Masterarbeit

am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Intelligente Systeme und Robotik

Fahrzeugerfassung mithilfe von Stereo-Vision sowie HOG-, LBP-, und Haar-Feature-basierter Bildklassifikatoren

Daniel Neumann

Matrikelnummer: 4298730

daniel.neumann@fu-berlin.de

Erstgutachter: Prof. Dr. Raúl Rojas

Zweitgutachter: Prof. Dr. Daniel Göhring

Betreuer: Tobias Langner, Fritz Ulbrich

Berlin, 11. Januar 2016

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

11. Januar 2016

Daniel Neumann

Zusammenfassung

Deutsch:
Die Erkundung der Umgebung ist eine essentielle Fähigkeit für autonomes Fahren. Mithilfe von Sensoren können in der Nähe befindliche Objekte sichtbar gemacht werden. In dieser Arbeit wird das Erkennen von Fahrzeugen anhand von Stereokamerabildern untersucht. Es wird maschinelles räumliches Sehen eingesetzt, um Objekte der Umgebung zu lokalisieren. Zusätzlich werden mit Bildklassifikatoren Objekte als Fahrzeuge identifiziert. Verschiedene Algorithmen aus dem Bereich der bildbasierten Mustererkennung kommen dabei zum Einsatz und werden evaluiert. Im Rahmen des AutoNOMOS-Projekt der Arbeitsgruppe 'Intelligente Systeme und Robotik' wurde ein System implementiert, dass es ermöglicht, Fahrzeuge in Echtzeit zu erkennen.

English:
Investigation of environment is an essential property of autonomous driving. With the help of sensors nearby objects can be uncovered. In this thesis, the detection of vehicles on the basis of stereo camera images is analyzed. Stereoscopic computer vision is appointed to localize objects of the environment. Additionally, objects will identified by image classifiers. Several algorithms within the scope of image based pattern recognition are used and evaluated. As part of the 'Intelligente Systeme und Robotik' research group's project AutoNOMOS, a system is implemented which provides vehicle detection in realtime.

Inhaltsverzeichnis

1 Einleitung

Fahrassistenzsysteme sind heute schon in modernen Fahrzeugen weit verbreitet. Dazu gehören Ein-/Ausparkhilfen, die dem Fahrer über Sensoren die Abstände zu anderen Objekten der Umgebung signalisieren oder sogar den gesamten Parkvorgang fast automatisch vornehmen können. Es gibt Abstandsregeltempomate, welche die Geschwindigkeit an vorausfahrende Fahrzeuge anpassen, sowie Notbremssysteme, die vor einer Kollision warnen bzw. eine Notbremsung selbständig einleiten können. Des weiteren werden Spurhalte- und Spurwechselassistenten eingesetzt, durch die das Fahrzeug automatisch in der momentanen Spur gehalten bzw. bei einem Spurwechsel über Kollisionsgefahren informiert wird.

Dies sind nur einige der bereits existierenden Fahrassistenzsysteme, die den Fahrer unterstützen und teilweise schon eine Art von Autonomie des Fahrzeugs darstellen. Langfristig werden sie dem Fahrer vermehrt Aufgaben abnehmen und können für mehr Komfort, Sicherheit und Effizienz im Straßenverkehr sorgen. Es ist abzusehen, dass in ferner Zukunft vollständig autonome Fahrzeuge zum Alltag gehören. An der Freien Universität wird daran im Rahmen des **Projekts AutoNOMOS** seit vielen Jahren geforscht.

1.1 Das AutoNOMOS-Projekt

Der Vorläufer zum AutoNOMOS-Projekt wurde im Jahr 2006 von Prof. Dr. Raúl Rojas und seinen Studenten innerhalb der Arbeitsgruppe Künstliche Intelligenz an der Freien Universität Berlin gegründet. Es wurde das Fahrzeug *Spirit of Berlin*, ein Dodge Grand Caravan, mit Sensoren und Computerhardware ausgestattet und zu einem autonomen Fahrzeug umgebaut. Bereits seit 2007 wurden Tests auf dem Gelände des ehemaligen Flughafens Berlin-Tempelhof im autonomen Betrieb durchgeführt.

Durch durch Unterstützung des Bundesministeriums für Bildung und Forschung wurde 2009 das AutoNOMOS-Projekt ins Leben gerufen und zwei neue Testfahrzeuge entwickelt. Das eine Fahrzeug ist *e-Instein*, ein Elektrofahrzeug des Typs Mitsubishi i-MiEV und das andere ist *MadeInGermany*, ein Volkswagen Passat Variant 3c. *MadeInGermany* ist mit mehr Technik ausgestattet und wird primär für Testfahrten genutzt, weshalb ich mich in dieser Arbeit ausschließlich auf dieses Fahrzeug beziehen werde.

Abbildung 1.1: Testfahrzeug MadeInGermany [1]

Das Auto enthält eine Drive-by-Wire-Technologie, die es ermöglicht Motor, Bremsen, Schaltung und andere mechanische Komponenten per Software über den so genannten vernetzten CAN-BUS anzusteuern. Außerdem besitzt es u.a. folgende Sensoren:

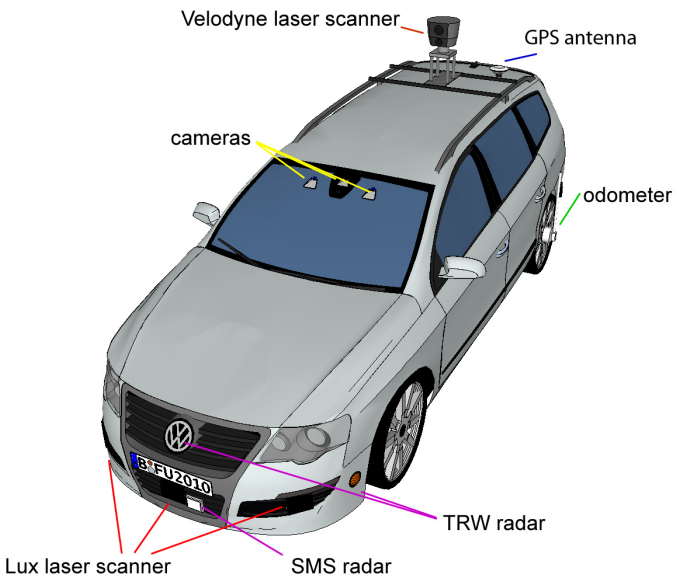


Abbildung 1.2: Sensoren von MadeInGermany [1]

- **Kamera im Rückspiegel (Serie):** Für Lichtautomatik und Spurhalteassistent.
- **Hella Aglaia INKA-Kameras:** Zwei nach vorn ausgerichtete Stereokameras, um den Rückspiegel montiert.

- **Lux Laser Scanner:** An Front und Heck im Stoßstangenbereich zur Erkennung von Hindernissen.
- **Smart Microwave Sensors GmH Radar:** An der Frontstoßstange zur Geschwindigkeitsermittlung vorausfahrender Fahrzeuge.
- **TRW / Hella Radare:** Im Front- sowie im Heckbereich zur Abstandsmessung zu umgebenen Objekten.
- **GPS-Antenne (Applanix POS/LV System):** Auf dem Dach zur Positionsbestimmung.
- **Velodyne HDL-64e:** LIDAR-System zur Hinderniserkennung rund um das Auto.
- **Odometer (Applanix POS/LV System):** Am Hinterrad zur Messung der Radumdrehungen und Berechnung zurückgelegter Distanzen.

Es sind hierbei nicht alle Sensoren aufgeführt, sondern nur die wesentlichsten jeder Art. Mit der Zeit ändern sich die Sensoren des Fahrzeugs hin und wieder. Es werden neue getestet, wie aktuell auch ein Infrarot-Sensor, oder wurden durch neuere ersetzt bzw. sind zur Zeit nicht in Betrieb und ausgebaut.

Für das Thema dieser Arbeit stütze ich mich einzig und allein auf die Bilder der beiden Hella Aglaia INKA-Kameras, da alles rein bildbasiert ablaufen soll. Natürlich könnte man auch noch die Daten Laser- und Radarsensoren heranziehen, aber das würde den Rahmen dieser Arbeit übersteigen. Der Vorteil der Beschränkung der Kamerabilder als Eingangssensoren ist außerdem, dass sich das rein bildbasierte Erkennungssystem leichter auf andere Fahrzeuge oder Geräte übertragen lässt, die lediglich mit Kameras ausgestattet sind.
Die beiden INKA-Kameras der Firma Hella Aglaia liefern **Graustufenbilder** der Auflösung von **768x488 Pixeln** bei bis zu **60fps** (Bildern pro Sekunde). Sie sind außerdem so konstruiert, dass sie sich gut für die Stereo-Vision, d.h. für räumliches Sehen eignen, worauf ich in den späteren Kapiteln eingehen werde.

Das Team des AutoNOMOS-Projekts erforscht Grundlagen und entwickelt intelligente Software für die Testfahrzeuge, die das autonome Fahren ermöglichen sollen. Dabei werden auch die Hardwarekomponenten der Testfahrzeuge kontinuierlich optimiert und erweitert.

1.2 Ziel und Aufbau der Arbeit

Das Ziel dieser Arbeit ist es, ein System für autonome Fahrzeuge zu entwerfen und zu entwickeln, dass eine **bildbasierte Erkennung und Identifizierung von Objekten in der Umgebung** ermöglicht. Als Anwendungsfall wurde die Erkennung von Rückansichten von **Personenkraftwagen (PKW)** gewählt und in das AutoNOMOS-Projekt implementiert.

Zur Erkennung der Fahrzeuge kommen zwei bildbasierte Verfahren zum Einsatz. Zum einen wird mit dem **Stereo-Vision-Ansatz** durch räumliches Sehen eine grobe Objekterfassung durchgeführt. Und zum anderen werden die Objekte anschließend durch **Bildklassifikation** mit Mustererkennungsalgorithmen identifiziert. Dabei wird konkret untersucht, ob es sich bei dem Objekt um eine PKW-Rückansicht handelt.

Im Kapitel 2 *Theoretische Grundlagen* werden die Prinzipien der eingesetzten Verfahren erklärt. Es wird ein Grundwissen zum generellen Verständnis der Funktionsweise vermittelt. Das danach folgende Kapitel 3 *Implementierung der Fahrzeugerkennung* gibt Aufschluss über die Details zur Umsetzung der Theorie in funktionierende Software. Im Anschluss werden im Kapitel 4 *Experimentelle Auswertung* meine Ergebnisse der Implementierung präsentiert und bewertet. Im letzten Kapitel 5 schließe ich die Arbeit mit einem Fazit und Ausblick ab.

1.3 Verwandte Arbeiten

Für die AutoNOMOS-Software existieren zahlreiche Möglichkeiten zur Hindernis- bzw. Objekterkennung. Die meisten davon nutzen die Radar-, LIDAR- oder Laser-Sensoren. Doch die lokalisierten Objekte können häufig noch nicht identifiziert werden. In vielen Situation wäre dies wichtig, um z.B. zu unterscheiden, ob es auf der Fahrbahn um einen Polizisten handelt, der den Verkehr regelt oder um ein Kind, dass plötzlich über die Straße läuft. Für solche Unterscheidungen sind die Daten aus Kameraaufnahmen essentiell. Objekterkennung in Kamerabildern ist teilweise schon im Projekt vorhanden, wie z.B. für Ampelerkennung. Aber die vorhandenen Implementierungen sind sehr speziell auf einen Anwendungsfall ausgerichtet. Eine allgemeingültige Objekterkennung mithilfe von Stereo-Vision und existierenden Mustererkennungsalgorithmen der vom Projekt verwendeten OpenCV-Bibliothek gab es vor meiner Arbeit nicht.

Im Bereich der Stereo-Vision gibt es viel Literatur zu den theoretischen Grundlagen des maschinellen räumlichen Sehens. In einigen Werken findet

man auch Ansätze zur Objekterkennung, siehe [2] [3] [4].
Sie basieren auf den Ansatz, die projizierten Bildpunkte der Kameras ins 3D-Koordinatensystem durch Clusteranalyse zu Objekten zusammen zu fassen. Diesen Ansatz hat auch schon einer meiner Betreuer der Arbeit verfolgt, Tobias Langner. Er hat bereits solch eine Clusteranalyse im AutoNOMOS-Projekt implementiert, die ich für mein Objekterkennungssystem verwenden konnte.

Um Objekte nicht nur lokalisieren, sondern auch identifizieren zu können, gibt es Algorithmen auf dem Gebiet der bildbasierten Mustererkennung. Zu den gängigen zählen Paul Viola und Michael Jones' Algorithmus [5], der zur Gesichtserkennung entworfen wurde, Navneet Dalal und Bill Triggs mit *Histograms of Oriented Gradients* [6] zur Fußgängererkennung sowie P.F. Felzenszwalb's Erweiterung *Deformable Part Models* [7] auf die ich mich in dieser Arbeit hauptsächlich beziehe.

Im Moment sind zudem Neuronale Netze sehr populär im Bereich des Mustererkennung und werden zunehmend auch für bildbasierte Verfahren eingesetzt. Zur gleichen Zeit in der diese Arbeit entstanden ist, wurde von einem anderen Studenten der Arbeitsgruppe in seiner Masterarbeit die Bildklassifikation mittels neuronaler Netze untersucht. Deshalb befasse ich mich mit anderen Algorithmen, die sich im Bereich der Musterkennung in Bildern etabliert haben.

2 Thereotische Grundlagen

Im folgenden Kapitel wird das Prinzip der Stereo-Vision und Mustererkennung erklärt. Die Stereo-Vision bietet eine Projektion von 2D-Kamerabildern in den dreidimensionalen Raum, wodurch eine Tiefenberechnung von den betrachteten Silhouetten möglich ist. Damit lassen sich Objekte lokalisieren, die im weiteren Schritt mit der Mustererkennung identifiziert werden können.

2.1 Stereo-Vision

Der englische Begriff **„Stereo-Vision“** leitet sich aus dem Griechischen „stereos“ für „Raum“ und dem Lateinischen „videre“ für „Sehen“ ab und bezeichnet somit das **räumliche Sehen**. In der Literatur werden u.a. auch die Begriffe **Stereoskopie** und **Stereoanalyse** verwendet. Da sich diese Arbeit nicht mit optischer Täuschung oder Filmtechnik befasst, sondern mit maschinellem Sehen (engl. computer vision), benutze ich den gängigen Begriff Stereo-Vision.

Mithilfe des maschinellen Sehens mit zwei oder mehr Kameras, ausgerichtet auf dieselbe Umgebung, kann ein **3D-Bild** konstruiert werden. Das heißt, die räumliche Tiefe lässt sich damit berechnen. Es sind prinzipiell nur zwei Kameras nötig, die ein leicht versetztes Bild im Vergleich zur anderen liefern. Bei sich bewegenden Perspektiven bzw. Objekten ist es außerdem möglich nur eine Kamera zu verwenden. Dabei werden allerdings zeitlich versetzte Kamerabilder benutzt, die durch Bewegung Aufnahmen entsprechen, als ob sie von verschiedenen Kameras zeitgleich an verschiedenen Orten aufgenommen wurden, deshalb auch als Struktur aus Bewegung (engl. structure from motion) bezeichnet.

Drei oder mehr Kameras dagegen können das Ergebnis im Vergleich zu zwei zusätzlich verbessern, ändern aber nichts am Grundprinzip der Tiefenberechnung. Deshalb beschränke ich mich in diesem Kapitel auf zwei Kameras.

Je nach Bedingungen und Qualität der Kameraaufnahmen ist das Ergebnis präziser. Wie bei uns Menschen ist das räumliche Sehen von den natürlichen Gegebenheiten, wie Licht bzw. Lichtreflektion sowie Kontrastunterschiede

der Objekte, abhängig. Zusätzlich spielen zudem auch die technischen Voraussetzungen der Kameras, wie Schärfe, Auflösung und Kontrast, eine wesentliche Rolle. Auch die freie Sicht auf ein Objekt ist entscheidend. Wenn es in einem oder gar in beiden Kamerabildern durch andere Objekte verdeckt ist, dann kann seine Tiefe nicht ermittelt werden.

2.1.1 Kameramodell

Zum generellen Grundverständnis muss zunächst das theoretische Modell einer Kamera im Raum beleuchtet werden. Zur einfachen Veranschaulichung ist in Abb. 2.1 der Aufbau einer **Lochkamera** dargestellt.

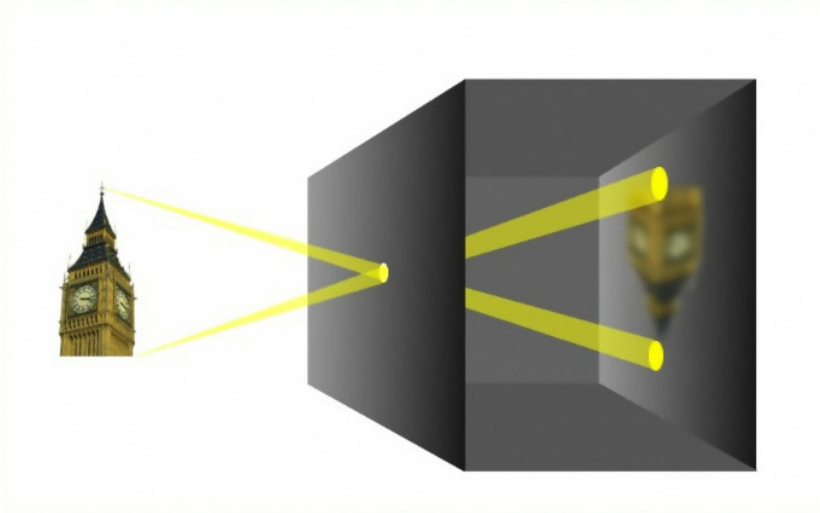


Abbildung 2.1: Prinzip einer Lochkamera [8]

Sie besteht aus einem schachtelartigen Körper mit einer kleinen Lochblende, durch welche die Lichtstrahlen von außen gebündelt eintreffen und auf die gegenüberliegende Innenwand gelenkt werden. Es entsteht dort ein auf dem Kopf stehendes Bild. Die Lochblende nennt man auch **optisches Zentrum** oder Brennpunkt der Kamera. Und die Ebene der Innenwand, auf der das Bild geworfen wird, ist die Bildebene oder verkürzt das Bild. Der Abstand zwischen Brennpunkt und der Bildebene ist die **Brennweite**.

