TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

Visuelle Fahrzeugverfolgung mittels Embedded Systems

Ausgeführt am Institut für
Rechnergestützte Automation
Arbeitsbereich für
Mustererkennung und Bildverarbeitung
der Technischen Universität Wien

unter der Anleitung von
Univ.Prof. Dipl.-Ing. Dr.techn. Horst Bischof

durch
Stephan Veigl
Hungereckstr. 32
1230 Wien
Matr.Nr.: 9526342

März 2008

Danksagung

An dieser Stelle möchte ich mich bei all meinen Freunden und Studienkollegen bedanken, die mich während meines Studiums begleitet und unterstützt haben, sei es durch fachliche Diskussionen, oder indem sie mich ermutigt haben weiter zu machen, wenn die Dinge nicht so liefen, wie ich es erhofft hatte. Somit wurden aus Studienkollegen Freunde.

Hier möchte ich mich besondere bei Oliver Fasching bedanken, der mir mathematische Fragen immer kompetent beantwort hat und auch sonst immer einen hilfreichen Rat hatte.

Bei Prof. Horst Bischof möchte ich mich besonders für seine Geduld und seine raschen und hilfreichen Antworten auf meine Fragen bedanken.

Weiters möchte ich mich noch bei meinen Arbeitskollegen bedanken, und zwar ganz besonders bei Gustavo Fernandez Dominguez und Roman Pflugfelder, die nicht müde wurden meine Fragen zu beantworten und verschiedenste Ansätze zu diskutieren.

Ein ganz besonderer Dank gilt meinen Eltern, die mich immer unterstützt und ermutigt haben. Die letzte Zeit war sicher auch für sie nicht immer einfach. Danke!

Schließlich möchte ich mich noch bei meiner Freundin Diana Nenning für ihre große Hilfe bedanken, und dafür, dass sie mich immer wieder ermuntert hat weiter zu machen. Danke.

Wien, am 2. März 2008

Stephan Veigl

Zusammenfassung

Bei steigendem Verkehr und damit größerer Auslastung des Straßennetzes wird es für Verkehrsleitzentralen immer wichtiger, genaue Informationen über die aktuelle Verkehrslage zu erhalten. Die Rohdaten stammen dabei von unterschiedlichen Sensoren, unter anderem auch von Verkehrskameras. In dieser Arbeit wird ein Bildverarbeitungssystem beschrieben, mit dessen Hilfe verkehrsrelevante Daten aus den Videos von statischen Verkehrskameras ermittelt werden können und zwar direkt vor Ort, mittels Embedded Systems.

Dazu werden die einzelnen Module des Bildverarbeitungssystems detailliert beschrieben und ein Prototyp erstellt, welcher abschließend evaluiert wird.

Als Basis wird ein Hintergrundmodell der Szene aufgebaut. In dieser Arbeit werden zuerst verschiedene Hintergrundalgorithmen beschrieben und verglichen, wobei der Approximated Median Algorithmus als der am besten für die Anwendung geeignete ausgewählt wurde. Um Fehler im Hintergrundmodell durch bewegte Fahrzeuge zu vermeiden, wurden eine selektive Hintergrundaktualisierung und eine globale Helligkeitsanpassung des Hintergrundmodells implementiert.

Als Objektmodell dienen ein 3-dimensionaler Quader, der das Fahrzeug einschließt, und ein Objektbasispunkt. Als Basispunkt wird der Schwerpunkt des Objektblobs verwendet, der mit einer konstanten Höhe in das Weltkoordinatensystem projiziert wird. Um auf dem DSP keine Matrizeninversen für die Rückprojektion berechnen zu müssen, wird die Inverse der Homographie der Grundebene vorausberechnet und die Höhe durch einen Korrekturvektor mit einbezogen.

Das Trackingmodul beruht auf einem histogrammbasierten Mean-Shift Algorithmus, der für die Implementierung auf einem DSP adaptiert wurde (Fixpunktarithmetik, vereinfachte Gewichtsfunktion, minimaler Bhattacharyya Koeffizient und Subpixel Skalierung). Die Ergebnisse des Mean-Shift Trackers werden mit den Blobs der Vordergrundmaske aus dem Hintergrundmodell korreliert. Dabei werden beide Ergebnisse mit dem Konfidenzwert des Trackers bzw. der Vordergrundmaske gewichtet.

Diese Module werden zu einem Gesamtsystem integriert, das sowohl auf einem PC als

auch auf einem DSP (*Texas Instruments TMS320C64xx*) lauffähig ist. Dazu wird ein DSP-via-PC Framework entwickelt, um Software, die für einen DSP gedacht ist, am PC entwickeln und testen zu können und gleichzeitig quellcodekompatibel zu bleiben.

In der abschließenden Evaluierung wird das Gesamtsystem sowohl mittels synthetisch erzeugter Testsequenzen als auch Videos von Tunnelkameras und Freilandautobahnen getestet. Somit wird gezeigt, dass Bildverarbeitung zur Fahrzeugverfolgung mit den beschriebenen Methoden, unter bestimmten Bedingungen, auch auf Embedded Systems möglich ist. Eine der wesentlichsten Bedingungen dabei ist, dass eine gute Objektinitialisierung durchgeführt werden kann. Für den entwickelten Prototyp bedeutet dies, dass die Vordergrundmasken der einzelnen Fahrzeuge während der Initialisierung nicht miteinander verschmelzen dürfen.

Acknowledgement

Diese Arbeit wurde vom österreichischen Bundesministerium für Verkehr, Innovation und Technologie (bmvit) im Rahmen des Programms I2-2-26p VITUS-2 unterstützt.
VITUS Homepage: `http://www.smart-systems.at/rd/rd_video_vitus_de.html`

Das Forschungsprojekt AVITRACK wurde von der Europäischen Union unterstützt (AST3-CT-2003-502818). Mehr Information zu dem Projekt befindet sich auf folgender Webseite: `http://www.avitrack.net`

Teile der Illustrationen basieren auf Grafiken der Open Clip Art Library:
`http://www.openclipart.org`.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

apron (Flughafen Vorfeld) Das *Vorfeld* ist jener Bereich des Flughafens, in dem die Flugzeuge geparkt, be- und entladen werden.

Bildhauptpunkt Der *Bildhauptpunkt* (c_x, c_y) ist der Schnittpunkt der optischen Achse mit der Bildebene (in Bildkoordinaten) [2](Seite 48).

Embedded System Geräte oder Systeme, in denen ein oder mehrere Digitale Signalprozessoren eingebaut sind und eine ganz bestimmte Aufgabe erledigen, werden als *Embedded Systems* bezeichnet. Die Programmierung erfolgt durch den Hersteller und ist vom Anwender nicht änderbar (*Firmware*).

externe Kameraparameter Die *externen* oder *äußeren Kameraparameter* beschreiben die Lage und Orientierung der Kamera (bzw. des Kamerakoordinatensystems) im Weltkoordinatensystem [3](Seite 13f).

Frame Ein *Frame* ist ein Einzelbild eines Films bzw. Videos.

Ground Truth Die Referenzdaten einer Evaluierung werden als *Ground Truth* bezeichnet.

Homographie Die *Homographie* ist eine bijektive Abbildung zwischen der Oberfläche zweier geometrischer Figuren bzw. zwischen (Teil-)Räumen.

HSV bezeichnet ein Farbmodell, das den Farbraum als umgedrehten Kegel darstellt. Die Mittelachse entspricht der Dunkelstufe (*value* V), wobei die Spitze Schwarz und die Basis Weiß sind. Der Radius beschreibt die Sättigung (*saturation* S), von Grau an der Mittelachse bis zur reinen Farbe am Rand. Der Farbton (*hue* H) wird durch den Winkel zu einer Bezugsebene durch die Mittelachse beschrieben.

interne Kameraparameter Die *internen Kameraparameter* sind die konstruktionsbedingten Parameter der Kamera: Fokale Länge, Lage des Bildhauptpunktes sowie Skalierung und Verzerrung [3](Seite 13 u. 17).

Kameramatrix Mit Hilfe der 4x3 *Kameramatrix* wird die Weltkoordinaten-in-Bildkoordinaten Transformation durchgeführt.

kartesisches Koordinatensystem Ein orthogonales, äquidistantes Koordinatensystem wird als *kartesisches Koordinatensystem* bezeichnet (benannt nach René Descartes).

persistenter Speicher Ein Speicher, dessen Inhalt auch zwischen Deaktivierung und neuerlicher Aktivierung einer Algorithmusinstanz erhalten bleibt, wird als *persistenter Speicher* bezeichnet.

RGB bezeichnet ein Farbmodell mit drei Farbkanälen: Rot, Grün und Blau.

scratch Speicher Ein Speicher, den sich mehrere Algorithmen teilen, wird als *scratch Speicher* bezeichnet. Der Inhalt kann überschrieben werden, nachdem die Algorithmusinstanz deaktiviert wurde.

sparse Eine *sparse* oder *dünnbesetzte* Datenstruktur bzw. Matrix besitzt überwiegend Einträge, die den Wert 0 haben.

YCrCb bezeichnet ein Farbmodell mit einem Helligkeitskanal Y und zwei Farbkanälen: Cr und Cb. Dieses Farbmodell bildet häufig die Grundlage zur analogen Codierung und Übertragung von Farbbildern.

Abkürzungsverzeichnis

API - Application Programming Interface *Programmierschnittstelle*
Unter API versteht man eine wohl definierte Programmierschnittstelle von Modulen bzw. Bibliotheken. Mittels dieser Schnittstelle können Anwendungsprogramme Funktionen dieser Module bzw. Bibliotheken aufrufen und benutzen.

DSP - Digital Signal Processor *Digitaler Signalprozessor*
DSPs sind Microprozessoren, die speziell für die Signalverarbeitung optimiert sind. Im Gegensatz zu üblichen Büroanwendungen, bei denen hauptsächlich Daten bewegt werden, kommt es bei der digitalen Signalverarbeitung darauf an, mathematische Algorithmen möglichst schnell (in Echtzeit) zu berechnen. [4](Seite 503f)

IDE - Integrated Development Environment *integrierte Entwicklungsumgebung*
Eine IDE ist eine Programmsammlung, die die wichtigsten Werkzeuge der Softwareentwicklung (Texteditor, Compiler, Linker bzw. Interpreter, Debugger, etc.) unter einer einheitlichen Oberfläche zur Verfügung stellt.

ROI - Region of Interest
Die ROI ist jener Bereich des Bildes oder der Matrix, auf den die Verarbeitung angewandt wird.

XDAIS - eXpressDSP Algorithm Interface Standard
XDAIS ist ein von Texas Instruments entwickelter Algorithmusstandard für TMS320x DSPs. [5]

1 Einleitung

1.1 Motivation & Ziel

Das Fahrzeugvolumen steigt bei gleichbleibender Kapazität der Straßen. Um in dieser Situation einen optimalen Verkehrsfluss gewährleisten zu können, muss im Bedarfsfall durch ein aktives Verkehrsmanagement regelnd eingegriffen werden. Dazu muss wiederum die aktuelle Verkehrslage bekannt sein, also beobachtet werden.

Dies geschieht einerseits durch automatisch ermittelte Kennzahlen wie die Anzahl der Fahrzeuge pro Zeiteinheit, die mittlere Geschwindigkeit, die Belegung einzelner Spuren, etc. Eine sehr verbreitete Methode zur Erfassung dieser Daten sind in die Fahrbahn eingelassene Induktionsschleifen. Diese Sensoren liefern aber immer nur punktuelle Messungen und erfordern zusätzliche bauliche Maßnahmen.
Andererseits werden live Videobilder ausgewählter Streckenabschnitte in Verkehrsleitzentralen beobachtet und beurteilt. Dabei können aber immer nur eine begrenzte Anzahl an Verkehrsvideos auf dem Bildschirm dargestellt werden, wobei diese Anzahl durch technische Rahmenbedingungen und vor allem auch durch die begrenzte Aufnahmekapazität des menschlichen Beobachters limitiert ist.

Bildverarbeitungssysteme können den Beobachter unterstützen, indem sie die aktuelle Verkehrslage durch Auswertung der Bilder *aller* Verkehrskameras feststellen und relevante Videos auf den Monitor aufschalten, Problemzonen visualisieren oder Meldungen ausgeben.

Die notwendige Bildverarbeitung kann dabei entweder in der Verkehrsleitzentrale, direkt an der Quelle (in der Nähe der Kamera) oder verteilt (Vorverarbeitung bei der Quelle, weitere Verarbeitung und Analyse in der Zentrale) erfolgen.
Abhängig von dem Ort, an dem die Daten verarbeitet werden, sind unterschiedliche Systeme denkbar. Ein PC System hat den großen Vorteil, dass Standardkomponenten (sowohl Hardware als auch Software) verwendet werden können, wohingegen die Hardware eines Embedded System individuell an die Umgebungsbedingungen angepasst werden kann.

Gerade bei dem Einsatz im „Feld“ haben Embedded Systems Vorteile gegenüber einem PC System:

- Embedded Systems sind unempfindlicher gegenüber Umwelteinflüssen als PC Systeme (insb. zulässige Umgebungstemperatur).

- Embedded Systems können so geplant und gebaut werden, dass sie keine bewegten Teile enthalten. Daher ist kein mechanischer Verschleiß möglich, was die Lebensdauer erhöht und sie wartungsfreundlich macht.

Somit können Embedded Systems direkt vor Ort in der Nähe der Kamera eingebaut werden, wodurch die Bildverarbeitung auf qualitativ hochwertigen Rohbildern durchgeführt werden kann, da die Bilder vorher nicht über weite Strecken übertragen und dazu komprimiert werden müssen (digitale Übertragung) bzw. durch Übertragungsstörungen an Qualität verlieren (analoge Übertragung).

Allerdings sind die Prozessoren von Embedded Systems nicht so leistungsstark wie die von PC Systemen und die verfügbaren Ressourcen sind vergleichsweise eingeschränkt, wodurch die Bildverarbeitung auf Embedded Systems schwieriger wird.

Das Ziel dieser Diplomarbeit ist es, entsprechende Algorithmen für ein Fahrzeugverfolgungssystem mittels Embedded Systems auszuwählen, einen Prototyp zu erstellen und zu evaluieren, um zu zeigen, ob, bzw. mit welcher Qualität, dies möglich ist.

1.2 Überblick

Der Prototyp zur Verfolgung von Fahrzeugen mittels Embedded Systems muss, wie jedes Trackingsystem, mehrere Teilaufgaben lösen. Diese Lösung der Teilaufgaben lässt sich durch mehrere nur lose miteinander verbundene Module realisieren (siehe Abbildung 1.1).

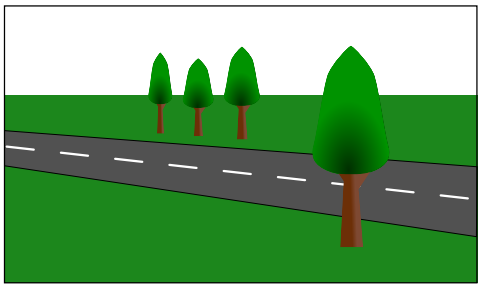
Der erste Teil dieser Arbeit beschäftigt sich mit den Modulen der Bildverarbeitung.

Abbildung 1.1: Blockdiagramm der Module eines Trackingsystems

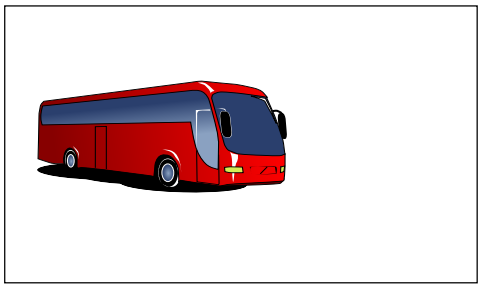
Das Hintergrundmodell erstellt ein Modell des aktuellen Hintergrundes. Dabei werden unter Hintergrund jene Bereiche im Bild verstanden, die über den Beobachtungszeitraum starr sind, oder sich in wohl definierter, vorhersagbarer Weise ändern (z.B. Umschalten einer Verkehrsampel). Ein einfaches Beispiel ist das Bühnenbild mit fest installierten Requisiten im Theater. Es kann vorkommen, dass eine Person oder ein Objekt von einem dem Hintergrund zugeordnetem Gegenstand (z.B. einen Baum) teilweise oder vollständig verdeckt wird (siehe Abbildung 1.2).
Durch einen Vergleich des aktuellen Hintergrundmodells mit dem aktuellen Bild ist es möglich, die Vordergrundobjekte zu extrahieren. Im einfachsten Fall kann dies durch eine Differenzmaskenbildung mit einer anschließenden Segmentierung und Labeling geschehen.
Zuerst werden einige bekannte Algorithmen vorgestellt und miteinander verglichen. Anschließend wird der erweiterte Hintergrundalgorithmus, so wie er im Prototyp implementiert wurde, beschrieben. Der Segmentierung und dem Labeling der vom Hintergrundalgorithmus erzeugten Masken wird ein eigenes Unterkapitel gewidmet.
Das Thema Hintergrundmodell wird in Kapitel 2 diskutiert.

Mit den Resultaten des Hintergrundmodells, der Segmentierung und dem Labeling kann für jedes relevante Objekt eine eigene Instanz des Objektmodells erzeugt werden. Dabei werden jene Segmente, die nicht dem Hintergrund entsprechen, als Blobs bezeichne. Die Vereinigung aller Blobs in einem Frame stellt den Vordergrund dar. Neue Blobs werden zur Initialisierung neuer Objekte verwendet, wenn dies vom Objektmodell unterstützt wird.
Außerdem wird in diesem Kapitel die Projektion von 3D Positionen in Pixel Koordinaten und deren Rückprojektion behandelt, da diese Transformation im Objektmodell benutzt wird.
Das Objektmodell wird in Kapitel 3 beschrieben.

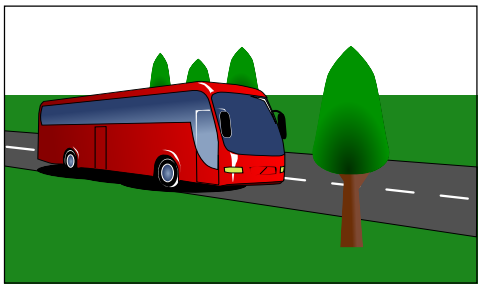
Beim Tracking wird das Abbild eines Objektes über mehrere Frames hinweg verfolgt. Das bedeutet, dass eine Korrelation des Objektes im vorangegangenen Bild zu einem Bereich im aktuellen Bild hergestellt wird. Dabei wird nach übereinstimmenden Merkmalen wie Position, Struktur, Farbe, etc. gesucht.
Wenn sich zwei oder mehr Objekte überlappen, verschmelzen diese zu einem als Gesamtes segmentierten Bereich, das heißt zu einem einzigen Blob. Dieser Blob muss dann durch einen höhere Logik geteilt und den jeweiligen Objekten zugeordnet werden. Andererseits ist es auch möglich, dass ein Objekt in mehrere Blobs zerfällt, was eintritt, wenn es von einem Gegenstand, der zum Hintergrund gehört, so überdeckt wird, dass die sichtbaren Teile nicht miteinander verbunden sind (siehe Abbildung 1.2(f)).
Zuerst wird der allgemeine Mean-Shift Algorithmus erläutert, anschließend werden die Erweiterungen und Abänderungen, wie sie im Prototyp implementiert wurden, diskutiert. Der Abgleich der Ergebnisse des Trackers mit dem verwendeten Objektmodell und weiteren Informationen (aus dem Hintergrundmodell) wird als weiterer Punkt in diesem



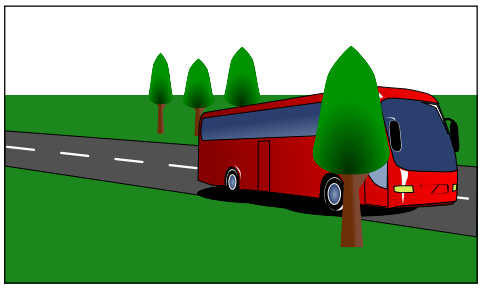
(a) Hintergrund der Szene



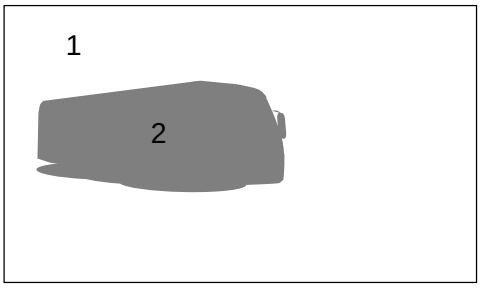
(b) durch die Szene bewegtes Objekt



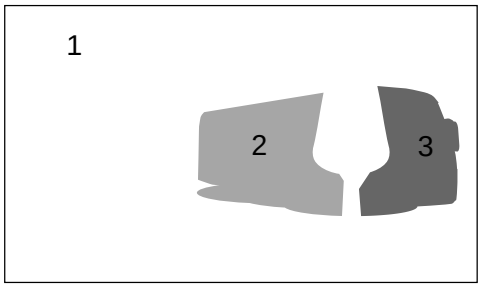
(c) Bild: Objekt (Bus) vollständig sichtbar



(d) Bild: Objekt (Bus) teilweise verdeckt



(e) Segmentierung: Objekt (Bus) vollständig sichtbar



(f) Segmentierung: Objekt (Bus) teilweise verdeckt

Abbildung 1.2: Ein Objekt zerfällt in mehrere Blobs.

Kapitel behandelt.
Das Trackingmodul ist in Kapitel 4 beschrieben.

Der zweite Teil dieser Arbeit beinhaltet die Integration der Bildverarbeitungsmodule zu einem Gesamtsystem, sowie die Evaluierung des Prototypen und die Ergebnisse.

Die Kernmodule des Prototypen wurden in Abbildung 1.1 illustriert und in den vorigen Absätzen kurz beschrieben. Für einen funktionierenden Prototyp werden allerdings noch ein Rahmenwerk und weitere Hilfsmodule benötigt, in das die Bildverarbeitungsmodule eingebettet werden.
Zuerst wird ein Überblick über die verwendeten Hilfsmodule sowie das DSP-via-PC Framework gegeben. Anschließend wird erläutert, wie die in den vorigen Kapiteln beschriebenen Module zu einem Gesamtsystem zusammengefügt werden.
Das Gesamtsystem wird in Kapitel 5 behandelt.

Nachdem unterschiedliche Bildverarbeitungsmodule zu einem Gesamtsystem integriert wurden, wird die Trackingqualität dieser Kombination unterschiedlicher Algorithmen getestet.
Zuerst wird die Evaluationsmethode mit der verwendete Metrik und den berechneten Kennzahlen erläutert. Die Evaluation wird sowohl mittels synthetisch erzeugten Testsequenzen als auch mit Videos echter Verkehrskameras durchgeführt. Die einzelnen Module werden gesondert evaluiert, bevor das Gesamtsystem getestet wird und die Ergebnisse ausgewertet werden. Zusätzlich wird eine Laufzeitabschätzung aufgrund von Messungen mit einem Simulator durchgeführt.
Die Evaluierung des Prototypen wird in Kapitel 6 beschrieben.

Abschließend wird diese Arbeit im Kapitel 7 zusammengefasst und mögliche zukünftige Erweiterungen werden diskutiert.

1.3 Allgemeine Annahmen

Voraussetzungen sowie grundlegende Annahmen, die für die gesamte Arbeit gelten, sind hier zusammengefasst.

konstante Kameraorientierung: Die Kameras sind unbeweglich montiert. Dies ist die Grundvoraussetzung um die beschriebenen Hintergrundmodelle einsetzen zu können.

Common Ground Plane: Alle Objekte bewegen sich auf genau einer Ebene (Grundfläche).

Fahrbahn: Alle interessanten Objekte bewegen sich nur auf der Fahrbahn (= Region of Interest (ROI)), Blobs außerhalb der ROI (z.B. Reflexionen, Rauschartefakte, etc.) werden ignoriert.

Fahrtrichtung / Fahrzeugorientierung: Fahrzeuge bewegen sich im Allgemeinen parallel zur Fahrspur. Spurwechsel sind eher die Ausnahme und bewirken nur eine geringfügige Drehung des Fahrzeuges.

definierter Ein- / Ausfahrtsbereich: Objekte können nicht plötzlich auftauchen bzw. verschwinden. Es gibt klar definierte Bereiche, in denen Objekte in das Bild eintreten bzw. das Bild verlassen können.

Quadermodell: Als Fahrzeugmodell dient ein 3D Quader, der das Fahrzeug einschließt und sich direkt auf der Grundfläche befindet.

starre Objekte: Alle relevanten Objekte sind starr und ändern ihre Erscheinung nicht.

Bei den betrachteten Objekten handelt es sich, falls nicht explizit anders beschrieben, ausschließlich um Fahrzeuge, wie sie am allgemeinen Straßenverkehr teilnehmen.
Die angestellten Überlegungen bzw. verwendeten Algorithmen sind nicht an einen bestimmten Objekttyp gebunden. Im Prototyp wurde keine Objekttyperkennung integriert, sondern angenommen, dass sich alle Fahrzeuge durch *ein* Objektmodell modellieren lassen.

2 Hintergrunderkennung

Um neue Objekte erkennen zu können, muss die Ausgangssituation wohldefiniert sein. Jedes neue Objekt stellt dann eine Veränderung dieser Ausgangssituation dar, die im Bild festgestellt werden kann, wobei die Ausgangssituation als Hintergrund bezeichnet wird.

In dieser Arbeit wird der Szenen-Begriff auf dem Szene-Begriff der Computergrafik aufgebaut: Unter einer Szene werden alle Objekte und die Beleuchtung einer Videosequenz verstanden.
Eine leere Szene ist die Menge aller statischen und für die Aufgabenstellung nicht relevanten Objekte.

Der Hintergrund wird somit durch eine „leere Szene“ repräsentiert, deren Bild aber nicht statisch ist (z.B. Beleuchtungsänderung). Das Modell des Hintergrundes muss diese Änderungen berücksichtigen und entsprechend integrieren, während Änderungen von Nicht-Hintergrundobjekten nicht berücksichtigt werden sollen.

2.1 Hintergrundmodelle in der Literatur

Cheung und Kamath vergleichen in [1] verschiedene Hintergrundalgorithmen. Dabei gehen sie von einem Flussdiagramm eines generischen Hintergrundalgorithmus aus (Abbildung 2.1).

Die Komponenten sind:

Vorverarbeitung (*Preprocessing*): notwendige Vorverarbeitungsschritte wie Skalierung, Rauschunterdrückung, Farbanpassung, Filter oder Registrierung des Bildausschnittes bei beweglichen Kameras.

Hintergrundmodellierung (*Background Modelling*): statistische Beschreibung der Hintergrundszene und Verarbeitung des aufbereiteten Bildes aus dem vorhergehenden

Schritt.

Vordergrunderkennung (*Foreground Detection*): Klassifizierung von Hintergrund- und Vordergrundbereichen und Erstellung einer entsprechenden Maske. Als Vordergrund werden alle Pixel gewertet, die nicht durch das aktuelle Hintergrundmodell beschrieben werden können.

Datenvalidierung (*Data Validation*): Überprüfung der Maske mit High-Level Methoden.

Rückkopplung (*Delay*): Durch die Rückkopplungsschleife wird das vorhergehende Bild, zusammen mit den Informationen der Vordergrunderkennung und Datenvalidierung, wieder in die Hintergrundmodellierung eingespeist.

Vordergrundmasken (*Foreground Masks*): Maske die, nach der Datenvalidierung, jene Bereiche enthält, die als Vordergrund klassifiziert wurden.

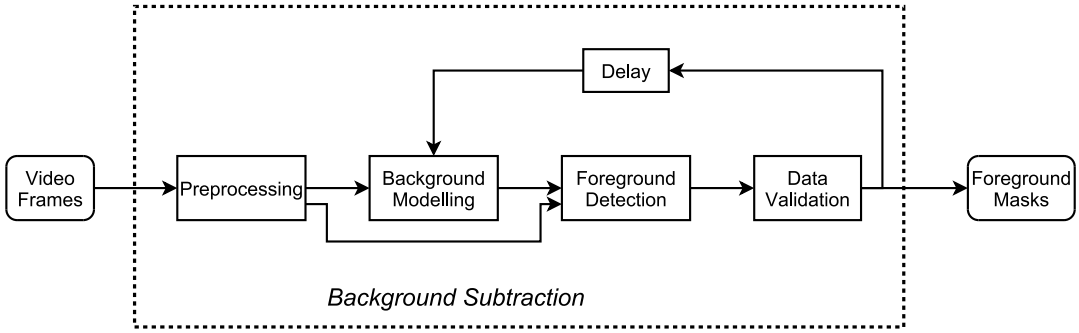


Abbildung 2.1: Generischer Hintergrundalgorithmus nach Cheung und Kamath [1]

Die Autoren teilen die von ihnen untersuchten Algorithmen in nicht-rekursive und rekursive Techniken ein.

Nicht-Rekursive Algorithmen

Nicht-Rekursive Methoden verwenden einen Buffer konstanter Größe, in dem die letzten n Frames, zeitlich geordnet, gespeichert sind. Das Hintergrundmodell bezieht sich ausschließlich auf Daten des aktuellen Buffers. Frames bzw. Daten die nicht mehr im Buffer gespeichert sind, werden für die Hintergrundmodellierung nicht mehr berücksichtigt. Diese Methoden können schnell auf Änderungen im Hintergrund reagieren. Andererseits kann der benötigte Buffer sehr groß werden, wenn sich langsam bewegte Objekte in der Szene befinden, die nicht in das Hintergrundmodell aufgenommen werden sollen. Je

nach Algorithmus und Implementierung kann daher der benötigte Speicherplatz sehr hoch sein.

Rekursive Algorithmen

Rekursive Methoden integrieren das aktuelle Frame direkt in das Hintergrundmodell, ohne es in einem Buffer zu speichern, daher benötigen sie im Vergleich zu Nicht-Rekursiven Methoden weniger Speicher. Alle vergangenen Frames haben einen Einfluss auf das aktuelle Hintergrundmodell, da sie nicht gespeichert wurden und daher nicht mehr gezielt aus dem Hintergrundmodell herausgerechnet werden können. Dies bedeutet, dass sich Fehler oder veraltete Daten nicht mehr aus dem Hintergrundmodell entfernen lassen. Um dieses Problem zu minimieren verwenden die meisten Algorithmen eine Gewichtung, in der die Vergangenheit weniger stark gewichtet wird. Zusätzlich zur zeitlichen Gewichtung wird meist ein selektives Hintergrundupdate verwendet, bei dem Vordergrundbereiche des aktuellen Frames gar nicht, oder deutlich schwächer, in das Hintergrundmodell integriert werden.

2.1.1 Hintergrundalgorithmen

In diesem Kapitel werden grundlegende Hintergrundalgorithmen beschrieben. Als Basis für die Auswahl der Algorithmen werden die folgenden Arbeiten verwendet, in denen mehrere Hintergrundalgorithmen verglichen werden.

- *„Robust techniques for background subtraction in urban traffic video“* (Cheung und Kamath) [1]
- *„Visual Surveillance of an Airport's Apron - An Overview of the AVITRACK Project“* (Blauensteiner und Kampel) [6]
- *„Deliverable D3.1 Motion Detection“* (Borg et al.) [7]

Aus den in diesen Arbeiten beschriebenen Algorithmen werden sieben grundlegende Hintergrundmodelle ausgewählt und weiter behandelt. Dabei handelt es sich um zwei nichtrekursive Algorithmen (*Frame Differencing* und *Median*), sowie fünf rekursive Algorithmen (*Gleitender Mittelwert*, *Mixture of Gaussians*, *Approximated Median*, *Min-Max / W4 Algorithmus* und *Kalman*).

Frame Differencing

Beim Frame Differencing Verfahren wird ein Differenzbild Δ_t von zwei aufeinanderfolgenden Bildern I_t, I_{t-1} berechnet. Mittels eines Schwellwert Φ wird eine Differenzmaske M_t erstellt, die zwischen bewegten und statischen Bereichen unterscheidet. [6]

$$\Delta_t = |I_t - I_{t-1}| \tag{2.1}$$

$$M_t(x,y) = \begin{cases} 1 & \text{für } \Delta_t(x,y) > \Phi \\ 0 & \text{sonst} \end{cases} \tag{2.2}$$

mit:

- Φ: Masken-Schwellwert
- I_t: Bild zum Zeitpunkt t
- Δ_t: Differenzbild zum Zeitpunkt t
- M_t: Differenzmaske zum Zeitpunkt t

Gleitender Mittelwert

Hier wird ein gleitendes Mittelwertbild als Hintergrundmodell verwendet. Die Vordergrund-/Hintergrundklassifikation erfolgt durch ein Differenzbild Δ_t, aus dem mittels Schwellwert Φ eine Differenzmaske M_t erstellt wird. [6]

$$B_t = \alpha I_t + (1-\alpha) B_{t-1} \tag{2.3}$$

$$\Delta_t = |I_t - B_t| \tag{2.4}$$

$$M_t(x,y) = \begin{cases} 1 & \text{für } \Delta_t(x,y) > \Phi \\ 0 & \text{sonst} \end{cases} \tag{2.5}$$

mit:

- α: Gewichtungsfaktor des aktuellen Bildes
- Φ: Hintergrund-Schwellwert

I_t: Bild zum Zeitpunkt t
B_t: Hintergrundmodell (Bild) zum Zeitpunkt t
Δ_t: Differenzbild zum Zeitpunkt t
M_t: Vordergrundmaske zum Zeitpunkt t

Mixture of Gaussian (MoG)

Der Mixture of Gaussian Algorithmus modelliert den Hintergrund durch eine Anzahl k von Gauß-Verteilungen, wodurch multimodale Hintergrundmodelle erstellt werden können [8]. Jedes Pixel wird dabei durch k Gauß-Verteilungen repräsentiert.

Zuerst wird die am besten passende Gauß-Verteilung b an der Position (x, y) ermittelt, wobei ein absoluter Schwellwert T miteinbezogen wird:

$$b(x,y) = \underset{i=1}{\overset{k}{\operatorname{argmin}}} \left(\frac{|I_t(x,y) - \mu_{i,t-1}|}{\sqrt{\sigma^2_{i,t-1}}} < T \right) \tag{2.6}$$

mit:

I_t: Bild zum Zeitpunkt t
$\mu_{i,t}$: Hintergrundmodell: Mittelwert der i-ten Gauß-Verteilung zum Zeitpunkt t
$\sigma^2_{i,t}$: Hintergrundmodell: Varianz der i-ten Gauß-Verteilung zum Zeitpunkt t
T: max. Standard Abweichung mit den Gauß-Verteilungen (Stauffer verwendet in [8] $T = 2.5$)

Anschließend werden die Gewichte der Verteilungen neu berechnet,

$$i = 1 \dots k : \tag{2.7}$$

$$\omega'_{i,t} = (1-\alpha)\omega_{i,t-1} + \alpha\delta_{b=i} \tag{2.8}$$

$$\delta_{b=i} = \begin{cases} 1 & \text{falls } b = i \\ 0 & \text{sonst} \end{cases} \quad \text{(Kronecker-Delta)} \tag{2.9}$$

$$\omega_{i,t} = \frac{\omega'_{i,t}}{\sum_{j=1}^{k} \omega'_{j,t}} \tag{2.10}$$

und die Verteilung aktualisiert.

$$\rho = \alpha\eta(I_t, \mu_{b,t}, \sigma_{b,t}) \tag{2.11}$$

$$\eta(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\ \sigma} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \quad \text{(Gaußsche Dichtefunktion)} \tag{2.12}$$

$$\mu_{b,t} = (1-\rho)\mu_{b,t-1} + \rho I_t \tag{2.13}$$

$$\sigma^2_{b,t} = (1-\rho)\sigma^2_{b,t} + \rho(I_t - \mu_{b,t})^2 \tag{2.14}$$

mit:

- δ: Kronecker-Delta
- ρ: Updateparameter
- α: Gewichtsupdate
- $\omega_{j,t}$: Hintergrundmodell: Gewicht der j-ten Gauß Verteilung zum Zeitpunkt t
- $\eta(x, \mu, \sigma)$: Gaußsche Dichtefunktion

Die Verteilungen werden nach ihrem $\omega_{i,t}/\sigma_{b,t}$ Wert sortiert. Dabei repräsentieren die ersten K Verteilungen mit $\sum_{i=1}^{K} \omega_{i,t} < \Phi$ die Hintergrundverteilungen.

mit:

- Φ: Hintergrund-Schwellwert

Die Vordergrundmaske M wird folgendermaßen berechnet:

$$M_t(x,y) = \begin{cases} 0 & \text{wenn } I_t(x,y) \in \mathit{Hintergrundverteilung} \\ & \left(\text{das heißt } \frac{|I_t(x,y)-\mu_{b,t}(x,y)|}{\sqrt{\sigma^2_{i,t}(x,y)}} < 2.5 \text{ und } b(x,y) \leq K\right) \\ 1 & \text{sonst} \end{cases} \tag{2.15}$$

Existiert im ersten Schritt keine passende Verteilung i mit: $\frac{|I_t(x,y)-\mu_{i,t-1}|}{\sqrt{\sigma^2_{i,t-1}}} < T$ (vgl. Gleichung: 2.6), so wird die Verteilung mit den geringsten Gewicht $\omega_{i,t-1}$ durch eine neue Gauß-Verteilung b ersetzt.

