FRAUNHOFER INSTITUT FÜR NACHRICHTENTECHNIK, HEINRICH-HERTZ-INSTITUT

EINSTEINUFER 37, 10587 BERLIN

TELEFON (030)31002-0, TELEFAX (030) 31002213

ABSCHLUSSBERICHT

ZUM VERBUNDPROJEKT OPTISCHE INFORMATIONSSYSTEME (OIS) ZUR VERKEHRSZENENANALYSE UND VERKEHRSLENKUNG

TEILBERICHT

VIDEOGESTÜTZTE VERKEHRSERFASSUNG (VVE) FÖRDERKENNZEICHEN 03WKJ02C

Berichtzeitraum: 01.09.2001 – 31.12.2003

Projektleiter: Dr.-Ing. A. Smolić

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 03WKJ02C gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

Geschäftsführungsgremium
Bereich Elektronische Bildtechnik

(Dr. R. Schäfer)

Leiter der Abteilung
Bildsignalverarbeitung

(Dr. R. Schäfer)

Inhaltsverzeichnis:

ANLAGE F M. Dröse, Entwicklung von Verfahren zur Segmentierung, Verfolgung und 3D-Rekonstruktion von bewegten Objekten in Verkehrsszenen mit mehreren Ansichten, Diplomarbeit an der TU-Berlin, Fakultät für Elektrotechnik und Informatik, Fachgebiet Nachrichtenübertragung, Januar 2003.

ANLAGE G P. Voigt, Client/Server Architektur für die Videoübertragung und -codierung mit mehreren Kameras, Studienarbeit an der TU-Berlin, Fakultät für Elektrotechnik und Informatik, Fachgebiet Nachrichtenübertragung, September 2002.

TEIL I AUFGABENSTELLUNG UND GRUNDLAGEN

1. Aufgabenstellung und Grundlagen des Teilprojekts VVE

Im Rahmen des Verbundprojekts OIS hat das FHG-HHI gemäß seiner Expertise im Teilprojekt „Videogestützte Verkehrserfassung (VVE)“ die Aufgaben der Videoübertragung und der Zwischenbildinterpolation übernommen. Im Gesamtorganigramm von OIS haben diese Arbeitspakete die Nummern 3200 und 3310 erhalten. Dieser erste Teil des Abschlussberichts beschreibt die Aufgabenstellung dieser Arbeitspakete im Einzelnen, den Ausgangsstatus (inkl. wissenschaftlichem und technischem Stand) bei Projektbeginn, Planung und Ablauf des Teilprojekts, sowie die Zusammenarbeit mit den Verbund- und anderen Partnern.

1.1. Aufgabenstellung des Teilprojekts VVE

1.1.1. Videoübertragung (Arbeitspaket 3200)

Neben der direkten Steuerung von Ampelanlagen mittels Datenerfassung mit optischen Sensoren, stellte die Übertragung der Kamerasignale zu einer Verkehrszentrale einen wichtigen Arbeitspunkt dar. Diese zusätzliche Nutzung der Videodaten stellt ein wesentliches Abgrenzungsmerkmal (Added Value) gegenüber herkömmlichen Steuerungsanlagen (z.B. auf Induktionsschleifen basierend) dar. Die Aufgabe in diesem Arbeitspaket bestand daher darin, ein leistungsfähiges und flexibles System zur Übertragung von Videosignalen einer Vielzahl von Kameras zu einer Verkehrszentrale zu entwickeln. Das Arbeitspaket ist weiter in die folgenden Unterpunkte gegliedert worden:

1.1.1.1. Codierung der Überwachungsvideos (Arbeitspaket 3201)

Um eine schnelle Übertragung der Überwachungsvideos in ausreichender Auflösung und Qualität zu gewährleisten, müssen moderne Codierverfahren direkt an den Kameras eingesetzt werden. Hierzu ist der Codierstandard MPEG-4 vorgesehen. Dazu soll ein auf MPEG-4 basierender Coder auf dem Prozessor an der Kamera implementiert werden. Insbesondere muss eine Optimierung des Encoders auf die zur Verfügung stehende Kanalbandbreite vorgenommen werden. Schliesslich müssen Aspekte des Fehlerschutzes und der Fehlerkorrektur berücksichtigt werden, da die Übertragung ggf. über einen drahtlosen Kanal erfolgen soll, wobei mit verschiedenen Störeinflüssen zu rechnen ist (Bitfehler, Paketfehler, Burstfehler, etc.) (7 PM, 09/01-03/02).

1.1.1.2. Visualisierung der Videosequenzen (Arbeitspaket 3202)

Die erhaltenen Videoströme müssen an der Leitstelle decodiert und angezeigt werden. Dazu muss ein zum MPEG-4-Standard kompatibler Decoder mit entsprechender Display-Applikation auf einem PC implementiert werden. Hierbei sind wiederum Mechanismen zur Fehlerkorrektur von Bedeutung (6 PM, 09/01-02/02).

1.1.1.3. Übertragung der codierten Videosequenzen an der Kreuzung (Arbeitspaket 3203)

Die mit MPEG-4 codierten Videosequenzen und Daten müssen von den einzelnen Kameras zum Kreuzungsrechner übertragen werden. Dies soll drahtlos über WLAN

bzw. über FireWire erfolgen, falls die Videoqualität (in Abhängigkeit von der Kanalkapazität) nicht ausreicht. In jedem Fall ist eine Einbettung der MPEG-4 Bitströme in ein Kanalprotokoll zur Übertragung vorzunehmen. Hierbei ist vorgesehen RTP/IP zu verwenden (4 PM, 04/02-07/02).

1.1.1.4. Übertragung der codierten Videosequenzen zur Leitstelle (Arbeitspaket 3204)

Die mit MPEG-4 codierten Videosequenzen und Daten müssen vom Kreuzungsrechner zur Leitstelle übertragen werden. Dies soll über das Internet bzw. drahtlos über UMTS erfolgen. Die Umsetzung auf einen anderen Übertragungskanal mit einer unterschiedlichen Kapazität macht ggf. eine Transcodierung der MPEG-4 Bitströme erforderlich. Es müssen Methoden zum Multiplexing verschiedene Bitströme von mehreren Kameras sowie weiterer relevanter Überwachungsdaten entwickelt werden. Als Übertragungsprotokoll ist wiederum RTP/IP vorgesehen, d.h. es muss wiederum eine Einbettung des resultierenden Datenstroms vorgenommen werden. (4 PM, 08/02-11/02).

1.1.1.5. Schalten zwischen Videosequenzen (Arbeitspaket 3205)

An der Leitstelle kann durch den Betrachter oder durch ein automatisches Verfahren ein Schalten zwischen den Kameras an der Kreuzung vorgenommen werden. Daher muss die Signalisierung von der Leitstelle zum Kreuzungsrechner die erhaltenen codierten Daten im Kreuzungsrechner umschalten. Wahlweise können auch mehrere Videosequenzen übertragen werden. Dies erfordert den Entwurf eines bidirektionalen Kommunikationsschemas über RTP/IP (3 PM, 12/02-02/03).

1.1.1.6. Gemeinsame Codierung von Video- und Infrarotsignalen (Arbeitspaket 3206)

Um eine verlässliche Steuerung auch bei Nacht und schlechten Sichtverhältnissen zu gewährleisten, ist vorgesehen auch Infrarotkameras zur Datenerfassung und Auswertung einzusetzen. Hierfür sind völlig neue Codierverfahren zu entwickeln, um solche Signale ggf. gemeinsam mit einem entsprechenden Videosignal zu übertragen (6 PM, 07/03-12/03).

1.1.2. Zwischenbildinterpolation (Arbeitspaket 3310)

Falls eine Kreuzung von mehr als einer Kamera überwacht wird, ist es möglich, virtuelle Ansichten an Zwischenpositionen zu erzeugen. Damit kann der Verkehr von jeder beliebigen Position aus betrachtet werden. Dies ermöglicht eine bessere Visualisierung des Geschehens, als es mit den 2D Ansichten allein möglich wäre. So können z.B. Gefahrensituationen und Unfälle genauer in ihrer Entstehung analysiert werden. Die Aufgabe in diesem Arbeitspaket bestand daher darin, ein leistungsfähiges und flexibles System zur Erzeugung von Zwischenansichten aus bestehenden Kameraansichten zu entwickeln. Das Arbeitspaket ist weiter in die folgenden Unterpunkte gegliedert worden:

1.1.2.1. Entwicklung von Disparitätsgestützten Verfahren zur Zwischenbildinterpolation (Arbeitspaket 3311)

Dabei wird davon ausgegangen, dass nur wenige Kamerasignale zur Verfügung stehen. Disparitäten zwischen einzelnen Bildbereichen werden zur Interpolation herangezogen. Bei der Entwicklung der Verfahren muss auch die zur Überwachung einer

Kreuzung optimale Anzahl und Anordnung von Kameras ermittelt werden, da die Zwischenbildinterpolation bestimmte Kamerapositionen unnötig machen könnte (4 PM, 03/02-06/02).

1.1.2.2. Navigation über die Kreuzung (Arbeitspaket 3312)
Dazu soll eine Verknüpfung mit MPEG-4-BIFS vorgenommen werden und die Szene durch einen MPEG-4-Compositor visualisiert werden (3 PM, 07/02-09/02).