Zur Vereinfachung wird ab nun die Bildebene vor anstatt hinter das optische Zentrum verlegt, wodurch die Spiegelung der vertikalen und horizontalen Achse entfällt.

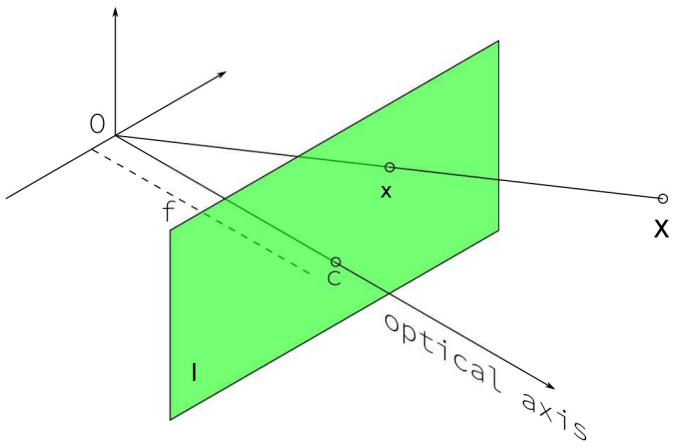


Abbildung 2.2: Lochkamera - mathematisches Modell

Ein Objektpunkt X mit den dreidimensionalen Koordinaten $X = (x, y, z)^T$ trifft auf das optische Zentrum O ein und wird im Punkt $x = (u, v)^T$ auf die Bildebene I projiziert.
Der Strahl, welcher durch O verläuft und genau orthogonal zur u- und v-Achse der Bildebene ist, wird optische Achse genannt und beschreibt die Blickrichtung der Kamera. Die optische Achse wird auf den Kamerahauptpunkt c abgebildet.

2.1.2 Epipolargeometrie

Die Epipolargeometrie beschreibt den mathematischen Zusammenhang, wenn mehrere Kameras das selbe Objekt aufnehmen. Es stellt die Beziehung zwischen Punkten eines Objektes her, die in den Kamerabildern an verschiedenen Bildkoordinaten wiederzufinden sind. Dadurch lässt sich der Suchraum für einen gegebenen Punkt aus Bild 1 der einen Kamera in Bild 2 einer anderen Kamera erheblich einschränken. Betrachten wir zunächst Abb. 2.3

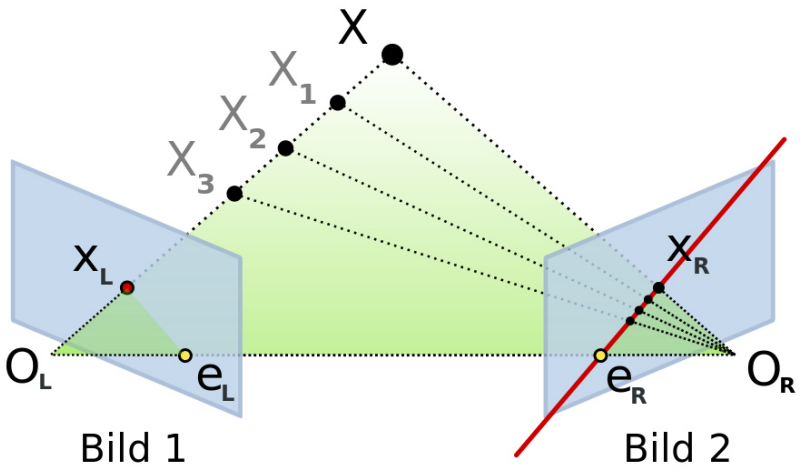


Abbildung 2.3: Epipolargeometrie [9]

Die Abbildung des Objektpunktes X wird in Bild 1 auf den Punkt x_L projiziert und liegt auf einem festen Strahl. Auf diesem Strahl befinden sich aber auch die näher liegenden Punkte $X1$, $X2$, $X3$, die ebenfalls auf x_L abgebildet werden. In Bild 2 liegen diese Punkte aber alle auf unterschiedlichen Strahlen. Die theoretisch unendlich vielen Punkte, die auf dem Strahl zwischen X und x_L liegen können spannen mit ihren Strahlen nach Kamera 2 eine Ebene auf, die so genannte Epipolarebene. Zusätzlich bilden diese Strahlen eine Gerade auf Bild 2 ab, die Epipolarlinie. Die Abbildung e_R des optischen Zentrums O_L der linken Kamera im Bild der rechten und umgekehrt e_L, nennt man Epipol.

Wenn die relative Lage und Ausrichtung der beiden Kameras zueinander sowie die internen Kameraeigenschaften wie z.B. die Brennweite bekannt sind, dann ist auch die Epipolarlinie in Bild 2 für den zugehörigen Bildpunkt x_L in Bild 1 bekannt, da der Objektpunkt geometrisch gesehen, nur dort abgebildet werden kann. Die beiden Bildpunkte, die den selben Objektpunkt abbilden, nennt man korrespondierende Punkte. Um einen korrespondierenden Bildpunkt aus Bild 1 in Bild 2 zu finden, muss man folglich nur auf der Epipolarlinie suchen, anstatt im ganzen Bild.

2.1.3 Kalibrierung

Um die relative Lage und Ausrichtung der beiden Kameras zu bestimmen, benötigt man die so genannten extrinsischen Kameraparameter.

Die **extrinsischen (äußere)** Parameter dienen der Orientierung und Position im Raum. Sie umfassen die Translation in Bezug auf einen festgelegten Koordinatenursprung sowie die Rotation um die Koordinatenachsen. Mathematisch lassen sich diese Parameter in einer Matrix, der **Essential-Matrix** E zusammenfassen.

Das Pendant zu den extrinsischen Kameraparametern sind die **intrinsischen**. Sie beschreiben die innere Geometrie einer Kamera. Dazu gehören im Wesentlichen die Brennweite, das optische Zentrum der Bildebene, Verzerrungskoeffizienten und die Pixelskalierung. Auch dafür kann man eine Berechnungsmatrix M aufstellen, die alle nötigen Parameter beinhaltet, um die Abbildung eines physischen Objektpunktes auf einen Bildpunkt zu bestimmen. Die Matrix M kann sich in der Realität auch aus einer Zusammensetzung mehrerer Matrizen ergeben, insbesondere wegen radialen Verzerrungen durch die Verwendung von Kameralinsen. Zur Vereinfachung beschränke ich mich jedoch auf eine Matrix, von der wir annehmen können, dass sie alle intrinsischen Parameter enthält.

Sind sowohl die extrinsischen als auch die intrinsischen Kameraparameter bekannt, dann sind die Kameras aufeinander abgestimmt (kalibriert).

Unabhängig von den intrinsischen Parametern stehen die beiden Kamerakoordinatensysteme über eine euklidische Transformation in Beziehung. Legt man den Ursprung in das Koordinatensystem KL von Kamera 1, so gilt für einen Punkt $pL \in \mathbb{R}^3$ in KL, dass er einen Punkt $pR \in \mathbb{R}^3$ im Koordinatensystem KR von Kamera 2 gleicht, verschoben um einen Translationsvektor $t \in \mathbb{R}^3$ und einer Drehung beschrieben durch die Rotationsmatrix $R \in \mathbb{R}^{3\times3}$:

$$p_L = R \cdot p_R + t$$

Rotation und Translation bilden die 3x3 Essential-Matrix E = t x R und zusammen mit pL und pR kann die so genannte **Epipolargleichung** aufgestellt werden:

$$p_R^T \cdot E \cdot p_L = 0$$

Für zwei korrespondierende Abbildungen eines 3D-Objektpunktes ist diese Gleichung erfüllt. Allerdings ist zu beachten, dass nicht umgekehrt 2 Bildpunkte automatisch korrespondierende Abbildungen sind, wenn sie die Epipolargleichung erfüllen.

Nimmt man im nächsten Schritt noch die intrinsischen Parameter hinzu, ergibt sich mit $x_L = M_L \cdot p_L$ bzw. $x_R = M_R \cdot p_R$ und $M_L, M_R \in \mathbb{R}^{3\times3}$, $x_L = (u_L, v_L, 1)^T$, $x_R = (u_R, v_R, 1)^T$ für die Abbildung auf einen konkreten Bildpunkt, folgende Gleichung:

$$x_R^T \cdot (M_R^{-1})^T \cdot E \cdot M_L^{-1} \cdot xL = x_R^T \cdot F \cdot x_L = 0$$

Die 3. Komponente der x_L- und x_R-Vektoren wird auf 1 gesetzt, da der Wert für die Tiefe in Bildkoordinaten unbekannt ist, aber zur Multiplikation mit den 3×3 Matrizen nötig ist. Die Bildkoordinaten werden somit in **homogene Koordinaten** $x = (k \cdot u, k \cdot v, k)^T$ umgewandelt.

Die Matrix $F = (M_R^{-1})^T \cdot E \cdot M_L^{-1}$ ist die 3×3 **Fundamental-Matrix** und beinhaltet alle nötigen Werte, um die Beziehung der beiden Bildebenen nach der Epipolargeometrie mathematisch fest zu halten. Damit ist es möglich für einen gegeben Punkt in Bild 1 alle Punkte in Bild 2 zu suchen, die auf der Epipolarlinie liegen, d.h. für welche die Epipolargleichung gilt.

Doch wie kann man die nötigen Parameter der Fundamental-Matrix ermitteln?

Schauen wir uns zunächst noch einmal die Epipolargleichung mit intrinsischen Parametern und Fundamental-Matrix an:

$$x_R^T \cdot F \cdot x_L = \begin{pmatrix} u_R & v_R & 1 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} u_L \\ v_L \\ 1 \end{pmatrix} = 0$$

Ausmultipliziert ergibt sich:

$$u_R u_L f_{11} + u_R v_L f_{12} + u_R f_{13} + v_R u_L f_{21} + v_R v_L f_{22} + v_R f_{23} + u_L f_{31} + v_L f_{32} + f_{33} = 0$$

Sind genügend Werte für x_i und y_i bekannt, so lassen sich mit einem linearen Gleichungssystem auch die Werte für alle f_i ermitteln. Das heißt also, mit einer **Mindestanzahl von Punktkorrespondenzen** kann man die Fundamental-Matrix erhalten. Es sind mindestens 7 Punktkorrespondenzen mit der Bedingung $det(F) = 0$ nötig. Mit 8 oder mehr lässt sich das Gleichungssystem einfacher lösen. Siehe dazu auch **7-Punkt bzw. 8-Punkt-Algorithmus**, auf die an dieser Steller aber nicht weiter eingegangen wird.

Die nötigen Punktkorrespondenzen können für diesen einmaligen Berechnungsschritt manuell festgelegt werden. Softwaregestützt kann dies auch automatisch z.B. mit der Computer Vision Bibliothek OpenCV geschehen. Dazu wird oft ein Schachbrettmuster von beiden Kameras mehrmals aufgenommen und die Software berechnet die nötigen Punktkorrespondenzen, die Fundamental-Matrix und somit auch die intrinsischen und extrinsischen Kameraparameter.

2.1.4 Rektifizierung

Um einen Punkt des einen Kamerabildes auch im anderen zu finden, ist es nützlich die beiden Bilder so auszurichten, dass sie achsenparallel sind und in der selben Ebene liegen (koplanar). Dazu rotiert man die Kamerakoordinatensysteme zueinander, so dass ihre optischen Achsen auch parallel sowie die Epipolarlinien parallel zu den horizontalen Achsen sind. Ein abgebildeter Punkt hat dann identische vertikale Koordinaten. Siehe Abb. 2.4.

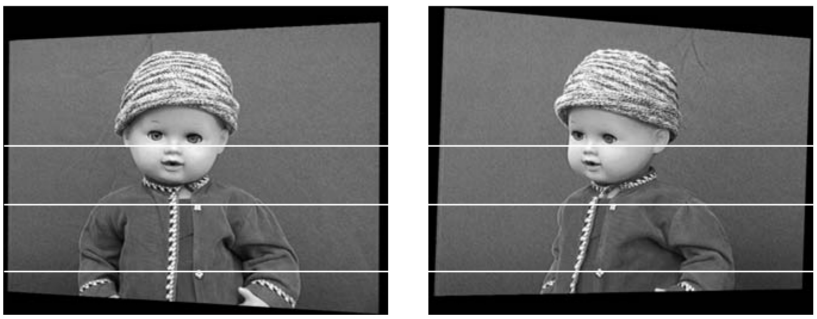


Abbildung 2.4: Rektifizierte Ansichten mit horizontalen Epipolarlinien [10]

Somit es nicht notwendig für jeden Punkt in Kamerabild 1 mühselig alle Punkte aus Bild 2 auf das Erfüllen der Epipolargleichung zu prüfen, um die Epipolarlinie zu bestimmen. Stattdessen werden die Kamerabilder so transformiert, dass die Epipolarlinien auf der selben Zeile in beiden Bildern liegen.

Diese Transformation nennt man **Rektifizierung** und hat den Vorteil, dass Punktabbildungen in beiden Bildern nur noch in einer Dimension, nämlich der in der horizontalen Achse, gesucht werden müssen. In der Vertikalen liegen die Punkte auf der selben Höhe. Man kann versuchen, einen Kameraaufbau zu verwenden, bei dem die beiden Bilder schon von vornherein so ausgerichtet sind, aber in der Realität ist das schwer umsetzbar bzw. gibt es meist geringe Abweichungen.

Eine Möglichkeit zur Rektifizierung ist Bouquet's Algorithmus [11].

Er setzt sich aus 2 Rotationsschritten zusammen. Zuerst werden beide Bildebenen jeweils zur Hälfte aufeinander zu gedreht. Dadurch ergibt sich eine Rotation $R = r_L^T \cdot r_R$, wobei r_L und r_R die beiden halben Drehung der linken bzw. rechten Bildebene sind.

Nach diesem Schritt sind die Bildebenen zwar koplanar, d.h. sie befinden sich in der selben Ebene, aber ihre Achsen sind noch nicht zwingend parallel.

Um auch das zu erreichen, benötigt es einer weiteren Rotation R_{rect}. Ziel ist es, beide Epipole ins Unendliche so zu verschieben, dass die zugehörigen Epipolarlinien auf der selben Geraden liegen, wie in Abb. 2.5 skizziert.

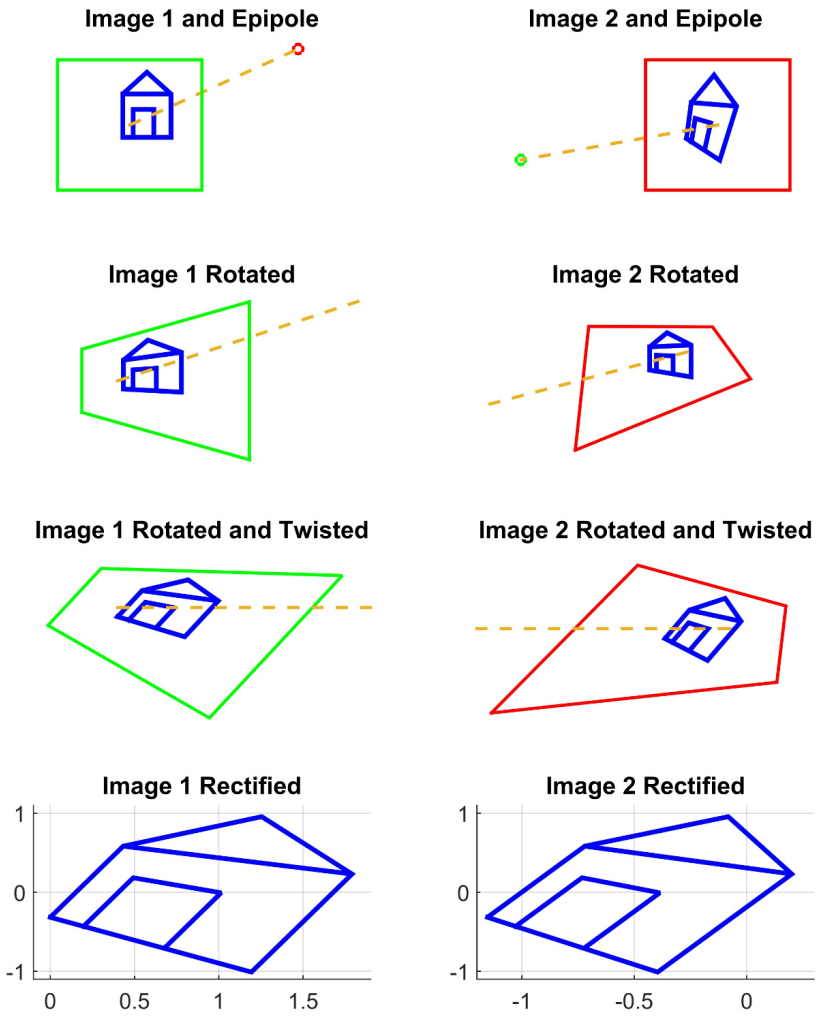


Abbildung 2.5: Die Rotationsschritte der Rektifizierung [12]

Die Vektoren der Epipole ergeben sich direkt aus dem Translationsvektor t der beiden optischen Kamerazentren zueinander:

$$R_{rect} = [e_1^T e_2^T e_3^T]^T$$

$$e_1 = \frac{t}{||t||}$$

$$e_2 = \frac{[-t_y t_x 0]^T}{\sqrt{t_x^2 + t_y^2}}$$
$$e_3 = e_1 \times e_2$$

Für die linke und rechte Kamera ergeben sich dann jeweils die folgenden Rotationen:

$$R_L = R_{rect} \cdot r_L$$
$$R_R = R_{rect} \cdot r_R$$

Nachdem diese beiden Rotationsschritte auf die Bildebene angewendet wurden, erhält man rektifizierte Bilder, deren Epipolarlinien auf der selben Bildzeile liegen.

