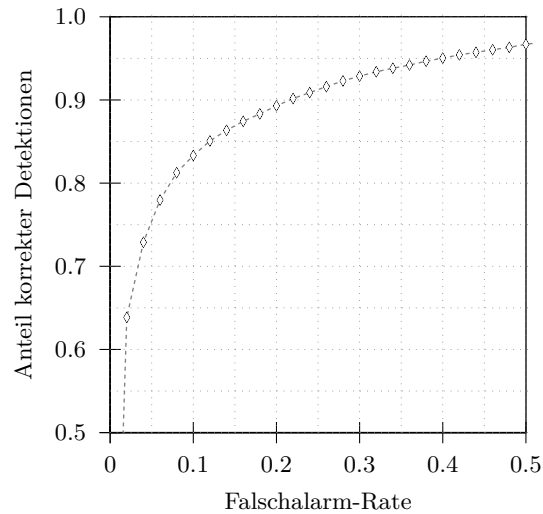


Abbildung 11.3.: *ROC*-Kurve des optimierten linearen Klassifikators.

Der hier untersuchte Klassifikator ist extrem schnell: Er ist lediglich erforderlich, sieben Haar-Merkmale an jedem zweiten Bildpunkt zu berechnen (insgesamt 952 Merkmale, s. o.). Die anschließende Klassifikation einer *ROI* der Größe 18×36 Pixel erfordert dann lediglich 952 Multiplikationen und Additionen. Das Verfahren eignet sich damit besonders gut, um eine initiale Detektion im gesamten Kamerabild durchzuführen. Für die endgültige Entscheidung können aufwendigere Verfahren eingesetzt werden, die z. B. einen größeren problemspezifischen Merkmalsatz verwenden (siehe Abschnitt 8.1).

11.4. Diskussion

Es wurde hier ein inkrementelles Update der Kovarianzmatrix vorgeschlagen für den Fall, dass ein linearer Klassifikator wiederholt mittels LDA trainiert wird und sich dabei die Merkmalsberechnung jeweils nur für wenige Komponenten ändert. Das neue Update reduziert dann die Komplexität des iterierten Trainings von kubisch, $O(n^3)$, auf $\Theta(Nn^2)$ wenn N Komponenten geändert und n Dimensionen betrachtet werden. Für $N = 1$ ist dies die untere Schranke.

Die vorgeschlagene Gleichung stellt eine Rang-2-Korrektur der Kovarianzmatrix dar. Ähnliche Rank-1-Korrekturen wurden bereits im Zusammenhang mit dem populären Kalman-Filter (Grewal und Andrews, 1993) sowie Evolutionsstrategien (Suttorp u. a., 2009) eingeführt.

Es wurden Experimente sowohl auf einem künstlichen Merkmals-Selektions-Problem als auch auf einem realen Merkmals-Optimierungs-Problem durchgeführt. In beiden Fällen konnte gezeigt werden, dass die neue Update-Gleichung signifikante Einsparungen an Rechenzeit beim Training erlaubt, während die Rechnungen numerisch stabil sind.

Zusammenfassend wird es dadurch möglich, größere Merkmalsätze genauer zu untersuchen. Wenn Meta-Heuristiken eingesetzt werden, bedeutet das konkret, dass mehr Iterationen und / oder größere Populationen betrachtet werden können. Praktisch alle Meta-Heuristiken, die LDA nutzen, profitieren damit von der neuen Update-Gleichung.

12. Bilder von Google Street View als Datenquelle

In diesem Kapitel wird ein neues Vorgehen vorgestellt, um große Mengen von Bilddaten für das Trainieren, Testen und Vergleichen von videobasierten Algorithmen zu gewinnen¹. Es werden dabei Daten von *Google Street View* genutzt, die normalerweise nur über eine Web-Anwendung zugreifbar sind.

Im ersten Abschnitt dieses Kapitels werden Problemstellung und Motivation genauer dargestellt. In Abschnitt 12.2 wird der Zugriff auf die Bilddaten und die technische Umsetzung des entwickelten *Frameworks* beschrieben. Die durchgeführten Experimente werden in Abschnitt 12.3 dokumentiert und die Ergebnisse in Abschnitt 12.4 diskutiert.

12.1. Problemstellung

Während der Entwicklung videobasierter FAS besteht eine nicht unwesentliche Herausforderung darin, ausreichend viele Daten zum Trainieren und Testen zu gewinnen. Das Aufzeichnen und Nachbereiten von Videodaten ist aufwendig, nicht selten aufwendiger als die Algorithmen-Entwicklung selbst. Das gilt insbesondere, wenn Verfahren zur Objekterkennung für unterschiedliche Länder entwickelt und getestet werden sollen.

Anstatt neue Sequenzen aufzuzeichnen, gibt es teilweise die Möglichkeit, auf öffentlich verfügbare Datensätze zurückzugreifen. Leider gibt es hier bisher nur eine geringe Zahl, z. B. für Fußgängererkennung (Dollár u. a., 2009,ENZweiler u. a., 2010, Keller u. a., 2011), Ampelerkennung (de Charette und Nashashibi, 2009) und Verkehrszeichenklassifikation (Stallkamp u. a., 2011).

Für das Beispiel Verkehrszeichenerkennung steht ein angemessener Datensatz zum Vergleich von Algorithmen nur für Deutschland zur Verfügung. Eine Schwäche aller öffentlichen Datensätze besteht darin, dass sie typischerweise mit Blick auf eine bestimmte Fragestellung zusammengestellt wurden und es daher häufig nicht möglich ist, sie für andere (selbst ähnliche) Zwecke zu nutzen.

¹Teile der in diesem Abschnitt vorgestellten Arbeiten und Ergebnisse wurden bereits in Salmen u. a. (2012) veröffentlicht.

In der Praxis werden teilweise auch synthetische Sequenzen verwendet. Solche Daten sind besonders nützlich, wenn Algorithmen zum Schätzen von Entfernungen in der Welt (z. B. Stereo-Verarbeitung) oder zum Schätzen von optischem Fluss untersucht werden sollen. In diesen Fällen ist es praktisch unmöglich, für reale Videosequenzen *ground truth*-Daten zu erzeugen (Vaudrey u. a., 2008). Es gibt einige öffentlich verfügbare synthetische Sequenzen, z. B. von van der Mark und Gavrilu (2006) und Vaudrey u. a. (2008). Eine Simulation erlaubt es, beliebig lange Strecken zu betrachten ohne dass dabei signifikante Kosten entstehen. Trotzdem kann man sich bei der Entwicklung von bildbasierten Algorithmen nie ausschließlich auf die Ergebnisse verlassen, die auf simulierten Daten erzielt werden – die Qualität der entsprechenden Simulationen ist noch zu weit von der (vollständigen) Abbildung realer Daten entfernt.



Abbildung 12.1.: Verfügbarkeit von *Google Street View* Bildern, Stand September 2012. Quelle <http://support.google.com>.

Hier wird vorgeschlagen auf Bilder von *Google Street View* zurückzugreifen. Die Bilder aus der entsprechenden Datenbank sind über die Web-Anwendung² öffentlich verfügbar. *Google Street View* stellt wohl die größte Sammlung von Bildern dar, die Straßenverkehrsszenen zeigen. Da die Bilder von einem fahrenden Auto aufgenommen wurden, entsprechen sie auch in ihrer Qualität genau dem, was für die Entwicklung von video-basierten FAS benötigt wird.

Die Bilder können insbesondere in frühen Phasen der Algorithmen-Entwicklung von großer Bedeutung sein, da sie es erlauben, ohne großen Aufwand auf riesige Datenmengen zurückzugreifen, wobei große Teile der Welt abgedeckt sind (vgl. Abbildung 12.1).

Da die *Google*-Bilder nur über die entsprechende Web-Anwendung zugänglich sind, ist

²<http://maps.google.de>

es zunächst praktisch nicht möglich, sie tatsächlich in der Algorithmen-Entwicklung zu verwenden. Das würde erfordern, einzelne Bilder manuell aus dem Browser zu kopieren und diesen entsprechend zu bedienen, was wiederum einen unverhältnismäßig großen Aufwand bedeuten würde. Außerdem sind relevante Objekte in den Bildern natürlich nicht markiert (*gelabelt*, *annotiert*). Aufgrund der großen Datenmenge ist es nicht angemessen, dies manuell zu machen. Vielmehr sollten halb-überwachte (*semi-supervised*), also halb-automatische Ansätze zur Objekterkennung berücksichtigt werden.

12.2. Zugriff auf Bilddaten

Auf die Bilddaten von *Google Street View* kann normalerweise ausschließlich über eine Web-Anwendung (siehe Abbildung 12.2) zugegriffen werden. Diese wiederum kommuniziert über öffentlich nicht dokumentierte Schnittstellen mit einem Server, um alle notwendigen Daten anzufordern. Um die *Street View*-Bilder für Algorithmen-Entwicklung, also außerhalb der Web-Oberfläche, nutzbar zu machen, können diese undokumentierten Schnittstellen genutzt werden.