1.1.2.3. Entwicklung von geeigneten Codierverfahren (Arbeitspaket 3313)
Dazu sollen die beiden Videosequenzen, aus denen die Zwischenansicht erzeugt wird prädikativ zueinander codiert und übertragen werden. Eine direkte Codierung der Zwischenansicht an der Master-Kamera ist nicht möglich, da mit mehreren 100 ms Umlaufzeit zu rechnen ist und somit eine von der Leitstelle gesteuerte Navigation nicht mit der ausreichenden Geschwindigkeit durchgeführt werden kann (3 PM, 10/02-12/02).

1.1.2.4. Entwicklung von geeigneten Algorithmen zur Verkehrsdatenextraktion aus den oben genannten Datengrundlagen (Arbeitspaket 3314)
Die Bereitstellung der Zwischenbildinterpolation soll mit der Extraktion der Verkehrsdaten aktiv gekoppelt und somit beide Ansätze verbessert werden (4 PM, 03/03-06/03).

1.1.2.5. Umsetzung der Algorithmen in echtzeitfähige Software (Arbeitspaket 3315)
Hierbei sollen Verfahren zur schnellen Disparitätsschätzung und Zwischenbildinterpolation untersucht werden, um in Echtzeit über die Kreuzung navigieren zu können (5 PM, 01/03-05/03).

1.1.2.6. Planung und Durchführungen der Tests im Labor und Feldversuch (Arbeitspaket 3316)
Die Navigation über die Kreuzung soll im Demonstrator gezeigt werden (7 PM, 06/03-12/03).

1.2. Ausgangsstatus für das Teilprojekt VVE

Die Abteilung Bildsignalverarbeitung des FHG-HHI ist ein weltweites Kompetenzzentrum für Videocodierung und –Übertragung. Daher konnte für das Arbeitspaket 3200 auf große Erfahrung aus vorangegangenen Projekten zurückgegriffen werden. Insbesondere standen leistungsfähige Softwareimplementierungen von MPEG-4 Videocodecs zur Verfügung. Auch zur Übertragung über Internet Protokoll konnte auf eigenes Know-how zurückgegriffen werden. Daher konnte das Multiview-Videostreamingsystem in Arbeitspaket 3200 aufbauend auf eigenes Know-how entwickelt werden.

Auch zur Zwischenbildinterpolation besaß das FHG-HHI bereits viel Erfahrung. Dies bezog sich jedoch vor allem auf Verfahren zur disparitätsgestützten Bildsynthese. Auf Know-how und Softwareimplementierungen zu den hierfür notwendigen Algorithmen der Segmentierung, Disparitätsschätzung, Warping und Interaktion sollte zurückgegriffen werden. Jedoch stellte sich nach Projektbeginn heraus, das ein solcher Ansatz aufgrund der besonderen Gegebenheiten im OIS Projekt (siehe Ergebnisse in Teil II) nicht verwendet werden konnte. Daher wurde ein neuer Ansatz entwickelt, der

auf expliziter 3D Rekonstruktion und View-dependent Texture Mapping basiert. Hierbei wurde auf aus der Literatur bekannte Verfahren aufgebaut:

P. Debevec, C. Taylor, and J. Malik, “Modeling and rendering architecture from photographs: A hybrid geometry- and image based approach”, Proceedings of SIGGRAPH 1996, pp. 11-20, 1996.
C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, “Unstructured Lumigraph Rendering”, Proceedings of SIGGRAPH 2001, pp. 425-432, 2001.

1.3. Planung und Ablauf des Teilprojekts VVE

Die in 1.1 aufgeführten Arbeitspunkte sollten während Projektlaufzeit von 28 Monaten bearbeitet werden. Die Priorität lag in den ersten 6 Monaten auf der Videocodierung und –Übertragung, da diese für die Integration im Gesamtprojekt am wichtigsten waren. Die wesentlichen Entwicklungen hierzu sollten nach 18 Monaten abgeschlossen sein. Später sollte das Gewicht auf die Zwischenbildinterpolation verlagert werden.

Die folgende Tabelle gibt die Allokation der Ressourcen an. Die folgenden Diagramme zeigen die Zeitplanung der einzelnen Arbeitspakete über die Projektlaufzeit.

Arbeitspaket	Arbeitsmonate	Arbeitspaket	Arbeitsmonate
3201	7	3311	4
3202	6	3312	3
3203	4	3313	3
3204	4	3314	4
3205	3	3315	5
3206	6	3316	7

Tab. 1 Allokation der Ressourcen

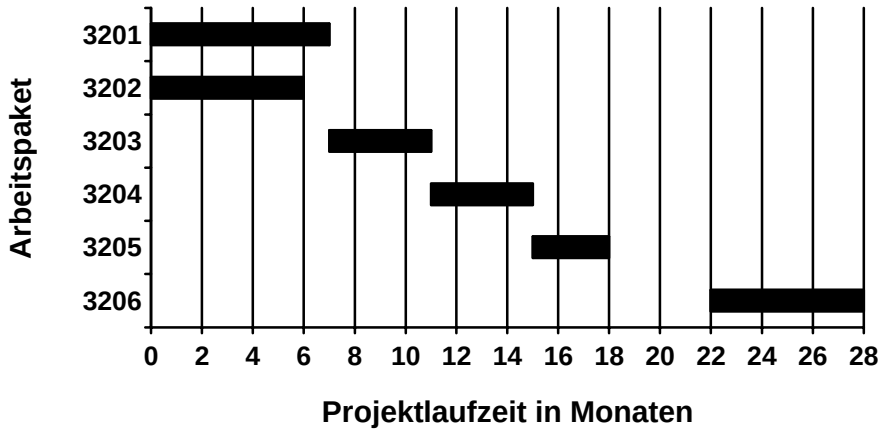


Abb. 1 Zeitplanung für das Arbeitspaket 3200 Videoübertragung

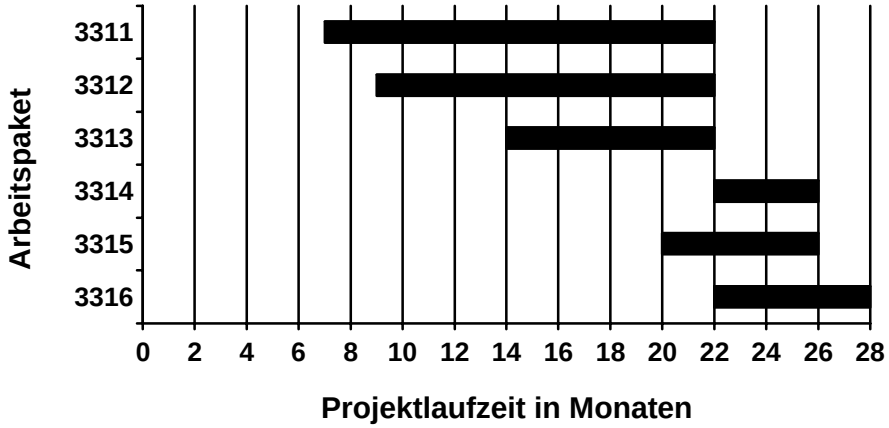


Abb. 2 Zeitplanung für das Arbeitspaket 3310 Zwischenbildinterpolation

1.4. Zusammenarbeit im Gesamtprojekt und mit anderen Partnern

In einem Verbundprojekt wie OIS kommt naturgemäß der Zusammenarbeit zwischen den beteiligten Partnern große Bedeutung zu. Dies betrifft sowohl die Koordination und Integration paralleler und aufbauender Arbeiten als auch den Austausch von erforderlichem Know-how. Daher fand in OIS über die gesamte Laufzeit des Projekts eine enge Abstimmung zwischen den Partnern auf verschiedenen Ebenen statt. Auf oberster Ebene waren dies Treffen des Gesamtprojekts und des Lenkungsausschusses. Weiterhin fanden regelmäßig Sitzungen zu einzelnen Arbeitspaketen statt, wobei für das FHG-HHI das Arbeitspaket 3000-Verkehrsrechner von Bedeutung war. Darüber hinaus fanden bei Bedarf Integrations- und Planungstreffen zwischen betroffenen Partnern statt, sowie rege Kommunikation per Email und Telefon. All dies ermöglichte in der Gesamtheit den erfolgreichen Abschluss des Gesamtprojekts und seiner einzelnen Teile.

Weiterhin kommt dem professionellen Softwareengineering bei einem solch großen Verbundprojekt mit verteilter und zeitkritischer Entwicklung besondere Bedeutung zu. Das FHG-HHI besitzt auf diesem Gebiet große Erfahrung. Dieses Know-how wurde zu Beginn des Projekts an die Partner weitergegeben (Regeln für Softwaredesign, C++ Implementierung, CVS-Versions-Management, etc.). Auf dieser Basis wurde die Softwareentwicklung im Gesamtprojekt weitgehend durchgeführt. Dies stellte die reibungslose Integration z.B. im Arbeitspaket 3000-Verkehrsrechner sicher.