2.1.5 Ermitteln von Punktpaaren mit Blockmatching

Wurden die beiden Bilder rektifiziert, kann zu einem gegebenen Punkt die Tiefe berechnet werden, wenn seine Lage in beiden Abbildungen bekannt ist. Doch wie findet man diese Lage?
Dieses Problem wird als **Korrespondenzproblem** bezeichnet, da zu einem gegebenen Punkt in Bild 1 der korrespondierende Punkt in Bild 2 gesucht wird, der das selbe Objekt abbildet. Mithilfe der Rektifizierung wissen wir bereits, dass die vertikalen Koordinaten identisch sind und wir horizontal vom Punkt in Bild 1 ausgehend in Bild 2 suchen müssen. Aufgrund der Epipolargeometrie können wir außerdem den Suchraum in den Bildern auf den überlappenden Bereich einschränken.

Allgemein lässt sich der Suchraum weiter einschränken, weil wir wissen, das sich Bild 1 links von Bild 2 befindet. Somit muss der korrespondierende Punkt in Bild 2 weiter links bzw. maximal auf der selben horizontalen Position wie in Bild 1 sein. Siehe Abb. 2.6:

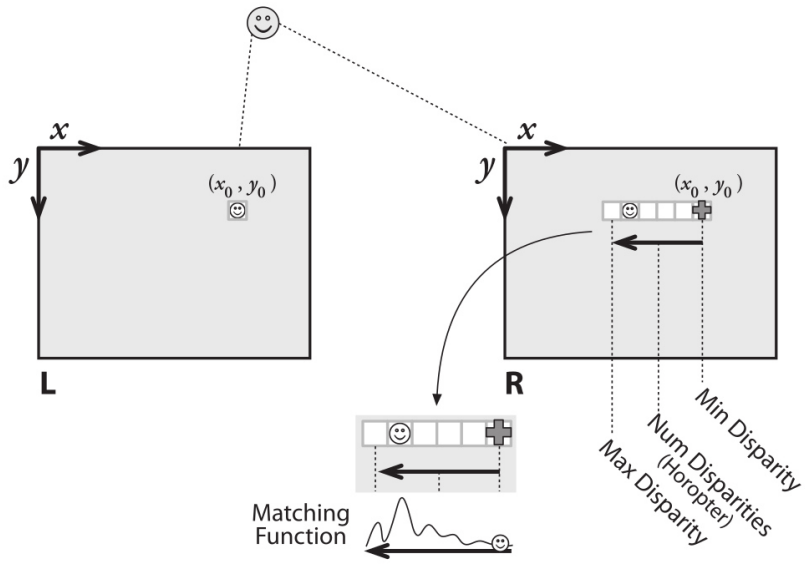


Abbildung 2.6: Suchraum für korrespondierende Punkte nach Rektifizierung [11]

Die genaue Position kann mit dem so genannten **Blockmatching-Alorithmus** bestimmt werden. Er wurde 1997 von Kurt Konolige entwickelt und ist ziemlich effektiv als auch schnell. Im Folgenden wird sein Prinzip erläutert.

Die drei Hauptschritte sind:

1. Vorverarbeitung der Bilder durch Normalisieren der Helligkeit
2. Korrespondenzsuche entlang der horizontalen Epipolarlinien mit einem SAD-Fenster
3. Herausfiltern von schlechten Korrespondenzergebnissen

Im ersten Schritt wird die Helligkeit normalisiert. Dies kann durch ein festgelegtes Fenster an Pixeln geschehen, z.B. 7x7, welches einmal für jeden Pixel des Bildes als Fensterzentrum berechnet wird. Der normalisierte Wert für das jeweilige Zentrum I_c wird unter Berücksichtigung des Durchschnitts aller Werte des momentanen Fensters $\bar{I}$ und einer globalen unteren bzw. oberen Schranke I_{cap} festgelegt.

$$min[max(I_c - \bar{I}, -I_{cap}), I_{cap}]$$

Als Zweites werden korrespondierende Punkte zwischen den beiden Bildern gesucht. Dabei wird ein Suchfenster entlang der horizontalen Epipolarlinie

geschoben und die beste Übereinstimmung ermittelt. Das heißt, im Bild 1 der linken Kamera wird an einer bestimmten Position das Suchfenster berechnet und im rechten Bild 2 ebenfalls an der selben sowie an weiteren Positionen links davon.

Es gibt verschiedene Berechnungsarten für das Suchfenster, z.B. Mean Square Error (MSE), Mean Absolute Difference (MAT). Ich beschränke mich auf die Variante des „Sum of Absolute Differences“ (SAD), da es auch im Blockmatching-Algorithmus der verwendeten OpenCV-Bibliothek zur Anwendung kommt. Ein SAD-Fenster ist in der Regel ein quadratischer Block von 5x5 bis 21x21 Pixeln innerhalb des Bildes. Um die Ähnlichkeit zweier Blöcke $B1$ und $B2$ zu bestimmen, ergibt sich der SAD-Wert nach folgender Formel:

$$SAD = \sum_{x_i} \sum_{y_i} |B_2(x_i, y_i) - B_1(x_i, y_i)|$$

Je niedriger der SAD-Wert, also die Summe der absoluten Differenzen, zweier Blöcke ist, desto ähnlicher sind sie. Die Verschiebung des am besten übereinstimmenden Blockes in Bild 2 im Vergleich zur ursprünglichen Position in Bild 1 ergibt die so genannte Disparität. Für weit entfernte Objekte ist sie kleiner als für näher befindliche, wie in Abb. 2.7 veranschaulicht.

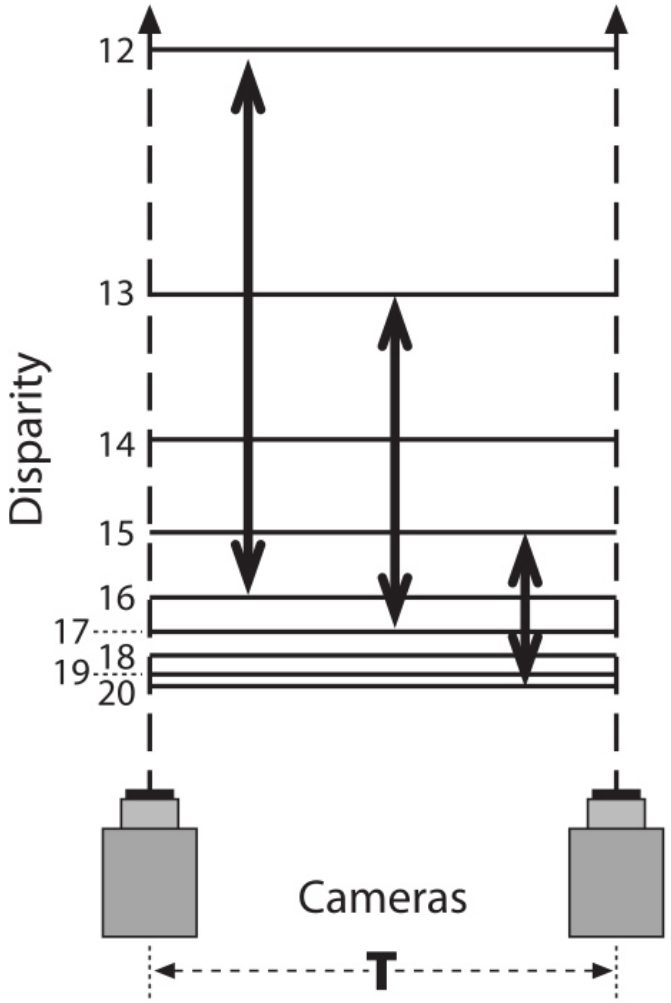


Abbildung 2.7: Verhältnis zwischen Disparität und Entfernung [11]

Die Größe des SAD-Fensters hat Einfluss auf die Ergebnisse. Große SAD-Fenster umfassen potenziell mehr Textur und somit einen größeren Unterschied im SAD-Wert zwischen 2 Blöcken. Jedoch dauert die Berechnung entsprechend länger und es kann bei sich wiederholenden Mustern innerhalb des Blockes dazu kommen, dass der SAD-Wert bei beiden Blöcken ähnlich ist, obwohl das Muster in den Blöcken zueinander verschoben ist. Größere SAD-Fenster liefern also ein geglätteres Ergebnis. Bei sehr kleinen Fenstern dagegen kommt es häufiger zu Fehlkorrespondenzen, da es bei geringer Textur weniger Unterschiede in den Pixelwerten gibt und sich somit 2 Blöcke häufiger ähneln. Natürlich spielt bei der Wahl der Fenstergröße auch die Bildauflösung eine entscheidende Rolle. Es gibt auch eine Erweiterung

des Blockmatching-Alorithmus mit einer Auflösungspyramide.
Dabei werden Korrespondenzen angefangen mit einer kleinen Bildauflösung durchsucht, deren Disparitäten den Startwert für die nächsthöhere Auflösung bilden. Der Suchbereich in der höheren Stufe wird dann auf den Bereich eingeschränkt der sich allein aus der Erhöhung der Auflösung ergibt. Dadurch lassen sich die Fehlkorrespondenzen von kleinen Suchfenstern in hohen Auflösungen minimieren, jedoch wirkt sich eine Fehlzuordnung in den ersten Stufen durch Fehlerfortpflanzung verheerender aus.

Um den Suchraum noch weiter einzugrenzen, bietet es sich an, eine Obergrenze für die Disparität festzulegen, die das Suchfenster in Bild 2 maximal nach links verschoben sein kann. Ansonsten würde die Berechnung für Pixel im rechten Teil des Bildes länger dauern als im linken, weil das Suchfenster im schlimmsten Fall vom rechten Ende der Bildzeile bis zum linken Ende geschoben wird. Mit steigender Disparität nimmt außerdem die Nähe zur Kamera exponentiell ab, so dass bei großen Disparitätswerten die ermittelte Tiefe nur minimal geringer ist, siehe Abb. 2.8:

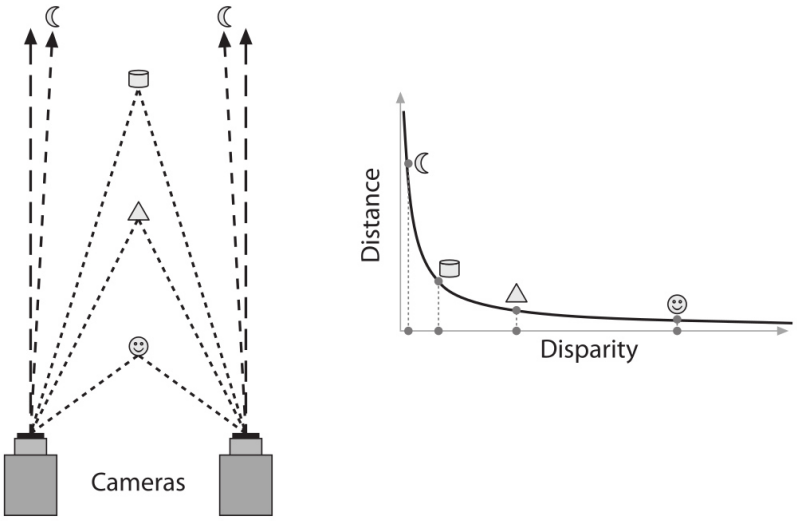


Abbildung 2.8: Verlauf der Entfernungen bei steigender Disparität [11]

Abhängig von der Auflösung der Kamerabilder ist auch die maximal mögliche Entfernung, die man anhand der Disparität berechnen kann. Ab einer bestimmten Entfernung geht die Disparität gegen 0, da der Objektpunkt aufgrund der endlichen Anzahl an Pixeln, in beiden Bildern auf die selbe Bildkoordinate abgebildet wird. Es kommt aber auch bei nicht so weit entfernten Objektpunkten vor, dass die Dispärität 0 ergibt, weil z.B. die Textur

sehr gleichmäßig ist und somit das Suchfenster keinen Unterschied an der selben Stelle in beiden Bildern feststellt. Diese Stellen mit einer Disparität von 0 werden deshalb als ungültig markiert, weil nicht bestimmt werden konnte, wie weit sie tatsächlich entfernt sind.

Zur Verbesserung der Ergebnisse gibt es auch noch weitere Nachverarbeitungsmöglichkeiten, wie z.B. Subpixeloptimierung oder Konsistenzprüfung.

Es ist noch zu erwähnen, dass in diesem Kapitel nur die horizontale Disparität beschrieben ist. Ebenso wäre es möglich die Kameras übereinander statt nebeneinander anzuordnen und vertikal in den Bildspalten nach korrespondierenden Punkten zu suchen.

2.1.6 Tiefenberechnung

Aus den Disparitätswerten können reale Distanzen durch Triangulation berechnet werden. Die Disparität d ist die Verschiebung in Pixel eines Bildpunktes x_R der rechten Kamera im Vergleich zum Bildpunkt x_L der linken, die beide den selben Objektpunkt abbilden.

$$d = x_L - x_R$$

Ein Objektpunkt $X = (X, Y, Z)^T$ im 3D-Koordinatensystem wird im Bildkoordinatensystem auf den Punkt $x = (u, v)^T$ abgebildet. Mit der Kamerabrennweite f ergibt sich folgende Relation:

$$\frac{u}{X} = \frac{v}{Y} = \frac{f}{Z}$$

Wie bereits im vorherigen Abschnitt erklärt wurde, ist die Tiefe indirekt proportional zur Disparität. Aus dem Abstand der beiden optischen Kamerazentren T (Absolutbetrag) und dem Strahlensatz lässt sich daraus eine weitere Gleichung aufstellen:

$$\frac{T}{Z} = \frac{T - d}{Z - f}$$

Veranschaulicht ist diese Beziehung auch in Abb. 2.9.

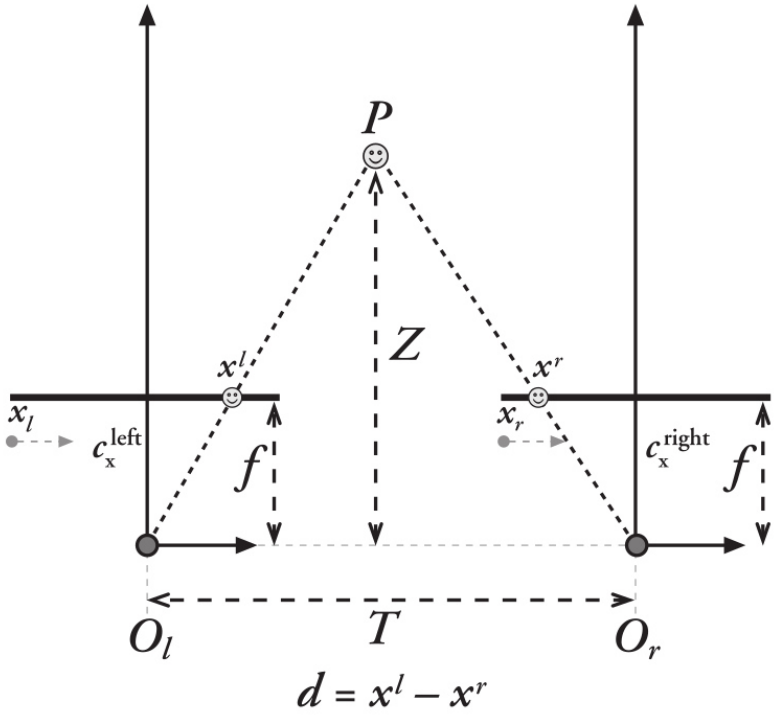


Abbildung 2.9: Epipolargeometrie nach Rektifizierung [11]

Für die Tiefe Z ergibt sich:

$$Z = \frac{f \cdot T}{d} = \frac{f \cdot T}{x_L - x_R}$$

Da f und T für die jeweilige Kamera konstant sind, lässt sich für jeden Bildpunkt mit vorhandener Disparität d die Tiefe Z berechnen. Mit den Werten für die Tiefe bzw. Disparitäten kann man ein Tiefen- / Disparitätsbild, d.h. ein Bild mit Intensitätswerten entsprechend der Tiefe generieren.

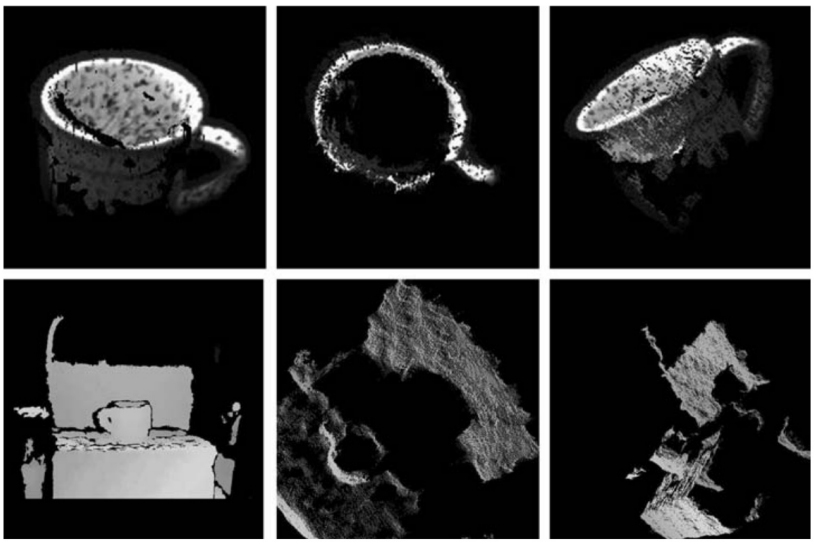


Abbildung 2.10: Tiefenbild einer Tasse [11]

2.1.7 Reprojektion

Für die Bildpunkte mit berechneten Disparitäten bzw. Tiefenwerten kann die Position im 3D-Weltkoordinatensystem bestimmt werden. Dazu bilden wir zunächst aus der Rotationsmatrix R und dem Translationsvektor t eine neue Matrix $[R|t]$, die der Rotationsmatrix erweitert um die Spalte t entspricht. Zusätzlich benötigen wir noch die Reprojektionsmatrix Q:

$$Q = \begin{pmatrix} 1 & 0 & 0 & -c_{u_L} \\ 0 & 1 & 0 & -c_{u_R} \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T & (c_{u_L} - c_{u_R})/T \end{pmatrix}$$

Wie im Abschnitt "Kameramodell" kurz erwähnt repräsentiert die Variable c den Kamerahauptpunkt. Somit sind c_{u_L} und c_{u_R} die u-Werte der jeweiligen Hauptpunkte der linken bzw. rechten Kamera. T ist der Abstand der beiden optischen Kamerazentren, wie im vorherigen Abschnitt.