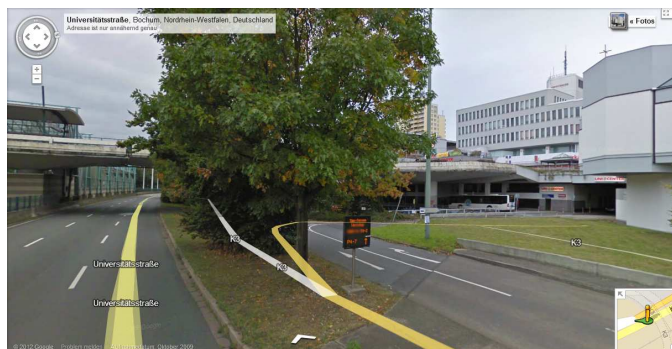


Abbildung 12.2.: Screenshot der Browser-Schnittstelle von *Google Street View*.

In einem Blog-Beitrag von Mai 2010 hat Jamie Thompson diese Schnittstellen beschrieben³. Er konnte, hauptsächlich durch geschicktes Ausprobieren, zahlreiche Funktionen nachvollziehen. Welche Möglichkeiten es darüber hinaus noch gibt, ist nicht bekannt. Unabhängig von diesem Blog-Beitrag gab es schon Vorschläge, Bilder von *Google Street View* für Forschungszwecke zu nutzen (Micusik und Kosecka, 2009, Zamir und Shah, 2010).

³<http://jamiethompson.co.uk/web/2010/05/15/google-streetview-static-api>

Es ist nicht ganz klar, inwieweit der Zugriff über die offiziell nicht dokumentierten Schnittstellen mit den Nutzungsbestimmungen⁴ von Google zu vereinbaren ist (zusätzliche Informationen finden sich in einem FAQ⁵). Zunächst ist aber anzunehmen, dass ein reiner Lesezugriff wohl zulässig sein muss, solange keine (veränderten) Kopien der Daten weitergegeben oder veröffentlicht werden. Insbesondere garantiert Google uneingeschränkten und kostenlosen Zugriff für „*non-profits and applications deemed in the public interest*“, also etwa Forschungstätigkeit an Universitäten. Aus den eben dargestellten Gründen werden aber vorerst keine Datensätze veröffentlicht, die aus Google-Daten entstammen (vgl. auch die Diskussion in Abschnitt 12.4).

Der Zugriff auf die Google-Daten kann konkret über einfache HTTP-Anfragen an den Server `cbk0.google.com/cbk` erfolgen. Es sind bislang vier verschiedene Arten solcher Anfragen bekannt, die im Folgenden der Vollständigkeit halber alle dokumentiert werden, obwohl die später vorgestellte Anwendung lediglich zwei davon benutzt.



Abbildung 12.3.: Beispiel eines Vorschau-Bildes (*thumbnail*) das über die statische API zugreifbar ist.

Thumbnail für GPS-Koordinaten

`http://cbk0.google.com/cbk?output=thumbnail&w=[SX]&h=[SY]&ll=[LAT,LNG]`

Diese Anfrage erwartet GPS-Koordinaten (Latitude LAT und Longitude LNG). Wenn nah an der spezifizierten Position ein Panoramabild verfügbar ist, wird eine stark verkleinerte Version (*Thumbnail*) dieses Bilds in *JPEG*-Format zurückgegeben. Die resultierende Bildgröße kann mithilfe der Parameter `SX` \times `SY` beeinflusst werden, die maximale Größe ist auf `300` \times `128` Pixel beschränkt. Das Vorschaubild zeigt das volle 360° Panorama, Abbildung 12.3 zeigt ein Beispiel.

⁴http://www.google.com/intl/en_ALL/help/terms_maps.html

⁵<http://code.google.com/intl/en/apis/maps/faq.html>

XML-Daten für GPS-Koordinaten

[http://cbk0.google.com/cbk?output=xml&ll=\[LAT,LNG\]](http://cbk0.google.com/cbk?output=xml&ll=[LAT,LNG])

Mithilfe dieser Anfrage können zusätzliche Informationen zu einem Panorama-Bild erfragt werden. Die Rückgabe erfolgt im *XML*-Format und erhält bspw. Ort (Adresse) und Zeitpunkt der Aufnahme. Eine besondere Rolle spielt die sog. Panorama-ID `pano_id`. Diese IDs sind Zeichenketten, die Knoten eindeutig bezeichnen. Das XML enthält nicht nur die ID des aktuellen Panoramas, sondern auch IDs angrenzender Panoramabilder (Einträge `link`), also typischerweise auf derselben Straße oder kreuzende Straßen.

Schließlich geht aus den XML-Daten auch der aktuelle Straßentyp (z. B. Autobahn, Landstraße, Innenstadt, etc.) hervor (Eintrag `road_argb`). Damit wird es möglich, nur eine Teilmenge der Bilder zu berücksichtigen, wenn eine Anwendungen das erfordert.

Tabelle 12.1.: Zoom-Stufen, die in der *Street View* API verfügbar sind, zugehörige Anzahl Kacheln und resultierend Größe der Panoramabilder.

Zoom-Stufe	Anzahl Kacheln	Resultierende Bildgröße
0	1 × 1	512 × 256
1	2 × 1	1.024 × 512
2	4 × 2	2.048 × 1.024
3	8 × 4	4.096 × 2.048
4	16 × 7	8.192 × 3.584
5	28 × 13	14.336 × 6.656



Abbildung 12.4.: Bild-Kacheln für Zoom-Stufen 3, 4 und 5 (von links nach rechts) aus dem Panoramabild aus Abb. 12.3. Der Bildausschnitt für die höchste Zoom-Stufe zeigt deutlich die hohe Auflösung der verfügbaren Bilder.

Bild-Kacheln für ID

`http://cbk0.google.com/cbk?output=tile&panoid=[ID]&zoom=[Z]&x=[X]&y=[Y]`

Unter Angabe der Panorama-ID, die man nach der entsprechenden Anfrage der XML-Daten erhält (s. o.), kann über diesen Aufruf ein komplettes Panoramabild angefragt werden. Im Unterschied zu der oben beschriebenen Anfrage eines *Thumbnail* über GPS-Koordinaten können hier deutlich größere Bilder betrachtet werden: Die hochaufgelösten 360°-Panoramabilder liegen unterteilt in einzelne Kacheln vor, die jeweils eine Größe von 512×512 Pixeln haben. Alle Kacheln eines Bildes müssen unabhängig voneinander unter Angabe ihrer jeweiligen Position X bzw. Y angefragt werden.

Die Bilddaten liegen in sechs verschiedenen Größen vor. Die gewünschte Größenstufe kann innerhalb der hier beschriebenen Anfrage über den Parameter $Z \in [0, \dots, 5]$ bestimmt werden. Da die Bild-Kacheln eine feste Größe haben, ergeben sich für verschiedene Zoom-Stufen unterschiedliche Größen des Gesamtbildes und eine unterschiedliche Anzahl an Kacheln. Tabelle 12.1 gibt einen Überblick über die entsprechenden Werte, Abbildung Figure 12.4 stellt einige Kacheln aus dem Bild in Abbildung 12.3 für verschiedene Zoom-Stufen dar. In der größten Stufe ergibt sich ein Panoramabild mit gut 90 *Megapixel* – das ist bemerkenswert und eröffnet zahlreiche Möglichkeiten für die Nutzung der Daten.

Thumbnail für ID

`http://cbk0.google.com/cbk?output=thumbnail&w=[SX]&h=[SY]&panoid=[ID]`

Dieser Aufruf hat dieselbe Ausgabe wie der hier zuerst genannte, erwartet aber im Unterschied dazu als Parameter eine Panorama-ID statt GPS-Koordinaten.

Basierend auf den oben vorgestellten Schnittstellen wurde eine Applikation entwickelt, die „virtuelles Fahren“ in *Street View* realisiert. Abbildung 12.5 zeigt die Benutzeroberfläche der Anwendung. Die Darstellung der Karte, die beliebig eingestellt werden kann, basiert auf der frei verfügbaren Software *Marble*⁶. Über die Oberfläche lassen sich ein Startpunkt und ein rechteckiger Suchbereich einstellen. Wenn auf Bilddaten zugegriffen wird, kann hier auch der bisherige Pfad eingezeichnet werden. Die Kartendaten stammen dabei aus dem öffentlichen *OpenStreetMap*-Projekt. Diese Daten könnten (in Bezug auf Koordinaten) prinzipiell von den Google-Daten abweichen. Das würde zu Fehlern insbesondere bei der Visualisierung führen. Solche Probleme konnten in Experimenten (siehe Abschnitt 12.3) jedoch nicht beobachtet werden.