Neben der Zusammenarbeit mit den Projektpartnern bildete auch die Kooperation mit anderen Partnern in der weltweiten wissenschaftlichen Community eine Grundlage für den Erfolg des Projekts. Besondere Bedeutung dabei hatte die Mitwirkung in der 3DAV-Gruppe von MPEG, die in Abschnitt 2.3 näher erläutert ist.

TEIL II ERGEBNISSE

2. Wissenschaftliche und technische Ergebnisse des Teilprojekts VVE

Im Folgenden sind die Arbeiten und Ergebnisse des Teilprojekts VVE im Detail dargestellt. Auf eine Beschreibung von Einzelheiten der entwickelten Konzepte und Algorithmen wurde zur besseren Übersichtlichkeit an dieser Stelle weitgehend verzichtet. Entsprechende Zusammenfassungen, Berichte und Veröffentlichungen sind in den Anhängen zusammengefasst. Auf die einzelnen Anhänge wird im Text verwiesen.

2.1. Videostreaming AP 3200

Ein wesentlicher Vorteil der optischen Verkehrserfassung z.B. gegenüber Induktionssystemen ist die Möglichkeit die aufgenommenen Videosignale auch für andere Zwecke als die im Zentrum stehende Verkehrsdatenextraktion zu nutzen. Die Kamerabilder sind dem Menschen unmittelbar zugänglich und können auch von Rechnern weiter verarbeitet werden. Die Voraussetzung hierfür ist die Übertragung der Videosignale von den Kameras zu anderen Orten, wo sie weiter verwendet werden, wie z.B. einer Verkehrsleitzentrale. Zu diesem Zweck wurde im Projekt ein flexibles und leistungsfähiges System zum Videostreaming entwickelt, das in Abb. 3 veranschaulicht ist. Die Kamerasignale (inkl. Infrarot) werden effizient komprimiert (MPEG-4) und über ein Netzwerk (Internet Protokoll, IP) zu einer Verkehrsleitzentrale übertragen. Hier findet eine zentrale Überwachung statt. Die Signale können live visualisiert aber auch aufgezeichnet werden. Dies ermöglicht die Erkennung von Problem- und Gefahrensituationen und damit die Einleitung geeigneter Gegenmaßnahmen. Auch die spätere Auswertung z.B. von Unfällen, Analyse von Fahrverhalten und ggf. Erkennung von Verkehrsdelikten wird bei Aufzeichnung der Bilder möglich. Weiterhin kann auch der Zustand der Strassen, Verkehrseinrichtungen, etc. überprüft werden (Fernwartung). Wie in Abb. 3 dargestellt ist, können die Videodaten über IP auch anderen ggf. mobilen Nutzern zur Verfügung gestellt werden, z.B. der Polizei, Feuerwehr, aber auch geschäftlichen und privaten Nutzern, die sie z.B. zur optimalen Routenplanung verwenden könnten.

Ausführliche Beschreibungen des Videostreamingsystems sind **Anlage E** und **Anlage G** zu entnehmen.

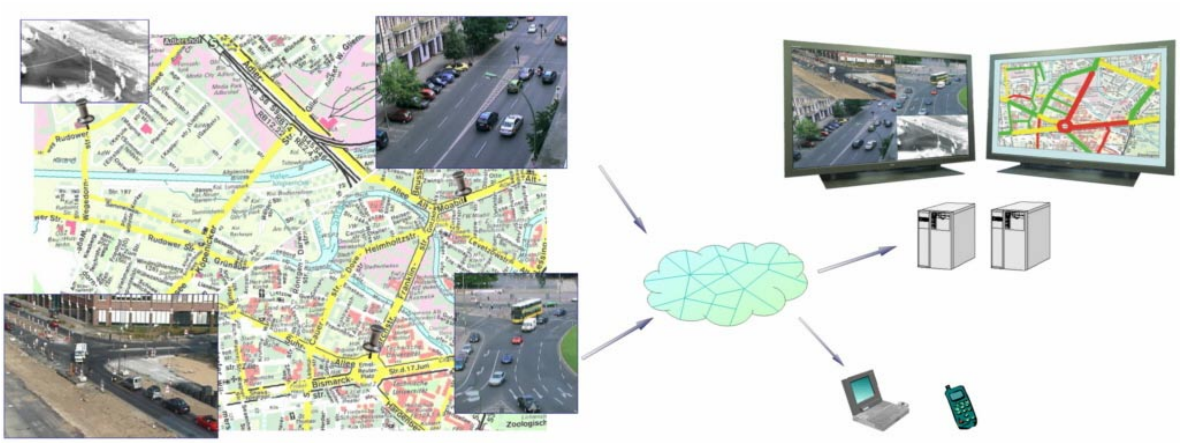


Abb. 3 Streaming der Videosignale von den Kameras zu einer Verkehrsleitzentrale und ggf. weiteren Nutzern

2.1.1. Softwarekonzept des Server/Multiclientsystems

Abb. 4 veranschaulicht das Softwarekonzept des entwickelten Server/Multiclientsystems zur Übertragung mehrerer Videosignale zu einer Zentrale. Jede Kamera (1…N) ist mit einem Videoserver versehen. Die vom Framegrabber eingehenden Bilder werden konvertiert, komprimiert und in IP-Pakete (RTSP/RTP) verpackt. Der Multiclient ist in der Zentrale implementiert. Diese Software übernimmt die Steuerung des Systems über Kommandos, Empfang, Decodierung, Darstellung und ggf. Aufzeichnung der Daten, sowie die Benutzerinteraktion.

Das System wurde nach modernsten Prinzipien und Methoden des Softwareengineering entwickelt. Alle Komponenten wurden in UML modelliert, die Verwaltung wurde unter Verwendung von CVS durchgeführt. Die Implementierung erfolgte objektorientiert in der Programmiersprache C++. Multithreading wurde zur optimalen Parallelisierung und Modularisierung der Rechenprozesse eingesetzt. Eine ausführliche Dokumentation der entwickelten Software und insbesondere der Schnittstellen wurde erstellt. Dies stellte sicher, dass die Integration der Softwarekomponenten in das OIS-Gesamtsystem in enger Abstimmung mit den beteiligten Partnern völlig problemlos durchgeführt werden konnte. Außerdem sind alle Module wiederverwertbar.

Zentrale Anforderungen an den Entwurf waren eine hohe Flexibilität, individuelle Konfigurierbarkeit, Komplexitätsskalierbarkeit, sowie die Fernsteuerbarkeit der Server. Die Server sind jeweils im Kameraknotenrechner implementiert. Je nach Rechenleistung, die hier verfügbar ist (d.h. Kosten) kann der Videoserver entsprechend aufwendig (d.h. Qualität) eingestellt werden. Im einfachsten Fall kann er z.B. deaktiviert werden. Ansonsten steht eine Reihe von Qualitätsstufen, die verschiedenen Rechenaufwand erfordern, zur Verfügung. Dies ermöglicht die Implementierung der OIS-Kameraknotenrechner in verschiedenen Qualitäts-/Kostenabstufungen. So könnten z.B. an besonders wichtigen Kreuzungen, die auch visuell überwacht/aufgezeichnet werden sollen, aufwendigere Kameraknotenrechner installiert werden, als an anderen Kreuzungen, bei denen nur Verkehrsdaten extrahiert werden sollen.

Weiterhin ist die entwickelte Serversoftware von einer Zentrale aus fernsteuerbar und fernwartbar. Man kann sie ein- und ausschalten sowie Qualität/Rechenaufwand zur Laufzeit umschalten. Dies ermöglicht zum einen die Anpassung an schwankende Übertragungsbedingungen (Netzauslastung) als auch die benutzergesteuerte Übertragung (z.B. nur bei besonderem Interesse). Diese Steuerung des Systems wird von der in der Zentrale implementierten Multiclientsoftware über ein eigens entwickeltes Kommandoprotokoll durchgeführt.

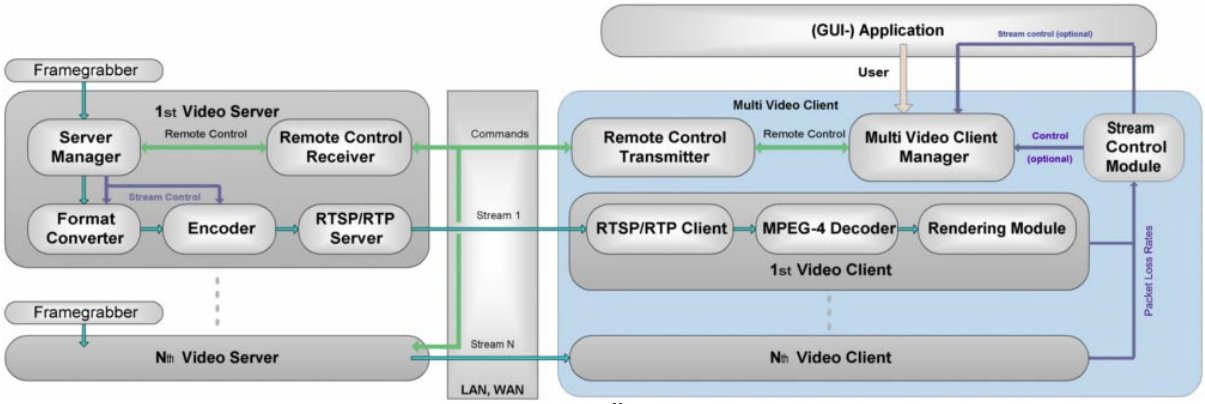


Abb. 4 Server/Multiclientsystem zur Übertragung mehrerer Videosignale zu einer Zentrale

2.1.2. Videoserver

Abb. 5 veranschaulicht den implementierten Videoserver, wobei die blau hinterlegten Module zur eigentlichen Videoserverlibrary gehören. Diese Library ist als DLL implementiert und in den OIS-Demonstrator sowie den HHI-Stand-alone-Demonstrator integriert. Daten werden mit dem Multiclient über ein IP-Netzwerk (grün hinterlegt) ausgetauscht. Dabei werden die Videodaten über UDP ausgetauscht, die wichtigen Kommando- und Steuerdaten werden über gesicherte TCP-Sockets ausgetauscht.