Für einen gegebenen Bildpunkt in homogenen Koordinaten mit Disparitätswert $\begin{pmatrix} u & v & d & 1 \end{pmatrix}^T$ ergibt sich nun folgender 3D-Punkt:

$$[R|t] \cdot Q \cdot \begin{pmatrix} u & v & d & 1 \end{pmatrix}^T = \begin{pmatrix} X & Y & Z & W \end{pmatrix}^T$$

Die 3D-Koordinate berechnet sich wie folgt: $(X/W, Y/W, Z/W)$

Durch rückwärtsgewandter Rotation, Translation und Anwendung der intrinsischen Parameter wird aus einem Bildpunkt die Position des zugehörigen Objektpunktes erfasst.

2.1.8 Objekterkennung durch Clustering

Aus der Reprojektion aller Bildpunkte mit ermittelter Disparität lässt sich eine Punktwolke im 3D-Weltkoordinatensystem erstellen. Unterteilt man das Koordinatensystem nun in kleine Blöcke, könnte man zählen wie viele sich jeweils darin befinden.

Enthalten mehrere Blöcke nebeneinander viele Punkte, so liegt die Vermutung nahe, dass sich dort ein zusammenhängendes Objekt befindet. Diese Methode bietet uns somit eine grobe Objekterkennung.

Doch wie findet man algorithmisch die zusammenhängenden Blöcke? Dieses Thema wird allgemein auch Clusteranalyse genannt. Es wird dabei versucht aus einer Menge an Objekten Ähnlichkeitsstrukturen zu entdecken und Gruppen, auch Cluster oder Partitionen genannt, zu bilden. Ein bekanntes Problem in der Graphentheorie besteht darin, zusammenhängende Komponenten zu finden, die in unserem Fall aus verbundenen Zellen bestünden. Für die Lösung dieses Problems gibt es den so genannten Union-Find-Algorithmus, den ich im folgenden in Zusammenhang mit unserer Problematik kurz erläutern möchte.

Union-Find-Algorithmus:
Gegeben sei eine Menge S von Blöcken, die in Cluster aufgeteilt (partitioniert) werden soll: $S = \{x_0, \ldots, x_N\}$

S soll in disjunkte Teilmengen partitioniert werden, so dass ein Block x_i nach einer bestimmten Regel eindeutig einer Teilmenge zugeordnet werden kann. In unserem Fall bedeutet diese Regel, dass x_i mit mindestens einem Block aus seiner Teilmenge und mit keinem anderen aus den restlichen Teilmengen verbunden bzw. benachbart ist.

Es gibt für jede Partition einen Block, der Repräsentant dieser Teilmenge ist, sowie folgende 3 Grundoperationen:

- Init(S): Initialer Schritt, jedes $x \in S$ wird eine eigene Teilmenge und Repräsentant derselben
- Union(x_i, x_j): Vereinigt die beiden Partitionen, die jeweils x_i und x_j als Repräsentant haben, falls x_i und x_j benachbarte Blöcke sind. x_i wird Repräsentant der neuen vereinigten Teilmenge.

- Find(x_i): Gibt den Repräsentanten aus derjenigen Teilmenge zurück, in der sich Block x_i befindet.

Algorithmus Union-Find (Pseudocode):

```
1  Function Init(S):
2          for all x_i ∈ S
3                  x_i.parent = x_i
4
5  Function Find(x_i):
6          if x_i.parent = x_i
7                  return x_i
8          else
9                  return Find(x_i.parent)
10
11 Function Union(x_i, x_j):
12         rootI = Find(x_i)
13         rootJ = Find(x_j)
14         rootI.parent = rootJ
```

Mithilfe des Union-Find-Algorithmus bekommen wir Partitionen, in unserem Fall bestehend aus Blöcken im 3D-Koordinatensystem. Durch Fehlkorrespondenzen oder sehr kleinen bzw. schmalen Objekten kann es passieren, dass dadurch Partitionen zu groß werden oder mit anderen verschmelzen. Um das zu verhindern, legt man für eine Mindestanzahl an Punkten fest, die ein Block enthalten muss, um überhaupt relevant für die Partitionierung sein zu können. Je nach dem wie fein das Koordinatensystem in Blöcke unterteilt ist, kann ein Cluster auch erst ab einer Mindestgröße an zusammenhängenden Blöcken als gültig angesehen werden.

Für eine höhere Toleranz gegenüber fehlenden Blöcken, könnte man die Regel zur Vereinigung von Partitionen ändern und z.B. erlauben, dass ein Block auch 2 oder 3 Blocklängen entfernt liegen darf, um als benachbart zu zählen. Dadurch wäre ein Objekt geglätteter. Es muss jedoch anhand der Blockgröße variiert werden, was sich am besten eignet.

Wir haben erfahren, dass wir durch die Stereo-Vision, unter Berücksichtigung erwähnter Fehlerfaktoren, einen dreidimensionalen Überblick des von den Kameras eingefangenen Umfeldes erhalten. Außerdem wurde zum Schluss gezeigt, wie sich mit dem Blockclustering relativ simpel eine grobe Objekterkennung realisieren lässt. Bestimmte Objekte mit charakteristischer Form ließen sich damit bereits interpretieren, wie z.B. Pfeiler oder Bäume.

Für eine genauere Interpretation bedarf es allerdings komplexeren Algorithmen, worauf ich im nächsten Abschnitt „Mustererkennung“ eingehen werde.

2.2 Mustererkennung

Im Bereich des Maschinellen Lernens hat die Mustererkennung in den letzten Jahren einen immer größer werdenden Stellenwert eingenommen. Was dabei vor einigen Jahrzehnten lediglich theoretisch erforscht wurde, wird heutzutage schon häufig in der Praxis eingesetzt und ständig weiterentwickelt. Mithilfe der Mustererkennung ist es möglich, dass wir Sprachbefehle an unsere Geräte geben, Handschriften in digitalen Text umwandeln oder Gesichter in Kamerabildern erkennen können und vieles mehr.

Bei der Mustererkennung werden im Allgemeinen Merkmale aus Daten extrahiert, die in einer bestimmten Komposition Muster ergeben. Diese abstrakten Muster werden gespeichert, häufig in Form eines mathematischen Vektors oder einer Matrix. Bei der Untersuchung von anderen Daten, kann man damit Ähnlichkeiten erkennen, wenn die gespeicherten Muster wieder auftreten. Um ein möglichst allgemeingültige Muster zu erhalten, werden in der so genannten Trainings- oder Lernphase mehrere Beispieldaten als Eingabe genommen, die den gewünschten Mustern entsprechen. Der Algorithmus lernt mit jedem Beispiel bestimmte Muster bzw. optimiert sie. Die Untersuchung von Daten auf gelernte Muster wird Klassifikation genannt. Es wird versucht, den Eingabedaten einem oder mehrere Muster zuzuordnen, die am ähnlichsten sind. Schematisch ist dieser Prozess auch in Abb. 2.11 dargestellt.

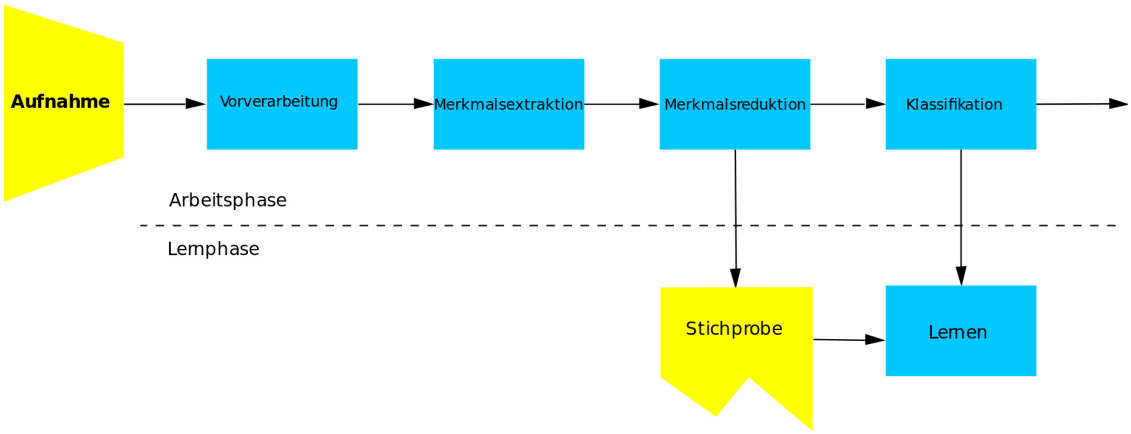


Abbildung 2.11: Schematischer Ablauf der Mustererkennung [13]

Da die Eingabedaten sehr unterschiedlich bezüglich des Formates sein können, werden sie häufig vor verarbeitet. Zum Beispiel werden alle digitalen

Bilder in die selbe Auflösung und ins selbe Farbformat gebracht. Als Zwischenschritt nach der Merkmalsgewinnung und vor der Klassifikation kann noch eine Merkmalsreduktion erfolgen, bei der unwesentliche Merkmale heraus gefiltert oder elementare Merkmale zu aussagekräftigeren zusammengesetzt werden.

Seit Beginn des 21. Jahrhunderts wurden analoge Foto- und Videoaufnahmen nach und nach durch digitale ersetzt. Im Zuge dessen wurden zunehmend Algorithmen für die bildbasierte Mustererkennung entwickelt. Im Folgenden gehe ich auf die theoretischen Grundlagen einiger dieser Algorithmen ein, die durch ihre Robustheit und Schnelligkeit große Popularität erreicht haben und in die Computer Vision Bibliothek OpenCV aufgenommen und von mir im praktischen Teil meiner Arbeit verwendet wurden.

2.2.1 Histograms of Oriented Gradients

Im Jahr 2005 veröffentlichten Navneet Dalal and Bill Triggs den *Histograms of Oriented Gradients* (HOG) Deskriptor [6], ein Objekterkennungsalgorithmus der zur Fußgängererkennung als Anwendungsfall entwickelt wurde.

Der Algorithmus verwendet eine bestimmte Art von Merkmalen, die so genannten HOG-Features sowie eine Support-Vector-Machine als Klassifikator.

HOG-Features

Die Merkmale bestehen aus Blöcken von *Histograms of Oriented Gradients* (HOG), zu deutsch: Histogrammen von gerichteten Gradienten.

Es werden für jeden Pixel des Eingangsbildes zunächst die Gradienten in x- und y-Richtung berechnet. Dazu werden diskrete Ableitungsmasken benutzt, die sich die benachbarten Pixel anschauen und daraus den Wert des Gradienten bestimmen. Diese werden auch von Kantenerkennungsalgorithmen in Bildern verwendet. Im Paper wurden verschiedene Ableitungsmasken wie 3×3 Sobel-Masken untersucht, aber es haben sich simple zentrierte 1D-Punkt Masken jeweils in x/y-Richtung am effektivsten herausgestellt.

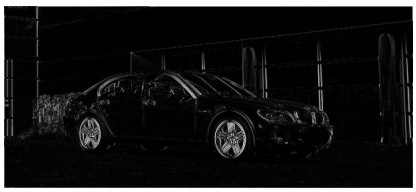
$$D_X = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}, D_Y = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}^T$$

Für einen Pixel berechnet sich dann der Gradient nach der folgenden Formel:

$$I_X * (x, y) = -1 \cdot I(x-1, y) + 0 \cdot I(x, y) + 1 \cdot I(x+1, y)$$

$$I_Y * (x, y) = -1 \cdot I(x, y-1) + 0 \cdot I(x, y) + 1 \cdot I(x, y+1)$$

(a) Eingangsbild



(b) Gradientenbild in x-Richtung

(c) Gradientenbild in y-Richtung

Abbildung 2.12: Eingangsbild mit zugehörigen Gradientenbilder in x- und y-Richtung

$I(x, y)$ bezeichnet den Pixelwert des Eingangsbildes I an der Stelle x, y und I_X*, I_Y* das Gradientenbild in x- bzw. y-Richtung. Als Pixelwert wird die Helligkeit genommen, die bei Graustufenbildern direkt aus dem Grauwert übernommen werden kann. Bei Farbbildern wird der Farbkanal mit dem größten normierten Wert verwendet.

Abbildung 2.12 zeigt ein Eingangsbild und die zugehörigen Gradientenbilder in x- und y-Richtung.

Aus den Gradientenbildern kann man außerdem eine Gradientenmatrix G mit absolutem Betrag sowie eine Gradientenrichtungsmatrix θ erhalten:

$$|G| = \sqrt{I_X^2 + I_Y^2}$$

$$\theta = arctan\frac{I_Y}{I_X}$$

Zur Berechnung der HOG-Features wird das Bild in Zellen und Blöcke unterteilt. Ein Block besteht aus mehreren Zellen. Im Paper hat sich eine Blockgröße von 18x18 Pixeln mit 3x3 enthaltenen Zellen der Größe 6x6 Pixel als beste Wahl für die Fußgängererkennung erwiesen.

Für jede Zelle wird ein Histogramm mit den Gradientenrichtungen von allen enthaltenen Pixeln erstellt, wobei die Richtungen vom $0° - 360°$ Raum auf eine feste Anzahl an Richtungskanälen abgebildet werden. Zum Beispiel umfasst Richtungskanal 1 den Bereich 0° bis 39°, Kanal 2 40° bis 79° usw.

Im Paper werden 9 Richtungskanäle verwendet. In Abb. 2.13 sind die Histogramme der gerichteten Gradienten aus dem Beispielbild veranschaulicht.

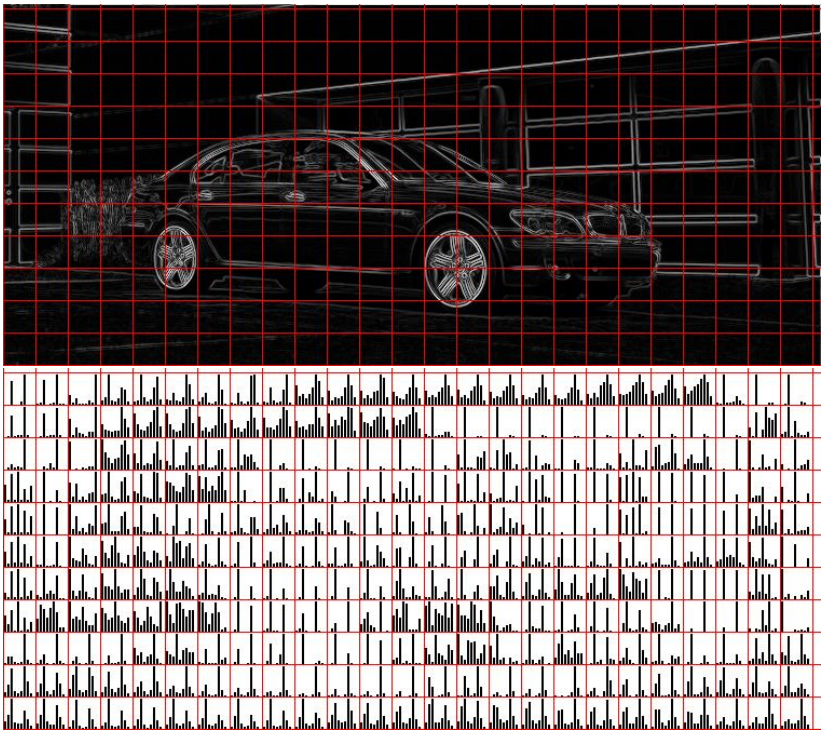


Abbildung 2.13: Histogramme der Bildzellen

Wie erwähnt werden Zellen in Blöcken zusammengefasst. Im Paper werden rechteckige (R-HOG) und kreisförmige (C-HOG) Blöcke vorgeschlagen. Die Zusammenfassung von Zellen zu Blöcken geschieht deshalb, weil die Gradienten je nach Helligkeit und Kontrast zwischen verschiedenen Zellen variieren können. Deshalb nimmt man größere Bereiche, die so genannten Blöcke und normalisiert deren Gradienten. Außerdem werden die Blöcke so angelegt, dass sie sich gegenseitig überlappen. Das heißt, eine Zelle gehört in der Regel zu mehreren Blöcken.

Die Normalisierung erfolgt nach einer der 3 folgenden Formeln, wobei v ein Vektor ist, der alle Histogramme eines Blocks enthält, sowie e eine kleine Konstante, deren Wert das Ergebnis kaum beeinflusst:

- L2-Norm: $f = \frac{v}{\sqrt{||v||_2^2 + e^2}}$
- L1-Norm: $f = \frac{v}{||v||_1 + e}$

- L1-Sqrt: $f = \sqrt{\frac{v}{||v||_1 + e}}$

Die einzelnen normalisierten Zellen der Blöcke stellen die HOG-Features dar. Alle normalisierten Blöcke zusammengefasst als Vektor ergeben zum Schluss den so genannten HOG-Deskriptor, der das Muster repräsentiert.

Klassifikation und Training

Für das Training und Klassifikation wird eine Support-Vector-Machine (SVM) genutzt.

Die Idee der Support-Vector-Machine geht auf die Arbeit von Vapnik und Chervonenkis aus dem Jahr 1974 zurück. Dabei werden so genannte Stützvektoren trainiert, die bei der Klassifikation unterschiedliche Muster voneinander trennt. Es wird versucht zwischen verschiedenen Mustern bzw. dem Muster und allem anderen durch die Stützvektoren eine Hyperebene aufzuspannen, die ein Muster von den anderen trennt.