Der Pseudo-Code für das virtuelle Fahren ist in Algorithmus 2 gegeben. Es wird, vom Startpunkt ausgehend, eine Tiefensuche durchgeführt. Das heißt, in jedem Schritt wird

⁶<http://edu.kde.org/marble>

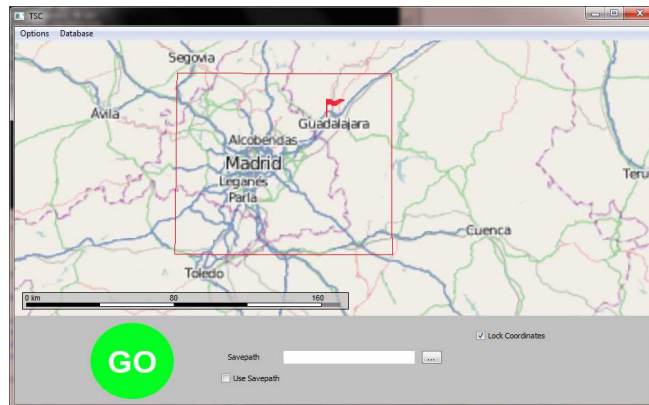


Abbildung 12.5.: Benutzer-Oberfläche der entwickelten Applikation zum Zugriff auf *StreetView*-Daten.

der ersten möglichen Abzweigung (*link*) gefolgt und ggf. weitere werden für die spätere Verarbeitung in der Liste *storedIds* gespeichert. Der Algorithmus muss alle bereits besuchten Knoten verwalten, um doppelte Wege zu vermeiden. Dafür wird jede verarbeitete ID in der Liste *visitedIds* gespeichert. Wenn ein Knoten zum ersten Mal erreicht wird und innerhalb des Suchbereichs liegt (Zeile 11), werden die zugehörigen XML-Daten angefragt, aus denen insbesondere alle Abzweigungen (s. o.) ausgelesen werden können. Optional können Bilddaten angefragt und verarbeitet werden (Zeilen 16 bzw. 17).

Der Algorithmus terminiert (Zeile 5), wenn alle *erreichbaren* Knoten innerhalb der Suchbereichs verarbeitet wurden. Das bedeutet allerdings nicht, dass *alle* Knoten innerhalb des Bereichs besucht wurden. Ein Knoten ist für den oben angegebenen Algorithmus dann erreichbar, wenn es vom Startpunkt aus eine Verbindung gibt, die komplett innerhalb des Suchbereichs liegt.

12.3. Experimente

Es wurden zwei verschiedene Arten von Experimenten durchgeführt. In den ersten Experimenten soll das *Framework* selbst untersucht werden. In den zweiten Experimenten wird dann anhand eines Anwendungsbeispiels der Zugriff der Google-Daten mit einem Bildverarbeitungsmodul kombiniert.

Algorithmus 2: Pseudo-Code für virtuelles Fahren in *Google Street View***Eingabe :**

- GPS-Koordinaten des Startpunkts (x, y)
- Rechteckiger Suchbereich r

```

1 aktuellePanoId ← holePanoIdAnGpsKoordinaten( $x, y$ );
2 Repeat
3   if enthaelt(besuchteIds, aktuellePanoId) then
4     if istLeer(gespeicherteIds) then
5        $\lfloor$  STOP;
6     else
7        $\lfloor$  aktuellePanoId ← entnimm(gespeicherteIds);
8   else
9     if istInnerhalb(aktuellePanoId,  $r$ ) then
10      haengeAn(visitedIds, currentId);
11      xmlData ← holeXmlDaten(aktuellePanoId);
12      links ← holeLinksAusXml(xmlData);
13      currentId ← entnimm(links);
14      for  $i \leftarrow 1$  to anzahl(links) do
15         $\lfloor$  haengeAn(gespeicherteIds, links[ $i$ ]);
16      bild ← frageBildAn(aktuellePanoId);
17      verarbeite(bild);

```

12.3.1. Aufbau

Um zunächst die Leistungsfähigkeit des neuen *Frameworks* zu bestimmen, werden verschiedene Größen berücksichtigt: Welche (virtuelle) Strecke kann pro Stunde betrachtet werden? Wie viele Panoramabilder sind währenddessen verfügbar? Wie verhält sich der vorgeschlagene Algorithmus im praktischen Test bzgl. Laufzeit und Speicher- auslastung? Die ersten Experimente sollen außerdem Aufschluss darüber geben, wie Bilddaten bei *Google Street View* vorliegen, also wie hoch die Abdeckung und Dichte in verschiedenen Gebieten ist und wieweit sie z. B. von verschiedenen Straßentypen abhängt.

Um die oben genannten Fragen zu beantworten, wurde in den ersten Experimenten zunächst nur auf *XML*-Daten zugegriffen und keine Bilder verarbeitet (d. h., die Zeilen 16 und 17 im Pseudo-Code in Algorithmus 2 werden nicht ausgeführt). Für die Experi-



Abbildung 12.6.: Regionen die für die initiale Evaluation ausgewählt wurden.

mente wurden verschiedene Städte in unterschiedlichen Kontinenten betrachtet. Die ausgewählten Gebiete sind in Abbildung 12.6 dargestellt. Alle Experimente wurden auf einem Desktop-PC (Intel Xeon W3520 CPU, 6 GB RAM) mit einer Breitband-Internetanbindung (Download ca. 5.000 kb/s) durchgeführt.



Abbildung 12.7.: Beispiel-Bilder aus dem GTSRB-Datensatz von Stallkamp u. a. (2011).

Von Stallkamp u. a. (2011) wurde ein Benchmark-Datensatz für die Klassifikation deutscher Verkehrszeichen vorgestellt. Der Datensatz enthält über 50.000 von mehr als 40 unterschiedlichen Klassen. Die Einzelbilder stammen von über 1.700 unterschiedlichen Schildern („Instanzen“). Abbildung 12.7 zeigt einige Beispiele.

Den Benchmark-Datensatz zu erstellen erforderte mehrere Monate Arbeit (Aufnahmen von Sequenzen, manuelles Labeln, Aufbereiten der Daten). Er war der erste öffentlich verfügbare Datensatz für Verkehrszeichen-Klassifikation in dieser Größe. Es kann nicht erwartet werden, dass ähnlich umfangreiche Datensätze in naher Zukunft auch für andere Länder verfügbar sein werden. Es wäre jedoch sehr interessant zu sehen, wie sich die Ergebnisse auf andere Länder übertragen lassen, in denen Verkehrsschilder andere Formen, Farben, usw. haben.

In einem Experiment soll daher untersucht werden, ob sich das hier präsentierte Verfahren eignet, ähnliche Datensätze mit weniger Aufwand zu erzeugen. Um Verkehrszeichen in Videobildern zu finden, wurde ein Viola-Jones-Detektor (vgl. Abschnitt 2.3) basierend auf Haar-Merkmalen (vgl. Abschnitt 1.2) verwendet.

Der Viola-Jones-Detektor basiert auf *AdaBoost*, einem Lernverfahren. Damit wird ein

halb-überwachtes (*semi-supervised*) Training möglich: Zunächst muss ein initialer Detektor auf einer kleinen Menge von Beispielen trainiert werden. Danach kann er auf einige Bilder von *Street View* angewendet werden. Die vom Detektor extrahierten Bildausschnitte können dann manuell sortiert, der ursprünglichen Trainingsmenge hinzugefügt und das Training erneut durchgeführt werden. Dieses Vorgehen kann prinzipiell beliebig oft wiederholt werden.

Um den ersten Detektor zu trainieren sind einige negative und einige positive Beispiele erforderlich. Die negativen Beispiele können leicht gewonnen werden (z. B. Kamerabilder aus *Street View*, die keine Verkehrszeichen erhalten). Die positiven Beispiele für das initiale Training des Detektors müssen ggf. etwas aufwendiger gewonnen werden. Es ist naheliegend, manuell markierte Bildausschnitte aus *Street View* zu verwenden. Alternativ können auch bereits vorhandene Daten zum Training verwendet werden. Für die Experimente hier wurden Bilder aus GTSRB verwendet.

Für das Sammeln von Verkehrszeichen wurde die Zoom-Stufe 3 verwendet (vgl. Abschnitt 12.2). Aufgrund der recht gut bekannten Position (insbesondere Höhe) von Schildern in der Welt ist es ausreichend, in vertikaler Richtung nur den mittleren Teil (eine von drei Kacheln) zu betrachten. Damit ergibt sich eine relevante Bildauflösung von 3.072×512 Pixel.

12.3.2. Ergebnisse

Tabelle 12.2 stellt die bei den ersten Tests erzielten Ergebnisse dar. Vergleicht man die beim „virtuellen Fahren“ erreichten Zahlen mit tatsächlichem Fahren, hat die hier vorgeschlagene Methode deutliche Vorteile. So konnte in den ersten Experimenten eine durchschnittliche „virtuelle Geschwindigkeit“ von ca. 300 km/h erreicht werden. Das heißt, dass pro Stunde Rechenzeit 300 km Straße betrachtet wurden. Als durchschnittlicher Abstand zwischen jeweils zwei Panoramabildern ergaben sich ca. 11,5 m, wobei dieser Wert für verschiedene Regionen und / oder Straßentypen nicht signifikant schwankt.