Die Server-DLL wird von Außen initialisiert, gestartet und gestoppt, z.B. vom Hauptmodul des OIS-Kameraknotenrechners oder von einer speziellen MFC-Anwendung im HHI-Demonstrator. Die Schnittstelle zur Kamera bildet ein Framegrabbermodul, von dem die zu verarbeitenden Bilder übernommen werden.

Der Server selbst besteht aus einer Reihe von Modulen (Threads), die miteinander kommunizieren. Eine Interface-Klasse bildet die Schnittstelle zum Framegrabber und zur erzeugenden Instanz. Das Zentralmodul verwaltet und steuert den Server. Ein Kommandomodul übernimmt die Kommunikation mit dem Multiclient über TCP. Der Konverter formt das Bildformat entsprechend dem eingestellten Qualitätsprofil um, s.u. (ggf. Unterabtastung, Farbformatkonversion). Der MPEG-4 Encoder komprimiert die Bilddaten entsprechen dem eingestellten Qualitätsprofil und gibt den Bitstrom an den RTSP/RTP-Server weiter. Hier werden die Daten in IP-Pakete unterteilt und über eine UDP-Verbindung zum Multiclient übertragen. Die Signalisierung erfolgt hierbei über eine gesicherte TCP-Verbindung.

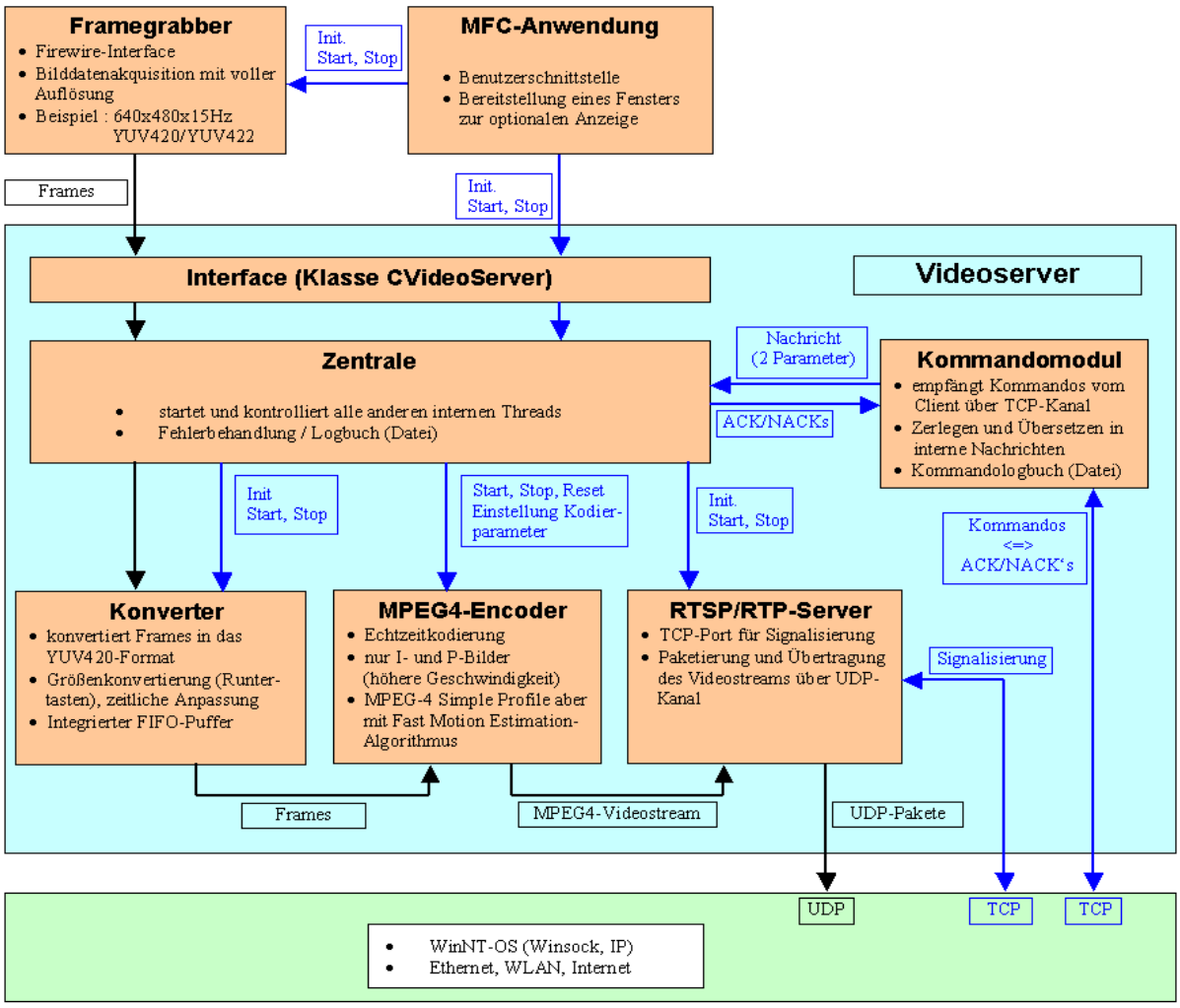


Abb. 5 Übersicht Videoserver (blau hinterlegt)

Wie bereits beschrieben und motiviert, kann der Videoserver in einer Reihe verschiedener Qualitätsstufen betrieben werden. Tabelle 2 spezifiziert die insgesamt 17 verschiedenen Profile, aus denen ausgewählt werden kann. Die Nummer des Profils wird bei der Initiierung von der aufrufenden Instanz übergeben. Sie kann aber auch wie beschrieben während der Laufzeit über ein Kommando vom Multiclient aus verändert werden. Die Profile stellen verschiedene Kombinationen von Bilddimensionen, Wiederholraten, Bitraten und I-Frame-Perioden dar. Der Rechenaufwand kann vor allem durch die Bilddimensionen und die Wiederholrate beeinflusst werden, wobei kleinere Bilder und geringere Wiederholraten zu einer Reduzierung des Rechenaufwands führen, aber auch die visuelle Qualität beschränken. Die Einstellung der Bitrate macht eine Anpassung an die zur Verfügung stehende Kanalkapazität möglich, die wie gesagt dynamisch angepasst werden kann.

Profil	Bildbreite	Bildhöhe	Framerate[Hz]	Bitrate [kbits/s]	IFrame-Periode
0	Benutzerdefiniert				
1	640	480	15	1200	7
2	640	480	5	600	3
3	640	480	15	1000	15
4	640	480	5	500	5
5	320	240	15	500	7
6	320	240	5	250	3
7	320	240	15	400	15
8	320	240	5	200	5
9	352	288	25	800	12
10	352	288	10	400	5
11	176	144	25	200	12
12	176	144	10	100	5
13	704	576	25	2000	12
14	720	576	25	2000	12
15	512	384	5	500	5
16	1024	768	5	2000	5

Tab. 2 Qualitätsprofile des Videoservers

2.1.3. Multiclient

Abb. 6 veranschaulicht die Implementierung des Multiclients, der wie gesagt in einer Zentrale eingesetzt wird. Es beinhaltet ein zentrales Manager-Modul, das die Steuerung aller Prozesse vornimmt. Zu jedem Videoserver im Netzwerk existiert ein eigener Videoclient, d.h. im Multiclient sind N einzelne Clients enthalten, die jeweils mit einem Server verbunden sind und vom zentralen Manager gesteuert und verwaltet werden. Jeder einzelne Videoclient enthält einen RTSP/RTP-Client, der die IP-Verbindung zum Server herstellt, die Pakete im Empfang nimmt und daraus einen MPEG-4 Bitstrom extrahiert. Der Bitstrom wird von jeweils einem MPEG-4 Decoder wieder in ein Videosignal umgewandelt. Das Rendering-Modul ist für die Darstellung bzw. Speicherung der Daten zuständig. Zur Überwachung der Netzwerkbedingungen ist ein eigenes Modul vorhanden. Es wertet die auftretenden Paketfehlerraten aller Clients aus, schließt daraus automatisch auf die Auslastung des IP-Netzwerks und erkennt ggf. Überlastungen. Bei Überlastung werden automatisch über den zentralen Manager Gegenmaßnahmen eingeleitet. Dazu werden die Profile der einzelnen Server über die Command-Verbindung umgeschaltet, wobei Profile mit weniger Bitrate vorgegeben werden, sodass die Netzauslastung sinkt.