Median

Das Hintergrundmodell wird durch den statistischen Median generiert. Dabei wird für jedes Pixel der Median aus einer Sequenz von n Bildern (Hintergrundsequenz) berechnet [6]. Ist der Hintergrund in mindestens der Hälfte der Bilder der Sequenz zu sehen (das heißt in mindestens $n/2$ Bildern), ist durch die Eigenschaft des Medians sichergestellt, dass das Hintergrundmodell den tatsächlichen Hintergrund wiedergibt.

$$BS_t = \left\{BS^2_{t-1}, BS^3_{t-1}, \ldots, BS^n_{t-1}, I_t\right\} \tag{2.16}$$

$$B_t(x,y) = \underset{i=1}{\overset{n}{\text{median}}}\, BS^i_t(x,y) \tag{2.17}$$

$$\Delta_t = |I_t - B_t| \tag{2.18}$$

$$M_t(x,y) = \begin{cases} 1 & \text{für } \Delta_t(x,y) > \Phi \\ 0 & \text{sonst} \end{cases} \tag{2.19}$$

mit:

- n: Anzahl der Bilder in der Hintergrundsequenz
- Φ: Hintergrund Schwellwert
- I_t: Bild zum Zeitpunkt t
- BS_t: Hintergrundsequenz zum Zeitpunkt t (Menge von n Bildern)
- B_t: Hintergrundmodell (Bild) zum Zeitpunkt t
- Δ_t: Differenzbild zum Zeitpunkt t

M_t: Vordergrundmaske zum Zeitpunkt t

Approximated Median

Der Approximated Median nähert den Median durch eine einfache und vor allem speicherschonende Methode an. Dabei werden die Farbwerte der Hintergrundpixel in jedem Frame um einen konstanten Wert κ an die Werte im aktuellen Bild angeglichen. Diese Methode wurde ursprünglich von McFarlane und Schofield beschrieben [9] und von Remagnino et al. [10] in einem Trackingsystem für Fahrzeuge und Personen angewendet.

$$B_t(x,y) = \begin{cases} B_{t-1}(x,y) & \text{für } B_{t-1}(x,y) = I_t(x,y) \\ B_{t-1}(x,y) + \kappa & \text{für } B_{t-1}(x,y) < I_t(x,y) \\ B_{t-1}(x,y) - \kappa & \text{für } B_{t-1}(x,y) > I_t(x,y) \end{cases} \tag{2.20}$$

$$\Delta_t = |I_t - B_t| \tag{2.21}$$

$$M_t(x,y) = \begin{cases} 1 & \text{für } \Delta_t(x,y) > \Phi \\ 0 & \text{sonst} \end{cases} \tag{2.22}$$

mit:

κ: Updatewert (Standardwert: $\kappa = 1$ [9])
Φ: Hintergrund-Schwellwert
I_t: Bild zum Zeitpunkt t
B_t: Hintergrundmodell (Bild) zum Zeitpunkt t
Δ_t: Differenzbild zum Zeitpunkt t
M_t: Vordergrundmaske zum Zeitpunkt t

Min-Max / W^4 Algorithmus

Dieser Algorithmus ermittelt während der Initialisierungsphase den Minimal- und Maximalwert für jedes Pixel, sowie die maximale Farbwertdifferenz eines Pixels zwischen zwei aufeinanderfolgenden Bildern. Diese drei Werte stellen das Hintergrundmodell eines Pixels dar. Liegt der Farbwert eines Pixels außerhalb dieser Parameter, wird das Pixel als Vordergrund klassifiziert. [11]

Der Min-Max Algorithmus benötigt eine Lernphase, in der nur Hintergrund im Bild zu sehen ist. Während dieser Lernphase werden die Parameter (Minimum, Maximum und Abweichung) für jedes Pixel ermittelt.

Initialisierung:

$$B_{min}(x,y) = \min_{i=0}^{n-1} I_{t-i}(x,y) \tag{2.23}$$

$$B_{max}(x,y) = \max_{i=0}^{n-1} I_{t-i}(x,y) \tag{2.24}$$

$$B_{diff}(x,y) = \max_{i=0}^{n-2} |I_{t-i}(x,y) - I_{t-i-1}(x,y)| \tag{2.25}$$

mit:

- n: Anzahl der Bilder in der Initialisierungsphase
- I_t: Bild zum Zeitpunkt t
- B_{min}: Hintergrundmodell: Minimalwerte
- B_{max}: Hintergrundmodell: Maximalwerte
- B_{diff}: Hintergrundmodell: maximale Differenzen

Hintergrunderkennung und Erstellung der Vordergrundmaske M:

$$M_t(x,y) = \begin{cases} & \text{wenn } I_t(x,y) < B_{min}(x,y) \\ 1 & \text{oder } I_t(x,y) > B_{max}(x,y) \\ & \text{oder } |I_t(x,y) - I_{t-1}(x,y)| > B_{diff}(x,y) \\ 0 & \text{sonst} \end{cases} \tag{2.26}$$

Kalman

Dieser Hintergrundalgorithmus modelliert jedes Pixel mit einem Kalmanfilter [12](S.304 ff).

Dabei wird der Farbwert (Intensitätswert) $I_t(x,y)$ sowie die Intensitätsänderung $D_t(x,y)$ zum vorigen Bild im Statusvektor $S_t(x,y)$ gespeichert. Aus den gemessenen Werten (dem aktuellen Bild I_t) und dem geschätzten Hintergrundmodellstatus, wird der neue

Hintergrundmodellstatus errechnet. Dabei wird berücksichtigt, ob das Pixel im letzten Frame dem Vordergrund oder dem Hintergrund zugeordnet war und der entsprechende Updateparameter verwendet. Zur Hintergrundklassifizierung wird die Differenz zwischen dem aktuellen Bild I_t und dem berechneten Hintergrundmodell $\widehat{S}_t$ gebildet und mit einem Schwellwert Φ verglichen. [13, 14]

$$D_t(x,y) = I_t(x,y) - I_{t-1}(x,y) \tag{2.27}$$

$$S_t(x,y) = (I_t(x,y), D_t(x,y)) \tag{2.28}$$

$$A = \begin{pmatrix} 1 & a_{1,2} \\ 0 & a_{2,2} \end{pmatrix} \qquad H = \begin{pmatrix} 1 & 0 \end{pmatrix} \qquad K_t(x,y) = \begin{pmatrix} k_1(x,y,t) \\ k_2(x,y,t) \end{pmatrix} \tag{2.29}$$

$$k_1(x,y,t) = k_2(x,y,t) = \begin{cases} \alpha & \text{für } M_{t-1}(x,y) = 1 \text{ (Vordergrund)} \\ \beta & \text{für } M_{t-1}(x,y) = 0 \text{ (Hintergrund)} \end{cases} \tag{2.30}$$

$$\widetilde{S}_t(x,y) = A \cdot \widehat{S}_{t-1}(x,y) \tag{2.31}$$

$$\widehat{S}_t = \widetilde{S}_t + K \cdot \Big(I_t(x,y) - H \cdot \widetilde{S}_t(x,y) \Big) \tag{2.32}$$

$$\Delta_t = \Big| I_t(x,y) - H \cdot \widehat{S}_t(x,y) \Big| \tag{2.33}$$

$$M_t(x,y) = \begin{cases} 1 & \text{für } \Delta_t(x,y) > \Phi \\ 0 & \text{sonst} \end{cases} \tag{2.34}$$

mit:

- Φ: Hintergrund-Schwellwert
- α: Vordergrund Kalman-Gewicht
- β: Hintergrund Kalman-Gewicht
- I_t: Bild zum Zeitpunkt t
- D_t: Differenzbild zwischen Bild I_t und I_{t-1}
- S_t: gemessener Statusvektor zum Zeitpunkt t
- $\widehat{S}_t$: berechneter Hintergrundmodellstatus zum Zeitpunkt t
- $\widetilde{S}_t$: geschätzter Hintergrundmodellstatus zum Zeitpunkt t
- Δ_t: Differenzbild zum Zeitpunkt t
- M_t: Hintergrundmasken zum Zeitpunkt t
- A: Kalman Systemmatrix
  $a_{1,2}, a_{2,2}$: Adaptionsparameter der Systemmatrix

H: Kalman Meßmatrix

K_t: Kalman Gewichtsmartix (Kalman gain)
$k_1(x, y, t), k_2(x, y, t)$: Gewichte des aktuellen Bildes (abhängig von Vordergrund-/Hintergundpixel)

2.1.2 Vergleiche dieser Hintergrundmodelle in der Literatur

Cheung und Kamath haben die Algorithmen in ihrer Arbeit mit vier Videosequenzen einer Straßenkreuzung getestet [1]. Dabei handelt es sich um eine Sequenz in hellem Tageslicht mit durchschnittlichem Verkehrsaufkommen, sowie Aufnahmen der selben Kreuzung bei Nebel und Schneefall. Die vierte Sequenz zeigt eine andere Kreuzung mit starkem Verkehrsaufkommen. Die Testsequenzen sind öffentlich zugänglich und stammen von der Webseite des Instituts für Algorithmen und Kognitive Systeme der Universität Karlsruhe [15].
Zur Beurteilung der Algorithmen verwenden Cheung und Kamath die zwei Kenngrößen Recall und Precision [1]. Recall ist ein Maß dafür, wie viele Vordergrundpixel als solche erkannt wurden, während Precision beschreibt, wie genau die Erkennung von Vordergrundbereichen ist.

$$\text{Recall} = \frac{\text{Anzahl der vom Alg. korrekt klassifizierten Vordergrundpixel}}{\text{Anzahl der Vordergrundpixel im Ground Truth}} \tag{2.35}$$

$$\text{Precision} = \frac{\text{Anzahl der vom Alg. korrekt klassifizierten Vordergrundpixel}}{\text{Anzahl der vom Alg. als Vordergrund klassifizierten Pixel}} \tag{2.36}$$

Blaunsteiner und Kampel testeten in [6] die Algorithmen auf zwei unterschiedlichen Datensätzen, einem Video von einem Parkplatz und einer Videosequenz eines Autobahnabschnitts. Zur Evaluierung wurden Referenzframes mit manuell gekennzeichneten Vordergrundobjekten bestimmt. Die Ergebnisse der Algorithmen auf diesen Referenzframes wurden mit den erstellten Vordergrundmasken verglichen und die Summe der False Positive und False Negative Pixel über alle Referenzframes ermittelt.

In [7] werden die Algorithmen auf einer Videosequenz eines Flughafen Aprons getestet, wobei eine qualitative Bewertung anhand von vier Kriterien (Empfindlichkeit gegenüber Bildrauschen bzw. Beleuchtungsänderung, Erkennungsrate und Laufzeit) durchgeführt und in einer Tabelle zusammengefasst wurde.

Frame Differencing

Sowohl [6] als auch [7] evaluierten eine Variante des Frame Differencing Algorithmus mit einem anschließenden Gaußfilter. Nur [1] evaluierte direkt den Frame Differencing Algorithmus, trotzdem waren die Ergebnisse eindeutig. Der Frame Differencing Algorithmus (mit oder ohne Gaußfilter) lieferte durchwegs sehr schlechte Resultate und wurde als sehr empfindlich gegenüber Bildrauschen beschrieben. In [7] werden die Ergebnisse des Frame Differencing Algorithmus, auch nach Aufbereitung mittels morphologischen Operationen, als unzureichend für ein Langzeithintergrundmodell bezeichnet.

Gleitender Mittelwert

Die Fehlerrate des Gleitenden Mittelwerts mit konstantem Schwellwert liegt bei [6] im Mittelfeld der getesteten Algorithmen. In [7] wurde sowohl ein Gleitender Mittelwert Algorithmus mit festem Schwellwert, als auch mehrere Varianten dieses Algorithmus (Color Mean and Variance) getestet. Die Erkennungsrate des Gleitenden Mittelwerts mit konstantem Schwellwert wurde als unzureichend eingestuft, wohingegen die Color Mean and Variance Varianten, die unterschiedliche Farbräume (RGB, normalisierter RGB, und HSV) verwenden, als gut eingestuft wurden. Diese Algorithmen benötigen keinen globalen Schwellwert (dieser wird aus der Varianz und einem konstanten Faktor berechnet) und können in Echtzeit berechnet werden. Sie sind robust gegenüber den meisten Beleuchtungsveränderungen, je nach Implementierung entstehen allerdings Löcher in der Segmentierung und teilweise fragmentierte Objekte.
Dieser Algorithmus wurde von Cheung und Kamath [1] nicht evaluiert.

Mixture of Gaussian (MoG)

Die AVITRACK Evaluierung [7] des Mixture of Gaussian Algorithmus ergab, dass dieser robust gegenüber den meisten Beleuchtungsveränderungen ist und dass nur vereinzelt Löcher in den Objektmasken entstehen. Allerdings ist dieser Algorithmus rechenintensiv, insbesondere während der Initialisierungsphase, und benötigt viel Speicherplatz im Vergleich zu den anderen Hintergrundmodellen. Cheung und Kamath [1] kamen zu ähnlichen Ergebnissen. Der Mixture of Gaussian Algorithmus lieferte die besten Resultate der Testreihe. Allerdings wiesen sie auch darauf hin, dass der Einstellung der Parameter besondere Aufmerksamkeit gewidmet werden muss, um optimale Ergebnisse zu erhalten, und dass der Algorithmus sensibel auf Helligkeitsschwankungen reagiert. Außerdem weist er eine relativ hohe Komplexität auf, was sich direkt in der Rechenzeit nieder-

schlägt.
Dieser Algorithmus wurde von Blauensteiner und Kampel [6] nicht evaluiert.

Median

Der Median Algorithmus wird in allen drei Arbeiten prinzipiell positiv beurteilt. In [6] lieferte der Median sogar die besten Resultate, und auch in [1] wurde er als knapper Zweiter beschrieben. Nur in [7] wird der einfache Median Algorithmus aufgrund des globalen Schwellwertes und aufgrund von Fehlern, die durch Geisterbilder langsam bewegter Objekte entstehen, nicht als Algorithmus zur Bewegungserkennung empfohlen. Stattdessen wird eine Variation des Medianalgorithmus (Median and Morphology) favorisiert, bei der ein Medianmodell mittels einer durch morphologische Operationen bereinigten Differenzmaske adaptiv aktualisiert wird.

Approximated Median

Cheung und Kamath haben gezeigt, dass die Ergebnisse des Approximated Median Algorithmus nahe an die eines echten Median Filters heranreichen. Nach [1] liefert der Mixture of Gaussians Algorithmus, knapp gefolgt vom Median Filter, die besten Ergebnisse. Danach folgt der Approximated Median Filter Algorithmus und weiters der Kalman Filter. Obwohl der Approximated Median Filter Algorithmus nicht so gut wie der Mixture of Gaussians Algorithmus oder der Median Filter ist, bietet er einen sehr guten Kompromiss zwischen Erkennungsqualität und Aufwand.
Dieser Algorithmus wurde von Blauensteiner und Kampel [6] sowie in der AVITRACK Evaluierung [7] nicht evaluiert.

Min-Max / W^4 Algorithmus

Der W^4 Algorithmus wurde in [7] nach der ersten Evaluierungsstufe nicht mehr weiter analysiert, da er zu viele False Positive Fehler produzierte. In [6] wurde eine ähnliche Fehlerrate wie die der Median Filter ermittelt. Genauer gesagt liegen die Ergebnisse zwischen denen des Median Filters mit konstantem Schwellwert und denen des Median Filters mit der Standardabweichung als adaptivem Schwellwert.
Dieser Algorithmus wurde von Cheung und Kamath [1] nicht evaluiert.

Kalman

Die Tests im Rahmen der AVITRACK Evaluierung [7] zeigten, dass dieser Algorithmus robust gegenüber den meisten Beleuchtungsveränderungen ist und in Echtzeit (Linux) implementierbar ist. Er zählt zu den empfohlenen Algorithmen nach der zweiten Evaluierungsphase. Im Gegensatz dazu wird der Kalman Filter von Cheung und Kamath [1] als der zweit schlechteste der evaluierten Algorithmen eingestuft, da er visuell die schlechteste Vordergrundmaske liefert und das Hintergrundmodell sehr leicht durch Vordergrundpixel gestört wird.
Dieser Algorithmus wurde von Blauensteiner und Kampel [6] nicht evaluiert.

2.2 Evaluierung der Hintergrundmodelle

Die Algorithmen wurden anhand von Videos evaluiert, die im Juli 2004 im Semmeringtunnel aufgenommen wurden und verschiedene Verkehrssituationen nachstellen. Diese Videos wurden mit den fest installierten Tunnelkameras aufgenommen und zur späteren Auswertung, Erstellung und Evaluierung von Bildverarbeitungsalgorithmen gespeichert. Es wurde eine Reihe von Testsequenzen ausgewählt, in der häufig auftretende Verkehrssituationen und ein paar Sonderfälle enthalten sind. Die verwendeten Sequenzen sind in Tabelle 2.1 aufgelistet und kurz beschrieben.

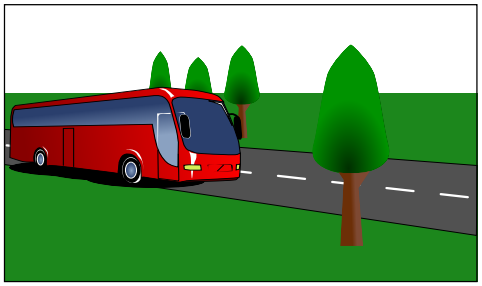
Sequenz	**Start** [Min:Sek]	**Ende** [Min:Sek]	**Kurzbeschreibung**
k503-3	00:10	01:50	mehrere PKW in Pannenbucht, dann Gegenverkehr
k503-4	00:40	02:10	LKW mit Nebelmaschine
k405-13	00:13	00:35	wenig Verkehr, zweispurig (in Fahrtrichtung)
k405-14	00:04	00:40	dichter Verkehr, einspurig (gegen Fahrtrichtung)
k405-19	00:06	00:50	mittlerer Verkehr (Gegenverkehr)
k405-25	00:25	00:50	Kolonnenverkehr, einspurig (gegen Fahrtrichtung)
k405-30	00:10	01:00	verlorenes Ladegut

Tabelle 2.1: Semmeringtunnel Testsequenzen

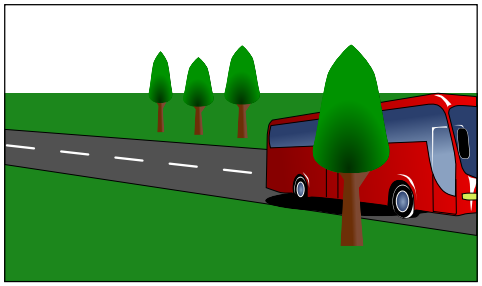
Die Algorithmen wurden in Hinblick auf eine Verwendung in einem Fahrzeugtrackingsystem mit typischen Szenen dieses Einsatzgebietes getestet. Für den ersten Test wurden die Sequenzen: *k503-3* und *k405-14* verwendet. Algorithmen, die schon bei diesen Testsequenzen schlecht abschnitten, wurden nach der ersten Phase aussortiert und nicht mehr weiter untersucht. Die anderen Algorithmen wurden in der zweiten Phase noch mit den

restlichen Sequenzen getestet. Bei der Bewertung handelt es sich um rein subjektive Eindrücke, die sich nach Betrachtung der Sequenzen ergeben (siehe Tabelle 2.2).

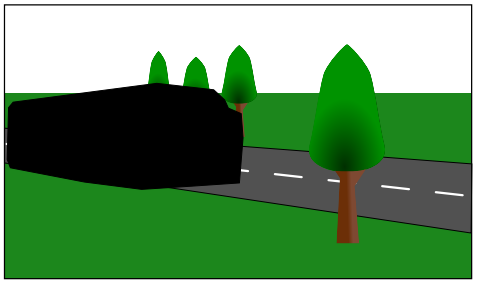
Zur Initialisierung der Hintergrundalgorithmen wurde ein Bild einer leeren Szene verwendet. War es nicht möglich ein Bild des Hintergrunds direkt aus der Videosequenz zu nehmen, weil der Hintergrund in keinem Frame vollständig sichtbar war, so wurde ein Hintergrundbild synthetisch erzeugt.
Dazu werden die Hintergrundbereiche mehrerer Frames kombiniert, sodass sich insgesamt ein vollständiges Hintergrundbild ergibt. (siehe Abbildung 2.2). Idealerweise wird über die Hintergrundbereiche mehrerer Frames ein Mittelwert gebildet. Die so entstandenen Hintergrundbilder sind natürlich nicht von der selben Qualität wie ein direkt aufgenommenes Bild. Zwischen Bereichen, die aus verschiedenen Frames stammen, kann es zu geringfügigen Helligkeitsschwankungen kommen. Dennoch stellt diese Methode eine relativ einfache Möglichkeit dar, ein oder mehrere Bilder für die Initialisierung eines Hintergrundmodells zu erstellen. Bei den verwendeten Testsequenzen wurde mit dieser Methode sichergestellt, dass zumindest die ersten 5 Frames eine leere Szene zeigen, damit der Hintergrundalgorithmus korrekt initialisiert werden kann.



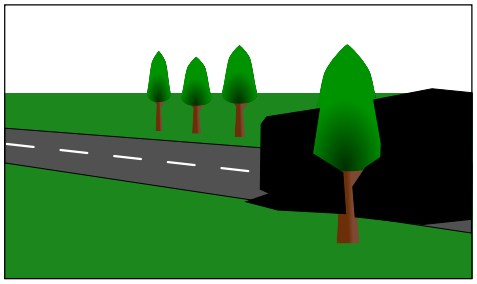
(a) Frame 1



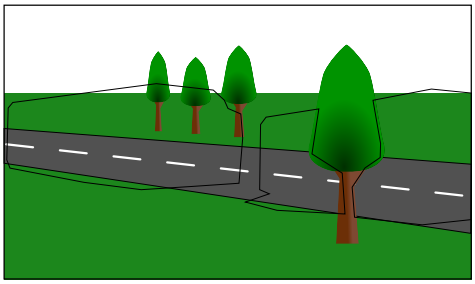
(b) Frame 2



(c) Frame 1, Vordergrund manuell ausmaskiert



(d) Frame 2, Vordergrund manuell ausmaskiert



(e) Manuell zusammengesetzter Hintergrund mit eingezeichneten Masken von Frame 1 und 2

Abbildung 2.2: Manuell erstelltes Hintergrundbild

2.2.1 Übersicht der Ergebnisse der einzelnen Hintergrundalgorithmen

In diesem Abschnitt werden die Ergebnisse der getesteten Hintergrundalgorithmen im Detail beschrieben, sowie in Tabelle 2.2 zusammengefasst.

Algorithmus	**Rauschen**	**Objekterkennung**	**Bemerkungen**
vollständiger Test:			
Gleitender Mittelwert	robust	relativ robust; Geisterbilder (false positive) bei zu großem Update-Parameter	
Median	robust	robust	hoher Ressourcenbedarf
Approximated Median	robust	robust; vereinzelte Geisterbilder bei langsamen Bewegungen	
Kalman	robust	robust; einige false positive Artefakte	Adaptionsparameter kritisch
nur erste Testphase:			
Frame Differencing	mäßig robust	nur Konturen in Bewegungsrichtung; diese relativ robust	kritischer Schwellwert; kein Hintergrundmodell
Min/Max W^4	empfindlich (Pumpen)	Fahrzeuge großteils nicht erkannt	kein externer Schwellwert

Tabelle 2.2: Ergebnisse der Hintergrundalgorithmen Evaluierung

Frame Differencing

Systembedingt wird das Innere von homogenen Objekten nicht detektiert. Der Frame Differencing Algorithmus ist primär zur Detektion von (schnellen) Bewegungen geeignet, erstellt aber kein echtes Modell des Hintergrunds und ist daher nur bedingt und mit entsprechenden Nachverarbeitungsschritten als Hintergrundalgorithmus einsetzbar.
Das Ergebnis des Frame Differencing Algorithmus wird in Abbildung 2.3 illustriert.

Gleitender Mittelwert

Um Rauschartefakte (speziell an Objektkanten) zu minimieren, sollten die Eingangsbilder in einem Vorverarbeitungsschritt (z.B. mittels Gauß-Filter) geglättet werden.

Wenn der Hintergrund nur selten zu sehen ist (wie eine Fahrbahn bei starkem Verkehr) liefert diese Methode ein sehr verschmiertes Hintergrundmodell. Für den reellen Einsatz sind sehr kleine Update-Parameter (α, Gewichtungsfaktor des aktuellen Bildes) für das gleitende Mittel notwendig, da sonst langsame Vordergrundobjekte den Hintergrund verschmieren oder gar vollständig in den Hintergrund integriert werden. Dies führt aber wieder zu Problemen bei der Implementierung auf einem DSP, da nicht alle Prozessoren Gleitkommaarithmetik unterstützen und bei der Verwendung von entsprechenden Softwarebibliotheken mit Leistungseinbußen zu rechnen ist. Eine mögliche Verbesserung stellt ein selektives Hintergrund-Update dar. Dadurch kann ein größerer Update-Parameter verwendet werden, was eine schnellere Adaption an Beleuchtungsveränderungen und eine einfachere Implementierung mittels Fixkommaarithmetik möglich macht.
Das Ergebnis des Gleitenden Mittelwert Algorithmus wird in Abbildung 2.5 illustriert.

Median

Ein Median Filter liefert die besten Ergebnisse, allerdings wird diese höhere Qualität des Hintergrundmodells durch einen unverhältnismäßig höheren Speicherplatzbedarf und Rechen- bzw. Sortieraufwand erkauft.

Ein Median-Hintergrundmodell ermöglicht eine robuste Modellierung des Hintergrunds. Wie auch beim gleitenden Mittelwert, treten bei langfristiger Verdeckung des Hintergrundes Artefakte im Hintergrundmodell auf. Diese sind aber nicht so stark ausgeprägt wie beim gleitenden Mittelwert.

Das Median-Modell ist theoretisch besser als Hintergrundmodell geeignet als das gleitende Mittel, da Vordergrundobjekte nicht in das Hintergrundmodell einfließen, wenn der Hintergrund zumindest die halbe Beobachtungszeit sichtbar ist. Allerdings werden bei vergleichbaren Zeiträumen auch deutlich mehr Ressourcen benötigt als bei anderen Algorithmen. Daher ist die Anzahl der Bilder der beobachteten Sequenz (n) stark begrenzt, was dazu führt, dass nur ein kurzfristiger Hintergrund modelliert werden kann. Bei der getesteten Implementierung werden bei langsamen Bewegungen Vordergrundartefakte in das Hintergrundmodell integriert.
Ein weiterer Nachteil ist, dass der Hintergrund mindestens über die Hälfte des Beob-

achtungszeitraums (Hintergrundsequenz) zu sehen sein sollte, andernfalls ist nicht sichergestellt, dass das Hintergrundmodell den tatsächlichen Hintergrund wiedergibt oder überhaupt nur annähert.
Das Ergebnis des Median Algorithmus ist in Abbildung 2.6 illustriert.

Approximated Median

Der Approximated Median hat Ähnlichkeiten mit dem gleitenden Mittelwert, der entscheidende Unterschied besteht jedoch darin, dass Pixel von (langsamen) Vordergrundobjekten mit hohem Farbwertunterschied zum Hintergrund beim Approximated Median nur langsam in das Hintergrundmodell übernommen werden, da die absolute Farbwertdifferenz keinen Einfluss auf den Updatewert hat, das heißt die entstehenden Fehler sind unabhängig von Farbwertunterschieden.

Das Hintergrundmodell des Approximated Median Filter wird bei viel Verkehr durch die teilweise Integration von Vordergrundobjekte verunreinigt. Allerdings sind die Fehler nicht so stark ausgeprägt wie beim Gleitenden Mittelwert. Andererseits entspricht die Qualität des erzeugten Hintergrundmodells nicht jener eines echten Median Filters. Der Approximated Median Filter stellt somit einen guten Kompromiss zwischen Detektionsqualität und Ressourcenbedarfs dar.
Das Ergebnis des Approximated Median Algorithmus ist in Abbildung 2.7 illustriert.

Min-Max / W^4 Algorithmus

Der Min-Max Algorithmus benötigt eine Lernphase, in der nur Hintergrund im Bild zu sehen ist. Während dieser Lernphase werden die Parameter (Minimum, Maximum und Abweichung) für jedes Pixel ermittelt.
Die vorliegende Implementierung zeigt ein deutliches Pumpen der Rauschartefakte und ein Großteil der Fahrzeuge wird nicht als Vordergrund erkannt. Daher wurde der Min-Max Algorithmus bereits in der ersten Testphase aussortiert.
Das Ergebnis des Min-Max Algorithmus ist in Abbildung 2.4 illustriert.

Kalman

Nach einer kurzen Initialisierungsphase liefert das Kalman Modell relativ gute Ergebnisse. Es zeigt sich, dass die Grauwert-Variante auf den Testbildern etwas stabiler läuft als die Farb-Variante.

Objekte werden gut erkannt, allerdings ziehen sie einen Schatten von Artefakten (Ghosts) hinter sich her (ähnlich der Frame Differencing Methode). Außerdem sind nur bewegte Objekte zu erkennen, da sich das Hintergrundmodell rasch adaptiert. Die Adaptionsparameter sind sehr kritische Parameter. Werden sie zu klein gewählt, hält sich ein fehlerhaftes Hintergrundmodell sehr lange. Zu große Parameter führen dazu, dass nur noch die Bewegungskonturen der Objekte erkannt werden, da die Objekte sehr rasch in den Hintergrund integriert werden. Bei einer häufigen Verdeckung des Hintergrundes (z.B. Fahrbahn bei Kolonnenverkehr) tritt ein ähnlicher Effekt wie beim gleitenden Mittelwert auf, der Hintergrund wird an dieser Stelle verschmiert.

Ränder von übersteuerten Lichtern (Fahrbahnrandleuchten, beleuchtete Hinweistafeln, etc.) werden häufig als Vordergrund klassifiziert. Dies ist vermutlich auf kleine Helligkeitsschwankungen bzw. Blendenregelung der Kamera zurück zu führen. Dadurch kann sich die Fläche des übersteuerten Bereiches geringfügig ändern. Da es sich dabei aber um isolierte Pixel handelt, können diese einfach durch morphologische Operationen eliminiert werden.
Das Ergebnis des Kalman Algorithmus ist in Abbildung 2.8 illustriert.

2.2.2 Zusammenfassung der Evaluierung der Hintergrundalgorithmen

Der Frame Differencing Algorithmus und der Min / Max W^4 Algorithmus lieferten in der ersten Testphase (Sequenzen: *k503-3* und *k405-14*) keine zufriedenstellenden Ergebnisse. (Der Frame Differencing Algorithmus erkennt nur bewegte Vordergrundbereiche und der Min / Max W^4 Algorithmus hat Schwierigkeiten bei der Erkennung der Fahrzeuge.) Daher wurden sie aus dem Pool der zu testenden Algorithmen gestrichen.
In der zweiten Testphase wurden die verbleibenden Algorithmen anhand den weiteren Sequenzen (siehe Tabelle 2.1) evaluiert.

Die besten Ergebnisse lieferte das Median Modell. Es ist robust gegenüber einfachem Bildrauschen und liefert eine gute Differenzmaske für die Objekterkennung. Allerdings ist der Median-Algorithmus sehr speicherplatzintensiv und rechenaufwendig. Er wurde zwar als Referenzalgorithmus weiter evaluiert, ist aber aufgrund des hohen Ressourcenbedarfs nicht als Hintergrundalgorithmus für ein DSP System geeignet.
Die übrigen Algorithmen der zweiten Testphase (Approximated Median, Gleitender Mittelwert und Kalman Filter) sind gleichermaßen robust, was das Bildrauschen betrifft, liefern allerdings nicht ganz so gute Ergebnisse bei der Differenzmaske.

Neben Schatten und globalen Beleuchtungsveränderungen durch Scheinwerfer können

reflektierende Objekte ein besonderes Problem darstellen. So sind z.B. Seitenmarkierungen, Absperrungen und Verkehrszeichen so beschaffen, dass sie das Licht in Richtung der Scheinwerfer zurück reflektieren. Ist nun die Kamera in Fahrtrichtung angebracht und schaut von schräg oben auf die Szene, so verursacht das Abblendlicht selbst zwar relativ wenig Störungen, allerdings reflektieren angeleuchtete Objekte das Licht genau in Kamerarichtung.

Bei übersteuerten Bereichen ist zwischen Scheinwerfern (insb. Abblendlicht, Rücklicht und Bremslicht) und Reflexionen zu unterscheiden. Dafür gibt es mehrere Möglichkeiten:

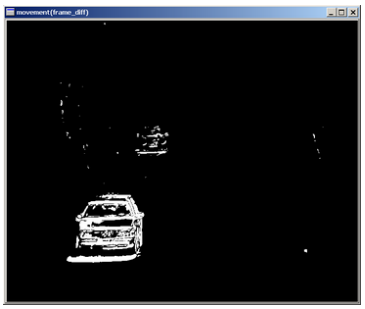
1. Die einfachste Lösung ist das entsprechende Setzen einer Region of Interest (ROI) sodass Reflexionen außerhalb der Fahrbahn (z.B. an Tunnelwänden, Verkehrszeichen, ...) nicht weiter behandelt werden.

2. Eine andere Variante besteht darin, die Form und Größe des übersteuerten Bereiches zu untersuchen. Wenn das umschließende Rechteck klein im Vergleich zu einem PKW ist und sich in der Nähe der Fahrbahn befindet, handelt es sich vermutlich um den Scheinwerfer selbst, da Reflexionen an der Fahrbahn oder an den Wänden in die Länge gezogen sind. Außerdem haben diese Reflexionen eine deutlich größere Fläche als typische Scheinwerfer.

3. Zusätzlich kann der Rand des übersteuerten Bereiches betrachtet werden. Weist dieser keine scharfen Kanten auf, so handelt es sich dabei um eine Reflexion an einer Fläche, da bei Reflexionen die Helligkeit am Rande des angestrahlten Bereiches kontinuierlich ab nimmt.