Tabelle 12.2.: Ergebnisse der Evaluation für die Regionen aus Abb. 12.6.

Region	Distanz [km]	Panorama-Bilder	Verarbeitungszeit [h:min]
San Fransisco, USA	3023,8	268.127	10:03
Penghu Islands, Taiwan	514,8	44.736	1:41
Port Elizabeth, Süd-Afrika	2105,1	175.369	5:58
Belo Horizonte, Brasilien	1426,3	128.459	4:33
Alcalá de Henares, Spanien	409,9	32.645	1:01
Goldküste, Australien	2509,8	219.558	8:34

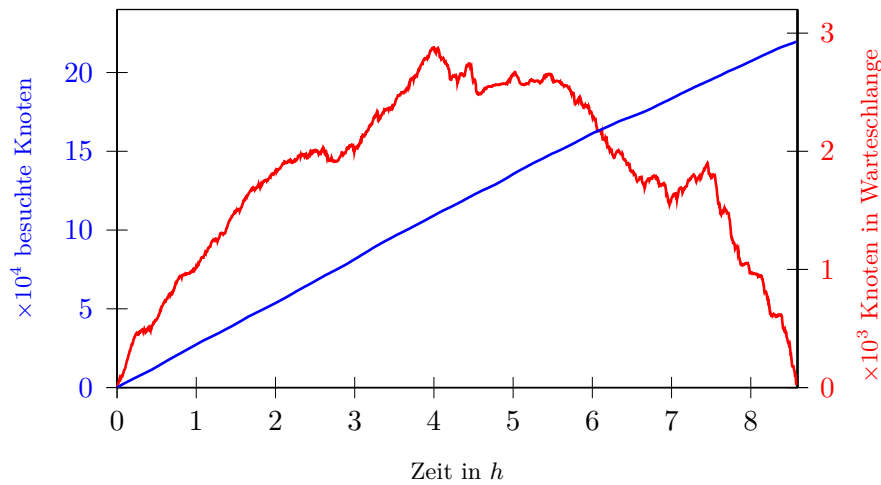


Abbildung 12.8.: Detaillierte Analyse des vorgeschlagenen Such-Algorithmus für das Beispiel Goldküste, Australien: Anzahl besuchter Knoten (blau) und Anzahl Knoten in der Warteschlange (rot) über die Zeit.

Beispielhaft für eine „virtuelle Fahrt“ stellt Abbildung 12.8 detaillierte Ergebnisse für die Region Goldküste/Australien dar. Die Anzahl der besuchten Knoten wächst über die Zeit fast linear. Ein Grund dafür, dass sich die Geschwindigkeit mit zunehmender Verarbeitungszeit minimal verringert ist wahrscheinlich, dass eine ständig wachsende Zahl von bereits besuchten Knoten verwaltet werden muss (vgl. Algorithmus 2, Zeile 3). Die Anzahl der zur späteren Verarbeitung gespeicherten IDs erhöhte sich im Experiment zunächst recht konstant (bis maximal 2.900) und wurde danach ebenfalls recht konstant abgebaut.

Für die ersten Tests mussten lediglich XML-Daten betrachtet werden. Sobald Bilddaten verarbeitet werden, sinkt die Verarbeitungsgeschwindigkeit in Abhängigkeit vom verwendeten Algorithmus. Insbesondere spielt beim Abrufen von Bildern höherer Zoom-Stufen die vorhandene Internet-Anbindung eine wichtige Rolle. Im Vergleich zu den ersten Experimenten reduzierte sich die „virtuelle Geschwindigkeit“ deutlich auf ca. 40 km/h.

Die beiden Schritte „automatisches Sammeln“ und „manuelle Kontrolle und Nachtrainieren“ wurden in den Experimenten fünf Mal wiederholt. Mit diesem halb-automatischen Vorgehen konnten innerhalb einer Woche mehr als 10,000 Bildausschnitte gesammelt werden, die deutsche Verkehrszeichen zeigen. Da jedes konkrete Verkehrsschild in der Welt (jede Instanz) in *Street View* typischerweise ein- oder zweimal zu finden ist, bedeutet das, dass der so erhaltene Datensatz bereits mehr Instanzen um-

fasst als der ursprüngliche GTSRB-Datensatz.

Schließlich kann die Leistung des iterativ trainierten Viola-Jones-Detektors beurteilt werden. Dafür werden *precision* und *recall* anhand 2.000 manuell gelabelten Panorama-Bildern bestimmt. Es ergab sich eine *precision* von 49,3% und ein *recall* von 81,6%. Das heißt, dass etwa die Hälfte der gefundenen Bildausschnitt Falsch-Alarme darstellten, während ein großer Anteil der vorhandenen Verkehrsschilder korrekt gefunden wurde.

12.4. Diskussion

Die von *Google Street View* verfügbaren Bilder sind offensichtlich nicht gleichermaßen gut geeignet für alle kamerabasierten Algorithmen, die für FAS relevant sind. Da zwischen zwei aufeinanderfolgenden Bildern verhältnismäßig viel räumlicher (ca. 11,5 m) und zeitlicher Abstand (teilweise mehrere Wochen oder Monate) liegt, ist z. B. das Trainieren oder Testen von Algorithmen zum Schätzen von optischem Fluss nicht sinnvoll. Die Bilder eignen sich jedoch sehr gut in der Entwicklung von Verfahren die Einzelbilder verarbeiten, also insbesondere Objekterkennung.

Es muss beachtet werden, dass die Aufnahmen von Google nur tagsüber und nur bei gutem Wetter gemacht wurden. Damit können mithilfe der Bilddaten keine Aussagen darüber getroffen werden, wie gut Algorithmen in anderen relevanten Situationen (z. B. nachts oder bei schlechteren äußeren Bedingungen) funktionierten. Dieser Nachteil gilt jedoch genauso für praktisch alle aktuell öffentlich verfügbaren Datensätze.

Wenn Daten von *Google Street View* auf eine Art und Weise genutzt werden wie hier z. B. in den Experimenten zur Verkehrszeichen-Erkennung beschrieben, wenn also ein maschinelles Lernverfahren halb-überwacht eingesetzt wird (vgl. Abschnitt 12.3), können sich besondere Probleme ergeben, auf die im Folgenden kurz eingegangen werden soll. Eine so große Datenmenge wie die von *Google Street View* kann nur automatisch verarbeitet werden. Es ist höchstens möglich, die erhaltenen Ergebnisse manuell nachzubearbeiten, also z. B. extrahierte Bildausschnitte zu sortieren. Durch die manuelle Korrektur kann die automatische Verarbeitung wiederum profitieren (wenn nachtrainiert wird) und so kontinuierlich verbessert werden. Jedoch können im Rahmen der manuellen Kontrolle nur Bildausschnitte betrachtet werden, die der ursprüngliche Algorithmus liefert. Wenn Objekte einer bestimmten Objektkategorie gesammelt werden sollen, gibt es für jeden Bildausschnitt, der vom Algorithmus betrachtet wird, vier Möglichkeiten:

- **True Positive:** Der Algorithmus liefert ein korrektes Ergebnis, das manuell bestätigt werden kann

- **False Positive:** Der Algorithmus liefert ein falsches Ergebnis, das manuell aussortiert wird. Es kann dann als zusätzliches Gegenbeispiel im folgenden Training berücksichtigt werden
- **True Negative:** Der Algorithmus erkennt korrekterweise, dass ein Bildausschnitt nicht relevant ist. Der Nutzer sieht dieses Ergebnis nicht, müsste aber auch nicht eingreifen. Auf Ergebnisse dieser Kategorie entfällt der größte Anteil
- **False Negative:** Der Algorithmus entscheidet fälschlicherweise, dass ein Bildausschnitt nicht relevant ist. Der Nutzer sieht dieses Ergebnis nicht und kann dementsprechend nicht korrigierend eingreifen

Der letzte der vier Fälle ist dabei der einzige, der problematisch ist. Ergebnisse, die der Algorithmus nicht findet, können zunächst nicht korrigiert werden, werden also systematisch vom endgültigen Datensatz ausgeschlossen. Die einzige Möglichkeit, die Auswirkungen dieses Problems sinnvoll zu minimieren, besteht darin, den semi-automatischen Prozess so zu erweitern, dass „ausreichend“ viele relevante Beispiele manuell aus Bildern extrahiert und im Training berücksichtigt werden.

Im hier betrachteten Beispiel im Kontext von Verkehrszeichenerkennung könnte für das Training des ersten Detektors auf eine große Menge an manuell annotierten Daten zurückgegriffen werden. Konkret profitiert man in diesem Beispiel von den Daten des GTSRB-Benchmark. Für andere Anwendungen, wo solche Datensätze noch nicht zur Verfügung stehen, könnten bspw. zufällige Bilder aus *Street View* gewählt und manuell annotiert werden.