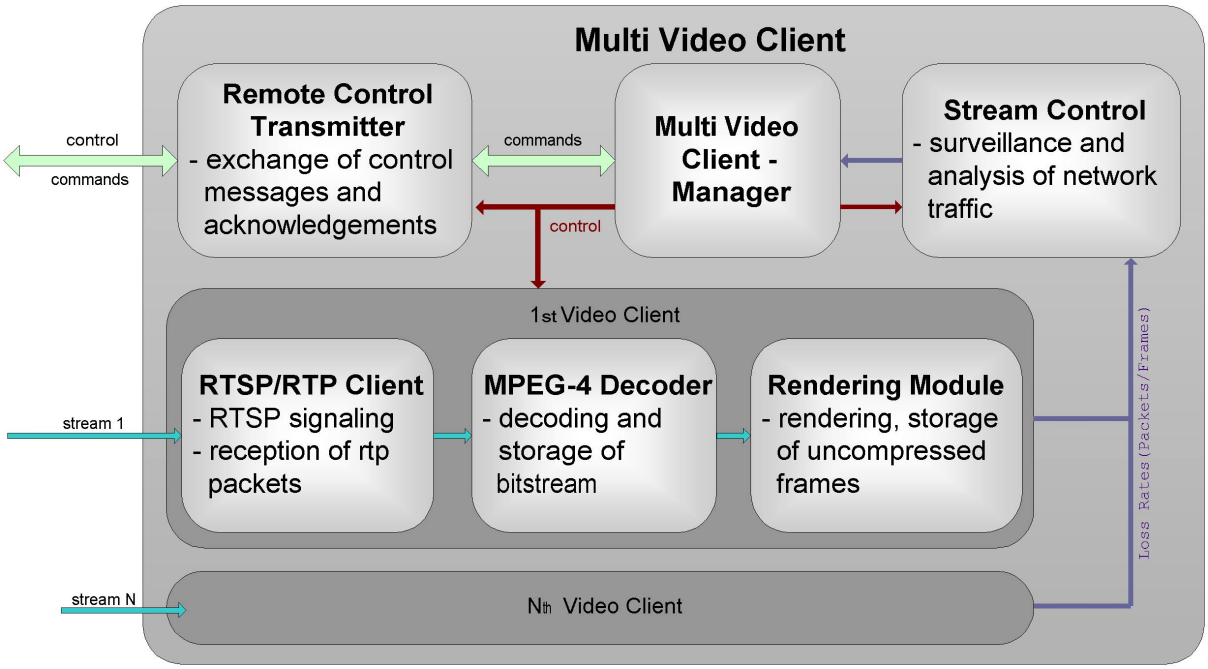


Abb. 6 Übersicht Multiclient

Der Multiclient wird über ein spezielles Interface gestartet und gesteuert, das z.B. von einem Hauptprogramm angesprochen werden kann (Abb. 7). Zunächst wird der Manager gestartet, der dann alle weiteren Module startet. TCP-Kontrollverbindungen zu den einzelnen Servern werden vom Remote-Control-Modul aufgebaut, die UDP-Datenverbindungen werden von den einzelnen Clients (d.h. ihren RTP/RTSP-Client-Modulen) aufgebaut.

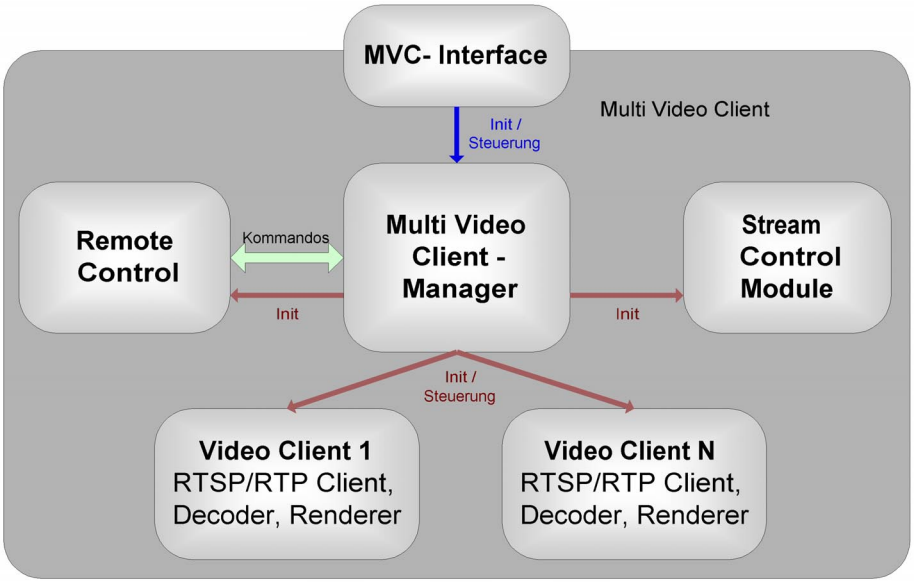


Abb. 7 Initiierung des Multiclients über Interface

Das Interface des Multiclients erlaubt es auch dem Benutzer eine Steuerung vorzunehmen (Übertragung einzelner Kameras ein-/ausschalten, Profile ändern) und Informationen aus dem System abzurufen (Abb. 8).

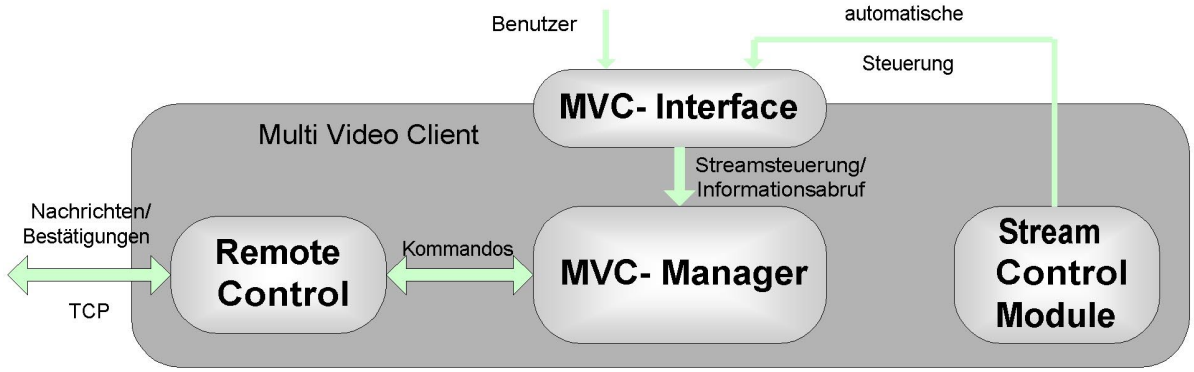


Abb. 8 Manuelle und automatische Fernsteuerung der Videoserver

Eine Benutzerinteraktion hat immer auch eine Auswirkung auf die automatische Steuerung die entsprechend angepasst werden muss. Die entsprechenden Reaktionen der automatischen Steuerung bei bestimmten Benutzerinteraktionen sind in Tabelle 3 beschrieben.

Benutzeraktion	**Reaktion des MultiVideoClients / SCM**
- Starten eines neuen Streams	Berechnen der neuen Gesamtbitrate Bei Überschreitung der maximalen Gesamtbitrate wird überprüft, ob durch Bitratenreduzierung der anderen Streams sowie des neuen Streams die Grenze eingehalten werden kann. Ist dies der Fall, so werden alle möglichen Bitraten gesenkt und der neue Stream gestartet.
- Stoppen eines Stream	Der Stream wird gestoppt. Die freigewordene Bitrate wird nicht auf die anderen Streams verteilt.
- Wechsel eines Videoprofiles in ein anderes mit höherer Bitrate - direkte Erhöhung der Bitrate bei einem laufenden Stream	Überprüfung und Änderung der Bitraten wie beim Starten eines neuen Streams, allerdings mit dem Unterschied, das nur die Bitraten der anderen Streams geändert werden dürfen
- Wechsel eines Videoprofiles in ein anderes mit geringerer Bitrate - direkte Senkung der Bitrate bei einem laufenden Stream	Das Videoprofil bzw. die Bitrate wird geändert. Die freigewordene Bitrate wird auf die anderen Streams der Priorität 1 und 2 verteilt, wobei deren Profilbitrate nicht überschritten wird. Die Verteilung kann durch den Benutzer unterbunden werden.

Tab. 3 Reaktionen des MVC auf Benutzeraktionen bei aktiviertem Kontrollmodul

2.1.4. Paketwiederholung bei gestörter Übertragung

Trotz der entwickelten Bitratenanpassung kann es bei gestörter Übertragung und starker Auslastung des Netzwerks temporär zu Paketverlusten kommen (die UDP-Verbindungen sind ungesichert). Um dem entgegenzuwirken, wurde ein Mechanismus zur Paketwiederholung entwickelt, dessen Arbeitsweise in Abb. 9 dargestellt ist.