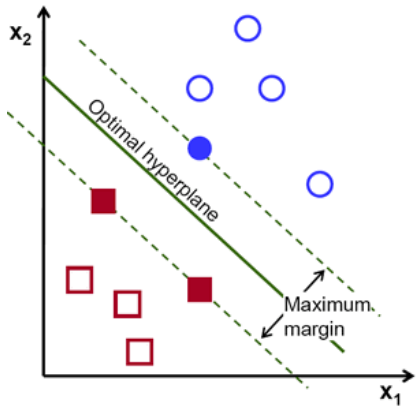


Abbildung 2.14: Trennung von zwei Klassen durch die Hyperebene einer SVM [14]

Für die Klassifikation werden die HOG-Features eines Eingangsbildes bzw. eines Ausschnitt als Suchfenster davon erstellt, die als Eingabevektor der SVM übergeben werden. Die SVM berechnet danach, auf Basis ihrer trainierten Parameter, auf welcher Seite von der Hyperbene aus gesehen, sich der Eingabevektor befindet und entscheidet demnach, ob das Bild zur Klasse des Musters gehört oder nicht.

Die SVM ist eine Funktion, die eine Klassifizierung nach folgender Formel berechnet:

$$f(x) = sgn(< w, x > +b)$$

Die Hyperebene ist definiert durch einen Vektor w und einem Bias b. Der Vektor w besteht aus gewichteten Komponenten, die sich aus den Trainingsbeispielen ergeben. Für den Eingabevektor x wird das Skalarprodukt mit w gebildet und der Bias hinzu addiert. Abschließend sorgt die *sgn*-Funktion für eine Abbildung auf -1 oder $+1$, Treffer oder kein Treffer. Für Klassifikationsprobleme mit nichtlinearen Daten kann anstelle des Skalarprodukts auch eine andere so genannte Kernelfunktion eingesetzt werden.

Im Falle der HOG-Features besteht der Vektor w aus den Werten der Histogramme von Gradientenrichtungen. Je nach dem wie entscheidend ein Histogramm für die Klassifizierung ist, desto höher ist das zugehörige Gewicht. Alle Histogramme mit einem Gewicht größer 0 stellen einen Stützvektor dar, der der Hyperebene definiert.

Das Trainieren der SVM-Funktion wird durch Anpassung der Hyperebene nach folgender Optimierung erreicht:

- Minimiere die quadratische Norm $\frac{1}{2}||w||_2^2$ durch Änderung von w und b,
- so dass $f(x_i) \cdot (< w, x_i > +b) \geq 1$ für alle $1 \leq i \leq m$ gilt.

Dabei ist x_i eins von m Trainingsbeispielen. Da in der Regel nicht alle HOG-Features der Trainingsbeispiele exakt von der Hyperebene getrennt werden können, fügt man dem zu minimierenden Term $\frac{1}{2}||w||_2^2$ noch so genannte Schlupfvariablen ξ_i hinzu, die angeben, wie stark ein Bild nicht eindeutig oder falsch von der Hyperebene getrennt werden konnte. Ziel ist es, auch diese Schlupfvariablen zu minimieren.

2.2.2 Deformable Part Models

Als Erweiterung zu Dalal und Triggs HOG-Deskriptoren hat P.F. Felzenszwalb et al. im Jahr 2010 den *Deformable Part Models* (DPM) Detektor [7] veröffentlicht.

DPM-Features

Die Innovation des DPM-Detektors ist die Erweiterung des HOG-Deskriptors um ein Sternmodell, dass aus mehreren HOG-Deskriptoren besteht. Es gibt einen so genannten Root-Filter, der dem normalen HOG-Deskriptor des Musters entspricht. Zusätzlich werden Part-Filter trainiert, die einzelne Teile des Musters repräsentieren, siehe Abb. 2.15.

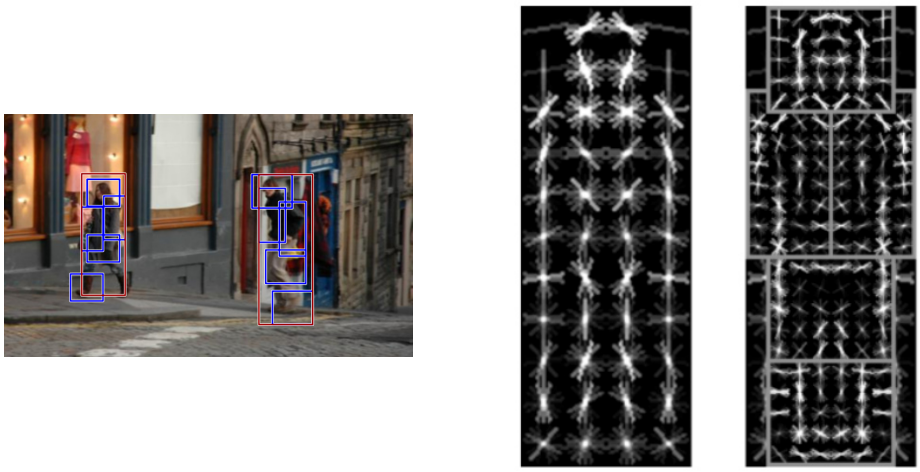


Abbildung 2.15: Root- und Part-Filter von Personen [7]

Das Sternmodell besteht also im Zentrum aus dem Root-Filter und außen weitere Part-Filter. Außerdem enthält das Modell noch Informationen über die Positionierung der Part-Filter. Ein Objekt das im Bild eine andere Lage einnimmt als in den Bildern der Trainingsmenge, kann trotzdem identifiziert werden, wenn die einzelnen Part-Filter ein positives Ergebnis liefern. Dabei wird ein so genannter Verformungskosten-Wert für einen Part-Filter berechnet, der die Abweichung der Positionierung im Vergleich zur originalen Position im gelernten Sternmodell angibt. Abbildung 2.16 zeigt ein Beispiel, bei dem ein Fahrrad erkannt wird, obwohl es in anderer Lage trainiert wurde.

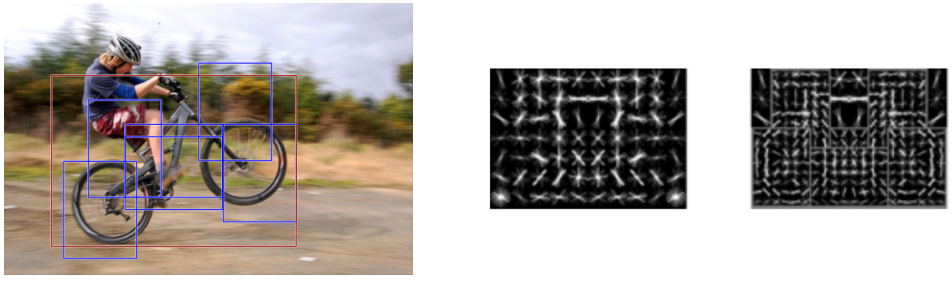


Abbildung 2.16: Erkennung eines Fahrrads anhand der Part-Filter [7]

Training und Klassifikation

Das Training und die Klassifikation geschieht beim DPM-Detektor analog zum HOG-Deskriptor durch Berechnung einer trainieren Support-Vector-Machine. Der einzige Unterschied ist, dass die Part-Filter verschoben sein

können, um Vergleich zum trainierten Modell. Deshalb muss bei der Klassifikation, ein Part-Filter auf verschiedene Positionen untersucht werden.

Beim Training wird der Suchbereich in dem sich ein bestimmter Part-Filter befinden kann zusätzlich ermittelt. Siehe dazu Abbildung 2.17.

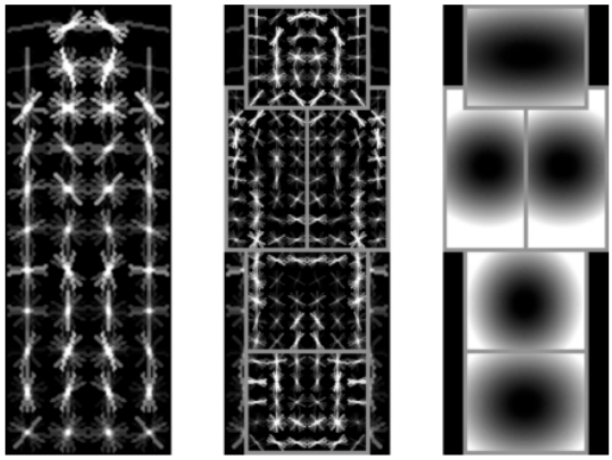


Abbildung 2.17: Muster einer Person mit Verschiebungsbereich der Part-Filter (rechts) [7]

Die Berechnung der SVM-Klassifikationsfunktion des HOG-Deskriptors wird um Variablen für die entsprechenden Suchbereiche erweitert:

$$f(x) = sgn(max_{z \in Z(x)} < w, \theta(x, z) > +b)$$

Der Vektor w enthält in diesem Fall alle Informationen des Sternmodells. Die Menge Z ist hierbei die Menge der möglichen Positionierungen für einen Part-Filter. Es wird die bestmögliche Positionierung z der Part-Filter gesucht, so dass $< w, \theta(x, z) >$ maximiert wird. $\theta(x, z)$ bezeichnet die Konkatenation aus den HOG-Features und der möglichen Part-Filter-Positionierung.

2.2.3 Viola-Jones Algorithmus

Der Viola-Jones Algorithmus stammt von Paul Viola und Michael Jones und wurde 2001 in einem wissenschaftlichen Paper [5] veröffentlicht.

Als Demonstrationsbeispiel wird im Paper die Erkennung von Gesichtern gezeigt. Aber es lassen sich auch beliebige andere Objekte erkennen, wenn der Algorithmus darauf trainiert wurde.

Haar-like Features

Aus den Bildern als Eingangsdaten werden so genannte Haar-like Features als Merkmale extrahiert. Für diese Merkmale werden rechteckige Bereiche im Bild zusammengefasst. Diese Bereiche sind weiter unterteilt in 2, 3 oder 4 Rechtecke nach folgenden Arten:

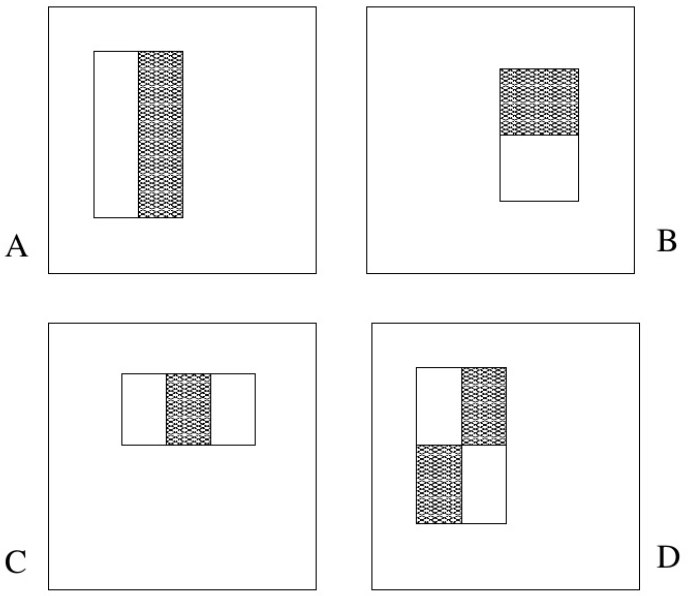


Abbildung 2.18: Arten der Haar-like Features [5]

Die Positionierung, Skalierung und Rotation (jeweils um 90 Grad) der Merkmale ist dabei beliebig. Es können aber auch andere Kombinationen von hellen und dunklen Rechtecken verwendet werden.

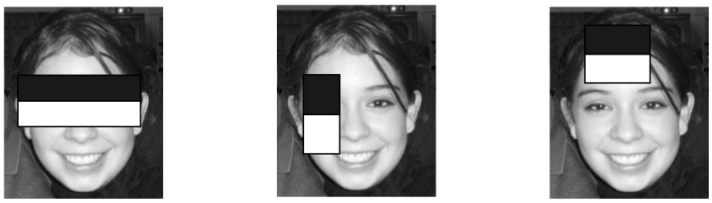


Abbildung 2.19: Beliebige Skalierung und Positionierung der Haar-like Features

Für alle Werte der Pixel, z.B. normierte Helligkeitswerte, eines hellen bzw.

dunklen Rechtecks wird der Durchschnitt gebildet. Anschließend wird die Differenz aus Werten der hellen minus dunklen Rechtecke gebildet.

Integralbild

Um die Berechnung der Rechtecke eines Merkmals zu beschleunigen, kann man einmalig pro Eingangsbild ein zugehöriges Integralbild erzeugen. Es enthält an der Position i,j alle Pixelwerte aus dem Originalbild aufsummiert, den das Rechteck von Position 0,0 bis i,j aufspannt.

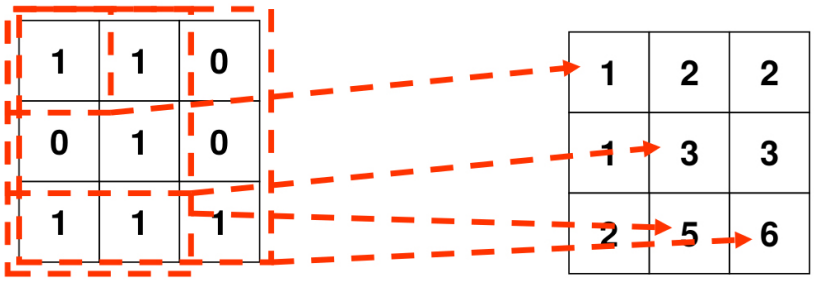


Abbildung 2.20: Schnellere Berechnung durch Integralbild

Damit lässt sich ein beliebiges Rechteck D innerhalb des Bildes in konstanter Zeit nach folgender Formel auswerten:

$$D = P4 - P2 - P3 + P1$$

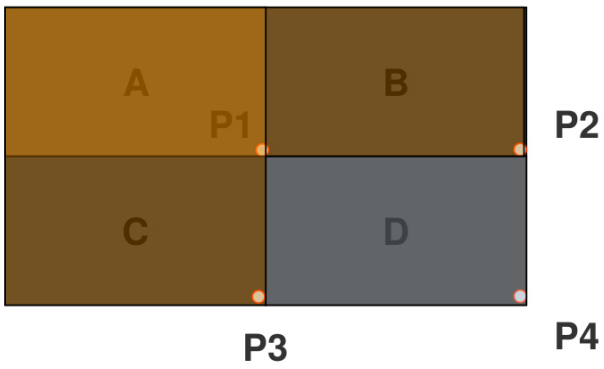


Abbildung 2.21: Rechteck D kann schnell berechnet werden

Klassifikation

Für jedes Haar-like Feature j wird ein Score f_j aus der Differenz der hellen und dunklen Rechtecke berechnet. Zusammen mit einem festgelegten Schwellwert θ_j und einer Polarität $p_j \in \{-1, 1\}$ ergibt sich ein so genannter schwacher Klassifikator als Funktionswert $h_j(x)$, wobei x das Eingangsbild ist:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

Alle schwachen Klassifikatoren und somit alle Merkmale eines Bildes zusammen bilden einen starken Klassifikator, der das gelernte Muster darstellt. Er berechnet sich aus der Linearkombination der mit α_j gewichteten, schwachen Klassifikatoren und bildet das Ergebnis auf Treffer "1" oder nicht Treffer "0" ab:

$$h(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

Da das Muster in einem Bild in verschiedenen Größen auftreten kann, wird das Muster in einer bestimmten Größe z.B. 24x24 Pixel trainiert und in skalierten Varianten des Bildes danach gesucht.

Training

Eine noch ungeklärte Frage ist, die Auswahl an Merkmalen für ein gewünschtes Muster. Für ein 24x24 großes Muster können sich mit den 3 Arten aus Abb. 2.18 bereits über 160.000 verschiedene Merkmale ergeben. Für eine Berechnung in Echtzeit sind das zu viele und viele davon sind nicht aussagekräftig, weil sie z.B. zu groß oder zu klein sind, um einen allgemeingültigen Wert zu liefern. Wenn wir ein Bild von einem Gesicht betrachten, können wir uns intuitiv vorstellen, welche Merkmale sich eigenen. Zum Beispiel gibt es bei den Augen einen Helligkeitsunterschied zur Stirn oder den Wangen, siehe Abb. 2.22.

Abbildung 2.22: Haar-like Features im Gesicht [5]

Um zu ermitteln, welche Merkmale mit bestimmter Position und Größe am aussagekräftigsten sind, gibt es ein automatisiertes Verfahren, das auf dem Adaboost-Algorithmus basiert.

Es wählt aus den vielen möglichen Merkmalen diejenigen aus, die die beste Unterscheidung zwischen einer Menge an Positiv- und Negativbeispielen von Mustern liefern und trainiert gleichzeitig den Klassifikator. Die Grundidee ist, dass man nacheinander einen schwachen Klassifikator aus den vielen möglichen hinzunimmt und die Werte für sein Gewicht, Schwellwert und Polarität so wählt, dass er so wenig wie möglich Bilder der Trainingsmenge falsch klassifiziert durch eine Fehlerkostenfunktion. Iterativ kommen nun neue schwache Klassifikatoren hinzu und die Gewichte der vorherigen angepasst, so dass für viele ungeeignete Klassifikatoren ein Nullgewicht entsteht und sie somit vernachlässigt werden können. Für weitere Details hierzu verweise ich auf die Seite 3 und 4 im Paper [5].

Als weitere Optimierung kann man auch mehrere Klassifikatoren verwenden, die als Kaskade hintereinander ausgewertet werden. D.h. anstatt nur einen komplexen Klassifikator zu haben, der direkt entscheidet, ob es sich um das Muster handelt oder nicht, kann man mehrere einfachere Klassifikatoren zu einem Entscheidungsbaum verknüpfen, siehe Abb. 2.23.