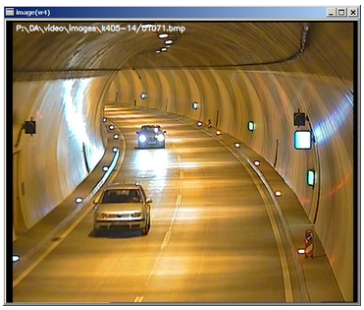
Wobei Methode 1 und 3 im Prototyp implementiert wurden.

Algorithmus erzeugt kein Hintergrundmodell

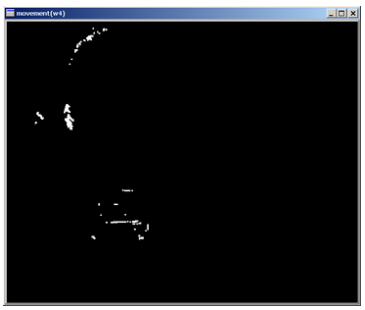


(a) Bild der Testsequenz (b) Hintergrundmodell (c) Differenzmaske

Abbildung 2.3: Frame Differencing

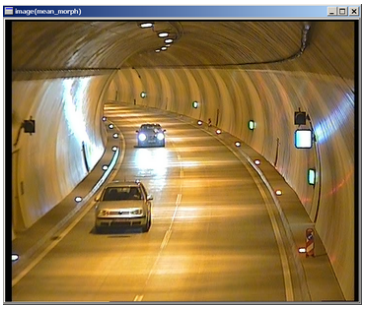


nicht als Einzelbild darstellbar

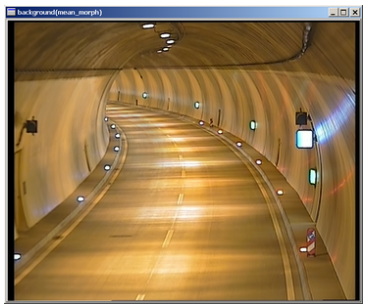


(a) Bild der Testsequenz (b) Hintergrundmodell (c) Differenzmaske

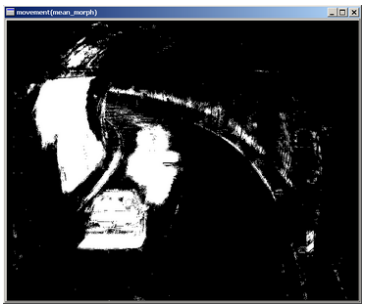
Abbildung 2.4: Min / Max W^4



(a) Bild der Testsequenz



(b) Hintergrundmodell



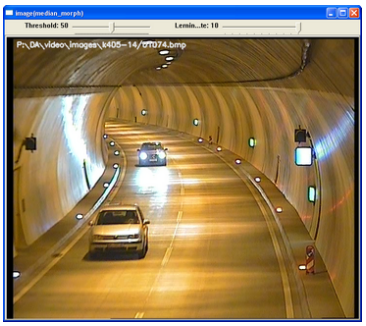
(c) Differenzmaske

(d) Bild der Testsequenz bei sehr viel Verkehr

(e) Hintergrundmodell bei sehr viel Verkehr

(f) Differenzmaske bei sehr viel Verkehr

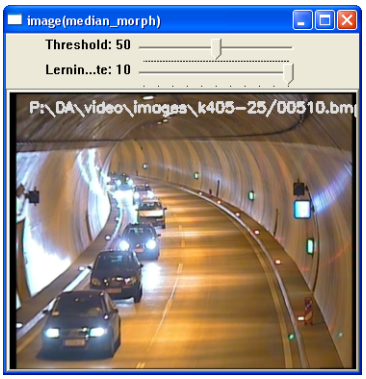
Abbildung 2.5: Gleitender Mittelwert



(a) Bild der Testsequenz

(b) Hintergrundmodell

(c) Differenzmaske

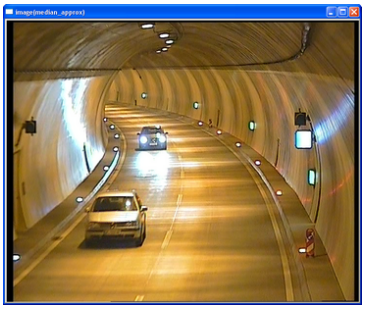


(d) Bild der Testsequenz bei sehr viel Verkehr

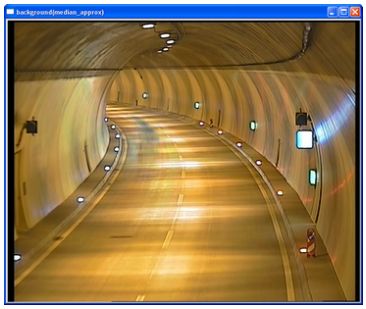
(e) Hintergrundmodell bei sehr viel Verkehr

(f) Differenzmaske bei sehr viel Verkehr

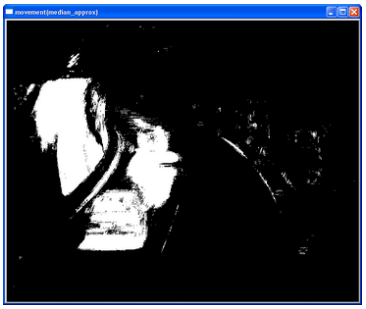
Abbildung 2.6: Median



(a) Bild der Testsequenz



(b) Hintergrundmodell



(c) Differenzmaske

(d) Bild der Testsequenz bei sehr viel Verkehr

(e) Hintergrundmodell bei sehr viel Verkehr

(f) Differenzmaske bei sehr viel Verkehr

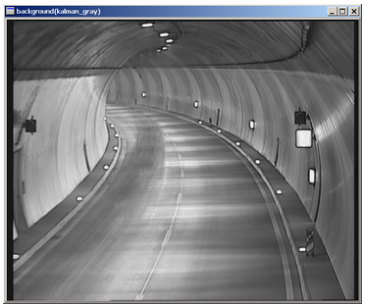
Abbildung 2.7: Approximated Median

(a) Bild der Testsequenz

(b) Hintergrundmodell

(c) Differenzmaske

(d) Bild der Testsequenz bei sehr viel Verkehr



(e) Hintergrundmodell bei sehr viel Verkehr

(f) Differenzmaske bei sehr viel Verkehr

Abbildung 2.8: Kalman Gray

2.3 Erweitertes Hintergrundmodell

In Kapitel 2.1 und 2.2 wurden verschiedene Hintergrundmodelle beschrieben und miteinander verglichen. Dabei hat der Median Algorithmus gute Ergebnisse geliefert. Allerdings sind bei der Auswahl des Hintergrundmodells nicht nur die Qualität des Algorithmus, sondern auch die Rahmenbedingungen der Zielplattform (Embedded Systems) zu berücksichtigen. In der Evaluierung hat sich gezeigt, dass der Approximated Median Algorithmus einen guten Kompromiss zwischen der Detektionsqualität und dem Ressourcenbedarf darstellt. Daher wird anstelle des Median Algorithmus mit seinem hohen Speicherplatzbedarf und Sortieraufwand der ressourcenschonende Approximated Median Algorithmus als Grundlage für das implementierte Hintergrundmodell verwendet.

2.3.1 Selektive Hintergrundaktualisierung

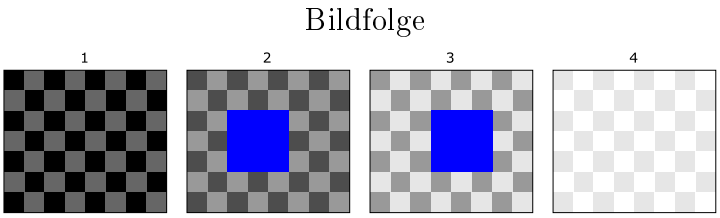
Ein wesentlicher Nachteil globaler Hintergrundmodelle, wie sie von einem Median bzw. Approximated Median Algorithmus erzeugt werden, ist, dass auch bewegte Objekte einen Einfluss auf das Hintergrundmodell haben. Dieser Einfluss kann zwar über Parameter beeinflusst werden (beim Median durch die Größe des verwendeten Bildbuffers n, beim Approximated Median durch den verwendeten Updatewert), allerdings muss hier immer ein Kompromiss zwischen der Reaktionsgeschwindigkeit auf veränderte Umgebungsbedingungen (z.B. Beleuchtungsänderung) und der Störungsminimierung eingegangen werden. Eine Lösung dieses Problems besteht darin, nur die Bereiche des Hintergrundmodells zu aktualisieren, die im aktuellen Frame auch den Hintergrund zeigen. Bei diesem Vorgehen müssen die Kriterien dafür, welcher Bereich als Hintergrund gilt, sorgfältig gewählt werden, da eine selektive Hintergrundaktualisierung eine Hebelwirkung hat. Wird ein Bereich als Hintergrund klassifiziert, der in Wirklichkeit ein bewegtes Vordergrundobjekt darstellt, so wird das Hintergrundmodell verfälscht und die selektive Hintergrundaktualisierung zeigt keinen positiven Effekt. Andererseits ist es möglich, dass ein Bereich, der den aktuellen Hintergrund im Frame darstellt, nicht als solcher erkannt wird und somit keine Anpassung des Hintergrundmodells an dieser Stelle erfolgt. Geschieht dies mehrmals hintereinander, ist damit zu rechnen, dass das Hintergrundmodell und der tatsächliche Hintergrund immer weiter auseinander driften. Ein weiterer Nachteil der selektiven Hintergrundaktualisierung ist, dass keine neuen Objekte in den Hintergrund integriert werden. Wird z.B. ein neues Verkehrszeichen aufgestellt, so wird dies durch ein selektives Update nicht automatisch in das Hintergrundmodell übernommen. Beide Probleme können durch eine Aktualisierung der Vordergrundbereiche mit geringer Gewichtung behoben werden. Durch die langsame Integration in das Hintergrundmodell werden fälschlich als Vordergrund klassifizierte Bereiche mit der Zeit wieder in das Hintergrundmodell übernommen. Andererseits wird dadurch das Hintergrundmodell

durch Vordergrundobjekte verfälscht. Dieser Fehler wird allerdings korregirt, sobald der echte Hintergrund wieder sichtbar wird.

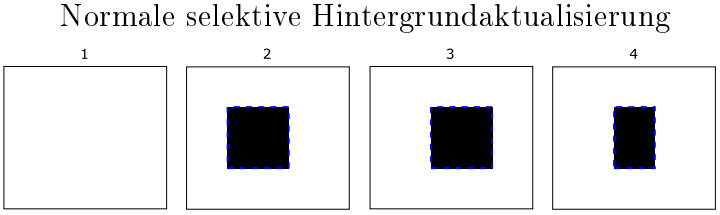
2.3.2 Globale Helligkeitsanpassung

Wie bereits erwähnt, ist es bei ungünstig gewählten Selektionskriterien möglich, dass das Hintergrundmodell und der tatsächliche Hintergrund auseinanderdriften. Dieser Effekt tritt aber auch auf, wenn die Vorder- / Hintergrunderkennung korrekt funktioniert, ein Bereich des Hintergrunds aber über einige Zeit verdeckt ist, während sich die Beleuchtung ändert bzw. die automatische Blendenregelung der Kamera die Blende nachjustiert und so in die globale Helligkeit eingreift. Nicht verdeckte Hintergrundbereiche werden entsprechend aktualisiert, die verdeckten Bereiche werden aber noch durch das alte Hintergrundmodell repräsentiert. Das kann dazu führen, dass nicht aktualisierte Hintergrundbereiche nicht mehr mit dem aktuellen Hintergrund übereinstimmen, wenn sie nicht mehr verdeckt sind, da sich inzwischen deren Helligkeit geändert hat.
Eine einfache, aber effektive Gegenmaßnahme ist, die globale Helligkeitsanpassung durch die durchschnittliche Helligkeitsveränderung zu approximieren. Dadurch wird ein Driften zwischen der Hintergrundhelligkeit verdeckter Bereiche und der tatsächlichen Helligkeit dieser Bereiche minimiert.

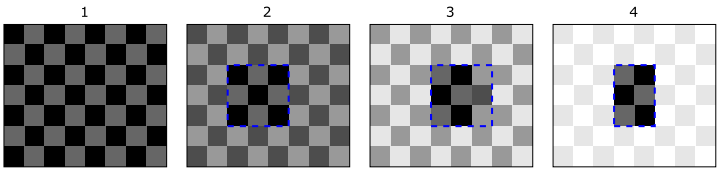
Abbildung 2.9 illustriert den Unterschied zwischen einer normalen selektiven Hintergrundaktualisierung und der helligkeitsanpassenden Hintergrundaktualisierung.
Die erste Zeile zeigt eine generierte Sequenz, wobei im ersten und letzten Frame der Hintergrund vollständig zu sehen ist. Im 2. und 3. Frame wird ein Teil des Hintergrundes von einem blauen Rechteck verdeckt, während er kontinuierlich heller wird. Danach folgt eine Zeile mit der Hintergrundmaske bzw. dem Hintergrundmodell für die selektive Hintergrundaktualisierung und für die erweiterte Hintergrundaktualisierung. Die Umrisse der Hintergrundmasken sind zur besseren Orientierung blau strichliert in das Hintergrundmodell eingezeichnet.
Wenn für das Hintergrundmodell keine Helligkeitsanpassung durchgeführt wird, kann es nach einer Verdeckung vorkommen, dass ein Bereich nicht mehr als Hintergrund erkannt wird, so wie es beim blau strichlierten Bereich in Abbildung 2.9(c), Frame 4 der Fall ist. Der mittlere Bereich wurde sowohl in Frame 2 als auch in Frame 3 verdeckt. Dieser Bereich wurde daher nicht aktualisiert und enthält noch das ursprüngliche Hintergrundmodell, während der tatsächliche Hintergrund immer heller wurde. Das Beispiel ist so gewählt, dass ein Bereich dem Hintergrund noch richtig zugeordnet werden kann, wenn er für ein Frame verdeckt war. Wurde der Bereich allerdings länger verdeckt, ist der Helligkeitsunterschied zwischen ursprünglichem Hintergrund wie er im Modell abgebildet wurde und aktuellem Hintergrund so groß, dass er von einem einfachen Schwellwertverfahren nicht mehr als Hintergrund erkannt wird und somit auch nicht mehr mittels



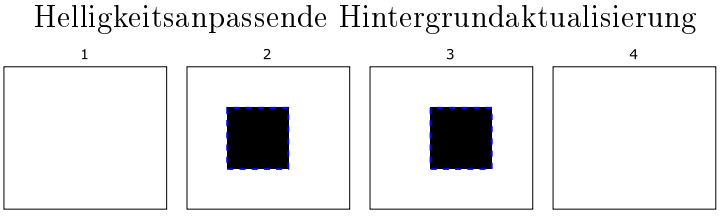
(a) aufeinanderfolgende Frames, bei denen sich die globale Hintergrundhelligkeit ändert, im 2. und 3. Frame wird ein Teil des Hintergrundes von einem blauen Quadrat verdeckt



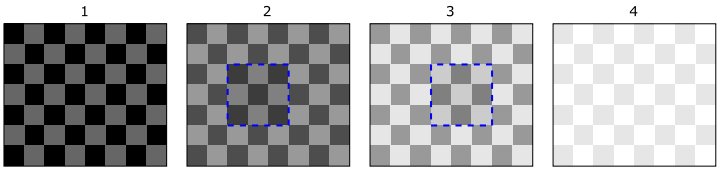
(b) Hintergrundmasken bei normaler selektiver Hintergrundaktualisierung



(c) Hintergrundmodell bei normaler selektiver Hintergrundaktualisierung



(d) Hintergrundmasken bei helligkeitsanpassender Hintergrundaktualisierung



(e) Hintergrundmodell bei helligkeitsanpassender Hintergrundaktualisierung

Abbildung 2.9: Vergleich zwischen normaler selektiver Hintergrundaktualisierung und helligkeitsanpassender Hintergrundaktualisierung

Median bzw. Approximated Median Algorithmus angepasst wird.
Bei der helligkeitsanpassenden Hintergrundaktualisierung (Abbildung 2.9(e)) ist deutlich zu sehen, dass verdeckte Bereiche nicht eingefroren werden, sondern ihre Helligkeit entsprechend der globalen Helligkeitsänderung geändert wird. Selbst wenn die Helligkeitsapproximation nicht die tatsächliche Helligkeit des Hintergrundes liefert, können ein zu schnelles Abdriften der Helligkeit und die damit verbundenen Probleme verhindert werden (siehe Frame 4 der normalen selektiven Hintergrundaktualisierung in Abbildung 2.9(c)).

Die Entscheidung, ob ein Bereich als Hintergrund klassifiziert wird, wird aufgrund der Differenz zwischen dem aktuellen Bild und dem aktuellen Hintergrundmodell getroffen. Dazu wird für jedes Pixel und jeden Farbkanal des Farbmodells (z.B. RGB), die absolute Differenz zwischen aktuellem Frame und Hintergrundmodell berechnet und mit der Standardabweichung σ_c des Farbkanals normiert. Als Differenzmaske wird pro Pixel der Maximalwert der normierten Farbdifferenzen verwendet.

$$\Delta(x,y) = \max_{c=1}^{3} \left(\frac{|I(x,y)_c - B(x,y)_c|}{\sigma_c} \right) \tag{2.37}$$

mit:

- c: Farbkanalindex
- I_c: Farbkanal c des aktuellen Bildes
- B_c: Farbkanal c des Hintergrundmodells
- σ_c: Standardabweichung des Farbkanals c
- Δ: normierte Differenzmaske

Um zu vermeiden, dass Ausreißer oder Vordergrundpixel die Berechnung der globalen Helligkeitsänderung und der neuen Standardabweichung verfälschen, werden dazu nur Pixel verwendet, deren Farbkanaldifferenz kleiner als das Dreifache der zuletzt berechneten Standardabweichung ist. Unter der Annahme, dass das Bildrauschen mit der ermittelten Standardabweichung normal verteilt ist, werden somit 99,7% (siehe [16] Seite 42–44) aller durch Rauschen verfälschten Hintergrundpixel berücksichtigt.

2.4 Segmentierung und Labeling

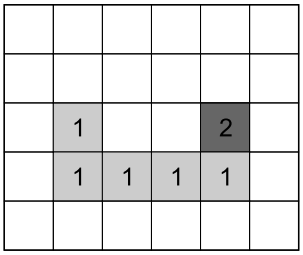
Unter Bildsegmentierung versteht man die Unterteilung eines Bildes in zusammenhängende Regionen, wobei in einer Region Pixel mit gemeinsamen Eigenschaften (z.B. Hel-

ligkeit, Farbe, ...) zusammengefasst sind [17](Seite 163ff) [18](Seite 337).

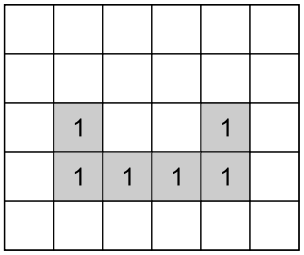
Durch das Labeling werden den einzelnen Segmenten eindeutige Labels zugeordnet, um diese bei der weiteren Verarbeitung eindeutig referenzieren zu können. Im einfachsten Fall werden die Segmente dazu durchnummeriert.

Im Prototyp wird ein Pixel-Klassifikator (4er-Nachbarschaft) auf die Vordergrundmaske des Hintergrundmodells angewandt.
Als Homogenitätskriterium wird der Mittelwert der Farbkanäle der Pixel verwendet. Liegt der Wert eines Pixels unter einem Schwellwert und der Wert des Nachbarpixels auf oder über der Schwelle, oder umgekehrt, so werden die Pixel unterschiedlichen Segmenten zugeordnet.

Aus Effizienzgründen wird ein 2-Pass Run-Label Pixelsegmentierungs-Algorithmus eingesetzt. Der implementierte Algorithmus arbeitet ähnlich dem von Hayashi et al. [19] beschriebenen Algorithmus mit einem 2×2 Fenster, allerdings wird dieses Fenster als look-behind Fenster (das aktuelle Pixel ist das untere, rechte Element) verwendet und nicht wie bei Hayashi et al. [19] als look-ahead Fenster. Dadurch steigt zwar die Anzahl der Fragmentnummern bei U-Formationen (siehe Abbildung 2.10) bzw. Linien von links unten nach rechts oben, dafür kann vollständig auf den Run-Label-Buffer verzichtet werden.



(a) Prototyp: Dem Vordergrundpixel rechts oben wird eine neue Fragmentnummer zugewiesen, die in der nächsten Zeile mit der Fragmentnummer des unteren Nachbarpixels verknüpft wird.



(b) Hayashi et al. : Duch das look-ahead Fenster wird dem Vordergrundpixel rechts oben die selbe Fragmentnummer wie dem unteren Nachbarpixel zugewiesen.

Abbildung 2.10: Fragmentzuordnung von Run-Label Segmentierungs-Algorithmus

In dem look-behind Algorithmus wird das Bild zeilenweise abgearbeitet und jedes Pixel mit dem linken und dem oberen Nachbarpixel verglichen. Ist die Homogenitätseigenschaft für keinen der beiden Vergleiche erfüllt, so wird dem Pixel eine neue Fragmentnummer zugeordnet. Ist die Homogenitätseigenschaft für genau einen der Vergleiche erfüllt, so wird dem Pixel die Fragmentnummer des entsprechenden Nachbarpixels zugeordnet. Ist sie für alle drei Pixel erfüllt und haben die Nachbarpixel die gleiche Fragmentnum-

mer, so wird diese auch dem betrachteten Pixel zugewiesen. Haben die Nachbarpixel unterschiedliche Fragmentnummern, so wird dem Pixel die niedrigere der beiden Fragmentnummern zugewiesen, und die höhere Fragmentnummer wird mit der niedrigeren verknüpft. Im zweiten Durchgang werden alle Fragmentnummern die zu einem Segment gehören zu einer eindeutigen Labelnummer vereinigt und jedem Pixel wird die endgültige Labelnummer, anstelle der temporären Fragmentnummer, zugewiesen.

Bei den Beispielbildern der verwendeten Testszenen (siehe Abbildung 6.3 und 6.4) beträgt die durchschnittliche Anzahl der temporären Fragmente 1203 bzw. 2113 im Gegensatz zu 593 bzw. 832 nach dem Algorithmus von Hayashi et al. Um diese Zahlen vergleichen zu können, muss man aber auch berücksichtigen, dass der Hayashi Algorithmus nur auf einer Binärmaske arbeitet, alle 0-Pixel als Hintergrund klassifiziert und keine eingeschlossenen Segmente detektiert. Daher erkennt er in den Beispielbildern im Durchschnitt auch nur 575 bzw. 745 Segmente im Gegensatz zu 655 bzw. 1168 Segmenten beim look-behind Algorithmus. Der Hayashi Algorithmus benötigt etwas weniger als die Hälfte an Speicher für die temporären Fragmentnummern, erkennt aber, aufgrund eines anderen Segmentierungskriteriums, im Durchschnitt um 1/4 weniger Segmente. Insgesamt kann der temporäre Fragmentnummernspeicher beim Hayashi Algorithmus kleiner gewählt werden, allerdings wird ein zusätzlicher Run-Label-Buffer verwendet, der mindestens die Größe einer halben Bildzeile haben muss, wodurch sich der Unterschied im Speicherbedarf der beiden Algorithmen deutlich verringert bzw. ausgleicht.

Wie sich bei Experimenten gezeigt hat, ist eine zusätzliche Nachverarbeitung notwendig, um aus den Ergebnissen des Segmentierung und Labeling Schritts einen Objektblob (das heißt die Menge der Segmente, die ein Objekt bestmöglich überdecken) zu erhalten (siehe Kapitel 4.2.1). Dafür ist es aber notwendig, dass

1. aneinandergrenzende Vordergrundbereiche mit unterschiedlichen Homogenitätseigenschaften als zwei getrennte Segmente behandelt werden (siehe das zuvor beschriebene Homogenitätskriterium), und

2. eingeschlossene Hintergrundbereiche (z.B. Fenster oder Bereiche des Fahrzeuges ohne Struktur, die fälschlicherweise als Hintergrund klassifiziert wurden) als normale Segmente, mit einer eigenen Labelnummer, erkannt werden.

Beide Bedingungen werden von dem Algorithmus, den Hayashi et al. in [19] beschreiben, nicht erfüllt, daher wird für den Prototypen der zuvor beschriebene look-behind Algorithmus verwendet.

2.5 Zusammenfassung

Aufgrund der Ergebnisse der Hintergrundevaluierung kann man zusammenfassen, dass von den getesteten Algorithmen der Median Algorithmus (bzw. dessen Derivate) und der Mixture of Gaussian Algorithmus die qualitativ besten Ergebnisse liefern. Der Approximated Median Algorithmus liefert erwartungsgemäß etwas schlechtere Resultate als die beiden zuvor genannten Algorithmen.

Bezieht man allerdings die Algorithmuskomplexität in die Überlegung mit ein, so ist der Approximated Median Algorithmus aufgrund seiner Einfachheit ein guter Kompromiss. Wie die beiden Kriterien Hintergrundqualität und Komplexität gewichtet werden ist von Projekt zu Projekt unterschiedlich.
In der vorliegenden Arbeit fiel die Wahl auf den Approximated Median Algorithmus, da dieser sich durch seine geringe Komplexität und niedrigen Ressourcenbedarf gut für eine DSP Implementierung eignet. Geringfügige Fehler im Hintergrundmodell werden im laufenden Betrieb durch einen unabhängigen Trackingalgorithmus kompensiert und können somit akzeptiert werden.

Um die Qualität des Hintergrundmodells zu verbessern wurden folgende Erweiterungen vorgenommen:

- *Selektive Hintergrundaktualisierung:* Es werden nur jene Bereiche des Hintergrundmodells aktualisiert, die sicher als Hintergrund erkannt wurden.

- *Globale Helligkeitsanpassung:* Für jene Bereiche, die nicht aktualisiert werden, wird eine globale Helligkeitsanpassung durchgeführt, um ein Auseinanderdriften zwischen Hintergrundmodell und tatsächlichen Hintergrund zu vermeiden.

Als Segmentierungsalgorithmus wird ein 2-Pass Run-Label Algorithmus verwendet. Dabei werden zuerst temporäre Fragmente der Vordergrundmaske klassifiziert und zusammenhängende Fragmente identifiziert. Nach Erstellung der endgültigen Labelnummern werden diese in einem zweiten Durchlauf anstelle der Fragmentnummern in die Labelmaske eingetragen.

3 Objektmodell

Der erste Teil dieses Kapitels beschreibt die Umrechnung zwischen Welt und Bildkoordinaten, wie sie in den nächsten Kapiteln benötigt wird.

Anschließend wird das Fahrzeugobjektmodell, so wie es im Prototyp verwendet wird, vorgestellt. Dabei wird insbesondere auf Farbhistogramme eingegangen, da diese die Grundlage des Trackingalgorithmus bilden.

Der letzte Teil dieses Kapitels beschäftigt sich mit der Objektdetektion. Das heißt mit der Initialisierung eines neuen Fahrzeugobjektes sobald ein noch nicht getracktes Objekt detektiert wird.

3.1 Welt-/Bildkoordinaten

Wird ein Objekt in einem Frame erkannt, so können Größe, Mittelpunkt und Fußpunkt der Bounding-Box, Schwerpunkt der Objektmaske, etc. in Bildkoordinaten ermittelt werden.

Für einfache Berechnungen und Schätzungen wie z.B. relative Größenänderung oder einfache Positionsvorhersagen für den nächsten Frame, sind Bildkoordinaten ausreichend, um aber absolute Zahlen für Position, Größe, Geschwindigkeit, etc. errechnen zu können, müssen die Größen erst in das Weltkoordinatensystem transferiert werden. Umgekehrt müssen bekannte Punkte (Referenzpunkte, Eckpunkte, etc.) des Weltkoordinatensystems in Bildkoordinaten abgebildet werden, um sie in dem aktuellen Bild einzeichnen zu können.

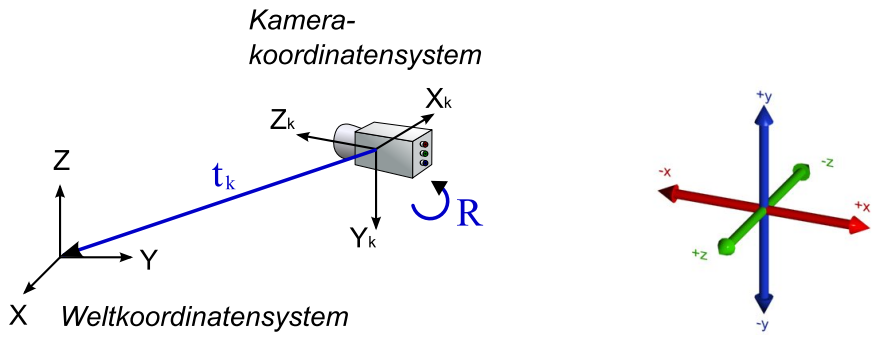
Weltkoordinatensystem Das Weltkoordinatensystem ist ein 3-dimensionales, kartesisches Koordinatensystem, das über die reale Welt gelegt wird, um aus Referenzpositionen und absoluten Maßen die Positionen, Geschwindigkeiten und weitere Daten von Objekten bestimmen zu können. Als Maßeinheit wird Meter als die SI-Einheit der Länge

verwendet. Ist in der betrachteten Szene eine eindeutige Grundebene vorhanden, so wird diese mit der X/Y-Ebene identifiziert, wobei die Z-Koordinate der Höhe entspricht.

Kamerakoordinatensystem Das Kamerakoordinatensystem ist ein 3-dimensionales, kartesisches Koordinatensystem, das an einer Kamera ausgerichtet ist, das heißt sein Ursprung liegt im Projektionszentrum der Kamera. Die Z_k-Achse ist in deren Blickrichtung ausgerichtet und entspricht daher der optischen Achse [2](Seite 47f) (siehe Abbildung 3.1(a)).

Bildkoordinatensystem Das Bildkoordinatensystem ist das 2-dimensionale Koordinatensystems eines Bildes im Computer. Jeder Bildpunkt hat eindeutige Koordinaten, die in Pixel gemessen werden. Der Ursprung liegt in der linken oberen Bildecke, die x-Koordinate entspricht der Spaltennummer und die y-Koordinate der Zeilennummer (jeweils bei 0 beginnend).

Bei all diesen Koordinatensystemen handelt es sich um rechtshändige Koordinatensysteme (siehe Abbildung 3.1(b)).



(a) Lagebeziehung zwischen Weltkoordinatensystem und Kamerakoordinatensystem

(b) dreidimensionales rechtshändiges Koordinatensystem [20]

Abbildung 3.1: Koordinatensysteme und Lagebeziehungen

3.1.1 Weltkoordinaten in Bildkoordinaten Transformation (3D nach 2D)

Die Weltkoordinaten in Bildkoordinaten Transformation setzt sich aus zwei Transformationen zusammen:

1. Translation und Rotation, um Weltkoordinaten in Kamerakoordinaten zu transformieren

2. Projektion, um einen Punkt in Kamerakoordinaten in Bildkoordinaten zu projizieren

Translation und Rotation

Dieser Abschnitt beschreibt, wie die Position eines reellen Objektes (als Weltkoordinaten $(X, Y, Z)^t$ gegeben) in Bildkoordinaten $(X_k, Y_k, Z_k)^t$ transformiert werden kann:

Als erster Schritt werden die Koordinaten eines Punktes P vom Weltkoordinatensystem $(X, Y, Z)^t$ in das Kamerakoordinatensystem $(X_k, Y_k, Z_k)^t$ transferiert. Die Lagebeziehung (externe Kameraparameter) zwischen Welt- und Kamerakoordinatensystem, sowie die für die Transformation notwendige Rotation $\mathbf{R}$ und Translation $\mathbf{t_k}$ werden in Abbildung 3.1(a) illustriert [2](Seite 48-50)) [21](Seite 315–317). Die Rotationsmatrix entspricht dabei den Einheitsvektoren des Weltkoordinatensystems gemessen in Kamerakoordinaten.

Durch homogene Koordinaten kann die Kombination aus Rotation und Translation als reines Matrix-Vektor-Produkt geschrieben werden [21](Seite 315–317).

Falls nicht anders erwähnt, werden homogene Koordinaten in dieser Arbeit immer normiert betrachtet, um eine eindeutige Darstellung zu erhalten. In Gleichungen wird die Gleichheit gegebenenfalls durch einen Normierungsfaktor (λ) sichergestellt.

Projektion

Geht man von einem einfachen Lochkameramodell aus, entspricht die Aufnahme eines Objektes mit der Kamera einer perspektivischen Projektion (Zentralprojektion) auf die Bildebene. Dabei entspricht das Projektionszentrum dem Brennpunkt des Objektivs und die Projektionsebene der Bildebene der Kamera.

Die Projektion eines Punktes von Kamerakoordinaten $(X_k, Y_k, Z_k)^t$ auf Bildkoordinaten $(x, y)^t$ kann mit Hilfe der *Kalibrierungsmatrix* $\mathbf{K}$ berechnet werden [21](Seite 314–315). Die Kalibrierungsmatrix setzt sich aus den internen Kameraparametern zusammen: Fo-

kale Länge, Lage des Bildhauptpunktes sowie Skalierung und Verzerrung.

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = (\mathbf{K} \,|\, \mathbf{0_{3x1}}) \begin{pmatrix} X_k \\ Y_k \\ Z_k \\ 1 \end{pmatrix} \tag{3.1}$$

$$\mathbf{K} = \begin{pmatrix} f/\mu_x & s & c_x \\ 0 & f/\mu_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

mit:

f: Abstand zwischen Projektionsebene und Projektionszentrum (Koordinatensystem-Ursprung)

(c_x, c_y): Bildhauptpunkt in Bildkoordinaten

(μ_x, μ_y): Pixelgröße in Kamerakoordinaten

s: Scherungsfaktor

Kameramatrix

Kombiniert man die Transformation von Weltkoordinaten in Kamerakoordinaten mit der Abbildung in das Bildkoordinatensystem, so erhält man die Gleichung zur Welt- in Bildkoordinatentransformation

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{P} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.3}$$

und die Kameramatrix $\mathbf{P}$ [21](Seite 316–317)

$$\mathbf{P} = \mathbf{K} \cdot \left(\mathbf{R}|\mathbf{t_k}\right) \tag{3.4}$$

mit:

(X, Y, Z): Koordinaten eines Punktes im Weltkoordinatensystem

(x, y): Koordinaten des Punktes im Bildkoordinatensystem
$\mathbf{P}$: Kameramatrix
λ: Normierungsfaktor für homogene Koordinaten
$\mathbf{I}$: 3×3 Einheitsmatrix
$\mathbf{K}$: Kalibrierungsmatrix
$\mathbf{R}$: Rotationsmatrix von Weltkoordinaten in Kamerakoordinaten
$\mathbf{t_k}$: Translationsvektor zwischen Weltkoordinatensystem und Kamerakoordinatensystem (Ursprung des Weltkoordinatensystems in Kamerakoordinaten)

3.1.2 Bildkoordinaten in Weltkoordinaten Transformation (2+1D nach 3D)

Mit der Gleichung 3.3 auf der vorherigen Seite kann nun jedes bekannte Objekt im Bild eingezeichnet bzw. markiert werden. Für eine Tracking Anwendung ist aber der umgekehrte Schritt, aus Bildkoordinaten die echte Position in Weltkoordinaten zu berechnen, die eigentliche Aufgabe. Das heißt, es ist die Lösung des inversen Problems gesucht.