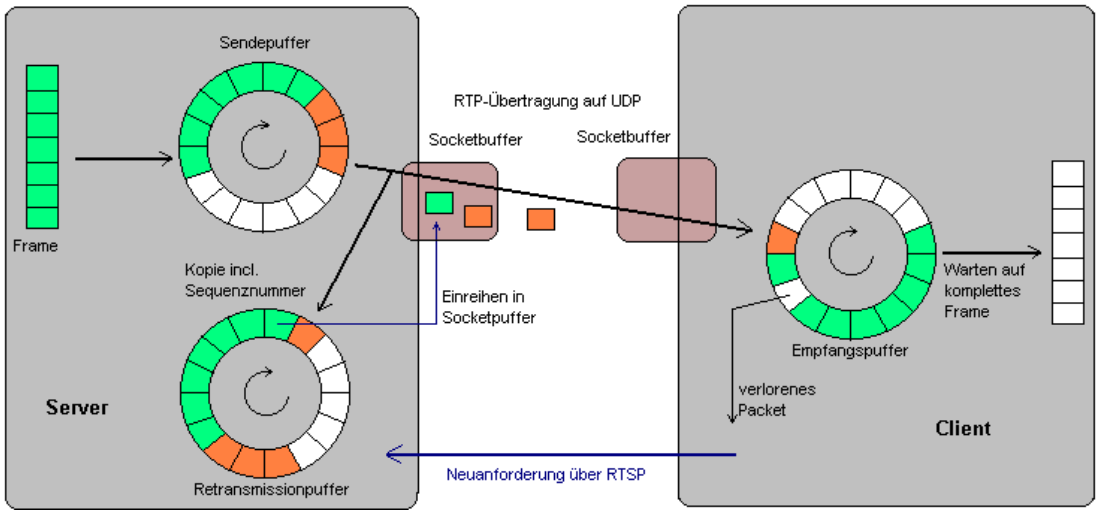


Abb. 9 Paketwiederholung bei gestörter Übertragung

2.1.5. Rendering-Modul

Das Rendering-Modul kann in verschiedenen Modi betrieben werden, die für verschiedene Benutzerinterfaces geeignet sind (Abb. 10). Im Standardmodus (single) wird für jeden Videokanal ein eigenes Fenster erzeugt (intern). Hierbei wird auch Zusatzinformation angezeigt (Servername, Uhrzeit, ggf. Aufnahme). Im Multi-Modus werden alle Signale in ein gemeinsames, extern erzeugtes Fenster gerendert (muss von der initiierenden Instanz über das Interface mitgeteilt werden). Auch im freien Modus wird das Fenster von außen erzeugt. Hierbei kann die initiierende Instanz alle Einstellungen (Position und Größe der einzelnen Bilder) vorgeben, muss aber auch die Anpassung bei Änderung des Gesamtfensters vornehmen.

Abb. 10 Renderer-Modi (Single-, Multi- und Free Mode)

2.1.6. Beispielanwendung TraVis

Zum Test des Systems wurde eine eigene Anwendung entwickelt Traffic Visualization and Surveillance (TraVis). Das Hauptmenü von TraVis ist in Abb. 11 gezeigt. Es erlaubt die Steuerung aller Funktionalitäten des entwickelten Verkehrsüberwachungssystems. Die Systemkonfiguration ist jeweils in einer Textdatei in einem speziellen Format beschrieben, die über das Menü „Database“ verwaltet werden kann. Diese Konfigurationsdatei enthält vor allem die IP-Adressen der zu verwaltenden Videoserver, deren Namen, etc.

Abb. 11 Hauptmenü der Anwendung TraVis

Über das „Connections“ Menü werden die Verbindungen zu den Servern verwaltet. Abb. 12 zeigt ein Beispiel für eine Kamera. Man erhält Informationen zur Verbindung (Name, IP-Adresse, Ports) und kann diese ggf. auch ändern. Man kann im „Server Control“ Bereich die Server starten, stoppen, ausschalten usw. „Video Control“ ermöglicht die Einstellung der Videoprofile. „Transmission“ erlaubt Einstellungen zur Paketwiederholung.

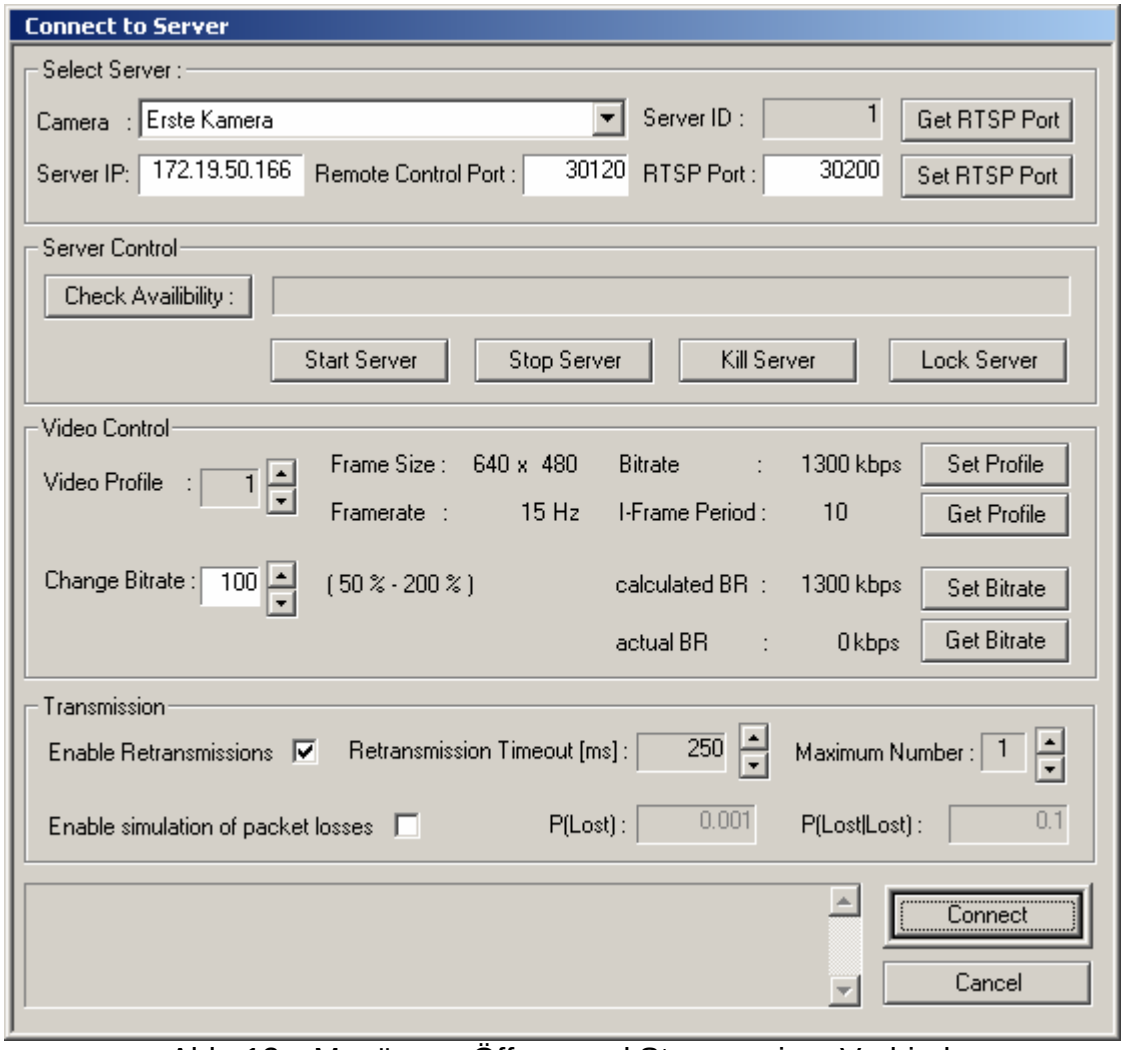


Abb. 12 Menü zum Öffnen und Steuern einer Verbindung

Abb. 13 zeigt weitere Menüs, die sich bei bestimmten Eingaben öffnen. Sie ermöglichen z.B. die Aufnahme und Speicherung von Bildern, Zoom, Abruf von Statistiken über die Verbindung, Start/Stop der Übertragung und die Einstellung von Profilen und Bitraten.