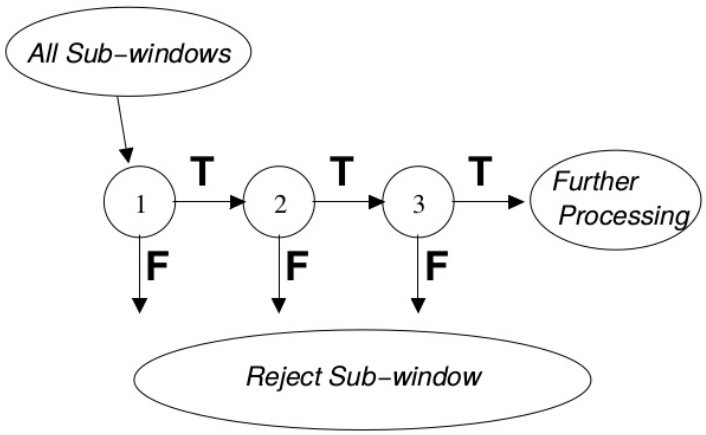


Abbildung 2.23: Kaskade von Klassifikatoren [5]

In der ersten Stufe kann z.B. nur ein bestimmtes Merkmal im Bild untersucht werden, dass für fast alle der Positivbeispiele zutrifft, aber wodurch schon viele Negativbeispiele herausfallen. Die Negativentscheidung ist in diesem Fall wesentlich schneller zu berechnen als mit einen komplexen Klassifikator. Erst wenn das Bild für alle Stufen positiv ausgewertet wird, gilt es als Treffer für das trainierte Muster.

HOG- und LBP-Features

In Open Source Computer Vision Bibliothek (OpenCV) ist neben der normalen Viola-Jones-Variante mit Haar-like Features auch die Möglichkeit implementiert HOG- und LBP-Features stattdessen zu verwenden.

HOG-Features wurden bereits näher erläutert. Deshalb erläutere ich an dieser Stelle noch Details zu den LBP-Features.

LBP-Features (Local binary patterns) sind den HOG-Features sehr ähnlich. Sie wurden bereits 1994 von T. Ojala et al. in [15] erstmalig erwähnt.

Das Bild wird dabei in Zellen z.B. 24x24 Pixel unterteilt. Für jeden Pixel der Zelle werden alle 8 Nachbarpixel in einer festen Reihenfolge betrachtet. Wenn der Wert des Nachbars größer oder gleich ist, dann wird dieser mit 1 markiert, ansonsten mit 0. Daraus ergibt sich eine 8-Bit Binärzahl.

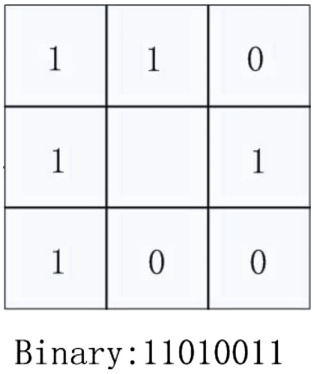


Abbildung 2.24: Binärzahl eines Pixels

Aus den ermittelten Binärzahlen wird ein Histogramm für jede Zelle erstellt. Alle Histogramme werden zu einem Vektor konkateniert, der das LBP-Feature darstellt. Optional können die Histogramme auch wie bei den HOG-Features normalisiert werden.

3 Implementierung der Fahrzeugerkennung

Nach dem ich die theoretischen Grundlagen erklärt habe, beschreibe ich in diesem Kapitel die praktische Umsetzung meiner Arbeit im AutoNOMOS-Projekt. Ich erläutere die Softwarearchitektur, den Ablauf der Datenverarbeitung sowie Implementierungsdetails zu den eingesetzten Algorithmen.

3.1 Softwarearchitektur

Das AutoNOMOS-Projekt nutzt das *Open Robot Control Software*-Framework (OROCOS).

Es bietet eine Infrastruktur bestehend aus Modulen, die vom Typ PROGRAM, DATA oder PLUGIN sein können. PROGRAM-Module sind ausführbare Programme, wie z.B. ein GUI-Programm zum Abspielen von aufgenommenen Fahrten mit dem autonomen Auto in Form von Log-Dateien. Ein PLUGIN-Modul kann innerhalb eines PROGRAM-Moduls de-/aktiviert werden und über so genannte Ports Daten mit anderen Modulen austauschen. DATA-Module repräsentieren abstrakte Datentypen, die von Modulen verwendet werden können.

Für diese Masterarbeit wurden PLUGIN- und DATA-Module entwickelt, die ein allgemeines Framework zur Objektlokalisierung mit Stereo-Vision und Objektidentifizierung mit den im theoretischen Teil erwähnten Mustererkennungsalgorithmen bieten.

Im Folgenden eine kurze Erläuterung zu den jeweiligen Modulen:

- StereoDetection: Führt ein Clustering mit den 3D-Punkten aus der Stereo-Vision durch, um Objekte zu lokalisieren.
- StereoDetectionData: Enthält Infomationen über die lokalisierten Objekte. Dazu gehören Abstand zum Mittelpunkt des autonomen Autos, reale Breite und Höhe sowie die Bildmaße der Objekte als Rechtecke.
- CascadeDetection / CascadeDetectionGPU: Mustererkennung mit OpenCV's Viola-Jones basierter Implementierung für Haar-, LBP- und HOG-Features, mit und ohne GPU-Unterstützung. GPU-Unterstützung ist derzeit in OpenCV nur für Haar- und LBP-Features vorhanden.

- HOGDescriptorDetection / HOGDescriptorDetectionGPU: Mustererkennung mit OpenCV's Dalal-Triggs basierter Implementierung für HOG-Deskriptoren, mit und ohne GPU-Unterstützung.
- DPMDetection / DPMDetectionFFLD: Mustererkennung mit OpenCV's Felzenszwalb basierter Implementierung für Deformable Part Models bzw. einer optimierten Implementierung mit der FFLD-Bibliothek.
- DetectedObject: Enthält ein Rechteck als Bereich im Bild, in dem sich das erkannte Objekt befindet sowie einen Trefferwert, der angibt mit welcher Wahrscheinlichkeit es sich um das gesuchte Objekt handelt.

Die Module lassen sich unabhängig voneinander verwenden. Für den konkreten Fall der Erkennung von Fahrzeugen wurden weitere Module implementiert, welche die allgemein nutzbaren Module verknüpfen, wie im Folgenden kurz erläutert wird.

- AbstractStereoCarDetection: Abstraktes Modul, das eine grobe Objektlokalisierung mit dem StereoDetection-Modul durchführt und mit einem nicht spezifizierten Mustererkennungsalgorithmus die Fahrzeugerkennung. Außerdem visualisiert es gefundene gefundene Treffer.
- HOGDescriptorStereoCarDetection: Erweitert das AbstractStereoCarDetection-Modul um die spezifische Mustererkennung mit der HOGDescriptorDetection bzw. HOGDescriptorDetectionGPU.
- DPMStereoCarDetection: Erweitert das AbstractStereoCarDetection-Modul um die spezifische Mustererkennung mit der DPMDetection bzw. DPMDetectionFFLD.
- CascadeHOGStereoCarDetection / CascadeHaarStereoCarDetection / CascadeLBPStereoCarDetection: Erweitert das AbstractStereoCarDetection-Modul um die spezifische Mustererkennung mit der CascadeDetection(GPU) für HOG-, Haar- bzw. LBP-Features.

3.2 Datenverarbeitung

3.2.1 Stereo-Vision

Aus den beiden Kamerabildern als Eingangsdaten wird mithilfe der bereits im Projekt existierenden Stereo-Vision-Module StereoCameraTask, StereoMatcherTask und StereoDataComposer ein Disparitätsbild und eine 3D-Punktwolke erzeugt.

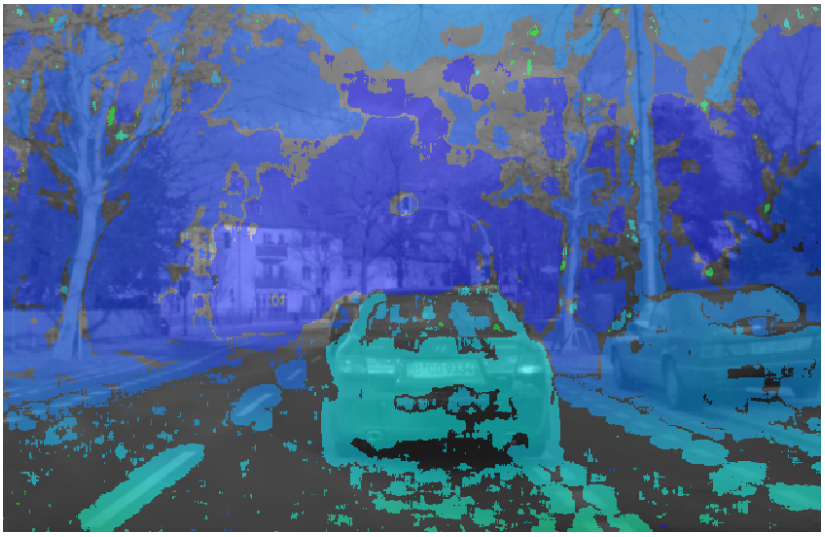


Abbildung 3.1: Disparitätsbild

Das StereoDetection-Modul erhält diese Daten über den Eingangsport und legt ein Gitternetz das OccupancyGrid an, dass in einer OcclusionMatrix festhält wie viele Punkte jeweils in einer Gitterzelle vorhanden sind. Um Rechenzeit zu sparen, wird das Gitternetz als eine Schicht mit festgelegter Höhe, Breite und Tiefe angelegt.

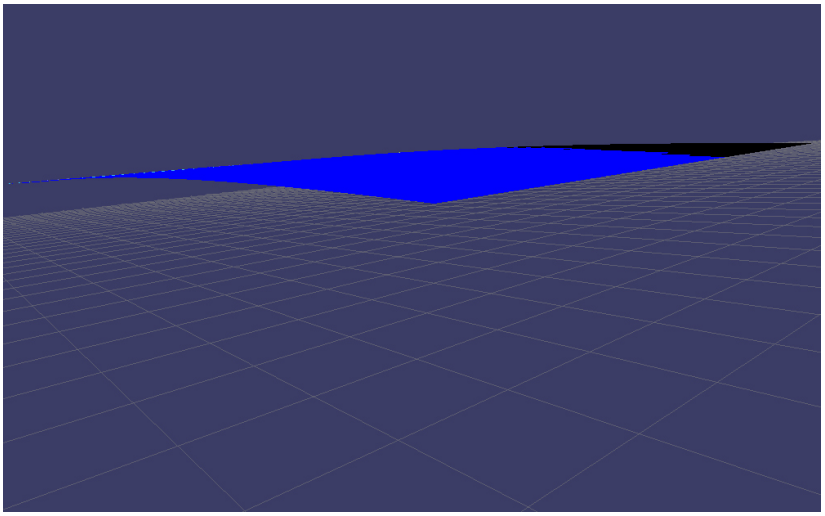


Abbildung 3.2: Gitternetz mit Schicht für das Clustering

Im nächsten Schritt wird das Clustering mithilfe der GridCellClustering-Klasse durchgeführt, welche den Union-Find-Algorithmus anwendet. Occu-

pancyGrid, OcclusionMatrix und GridCellClustering befinden sich im Unterprojekt viskos-cam.

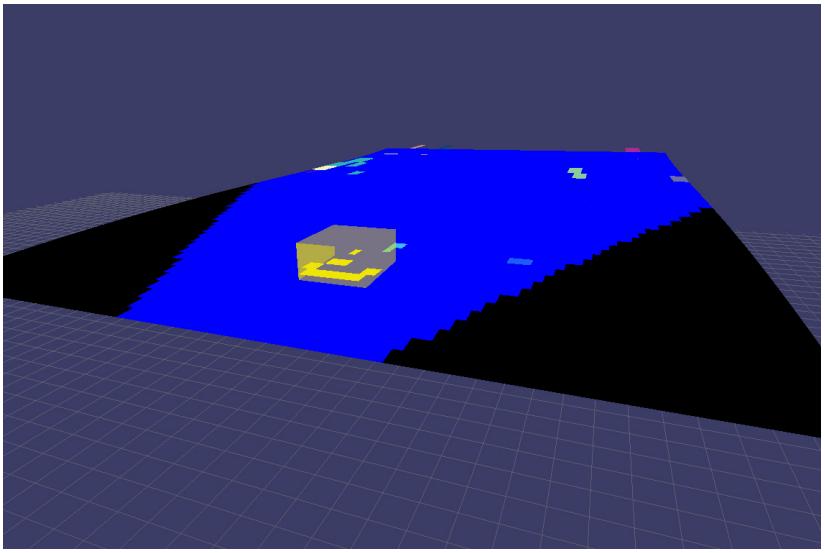


Abbildung 3.3: Clusteranalyse

Aus den erhaltenen Clustern lassen sich zum Schluss die vermuteten Objekte gewinnen. Es wird der reale Abstand zum Mittelpunkt des autonomen Autos in Metern erfasst. Außerdem wird ein Rechteck im Bild durch Projektion der 3D-Punkte eines Clusters auf die Bildebene berechnet. Alle Informationen eines Clusters zusammen werden in einem StereoBoundingBox-Struct und alle gefundenen StereoBoundingBox-Structs als Vektor im StereoDetectionData-Modul.

3.2.2 Mustererkennung

Unabhängig vom eingesetztem Verfahren zur Mustererkennung gibt es das gemeinsam benutztes Modul AbstractStereoCarDetection, das die Vorauswahl an Objektlokalisierungen in Form der StereoBoundingBox-Structs über den Eingangsport aufnimmt. Außerdem erhält es noch das linke Kamerabild, da die Projektion der Stereo-Vision-Cluster auch auf dieses erfolgt.

Abbildung 3.4: Vorauswahl durch Stereo-Vision

Um den Suchraum nach Fahrzeugrückansichtenmustern auf die bereits lokalisierten Objekte einzuschränken werden die übrigen Bildbereiche eingeschwärzt und das Bild auf die relevanten Bereiche verkleinert.

Abbildung 3.5: Vorauswahl reduziert das Eingangsbild auf relevante Bereiche

Im Anschluss erfolgt die jeweilige Mustererkennung mit dem entsprechenden Algorithmus. Man erhält letztendlich die Bereiche im Bild, die als Fahrzeugrückansichten identifiziert wurden.

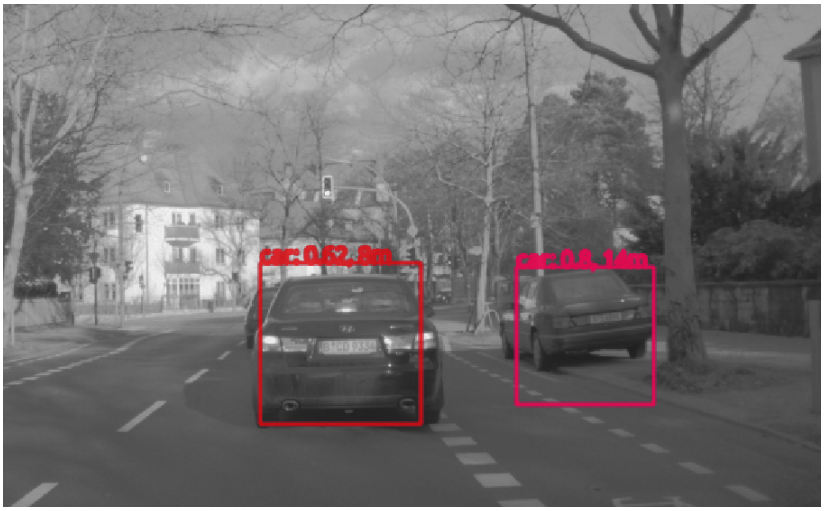


Abbildung 3.6: Klassifikation als Fahrzeug mit Trefferwert und Distanz

Bis auf die OpenCV-GPU-Varianten liefern die Mustererkennungsalgorithmen zusätzlich einen Trefferwert für das jeweilige Fahrzeug. Durch das Stereo-Vision-Clustering ist außerdem der Abstand zu den gefundenen Fahrzeugen bekannt. Diese Informationen werden oberhalb der rechteckigen Markierung um das Fahrzeug angezeigt.

3.3 Implementierungsdetails

Für Module zur Erkennung von Fahrzeugrückansichten mit Stereo-Vision als Vorauswahl wurden folgende grundlegende Optionen implementiert:

- StereoUse: An-/Abschalten der Stereo-Vision als Vorauswahl. Bei Deaktivierung wird die Mustererkennung auf dem gesamten Kamerabild ausgeführt. Dadurch lassen sich auch Fahrzeuge erkennen, die außerhalb des Stereo-Vision-Gitternetzes liegen. Allerdings führt dies zu einer erheblich höheren Laufzeit und die Informationen zum Abstand werden nicht berechnet.

- StereoBoundingboxMargin: Vergrößerung der Rechtecke in Pixel, die aus dem Stereo-Vision-Clustering ermittelt wurden. Da die Rechtecke der Objekte eventuell etwas kleiner ausfallen können als die tatsächliche Größe der Objekte im Bild, werden die Rechtecke noch um einige Pixel vergrößert.

- SkipImagesToNextComputation: Anzahl der Eingangsbilder, die bis zur nächsten Berechnung übersprungen werden sollen. Um die Geschwindigkeit zu erhöhen, können einige Bilder ausgelassen werden bis eine erneute Objekterkennung ausgeführt wird.
- ShowFramerate: Berechnet und zeigt die Anzahl der verarbeiteten Bilder pro Sekunde an.
- HighContrastUse: An-/Abschalten des "Hohen-Kontrast-Modus". Für die Mustererkennung wird der Kontrast des Kamerabildes stark erhöht, so dass sich Objekte stärker von der Umgebung abheben.