Aufgrund der unterschiedlichen Dimensionen der Vektorräume ist offensichtlich, dass hier keine eindeutige Lösung existiert. Wenn ein Punkt aus dem Bildkoordinatensystem in das Weltkoordinatensystem abgebildet werden soll, müssen zusätzliche Informationen vorhanden sein bzw. Annahmen getroffen werden. Eine häufig verwendete Annahme ist, dass sich Objekte nur auf einer Ebene im Raum (z.B. Fahrbahn) bewegen.
Daher verwenden viele Trackingsysteme einen Basis- bzw. Fußpunkt des zu verfolgenden Objekts, mit der Annahme, dass sich dieser Basispunkt auf einer bekannten Referenzebene befindet, wie es zum Beispiel in den Arbeiten von Koler et al. [22] im Rahmen von *PATH* (California Partners for Advanced Transit and Highways - University of California, Berkeley) sowie in Hu et al. [23] beschrieben wird. Als Referenzebene wird dabei die Fahrbahn bzw. der Fußboden verwendet, mit der natürlichen Annahme, dass sich Fahrzeuge und Personen bodengebunden bewegen. Mathematisch entspricht dies einer eindeutigen Transformation von der 2-dimensionalen Bildebene auf eine 2-dimensionale Referenzebene Π, die in einen 3-dimensionalen Raum (Welt) eingebettet ist. Diese Transformation wird Homographie oder Kollineation genannt.

Entspricht die Referenzebene Π einer durch zwei Basisvektoren des Weltkoordinatensystems aufgespannten Ebene (im Allgemeinen die X/Y-Ebene), so lässt sich die Homographie direkt aus der Kameramatrix $\mathbf{P}$ ermitteln. Dabei wird vorausgesetzt, dass $\mathbf{P}$ bereits richtig skaliert ist, so dass die Gleichung 3.3 korrekt ist.
Die Homographie $\mathbf{H_z}$ zwischen der X/Y-Ebene des Weltkoordinatensystems und der

Bildebene besteht aus der 1., 2. und 4. Spalte von $\mathbf{P}$. Die 3. Spalte bzw. Z-Spalte wird durch $Z \equiv 0$ eliminiert.

$$\mathbf{H_z} = \begin{pmatrix} | & | & | \\ \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_4 \\ | & | & | \end{pmatrix} \tag{3.5}$$

Mittels $\mathbf{H_z}$ kann ein Punkt $A = (X, Y, Z = 0)$ der Referenzebene Π eindeutig auf einen Punkt $a = (x, y)$ der Bildebene abgebildet werden (vlg. Abbildung 3.2).
Ist die durch $\mathbf{H_z}$ beschriebene Abbildung bijektiv, so kann $\mathbf{H_z}$ invertiert werden $\mathbf{H_z^{-1}}$ und jeder Punkt der Bildebene kann eindeutig auf die Referenzebene Π abgebildet werden. Dieser Vorgang nennt sich Rückprojektion:

$$\mathbf{H_z} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.6}$$

$$\frac{1}{\lambda} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \mathbf{H_z^{-1}} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.7}$$

Soll der Punkt a nicht auf die Ebene Π projiziert werden, sondern auf eine Parallelebene Π' mit der Höhe Z (Abbildung 3.2), so kann für Π' eine neue Homographie $\mathbf{H_z}'$ berechnet werden.

Ist auch $\mathbf{H_z}'$ bijektiv, so existiert eine Inverse $\mathbf{H_z}'^{-1}$ mit der a in den Punkt A' auf der Ebene Π' abgebildet wird.

$$\mathbf{H'_z} \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.8}$$

$$\frac{1}{\lambda} \underbrace{\begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix}}_{A'} = \mathbf{H_z'^{-1}} \underbrace{\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}_{a} \tag{3.9}$$

Um Punkte der Bildebene auf verschiedene Ebenen im Weltkoordinatensystem projizieren zu können, müssten für jede Ebene eine eigene Homographie und dazu passende Inverse berechnet werden. Die unterschiedlichen Homographien und deren Inverse können zwar direkt aus der Kameramatrix $\mathbf{P}$ berechnet werden, allerdings ist dies bei

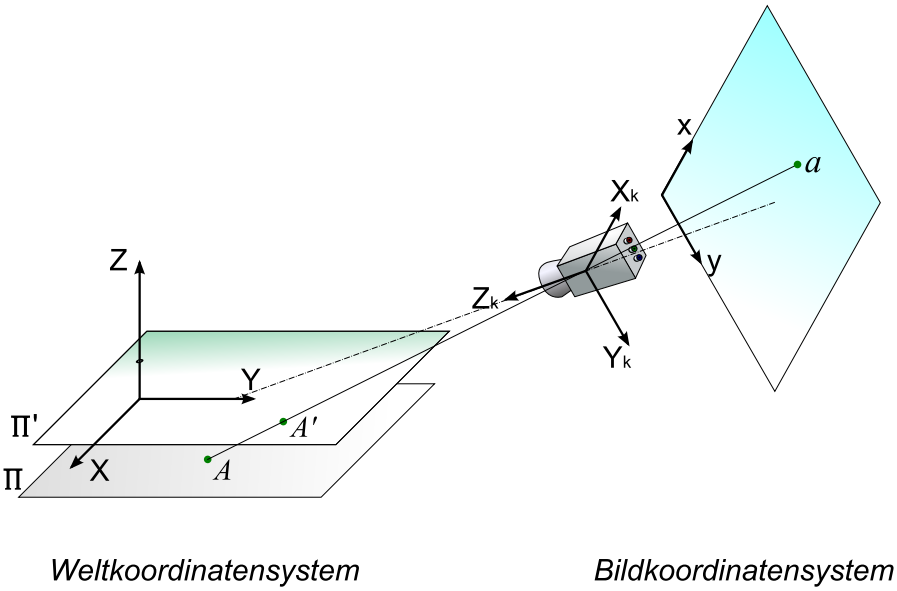


Abbildung 3.2: Skizze der Abbildung eines Punktes zwischen einer Ebene Π' parallel zur Referenzebene Π und der Bildebene

Echtzeitanwendungen mit beschränkten Ressourcen nicht durchführbar. Eine Vorausberechnung der inversen Homographien für die Rückprojektion ist zwar prinzipiell möglich, aber unpraktikabel und beschränkt die Rückprojektion auf fest vorgegebene Ebenen.

Wie die folgenden mathematischen Überlegungen zeigen werden, ist es möglich die Rückprojektion auf eine beliebige Ebene parallel zur X/Y-Ebene mit der Höhe Z auf eine Projektion auf die X/Y-Ebene (Referenzebene Π) und einen Korrekturvektor $\mathbf{O_z}^{-1} \cdot Z$ zurückzuführen.

(X, Y, Z): Koordinaten eines Punktes im Weltkoordinatensystem

(x, y): Koordinaten des Punktes im Bildkoordinatensystem

$\mathbf{P}$: Kameramatrix

$\mathbf{P}_i$: Spaltenvektoren der Kameramatrix

$\mathbf{H_z}$: Homographiematrix zwischen der X/Y-Ebene des Weltkoordinatensystems und der Bildebene

$\mathbf{H_z}^{-1}$: inverse Homographiematrix zur Rückprojektion auf die X/Y-Ebene des Weltkoordinatensystems

$\mathbf{O_z}^{-1}$: Korrekturvektor zur Anpassung der Rückprojektion an eine beliebige Orthogonalebene der Z-Achse

λ: Skalierungsfaktor der homogenen Koordinaten

Ausgangsbasis ist die Projektion eine beliebigen Punktes im Weltkoordinatensystem (X, Y, Z) auf einen Punkt (x, y) der Bildebene mit Hilfe der Kameramatrix $\mathbf{P}$, wie es in Gleichung 3.3 hergeleitet wurde:

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} | & | & | \\ \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_4 \\ | & | & | \end{pmatrix}}_{\mathbf{H_z}} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} + \begin{pmatrix} | \\ \mathbf{P}_3 \\ | \end{pmatrix} Z \tag{3.10}$$

Durch arithmetische Umformungen kann die Gleichung in die Form der Rückprojektion auf die X/Y Basisebene (3.9) minus eines Korrekturvektors gebracht werden.

$$\lambda \mathbf{H_z}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} - \underbrace{\mathbf{H_z}^{-1} \mathbf{P}_3}_{\mathbf{O_z}^{-1}} \cdot Z = \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

$$\underbrace{\lambda \mathbf{H_z}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}_{\text{Rückprojektion}} - \underbrace{\mathbf{O_z}^{-1} \cdot Z}_{\text{skalierter Korrekturvektor}} = \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{3.11}$$

Gleichung 3.11 hat auf der linken Seite 4 Variablen: λ, x, y und Z, wobei (x, y) durch den gegebenen Bildpunkt bestimmt sind und auch Z durch Π' vorgegeben ist. So bleibt nur noch eine Unbekannte, die Skalierung λ. Diese kann direkt aus der homogenen Koordinate berechnet werden.

Sei $\mathbf{q}$ der nicht normierte homogene Vektor der Rückprojektion auf die Referenzebene:

$$\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \mathbf{H_z}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.12}$$

$$\lambda \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} - \mathbf{O_z}^{-1} \cdot Z = \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{3.13}$$

Aus der 3. Komponente der Vektorgleichung folgt:

$$\lambda = \frac{1 + \mathbf{O_z}_3^{-1} \cdot Z}{q_3} \tag{3.14}$$

Bei gegebener Höhe Z können somit die verbleibenden Weltkoordinaten (X, Y) aus den Bildkoordinaten (x, y) berechnet werden:

$$\mathbf{q} = \mathbf{H_z}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.15}$$

$$\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \lambda \mathbf{q} - \mathbf{O_z}^{-1} \cdot Z \tag{3.16}$$

Für die DSP-Implementierung ist es wichtig, dass die Koordinatentransformation einfach und direkt berechnet werden kann. Da die Homographiematrizen und der Korrekturvektor bei einer statischen Kamera nur von festen Kameraparametern abhängen, können $\mathbf{H_z}$, $\mathbf{H_z}^{-1}$ und $\mathbf{O_z}^{-1}$ a priori berechnet und als Konfigurationsparameter übergeben werden. Die eigentliche Transformationsberechnung beschränkt sich dann auf Skalaroperationen, eine Matrizenmultiplikation sowie Vektormultiplikationen und -additionen.

3.2 Das Fahrzeug Objektmodell

Fahrzeuge werden im Bildraum durch ihre Bounding-Box, Basispunkt und Farbhistogramm beschrieben. Daraus wird die Position und Größe eines 3D Quaders berechnet, der als Objektmodell im 3 dimensionalen Raum dient. Dabei ist die Initialgröße des Quaders so gewählt, dass damit ein typischer PKW eingeschlossen wird.

Abbildung 3.3 zeigt einen Frame eines Videos, in dem der 3D Quader (cyan) als Objektmodell sowie die getrackte Bounding-Box (grün) eingezeichnet sind. Weiters sind die aktuellen, sowie die letzten 9 Basispunkte jedes Fahrzeuges eingezeichnet.

Folgende Eigenschaften bzw. Parameter werden für jedes Fahrzeug ausgewertet:

- *Bounding-Box*
  Die Größe und Position der Bounding-Box wird vom Trackingmodul für jedes Objekt ermittelt.

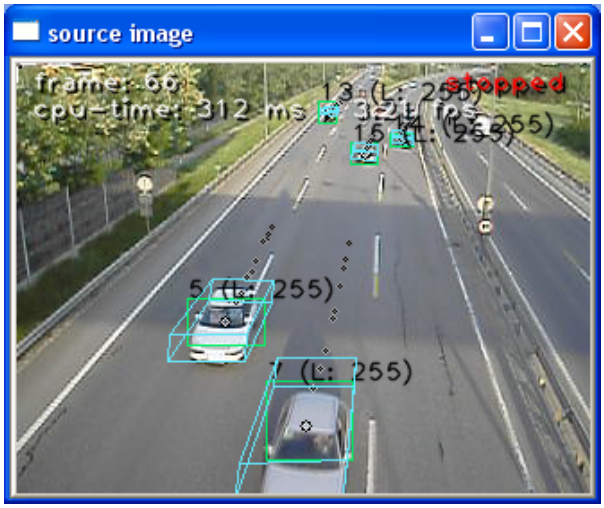


Abbildung 3.3: Bild einer Verkehrskamera mit den eingezeichneten Objektmodellen der getrackten Fahrzeuge

- *Basispunkt*
  Unter der Annahme, dass das Bild eines PKWs viele Symmetrien aufweist, liegt der Kantenschwerpunkt in der Mitte des Fahrzeuges und wird als Basispunkt verwendet. Aus dem Kantenbild des getrackten Objektes (Kantenbild des aktuellen Frames mit der Objektmaske gefiltert) wird der Basispunkt im Bildkoordinatensystem errechnet. Der Basispunkt ist gleich dem Schwerpunkt der Objektkanten. Um Fehler durch Bildrauschen zu vermeiden, werden nur Kantenpixel betrachtet, die über einem Schwellwert liegen. Der Bild-Basispunkt wird mit einer angenommenen Höhe in das Weltkoordinatensystem übertragen. Da das Kantenbild anstelle des Originalbildes verwendet wird, können Fehler durch Schatten bzw. Lichtkegel minimiert werden, da diese weniger ausgeprägte Kanten haben.

- *kompakter Körper*
  Ein Fahrzeug ist ein kompakter Körper, daher sollten auch in der Objektmaske eines Fahrzeuges keine Löcher zu sehen sein. Diese Kenngröße gibt an, wie viel Prozent der Fläche der 2D Bounding-Box durch die Objektmaske überdeckt werden. Diese Größe wird in Kapitel 4.2.2 verwendet, um die Blob-Objekt-Zuordnung zu optimieren.

- *3D Objektmodel*
  So lange noch keine stabilen Objektdaten gemessen werden können, wird die Größe des 3D Objektmodells (Quader) gleich dem Initialmodell gesetzt, andernfalls wird sie als gewichtetes Mittel zwischen der zuletzt gespeicherten Objektgröße und dem Objektmodell, das aus dem aktuellen Frame ermittelt wird, berechnet. Dazu wird

die Größe eines maximalen 3D Quaders berechnet, der in die Objekt Bounding-Box eingeschrieben werden kann (siehe Kapitel 3.1.2).

- *Histogramm*
  Das Farbhistogramm bildet die Grundlage für den verwendeten Mean-Shift Tracker und wird im folgenden Abschnitt detailliert beschrieben.

- *Geschwindigkeit*
  Die Geschwindigkeit wird durch die Bewegung des Basispunktes im Vergleich zum letzten Frame errechnet. Dabei wird einerseits die Geschwindigkeit im Bild in Pixel pro Frame ermittelt, mit der ein einfaches Vorhersagemodell in Bildkoordinaten möglich ist, und andererseits die absolute Geschwindigkeit des Basispunktes in m/s.

- *Fahrbahn*
  Es wird nicht nur die Position und Geschwindigkeit des Fahrzeuges ermittelt, sondern auch die aktuelle Fahrbahn, oder allgemeiner, die aktuelle Region of Interest (ROI) aus einer Menge von vordefinierten ROIs. Diese Information kann in weiterer Folge zur Berechnung von Fahrbahnbelegungen und Spurwechseln verwendet werden.

- *Konfidenz*
  Zu jedem Objekt wird ein Konfidenzwert in Prozent gespeichert, der ein Maß für die Qualität der Objektwerte (insb. der Trackingergebnisse) ist. Unterschreitet die Konfidenz einen Schwellwert, so konnte das Objekt über mehrere Frames nicht mehr getracked werden und wird aus der Liste der im Bild befindlichen und getrackten Objekte entfernt.

- *Stabilität*
  Ein weiterer Qualitätsparameter gibt an, wie stabil der Tracker ist. Im Prototyp wird dies durch den Vergleich der Größe der geschätzten Bounding-Box BB_p mit der tatsächlich getrackten Bounding-Box BB_t ermittelt. Dazu wird die relative Differenz der Flächen der beiden Bounding-Boxen berechnet.

$$Stabilität = \left(1 - \frac{|BB_p - BB_t|}{BB_p}\right) \cdot 100 \tag{3.17}$$

  Dabei kann die Stabilität einen Wert zwischen 0 und 100 annehmen, wobei ein Wert von 100 bedeutet, dass die Größe der geschätzten Bounding-Box zu 100% gleich der getrackten Bounding-Box ist. Wenn die getrackte Bounding-Box doppelt so groß oder noch größer als die geschätzte Bounding-Box ist, wird der Stabilitätswert auf 0 gesetzt. Während der Objektinitialisierungsphase wird kein Stabilitätswert

berechnet.

Als globales und Initialobjektmodell wird im Prototyp der 3D Quader, bzw. die Bounding-Box der Projektion des Quaders in das Bild, verwendet.
Für bereits gefundene Objekte wird dieses Modell um das jeweilige Farbhistogramm erweitert.

3.2.1 Histogramme

Der Begriff Histogramm wird folgendermaßen definiert:

Definition 1 (Histogramm) *Ein Histogramm beschreibt die relative Häufigkeit, mit der Klassen (z.B. Farben bzw. Farbbereiche) in einer Gesamtmenge vertreten sind, durch die Intervalllänge der zugehörigen Klassen (siehe Gleichung 3.18) [24](Seite 44–45). Die Intervalllänge gibt dabei die Gewichtung der Klassen vor. Wird jedem Intervall genau eine Klasse zugeordnet, und sollen alle Klassen gleich gewichtet werden, so wird eine konstante Intervalllänge verwendet.*

$$Histogrammbin_j = \frac{relative\ Häufigkeit\ der\ zum\ Intervall_j\ gehörigen\ Klassen}{Intervalllänge_j} \tag{3.18}$$

In der Bildverarbeitung werden häufig die Grauwerte oder Grauwertbereiche als Klasseneigenschaft für Helligkeitshistogramme benutzt. Bei Farbbildern kann ein Histogramm über alle möglichen Farben (im 3D Farbraum) erstellt werden, oder je ein Histogramm pro Farbkanal bzw. Farbkoordinate der unterschiedlichen Farbmodelle.

Histogramme über einzelne Farbkanäle / projizierte Histogramme

Im ersten Ansatz wurden die Histogramme nicht im 3D-Farbraum (RGB) erstellt, sondern es wurde für jede Farbe ein eigenes Histogramm berechnet. Dies entspricht einer Projektion eines vollständigen 3D-Histogramms auf die jeweiligen Farbachsen, also drei 1D-Histogramme. Dadurch ergibt sich eine wesentliche Reduktion der Histogrammbins (von m^3 auf $3m$). Somit können die Histogramme als eindimensionale Arrays gespeichert werden, was einen einfachen, direkten Zugriff möglich macht. Durch die Projektion sind die meisten Histogramme nicht mehr dünn besetzt, wie es bei den vollständigen 3D-Histogrammen der Fall ist. Selbst im Fall, dass einige Bins keine Einträge haben, ist

der geringfügig höhere Speicherbedarf eines einfachen Arrays im Vergleich zu einer Liste oder ähnlichen Datenstrukturen durch den direkten Zugriff durchaus gerechtfertigt.

Dünn besetzte 3D-Histogramme

Experimente zeigten allerdings, dass projizierte Histogramme in einigen Fällen zu unspezifisch sind, um ein gutes Modell des Objektbildes zu liefern.

Dies liegt einerseits an der Reduktion der Bins ($3m$ vs. m^3), andererseits an dem Verlust der Korrelation der Farbenkanäle. Die Reduktion der Bins könnte durch eine Erhöhung der Binzahl m abgefedert werden. Der gravierendste Nachteil ist allerdings der Verlust der Korrelationsinformation der Farbkanäle.

Somit musste der Ansatz mit den projizierten Histogrammen wieder verworfen werden. Stattdessen werden die vollständigen 3D-Histogramme verwendet, die allerdings aus Speicherplatzeffizienz nicht mehr als einfache Arrays implementierbar sind. In der überwiegenden Zahl der Fälle handelt es sich aber um dünn besetzte Histogramme, was bedeutet, dass die 3D-Histogramme ähnlich wie Matrizen in einer *sparse* Datenstruktur gespeichert werden können. Dabei werden nur die von Null verschiedenen Werte zusammen mit den jeweiligen Farbindizes in einer Liste gespeichert (siehe Quellcode 3.1). Um den Zugriff dennoch effizient gestalten zu können, wurden die sparse Histogramme als sortierte Liste implementiert, wobei die erste Farbkomponente das primäre Sortierkriterium darstellt, die zweite Komponente das sekundäre, etc.

Als Kompromiss zwischen projizierten 1D- und 3D-Histogrammen sind in anderen Farbräumen auch verschiedene 2D-Histogramme denkbar, bei denen eine Farbkomponente nicht berücksichtigt wird, oder bei denen zwei Farbkomponenten in eine Histogrammkomponente projiziert werden.
So wurden, neben den Experimenten mit dem RGB Farbraum, zusätzliche Experimente mit verschiedenen Kombinationen der Komponenten der HSV bzw. YCrCb Farbmodelle durchgeführt. Anstelle des vollen 3D Farbhistogramms werden dabei auch 2D und 1D Histogramme verwendet. Durch die Reduktion des Histogrammraums wird sowohl der benötigte Speicherplatz als auch die Laufzeit verringert.
Welchen Einfluss die Verwendung von verschiedene Farbräume bzw. Teilfarbräume hat, wird in Kapitel 6 untersucht.

```
typedef struct Histogram_Node
{
    UInt32 bin[4];  ///< color-bin indizes
    UInt32 count;   ///< count for this bin
    struct Histogram_Node *next;
} Histogram_Node;
```

Quellcode 3.1: sparse Histogramm Struktur

3.3 Fahrzeugdetektion / Initialisierung des Objektmodells

Aus der Annahme, dass relevante Objekte (insbesondere Fahrzeuge) in Bewegung sind, folgt, dass diese Objekte nicht in ein Hintergrundmodell integriert werden. Eine Ausnahme stellt dabei ein Stau dar, für dessen Behandlung zusätzliche Methoden verwendet werden müssen. Ein Hintergrundalgorithmus erkennt Fahrzeuge also als Vordergrundbereich. Die Umkehrung, dass der Vordergrund ausschließlich aus relevanten Objekten besteht, gilt allerdings nicht allgemein – obwohl dies die Basis der Objekterkennung mittels Hintergrundmodell darstellt. Trotz dieser Tatsache kann diese vereinfachende Annahme unter bestimmten Voraussetzungen zielführend sein.

Der beobachtete Bereich (ROI) kann den gesuchten Objekten entsprechend eingeschränkt werden. Bei einem Fahrzeugtrackingsystem ist es sinnvoll, die ROI auf die Fahrbahn zu beschränken, da anzunehmen ist, dass sich Fahrzeuge nur auf der Straße bewegen.
Außerdem können Fahrzeuge nicht einfach im Bild auftauchen bzw. verschwinden (siehe Kapitel 1.3), sondern fahren an definierten Stellen in das Bild ein. Daher ist es ausreichend, die Initialisierung neuer Fahrzeuge auf einen kleinen Teil der allgemeinen ROI, nämlich den Einfahrtsbereich, zu beschränken.

Aus dem Hintergrundmodell wird eine Grauwertmaske erstellt, die mittels der globalen Standardabweichung normiert wird. Aus dieser Maske werden im Segmentierung & Labeling Schritt (siehe Kapitel 2.4) Vordergrundbereiche extrahiert und einzelnen Blobs zugeordnet. Um einzelne Segmente und den Hintergrund voneinander unterscheiden zu können wird der Vordergrundschwellwert bei der Segmentierung verwendet. Die Ermittlung des genauen Vordergrundschwellwertes (als Vielfaches der Standardabweichung) wird in Kapitel 6.3 beschrieben.

Aus der Menge der so erhaltenen Blobs werden bereits getrackte Objekte herausgefiltert. Ein weiterer Filter überprüft, ob die verbleibenden Blobs bestimmte Kriterien

(wie z.B. eine bestimmte Größe, ein vorgegebenes Seitenverhältnis der Bounding-Box, etc.) erfüllen. Ist dies der Fall, so dienen sie zur Initialisierung eines neu zu trackenden Objektes.

Wenn ein Fahrzeug in den Sichtbereich der Kamera fährt und das erste Mal im Bild auftaucht, muss ein neue Objektinstanz erzeugt werden. Für die weitere Verarbeitung ist es entscheidend, dass diese neue Instanz das Fahrzeug möglichst gut und vollständig beschreibt. Das bedeutet, dass das neue Objekt optimalerweise erst initialisiert wird (und somit getrackt werden kann) wenn es vollständig sichtbar ist. Um die Trackingqualität zu verbessern, kann die Objektinitialisierung über mehrere Frames erfolgen, nicht nur über ein Frame. Dadurch wird allerdings die Zeit zwischen dem ersten Erscheinen des Fahrzeuges im Bild und der abgeschlossenen Initialisierung der Objektinstanz weiter vergrößert. Da das Fahrzeug aber möglichst von Anfang an getrackt werden soll, muss hier ein Kompromiss zwischen Trackingverzögerung und Initialisierungsqualität eingegangen werden.

Wenn ein Vordergrundblob gefunden wird, der keinem bereits bekannten Fahrzeug zugeordnet werden kann, und dessen Schwerpunkt in einem definierten Einfahrtsbereich liegt, wird eine neue Objektinstanz erzeugt. Die Verwendung von Einfahrtsbereichen hat den Vorteil, dass Blobs, die außerhalb der Fahrbahn erscheinen (z.B. Reflexionen an Tunnelwänden), sowie Schlagschatten außerhalb des Einfahrtsbereiches nicht fälschlicherweise als Fahrzeuge getrackt werden. Allerdings hat diese Methode auch den Nachteil, dass Fahrzeuge, die der Tracker „verloren“ hat, nicht mehr neu initialisiert und weiter getrackt werden können. Der Prototyp liest die Definition des Einfahrtsbereiches beim Start für jede Kamera ein. Dadurch kann der Einfahrtsbereich flexibel an die Szene und Aufgabenstellung angepasst werden.

Der genaue Algorithmus zum Erzeugen neuer Objekten des Prototypen ist in Kapitel 4.2.3 beschrieben.

4 Tracking

Ziel des Trackings ist es, bereits bekannte Objekte im aktuellen Frame wieder zu finden und die neue Position im nächsten Bild zu ermitteln. Dazu werden charakteristische Merkmale des Objektes (*features*) im aktuellen Bild bzw. einem Ausschnitt des aktuellen Bildes gesucht. Aus der Geschichte des Objektes, allgemeinen Informationen (z.B. Kameraparametern und Szenengeometrie) und der Position im aktuellen Frame können weitere Daten wie Geschwindigkeit, Beschleunigung (bzw. Abbremsen), Position in Weltkoordinaten, etc. berechnet werden.

In diesem Kapitel wird zuerst die allgemeine Funktionsweise von Trackingalgorithmen erläutert, um anschließend einen vom Hintergrundmodell unabhängigen Tracker, den Mean-Shift Algorithmus nach Cheng, näher zu beschreiben sowie die durchgeführten Anpassungen zu erläutern.

Der zweite Teil dieses Kapitels beschreibt die Korrelation der Ergebnisse des Mean-Shift Trackers mit den Blobs aus der Vordergrundmaske und dem Objektmodell.

4.1 Tracking Algorithmus

Grundsätzlich kann man alle Trackingalgorithmen in zwei Gruppen einteilen:

- Algorithmen, die aktuelle Blobs als Eingabe benötigen
  (Das heißt, es werden geeignete Vorverarbeitungsschritte benötigt, um passende Blobs zu generieren. Diese Schritte beinhalten unter anderem Hintergrundmodell, Vordergrundmaske und Segmentierung.)

- Algorithmen, die ohne Blob-Informationen (insbesondere ohne Hintergrundmodell) auskommen

Die erste Gruppe hat den Vorteil, dass das Tackingmodul selbst relativ einfach ausfallen kann. Allerdings müssen dafür schon gute Blobs vorhanden sein. Außerdem müssen

Objekte, die in mehrere Blobs zerfallen sind, explizit zusammengefasst werden (siehe Abbildung 1.2). Das heißt, es muss schon vor dem Tracking sichergestellt sein, dass qualitativ gute Blobs vorhanden sind.

Die Alternative dazu stellen Algorithmen dar, die keine Blobinformation benötigen. Diese Algorithmen suchen das Objekt im aktuellen Bild anhand von positionsinvarianten Objektmerkmalen. Um die benötigten Ressourcen (Speicherplatz und/oder Rechenzeit) in Grenzen zu halten, wird eine geeignete Startposition für die Objektsuche im aktuellen Frame verwendet. Im einfachsten Fall kann dies die Annahme sein, dass sich das Objekt noch an derselben Position wie im vorangegangenen Frame befindet. Sind weitere Informationen wie Objektgeschwindigkeit, Szenengeometrie, etc. vorhanden, so können diese verwendet werden, um die Schätzung der Initialposition zu verbessern. Ein entscheidender Vorteil dieser Algorithmen ist, dass kein Hintergrundmodell benötigt wird. Daher eignen sie sich besonders für Anwendungen, für die kein geeignetes Hintergrundmodell erstellt werden kann (z.B. bewegliche Kameras).

Obwohl der Prototyp ein Hintergrundmodell (zur Erkennung neuer Objekte) enthält, wurde für das Trackingmodul ein Algorithmus gewählt, der, außer zur Initialisierung, keine Blobinformationen benötigt: der Mean-Shift Algorithmus.
Die Motivation dabei ist, den Mean-Shift Algorithmus anhand von Verkehrsvideos zu testen. Weiters bringt diese Vorgehensweise für das Gesamtsystem den Vorteil, dass die Ergebnisse des Mean-Shift Trackers mit denen eines einfachen Blob Trackers kombiniert werden können und so die Gesamtgüte der Objektposition erhöhen.

Aus Kapitel 1.3 (Allgemeine Annahmen) folgt, dass das Abbild der Fahrzeuge, bis auf perspektivische Größenänderungen, annähernd gleich bleibt.

Ein einfacher Trackingansatz wäre daher, direkt das Bild des Fahrzeuges in unterschiedlichen Auflösungen im nächsten Frame zu suchen. Um robuster gegenüber Verzerrungen und Größenänderungen zu sein, wird nicht direkt auf den Bilddaten, sondern im Histogrammraum gearbeitet. Als Optimierungsverfahren wird der im Folgenden beschriebene Mean-Shift Algorithmus verwendet.

4.1.1 Der Mean-Shift Algorithmus

Der Mean-Shift Algorithmus wurde von Fukunaga und Hostetler entwickelt [25]. Dabei wurde er ursprünglich zur Cluster Analyse eingesetzt. Cheng hat eine Erweiterung des Mean-Shifts definiert, die einen flexibleren Kernel und Gewichte zulässt [26]. Im Folgenden wird unter Mean-Shift immer die Cheng Definition von Mean-Shift verstanden.

Definition 2 (Mean-Shift) *Sei $X = \mathbb{R}^n$ die Domäne, $S \subset X$ eine endliche Menge (die* Datenmenge *oder* Samples*), $K : X \to \mathbb{R}^n$ eine Kernelabbildung (kurz Kernel) und $w : S \to (0, \infty)$ eine Gewichtungsfunktion. [26]*

Der Datenmittelwert *mit dem Kernel K an $x \in X$ ist definiert als*

$$m(x) = \frac{\sum_{s \in S} K(s - x)w(s)s}{\sum_{s \in S} K(s - x)w(s)} \tag{4.1}$$

Die Differenz $m(x) - x$ wird als Mean-Shift *bezeichnet.*

Sei $T \in X$ eine endliche Menge (die Menge der Cluster Centers*). Die wiederholte Anwendung der Iteration $T \leftarrow m(T)$ wird als* Mean-Shift Algorithmus *bezeichnet.*

Weiters hat Cheng gezeigt [26], wie der Mean-Shift Algorithmus zur Extremwertsuche verwendet werden kann, wobei die zu optimierende Funktion f in die Gewichte einfließt.

$$w(s) = \frac{f(s)}{\sum_{t \in S} K(t - s)} \tag{4.2}$$

Dadurch verschiebt sich die zugrundeliegende Datenmenge in jedem Iterationsschritt in Richtung der Extremwerte.

In der Bildverarbeitung wird ein Template des zu suchenden Objekts verwendet. Das Template kann ein einfacher Ausschnitt des Luminanzbildes sein oder aus beliebigen anderen Daten bestehen, die sich aus dem Bild des Objektes berechnen und dieses genügend genau beschreiben. Ein anderer Datensatz kann mit Hilfe einer Vergleichsfunktion mit dem Template verglichen werden. Die Vergleichsfunktion liefert dabei ein quantitatives Maß für die Ähnlichkeit eines Templates mit dem zweiten Datensatz. Das Ziel ist, einen Datensatz zu finden, der maximale Ähnlichkeit mit dem Template aufweist.

Bei dem histogrammbasierten Mean-Shift Algorithmus besteht das Template aus dem Histogramm des zu suchenden Objekts. Dabei soll die Ähnlichkeit des Template Histogramms und des Histogramms des aktuellen Bildausschnitts optimiert werden. Analog zu der von Cheng vorgestellten Extremwertsuche fließen die Verhältnisse der Histogrammbins (Template / aktueller Bildausschnitt) als Gewichte in den Mean-Shift Algorithmus ein [26]. Die Qualitätsverbesserung eines jeden Iterationsschrittes wird mittels Bhattacharyya Distanz überprüft, wobei, an Stelle der Bhattacharyya Distanz selbst, der Bhattacharyya Koeffizient berechnet wird [27].

4.1.2 Mean-Shift Implementierung

Die Implementierung orientiert sich an der von Comaniciu, Ramesh und Meer in [27] vorgestellten Methode. Dieser Algorithmus wird hier zusammengefasst:

Mean-Shift Algorithmus nach Comaniciu et al.

Algorithmus 1 (Kernel Based Mean-Shift)

- h: *Skalierungsfaktor für die Distanz*
- l_h: *Anzahl der normalisierten Pixelpositionen des betrachteten Bildausschnittes, bei Skalierung h*
- m: *Anzahl der Histogrammbins*
- y_0: *Initialposition*
- x_i: *Pixel im betrachteten Bildausschnitt* $i \in \{1, \dots l_h\}$
- q: *normalisiertes Histogrammtemplate* $q[i]$*: i-ter Bin des Histogramms*
- $p(y)$: *normalisiertes Histogramms des Bildausschnittes an der Stelle y* $p[i](y)$*: i-ter Bin des Histogramms*
- $b(x)$: $\mathbb{R}^2 \rightarrow \{1, \dots m\}$ *Abbildung einer Bildposition in die entsprechende Histogrammbin Nummer*
- $g()$: $[0, \infty) \rightarrow \mathbb{R}$ *Kernelfunktion*

1. *Initialisierung und Berechnung der Bhattacharyya Koeffizienten für die Initialposition*

$$\rho(y_0) = \sum_{u=1}^{m} \sqrt{p[u](y_0)q[u]}$$

2. *Berechnung der Gewichte über alle Pixel x_i des Bildausschnittes*

$$u = b(x_i) \dots \textit{Nummer des entsprechenden Histogrammbins}$$

$$w_i = \sqrt{\frac{q[u]}{p[u](y_0)}}$$

3. *Mean-Shift um neue Position y_1 zu berechnen*

$$y_1 = \frac{\sum_{i=1}^{l_h} x_i w_i g(\left\| \frac{y_0 - x_i}{h} \right\|^2)}{\sum_{i=1}^{l_h} w_i g\left(\left\| \frac{y_0 - x_i}{h} \right\|^2\right)}$$

4. *Bhattacharyya Koeffizienten für y_1*

$$\rho(y_1) = \sum_{u=1}^{m} \sqrt{p[u](y_1)q[u]}$$

5. *innere Schleife, optimiere auf Mean-Shift Vektor*
   if $\rho(y_1) < \rho(y_0)$
   $y_1 = (y_0 + y_1)/2$; **goto** *4*

6. *Maximum erreicht?*
   if *(* $\|y_1 - y_0\| < \varepsilon$ *)*
   exit
   else
   $y_0 = y_1$; **goto** *2*

Vereinfachung

Um den Algorithmus zu beschleunigen und die Prototypimplementierung zu vereinfachen wurden folgende Modifikationen bzw. Anpassungen vorgenommen:

Fixpunktarithmetik Da die endgültige Zielplattform Gleitkommaarithmetik nicht direkt unterstützt, wird großteils mit Integer- bzw. Fixpunktarithmetik gerechnet. Gleitkommaarithmetik kann auf der Zielplattform nur mittels entsprechender Bibliotheken durchgeführt werden. Es gibt nur wenige Funktionen, die keine Fixkommazahlen, sondern Gleitkommazahlen verwenden, da für die entsprechenden Werte keine einfache a priori Abschätzung des Wertebereich bzw. der benötigten Genauigkeit möglich ist. Vor der endgültigen DSP Implementierung sind diese Parameter zu untersuchen und alle Berechnungen als Fixpunktarithmetik zu implementieren.