Zusammenfassend stellen die Daten von *Google Street View* eine wertvolle Datenquelle dar, insbesondere mit Blick auf Forschung und Vorentwicklung. Die Daten ermöglichen es, vorhandene Algorithmen schnell auf großen Mengen zu trainieren und zu testen. Dabei können Bilddaten aus vielen unterschiedlichen Ländern berücksichtigt werden, was sonst mit erheblichem Aufwand verbunden wäre.

13. Zusammenfassung, Bewertung und Ausblick

Aktive Sicherheitssysteme in Kraftfahrzeugen spielen eine wichtige Rolle beim Schutz der Fahrzeuginsassen, insbesondere aber auch beim Schutz schwächerer Verkehrsteilnehmer. Kamerabasierte Assistenzsysteme bieten besonders großes Potential für zukünftige Entwicklungen, sie sind außerdem notwendig, um langfristig autonom fahrende Serienfahrzeuge möglich zu machen.

Verschiedene kamerabasierte Module gleichzeitig in ein Serienfahrzeug zu integrieren, stellt eine große technische Herausforderung dar. Problematisch ist insbesondere, dass populäre Algorithmen für wichtige Anwendungen (wie Objekterkennung, Stereo-Verarbeitung oder Schätzen von optischem Fluss) auf unterschiedliche Bildmerkmale zurückgreifen. Das resultierende Gesamtsystem wird dadurch komplex und dementsprechend teuer in der praktischen Umsetzung. Dieser Umstand trägt mit dazu bei, die Verbreitung videobasierter FAS in Serienfahrzeugen zu verlangsamen.

In der vorliegenden Arbeit wurde eine neue Architektur vorgeschlagen, in der alle relevanten Module auf dieselben Bildmerkmale zurückgreifen. Die universelle Vorverarbeitung nutzt Haar-Merkmale, die sich, insbesondere auch in eingebetteter Hardware, sehr effizient berechnen lassen. Es konnte gezeigt werden, dass die Leistungsfähigkeit der betrachteten Module in der neu vorgeschlagenen Architektur mindestens vergleichbar mit den vorherigen Alternativen ist. Dafür wurden u. a. Ansätze für das Erkennen unterschiedlicher Objekte anhand der Beispiele Fußgänger- und Verkehrszeichendetektion untersucht.

Für die bildbasierte Erkennung von Fußgängern wurde vorgeschlagen, einen linearen Klassifikator einzusetzen und geeignete Haar-Merkmale durch evolutionäre Optimierung automatisch zu erzeugen. Mit diesem Vorgehen konnten auf einem öffentlich verfügbaren Benchmark-Datensatz sehr gute Ergebnisse erzielt werden: Das neue Verfahren erreichte eine Erkennungsleistung vergleichbar mit der einer Support-Vektor-Maschine, die auf einen größeren Merkmalsatz zurückgreift, ist aber mehrere Größenordnungen schneller.

Für das Erkennen von Verkehrszeichen wurde ein neuer umfangreicher Benchmark-Datensatz vorgestellt. Er diene als Grundlage für den systematischen Vergleich mehrerer populärer Verfahren. Ein Viola-Jones-Detektor basierend auf Haar-Merkmalen

stellte sich dabei als bester Ansatz heraus. Das ist insbesondere deshalb bemerkenswert, da das zugrunde liegende Verfahren ein allgemeines Lernverfahren ist und außerdem im hier betrachteten Fall keine Farbinformation berücksichtigt. Trotzdem ist die Erkennungsleistung höher als die von typischen problemspezifischen Ansätzen.

Zusätzlich zu den Verfahren zur Objekterkennung wurde auch ein Modul zur Stereo-Verarbeitung innerhalb der neuen Architektur entwickelt. Dafür wurden bei der Kostenberechnung Haar-Merkmale verschiedener Typen verwendet, die an jedem Pixel im Kamerabild ausgewertet werden. In den Experimenten wurden drei populäre echtzeitfähige Algorithmen betrachtet. Alle Parameter wurden mit der Kovarianzmatrix-Adaption-Evolutionsstrategie optimiert, dadurch wurde ein objektiver Vergleich gewährleistet. Mithilfe der Kostenberechnung basierend auf Haar-Merkmalen konnten die Ergebnisse aller untersuchten Algorithmen verbessert werden: Im Vergleich zum ursprünglichen Vorgehen konnte die Anzahl fehlerhafter Pixel um bis zu 20 % reduziert werden.

Schließlich wurde auch ein Modul zum Schätzen von optischem Fluss vorgestellt, das in der neuen Architektur eingesetzt werden kann. Dafür wurde der *PowerFlow*-Algorithmus, der bereits in Serienfahrzeugen eingesetzt wird, entsprechend angepasst. In umfangreichen Experimenten auf synthetischen Benchmark-Sequenzen wurden Parameter verschiedener Verfahren mittels Mehrziel-Optimierung bestimmt. Die neue angepasste Methode basierend auf Haar-Merkmalen erreichte dabei deutlich bessere Ergebnisse als die Original-Methode, z. B. erlaubte sie im Durchschnitt bis zur 5-fachen Anzahl Bewegungsvektoren zu schätzen.

Die im Rahmen der vorliegenden Arbeit erzielten Ergebnisse zeigen, dass eine Systemarchitektur mit einer universellen Vorverarbeitung basierend auf Haar-Merkmalen sehr vielversprechend ist. Im Vergleich zu einem naiven Ansatz können mehrere Module ersetzt werden, es ergibt sich ein einfacherer und effizienterer Aufbau. Relevante Algorithmen können angepasst werden, ohne dass ihre Leistung abnimmt. In vielen Fällen konnten sogar bessere Ergebnisse erzielt werden.

Wenn Algorithmen basierend auf Haar-Merkmalen verwendet werden, ist die Zahl der freien Parameter typischerweise sehr hoch, da ein geeigneter Merkmalsatz festgelegt werden muss und jedes einzelne Merkmal diverse Freiheitsgrade hat (z. B. Typ, Größe, Position im Bild, usw.). Im Rahmen der vorliegenden Arbeit wurde vorgeschlagen, alle Parameter, also insbesondere die Auswahl der Merkmale, durch automatische Optimierung zu bestimmen. In verschiedenen Experimenten konnte gezeigt werden, dass evolutionäre (Mehrziel-)Optimierung im betrachteten Kontext gute geeignet ist.

Das Bestimmen eines geeigneten Merkmalsatzes verursacht *offline* recht hohen Aufwand. Hier wurde eine neue Methode vorgeschlagen, um die Selektion und Optimierung geeigneter Merkmale effizienter durchführen zu können, wenn Heuristiken basie-

rend auf linearer Diskriminanzanalyse (LDA) verwendet werden. Für das wiederholte Training mittels LDA wurde eine neue Update-Gleichung vorgeschlagen. Die durchgeführten Experimente konnten zeigen, dass dieser neue Ansatz numerisch stabil ist und in der praktischen Anwendung zu deutlichen Einsparungen von Rechenzeit führt.

Beim Entwickeln von bildbasierten Applikationen wird typischerweise auf Lernverfahren zurückgegriffen. Für das Trainieren und Testen sind dann umfangreiche Beispieldaten erforderlich. Das ist normalerweise gleichbedeutend mit großem Aufwand, nicht selten entfällt in der Praxis mehr Arbeitszeit auf das Sammeln von Daten als auf die Algorithmen-Entwicklung selbst. In der vorliegenden Arbeit wurde hier eine neue Möglichkeit, frei verfügbare Bilddaten von *Google Street View* zu verwenden, vorgestellt und diskutiert.

Aus den hier präsentierten Ansätzen ergeben sich verschiedene Anknüpfungspunkte für zukünftige Forschungsarbeiten. Bisher wurden optimale Haar-Merkmale für verschiedene Applikationen jeweils unabhängig voneinander bestimmt. Es lassen sich wahrscheinlich zusätzliche Verbesserungen mit Blick auf die Gesamtlaufzeit erzielen, indem das Training der Module gemeinsam durchgeführt wird mit dem Ziel, einen gemeinsamen Merkmalsatz zu bestimmen.

Es ist anzunehmen, dass für unterschiedliche Szenarien (z. B. Autobahn gegenüber Innenstadt, Tag gegenüber Nacht, verschiedene Wetterverhältnisse, usw.) jeweils andere Merkmalsätze am besten geeignet sind. Wenn alle Module auf dieselbe Menge von Merkmalen zurückgreifen, lassen sich entsprechende Anpassungen *online* besonders leicht durchführen.

Schließlich verarbeiten alle in der neuen Architektur betrachteten Algorithmen Grauwertbilder. Farbinformation zu berücksichtigen kann für verschiedene Anwendungen nützlich sein. Das Konzept der Haar-Merkmale (und ihre effiziente Berechnung über das Integralbild) lassen sich leicht auf einzelne Kanäle geeigneter Farbräume übertragen.