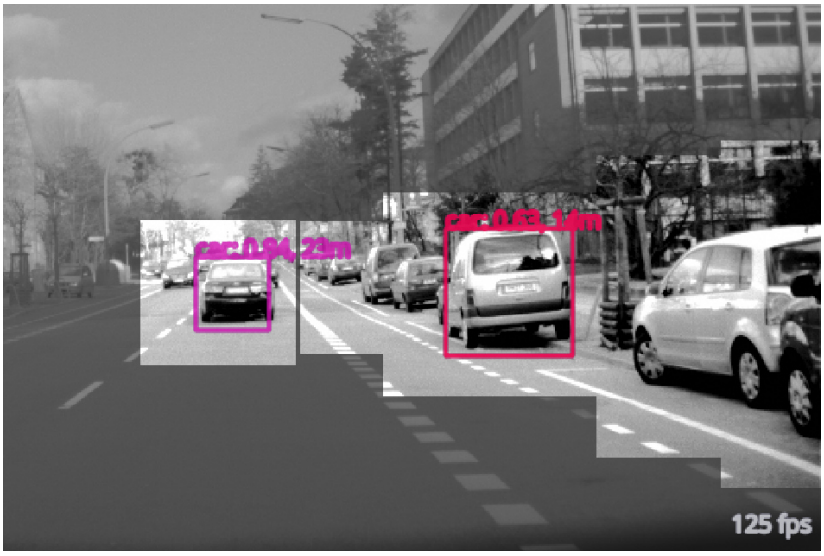


Abbildung 3.7: Hoher Kontrast Modus

- HighContrast_Brightness: Helligkeitswert für den "Hohen-Kontrast-Modus"
- HighContrast_Contrast: Kontrastwert für den "Hohen-Kontrast-Modus"
- NomalizeImage: Normalisiert für die Mustererkennung die Intensität im gesamten Kamerabild durch einen Histogrammausgleich. Das führt zu Kontrastverbesserungen unabhängig vom "Hohen-Kontrast-Modus".

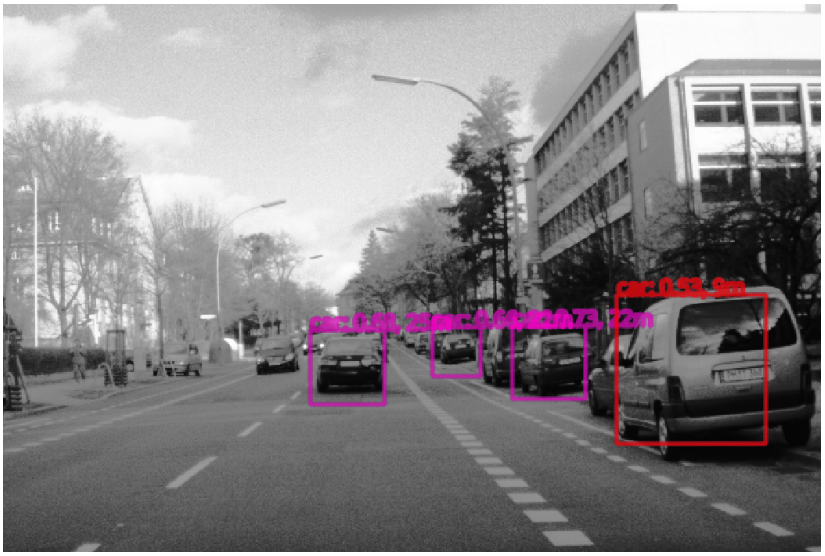


Abbildung 3.8: Kontrast-Normalisierung

Stereo-Vision Gitternetz

Das Gitternetz für die Objekterkennung der Stereo-Vision kann aus mehreren Schichten bestehen. Eine Schicht ist vom autonomen Fahrzeug aus nach vorn gerichtet. Es können x-/y-Koordinaten für die Länge und Breite des Gitternetzes angegeben werden sowie die Höhe. Die Gitterzellen werden anhand der Werte für Länge und Breite geteilt durch eine festlegbare Anzahl in x- und y-Richtung festgelegt. Somit ergibt sich für die Gitterzeilen- und Gitterspaltengröße folgende Formel:

$$Gitterzeilengröße = ceil((maxX - minX)/xSize)$$
$$Gitterspaltengre = ceil((maxY - minY)/ySize)$$

Dabei sind $minX$ und $minY$ die Startpositionen bzw. $maxX$ und $maxY$ die Endpositionen in x- und y-Richtung in Metern ausgehend von der Mitte der Frontachse des Autos. $xSize$ und $ySize$ sind die Anzahl der Zeilen bzw. Spalten des Gitternetzes. In z-Richtung gibt es pro Schicht nur eine Zelle.

Mustererkennung

Für die Mustererkennungsalgorithmen wurden Fahrzeugrückansichten trainiert. Dazu wurde eine Trainingsmenge bestehend aus ca. 1000 Positivbei-


Abbildung 3.9: Positiv- und Negativbeispiel für das Training

spielen und 1500 Negativbeispielen angelegt. Die Bilder sind in den Formaten 64x64, 128x128 und 256x256 Pixeln vorhanden.

Als Positivbeispiele wurden Fahrzeugrückansichten aus Log-Dateien extrahiert, die mit dem autonomen Auto aufgenommen wurden, sowie zahlreiche Bilder aus Internetrecherchen. Negativbeispiele sind Ausschnitte von Straßen, Gebäuden, Verkehrszeichen, Bäumen etc., die aus Log-Dateien stammen.

HOG-Deskriptor

Für das Trainieren des HOG-Deskriptors existiert ein Open-Source-Projekt [16], dass einen OpenCV-kompatiblen Klassifikator erstellt.

Es lassen sich folgende Trainings-Parameter einstellen:

- win_size: Suchfenstergröße in Pixeln, Standardgröße: 64x128 für die Fußgängererkennung
- cell_size: Zellgröße in Pixeln. Standardgröße: 8x8
- block_size: Blockgröße in Pixeln. Es werden nur rechteckige Blöcke unterstützt (R-HOG). Die Blockgröße muss ganzzahlig durch die Zellgröße dividierbar sein. Standardgröße: 16x16
- block_stride: Blockschrittweite, Standardgröße: 8x8. Legt fest, wie stark sich Blöcke überlappen.
- nbins: Anzahl der Richtungskanäle, Standard: 9. Legt fest, wie fein Richtungen in den Histogrammen unterteilt werden.

- win_sigma: Gaussian Smoothing (Weichzeichnen), Standard: -1 (deaktiviert). Legt fest, wie stark das Bild weichgezeichnet wird bevor die Ableitungsmaske darauf angewendet wird.

  Dalal und Triggs kamen zur Erkenntnis, dass ohne Weichzeichnung bessere Resultate erzielt werden.

- threshold_L2hys: Verkleinerung für die Histogrammwerte bei der Blocknormalisierung, Standard: 0.2

- gamma_correction: Boolean, ob eine Gammakorrektur des Bildes vorgenommen werden soll oder nicht, Standard: True

- nlevels: Maximale Anzahl der Suchfenstervergrößerungen, Standard: 64

Die Trainingsbilder wurden anstatt der höchsten vorhanden Auflösung von 256x256 Pixeln nur im Format 64x64 Pixel verwendet, da das Suchfenster von dieser Auflösung ausgehend nur vergrößert und nicht verkleinert wird.

Für die Erkennung können zur Laufzeit weitere Parameter angeben werden:

- hit_threshold: Schwellwert für den Abstand zwischen den HOG-Features und der SVM-Hyperebene. Je größer, desto besser ist die Trefferwahrscheinlichkeit für ein Muster. Standard: 0

- win_stride: Fensterschrittweite, Standardgröße: leer (entspricht Blockschrittweite). Legt fest, wie weit das Suchfenster in jedem Schritt verschoben wird. Muss ein Vielfaches der Blockschrittweite sein.

- scale: Skalierungsfaktor des Suchfensters, Standard: 1.05

- group_threshold: Gruppierungsschwellwert für ähnliche Treffer, Standard: 2. Erkennung liefert bei group_threshold $= 0$ sehr viele Treffer für dasselbe Objekt. group_threshold > 0 gibt an, wie viele Treffer mit ähnlicher Größe und Position vorhanden sein müssen, damit die Treffer gruppiert und als ein Ergebnis zurückgegeben werden.

- useMeanshiftGrouping: Boolean, ob statt mit Non-maximum suppression die Gruppierung per Mean-Shift erfolgen soll. Standard: False

DPM-Detektor

Das Training für den DPM-Detektor wurde auf 2 verschiedene Arten ausgeführt. Für die OpenCV-Implementierung gibt es eine Trainingsumgebung in MATLAB-Code, die von P.F. Felzenszwalb selbst für die PASCAL-Challenge, einem Bildklassifikationswettbewerb, entwickelt wurde. Diese erstellt einen Klassifikator der in ein OpenCV-kompatibles Format umgewandelt werden kann.

Für die FFLD-Implementierung wird direkt von der Bibliothek selbst ein Trainingsprogramm bereitgestellt.

Die MATLAB-Trainingsumgebung bietet keine einstellbaren Parameter an. Für die FFLD-Variante können für das Training und Ausführung des Klassifikators folgende Parameter eingestellt werden:

- overlapThreshold: Schwellwert, der angibt, bis zu wie viel Prozent ein Treffer einen anderen überlappen darf. Standard: 0.5
- intervall: Anzahl der Skalierungsstufen der HOG-Pyramide. Standard: 5
- padding: Anzahl zusätzlicher Nullen in HOG-Zellen. Standard: 6

Zur Laufzeit können in der OpenCV-Variante lediglich folgende Parameter angegeben werden:

- overlapThreshold: Schwellwert, der angibt, bis zu wie viel Prozent ein Treffer einen anderen überlappen darf. Standard: 0.5
- numThreads: Anzahl der parallel ausführbaren Threads, wenn das System dies unterstützt.

Viola-Jones Algorithmus

OpenCV bietet mit dem Programm "opencv_traincascade" eine Trainingsumgebung an, die umfassend in der OpenCV-Dokumentation beschrieben ist. Die Klassifikatoren für Haar-, LBP- und HOG-Features wurden alle damit trainiert.

Zur Laufzeit lassen sich folgende Parameter angeben:

- scaleFactor: Skalierungsfaktor des Suchfensters, Standard: 1.1
- minNeighbors: Entspricht dem group_threshold des HOG-Deskriptors. Standard: 3

4 Experimentelle Auswertung

Alle Tests wurden auf einem PC mit einem Intel i7-4710HQ 64-Bit Prozessor, 16 GByte RAM und einer NVIDIA GeForce GTX 860M Grafikkarte durchgeführt. Das verwendete Betriebssystem ist elementary OS 0.3 in der 64-Bit Variante.

Die grobe Objekterkennung durch Stereo-Vision ist fest im AutoNOMOS-Projekt implementiert. Es wurde dafür keine externe Testumgebung entwickelt. Dennoch gehe ich im Abschnitt Ergebnisse noch näher darauf ein.

Für die Auswertung der Musterkennungsalgorithmen wurde eine Testmenge von rund 150 Bildern mit Positiv- und Negativbeispielen angelegt. Alle Bilder sind graustufig und haben unterschiedliche Größen, aber in der Regel 750x500 Pixel, so dass sie den Bildgrößen, der Kameraaufnahmen des autonomen Autos entsprechen. Die Testmenge ist wie die Trainingsmenge gemischt mit Bildern aus Internetrecherchen sowie aus AutoNOMOS-Log-Dateien.

Alle Fahrzeugrückansichten sind annotiert mit 2 Bounding-Boxen. Eine kleinere Bounding-Box befindet sich innerhalb des Bereichs der Fahrzeugrückansicht und die andere etwas großzügiger außerhalb, siehe Abb. 4.1.

Abbildung 4.1: Bild aus Testmenge mit Annotationen

Um zu evaluieren, ob der Algorithmus das Fahrzeug erkannt hat, gilt folgende Regel. Für eine korrekte Klassifizierung muss ein Treffer die innere Bounding-Box enthalten sowie innerhalb der äußeren liegen.

Zusätzlich gibt es für jede Annotation noch eine Information, ob das Fahrzeug relevant für die Evaluation ist. Manche Fahrzeuge sind nur sehr klein abgebildet, von einigen ist die Vorderansicht zu sehen, die einer Rückansicht stark ähnelt oder das Fahrzeug ist von anderen Objekten etwas verdeckt. Diese Fälle sind mit Bounding-Boxen versehen, aber als unrelevant eingestuft. Das heißt, wenn der Algorithmus diese Fahrzeuge erkennt, zählt das nicht als Fehler, aber auch nicht als Treffer.

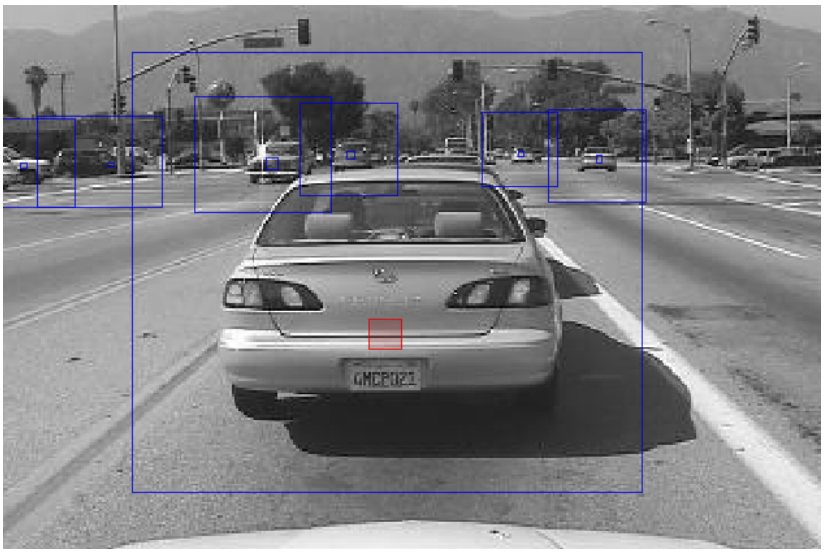


Abbildung 4.2: Die kleineren Annotationen im hinteren Bereich sind als unrelevant eingestuft

4.1 Bewertungskriterien

Es gibt 3 Bewertungskriterien, die Laufzeit sowie Precision und Recall. Die Laufzeit wurde mit der C++ chrono-Bibliothek gemessen.

Precision und Recall ergeben sich aus den folgenden Formeln:

$$Recall = \frac{True\ positives}{Overall\ positives}$$

$$Precision = \frac{True\ positives}{True\ positives\ +\ False\ positives}$$

Tabelle 4.1

Algorithmus	Laufzeit in Sekunden	Bilder pro Sekunde
HOG-Deskriptor	11,59	11,9
HOG-Deskriptor (GPU)	3,167	43,57436
DPM (OpenCV)	98,545	1,40038
DPM (FFLD)	23,062	5,98387
CascadeHaar	25,249	5,46556
CascadeHaar (GPU)	5,172	26,68213
CascadeHOG	45,338	3,0438
CascadeLBP	41,793	3,30199
CascadeLBP (GPU)	8,465	16,30242

"Overall positives" ist die Gesamtzahl der Fahrzeugrückansichten, die als relevant deklariert wurden. In der Testmenge sind 196 davon enthalten. "True positives" ist die Anzahl der gefundenen relevanten Fahrzeugrückansichten und "False positives" sind vom Algorithmus falsch klassifizierte Treffer.

4.2 Ergebnisse

Die Ergebnisse der Stereo-Vision sind sehr zufriedenstellend. Bei der Auswertung der Log-Dateien hat sich herausgestellt, dass die grobe Objekterkennung per Clustering der Stereo-Vision 3D-Punktwolke effizient und zuverlässig läuft. Verdeckte, kleine bzw. weit entfernte Objekte werden oft nicht erkannt, aber größere Objekte, die sich innerhalb von 30m Abstand zum autonomen Auto befinden, dagegen fast immer. Die Laufzeit, gemessen in verarbeiteten Bilderpaaren pro Sekunde, ist mit durchschnittlich 160fps bei einer Gitternetzgröße von 16m x 35m schneller als die INKA-Kameras aufnehmen können (60fps). Somit ist eine Verarbeitung in Echtzeit möglich.

Die Laufzeitmessung der einzelnen Mustererkennungsalgorithmen wurde in zwei verschiedenen Szenarien durchgeführt. Als erstes wurden die Zeiten für die Klassifizierung der gesamten Testmenge gemessen, siehe Tabelle X.
Um auch das Laufzeitverhalten bei unterschiedlicher Bildgröße zu ermitteln, wurde eine große Collage aus mehreren Bildern zu einem großen mit einer Auflösung von 4500x1854 Pixeln erstellt. Es wurde die Laufzeit von Skalierungen im Intervall $\{0.1, 0.2, ..., 0.9, 1.0\}$ der originalen Bildgröße gemessen. Da die Laufzeiten stark variieren, insbesondere zwischen den GPU- und CPU-Varianten, habe ich die Ergebnisse in 2 Diagramme aufgeteilt,

siehe Abb. 4.3 und Abb. 4.4.