Gewichtsfunktion Außerdem wird als Gewichtsfunktion nur eine einfache Division und keine Quadratwurzel der Division verwendet (vgl. Algorithmus 1 Schritt 2). Dadurch werden die Gewichte zwar verzerrt, die wesentliche Charakteristik bleibt aber erhalten, da beide Funktionen streng monoton wachsend sind.

Flacher Rechteckiger Kernel Als Kernelfunktion wird ein flacher, rechteckiger Kernel verwendet. Dieser hat den Vorteil, dass er sehr leicht zu implementieren ist. Die Größe des Kernels wird der jeweiligen Objektgröße angepasst, das heißt, es wird nur auf einem kleinen Teil des Bildes gearbeitet, der genau so groß ist wie das gesuchte Objekt.

Durch den flachen Kernel kommt es zu keiner weiteren Modifikation der Gewichte. Dadurch fließen die Objekt-Randbereiche auf gleiche Weise in die Berechnung mit ein wie das Objekt-Zentrum. Bei stark nicht-konvexen Objekten (wie z.B. Personen) die mit einem rechteckigen oder ellipsoiden Kernel modelliert werden, ist die genaue Position der Arme und Beine für den Mean-Shift Tracker nicht bekannt. Somit werden, gerade am Rand, viele Hintergrundbereiche vom Kernel überdeckt. In solchen Fälle ist es sinnvoll, eine nicht rechteckige Kernelfunktion zu verwenden.
Dies ist bei Fahrzeugen aber nicht der Fall, somit würde ein komplexer Kernel nur wenig Qualitätsverbesserung bringen, aber deutlich mehr Rechenzeit kosten und eine komplexere Implementierung nach sich ziehen, was gerade bei embedded Systems zu vermeiden ist.

Objektvorgeschichte

Das Objekt-Modell (das heißt das Histogrammtemplate) ist direkt verantwortlich für die Qualität der Tracking-Ergebnisse. Allerdings ist die Wahl eines geeigneten Objekt-Modells nicht trivial, da sich das Abbild eines Objekts von Bild zu Bild verändert (z.B. durch Bewegung des Objekts, Beleuchtungsänderung, etc.). Das bedeutet, dass das Histogrammtemplate nicht konstant auf jenes Histogramm gesetzt werden kann, das bei der ersten Detektion des Objektes erstellt wurde. Selbst wenn sich das Abbild des Objektes innerhalb einzelner Frames nur leicht ändert (z.B. durch unterschiedleiche Beleuchtung), so kann die kumulierte Änderung nach einiger Zeit so groß sein, dass das Objekt nicht mehr richtig wiedergefunden und somit auch nicht mehr verfolgt werden kann. Andererseits ist es auch nicht sinnvoll, jeweils das zuletzt erzeugte Histogramm des Objektes als Template zu verwenden. Bei diesem Ansatz werden kleine Fehler aufsummiert, bis das Objekt-Modell nicht mehr dem zu trackenden Objekt entspricht.

Eine relativ einfache und wirkungsvolle Lösung für das Problem ist die Verwendung von mehreren Histogrammtemplates, statt nur einem einzigen. Dabei werden die letzten n

Histogramme q_n eines jeden Objektes gespeichert, um Änderungen des Objektbildes in das Template zu integrieren, ohne dass Einzelfehler signifikanten Einfluss auf das gesamte Template haben.

Dabei werden die Gewichte zur Berechnung des Mean-Shift Vektors für die einzelnen Histogramme berechnet und aufsummiert. Zur Berechnung der Bhattacharyya Koeffizienten wird der Mittelwert der einzelnen Bhattacharyya Koeffizienten verwendet. (siehe Algorithmus 2 Schritt 2 und 5)

Minimaler Bhattacharyya Koeffizient

Das Tracking basiert auf einer Maximum Optimierung der Bhattacharyya Koeffizienten. Dies funktioniert sehr gut, wenn sich das gesuchte Objekt in der Nachbarschaft der Initialposition befindet. Wenn sich das Objekt nicht im Bild befindet, oder zu weit von der Initialposition entfernt ist, liefert der naive Mean-Shift Algorithmus eine falsche Position, da keine Überprüfung eines Mindestmaßes an Übereinstimmung (minimaler Bhattacharyya Koeffizient) existiert.

Dieser minimale Bhattacharyya Koeffizient kann auf einen festen Wert gesetzt werden. So kann z.B. eine Übereinstimmung von mind. 50% gefordert werden. Allerdings gibt es in den meisten Fällen keinen Schwellwert, der a priori festgesetzt und ausreichend begründet werden könnte. Ein willkürliches Setzen dieses Schwellwertes birgt Probleme. Ist der Schwellwert zu hoch angesetzt, kann es leicht vorkommen, dass ein Objekt nicht mehr korrekt verfolgt wird, weil sich sein Bild durch Drehung bzw. unterschiedliche Perspektive verändert. Ist der Wert zu niedrig gesetzt, kann sich der Tracker im Hintergrund „verfangen“.

Um diese Probleme zu vermeiden, wurde ein dynamischer Schwellwert implementiert. Die Grundannahme dabei ist, dass die Übereinstimmung eines Kandidaten mit dem Objekt besser sein muss, als die Übereinstimmung des Objekts mit dem Hintergrund. Dazu wird der Mittelwert der Bhattacharyya Koeffizienten zwischen den Objekt-Modell-Histogrammen und dem Hintergund an der aktuellen Position als Schwellwert verwendet (siehe Objektvorgeschichte).

Berücksichtigung des Hintergrundhistogramms

Der von Comaniciu et al. [27] ursprünglich vorgestellte Mean-Shift Algorithmus berücksichtigt keine Daten eines Hintergrundmodells. Dadurch kann dieser Algorithmus sehr

flexibel und in vielen Anwendungsbereichen eingesetzt werden. Allergings gehen damit viele Informationen verloren, die zur Verbesserung der Trackingqualität genutzt werden können. Dies kann einerseits die automatische Generierung von Schwellwerten sein, aber auch die Berücksichtigung des Hintergrundes bei den Mean-Shift Gewichten. In [28] präsentierten Comaniciu et al. eine entsprechende Erweiterung. So werden Farben, die im Objekt öfter vorkommen als im Hintergrund, stärker gewichtet, während Farben, die im Hintergrund dominanter sind, entsprechend weniger gewichtet werden.

Beide Erweiterungen werden in der konkreten Implementierung verwendet. In Algorithmus 2 findet die Berechnung des adaptiven Schwellwertes in Schritt 1, und die Anpassung der Gewichte durch das Hintergrundhistogramm in Schritt 2 statt.

Subpixel-Skalierung

Wenn sich die berechnete Position eines Objekts der wahren Zielposition annähert, wird der Betrag des Mean-Shift Vektors immer kleiner, bis er kleiner als der Pixelabstand ist. Außerdem korreliert der Betrag des Mean-Shift Vektors, der mit den vereinfachten Gewichten berechnet wurde, nicht immer mit der wahren Entfernung zur Zielposition. So kann es zu Subpixelbeträgen kommen, obwohl sich die derzeitige Position noch deutlich von der Zielposition unterscheidet. Da die Position in Pixel, also ganzzahlig, verarbeitet wird, würde es hier zu einem Stillstand, und in direkter Folge, zu einem Abbruch der Iteration kommen. Um diesen frühzeitigen Abbruch zu vermeiden, wird der Mean-Shift Vektor s bei Subpixel-Beträgen so skaliert, dass der Betrag der längsten Komponente gleich 1 wird.

wenn $\max(|(|\, s)) < 0$ und $\max(s) \neq 0$:

$$s = \begin{pmatrix} s_x \\ s_y \end{pmatrix} / \max(s) \tag{4.3}$$

Implementierter Algorithmus

Algorithmus 2 (Mean-Shift Tracker)

n: *Anzahl der Histogramme im Objekt Modell*
m: *Anzahl der Histogrammbins*

l: *Anzahl der normalisierten Pixelpositionen des betrachteten Bildausschnittes*

y_0: *Initialposition bzw. letzter Iterationsschritt*

Δx_i: *Pixel im betrachteten Bildausschnitt relativ zum Mittelpunkt des Bildausschnittes* $i \in \{1, \dots l\}$

q: *normalisiertes Histogramm des Templates*
$q[k]$: *k-ter Bin des Histogramms*

$p(y)$: *normalisiertes Histogramm des Bildausschnittes an der Stelle y*
$p[k](y)$: *k-ter Bin des Histogramms*

$bg(y)$: *normalisiertes Histogramm des Hintergrundbildausschnittes an der Stelle y*
$bg[k](y)$: *k-ter Bin des Histogramms*

$b(x)$: $\mathbb{R}^2 \rightarrow \{1, \dots m\}$ *Abbildung einer Bildposition in die entsprechende Histogramm-Bin-Nummer*

Γ: *bester gefundener Bhattacharyya Koeffizient (initialisiert mit:* $\Gamma = 0$*)*

$\Gamma_{min}(y_0)$: *minimaler Bhattacharyya Koeffizient*

s: *Mean-Shift Vektor*

1. *minimaler Bhattacharyya Koeffizient*
   berechne Bhattacharyya Hintergrund Koeffizient Γ_{min} *an Position* y_0

$$\Gamma_{min}(y_0) = \frac{\sum_{k=1}^{n} \sum_{u=1}^{m} \sqrt{bg[u](y_0) q_k[u]}}{n}$$

2. *Berechnung der Gewichte*
   (siehe Vereinfachung, Objekt Vorgeschichte und Berücksichtigung des Hintergrundhistogramms)

$$u = b(\Delta x_i + y_0) \dots \textit{Nummer des entsprechenden Histogrammbins}$$

$$w_i = \sum_{k=1}^{n} \underbrace{\frac{q_k[u]}{p[u](y_0)}}_{\textit{einfache Gewichte}} \underbrace{\left(\frac{q_k[u] - bg[u](y_0)}{q_k[u] + bg[u](y_0)} + 1 \right)}_{\textit{Modifikation durch Hintergrund}}$$

3. *Berechnung des Mean-Shift Vektors*

$$s = \frac{\sum_{i=1}^{l} \Delta x_i w_i}{\sum_{i=1}^{l} w_i}$$

4. *Sub Pixel Scale*

   if $|s_x| < 1$ **and** $|s_y| < 1$ **and** $\max(s) \neq 0$
   $s = \frac{s}{\max(|s_x|,|s_y|)}$

5. *Mean-Shift anwenden*

   for $\lambda = (0, \ldots 3)$
   $y = y_0 + \lambda \cdot s$
   Bhattacharyya Koeffizient an Position y:

$$\rho(y) = \frac{\sum_{k=1}^{n} \sum_{u=1}^{m} \sqrt{p[u](y) q_k[u]}}{n}$$

   if $\rho(y) > \Gamma$ **and** $\rho(y) > \Gamma_{min}(y_0)$
   $y_1 = y$
   $\Gamma = \rho(y)$

6. **if** *neue Position gefunden*
   $y_0 = y_1$
   goto *1*
   else
   exit

Parameter des implementierten Mean-Shift Algorithmus

Die Anzahl der Histogramme im Objekt Modell bestimmt, wie schnell das Modell auf Änderungen im Histogramm des Objektes reagiert. Bei einer geringe Anzahl von Histogrammtemplates passt sich das Objekt Modell des Tracking Algorithmus schneller an. Gleichzeitig steigt die Wahrscheinlichkeit, dass Fehler in das Modell integriert werden und das Objekt nicht mehr richtig erkannt wird (siehe Absatz: 4.1.2 Objektvorgeschichte oben). Im Prototyp hat sich gezeigt, dass das Objekt Modell mit 5 Histogrammtemplates ausreichend stabil ist.

Der Prototyp verwendet 3D-Farbhistogramme, wobei jeder Farbkanal in 8 Bereiche eingeteilt wird. Dadurch ergeben sich 512 verschiedene Histogrammbins, wobei im Durchschnitt nur ca. 40 Bins besetzt sind und in einer sparse Datenstruktur gespeichert werden. (siehe Kapitel 3.2.1)

An der Initialposition beginnt der Mean-Shift Algorithmus mit der Maximumssuche. Je besser diese Position gewählt wurde, umso schneller (mit weniger Iterationsschritten) bzw. umso genauer (bei fester Maximalanzahl der Iterationsschritte) kann die beste Übereinstimmung mit dem Objektmodell gefunden werden. Der Prototyp verwendet, falls bekannt, die Objektgeschwindigkeit und die Zeitdifferenz zwischen der letzten bekannten Position und dem Zeitstempel des aktuellen Frames, um die neue Position des Objektes zu schätzen. Diese Prognose dient als Initialposition für den Tracking Algorithmus.

4.2 Abgleich mit Objektmodell

Der Tracking Algorithmus ist ein Teil eines Gesamtsystems und muss mittels passender Schnittstellen integriert werden. Dabei benötigt der Tracker Informationen über das zu suchende Objekt.
Beim verwendeten Mean-Shift Tracker heißt dies, dass dem Algorithmus das aktuelle Bild, eine Initialposition und die Größe des Suchfensters (siehe Kapitel 4.1.2) übergeben werden. Das Suchfensters wird im Prototyp als Bounding-Box repräsentiert. Als initiale Bounding-Box kann die Bounding-Box des Objektes im vorhergehenden Frame, oder, wie im Prototyp, ein Schätzwert für die neue Position der Bounding-Box im aktuellen Frame verwendet werden. Diese geschätzte Bounding-Box wird aufgrund der letzten bekannten Position, der Geschwindigkeit und der Zeitdifferenz ermittelt. Außer der Bounding-Box werden auch noch die neuen Werte für den Basispunkt und das 3D Modell für das aktuelle Frame prognostiziert.

Nachdem die neue Position eines Objektes durch den Tracker ermittelt wurde, müssen eine Korrelation der durch den Tracker gefundenen Bounding-Box und der im Segmentierungsschritt gefundenen Blobs hergestellt werden. Blobs, die keinem bereits getracktem Objekt zugeordnet werden können, initialisieren ein neues Objekt.

Dieser Abgleich zwischen Trackingergebnissen und Blobs erfolgt im Prototyp in drei Schritten:

1. Eine Liste von Objekt-Blob-Korrelationen wird erstellt, und für jedes Objekt wird der Kernblob ermittelt.

2. Ausgehend von den Objekt-Kernblobs werden weitere Blobs dem Objekt zugeordnet, wenn sich die Übereinstimmung mit dem Objektmodell dadurch verbessert.

3. Verbleibende Blobs erzeugen neue Objekte, wenn sie dem Objektmodell und den

Rahmenbedingungen entsprechen.

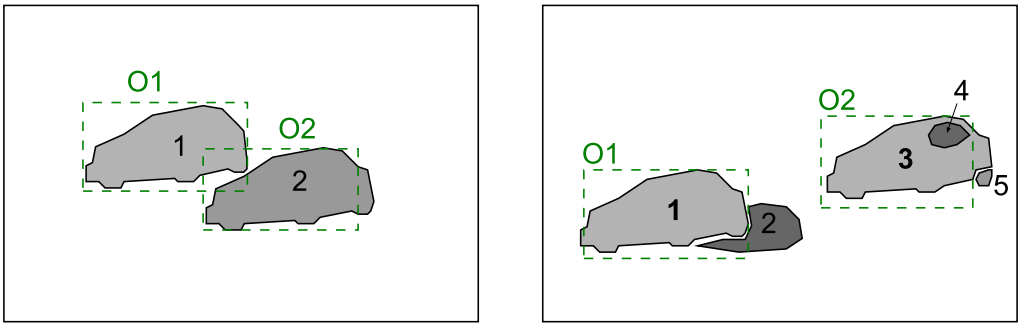
Nachdem die Ergebnisse der Prognose der Objekteigenschaften, des Trackingalgorithmus und der Vordergrundmaskensegementierung miteinander korreliert wurden, werden sie zusammengefasst und die Objekteigenschaften (siehe Kapitel 3.2) entsprechend aktualisiert.

4.2.1 Objekt-Blob-Korrelation

Als Kriterium für die Objekt-Blob-Korrelation wird die prozentuelle Überdeckung verwendet. Damit wird sowohl die örtliche Korrelation berücksichtigt, wie dies bei der Distanz der Schwerpunkte bzw. Bounding-Box Mittelpunkte der Fall wäre, als auch das Größenverhältnis der beiden Bounding-Boxen. Ein Blob wird jenem Objekt zugeordnet, in dessen Tracker Bounding-Box die größte Fläche der Blobmaske liegt.

$$blob.related\_object = \underset{\forall obj \in objects}{\operatorname{argmax}} \left(\text{Area}(blob \bigcap \text{BoundingBox}(obj)) \right) \tag{4.4}$$

Abbildung 4.1(a) illustriert diese Zuordnung. Der Blob 1 wird dem Objekt O1 zugeordnet, da er zu 100% in der Bounding-Box von O1 liegt, während nur ca. 10% der Blobfläche in der Bounding-Box von O2 liegen. Blob 2 wird entsprechend O2 zugewiesen.



(a) überlappende Bounding-Boxen (b) in mehrere Blobs zerfallene Objekte

Abbildung 4.1: Objekt-Blob-Korrelation mit eingezeichneten Tracker Bounding-Boxen (grün strichliert)

In der Praxis zerfallen die Fahrzeuge oft in mehrere Blobs. Wie diese Blobs Objekten zugeordnet werden, ist in Abbildung 4.1(b) dargestellt. Blob 1 liegt fast vollständig in

der Bounding-Box von Objekt O1 und wird diesem zugewiesen. Auch Blob 2 wird dem Objekt O1 zugewiesen, da ein Teil der Blobfläche innerhalb der Bounding-Box von O1 liegt und keine größere Überdeckung mit einer anderen Objekt Bounding-Box existiert. Die Blobs 3 und 4 werden dem Objekt O2 zugeordnet. Blob 5 kann keinem bestehenden Objekt zugeordnet werden.

Von allen Blobs, die einem Objekt zugeordnet werden, wird der Blob mit der größten Überdeckung der Objekt Bounding-Box als *Kernblob* markiert. In diesem Beispiel ist Blob 1 der Kernblob für das Objekt O1 und Blob 3 der Kernblob für das Objekt O2.

4.2.2 Objektmasken Optimierung

Im zweiten Schritt wird für jedes Objekt aus den Blobs der Vordergrundmaske eine Objektmaske erstellt, die möglichst kompakt ist, das heißt, es wird versucht, die Kompaktheit (= Objektmaskenfläche/Bounding-Box Fläche) der Objektmaske zu maximieren.

Dazu wird die Objektmaske mit dem Objekt-Kernblob initialisiert. Andere Blobs werden genau dann zur Objektmaske hinzugefügt, wenn der Blobschwerpunkt innerhalb der Objektmasken Bounding-Box liegt und die entstehende Objektmaske kompakter wird.

Im vorherigen Schritt wurde Blob 2 der Abbildung 4.1(b) dem Objekt O1 zugeordnet, obwohl dieser Blob als Schatten nicht zum Fahrzeug gehört und die Trackingergebnisse verschlechtert, wenn er mitgetrackt wird. Andererseits konnte Blob 5 dem Objekt O2 nicht zugeordnet werden, da er nicht in der vom Tracking Algorithmus ermittelten Bounding-Box liegt, obwohl Blob 5 als Hinterrad offensichtlich Teil des Fahrzeuges ist. Diese fehlerhaften Korrelationen können durch das eben beschriebene Verfahren verbessert werden, wie es in Abbildung 4.2 auf der nächsten Seite illustriert ist. Diese Abbildung zeigt nochmals das selbe Beispiel aus Abbildung 4.1(b), diesmal ist allerdings nicht die Bounding-Box aus dem Tracker, sondern die Bounding-Box der Objektmaske eingezeichnet. Außerdem ist die Fläche von Blobs, die in die Objektmaske integriert wurden, grau gefärbt, während von Blobs, die in keiner Objektmaske vorkommen, nur der Umriss gezeichnet wurde. Die Position der Schwerpunkte der einzelnen Blobs ist durch ein entsprechendes Symbol eingezeichnet.

Es ist leicht zu sehen, dass Blob 2 diesmal nicht zur Objektmaske von O1 hinzugefügt wurde, da der Schwerpunkt von Blob 2 nicht innerhalb der Bounding-Box von O1 liegt. Andererseits kann durch das Hinzufügen von Blob 5 zur Objektmaske von O2 die Maskenfläche vergrößert werden, ohne dass sich die Bounding-Box ändert.

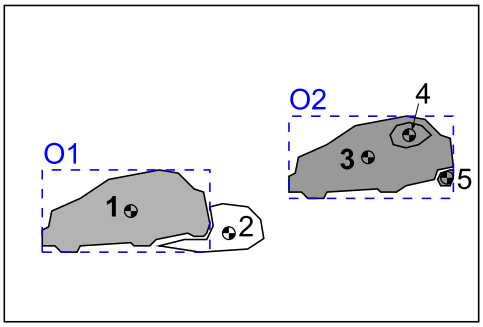


Abbildung 4.2: Bounding-Boxen der Objektmasken (blau strichliert) und dazugehörige Blobs (grau) sowie nicht integrierbare Blobs (weiß)

4.2.3 Initialisierung neuer Objekte

Blobs, die weder im ersten noch im zweiten Schritt einem Objekt zugeordnet werden konnten, erzeugen unter den folgenden Bedingungen ein neues Objekt.

1. Um Fehler durch Bildrauschen und für das System uninteressante Objekte zu vermeiden, muss die Blobfläche größer als ein Schwellwert sein.

2. Wenn ein Einfahrtsbereich definiert ist, muss der Schwerpunkt des Blobs im Einfahrtsbereich liegen, wobei der Einfahrtsbereich jener Teilbereich der ROI ist, in dem neue Fahrzeuge in den Sichtbereich der Kamera fahren. Durch die geeignete Wahl des Einfahrtsbereichs wird außerdem sichergestellt, dass keine Objekte aus nur halb sichtbaren Fahrzeuge erzeugt werden.

3. Die Anzahl der Kantendifferenzpixel muss größer als ein einstellbarer Schwellwert sein. Es wird angenommen, dass der Umriss von Fahrzeugen deutlich als Kanten im Kantendifferenzbild zu erkennen ist, wohingegen Schatten bzw. durch die Scheinwerfer beleuchtete Bereiche keine scharfen Kanten aufweisen.

Sind alle Bedingungen erfüllt, wird ein neues Objekt mit Standardwerten erstellt, der erzeugende Blob als Kernblob definiert und die Objektmaske damit initialisiert. Danach werden dieser, wie im vorigen Abschnitt beschrieben, weitere Blobs hinzugefügt, wenn sie das Objektmodell unterstützen und die Kompaktheit der Objektmaske verbessern.

Wurde ein neues Objekt erstellt, so befindet es sich zuerst in der Initialisierungsphase. Die Länge der Initialisierungsphase ist doppelt so groß gewählt, wie ein Fahrzeug mit der aktuellen Durchschnittsgeschwindigkeit benötigt, um vollständig im Bild sichtbar zu werden. Damit soll sichergestellt werden, dass das Histogrammtemplate, selbst bei

ungünstigen Bedingungen, aus Daten des vollständig sichtbaren Fahrzeuges besteht. Während dieser Zeit läuft der Mean-Shift Tracker, um Daten zu sammeln, das Objektmodell aufzubauen und zu stabilisieren (insb. Histogrammtemplate und Fahrzeuggröße), die Trackingergebnisse werden allerdings noch nicht weiter verarbeitet. Nach der Initialisierungsphase werden statische Größen wie die Fahrzeuggröße eingefroren, während das Histogramm laufend aktualisiert wird.

4.2.4 Objekt Nachverarbeitung

Nachdem die Blob-Objekt-Korrelation vollständig durchgeführt und für jedes Objekt eine Objektmaske erstellt wurde, hat jedes Objekt drei Bounding-Boxen aus unterschiedlichen Quellen, aus denen eine eindeutige Bounding-Box und Objektposition errechnet werden muss:

- Prognose der Objekteigenschaften
- Ergebnisse der Trackingalgorithmus
- Objektmaske der korrelierten Blobs

Der Konfidenzwert des Objektes wird als gewichteter Mittelwert aus den Konfidenzwerten des Trackingergebnisses und der Objektmaske berechnet. Das Gewicht ist abhängig vom aktuellen Zustand des Objektes. Während der Objektinitialisierungsphase wird ausschließlich der Konfidenzwert der Objektmaske berücksichtigt, ansonsten ist das Gewicht für die Objektmaske frei wählbar (das Gewicht für die Trackingergebnisse ist dann entsprechend 1 − Objektmaskengewicht).

Unterschreitet der aus Tracking und Objektmaske resultierende Konfidenzwert einen Schwellwert, wird für die weiteren Berechnungen der Schätzwert der Bounding-Box verwendet. Ansonsten wird die Objekt Bounding-Box als gewichteter Mittelwert wie folgt berechnet:

$$BB = \frac{BB_t \cdot \omega_t \kappa_t + BB_m \cdot \omega_m \kappa_m}{\omega_t \kappa_t + \omega_m \kappa_m} \tag{4.5}$$

mit:

BB_s: Objekt Bounding-Box als Vektor: $(x_{links}, y_{oben}, x_{rechts}, y_{unten})$
BB_t: Bounding-Box der Trackingergebnisse
BB_m: Bounding-Box der Objektmaske

ω_t: Gewichtungsfaktor der Trackingergebnisse $\omega_t \in [0-1]$
ω_m: Gewichtungsfaktor der Objektmaske $\omega_m = 1 - \omega_t$
κ_t: Konfidenzwert der Trackingergebnisse
κ_m: Konfidenzwert der Objektmaske

Anschließend wird der Basispunkt des Objektes berechnet. Der Basispunkt ist der Schwerpunkt der Kanten des Kantendifferenzbildes, das durch die Objektmaske gefiltert wird. Das Kantendifferenzbildes ist das absolute Differenzbild des aktuellen Kantenbildes und des Hintergrundkantenbildes, wobei als Kantenbild der maximale Absolutwert der x- bzw. y-Gradienten (die durch ein Sobelfilter [29](Seite 348–350) erstellt wurden) verwendet wird.

Dadurch wird der Basispunkt primär aufgrund der symmetrischen Fahrzeugkanten berechnet. Bereiche, die nur Helligkeitsänderungen, aber keine Strukturänderung aufweisen, werden nicht berücksichtigt und verfälschen die Position des Basispunktes somit nicht.

Mit Hilfe des Bild-Basispunktes wird der Basispunkt in Weltkoordinaten und die Geschwindigkeit (sowohl im Bildkoordinatensystem als auch im Weltkoordinatensystem) berechnet.

Die Stabilität des Trackers wird mit folgender Formel berechnet:

$$\text{Stabilität} = 1 - \frac{|w_o h_p - w_p h_p|}{w_p h_p} \tag{4.6}$$

mit:

w_o, h_o: Breite bzw. Höhe der ermittelten Objekt Bounding-Box
w_p, h_p: Breite bzw. Höhe der geschätzten Objekt Bounding-Box

Während der Initialisierungsphase wird aus der Objekt Bounding-Box und der Position des 3D-Basispunktes die Länge und Breite des Fahrzeuges geschätzt.

Abschließend werden alle Objekte, deren Konfidenzwert einen Minimalwert unterschreitet, aus der Liste der aktiven Objekte gestrichen und nicht mehr weiter getrackt.

4.3 Zusammenfassung

In diesem Kapitel wurde der vom Hintergrundmodell unabhängige Mean-Shift Tracking Algorithmus nach Cheng beschrieben [26]. Anschließend wurden Anpassungen wie Fixpunktarithmetik, Objekttemplate Vorgeschichte und Berücksichtigung des Hintergrunds für den allgemeinen Mean-Shift Algorithmus, so wie sie im Prototyp implementiert sind, erläutert.

Anschließend wurde der Abgleich des Objektmodells mit den Ergebnissen des Mean-Shift Trackers und den Blobs der Vordergrundsegmentierung beschrieben.
Dabei werden einerseits die Blobs aus der Vordergrundsegmentierung einer aus dem Tracker stammenden Bounding-Box zugeordnet, andererseits eine Objektmaske um einen Kernblob aufgebaut, so dass die Kompaktheitseigenschaft der Bounding-Box (siehe Kapitel 3.2) optimiert wird. Blobs, die keinem bereits bekannten Objekt zugeordnet werden können, werden zur Erstellung eines neuen Objektes verwendet.
Abschließend wird für jedes Objekt ein Basispunkt errechnet, so dass die Position des Objektes in das Weltkoordinatensystem übertragen werden kann.

5 System

In diesem Kapitel wird beschrieben, wie die in den vorigen Kapiteln vorgestellten Methoden zu einem Gesamtsystem integriert werden.

Die eigentliche Zielplattform ist ein Embedded System. Um die Entwicklung und den Test des Prototypen zu vereinfachen, kann das System mit Hilfe des DSP-via-PC Frameworks auch auf einem PC kompiliert werden. Abbildung 5.1 gibt einen Überblick über die verwendeten Module des Prototyps und die möglichen Plattformen. Die Bildverarbeitungsmodule Hintergrund, Tracking und Geometrie wurden schon in den früheren Kapiteln eingehend behandelt. Die weiteren Module Datenbank und Statistik sowie das DSP-via-PC Framework werden in diesem Kapitel kurz vorgestellt. Anschließend wird die Integration zu dem Gesamtsystem beschrieben, das im nächsten Kapitel evaluiert wird.

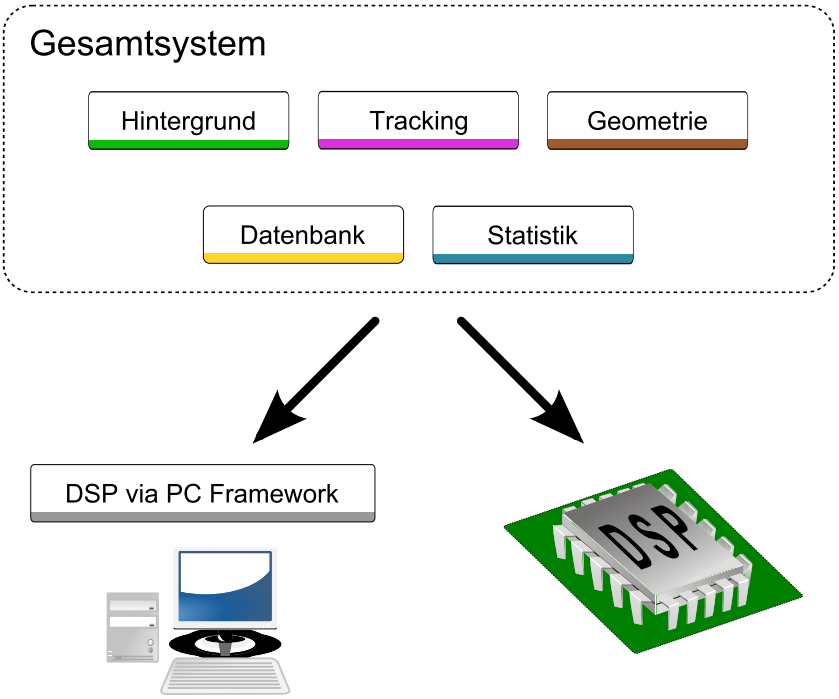


Abbildung 5.1: Gesamtsystem Übersicht

5.1 Hilfsmodule

Die Hilfsmodule erledigen keine Bildverarbeitungsaufgaben, sind allerdings für die Kommunikation zwischen den Bildverarbeitungsmodulen notwendig, außerdem übernehmen sie die Auswertung und Verwaltungsaufgeben.

5.1.1 Datenmanagementmodul

Das Datenbankmodul dient der Verwaltung von Blob- und Objektdaten und stellt ein wichtiges Bindeglied zwischen den einzelnen Modulen dar. Alle Daten eines Blobs werden in einer Struktur zusammengefasst. Diese Struktur kann dann von allen Modulen gelesen und aktualisiert werden. Nicht mehr benötigte Blobs werden gelöscht, nachdem das entsprechende Frame vollständig abgearbeitet wurde. Gleiches gilt sinngemäß für getrackte Objekte. Objekte werden aus der Datenbank entfernt, wenn sie nicht mehr getrackt werden können. Objektinstanzen werden vom Trackingmodul angelegt und vom Statistikmodul und weiteren Auswertungsfunktionen analysiert.

Das Datenbankmodul erzeugt für jeden Datentyp eine doppelt verkettete Liste, wobei die Listendaten für schnellen Zugriff im internen Speicher abgelegt werden. Die eigentlichen Nutzdaten werden im externen Speicher gespeichert.

5.1.2 Statistikmodul

Im Prototyp ist nur ein rudimentäres Statistikmodul implementiert, das die minimale, maximale und durchschnittliche Geschwindigkeit der aktuell getrackten Fahrzeuge ermittelt.

Über jedes Frame wird die minimale, maximale und gewichtete durchschnittliche Framegeschwindigkeit (Geschwindigkeiten für dieses spezielle Frame) errechnet. Die durchschnittliche Geschwindigkeit wird dabei mit dem Konfidenzwert (siehe Kapitel 3.2) der Fahrzeuge gewichtet. Diese Framegeschwindigkeiten werden dann über das eingestellte Zeitfenster gemittelt und ausgegeben.

5.2 DSP-via-PC Framework

Das DSP-via-PC Framework ermöglicht es, Programme, insbesondere XDAIS Algorithmen, die für einen Texas Instruments DSP entwickelt wurden, auf einem PC mit Microsoft Visual C++ zu kompilieren und auszuführen. Diese Quellcodekompatibilität ist natürlich auf die reine Softwarefunktionalität beschränkt. Die spezielle Hardware des jeweiligen Embedded Systems (ausgenommen des DSPs selbst) wird dabei nicht berücksichtigt.

Der erste Teil dieses Kapitels gibt einen Überblick über das Einsatzgebiet des DSP-via-PC Framework.
Anschließend werden der Funktionsumfang und die einzelnen Module vorgestellt.

5.2.1 Motivation und Zielsetzung

Bei der Entwicklung von Embedded Systems kommt es oft vor, dass Software und Hardware parallel entwickelt werden. Das heißt, während der Anfangsphase der Softwareentwicklung steht die Zielplattform noch nicht zur Verfügung. Doch selbst wenn die Hardware verfügbar ist, sind die Ein- und Ausgabemöglichkeiten von Embedded Systems stark eingeschränkt und auf die jeweilige Anwendung zugeschnitten. Aufgrund dieser Einschränkungen eignen sich Embedded Systems nur bedingt zur Algorithmusentwicklung und Prototypimplementierung. Es stehen zwar mächtige Werkzeuge (Integrated Development Environment (IDE), Debugger, Profiler, Simulator, etc.) für die DSP-Entwicklung zur Verfügung, diese haben aber ihr Hauptanwendungsgebiet in der Portierung von bekannten und getesteten Algorithmen.