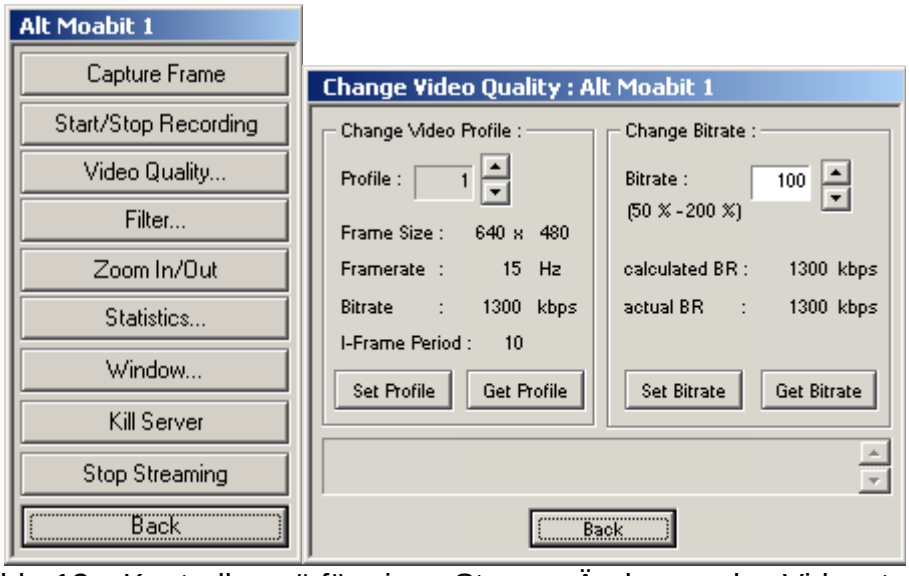


Abb. 13 Kontrollmenü für einen Stream, Änderung des Videostreams

2.1.7. Integration, Test und Schlussfolgerungen

Die beschriebenen Module des Videoservers und des Multiclients sind problemlos in den OIS-Demonstrator integriert und bezüglich Funktionalität und Robustheit erfolgreich getestet worden. Weiterhin wurde am FHG-HHI ein Stand-alone-Demonstrator realisiert, der aus 4 Videoservern (Standard PCs mit der entwickelten Software) mit Kameras und einem Multiclient (Standard PC) besteht. Dieses System wurde ausgiebig getestet (z.B. mit LAN und WLAN Verbindungen) und optimiert. Dabei konnte die Funktionalität und Robustheit verifiziert werden.

Der HHI-Demonstrator wurde erfolgreich auf zahlreichen internen Vorführungen und auch einer breiten Öffentlichkeit auf verschiedenen Events (Lange Nacht der Wissenschaften, International Funkausstellung, CEBIT 2004) präsentiert.

Bezüglich des Arbeitspakets 3206 (Gemeinsame Codierung von Video- und Infrarotsignalen, siehe 1.1.1.6) sind verschiedene Tests anhand von der DLR bereitgestellter Infrarotsignale durchgeführt worden. Dabei hat sich herausgestellt, dass die Codierung von Infrarotsignalen problemlos mit den entwickelten Verfahren zur Videocodierung möglich ist. Aufgrund der besonderen Quellcharakteristik der Infrarotsignale ist sogar eine noch stärkere Kompression als bei Video bei guter Qualität möglich. Daher war keine Entwicklung spezieller Verfahren notwendig. Die Kompression von Infrarotsignalen mittels MPEG-4 wurde in den OIS-Demonstrator integriert und erfolgreich z.B. auf der Abschlusspräsentation des Projekts demonstriert.

Damit konnten alle in 1.1.1 beschriebenen Arbeitspunkte im Paket 3200 erfolgreich abgeschlossen werden.

2.1.8. Verwertung

Das entwickelte System und die enthaltenen Module sind nicht nur für das OIS Szenario der Verkehrüberwachung geeignet, sondern können auch für eine Vielzahl an-

derer Anwendungen der Videoübertragung eingesetzt werden, insbesondere bei Verwendung in Systemen mit mehreren Kameras. In Frage kommen z.B. allgemeine Überwachungssysteme (insb. große Gelände und Gebäude, Flughäfen, Bahnhöfe, etc.), Fernwartung oder Videokonferenzen mit mehreren Teilnehmern.

2.2. Zwischenbildinterpolation AP 3310

Falls eine Kreuzung von mehr als einer Kamera überwacht wird, ist es möglich, ein interaktives, dynamisches 3D Modell der Szene zu erzeugen. Ein solches 3D Modell ermöglicht es, die Szene von jedem beliebigen Standpunkt und Blickwinkel aus zu betrachten, wie es z.B. von Computergrafikmodellen (Spiele, virtuelle Welten) bekannt ist. Dies ermöglicht eine bessere Visualisierung des Geschehens, als es mit den 2D Ansichten allein möglich wäre. So können z.B. Gefahrensituationen und Unfälle sehr viel genauer in ihrer Entstehung analysiert werden. Die 3D Positionen und Trajektorien (auch Geschwindigkeiten) von Fahrzeugen können genau nachvollzogen werden. Ein Unfall kann z.B. genau aus dem Blickwinkel eines beteiligten Fahrers im zeitlichen Verlauf rekonstruiert werden, was ggf. Rückschlüsse auf ein Verschulden zulässt. Weiterhin ermöglicht die Kenntnis der 3D Positionen und Trajektorien auch neue Verfahren der Verkehrsanalyse. So kann z.B. das Abbiegeverhalten statistisch erfasst werden.

2.2.1. Systemkonzept zur Zwischenbildinterpolation

Zu diesem Zweck wurde das Videostreamingsystem aus 2.1 erweitert, wie in Abb. 14 dargestellt ist. Hierzu werden die Videosignale von den Videoclients an ein 3D Analyse Modul weitergeleitet. Die rekonstruierte 3D Szene kann ggf. weiter übertragen und zur Synthese beliebiger Ansichten verwendet werden.

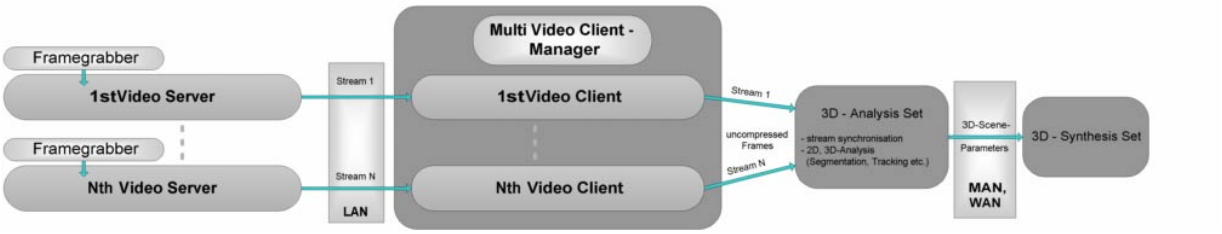


Abb. 14 Erweiterung des Videostreamingsystems zur 3D Rekonstruktion

Die ursprüngliche Idee bei Antragstellung war, zur 3D Rekonstruktion auf am FHG-HHI vorhandenes Know-How zur Stereorekonstruktion aufzubauen, jedoch musste dieser Ansatz verworfen werden. Die vorhandenen Verfahren gehen von einer sehr engen und gleich gerichteten Positionierung der Kameras aus (Stereosetup, an den menschlichen Augen orientiert), wobei die verschiedenen Ansichten einen sehr großen Überlappungsbereich haben und die korrespondierenden 2D Abbildungen der 3D Punkte in den Ansichten relativ eng beieinander liegen. Bei einer Überwachungsanwendung hat man aber genau gegenteilige Vorgaben. Der zu überwachende Bereich soll möglichst gut mit sowenig Kameras wie möglich abgedeckt werden, um die Kosten gering zu halten.

Daher musste ein völlig neuer Ansatz entwickelt werden. Hierzu wird die Szene in statische (Hintergrund) und dynamische (bewegte Verkehrsobjekte) Bestandteile

aufgeteilt, die getrennt voneinander modelliert und in einem Gesamtmodell integriert und visualisiert werden. Dies ist in Abb. 15 veranschaulicht. Einzelheiten zur Zwischenbildinterpolation sind **Anlage B** und **Anlage F** zu entnehmen.

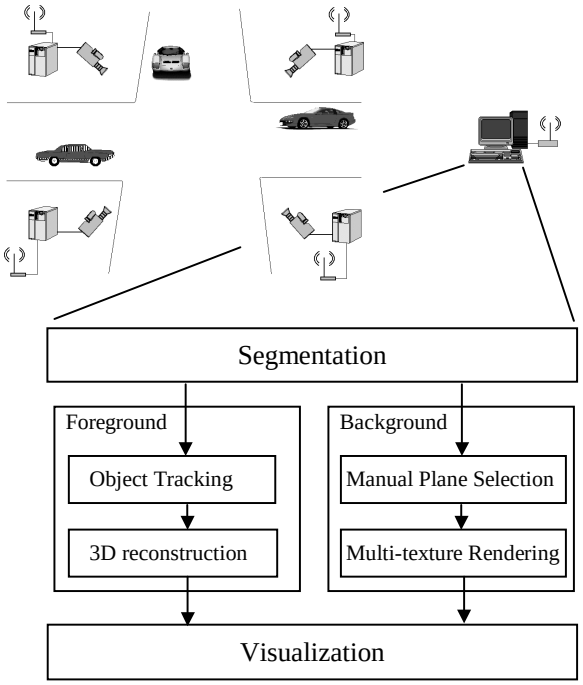


Abb. 15 Prinzip der 3D Rekonstruktion

2.2.2. Kalibrierung

Zur 3D Rekonstruktion aus 2D Ansichten ist die Kenntnis der Beziehung der 2D Bildkoordinaten zu einem relativen 3D Koordinatensystem notwendig (Kalibrierung). Zu diesem Zweck müssen die 3D Koordinaten von mehreren Punkten, die im Blickfeld der Kamera liegen, vermessen werden. Die Idee hierzu ist, dass bei der Installation des Überwachungssystems ohnehin manuelle Arbeiten notwendig sind (Anbringen der Kameras) und dass hierbei auch die Vermessung stattfinden kann. Aus den Koordinaten können dann die Kalibrierungsparameter (extrinsische und intrinsische Kameraparameter) mit einem Standardalgorithmus berechnet werden.