A. Lebenslauf

Persönliche Daten

Name: Jan Salmen
Geburtsdatum / -ort: 20.10.1980 in Bochum

Ausbildung

2006 – 2013 Promotionsstudium, voraussichtlicher Abschluss Promotion
Elektrotechnik und Informationstechnik, Ruhr-Universität Bochum
2001 – 2006 Studium Kerninformatik, Abschluss Diplom (*sehr gut*)
Technische Universität Dortmund
1991 – 2000 Gymnasium, Abschluss Abitur (*gut*)
Hildegardisschule Bochum

Literaturverzeichnis

- [Alefs 2006] ALEFS, B.: Embedded Vehicle Detection by Boosting. In: *Proceedings of the IEEE Intelligent Transportation Systems Conference*, 2006, S. 536–541
- [Arrospide u. a. 2012] ARROSPIDE, J. ; SALGADO, L. ; MARINAS, J.: HOG-like gradient-based descriptor for visual vehicle detection. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2012, S. 223–228
- [Baker u. a. 2007] BAKER, A. ; SCHARSTEIN, D. ; LEWIS, J. ; ROTH, S. ; BLAK, M. ; SZELISKI, R.: A database and evaluation methodology for optical flow. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2007, S. 1–8
- [Barnes u. a. 2008] BARNES, N. ; ZELINSKY, A. ; FLETCHER, L.: Real-Time Speed Sign Detection Using the Radial Symmetry Detector. In: *IEEE Transactions on Intelligent Transportation Systems* 9 (2008), Nr. 2, S. 322–332
- [Bayer 1975] BAYER, B. E.: *U.S. Patent 3971065: Color imaging array*. Eastman Kodak Company, 1975
- [Belaroussi und Tarel 2009] BELAROUSSE, R. ; TAREL, J.-Ph.: Angle Vertex and Bisector Geometric Model for Triangular Road Sign Detection. In: *Proceedings of the IEEE Workshop on Applications of Computer Vision*, 2009, S. 577–583
- [Bertsekas und Tsitsiklis 1996] BERTSEKAS, D. P. ; TSITSIKLIS, J.: *Neuro-Dynamic Programming*. Athena Scientific, 1996
- [Beume u. a. 2007] BEUME, N. ; NAUJOKS, B. ; EMMERICH, M.: SMS-EMOA: Multiobjective selection based on dominated hypervolume. In: *European Journal of Operational Research* 181 (2007), Nr. 3, S. 1653–1669
- [Beyer 2007] BEYER, H.-G.: Evolution strategies. In: *Scholarpedia* 2 (2007), Nr. 8, S. 1965
- [Beyer und Schwefel 2002] BEYER, H.-G. ; SCHWEFEL, H.-P.: Evolution Strategies: A Comprehensive Introduction. In: *In Natural Computing* 1 (2002), Nr. 1, S. 3–52

- [Birchfield und Tomasi 1999] BIRCHFIELD, S. ; TOMASI, C.: Depth discontinuities by pixel-to-pixel stereo. In: *International Journal of Computer Vision* 35 (1999), S. 1073–1080
- [Bobick und Intille 1999] BOBICK, A. F. ; INTILLE, S. S.: Large occlusion stereo. In: *International Journal of Computer Vision* 33 (1999), Nr. 3, S. 181–200
- [Bouguet 2000] BOUGUET, J.-Y.: Pyramid Implementation of the Lucas Kanade Feature Tracker. In: *OpenCV Documentation*, Intel Corporation Microprocessor Research Labs, 2000
- [Broggi u. a. 2007] BROGGI, A. ; CERRI, P. ; MEDICI, P. ; PORTA, P. P. ; GHISIO, G.: Real Time Road Signs Recognition. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2007, S. 981–986
- [Brown u. a. 2003] BROWN, M. Z. ; BURSCHKA, D. ; HAGER, G. D.: Advances in computational stereo. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25 (2003), S. 993–1008
- [Brox u. a. 2009] BROX, T. ; BREGLER, C. ; MALIK, J.: Large displacement optical flow. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, S. 41–48
- [Bruhn und Weickert 2005] BRUHN, A. ; WEICKERT, J.: Lucas/Kanade Meets Horn/Schunck: Combining Local and Global Optic Flow Methods. In: *International Journal of Computer Vision* 61 (2005), Nr. 3, S. 211–231
- [Bruhn u. a. 2006] BRUHN, A. ; WEICKERT, J. ; KOHLBERGER, T. ; SCHNÖRR, C.: A Multigrid Platform for Real-Time Motion Computation with Discontinuity-Preserving Variational Methods. In: *International Journal of Computer Vision* 70 (2006), Nr. 3, S. 257–277
- [Buder 2012] BUDER, M.: Dense realtime stereo matching using a memory efficient Semi-Global-Matching variant based on FPGAs. In: *Proceedings of the SPIE Real-Time Image and Video Processing Conference*, 2012
- [Burger und Burge 2006] BURGER, W. ; BURGE, M. J.: *Digitale Bildverarbeitung*. 2. Auflage. Springer, 2006
- [Canny 1986] CANNY, J.: A Computational Approach To Edge Detection. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8 (1986), Nr. 6, S. 679–698
- [Cerný 1985] CERNÝ, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. In: *Journal of Optimization Theory and Applications* 45 (1985), S. 41–51

- [de Charette und Nashashibi 2009] CHARETTE, R. de ; NASHASHIBI, F.: Real time visual traffic lights recognition based on Spot Light Detection and adaptive traffic lights templates. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2009, S. 358–363
- [Crow 1984] CROW, F. C.: Summed-area tables for texture mapping. In: *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, 1984, S. 207–212
- [Dalal und Triggs 2005] DALAL, N. ; TRIGGS, B.: Histograms of Oriented Gradients for Human Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005, S. 886–893
- [Derpanis u. a. 2007] DERPANIS, K. G. ; LEUNG, E. T. H. ; SIZINTSEV, M.: Fast Scale-Space Feature Representations by Generalized Integral Images. In: *Proceedings of the IEEE International Conference on Image Processing*, 2007, S. 521–524
- [Dollár u. a. 2009] DOLLÁR, P. ; WOJEK, C. ; SCHIELE, B. ; PERONA, P.: Pedestrian detection: A benchmark. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, S. 304–311
- [Duda und Hart 1972] DUDA, R. O. ; HART, P. E.: Use of the Hough transformation to detect lines and curves in pictures. In: *Communications of the ACM* 15 (1972), Nr. 1, S. 11–15
- [Enzweiler u. a. 2010] ENZWEILER, M. ; EIGENSTETTER, A. ; SCHIELE, B. ; GAVRILA, D. M.: Multi-Cue Pedestrian Classification with Partial Occlusion Handling. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010, S. 990–997
- [Enzweiler und Gavrila 2009] ENZWEILER, M. ; GAVRILA, D. M.: Monocular Pedestrian Detection: Survey and Experiments. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (2009), Nr. 12, S. 2179–2195
- [Ernst und Hirschmüller 2008] ERNST, I. ; HIRSCHMÜLLER, H.: Mutual Information Based Semi-Global Stereo Matching on the GPU. In: *Proceedings of the International Symposium on Advances in Visual Computing*, 2008, S. 228–239
- [Freund und Schapire 1997] FREUND, Y. ; SCHAPIRE, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: *Journal of Computer and System Sciences* 55 (1997), Nr. 1, S. 119–139
- [Fukushima u. a. 1983] FUKUSHIMA, K. ; MIYAKE, S. ; ITO, T.: Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition. In: *IEEE Transactions on Systems, Man, and Cybernetics* 13 (1983), S. 826–834

- [Gehrig und Rabe 2010] GEHRIG, S.K. ; RABE, C.: Real-time Semi-Global Matching on the CPU. In: *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2010, S. 85–92
- [Geiger u. a. 1992] GEIGER, D. ; LADENDORF, B. ; YUILLE, A. L.: Occlusions and Binocular Stereo. In: *Proceedings of the European Conference on Computer Vision*, 1992, S. 425–433
- [Glover und Laguna 1997] GLOVER, F. ; LAGUNA, M.: *Tabu Search*. Kluwer Academic Publishers, 1997
- [Grewal und Andrews 1993] GREWAL, M. S. ; ANDREWS, A. P.: *Kalman Filtering: Theory and Practice*. Prentice-Hall, 1993
- [Gunturk u. a. 2005] GUNTURK, B. K. ; GLOTZBACH, J. ; ALTUNBASAK, Y. ; SCHAFER, R. W. ; MERSEREAU, R. M.: Demosaicking: Color filter array interpolation. In: *IEEE Signal Processing Magazine* 22 (2005), Nr. 1, S. 44–54
- [Hansen u. a. 2010] HANSEN, N. ; AUGER, A. ; ROS, R. ; FINCK, S. ; POSÍK, P.: Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, 2010, S. 1689–1696
- [Hansen und Kern 2004] HANSEN, N. ; KERN, S.: Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In: *Proceedings of the International Conference on Parallel Problem Solving from Nature*, 2004, S. 282–291
- [Hansen u. a. 2003] HANSEN, N. ; MÜLLER, S.D. ; KOUMOUTSAKOS, P.: Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). In: *Evolutionary Computation* 11 (2003), Nr. 1, S. 1–18
- [Hansen und Ostermeier 1996] HANSEN, N. ; OSTERMEIER, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1996, S. 312–317
- [Hansen und Ostermeier 2001] HANSEN, N. ; OSTERMEIER, A.: Completely Derandomized Self-Adaptation in Evolution Strategies. In: *Evolutionary Computation* 9 (2001), Nr. 2, S. 159–195
- [Hansen u. a. 2012] HANSEN, P. ; ALISMAIL, H. ; RANER, P. ; BROWNING, B.: Online continuous stereo extrinsic parameter estimation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012, S. 1059–1066
- [Hartley und Zisserman 2004] HARTLEY, R. I. ; ZISSERMAN, A.: *Multiple View Geometry in Computer Vision*. 2. Auflage. Cambridge University Press, 2004