Die Algorithmusentwicklung und Implementierung erfolgt in mehreren Entwicklungsstufen. Um eine Platzierung dieser Arbeit innerhalb dieses Systems vornehmen zu können, sei hier eine grobe Einteilung beschrieben. Diese Aufzählung erhebt keinesfalls Anspruch auf Vollständigkeit, sondern soll nur einen groben Überblick bieten.

1. Algorithmenentwicklung (z.B. in MatLab)

2. Prototypimplementierung (Umsetzung in Hochsprache und Ausrichtung auf die Zielplattform, allerdings ohne Optimierung)

3. Optimierte Implementierung für Zielplattform (z.B. Digital Signal Processor (DSP))

Der Hauptteil dieser Arbeit beschäftigt sich mit der Auswahl der Algorithmen, der Prototypentwicklung und der Evaluierung. Um dabei schon auf die charakteristischen Besonderheiten der Zielplattform Rücksicht nehmen zu können, werden Systeme benötigt, die diese auf der Entwicklungsplattform simulieren. Plattformspezifische Funktionalität wird dabei in Form von Application Programming Interfaces (API) zur Verfügung gestellt. Diese APIs können durch Softwarebibliotheken nachgebildet und in ihrer Funktionalität simuliert werden. Im Unterschied zu Emulatoren bzw. Plattform Simulatoren wird bei Bibliotheken nur die Funktionalität simuliert, aber nicht deren Implementierung nachgebildet.
In anderen Worten:

- Bibliotheken ermöglichen Quellcodekompatibilität auf verschiedenen Systemen.
- Ein Simulator bzw. Emulator bietet Kompatibilität für kompilierten Binärcode.

Um quellcodekompatible Software für einen Texas Instruments DSP (TMS320C64xx) auf dem PC entwickeln zu können, wurde im Rahmen dieser Arbeit ein DSP-via-PC Framework erstellt. Dieses Framework besteht aus Bibliotheken, die häufig verwendete API Funktionen des TI-DSPs nachbilden. Neben der einfachen Funktionssimulation werden automatische Integritätsüberprüfungen durchgeführt und Tools zur Datenkonvertierung zur Verfügung gestellt.

5.2.2 Funktionsumfang

Es wurden das eXpressDSP Algorithm Interface Standard (XDAIS) Framework [5, 30] und grundlegende DSP Systemfunktionen nachgebildet:

- XDAIS Datentypen
- ALG – das XDAIS Algorithmus Framework
- DMAN / ACPY2 – das XDAIS DMA Modul
- LOG – das TMS320C64xx Ereignisprotokoll
- MEM – die TMS320C64xx spezifische Speicherverwaltungs-Funktionen

XDAIS Datentypen

Die gebräuchlichsten XDAIS Datentypen und Konstanten werden auf äquivalente Microsoft Visual C++ Datentypen abgebildet.

ALG – das XDAIS Algorithmus Framework

Dieses Modul wird benötigt, um einen XDAIS Algorithmus in einer Applikation verwenden zu können. Die Kernfunktionen sind dabei das Erzeugen und Löschen einer Algorithmusinstanz. Während der Initialisierung wird der Speicherbedarf des Algorithmus ermittelt (interner und externer bzw. scratch und persistent Speicher) und zugewiesen. Diese Speicherzuweisung, insbesondere die Größe des Speichers, kann während der Laufzeit nicht mehr vom Algorithmus geändert werden. Die ALG Emulation reserviert alle Speicherblöcke im Hauptspeicher des PCs unabhängig von den DSP spezifischen Speicherplätzen.

Außerdem sieht das Algorithmus Framework vor, dass eine Algorithmusinstanz vor der Ausführung des eigentlichen Algorithmus zuerst aktiviert und anschließend wieder deaktiviert werden muss. Diese Aktivierungs- und Deaktivierungsfunktionen des ALG Frameworks wurden im Debug Modus um einen, an den in [31] beschriebenen Stack Overflow Test angelehnten, Speicher Überlauf- und Unterlauf-Test erweitert. Dabei wird beim Anlegen der Instanz vor und nach den vom Algorithmus angeforderten Speicherblöcken ein bekanntes Muster (*canary words*) in den Speicher geschrieben. Die Aktivierungs- und Deaktivierungsfunktionen prüfen vor bzw. nach dem Starten des Algorithmus, ob der Speicher noch das korrekte Testmuster enthält.

DMAN / ACPY2 – das XDAIS DMA Modul

Dieses Modul stellt dem Algorithmus DMA-Funktionen zur Verfügung. Dabei ist die Funktionalität auf zwei Bibliotheken aufgeteilt:

- Der DMAN (DMA-Manager) übernimmt administrative Aufgaben wie das Reservieren von Ressourcen beim Erzeugen einer Algorithmusinstanz und das Freigeben beim Löschen.

- Die ACPY2 Bibliothek stellt die eigentliche DMA Funktionalität bereit.

In der Emulation wird der Zielbereich beim Starten eines DMA Transfers mit einem konstanten Muster beschrieben (`ACPY2_start()`). Der eigentliche Datentransfer mittels `memcpy()` erfolgt erst, wenn der Algorithmus die ACPY2 Funktion zum Warten auf die Beendigung des DMA Transfers aufruft (`ACPY2_wait()`). Somit kann leicht überprüft werden, ob der Algorithmus auf Bereiche im Zielspeicher zugreift, bevor sichergestellt wurde, dass die Übertragung abgeschlossen ist. Jene Funktion, die überprüft, ob der Transfer abgeschlossen ist (`ACPY2_complete()`), könnte in der Emulation immer einen erfolgreichen Datentransfer melden. Dadurch würde die Behandlung eines noch nicht abgeschlossenen Transfers in der Algorithmusimplementierung allerdings nie durchgeführt und getestet werden. Um die Reaktion der Algorithmusimplementierung in beiden Fällen (abgeschlossener bzw. nicht abgeschlossener Transfer) testen zu können, liefert die Emulation der `ACPY2_complete()` Funktion mit 50% Wahrscheinlichkeit die Meldung eines noch nicht abgeschlossenen Transfer, andernfalls werden die Daten mittels `memcpy()` kopiert und der Transfer als abgeschlossen bestätigt.

LOG – das TMS320C64xx Ereignisprotokoll

Dieses Modul emuliert die wichtigsten Funktionen des Ereignisprotokolls der TMS320-C64xx Serie, mit der Fehler und Statusmeldungen an das LOG System geschickt werden können. Die im LOG System gespeicherten Daten können am DSP mit geeigneten Werkzeugen ausgelesen werden, im DSP-via-PC Framework werden sie direkt auf der Konsole ausgegeben.

MEM – TMS320C64xx spezifische Speicherverwaltungsfunktionen

Dieses Modul stellt Speichermanagementfunktionen des DSPs zur Verfügung. Die DSP Funktionen berücksichtigen dabei TMS320C64xx spezifische Gegebenheiten wie unterschiedliche Speichersegmente (intern, extern) und Alignmentbedingungen. Die emulierten Funktionen bilden diese Speicherfunktionen auf Windows Speichermanagementfunktionen ab, ohne Speichersegment- oder Alignmenteinstellungen zu berücksichtigen.

5.3 Gesamtsystemintegration

Die beschriebenen Module werden im Prototyp zu einem Gesamtsystem zusammengefügt.

Dabei kann der Programmablauf in zwei Teile geteilt werden, die Systeminitialisierung, die einmalig beim Programmstart ausgeführt wird und alle Module startet und initialisiert, sowie die Hauptschleife, die für jedes Frame abgearbeitet wird und in der die eigentliche Bildverarbeitung und Auswertung durchgeführt wird. Der Programmablauf ist in Abbildung 5.2 dargestellt.

5.3.1 Systeminitialisierung

Das Datenbankmodul wird initialisiert und die temporäre Datenbank erzeugt, wobei der Speicherbereich, in dem die Datenbank angelegt werden soll, als Parameter übergeben wird (`DB_init()`).

Die Funktion `background_init()` wird aufgerufen, um das Hintergrundmodell (selektiver Approximated Median, siehe Kapitel 2.3) zu initialisieren, wobei die grundlegenden Bildparameter (Breite, Höhe und Anzahl der Farbkanäle) und eine initiale Standardabweichung σ sowie ein Updatewert für die Farbwerte κ und die Standardabweichung α_σ als Parameter übergeben werden.

Das Geometriemodul wird mit der Kameramatrix, der inversen Homographie und dem Korrekturvektor gestartet (`geometry_init()`).

Das Trackingmodul wird mit den grundlegenden Bildparametern (Breite, Höhe und Anzahl der Farbkanäle), einer Anfangsgeschwindigkeit v_{init}, der Größe des Objektmodells (Fahrzeuge) sowie der Definition des Einfahrtsbereiches und der zu überwachenden Fahrspuren initialisiert (`tracker_init()`), weiters wird das Objektmaskengewicht ω_m (siehe Kapitel 4.2.4) und ein Größenfaktor *scale* angegeben. Der Größenfaktor beschreibt die Größenanpassung der verwendeten Suchfenster für die dreistufige Trackingkaskade. Der minimale Kantenfaktor η gibt an, wie viele Kanten in Relation zum Umfang der Bounding-Box in einem Blob vorhanden sein müssen, damit dieser Blob als Kernblob eines neuen Objektes verwendet wird.

Das Statistikmodul wird mit dem für die Auswertung zu verwendenden Aktualisierungsintervall initialisiert (`statistic_init()`).

5.3.2 Hauptschleife

Die Hauptschleife wird pro Frame einmal ausgeführt.

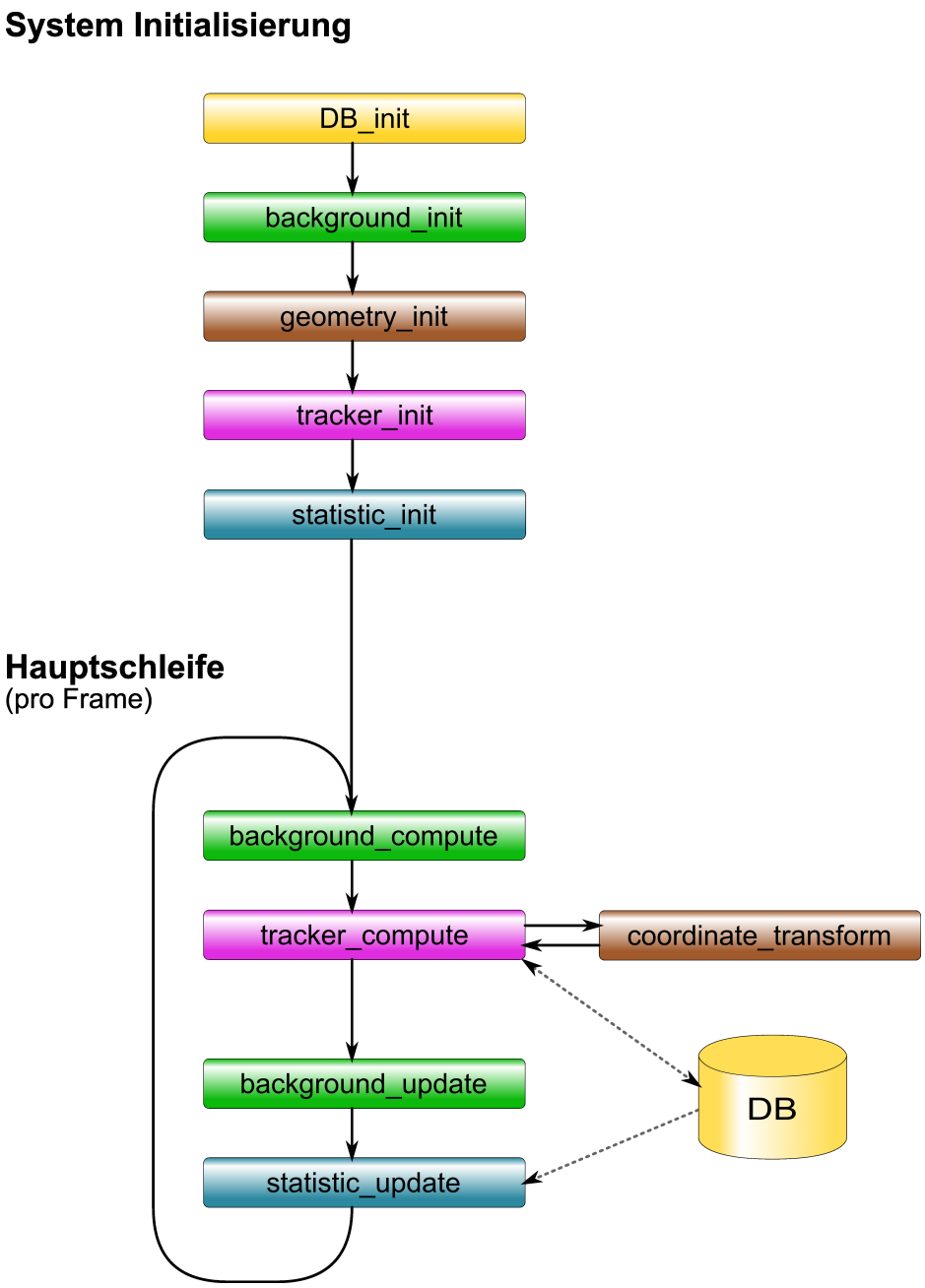


Abbildung 5.2: Diagramm Funktionensaufrufe im Gesamtsystem

Die Arbeit des Hintergrundmoduls wird in zwei Teile aufgeteilt. Zuerst wird die Funktion `background_compute()` aufgerufen, welche die globale Helligkeitsänderung und die Hintergrundstabilität ermittelt sowie die Vordergrundmaske erstellt.

Anschließend wird das Trackingmodul mit der Funktion `tracker_compute()` aufgerufen. In einem Vorverarbeitungsschritt wird die Objektposition im aktuellen Frame durch die alte Position und Geschwindigkeit geschätzt. Das Suchfenster des ersten Trackingdurchlaufes wird um den Größenfaktor *scale* größer gewählt, als die Größe der geschätzten Bounding-Box des Objektes für das aktuelle Frame. Der erste Trackingdurchlauf wird mit der zuvor geschätzten Objektposition und der vergrößerten Suchmaske gestartet. Das Suchfenster des zweiten Durchlaufes entspricht der Größe der geschätzten Bounding-Box und wird beim dritten Durchlauf um den Größenfaktor verkleinert.
Diese Größenkaskade beruht auf der Annahme, dass die wahre Position eines Objektes, das kleiner als die Größe des verwendeten Suchfensters ist, sich innerhalb der Ergebnis-Bounding-Box des histogrammbasierten Mean-Shift Algorithmus befindet und die Suche im nächsten Schritt mit einem kleineren Fenster verfeinert werden kann.
Danach erfolgt die Korrelation mit den Objektdaten (siehe Kapitel 4.2.1). Blobs, die keinem bekannten Objekt zugeordnet werden konnten und die Bedingungen zur Erstellung eines neues Objektes erfüllen, erzeugen eine neue Objektinstanz (siehe Kapitel 4.2.3). Abschließend werden die neue Objektposition mittels Funktionen des Geometriemoduls in Weltkoordinaten umgerechnet und die Objekteigenschaften (z.B. Geschwindigkeit, Stabilität, Fahrbahn, etc.) aktualisiert.

Anschließend wird die zweite Hauptfunktion des Hintergrundmoduls `background_update()` ausgeführt, die aufgrund der Vordergrundmaske und den vom Tracker erhaltenen Informationen eine Aktualisierung des Hintergrundmodells (selektiver Approximated Median Algorithmus mit globaler Helligkeitsanpassung) durchführt.

Das Statistikmodul wird aufgerufen, um eine Aktualisierung der Statistik entsprechend den aus der Datenbank gelesenen neuen Objektdaten durchzuführen (`statistic_update()`).

6 Evaluation

In diesem Kapitel werden die Evaluation des Prototypen und die Ergebnisse beschrieben. Dabei werden zuerst die Detektions- bzw. Trackingqualität der beiden Hauptmodule (Objektinitialisierung sowie Tracker) einzeln ermittelt, anschließend wird das Gesamtsystem evaluiert und die Resultate beschrieben. Die darauf folgende Laufzeitevaluation wurde bewusst kurz gehalten, da es sich um eine unoptimierte Prototypimplementierung handelt und somit nur prinzipielle Laufzeitabschätzungen möglich sind.

6.1 Evaluationsmethode

Als Basis der Evaluierung werden die Bounding-Boxen der gefundenen Objekte verwendet. Dabei wird die Übereinstimmung mit den Bounding-Boxen aller Ground Truth Objekte des jeweiligen Frames mittels einer distanzähnlichen[1] Funktion d (Gleichung 6.1) bestimmt. Dabei ist die Funktion d unabhängig von der absoluten Größe der Bounding-Boxen und kann Werte von 0 bis 1 annehmen.

Area(BB_t): Fläche der Bounding-Box des getrackten Objekts BB_t
Area(BB_g): Fläche der Bounding-Box des Ground Truth Objekts BB_g

$$d(BB_t, BB_g) = 1 - \frac{2\,\text{Area}(BB_t \cap BB_g)}{\text{Area}(BB_t) + \text{Area}(BB_g)} \tag{6.1}$$

Als Ground Truth Daten werden die Bounding-Box, eine eindeutige Objektnummer (ID) und der Typ jedes Objekts verwendet. Die Bounding-Box dient als Positionsangabe und wird zur Qualitätsbestimmung des Trackings verwendet. Die ID wird benötigt, um eine Relation der Objekte zwischen verschiedenen Frames herstellen zu können.

Für die synthetisch generierten Testsequenzen wurden die Ground Truth Daten gemeinsam mit den Bildern erzeugt, wobei jedes Frame genau ein Objekt vom Typ *unknown*

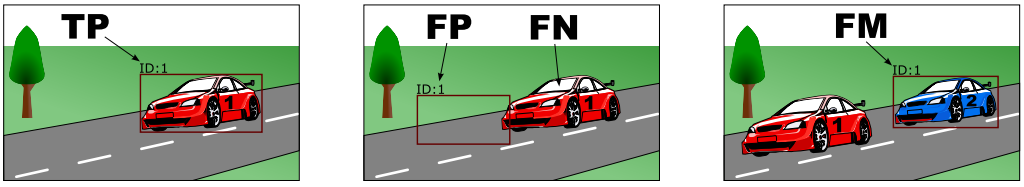
[1] $d(t, g)$ ist nicht transitiv, daher auch keine Distanz im mathematischem Sinn.

enthält. Die Ground Truth Daten der realen Testsequenzen wurden im Rahmen des Video-Based Image Analysis for Tunnel Safety (VITUS) Projekts [32] manuell erstellt, wobei nicht nur Fahrzeuge, sondern auch Schatten, Reflexionen und andere Unterschiede zum Hintergrundmodell annotiert und entsprechend klassifiziert wurden. Bei der Evaluierung werden allerdings nur für die Anwendung interessante Objekte berücksichtigt (Objekte vom Typ *CAR* für reale Testsequenzen bzw. *unknown* für generierte Testsequenzen).
Die verwendeten Testsequenzen werden in Kapitel 6.2 genauer beschrieben.

Eine True Positive (TP) Klassifizierung liegt vor, wenn das Ergebnis-Event eines Algorithmus dem erwarteten Ergebnis-Event der Ground Truth entspricht. Für die Detektion bzw. das Tracking bedeutet das, dass die Abweichungen der Bounding-Box eines getrackten Objekts t von den entsprechenden Daten des Ground Truth Objektes g innerhalb der Detektionstoleranz θ liegen ($d(t, g) < \theta$).
Existiert ein Ground Truth Event und kein entsprechendes Ergebnis-Event des Algorithmus, so wird dies als False Negative (FN) bezeichnet. Bei der Detektion bzw. dem Tracking liegt dieser Fall vor, wenn einem Ground Truth Objekt kein detektiertes bzw. getracktes Objekt zugeordnet werden kann. Das heißt, es existiert keine Bounding-Box, die innerhalb der zulässigen Abweichungen einer Ground Truth Bounding-Box liegt.
Existiert zu einem Ergebnis-Event des Algorithmus kein entsprechendes Ground Truth Event, so handelt es sich um ein False Positive (FP) Event.

Neben diesen in der Statistik gebräuchlichen Kennzahlen wird zur Bewertung der Trackingdaten eine weitere Größe definiert: False Match (FM). Ein False Match tritt auf, wenn die getrackte Bounding-Box eines Objekts der Ground Truth Bounding-Box eines anderen Objekts (mit unterschiedlicher ID) zugeordnet wird. Dies ist der Fall, wenn der Tracker von einem Objekt auf ein anderes umgesprungen ist.

Die beschriebenen Fehlerkategorien sind in Abbildung 6.1 illustriert.



(a) true positive (TP) Bounding-Box

(b) false positive (FP) und false negative (FN) Fehler

(c) false match (FM) Fehler

Abbildung 6.1: statistische Fehlerkategorien

Mit diesen Fehlerkategorien können folgende Funktionen zur Evaluierung eines Detektions bzw. Tracking Algorithmus definiert werden (vgl. [33]):

- Tracker Detection Rate: $TRDR = \frac{TP_t}{(TP_t+FN_t)}$
- False Alarm Rate: $FAR = \frac{FP_t}{(TP_t+FP_t)}$
- Track Fragmentation: TF = Anzahl der Trajektorien eines getrackten Objekts, die einer Ground Truth Trajektorie entsprechen

TP_t, FN_t und FP_t stehen dabei für die Gesamtzahl (*total*) der true positive, false negative bzw. false positive Bewertungen einer Sequenz.

Die Tracker Detection Rate ($TRDR$) gibt an, wie gut die Objekte bei vorgegebener Toleranz getrackt werden. Die False Alarm Rate (FAR) ist ein Maß dafür, wie viele Trackingergebnisse keinem Ground Truth Objekt entsprechen.
Werden vom Tracker keine Objekte erstellt oder verworfen, und werden alle Objekte über die gesamte Sequenz getrackt, so gilt: $TRDR + FAR = 1$
Die Track Fragmentation (TF) gibt an, wie oft ein getracktes Objekte neu initialisiert wurde, das heißt wie viele Tracking-IDs einem Ground Truth Objekt gegeben wurden. Die Track Fragmentation ist pro Ground Truth Objekt definiert. Um eine ähnliche Maßzahl für eine gesamte Sequenz mit mehreren Objekten zu erhalten, wird die normalized Track Fragmentation (nTF) verwendet.

$$nTF = \frac{TF}{\text{Anzahl der Ground Truth Objekte}} \tag{6.2}$$

Im Idealfall ist nTF gleich 1 (bzw. für jedes Objekt: TF gleich 1), das heißt zu jedem Ground Truth Objekt existiert genau ein getracktes Objekt mit durchgehend passender Trajektorie.

Die False Match Rate (FMR) wird, ähnlich wie die $TRDR$, als Verhältnis zur Gesamtzahl der Ground Truth Objekte ($TP_t + FN_t$) definiert:

- False Match Rate: $FMR = \frac{FM_t}{(TP_t+FN_t)}$

FM_t stehen dabei für die Gesamtzahl der False Match Bewertungen einer Sequenz.

Die Ergebnisse der Messungen (Objektnummer, Bounding-Box und Type der getrackten Objekte) werden gespeichert und automatisch mit den Ground Truth Daten verglichen.

6.2 Testsequenzen

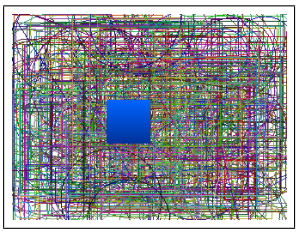
Die Implementierung des Algorithmus wurde sowohl mittels synthetisch erzeugter Sequenzen als auch mittels realer Videos evaluiert. Eine Übersicht der Testsequenzen befindet sich in Tabelle 6.1.

Sequenzname	Abb.	Frames	Objekte	Kurzbeschreibung
noise_box	6.2(a)	500	1	blaues Quadrat mit konstanter Größe bewegt sich auf gemustertem Hintergrund
zoom_box	6.2(b)	500	1	blaues Quadrat bewegt sich auf weißem Hintergrund und verändert dabei seine Größe
noise_circle	6.2(c)	500	1	mehrfärbige Zielscheibe bewegt sich auf gemustertem Hintergrund
k131-1_3_2	6.3	500	1	PKW in geradem Tunnelabschnitt verliert Ladegut (Plabutsch Kamera 131, in Fahrtrichtung)
k131-1_5_3	6.4	500	7	Fahrzeugkolonne in geradem Tunnelabschnitt (Plabutsch Kamera 131, in Fahrtrichtung)
Altmannsdorf_08	6.5	900	40	mehrere Gruppen von PKWs, Motorrädern sowie ein Autobus (A23, gegen Fahrtrichtung)
Altmannsdorf_09	6.6	400	10	nebeneinander fahrende PKWs (A23, in Fahrtrichtung)
Altmannsdorf_10	6.7	350	4	langgezogene Gruppe von PKWs (A23, in Fahrtrichtung)

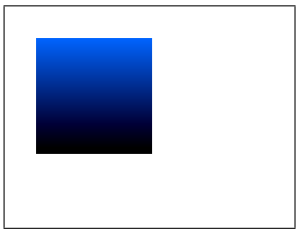
Tabelle 6.1: Beschreibung der Testsequenzen

In den synthetischen Testsequenzen (*noise_box* und *noise_circle*) wurden einfache geometrische Figuren (Quadrat bzw. Kreis) verwendet, die mit einer nicht monochromen Textur versehen wurden. Dabei bewegt sich das Objekt mit konstanter Größe zuerst horizontal von links nach rechts und wieder zurück, danach ist die Bewegung zufällig. Für jede Objektklasse wurde eine Sequenz mit einem zufälligem Hintergrundmuster erzeugt. Zusätzlich wurde eine Testsequenz (*zoom_box*) zur Evaluation der Größenadaption des Algorithmus erstellt. Dabei wird ein Quadrat an fixer Position zuerst verkleinert und wieder vergrößert, anschließend ist die Änderung der Position und Größe zufällig. Abbildung 6.2 zeigt Beispielbilder der synthetischen Testsequenzen.

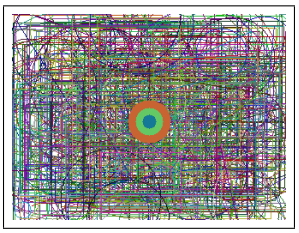
Als reale Testsequenzen wurden annotierte Videosequenzen von fest installierten Kameras im Plabutsch Tunnel verwendet, sowie von einer mobilen Kamera, mit der Aufnahmen von der A23 (Altmannsdorfer Ast) gemacht wurden.
Bei den Aufnahmen aus dem Plabutsch Tunnel handelt es sich um nachgestellte Szenen, die im Dezember 2004 erstellt wurden. Sequenz *k131-1_3_2* zeigt einen geraden Tunnelabschnitt mit Pannenbucht auf der rechten Seite, die Kamera blickt in Fahrtrichtung. Ein PKW fährt auf der rechten Spur und verliert Ladegut (ein Müllsack, zwei Kartons) aus der Heckklappe (siehe Abbildung 6.3). Es wird nur der PKW getrackt. Sequenz *k131-1_5_3* stammt von der selben Kamera und zeigt eine Szene, in der mehrere PKWs mit geringem Abstand (ca. 2 bis 3-fache Fahrzeuglänge) auf der rechten Spur fahren (siehe Abbildung 6.4).
Die Videos vom Altmannsdorfer Ast zeigen typischen Nachmittagsverkehr mit mittlerer Verkehrsdichte. Dabei ist zu beobachten, dass es immer wieder kurze Zeiten ohne Fahrzeuge gibt, gefolgt von einer kleinen Gruppe von 2–3 Autos. Sequenz *Altmannsdorf_408* zeigt einen geraden Autobahnabschnitt in dem Fahrzeuge von zwei Zufahrten auf insgesamt drei Fahrspuren zusammen kommen. Die Kamera blickt mittig auf die Fahrbahn, gegen die Fahrtrichtung. Dies ist die längste Testsequenz und zeigt 40 Fahrzeuge, wovon 3 Motorräder sind, 1 Autobus und 36 PKWs. Die Sequenzen *Altmannsdorf_409* und *Altmannsdorf_410* zeigen jeweils den selben zweispurigen Autobahnabschnitt, wobei die Kamera in Fahrtrichtung blickt. In Video *Altmannsdorf_409* fahren mehrfach zwei PKWs nebeneinander bzw. überholen einander. In der Sequenz *Altmannsdorf_410* ist eine langgestreckte Gruppe (über die halbe Videosequenz) von 6 PKWs zu sehen, anschließend befindet sich kein Fahrzeug mehr auf der Fahrbahn. Diese Sequenz dient als Testsequenz für False Positive Events.



(a) Sequenz: noise_box



(b) Sequenz: zoom_box



(c) Sequenz: noise_circle

Abbildung 6.2: Beispielbilder der generierten Testsequenzen

(a) Start der Sequenz

(b) Mitte der Sequenz

(c) Ende der Sequenz

Abbildung 6.3: Beispielbilder der Testsequenz: k131-1_3_2

(a) Start der Sequenz

(b) Mitte der Sequenz

(c) Ende der Sequenz

Abbildung 6.4: Beispielbilder der Testsequenz: k131-1_5_3

(a) Start der Sequenz (b) Mitte der Sequenz (c) Ende der Sequenz

Abbildung 6.5: Beispielbilder der Testsequenz: Altmannsdorf_08

(a) Start der Sequenz (b) Mitte der Sequenz (c) Ende der Sequenz

Abbildung 6.6: Beispielbilder der Testsequenz: Altmannsdorf_09

(a) Start der Sequenz (b) Mitte der Sequenz (c) Ende der Sequenz

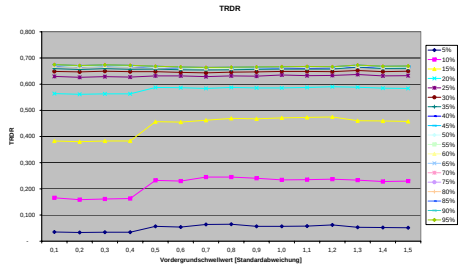
Abbildung 6.7: Beispielbilder der Testsequenz: Altmannsdorf_10

6.3 Resultate der Objektinitialisierung

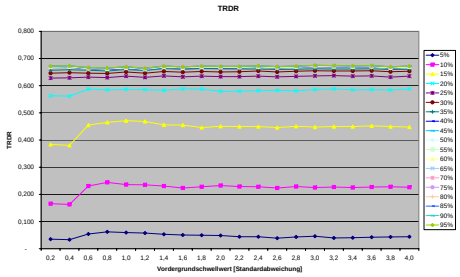
Bevor Objekte (Fahrzeuge) getrackt werden können, muss zuerst festgestellt werden, wo sich noch nicht getrackte Objekte befinden, und eine neue Objektinstanz angelegt werden. In diesem Kapitel wird die Objektinitialisierung der Prototypimplementierung evaluiert. Die Bounding-Boxen der Blobs der Vordergrundmaske liefern neue Objektkandidaten (siehe Kapitel 4.2.3). Die Bounding-Boxen der neuen Objekte werden mit den Ground Truth Bounding-Boxen der Fahrzeuge des jeweiligen Frames verglichen.

Zuerst wird die Tracker Detection Rate ($TRDR$) bzw. False Alarm Rate (FAR) für die Testsequenz *Altmannsdorf_ 08* in Abhängigkeit vom Vordergrundschwellwert (γ) ermittelt. Anschließend werden die $TRDR$ und FAR über alle Testsequenzen ermittelt und als Funktion über die Detektionstoleranz θ dargestellt.

Der Vordergrundschwellwert γ ist der Schwellwert der Segmentierung der (durch die globale Standardabweichung normierten) Maske des Hintergrundmodells. Wie die Diagramme 6.8(a) 6.9(a) zeigen, gibt es zwischen $\gamma = 0,4$ und $\gamma = 0,5$ einen Sprung der $TRDR$ und FAR, zwischen $\gamma = 0,5$ und $\gamma = 1,3$ gibt es keine wesentliche Veränderung der Werte, ab $\gamma > 1,3$ fällt die $TRDR$ leicht ein und die FAR steigt geringfügig an, wie in den Diagrammen 6.8(b) und 6.9(b) bei größerer Skalierung zu sehen ist. Aufgrund dieser Ergebnisse wird der Vordergrundschwellwert γ für die weitere Evaluation auf $\gamma = 1,3$ gesetzt.



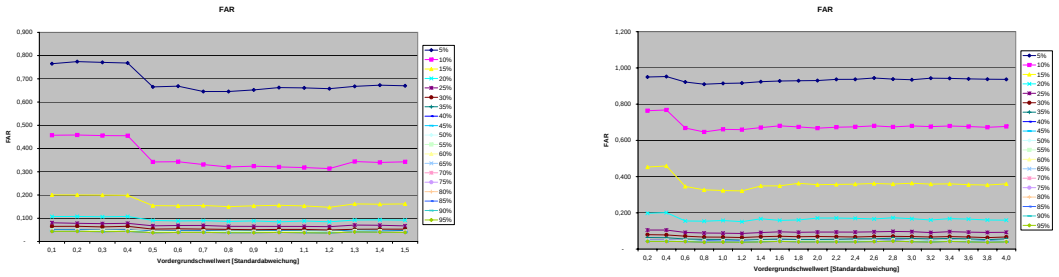
(a) Vordergrundschwellwert γ = 0,1 ... 1,5



(b) Vordergrundschwellwert γ = 0,2 ... 4,0

Abbildung 6.8: Tracker Detection Rate ($TRDR$) der Objektinitialisierung in Abhängigkeit vom Vordergrundschwellwert γ (als Vielfaches der Standardabweichung) der Testsequenz: *Altmannsdorf_ 08*

Die Objektinitialisierung (und somit das Hintergrundmodell) wurde mit verschiedenen



(a) Vordergrundschwellwert $\gamma = 0{,}1 \ldots 1{,}5$ (b) Vordergrundschwellwert $\gamma = 0{,}2 \ldots 4{,}0$

Abbildung 6.9: False Alarm Rate (FAR) der Objektinitialisierung in Abhängigkeit vom Vordergrundschwellwert γ (als Vielfaches der Standardabweichung) der Testsequenz: *Altmannsdorf_ 08*

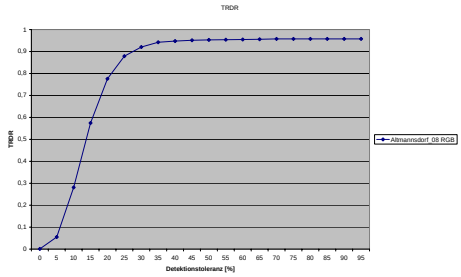
Farbmodellen getestet. Dabei wurde die Objektinitialisierung zuerst auf drei Testsequenzen evaluiert, um interessante Farbräume auswählen zu können, die anschließend mit allen Testsequenzen evaluiert werden. Um eine möglichst gute Abdeckung der Testsequenzen zu erzielen, wurden je eine Tunnelsequenz (*k131-1_ 3_ 2*), eine Freilandsequenz (*Altmannsdorf_ 08*) und eine synthetisch erzeugten Sequenz (*noise_ circle*) ausgewählt. Als Farbräume wurden der RGB, HSV und YCrCb Farbraum sowie die HS, HV, SV, H, S, V, Y und $CrCb$ Teilfarbräume verwendet.

Die Ergebnisse der synthetisch erzeugten Sequenz sind bei allen vollen, 3-dimensionalen Farbräumen und auch bei den Teilfarbräume sehr gut, lediglich der SV Teilraum und die 1-dimensionalen Teilräume schneiden hier etwas schlechter ab und erreichen die optimale $TRDR$ erst bei einer Detektionstoleranz θ größer als 10%. Insgesamt liefert die synthetisch Sequenz keine Präferenz einzelner Farbräume, daher werden nur die Diagramme der beiden realen Sequenzen abgebildet und näher analysiert.