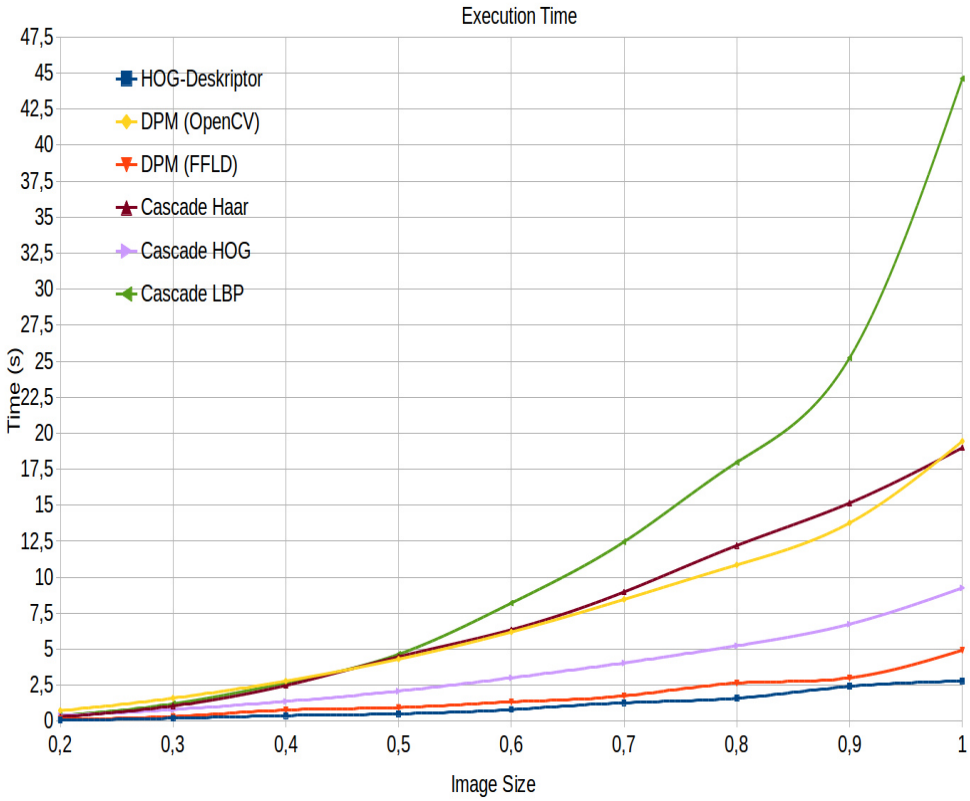


Abbildung 4.3: Laufzeiten der Mustererkennungsalgorithmen

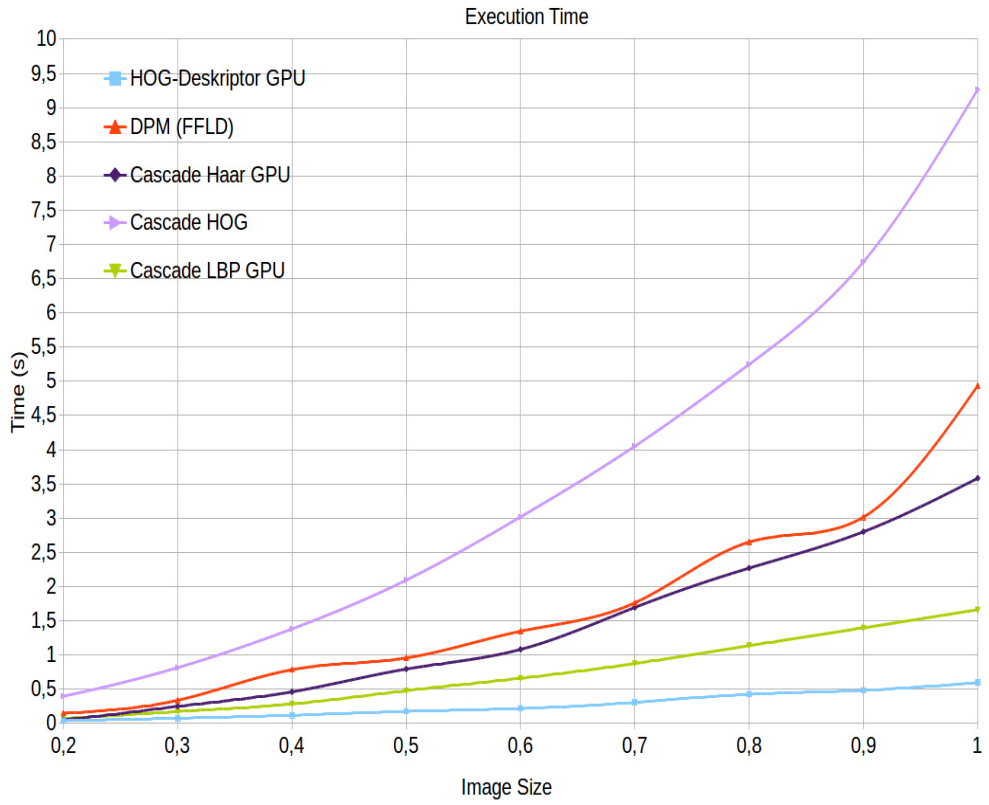


Abbildung 4.4: Laufzeiten der Mustererkennungsalgorithmen

Es zeigt sich, dass der HOG-Deskriptor insgesamt am schnellsten ist, sogar in ohne GPU-Unterstützung. Die optimierte FFLD-Variante des DPM-Algorithmus sowie die GPU-Varianten CascadeHaar-GPU/CascadeLBP-GPU des Viola-Jones-Algorithmus sind mehr als 4 mal so schnell als deren alternative Implementierungen.

Die Verarbeitungszeit eines einzelnen Bildes hängt beim DPM- und Viola-Jones-Ansatz auch stark vom Inhalt des Bildes ab. Da diese Algorithmen nach einem Entscheidungsbaum bzw. einem Sternmodell ein Muster klassifizieren, kommt es darauf an, ab welcher Stufe Bildbereiche zurückgeworfen werden. Bilder die viele Muster enthalten, die dem gelernten Muster ähnlich sind, werden deshalb langsamer ausgewertet als Bilder mit wenigen oder keinem dieser Muster. Beim Einsatz im implementierten System werden wesentlich kleinere Bildausschnitte durch die Stereo-Vision als Vorauswahl an die Mustererkennungsalgorithmen übergeben, so dass die Laufzeiten dort wesentlich höher sind.

Für die Ermittlung der Recall-/Precision-Raten wurde die Testmenge mit

verschiedenen Schwellwerten für den minimalen Trefferwert der einzelnen Algorithmen klassifiziert. Es haben sich folgende Recall-/Precision-Verläufe ergeben, siehe Abb. 4.5.

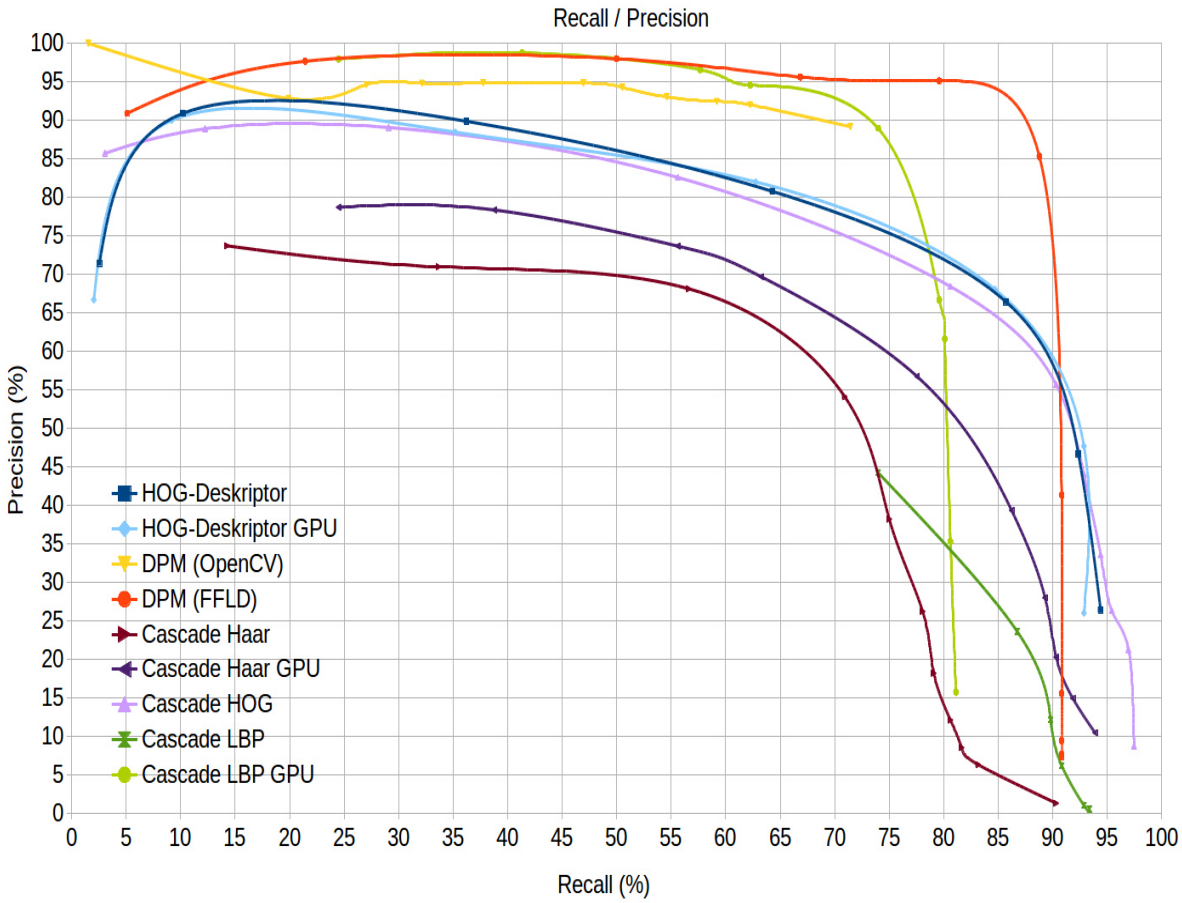


Abbildung 4.5: Ergebnisse für Recall und Precision der Mustererkennungsalgorithmen

Man kann erkennen, dass der DPM-Detektor in der FFLD-Variante bis zu einer Recall-Rate von 85% eine sehr hohe Precision-Rate von $\geq 95\%$ aufweist. Die GPU-Variante von CascadeLBP hat bis zu einem Recall von 65% eine ähnliche hohe Precision wie der DPM-Detektor. Dagegen ist erreicht die CPU-Variante nur Resultate mit höherem Recall, aber viel weniger Precision. Im Mittelfeld bewegt sich der HOG-Deskriptor und die Viola-Jones-Variante mit HOG-Features (CascadeHOG) mit sehr ähnlichen Verläufen. Die Haar-like Features schneiden insgesamt am schlechtesten ab, da deren Recall-/Precision-Kurven unterhalb der anderen liegen. Es ist aber zu

beachten, dass für die Tests Bilder von gesamten Straßenszenen verwendet wurden. Im implementierten System werden durch Stereo-Vision Ausschnitte von in der Nähe befindlichen Objekten geliefert, die mit höherer Präzision als Fahrzeuge klassifiziert werden.

Für den im Kapitel 3 erwähnten "Hohen-Kontrast-Modus" wurden ebenfalls die Recall-/Precision-Raten gemessen, siehe Abb. 4.6. Für die Trainings- und Testmenge wurde der Kontrast entsprechend erhöht.

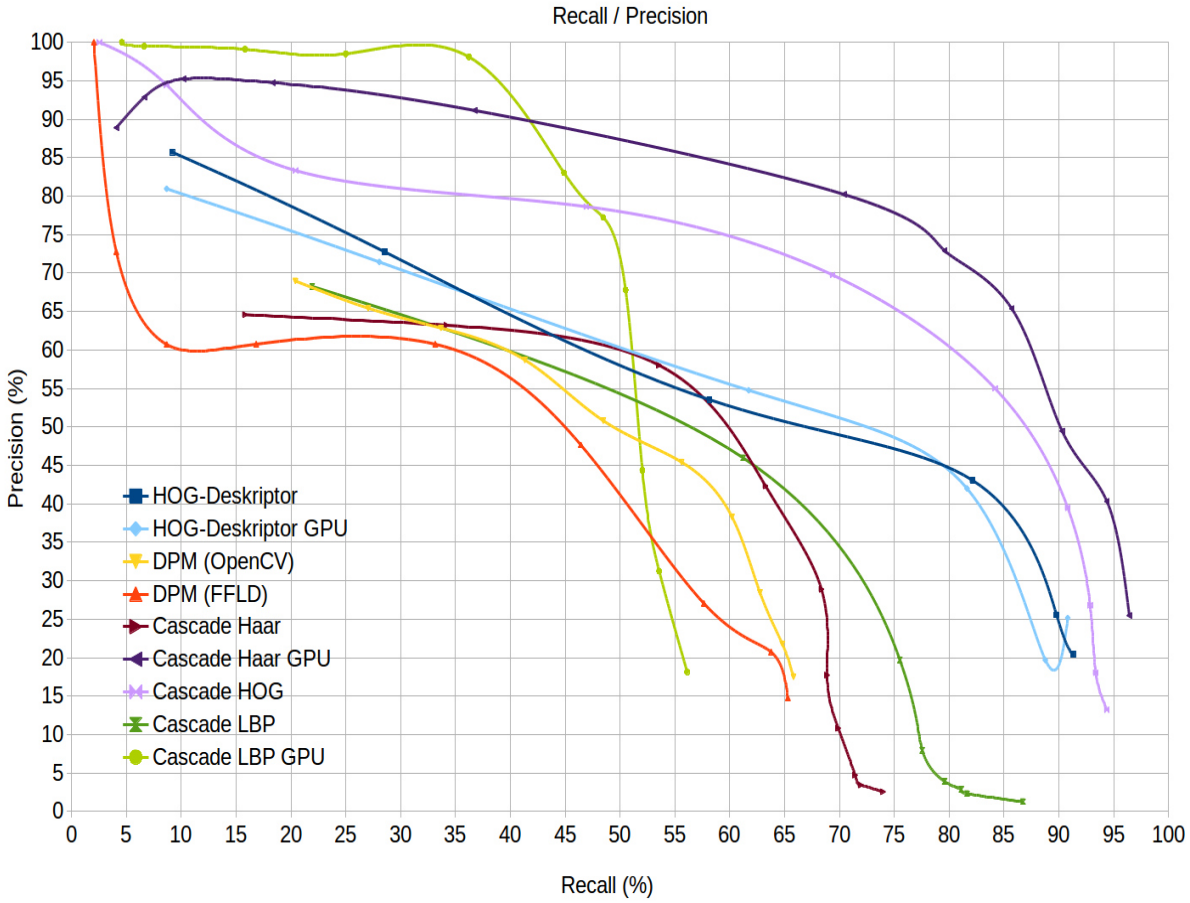


Abbildung 4.6: Ergebnisse für Recall und Precision der Mustererkennungsalgorithmen

Hier sieht man, dass die Recall-/Precision-Raten insgesamt niedriger sind als im Test mit unmodifizierten Bildern. Als einziger Algorithmus erreicht der CascadeHaar-GPU die besten Ergebnisse und ist im Vergleich zu den

unmodifizierten Bildern deutlich besser. CascadeHOG hat noch einen ungefähr gleichen Verlauf wie im vorherigen Test bei weniger Precision. Alle anderen Algorithmen schneiden erheblich schlechter ab. Somit kann man eine Verwendung des "Hohen-Kontrast-Modus" nur für die CascadeHaar-GPU-Variante in Betracht ziehen.

5 Fazit und Ausblick

Im Rahmen dieser Arbeit wurde ein Fahrzeugerkennungssystem implementiert, dass zuverlässig PKW-Rückansichten in der Umgebung des autonomen Autos lokalisieren kann. Das System wurde in abstrakte Objekterkennungsmodule aufgegliedert, so dass es leicht auch an andere Anwendungsfälle, wie z.B. das Erkennen von Fußgängern oder Verkehrsschilder, angepasst werden kann.

Mit dem Einsatz von GPU-unterstützten Implementierungen der Mustererkennungsalgorithmen kann das System in Echtzeit Fahrzeuge in den Kamerabildern identifizieren und durch Stereo-Vision-Clustering sogar deren Position im 3D-Koordinaten ermittelt werden.

Verbesserung in den Erkennungsraten der Bildklassifikatoren sind sicherlich noch möglich, da die Trainingsmenge mit rund 1000 Positivbeispielen relativ klein ist und einige Fahrzeugrückansichten darin mehrmals auftreten, da sie aus den Log-Dateien extrahiert wurden.
Weitere Anwendungszenarios in denen dieses System zum Einsatz kommen könnte:

- **Verfolgen von bestimmten Fahrzeugen:** Es kann mit dem gelernten Klassifikator eine spezielle Fahrzeugrückansicht ausgewählt werden und diese als neuer Klassifikator mit einigen Aufnahmen als Beispielen trainiert werden.
- **Verkehrszählungen:** Ein fest installiertes Stereokamerasystem z.B. an einer Autobahnbrücke, kann automatisch alle erkannten Fahrzeugrückansichten zählen, um so Statistiken über die momentanen Verkehrsituation zu senden.
- **Einparkassistent:** Mithilfe mehrere Stereokameras, die rund um das Fahrzeug sehen können, kann allein mit dem Stereo-Vision-Ansatz eine Erkundung der Umgebung erfolgen.

6 Literaturverzeichnis

[1] AutoNOMOS-Labs. `http://autonomos-labs.com/`. Zugriff: 10.01.2016. 2

[2] Ling Cai, Lei He, Yiren Xu, Yuming Zhao, and Xin Yang. Multi-object detection and tracking by stereo vision. *Pattern Recogn.*, 43(12):4028–4041, December 2010. 5

[3] Safaa Moqqaddem, A Sbihi, R Touahni, and Y Ruichek. *Objects Detection and Tracking Using Points Cloud Reconstructed from Linear Stereo Vision*. INTECH Open Access Publisher, 2012. 5

[4] Rupam Kr Dewan and Ranjit K Barai. Fast kernel based object detection and tracking for stereo vision system. 5

[5] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001. 5, 32, 33, 35, 36

[6] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society. 5, 26

[7] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, September 2010. 5, 30, 31, 32

[8] Wikipedia Lochkamera. `https://de.wikipedia.org/wiki/Lochkamera`. Zugriff: 10.01.2016. 8

[9] Wikipedia Epipolargeometrie. `https://de.wikipedia.org/wiki/Epipolargeometrie`. Zugriff: 10.01.2016. 10

[10] O. Schreer. *Stereoanalyse Und Bildsynthese*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. 13

[11] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning Opencv, 1st Edition*. O'Reilly Media, Inc., first edition, 2008. 13, 16, 18, 19, 21, 22

[12] Wikipedia Image rectification. `https://en.wikipedia.org/wiki/Image_rectification`. Zugriff: 10.01.2016. 14

[13] Wikipedia Mustererkennung. `https://de.wikipedia.org/wiki/Mustererkennung`. Zugriff: 10.01.2016. 25

[14] OpenCV Documentation. `http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`. Zugriff: 10.01.2016. 29

[15] Timo Ojala, Matti Pietikainen, and David Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Proc. 12th International Conference on Pattern Recognition (ICPR 1994), Jerusalem, Israel*, volume 1, pages 582–585. IEEE, 1994. 37

[16] trainHOG Github. `https://github.com/DaHoC/trainHOG`. Zugriff: 10.01.2016. 47