- [Hastie u. a. 2001] HASTIE, T. ; TIBSHIRANI, R. ; FRIEDMAN, J.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001
- [Hermann u. a. 2011] HERMANN, S. ; MORALES, S. ; KLETTE, R.: Half-resolution semi-global stereo matching. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011, S. 201–206
- [Hirschmüller 2005] HIRSCHMÜLLER, H.: Accurate and efficient stereo processing by semi-global matching and mutual information. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005, S. 807–814
- [Hirschmüller 2006] HIRSCHMÜLLER, H.: Stereo Vision in Structured Environments by Consistent Semi-Global Matching. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006, S. 2386–2393
- [Hirschmüller 2008] HIRSCHMÜLLER, H.: Stereo Processing by Semiglobal Matching and Mutual Information. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30 (2008), Nr. 2, S. 328–241
- [Hirschmüller und Scharstein 2007] HIRSCHMÜLLER, H. ; SCHARSTEIN, D.: Evaluation of Cost Functions for Stereo Matching. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007, S. 1–8
- [Horn und Schunck 1981] HORN, B. K. P. ; SCHUNCK, B. G.: Determining Optical Flow. 17 (1981), S. 185–203
- [Houben 2011] HOUBEN, S.: A single target voting scheme for traffic sign detection. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011, S. 124–129
- [Houben u. a. 2013] HOUBEN, S. ; SCHLIPSING, M. ; STALLKAMP, J. ; SALMEN, J. ; IGEL, C.: Proposal for IJCNN 2013 competition: The German Traffic Sign Detection Benchmark. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2013
- [Huang u. a. 2009] HUANG, Z. ; DING, K. ; JIN, L. ; GAO, X.: Writer Adaptive Online Handwriting Recognition Using Incremental Linear Discriminant Analysis. In: *Proceedings of the IEEE International Conference on Document Analysis and Recognition*, 2009, S. 91–95
- [Huguet und Deverna 2007] HUGUET, F. ; DEVERNA, F.: A Variational Method for Scene Flow Estimation from Stereo Sequences. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2007, S. 1–7
- [Humenberger u. a. 2010a] HUMENBERGER, M. ; ENGELKE, T. ; KUBINGER, W.: A census-based stereo vision algorithm using modified Semi-Global Matching and plane fitting to improve matching quality. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2010, S. 77–84

- [Humenberger u. a. 2010b] HUMENBERGER, M. ; ZINNER, C. ; WEBER, M. ; KUBINGER, W. ; VINCZE, M.: A fast stereo matching algorithm suitable for embedded real-time systems. In: *Computer Vision and Image Understanding* 114 (2010), Nr. 11, S. 1180–1202
- [Igel u. a. 2008] IGEL, C. ; GLASMACHERS, T. ; HEIDRICH-MEISNER, V.: Shark. In: *Journal of Machine Learning Research* 9 (2008), S. 993–996
- [Igel u. a. 2007] IGEL, C. ; HANSEN, N. ; ROTH, S.: Covariance Matrix Adaptation for Multi-objective Optimization. In: *Evolutionary Computation* 15 (2007), Nr. 1, S. 1–28
- [Igel und Kreutz 2003] IGEL, C. ; KREUTZ, M.: Operator Adaptation in Evolutionary Computation and its Application to Structure Optimization of Neural Networks. In: *Neurocomputing* 55 (2003), Nr. 1–2, S. 347–361
- [Jain u. a. 2000] JAIN, A. K. ; DUIN, R. P. W. ; MAO, J.: Statistical Pattern Recognition: A Review. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), Nr. 1, S. 4–37
- [Keller u. a. 2011] KELLER, C. ; ENZWEILER, M. ; GAVRILA, D. M.: A New Benchmark for Stereo-based Pedestrian Detection. In: *Proceedings of the IEEE Intelligent Vehicles Symposium, 2011*, S. 691–696
- [Khammari u. a. 2005] KHAMMARI, A. ; NASHASHIBI, F. ; ABRAMSON, Y. ; LAURGEAU, C.: Vehicle detection combining gradient analysis and AdaBoost classification. In: *Proceedings of the IEEE Intelligent Transportation Systems Conference, 2005*, S. 66–71
- [Kim u. a. 2007] KIM, T.-K. ; WONG, S.-F. ; STENGER, B. ; KITTLER, J. ; CIPOLLA, R.: Incremental Linear Discriminant Analysis Using Sufficient Spanning Set Approximations. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2007*, S. 1–8
- [Kirkpatrick u. a. 1983] KIRKPATRICK, S. ; GELATT, C.D. ; VECCHI, M.P.: Optimization by Simulated Annealing. In: *Science* 220 (1983), Nr. 4598, S. 671–680
- [Klette u. a. 2011] KLETTE, R. ; KRUGER, N. ; VAUDREY, T. ; PAUWELS, K. ; HULLE, M. van ; MORALES, S. ; KANDIL, F.I. ; HAEUSLER, R. ; PUGEAULT, N. ; RABE, C. ; LAPPE, M.: Performance of Correspondence Algorithms in Vision-Based Driver Assistance Using an Online Image Sequence Database. In: *IEEE Transactions on Vehicular Technology* 60 (2011), Nr. 5, S. 2012–2026
- [Lienhart und Maydt 2002] LIENHART, R. ; MAYDT, J.: An Extended Set of Haar-Like Features for Rapid Object Detection. In: *Proceedings of the IEEE International Conference on Image Processing, 2002*, S. 900–903

- [van der Mark und Gavrila 2006] MARK, W. van der ; GAVRILA, D. M.: Real-time dense stereo for intelligent vehicles. In: *IEEE Transactions on Intelligent Transportation Systems* 7 (2006), Nr. 1, S. 38–50
- [Michael u. a. 2013] MICHAEL, M. ; SALMEN, J. ; STALLKAMP, J. ; SCHLIPSING, M.: Real-Time Stereo Vision: Optimizing Semi-Global Matching. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2013
- [Micusik und Kosecka 2009] MICUSIK, B. ; KOSECKA, J.: Piecewise planar city 3D modeling from street view panoramic sequences. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, S. 2906–2912
- [Mohri u. a. 2012] MOHRI, M. ; ROSTAMIZADEH, A. ; TALWALKAR, A.: *Foundations of Machine Learning*. 1. Auflage. MIT Press, 2012
- [Morales und Klette 2009] MORALES, S. ; KLETTE, R.: A Third Eye for Performance Evaluation in Stereo Sequence Analysis. In: *Proceedings of the International Conference on Computer Analysis of Images and Patterns*, 2009, S. 1078–1086
- [Morales u. a. 2009] MORALES, S. ; VAUDREY, T. ; KLETTE, R.: Robustness Evaluation of Stereo Algorithms on Long Stereo Sequences. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2009, S. 347–352
- [Munder und Gavrila 2006] MUNDER, S. ; GAVRILA, D. M.: An Experimental Study on Pedestrian Classification. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28 (2006), Nr. 11, S. 1863–1868
- [Ohta und Kanade 1985] OHTA, Y. ; KANADE, T.: Stereo by Intra- and Inter-Scanline Search using Dynamic Programming. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 7 (1985), S. 139–154
- [Overett und Petersson 2011] OVERETT, G. ; PETERSSON, L.: Large scale sign detection using HOG feature variants. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011, S. 326–331
- [Pang u. a. 2005] PANG, S. ; OZAWA, S. ; KASABOV, N.: Incremental linear discriminant analysis for classification of data streams. In: *IEEE Transactions on Systems, Man, and Cybernetics* 35 (2005), Nr. 5, S. 905–914
- [Ramanath u. a. 2002] RAMANATH, R. ; SNYDER, W. E. ; BILBRO, G. L. ; SANDER, W. A.: Demosaicking methods for Bayer color arrays. In: *Journal of Electronic Imaging* 11 (2002), S. 306–315
- [Rechenberg 1973] RECHENBERG, I.: *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973