Die Ergebnisse der Freiland- und Tunnelsequenz sind in Diagrammen (Abbildung 6.10) dargestellt, dabei werden alle Teilfarbräume mit dem jeweiligen vollen Farbraum in einem Diagramm zusammengefasst. In der linken Spalte befinden sich alle Diagramme der Freilandsequenz (*Altmannsdorf_ 08*), in der rechten Spalte die der Tunnelsequenz (*k131-1_ 3_ 2*).

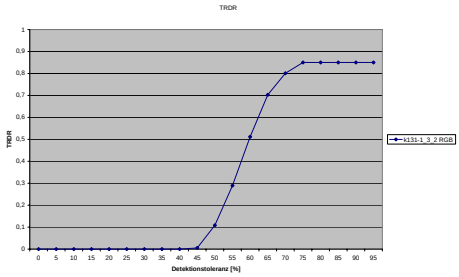
Die Ergebnisse der einzelnen Farbmodelle fallen dabei höchst unterschiedlich aus, doch obwohl die absoluten Zahlen zwischen den beiden Sequenzen stark variieren, ist die relative Ordnung der (Teil-)Farbräume sehr ähnlich.

Sequenz:
Altmannsdorf_ 08

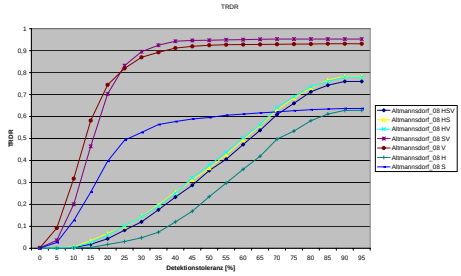


(a) RGB

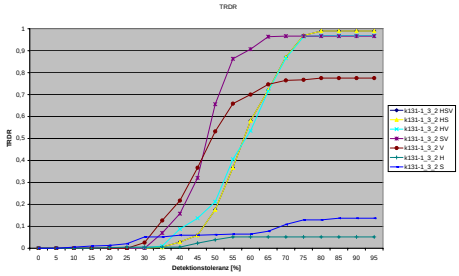
Sequenz:
k131_ 1_ 3_ 2



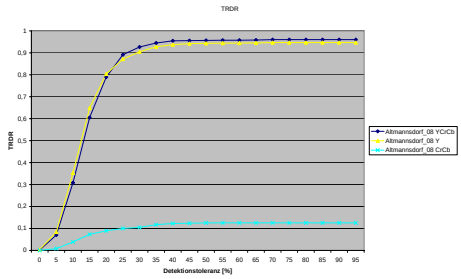
(b) RGB



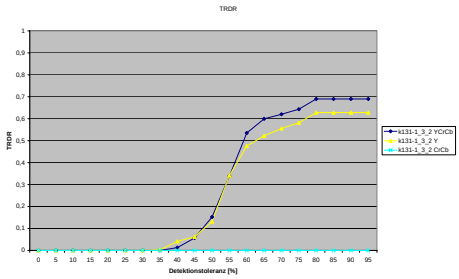
(c) HSV



(d) HSV



(e) YCrCb



(f) YCrCb

Abbildung 6.10: Tracker Detection Rate ($TRDR$) der Objektinitialisierung unterschiedlicher Farbmodelle

Das RGB Farbmodell liefert für beide Sequenzen gute Ergebnisse, für die Freilandsequenz zählt es sogar zu den besten der getesteten Farbmodelle.

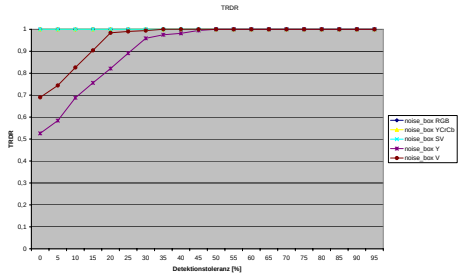
Im HSV Farbraum sind die HSV, HS und HV Farbmodelle sehr ähnlich, die Kurven überlagern sich sogar teilweise, die Gesamtergebnisse liegen aber im Mittelfeld. Da auch die Ergebnisse des reinen H Kanals als schlecht zu bewerten sind, kann angenommen werden, dass die Ergebnisse der HSV, HS & HV Gruppe durch Fehler im H Kanal verschlechtert werden. Dies wird auch durch die Kurven der S und V Kanäle untermauert, die nahe an, manchmal sogar über der HSV Kurve liegen. Eine Erklärung für den destruktiven Einfluss des Farbtonkanals (H) ist die allgemein niedrige Farbsättigung der Fahrzeugfarben und die damit verbundene Verstärkung von kleinen Schwankungen des Farbtons im H Kanal. Das beste Ergebnis dieses Farbraums liefert der SV Teilraum bzw. der V Kanal alleine (siehe Abbildung 6.10(c) und 6.10(d)).

Die reinen Farbkanäle $CrCb$ des $YCrCb$ Farbraums liefern die schlechtesten Ergebnisse des gesamten Testreihe, können aber in Kombination mit dem Helligkeitskanal $YCrCb$ die Ergebnisse gegenüber dem reinen Helligkeitskanal Y noch verbessern. In der Freilandsequenz zählt das $YCrCb$ Farbmodell zu den drei besten Modellen, bei der Tunnelsequenz ist es aber im hinteren Mittelfeld einzureihen.

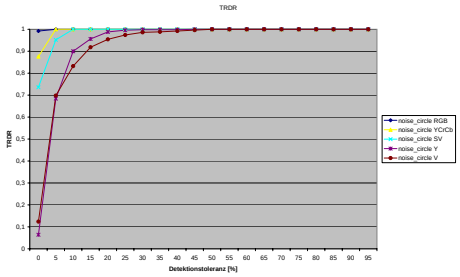
Aufgrund dieser Untersuchung werden folgende Farbmodelle mit allen Testsequenzen nochmals evaluiert: RGB, $YCrCb$, SV, Y, V.
Dabei wurden die drei Farbräume RGB, $YCrCb$ und SV aufgrund der guten Ergebnisse der vorigen Untersuchung ausgewählt. Die beiden einkanaligen Modelle Y und V wurden ausgewählt, um in Hinblick auf eine DSP Implementierung auch einfache und ressourcenschonende Methoden zu evaluieren.

Abbildungen 6.11 und 6.12 zeigen die $TRDR$-Diagramme der Objektinitialisierung der fünf ausgewählten Farbmodelle über alle synthetischen bzw. realen Testsequenzen. Dabei sind die Ergebnisse aller Farbräume bzw. Teilfarbräume einer Sequenz in einem Diagramm zusammengefasst.

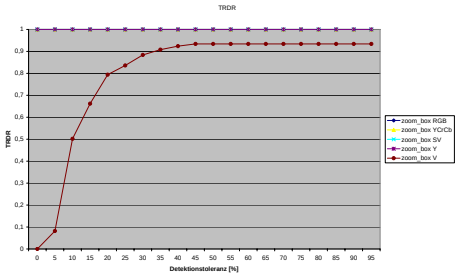
Die $TRDR$ für die mehrdimensionalen Farbmodelle (RGB, $YCrCb$ und SV) erreicht für alle synthetischen Sequenzen ($noise\_box$, $noise\_circle$ und $zoom\_box$) bei einer Detektionstoleranz $\theta \geq 10\%$ das Optimum (siehe Abbildung 6.11). In anderen Worten: Die Überdeckung der Ground Truth Bounding-Box mit der zur Objektinitialisierung verwendeten Bounding-Box beträgt in allen Frames mindestens 90% der Bounding-Box Fläche.
Wird nur ein Helligkeitskanal (Y oder V) verwendet, sinkt die $TRDR$, gerade bei geringen Detektionstoleranzen, deutlich ab. In Diagram 6.11(c) ist dies besonders deutlich zu erkennen, außerdem tritt hier noch ein systematischer Fehler im V Farbmodell auf, da sich die Dunkelstufe des Rechteckes (blauer hell-dunkel Farbverlauf) in bestimmten



(a) noise_box

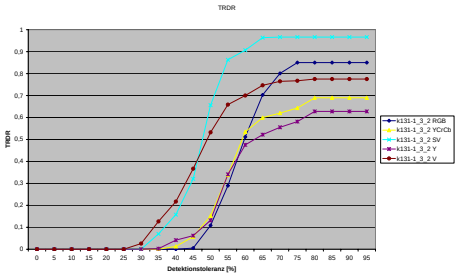


(b) noise_circle

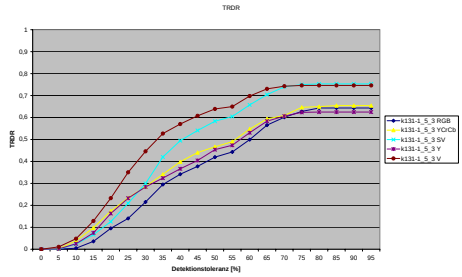


(c) zoom_box

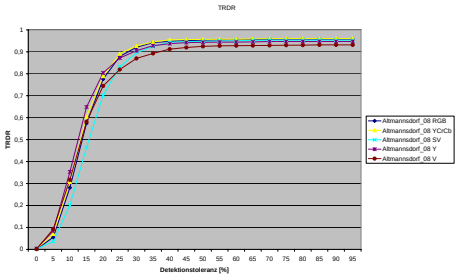
Abbildung 6.11: Tracker Detection Rate ($TRDR$) der Objektinitialisierung der fünf ausgewählten Farbmodelle für die synthetischen Testsequenzen



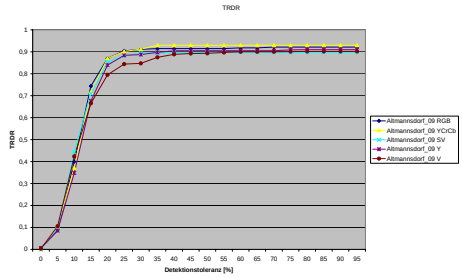
(a) k131-1_3_2



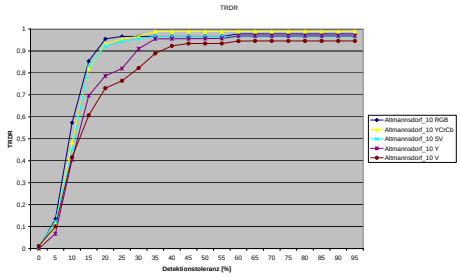
(b) k131-1_5_3



(c) Altmannsdorf_08



(d) Altmannsdorf_09



(e) Altmannsdorf_10

Abbildung 6.12: Tracker Detection Rate ($TRDR$) der Objektinitialisierung der fünf ausgewählten Farbmodelle für die realen Testsequenzen

Bereichen nicht ausreichend vom Hintergrund unterscheidet.

Die $TRDR$ Diagramme der Tunnelsequenzen (*k131-1_3_2* und *k131-1_5_3*) unterscheiden sich deutlich von denen der Freilandautobahn (*Altmannsdorf_08*, *Altmannsdorf_09* und *Altmannsdorf_10*). Während die $TRDR$ der Tunnelsequenzen erst bei einer Detektionstoleranz $\theta > 60\%$ ein lokales Maximum zu erreichen beginnt, steigt die $TRDR$ bei den Freilandsequenzen steiler und bei einer geringeren Detektionstoleranz zu einem Maximum an.

Bei den Tunnelsequenzen liefert das 2-dimensionale SV und das 1-dimensionale V Farbmodell gute Resultate im Vergleich zu den restlichen evaluierten Farbmodellen. Außerdem ist zu beobachten, dass die Dunkelstufe V (als Maximum der RGB Farben) deutlich bessere $TRDR$ Ergebnisse liefert als der Grauwertkanal Y (gewichtetes Mittel der RGB Farben), obwohl beide ein Maß für die Helligkeit eines Pixels darstellen (siehe Abbildung 6.12(a) und 6.12(b)). Die $TRDR$ Kurven der einzelnen Farbmodelle bei den synthetischen und den Freilandsequenzen liegen zu dicht nebeneinander, um hier eine eindeutige Beobachtung treffen zu können.
Zwei grundlegende Probleme sind bei den Tunnelsequenzen identifizierbar, die die schlechte $TRDR$ der Objektinitialisierung erklären. Einerseits treten Schatten und Spiegelungen der Scheinwerfer in der Nähe der Fahrzeuge auf, die zu einer Verfälschung der Bounding-Box führen, dies ist besonders bei der *k131-1_3_2* Sequenz zu beobachten. Im $TRDR$ Diagramm (Abbildung 6.12(a)) zeigt sich dies durch einen verspäteten (θ um 50%), dann aber steilen Anstieg der $TRDR$ zu einem hohen Maximum. Andererseits verschmelzen bei Kolonnenverkehr mehrere Fahrzeuge zu einer großen Bounding-Box (Sequenz: *k131-_5_3*).

Die konstante $TRDR$ bedeutet, dass der Fehler zwischen der detektierten Bounding-Box und der entsprechenden Ground Truth Bounding-Box der Detektionstoleranz θ am Knickpunkt der $TRDR$ Kurve entspricht. Erreicht die Kurve dabei nicht den optimalen Wert von $TRDR = 1$, so bedeutet dies, dass einige Ground Truth Objekte in einzelnen Frames gar nicht erkannt wurden. Dies betrifft insbesondere Objekte, die gerade in die Region of Interest (ROI) ein- bzw. ausfahren, da nur Objekte gezählt werden, deren Basispunkt (bzw. Blobmittelpunkt bei Ground Truth Objekten) in der ROI liegt. Liegt der Basispunkt eines Objektes schon außerhalb, der Blobmittelpunkt des korrespondierenden Ground Truth Objektes allerdings noch in der ROI, so wird das Objekt als FN klassifiziert und dadurch die $TRDR$ verringert.

6.4 Resultate des Mean-Shift Trackings

Dieses Kapitel beschreibt die Resultate der Evaluation des Trackers. Dabei wird der Tracker mit den Ground Truth Daten der Objekte initialisiert. Kann ein Objekt vom Tracker nicht mehr weiter verfolgt werden, wird im nächsten Frame ein neues Objekt mittels der Ground Truth Daten erzeugt. Jedes Mal wenn ein Objekt neu initialisiert wird, wird die Track Fragmentation TF um eins erhöht. Da jedes Ground Truth Objekt somit mindestens ein Mal initialisiert wird, ist die optimale normalized Track Fragmentation $nTF = 1$ (siehe Gleichung 6.2). Die tatsächliche normalized Track Fragmentation nTF der Testsequenzen ist in Tabelle 6.2 zusammengefasst.
In der darauf folgenden Tabelle 6.3 ist die False Match Rate FMR der Sequenzen bei einer Detektionstoleranz $\theta = 50\%$ dargestellt.

	normalized Track Fragmentation				
Sequenzname	*RGB*	*YCrCb*	*SV*	*Y*	*V*
noise_box	1,00	1,00	1,00	1,00	1,00
zoom_box	16,00	10,00	16,00	12,00	2,00
noise_circle	1,00	1,00	1,00	1,00	1,00
k131-1_3_2	2,00	2,00	2,00	2,00	3,00
k131-1_5_3	1,29	1,86	1,14	1,86	1,71
Altmannsdorf_08	1,00	1,08	1,03	1,03	1,05
Altmannsdorf_09	1,00	1,00	1,00	1,00	1,00
Altmannsdorf_10	1,00	1,00	1,00	1,00	1,00

Tabelle 6.2: normalized Track Fragmentation des Mean-Shift Trackers

	False Match Rate				
Sequenzname	*RGB*	*YCrCb*	*SV*	*Y*	*V*
noise_box	0,00	0,00	0,00	0,00	0,00
zoom_box	0,00	0,00	0,00	0,00	0,00
noise_circle	0,00	0,00	0,00	0,00	0,00
k131-1_3_2	0,00	0,00	0,00	0,00	0,00
k131-1_5_3	0,00	0,05	0,00	0,00	0,01
Altmannsdorf_08	0,00	0,00	0,00	0,00	0,01
Altmannsdorf_09	0,00	0,00	0,00	0,00	0,00
Altmannsdorf_10	0,00	0,00	0,00	0,00	0,00

Tabelle 6.3: False Match Rate FMR des Mean-Shift Trackers (Detektionstoleranz $\theta = 50\%$)

Die Tracker Detection Rate $TRDR$ wurde mit den selben fünf Farbmodellen evaluiert,

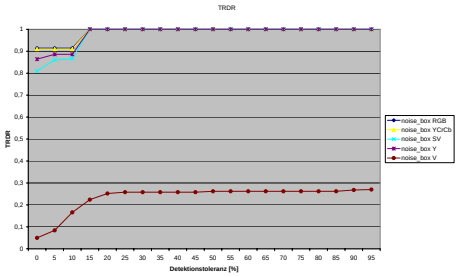
wie auch die Objektinitialisierung (siehe Abbildung 6.13 und 6.14).

Die Analyse der Tracker Detection Rate ($TRDR$) zeigt, dass der Tracking Algorithmus bei Objekten mit konstanter Größe gute Ergebnisse liefert. Die Diagramme 6.13(a) und 6.13(b) stellen die $TRDR$ in Abhängigkeit von der Detektionstoleranz θ für verschiedene Farbmodelle der *noise_ box* bzw. *noise_ circle* Sequenz dar.
Für rechteckige Objekte mit konstanter Größe (*noise_ box*) wird die optimale $TRDR = 1$, außer für das V-Farbmodell, bei einer Detektionstoleranz $\theta \geq 15\%$ erreicht. Da im Dunkelkanal V sowohl die Objektfarbe als auch der Hintergrund einen sehr ähnlichen Wert (nahe dem Maximum von 255) aufweisen, konnte das Quadrat im V-Farbmodell nicht korrekt getrackt werden, was deutlich im $TRDR$ Diagramm zu sehen ist.
Der Kreis (*noise_ circle*) wurde ab einer Detektionstoleranz $\theta \geq 30\%$ mit einer $TRDR = 1$ erkannt, lediglich das Y-Farbmodell weist etliche Ausreißer auf, so dass sich die $TRDR$ nur langsam an das Optimum annähert. Die allgemein höhere Detektionstoleranz der Kreissequenz lässt sich dadurch erklären, dass ein Versatz der Bounding-Box bei nicht rechteckigen Objekten weniger echte Objektpixel betrifft, als bei rechteckigen Objekten. Dadurch ist auch der korrigierende Mean-Shift Vektor entsprechend schwächer und die resultierende Bounding-Box ungenauer als bei rechteckigen Objekten.

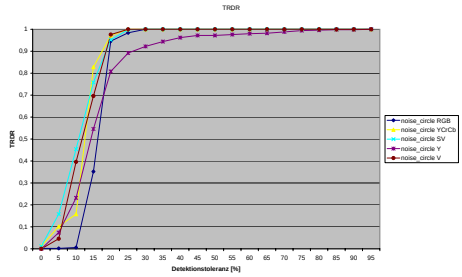
Die normalized Track Fragmentation nTF der Sequenzen mit konstanter Objektgröße ist für alle Farbmodelle gleich 1. Das heißt der Tracker hat das Objekt niemals verloren, selbst bei False Positive Fällen (wenn die Überdeckung der getrackten Bounding-Box mit der Ground Truth Bounding-Box kleiner als die Detektionstoleranz θ ist) konnte das Objekt später wieder gefunden werden, ohne dass es neu initialisiert werden musste.

Bei der synthetischen Testsequenz mit variabler Objektgröße *zoom_ box* steigt die nTF auf über 10 fragmentierte Tracks, lediglich beim V Farbmodell existieren weniger fragmentierte Tracks, aber hier konnte das Objekt so gut wie gar nicht getrackt werden. Dies und die schlechte $TRDR$ dieser Sequenz (Diagramm 6.13(c)) zeigen ein prinzipielles Problem des Mean-Shift Trackers auf: Das Verfolgen von Objekten mit variabler Objektgröße.

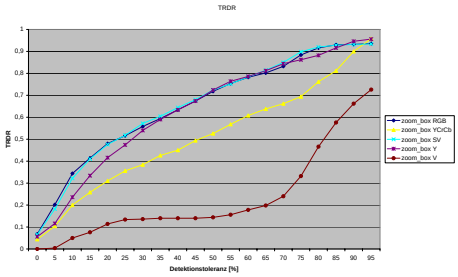
Im Idealfall wäre eine $TRDR$ ähnlich der $TRDR$ des Quadrates mit konstanter Größe (Diagramm 6.13(c)) zu erwarten. Stattdessen ist ein relativ flacher Anstieg der $TRDR$ mit steigender Detektionstoleranz θ zu beobachten, der teilweise auf die stufenweise Größenanpassung der Trackers zurückzuführen ist. Der Mean-Shift Tracker wird mit drei Größenstufen aufgerufen: größer, gleich groß und kleiner als die letzte bekannte Größe. Anschließend wird von den drei neuen Objektkandidaten mit unterschiedlicher Position und Größe jener ausgewählt, der die beste Übereinstimmung mit dem Objekttemplate hat (größter Bhattacharyya Koeffizient, siehe Algorithmus 1 Schritt 4). Außerdem ergibt sich aufgrund der monochromen Struktur des Quadrates das Problem, dass die Übereinstimmungen der drei unterschiedlich großen Objektkandidaten mit dem Templa-



(a) noise_box

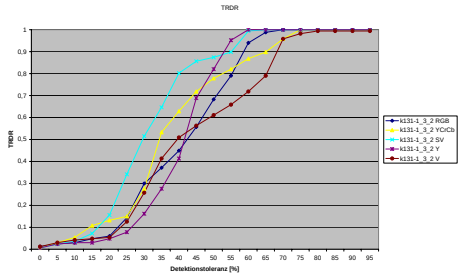


(b) noise_circle

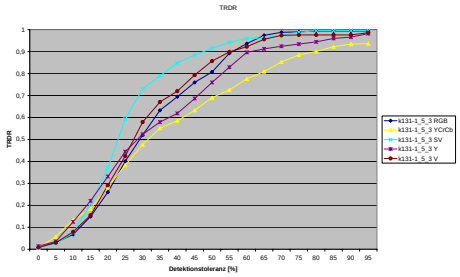


(c) zoom_box

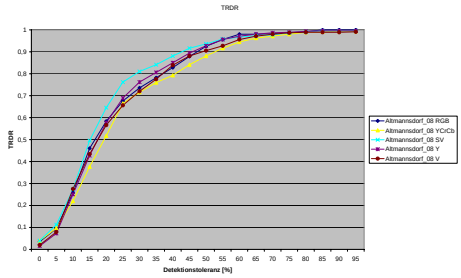
Abbildung 6.13: Tracker Detection Rate ($TRDR$) des Trackers für die synthetischen Testsequenzen



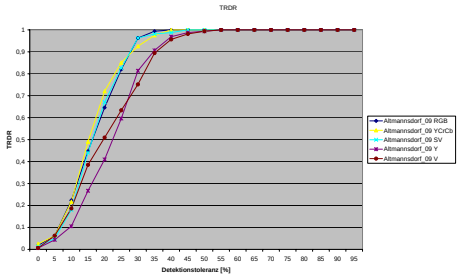
(a) k131-1_3_2



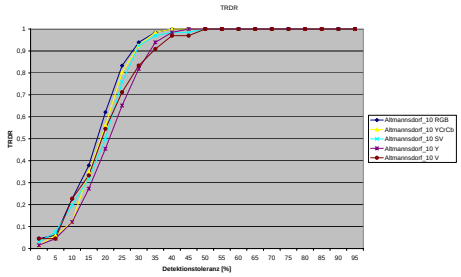
(b) k131-1_5_3



(c) Altmannsdorf_08



(d) Altmannsdorf_09



(e) Altmannsdorf_10

Abbildung 6.14: Tracker Detection Rate ($TRDR$) des Trackers für die realen Testsequenzen

te gleichwertig sind, und somit keine eindeutige Größenzuordnung gegeben werden kann, sobald das Objekt größer als der größte Objektkandidat ist. In dieser Situation bleibt der Tracker „hängen“ und passt die Objektgröße nicht mehr korrekt an, verfolgt aber sehr wohl die richtige Objektposition.

Als Resümee der Evaluation des Trackers über die synthetischen Testsequenzen kann festgestellt werden, dass das RGB und SV Farbmodell konstant gute Ergebnisse für diese Sequenzen liefern. Die beiden einkanaligen Teilräume wie Y und insbesondere V sind für synthetische Sequenzen mit einfärbigen und unstrukturierten Objekten nur bedingt geeignet.
Da in den synthetischen Sequenzen nur ein Objekt vorhanden ist, kann es auch zu keinem False Match kommen, daher ist $FMR \equiv 0$.

Die drei Freilandsequenzen zeigen alle ein ähnliches $TRDR$ Diagramm 6.14(c) bis 6.14(e). Die Kurven der unterschiedlichen Farbmodelle liegen alle dicht neben- bzw. übereinander so dass kein eindeutiger Favorit zu erkennen ist. Bis auf die *Altmannsdorf_ 08* Sequenz ist der Anstieg der $TRDR$ Kurven sehr steil und erreicht bald das Maximum, was bedeutet, dass die Fehler alle getrackten Bounding-Box ähnlich groß sind. Der Hell-Dunkel Übergang (beim Schatten der Brücke) ist eine Erklärung für die flacheren $TRDR$ Kurven der *Altmannsdorf_ 08* Sequenz, da sich an diesem Übergang die Farbverteilung des Objektes ändert und sich somit eine kontinuierliche Änderung zum Objekttemplate ergibt.
Die beiden Sequenzen *Altmannsdorf_ 09* und *Altmannsdorf_ 10* haben eine optimale $nTF = 1$ und $FMR = 0$ (bei $\theta = 50\%$). Bei der *Altmannsdorf_ 08* Testsequenz mussten je nach Farbmodell 0 bis 3 Objekte neu initialisiert werden, was zu einer nTF von 0,00 (RGB) bis zu 1,08 (YCrCb) führt (siehe Tabelle 6.2). Außerdem wurden beim V Farbmodell in vier Frames getrackte Objekte zu falschen Ground Truth Objekten zugeordnet, was zu einer $FMR = 0,01$ führt.

Bei den Tunnelsequenzen liefern die helligkeitsbasierten Farbmodelle Y und V im Vergleich zu den anderen Farbmodellen die besten Resultate (siehe Diagramme 6.14(a) und 6.14(b)). Dies ist wohl darauf zurückzuführen, dass es gerade im Tunnel durch die Scheinwerfer und Deckenleuchten oft zu Farbverschiebungen kommt. Dies betrifft zwar auch die Helligkeit, wirkt sich hier durch die Quantisierung der Verteilung aber offensichtlich nicht so stark aus wie bei Farbmodellen mit Farbinformation.
Außerdem mussten die Objekte bei den Tunnelsequenzen öfter neuinitialisiert werden als bei den Freilandsequenzen, was sich in einer höheren normalized Track Fragmentation nTF niederschlägt.
In der *k131-1_ 3_ 2* Testsequenz befindet sich nur ein Fahrzeug, somit ist die FMR gleich Null. Auch bei der Fahrzeugkolonne in Sequenz *k131-1_ 5_ 3* kommt es nur zu wenigen falschen Objektzuordnungen, was sich in eine FMR von 0,05 für das $YCrCb$ Farbmodell bzw. 0,01 für das V Modell niederschlägt. Bei den übrigen Farbmodellen

dieser Sequenz wurde bei einer Detektionstoleranz von $\theta = 50\%$ keine einzige falsche Objektzuordnung gemessen.

Im Gegensatz zu den synthetischen Sequenzen liefern die einkanaligen Farbmodelle (Y und V) bei den realen Testsequenzen gute Ergebnisse, teilweise sogar bessere als die vollen Farbräume. Wie schon bei den Tunnelsequenzen beschrieben, könnte dies auf Farbveränderungen durch unterschiedliche Beleuchtung bzw. Schatten zurückzuführen sein, da gerade bei dunklen Farben und Farben mit wenig Sättigung kaum relevante Information in den Farbkanälen enthalten ist und in solchen Fällen die zusätzlichen Farbkanäle das Objektmodell eher stören als robuster machen. Außerdem enthalten reale Objekte mehr Struktur als die künstlichen Objekte der Testsequenzen. Dadurch kann auch bei weniger Kanälen ein markanteres Objekttemplate erstellt werden.

Für statistische Aussagen über den Verkehrsfluss ist die genaue Übereinstimmung der getrackten Objekte mit den Ground Truth Objekten nicht so relevant wie sie Stabilität des Tracks selbst. Hier zeigen die Tests mit Sequenzen in denen mehrere Fahrzeuge getrackt wurden (*k131-1_5_3*, *Altmannsdorf_08*, *Altmannsdorf_09*, *Altmannsdorf_10*) gute Resultate. Die normalized Track Fragmentation nTF ist für den SV Farbmodell kleiner gleich 1,14 und über alle Farbmodelle kleiner gleich 1,86. Auch die FMR ist mit 0 für das SV Farbmodell bzw. kleiner gleich 0,05 über alle Farbmodelle sehr klein.

6.5 Evaluation des Gesamtsystems

In diesem Kapitel wird die Evaluation des Gesamtsystems beschrieben. Zuerst wird die Systeminitialisierung mit den verwendeten Parameter erläutert. Anschließend werden die Trackingresultate des Gesamtsystems diskutiert.

6.5.1 Systeminitialisierung

Alle Objekte werden automatisch initialisiert (Fahrzeugdetektion, Kapitel 3.3 und Kapitel 4.2.3) und in den weiteren Frames getrackt. Das Trackingmodul des Gesamtsystems verwendet die Ergebnisse der Vordergrundmaskensegmentierung des Hintergrundmodells, das Objektmodell und den Mean-Shift Tracker um die neue Position der Objekten zu ermitteln (Kapitel 4.2).

Das Hintergrundmodell wird dabei mit folgenden Parametern initialisiert:

- Standardabweichung: $\sigma = (4.0, 3.5, 8.0)$,
- Updatewert der Standardabweichung: $\alpha_\sigma = 0.1$
- Approximated Median Updatewert: $\kappa = 1$

Außerdem wird in der Prototypimplementierung ein Schwellwert für die minimale Fläche zur Objektinitialisierung von 64 Pixel verwendet. Das entspricht dem Doppelten der minimalen Blobgröße (32 Pixel) des Trackingmoduls.

Das Trackingmodul besitzt folgende Parameter und Initialwerte:

- 3D-Objektquader:
  synthetischen Sequenzen: jeweilige Objektgröße
  reale Sequenzen: Breite = 2m, Länge = 5m, Höhe = 1,5m
- Durchschnittsgeschwindigkeit: $v_{init} = 80$ km/h
- Größenfaktor: $scale = 10$ %
  $\Rightarrow$ Mean-Shift Suchfenstergröße: 110%, 100%, 90% der geschätzten Objektgröße
- minimale Kanten bezogen auf den Umfang der Bounding-Box : $\eta = 75\%$
- Objektmaskengewicht: $\omega_m = 0.33$

Es wird angenommen, dass der Umriss von Fahrzeugen deutlich als Kanten im Kantendifferenzbild zu erkennen ist, wohingegen Schatten bzw. durch Scheinwerfer beleuchtete Bereiche keine scharfen Kanten aufweisen. Daher muss die Anzahl der Kantendifferenzpixel im Vergleich zum Umriss der Bounding-Box größer als ein einstellbarer Kantenschwellwert η sein.

Unterschreitet der aus Tracking und Objektmaske resultierende Konfidenzwert einen Schwellwert (33%), wird für die weiteren Berechnungen der Schätzwert der Bounding-Box verwendet. Ansonsten wird die Objekt Bounding-Box als gewichteter Mittelwert mit Hilfe des Objektmaskengewichts w_m berechnet.

$$BB_s = BB_m \cdot w_m + BB_t \cdot (1 - w_m) \tag{6.3}$$

mit:

BB_s: Bounding-Box des Gesamtsystems als Vektor: $(x_{links}, y_{oben}, x_{rechts}, y_{unten})$

BB_m: Bounding-Box des Objektmodells aus Blobs der Vordergrundmaske (siehe Kapitel 4.2)

BB_t: Bounding-Box des Mean-Shift Trackers

w_m: Objektmaskengewicht

Abschließend werden alle Objekte, deren Konfidenzwert einen Minimalwert (20% in der Prototypimplementierung) unterschreitet, aus der Liste der aktiven Objekte gestrichen und nicht mehr weiter getrackt.

6.5.2 Resultate

Das Gesamtsystem verwendet die Szenengeometrie, um perspektivische Größenveränderungen zu berücksichtigen. Bei den synthetischen Testsequenzen wird eine Vogelperspektive auf eine waagrechte Ebene angenommen, das heißt die Objekte unterliegen, unabhängig von der Position im Bild, keiner perspektivischen Verzerrung. Die Größenveränderung des Quadrates in der *zoom_box* Testsequenz ist dabei rein zufällig und ist somit weder durch die Szenengeometrie noch das Objektmodell beschreibbar.

Wie bei der Evaluation des Mean-Shift Trackers wird auch bei der Evaluation des Gesamtsystems die Track Fragmentation TF jedesmal um eins erhöht wenn ein neues Objekt initialisiert wird. Die daraus berechnete normalized Track Fragmentation nTF ist in Tabelle 6.4 aufgelistet.
Die False Match Rate FMR der Prototypimplementierung ist in Tabelle 6.5 zusammengefasst.

Um die Gesamtqualität des Prototypen zu beschreiben wurde die Tracker Detection

	normalized Track Fragmentation				
Sequenzname	*RGB*	*YCrCb*	*SV*	*Y*	*V*
noise_box	1,00	1,00	1,00	11,00	7,00
zoom_box	4,00	4,00	7,00	4,00	12,00
noise_circle	1,00	1,00	1,00	2,00	1,00
k131-1_3_2	1,00	1,00	1,00	1,00	1,00
k131-1_5_3	0,71	1,42	1,14	1,29	1,14
Altmannsdorf_08	1,03	1,03	1,05	1,18	1,33
Altmannsdorf_09	1,00	1,00	1,00	1,00	1,40
Altmannsdorf_10	1,00	1,00	1,00	1,25	1,25

Tabelle 6.4: normalized Track Fragmentation des Gesamtsystems

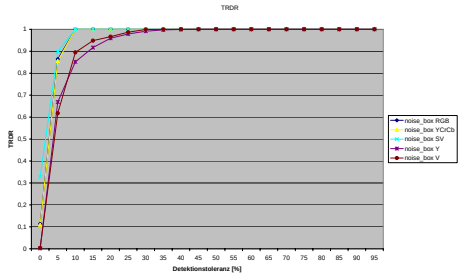
Sequenzname	False Match Rate				
	RGB	$YCrCb$	SV	Y	V
noise_box	0,00	0,00	0,00	0,00	0,00
zoom_box	0,00	0,00	0,00	0,00	0,00
noise_circle	0,00	0,00	0,00	0,00	0,00
k131-1_3_2	0,00	0,00	0,00	0,00	0,00
k131-1_5_3	0,16	0,21	0,07	0,15	0,04
Altmannsdorf_08	0,00	0,00	0,00	0,00	0,00
Altmannsdorf_09	0,00	0,00	0,00	0,00	0,01
Altmannsdorf_10	0,00	0,00	0,00	0,00	0,00

Tabelle 6.5: False Match Rate FMR des Gesamtsystems (Detektionstoleranz $\theta = 50\%$)

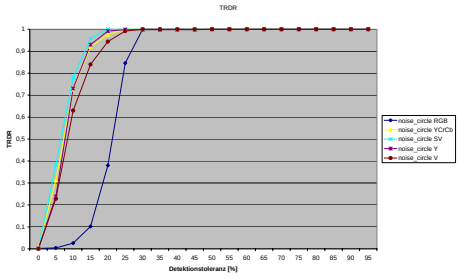
Rate $TRDR$ und das Verhältnis Tracker Detection Rate $TRDR$ zu False Alarm Rate FAR (Receiver Operating Characteristic ROC) für die fünf Farbmodelle (RGB, $YCrCb$, SV, Y und V) berechnet und in den Abbildungen 6.15 und 6.16 bzw. 6.17 und 6.18 dargestellt.

Beim Vergleich der $TRDR$ des Gesamtsystems (Abbildung 6.15) mit der $TRDR$ des Mean-Shift Trackers (Abbildung 6.13) ist bei den synthetischen Sequenzen eine tendenzielle Verbesserung fest zu stellen. Dies ist darauf zurück zu führen, dass das Gesamtsystem sowohl die Ergebnisse des Mean-Shift Tracker als auch einen Abgleich mit den Blobs der Vordergrundmaske zur Ermittlung der Position eines Objekts verwendet. Durch diese Kombination von zwei unterschiedlichen Methoden kann die Gesamtqualität des Trackings verbessert werden. Insbesondere die $TDRD$ des V Farbmodells, die bei der Mean-Shift Tracker Evaluation sehr schlechte Werte lieferte, wurde dadurch deutlich besser.

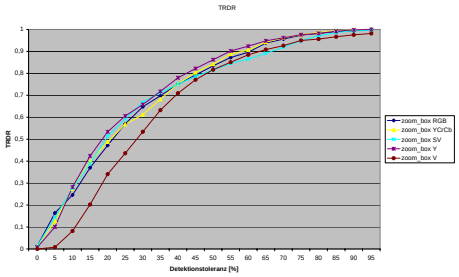
In der Receiver Operating Characteristic ROC (siehe Diagramm Abbildung 6.17) wird für die Sequenzen mit konstanter Objektgröße (*noise_ box* bzw. *noise_ circle*) sogar der optimale Punkt: $TRDR = 1, FAR = 0$ erreicht, wobei eine Häufung der Messpunkte in dem Bereich um $TRDR = 1$ zu beobachten ist. Die allgemeine Form der ROC Kurve entlang der zweiten Hauptdiagonale ergibt sich dadurch, dass ein gefundenes Objekt je nach verwendeter Detektionstoleranz θ entweder als TP oder FP gewertet wird und sich der ROC Messpunkt bei unterschiedlichem θ entlang der Diagonale verschiebt. Eine Abweichung von der Diagonale in das rechte obere Dreieck bedeutet, dass weitere Fehldetektionen aufgetreten sind, zusätzlich zu den FP Detektionen die durch eine geringere Detektionstoleranz θ entstehen. Eine Abweichung der Kurve unter die Hauptdiagonale bedeutet, dass die $TRDR$ bei gleichbleibender FAR sinkt, also einige Ground Truth Objekte gar nicht erkannt werden.



(a) noise_box

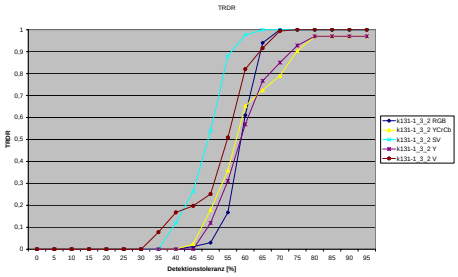


(b) noise_circle

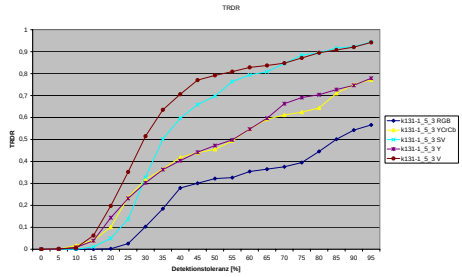


(c) zoom_box

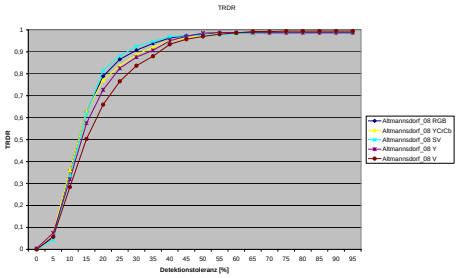
Abbildung 6.15: Tracker Detection Rate ($TRDR$) des Gesamtsystems für die synthetischen Testsequenzen



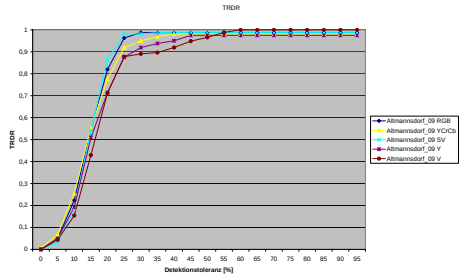
(a) k131-1_3_2



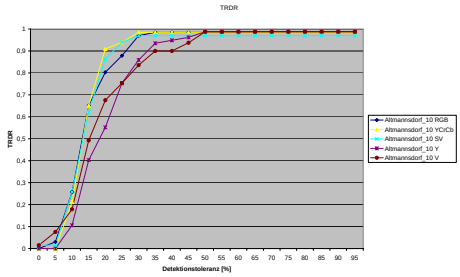
(b) k131-1_5_3



(c) Altmannsdorf_08

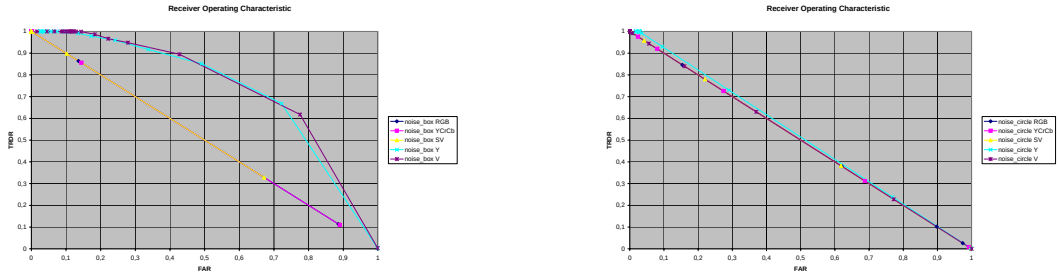


(d) Altmannsdorf_09



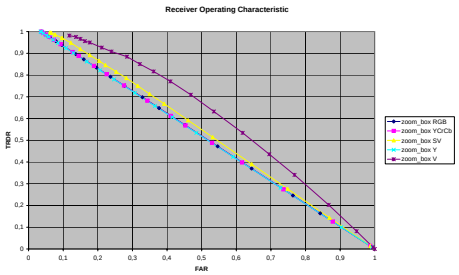
(e) Altmannsdorf_10

Abbildung 6.16: Tracker Detection Rate ($TRDR$) des Gesamtsystems für die realen Testsequenzen



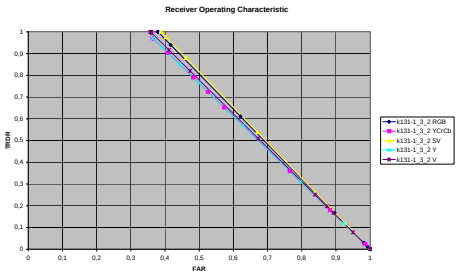
(a) noise_box

(b) noise_circle

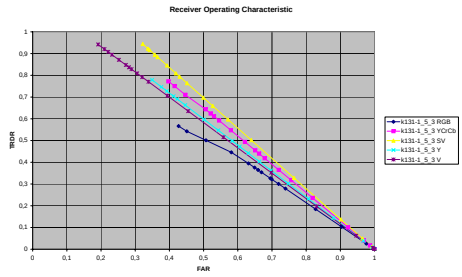


(c) zoom_box

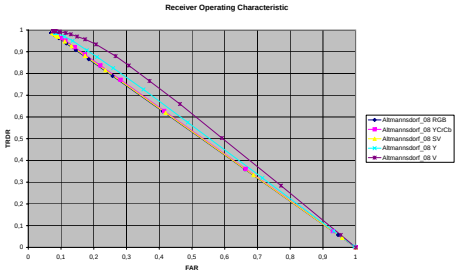
Abbildung 6.17: Receiver Operating Characteristic (*ROC*) des Gesamtsystems für die synthetischen Testsequenzen



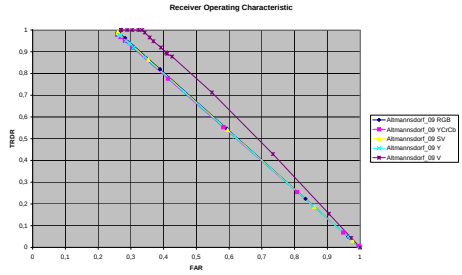
(a) k131-1_3_2



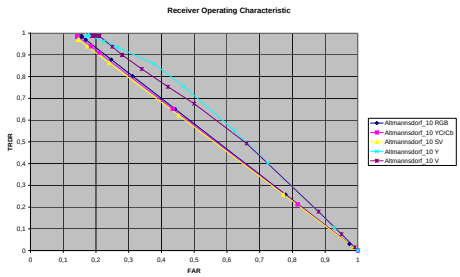
(b) k131-1_5_3



(c) Altmannsdorf_08



(d) Altmannsdorf_09



(e) Altmannsdorf_10

Abbildung 6.18: Receiver Operating Characteristic (*ROC*) des Gesamtsystems für die realen Testsequenzen

Bei den mehrdimensionalen Farbmodellen (RGB, YCr,Cb und SV) wurde das Objekt bei Sequenzen mit konstanter Objektgröße genau ein Mal initialisiert ($nTF = 1$). Bei den beiden eindimensionalen Teilfarbräumen (Y und V) konnte das Objekt aufgrund des indifferenten Objekttemplates nicht kontinuierlich verfolgt werden und musste daher öfter neu initialisiert werden, was zu einer erhöhten normalized Track Fragmentation nTF führt (siehe Tabelle 6.4).

Die Sequenz mit variabler Objektgröße (*zoom_ box*) konnte, im Vergleich zu den Sequenzen mit konstanter Objektgröße, deutlich schlechter getrackt werden. Wie schon im vorigen Kapitel 6.4 beschrieben wurde, kann der Mean-Shift Tracker nur bedingt auf Größenänderungen von Objekten reagieren. Außerdem sind die Größenveränderungen in dieser Sequenz nicht durch die Perspektive verursacht und können somit nicht durch die Szenengeometrie und das Objektmodell vorhergesagt werden. Einzig der Abgleich mit den Blobs der Vordergrundmaske kann hier eine Verbesserung der Trackingqualität bringen, was auch ein Vergleich der $TRDR$ des Gesamtsystems (Abbildung 6.15(c)) mit der des Mean-Shift Trackers (Abbildung 6.13(c)) bestätigt.

Über alle synthetischen Testsequenzen hinweg haben die $YCrCb$ und SV Farbmodell die besten Resultate geliefert. Die beiden eindimensionalen Farbräume (Y und V) haben zwar eine gute $TRDR$ (die Kurven liegen nahe oder über denen der anderen Farbräume) zeigen im ROC Diagramm 6.17 aber eine deutlich höhere FAR.

Wie schon bei den synthetischen Testsequenzen konnte die $TRDR$ des Gesamtsystems bei Freilandsequenzen (*Altmannsdorf_ 08*, *Altmannsdorf_ 09* und *Altmannsdorf_ 10*) gegenüber der $TRDR$ des Mean-Shift Trackers deutlich verbessert werden. So liegt die $TRDR$ aller Freilandsequenzen und aller Farbmodelle für die Detektionstoleranz $\theta = 50\%$ bei 0,97 oder darüber.
Durch das kombinierte Verfahren konnte insbesondere die $TRDR$ der *Altmannsdorf_ 08* verbessert werden, da der Mean-Shift Tracker beim Wechsel der Objekte vom beschatteten in den hellen Bereich durch die Blobs der Vordergrundmaske unterstützt wird.
Die nTF ist für die mehrdimensionalen Farbmodelle (RGB, $YCrCb$ und SV) kleiner gleich 1,05. Bei den eindimensionalen Farbmodellen (Y und V) mussten die Objekte öfter neu initialisiert werden, was zu einer höheren nTF von bis zu 1,40 führt. Außerdem weisen diese zwei Farbmodelle eine höhere FAR in der ROC (Abbildung 6.18(c) bis 6.18(e)) auf.
Bis auf den V Farbraum wurden bei den Freilandsequenzen alle Objekte dem korrekten Ground Truth Objekt zugeordnet. Nur beim V Farbraum der *Altmannsdorf_ 09* Sequenz wurde ein Objekt in zwei Frames einem falschen Ground Truth Objekt zugeordnet, was zu einer $FMR = 0,01$ führt (siehe Tabelle 6.5).

Die Ergebnisse der Tunnelsequenzen konnten im Gesamtsystem nur teilweise verbessert werden. So konnte bei der *k131-1_ 3_ 2* Sequenz und dem SV Farbmodell eine optimale

$TRDR = 1$ bei einer Detektionstoleranz $\theta \geq 65\%$ erreicht werden, zuvor lag sie aber unter denen des Mean-Shift Trackers. Auch die Ergebnisse der *k131-1_ 5_ 3* Sequenz liegen unter denen des Mean-Shift Trackers. Dies ist aber durch die unterschiedliche Objektinitialisierung erklärbar. Bei der Evaluation des Mean-Shift Trackers wurden die Objekte mit den Ground Truth Daten initialisiert, wohingegen die Objektinitialisierung beim Gesamtsystem auf Blobs der Vordergrundmaske beruht. Dadurch werden die Objekttemplates schon mit Fehlern (Reflexionen der Scheinwerfer, Schatten, etc.) gegenüber den Ground Truth Objekten initialisiert. Dies ist insbesondere bei der Testsequenz *k131-1_ 5_ 3* ein Problem, da die Blobs der einzelnen Fahrzeuge in der Kolonne miteinander verschmelzen.
Um auch unter diesen Bedingungen eine gute Objektinitialisierung vornehmen zu können, kann diese nicht alleine auf einem einfachen Hintergrundmodell basieren, sondern es müssen erweiterte Detektions- und Klassifikationsalgorithmen verwendet werden.

Bei der Sequenz *k131-1_ 5_ 3* ist zu beobachten, dass die nTF bei dem RGB Farbmodell kleiner als eins ist (siehe Tabelle 6.4). Dies bedeutet, dass, durch die Verschmelzung von Blobs der Fahrzeugkolonne, nicht alle Fahrzeuge als Einzelfahrzeuge erkannt wurden. Außerdem weist diese Sequenz beim $YCrCb$ Farbmodell die höchste FMR von 0,21 auf. Dies bedeutet, dass im Durchschnitt die Trajektorie jedes fünften Fahrzeuges auf ein anderes Fahrzeug übergesprungen ist.

Bei den realen Testsequenzen konnte das zweidimensionale SV die besten Resultate liefern. Die $TRDR$ beim V Farbmodell war zwar bei den meisten Sequenzen ähnlich bzw. besser als die des SV Modells, allerdings haben die eindimensionalen Farbmodelle auch eine höhere FAR, wie die ROC Diagramme zeigen. Einzig bei der Testsequenz mit Kolonnenverkehr im Tunnel *k131-1_ 5_ 3* wäre das V Farbmodell besser. Die restlichen Farbmodelle lieferten bei den Tunnelsequenzen deutlich schlechtere $TRDR$ Resultate, bei den Freilandsequenzen liegen alle $TRDR$ Kurven knapp nebeneinander, sodass hier kein eindeutiger Favorit zu erkennen ist.

6.6 Laufzeit Evaluation

Neben der Trackingqualität wurde auch die Laufzeit der Implementierung evaluiert, um eine Größenordnung der Laufzeit des Prototypen zu ermitteln, da das endgültige System in Echtzeit laufen soll.
Es wurde zwar bei der Auswahl der Algorithmen Rücksicht auf die besonderen Gegebenheiten von Embedded Systems (beschränkte Ressourcen, emulierte Gleitkommaarithmetik, etc.) genommen, die Implementierung selbst wurde aber als Prototyp und *proof of concept* erstellt und ist in keiner Weise optimiert, weder auf Algorithmenebene noch

plattformspezifisch. Daher sind die Messwerte als maximale, obere Grenzwerte zu sehen und nicht als vergleichbare Laufzeit eines optimierten Tracking Systems.

Wie die Beispieloptimierungen in *TMS320C6000 Programmer's Guide* zeigen [34](Seite 5-57, Tabelle 5-10 und 5-11), ist auf einem DSP alleine durch plattformspezifische Optimierung eine Beschleunigung um den Faktor 25 möglich. Es ist klar, dass dieser Wert nicht als genereller Optimierungsfaktor zu verstehen ist, er ist aber ein Beispiel für die Größenordnung der Laufzeitverbesserungen durch DSP spezifische Optimierung.

Die Laufzeitmessung wurde mit dem *C6416 Device Functional Simulator* des *TI - Code Composer Studio* 3.0 durchgeführt, der einen TMS320C6416 DSP mit 1MHz simuliert. Da der *Device Functional Simulator* selbst sehr viel Rechenzeit benötigt, ist diese Simulation sehr zeitaufwendig und wurde darum auf die erste Sekunde jeder Sequenz beschränkt. Die Ergebnisse der einzelnen Sequenzen sind in Tabelle 6.6 aufgelistet.

Sequenzname	**Objekte in ROI**	**durchschnittliche Laufzeit [ms/Frame]**
Altmannsdorf_08	2	183,8
Altmannsdorf_09	2	247,3
Altmannsdorf_10	0	155,8
k131-1_3_2	0	209,2
k131-1_5_3	0	208,8
noise_box	1	240,5
noise_circle	1	254,5
zoom_box	1	544,2

Tabelle 6.6: Laufzeitmessung der Testsequenzen am DSP Simulator

Die benötigte Rechenzeit steigt mit der Anzahl und der Größe der Objekte. Außerdem hat eine schlechte Positionsschätzung eine Erhöhung der Iterationsschritte des Mean-Shift Trackers, und somit eine Erhöhung der Laufzeit zur Folge. Die deutlichsten Auswirkungen auf die Laufzeit hat aber die absolute Größe des Objektes und laufende Größenanpassung der Objekt Bounding-Box, wie bei der Sequenz *zoom_box* zu beobachten ist.
Die restlichen Laufzeiten liegen in der Größenordnung von 4 Bilder pro Sekunde (fps). Somit könnte man schon mit einer Optimierung um den Faktor drei eine Framerate von 12 fps erreichen, was für ein Tracking zur statistischen Auswertung vollkommen ausreichend ist. Zum Thema Echtzeitfähigkeit des Systems sind noch weitere Punkte zu beachten, insbesondere die schlechte Skalierung bei großen Objekten. Hier ist je nach Anwendungsfall zu entscheiden, ob ein *best effort* Ansatz ausreichend ist, oder ob harte Echtzeit benötigt wird.

6.7 Zusammenfassung

Die Initialisierung der Objekte des Gesamtsystems erfolgt durch die Blobs der Vordergrundmaske und nicht, wie bei der Evaluation des Mean-Shift Trackers, durch Ground Truth Daten. Wie die Resultate zeigen, kann die Kombination aus Mean-Shift Tracker, Berücksichtigung des Hintergrunds und Abgleich mit dem Objektmodell, wie es im Gesamtsystem verwendet wird, trotz teilweise ungenauer Objektinitialisierung eine gute Tracker Detection Rate erzielen. Dies ist insbesondere in jenen Situationen von Vorteil, in denen Objektblobs mehrerer Fahrzeuge miteinander verschmelzen und ein Tracking alleine durch ein Hintergrundmodell nicht mehr möglich ist.

Da der Prototyp auf eine DSP Implementierung ausgelegt ist, wurden bewusst einfache Algorithmen verwendet. Wie das Tracker Detection Rate Diagram (Abbildung 6.15 und 6.16) und das normalized Track Fragmentation Diagram (Tabelle 6.4) zeigen, kann auch mit den beschränkten Ressourcen eines Embedded Systems und entsprechenden Algorithmen ein Trackingsystem entwickelt werden, das Trackingdaten zur weiteren statistischen Auswertung liefert. Insgesamt konnten dabei mit dem zweidimensionales Saturation-Value SV Farbmodell die besten Resultate erziehlt werden.

Die Laufzeit der Prototypimplementierung wurde auf dem *C6416 Device Functional Simulator* gemessen. Da die Prototypimplementierung nicht optimiert ist (weder auf Algorithmusebene noch plattformspezifisch), sind die Ergebnisse dieser Messung nur als maximale Grenzen zu verstehen.
Die Laufzeitsimulation hat allerdings gezeigt, dass ein echtzeitfähiges Fahrzeugverfolgungssystem auf Embedded Systems mit einfachen Bildverarbeitungsalgorithmen möglich ist.

7 Conclusio

In dieser Arbeit wurde ein Bildverarbeitungssystem zur visuellen Fahrzeugverfolgung mittels Embedded Systems beschrieben und als Prototyp implementiert. Dabei sind die wichtigsten Module in je einem Kapitel beschrieben worden. Außerdem wurden diese Module in der Prototypimplementierung zu einem Gesamtsystem integriert und als solches erfolgreich evaluiert.

Trotz der bildverarbeitungstechnisch gesehen einfachen Algorithmen konnte ein System implementiert werden, das auf Videos von Freilandautobahnen eine ausreichende Trackingqualität für statistische Zwecke (Anzahl der Fahrzeuge, Spurwechsel, etc.) erreichte. Bei schwierigen Bildern aus Tunneln (überlappende Fahrzeuge, Reflexionen der Scheinwerfer, Schatten und stark inhomogene Beleuchtung durch Deckenlampen) fällt die Trackingqualität deutlich ab, und es sind andere Methoden (insbesondere der Objektinitialisierung) notwendig, um eine ausreichende Qualität der Trackingresultate gewährleisten zu können.

Die Arbeit gliederte sich in drei Phasen:

1. *Algorithmenauswahl*
   Zuerst wurden in Frage kommende Algorithmen analysiert und evaluiert. Dabei wurde schon bei der Auswahl der Kandidaten auf die Laufzeit und den Ressourcenbedarf der Algorithmen geachtet. So sind einfache, modular aufgebaute und linear skalierende Algorithmen komplexen, datenintensiven bzw. rekursiven Algorithmen vorzuziehen. Außerdem muss berücksichtigt werden, welche Ressourcen der gewählte DSP zur Verfügung stellt und welche weiteren Einschränkungen durch die Hardware bzw. Anwendung entstehen. Dazu gehören unter anderem:

   - unterstützte Zahlentypen / Arithmetiken (Gleitkommaarithmetik in Hardware, durch Softwarebibliothek oder nur Fixkommaarithmetik)
   - vorhandener interner / externer Speicher
   - Laufzeitvorgaben

Um diesen Anforderungen Rechnung zu tragen, wurde für den Prototyp ein Approximated Median Hintergrundmodell anstelle eines Median Modells gewählt (siehe Kapitel 2.2) und der Mean-Shift Algorithmus für einen DSP Einsatz angepasst (siehe Kapitel 4.1).

2. *Implementierung in einer Hochsprache*
   Nachdem die grundlegenden Algorithmen ausgewählt worden sind, wurden sie in C auf einem PC-System implementiert. Um DSP spezifische Funktionen und Eigenheiten am PC nachbilden zu können wurde dazu das DSP-via-PC Framework entwickelt (siehe Kapitel 5.2). Durch dieses Framework war es möglich, die gesamte Entwicklung am PC (ohne DSP Hardware) durchführen zu können, dabei alle Vorteile der Entwicklungssysteme für normale PC Software nutzen zu können und trotzdem quellcodekompatibel zu dem DSP Zielsystem zu bleiben.
   Des weiteren mussten zusätzliche Module zur Transformation zwischen Bildkoordinaten und Weltkoordinaten sowie zur Verwaltung der anfallenden Blob- und Objektdaten erstellt werden.

3. *Evaluierung*
   In der dritten Phase wurden das entwickelte Gesamtsystem und die einzelnen Algorithmen evaluiert. Dabei erfolgte die Evaluierung für unterschiedliche Farbräume bzw. Teilfarbräume. Einerseits sollte damit die Frage beantwortet werden, welchen Einfluss unterschiedliche Farbmodelle auf die Trackingqualität des Gesamtsystems haben. Andererseits hängt die Laufzeit direkt von der Anzahl der Dimensionen der verwendeten Farbräume ab. Wenn mit einem Teilfarbraum ähnliche oder sogar bessere Ergebnisse als mit den vollen Farbmodellen erzielt werden können, so kann dadurch die gesamte Laufzeit verringert werden.
   Die Evaluierung in Kapitel 6 hat gezeigt, dass mit zwei einfachen Methoden (einem Approximated Median Hintergrundmodell und einem Mean-Shift Tracker) Fahrzeuge auf Freilandautobahnen unter Tageslichtbedingungen getrackt werden können. Bei schwierigen Sequenzen wie Kolonnenverkehr in einem Tunnelabschnitt sind diese Methoden nicht ausreichend. Hier müssten bessere Methoden der Objektinitialisierung verwendet werden, um korrekte Objekttemplates für das Tracking erstellen zu können.
   Von den getesteten Farbmodellen lieferte das SV Modell über alle Testsequenzen die besten Ergebnisse. Eine Alternative dazu wäre das eindimensionale V Modell, das allerdings eine höhere False Alarm Rate FAR aufweist.

Es hat sich gezeigt, dass es sehr schwierig ist, einen Algorithmus direkt in eine DSP Implementierung umzusetzen. Durch die besonderen Rahmenbedingungen (z.B. Gleitkommaarithmetik in Fixkommaarithmetik umsetzen, spezielle DSP Bibliotheken, Unterscheidung zwischen internem und externem Speicher, etc.) steigt die Programmierfehlerwahrscheinlichkeit, außerdem ist die Fehlersuche (insbesondere in Bildverarbeitungs-

software) durch fehlende Visualisierungsmöglichkeiten nur bedingt möglich. Daher wurde der Prototyp zuerst auf einem PC erstellt. Allerdings ist dabei schon berücksichtigt worden, dass die eigentliche Zielplattform ein Embedded System ist. Idealerweise ist dabei der selbe Quellcode sowohl am PC als auch auf dem DSP lauffähig, wie es durch das DSP-via-PC Framework in der Prototypimplementierung dieser Arbeit erreicht wurde.

7.1 Ausblick

Um Laufzeitmessungen mit aussagekräftigen Ergebnissen durchführen zu können, muss die Implementierung optimiert werden. Hier kann das DSP-via-PC Framework helfen, einfache Optimierungen auch am PC durchzuführen. Letzten Endes aber müssen solche Optimierungen für jede DSP Familie und jedes Embedded System Design individuell durchgeführt werden, da gerade einer der wesendlichsten Optimierungsbereiche, die Speicheroptimierung, direkt auf den vorhandenen internen und externen Speicher abgestimmt werden muss.

Durch den modularen Aufbau des Systems ist es relativ einfach, ein einzelnes Modul auszutauschen. So könnte zum Beispiel die Objektinitialisierung mittels einer erscheinungsbasierten Objektdetektion, anstelle der hintergrundbasierten Blobmethode des Prototypen, verwendet werden. Dies würde zu einer Verbesserung der Objekttemplates und somit direkt zu besseren Trackingergebnissen führen.

Weiters ist es möglich, das Tracking Modul durch einen anderen Trackingalgorithmus zu ersetzten, um die Ergebnisse mit denen des Mean-Shift Trackers zu vergleichen.
Es ist prinzipiell auch möglich, mehrere unterschiedliche Trackingalgorithmen parallel oder je nach Bedarf zu verwenden. Ein Beispiel dafür ist in der Arbeit von Cucchiara und Piccardi [35] beschrieben, wobei je ein Tracker für Tageslicht- und einer für Nachtaufnahmen verwendet wird. Dazu muss nur die übergeordnete Logik entsprechend erweitert werden, um die zusätzlichen Trackingresultate auszuwerten. Zur Zeit werden hierfür nur die Resultate der Vordergrundmaskensegmentierung und des Mean-Shift Trackers verwendet.

Literaturverzeichnis

[1] Sen-Ching S. Cheung und Chandrika Kamath: *Robust techniques for background subtraction in urban traffic video.* in III; Wong Ping W Delp, Edward J. (Editor): *Security, Steganography, and Watermarking of Multimedia Contents VI*, Vol. 5308, Seiten: 881–892. SPIE, SPIE–The International Society for Optical Engineering, January 2004.

[2] Reinhard Klette, Andreas Koschan und Karsten Schlüns: *Computer Vision - Räumliche Information aus digitalen Bildern*, Kapitel 2 Bildaufnahme, Seiten: 39–86. Vieweg, 1996.

[3] Robert Sablatnig: *3D Vision: Maschinelles Sehen in 3D.* Institut für Rechnergestützte Automation, Arbeitsbereich für Mustererkennung und Bildverarbeitung, 3 Ausgabe, 2006. Lecture Notes: 3D Vision VO18.3129, LU183.130.

[4] Steven W. Smith: *Digital Signal Processing: A Practical Guide for Engineers and Scientists.* Newnes, 2003.

[5] Steve Blonstein: *The TMS320 DSP Algorithm Standard.* techn. Report SPRA581C, Texas Instruments, 2002.

[6] Philipp Blauensteiner und Martin Kampel: *Visual Surveillance of an Airport's Apron - An Overview of the AVITRACK Project.* in W. Burger und J. Scharinger (Editor): *Digital Imaging in Media and Education, Proc. of the 28th Workshop of the Austrian Association for Pattern Recognition (OAGM/AAPR)*, Vol. 179, Seiten: 213–220. Schriftenreihe der OCG, 2004.

[7] University of Reading Team, PRIP Team, INRIA Team Mark Borg, David Thirde, James Ferryman, Florent Fusier, Valery Valentin, Monique Thonnat, François Brémond, Josep Aguilera, Philipp Blauensteiner, Horst Wildenauer und Martin Kampel: *AVITRACK Project: D3.1: Motion Detection.* techn. Report, AVITRACK Consortium, September 2004.

[8] Chris Stauffer und W.E.L. Grimson: *Adaptive background mixture models for real-time tracking.* in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2, Seiten: 246–252. IEEE Computer Society, IEEE Computer Society, June 1999.

[9] N.J.B. McFarlane und C.P. Schofield: *Segmentation and tracking of piglets in images.* Machine Vission and Applications, 8(8):187–193, 1995.

[10] P. Remagnino, A. Baumberg, T. Grove, T. Tan, D. Hogg, K. Baker und A. Worrall: *An integrated traffic and pedestrian model-based vision system.* in *Proceedings of the Eighth British Machine Vision Conference*, Seiten: 380–389, 1997.

[11] Ismail Haritaoglu, Davis Harwood und Larry S. David: *W4: Real-Time Surveillance of People and Their Activities.* IEEE Transaction Pattern Analysis and Machine Intelligence, 22(8):809–830, 2000.

[12] Anil K. Jain: *Fundamentals of Digital Image Processing*, Kapitel 8 - Image Filtering and Restoration, Seiten: 267–341. Prentice-Hall, 1989.

[13] Christof Ridder, Olaf Munkelt und Harald Kirchner: *Adaptive Background Estimation and Foreground Detection using Kalman-Filtering.* in N. Bekiroglu I. Tunay O. Kaynak, M. Özkan (Editor): *Conference on recent Advances in Mechatronics*, Seiten: 193–199. ICRAM'95, August 1995.

[14] Greg Welch und Gary Bishop: *An Introduction to the Kalman Filter.* techn. Report, Department of Computer Science / University of North Carolina, Department of Computer Science University of North Carolina at Chapel Hill Chapel Hill, NC 27599-3175, 2004.

[15] Instituts für Algorithmen und Kognitive Systeme der Universität Karlsruhe: *Image Sequence Server.* http://i21www.ira.uka.de/image_sequences/, Nov 2007.

[16] Julius Schärf: *Funktionen- und Zahlentafeln.* R. Oldenbourg Verlag Wien, 14 Ausgabe, 1990.

[17] Norbert Brändle: *Bildverstehen.* Institut für Rechnergestützte Automation, Arbeitsbereich für Mustererkennung und Bildverarbeitung, 2000. Lecture Notes: Bildverstehen.

[18] Joachim Steinwendner, Werner Schneider und Renate Bartl: *Digital Image Analysis*, Kapitel 12 Image Understanding Methods for Remote Sensing, Seiten: 337–357. Springer, 2001.

[19] Nobuo Hayashi, Hiroshi Nittaya, Masahiro Kohno und Masahiro Kato: *New Approach to and Implementation of an LSI for High-Speed Image Labeling.* in *International Conference on Power Electronics and Motion Control*, Vol. 2, Seiten: 767–771, Nov 1992.

[20] Wikipedia: *Koordinatensystem.* http://de.wikipedia.org/wiki/Koordinatensystem, Nov 2007.

[21] Anders Heyden: *Handbook of Geometric Computing*, Kapitel 10 Three-Dimensional Geometric Computer Vision, Seiten: 305–347. Springer, 2005.

[22] Dieter Koler, Joseph Weber und Jitendra Malik: *Robust Multiple Car Tracking With Occlusion Reasoning.* techn. Report, Institute of Transportation Studies California Partners for Advanced Transit and Highways (PATH) (University of California, Berkeley), Jan 1994.

[23] Weiming Hu, Xuejuan Xiao, Dan Xie, Tieniu Tan und Steve Maybank: *Traffic accident prediction using 3-D model-based vehicle tracking.* IEEE Transactions on Vehicular Technology, 53(3):677–694, May 2004.

[24] Erhard Gramer und Udo Kamps: *Grundlagen der Wahrscheinlichkeitsrechnung und Statistik.* Springer, 2007.

[25] Keinosuke Fukunaga und Larry D. Hostetler: *The estimation of the gradient of a density function, with applications in pattern recognition.* IEEE Transaction on Information Theory, 21(1):32–40, January 1975.

[26] Yizong Cheng: *Mean Shift, Mode Seeking, and Clustering.* IEEE Transaction Pattern Analysis and Machine Intelligence, 17(8):790–799, 1995.

[27] Dorin Comaniciu, Visvanathan Ramesh und PeterMeer: *Real-Time Tracking of Non-Rigid Objects using Mean Shift.* in *IEEE Computer Vision and Pattern Recognition or CVPR*, Vol. 2, Seiten: 142–149. IEEE, June 2000.

[28] Dorin Comaniciu, Peter Meer und Visvanathan Ramesh: *Kernel-Based Object Tracking.* IEEE Transaction Pattern Analysis and Machine Intelligence, 25(5):564–577, May 2003.

[29] Anil K. Jain: *Fundamentals of Digital Image Processing*, Kapitel 9 -Image Analysis and Computer Vision, Seiten: 342–422. Prentice-Hall, 1989.

[30] Texas Instruments: *TMS320 DSP Algorithm Standard API Reference.* techn. Report SPRU360C, Texas Instruments, 2002.

[31] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang und Heather Hinton: *StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks.* in *Proceedings of the 7th USENIX Security Symposium*, Seiten: 63–78, San Antonio, Texas, Jan 1998.

[32] *Video-Based Image Analysis for Tunnel Safety (VITUS).* http://www.smart-systems.at/rd/rd_video_vitus_de.html.

[33] James Black, Tim Ellis und Paul Rosin: *A Novel Method for Video Tracking Performance Evaluation.* in *The Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, Seiten: 125–132, Nice, France, October 2003.

[34] Texas Instruments: *TMS320C6000 Programmer's Guide.* techn. Report SPRU198I, Texas Instruments, 2006.

[35] R. Cucchiara und M. Piccardi: *Vehicle Detection under Day and Night Illumination.* in *International ICSC Symposium on Intelligent Industrial Automation (ISCS-IIA99)*, Seiten: 789–794, Genoa, Italy, 1999. Special Session on Vehicle Traffic and Surveillance.