- [Salmen u. a. 2011] SALMEN, J. ; CAUP, L. ; IGEL, C.: Real-Time Estimation of Optical Flow based on Optimized Haar Wavelet Features. In: *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, 2011, S. 448–461
- [Salmen u. a. 2012] SALMEN, J. ; HOUBEN, S. ; SCHLIPSING, M.: Google Street View Images Support the Development of Vision-Based Driver Assistance Systems. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2012, S. 891–895
- [Salmen u. a. 2009] SALMEN, J. ; SCHLIPSING, M. ; EDELBRUNNER, J. ; HEGEMANN, S. ; LUEKE, S.: Real-Time Stereo Vision: Making more out of Dynamic Programming. In: *Proceedings of the International Conference on Computer Analysis of Images and Patterns*, 2009, S. 1096–1103
- [Salmen u. a. 2010] SALMEN, J. ; SCHLIPSING, M. ; IGEL, C.: Efficient Update of the Covariance Matrix Inverse in Iterated Linear Discriminant Analysis. In: *Pattern Recognition Letters* 31 (2010), S. 1903–1907
- [Salmen u. a. 2007] SALMEN, J. ; SUTTORP, T. ; EDELBRUNNER, J. ; IGEL, C.: Evolutionary Optimization of Wavelet Feature Sets for Real-Time Pedestrian Classification. In: *Proceedings of the IEEE Conference on Hybrid Intelligent Systems*, 2007, S. 222–227
- [Scharstein und Szeliski 2002] SCHARSTEIN, D. ; SZELISKI, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In: *International Journal of Computer Vision* 47 (2002), S. 7–42
- [Scharstein und Szeliski 2003] SCHARSTEIN, D. ; SZELISKI, R.: High-accuracy stereo depth maps using structured light. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* Bd. 1, 2003, S. 195–202
- [Schlipping u. a. 2012] SCHLIPSING, M. ; SALMEN, J. ; LATTKE, B. ; SCHRÖTER, K. ; WINNER, H.: Roll Angle Estimation for Motorcycles: Comparing Video and Inertial Sensor Approaches. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2012, S. 500–505
- [Schlipping u. a. 2011] SCHLIPSING, M. ; SCHEPANEK, J. ; SALMEN, J.: Video-Based Roll Angle Estimation for Two-Wheeled Vehicles. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011, S. 876–881
- [Schwefel 1977] SCHWEFEL, H.P.: *Interdisciplinary Systems Research*. Bd. 26: *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser, 1977

- [Shi und Tomasi 1994] SHI, J. ; TOMASI, C.: Good features to track. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1994, S. 593–600
- [Stallkamp u. a. 2011] STALLKAMP, J. ; SCHLIPSING, M. ; SALMEN, J. ; IGEL, C.: The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In: *Proceedings of the IEEE International Joint Conference on Neural Networks*, 2011, S. 1453–1460
- [Stallkamp u. a. 2012] STALLKAMP, J. ; SCHLIPSING, M. ; SALMEN, J. ; IGEL, C.: Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition. In: *Neural Networks* 32 (2012), S. 323–332
- [Statistisches Bundesamt 2011a] STATISTISCHES BUNDESAMT: *Verkehrsunfälle – Zweiradunfälle im Sträß $\frac{1}{2}$ enverkehr 2010*. 2011
- [Statistisches Bundesamt 2011b] STATISTISCHES BUNDESAMT: *Verkehrsunfälle 2010*. Fachserie 8 Reihe 7, 2011
- [Stein 2004] STEIN, F.: Efficient Computation of Optical Flow Using the Census Transform. In: *Proceedings of the DAGM Symposium*, 2004, S. 79–86
- [Steinbruecker u. a. 2009] STEINBRUECKER, F. ; POCK, T. ; CREMERS, D.: Large Displacement Optical Flow Computation without Warping. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2009, S. 1609–1614
- [Sun u. a. 2003] SUN, J. ; SHUM, H.-Y. ; ZHENG, N.-N.: Stereo Matching using Belief Propagation. In: *IEEE Transaction on Pattern Analysis and Machine Intelligence* 25 (2003), Nr. 7, S. 787–800
- [Suttorp u. a. 2009] SUTTORP, T. ; HANSEN, N. ; IGEL, C.: Efficient Covariance Matrix Update for Variable Metric Evolution Strategies. In: *Machine Learning* 75 (2009), Nr. 2, S. 167–197
- [Tapp und Kemsley 2008] TAPP, H. S. ; KEMSLEY, E. K.: Optimizing the efficiency of cross-validation in linear discriminant analysis through selective use of the Sherman-Morrison-Woodbury inversion formula. In: *Journal of Chemometrics* 22 (2008), Nr. 6, S. 419–421
- [Trunk 1979] TRUNK, G. V.: A Problem of Dimensionality: A Simple Example. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1 (1979), Nr. 3, S. 306–307
- [Vapnik 1998] VAPNIK, V.: *Statistical Learning Theory*. Wiley, 1998

- [Vaudrey u. a. 2011] VAUDREY, T. ; MORALES, S. ; WEDEL, A. ; KLETTE, R.: Generalised residual images' effect on illumination artifact removal for correspondence algorithms. In: *Pattern Recognition* 44 (2011), Nr. 9, S. 2034–2046
- [Vaudrey u. a. 2008] VAUDREY, T. ; RABE, C. ; KLETTE, R. ; MILBURN, J.: Differences Between Stereo and Motion Behaviour on Synthetic and Real-World Stereo Sequences. In: *Proceedings of the Conference on Image and Vision Computing New Zealand*, IEEE Press, 2008, S. 1–6
- [Veksler 2005] VEKSLER, O.: Stereo Correspondence by Dynamic Programming on a Tree. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* Bd. 2, 2005, S. 384–390
- [Villamizar u. a. 2006] VILLAMIZAR, M. ; SANFELIU, A. ; ANDRADE-CETTO, J.: Computation of Rotation Local Invariant Features using the Integral Image for Real Time Object Detection. In: *Proceedings of the IEEE International Conference on Pattern Recognition* Bd. 4, 2006, S. 81–85
- [Viola und Jones 2001] VIOLA, P. ; JONES, M.: Robust Real-time Object Detection. In: *International Journal of Computer Vision* 57 (2001), Nr. 2, S. 137–154
- [Walk u. a. 2010a] WALK, S. ; MAJER, N. ; SCHINDLER, K. ; SCHIELE, B.: New features and insights for pedestrian detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010, S. 1030–1037
- [Walk u. a. 2010b] WALK, S. ; SCHINDLER, K. ; SCHIELE, B.: Disparity Statistics for Pedestrian Detection: Combining Appearance, Motion and Stereo. In: *Proceedings of the European Conference on Computer Vision*, 2010, S. 182–195
- [Wedel u. a. 2011] WEDEL, A. ; BROX, T. ; VAUDREY, T. ; RABE, C. ; FRANKE, U. ; CREMERS, D.: Stereoscopic Scene Flow Computation for 3D Motion Understanding. In: *International Journal of Computer Vision* 95 (2011), Nr. 1, S. 29–51
- [Wesierski u. a. 2012] WESIERSKI, D. ; MKHININI, M. ; HORAIN, P. ; JEZIERSKA, A.: Fast recursive ensemble convolution of Haar-like features. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012, S. 3689–3696
- [Wojek und Schiele 2008] WOJEK, C. ; SCHIELE, B.: A Performance Evaluation of Single and Multi-feature People Detection. In: *Proceedings of the 30th DAGM symposium on Pattern Recognition*, 2008, S. 82–91
- [Xu u. a. 2012] XU, L. ; JIA, J. ; MATSUSHITA, Y.: Motion Detail Preserving Optical Flow Estimation. In: *IEEE Transaction on Pattern Analysis and Machine Intelligence* 34 (2012), Nr. 9, S. 1744–1757

- [Yong u. a. 2011] YONG, X. ; ZHANG, L. ; SONG, Z. ; HU, Y. ; ZHENG, L. ; ZHANG, J.: Real-time vehicle detection based on Haar features and Pairwise Geometrical Histograms. In: *IEEE International Conference on Information and Automation*, 2011, S. 390–395
- [Zabih und Woodfill 1994] ZABIH, R. ; WOODFILL, J.: Non-Parametric Local Transforms for Computing Visual Correspondence. In: *Proceedings of the European Conference on Computer Vision*, 1994, S. 151–158
- [Zaklouta und Stanciulescu 2011a] ZAKLOUTA, F. ; STANCIULESCU, B.: Segmentation masks for real-time traffic sign recognition using weighted HOG-based trees. In: *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, 2011, S. 1954–1959
- [Zaklouta und Stanciulescu 2011b] ZAKLOUTA, F. ; STANCIULESCU, B.: Warning Traffic Sign Recognition using a HOG-based K-d Tree. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011, S. 1019–1024
- [Zamir und Shah 2010] ZAMIR, A. R. ; SHAH, M.: Accurate Image Localization Based on Google Maps Street View. In: *Proceedings of the European Conference on Computer Vision*, 2010, S. 255–268
- [Zhao 2000] ZHAO, H.: Global Optimal Surface from Stereo. In: *Proceedings of the IEEE International Conference on Pattern Recognition* Bd. 1, 2000, S. 101–104
- [Zheng und Liang 2009] ZHENG, W. ; LIANG, L.: Fast car detection using image strip features. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, S. 2703–2710