2.2.3. Segmentierung

Der nächste Schritt ist die Segmentierung der bewegten Objekte vom unbewegten Hintergrund. Eine solche Segmentierung wird bereits im Kamerarechner zur Extraktion der Verkehrsdaten vorgenommen, jedoch wird die Segmentierungsinformation nicht übertragen. Dies hätte auch keinen Sinn, da die 3D Rekonstruktion sehr viel höhere Ansprüche an die Qualität der Segmentierung stellt. Es sind sehr genaue Konturen der Objekte notwendig. Dies erfordert aufwendigere Algorithmen. Beim Systemkonzept wird davon ausgegangen, dass in der Zentrale (wo die 3D Rekonstruktion stattfindet) im Gegensatz zu den Kamerarechnern genügend Rechenleistung vorhanden ist, um die notwendigen aufwendigen Algorithmen durchzuführen.

Zur Realisierung der Segmentierung wurden Standardalgorithmen aus einer kommerziellen Softwaretoolbox (HALCON) verwendet und auf die Gegebenheiten bei

den vorliegenden Verkehrsszenen optimiert. Später wurden die Algorithmen durch eigene Implementierungen ersetzt, um im Endsystem nicht auf die kommerzielle Library angewiesen zu sein.

Abb. 16 zeigt ein Beispiel einer segmentierten Verkehrsszene. Die bewegten Vordergrundobjekte sind durch weiße Polygone markiert. Hierbei zeigt sich ein Vorteil der optischen Verkehrserfassung gegenüber Induktionsschleifen, da auch Fußgänger und Radfahrer erfasst werden können.

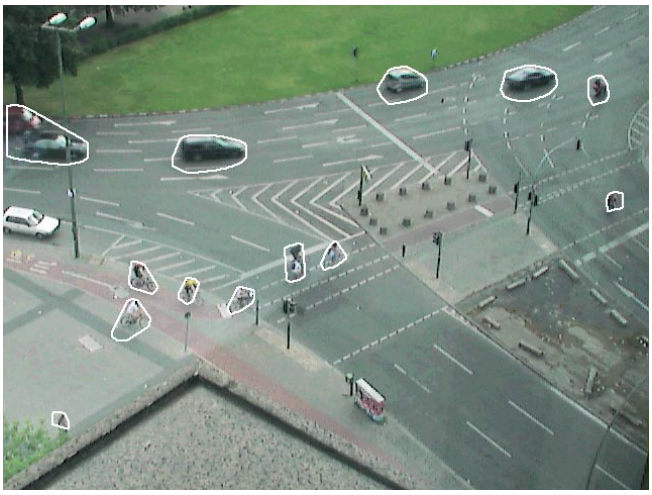


Abb. 16 Segmentierung von bewegten Objekten

Vom Segmentierungsalgorithmus wird automatisch ein Hintergrundbild erzeugt, das die bewegten Objekte nicht mehr enthält. Ein Beispiel ist in Abb. 17 gezeigt.

Abb. 17 Erzeugtes Hintergrundbild

2.2.4. Hintergrundmodell

Die Erzeugung des Hintergrundmodells erfolgt interaktiv. Das Konzept hierzu ist, das eine solche interaktive Modellierung einmal bei der Installation des Systems erfolgt. Im weiteren Verlauf ist die Geometrie fest und es muss nur noch das aktuelle Er-

scheinungsbild angepasst werden. Nur bei signifikanten Änderungen (Baumaßnahmen, Änderung der Kameraposition) muss eine neue Initialisierung vorgenommen werden.

Für alle verfügbaren Kameras werden Hintergrundbilder wie in Abb. 18 gezeigt, erzeugt. Mit einem interaktiven Tool markiert der Benutzer die Grundfläche, zu der die Strasse, Bürgersteige, etc. gehören, sowie senkrechte Seitenflächen wie in Abb. 18 gezeigt.

Abb. 18 Selektierte Grundfläche (schwarz) und Seitenflächen (weiß)

Für die weitere Modellierung wird Vorwissen über die Szene verwendet. Die selektierten Grundflächen gehören zu einem gemeinsamen Objekt, das näherungsweise durch eine Ebene repräsentiert werden kann. Es wird ein virtueller 3D Raum definiert. Die Grundfläche wird als eine Ebene in diesem 3D Raum modelliert. Durch die Kenntnis der Kalibrierungsparameter können die in den Ansichten selektierten Grundflächen auf diese Ebene projiziert werden. Abb. 19 zeigt ein Beispiel, wobei die Ansichten aus 3 Kameras auf dieselbe Ebene projiziert wurden. Man erkennt die Aufnahmerichtungen der Kameras anhand der Projektion des Bildinhalts.

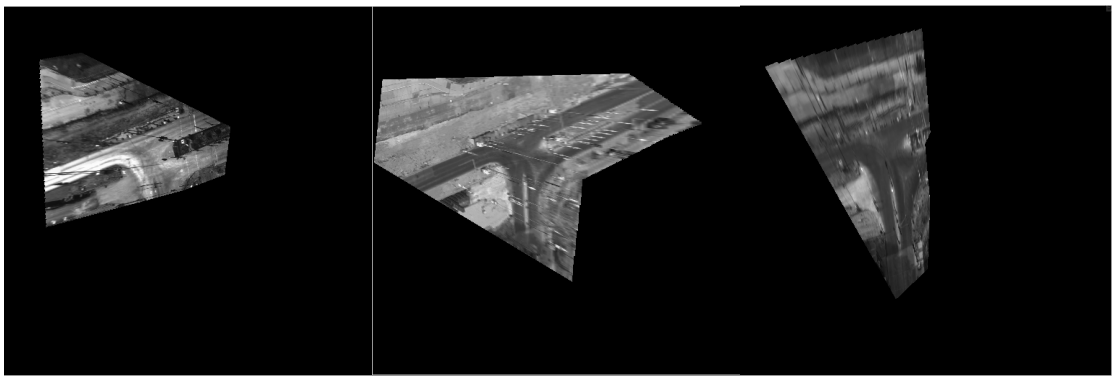


Abb. 19 Projektion der Grundflächen auf eine gemeinsame 3D Ebene

Selektierte Seitenflächen werden dem Modell hinzugefügt, wie in Abb. 20 gezeigt. Die Überlagerung der einzelnen Texturen ist weiter unten beschrieben.

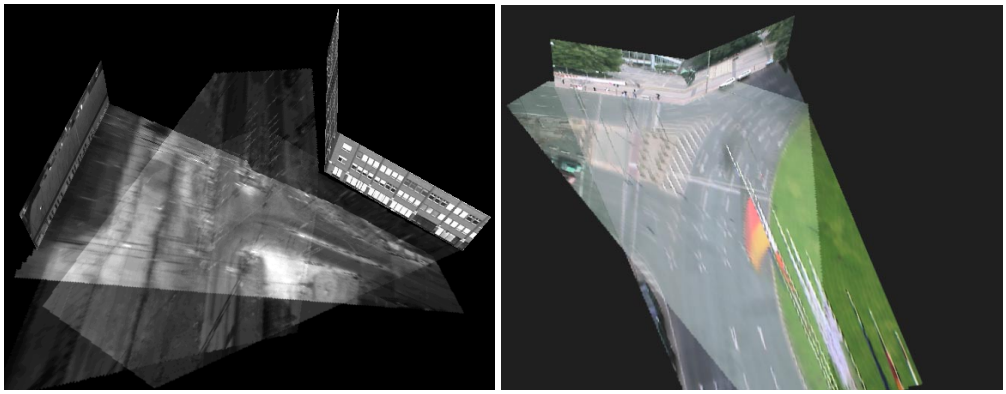


Abb. 20 3D Hintergrundmodelle mit Grundflächen und Seitenflächen

2.2.5. View-dependent Texture Mapping

Natürliche Materialien sehen je nach Reflexionseigenschaften und Lichteinfall oft aus unterschiedlichen Richtungen sehr verschieden aus. Samt ist ein gutes Beispiel, das je nach Blickwinkel sehr unterschiedlich erscheint. Insbesondere aber auch der Asphaltbelag einer Strasse kann je nach Sonneneinstrahlung, Reflektion bei Nässe und Kameraposition fast weiß oder schwarz aussehen. Eine einfache Interpolation der Hintergrundbilder würde daher nur unbefriedigende Ergebnisse bei der Darstellung liefern.

Zur Lösung des Problems wurde ein Verfahren zur blickwinkelangepassten Interpolation (View-dependent Texture Mapping) entwickelt. Das Prinzip ist in Abb. 21 veranschaulicht. Es ist eine Grundfläche mit 2 Texturen vorhanden. Zu jeder projizierten Textur wird ein Normalenvektor $\boldsymbol{n}_i$ definiert, der in Richtung der entsprechenden Kamera zeigt. Es wird eine bezüglich der Szene senkrechte Lichtquelle $\boldsymbol{v}_{LS}$ definiert.

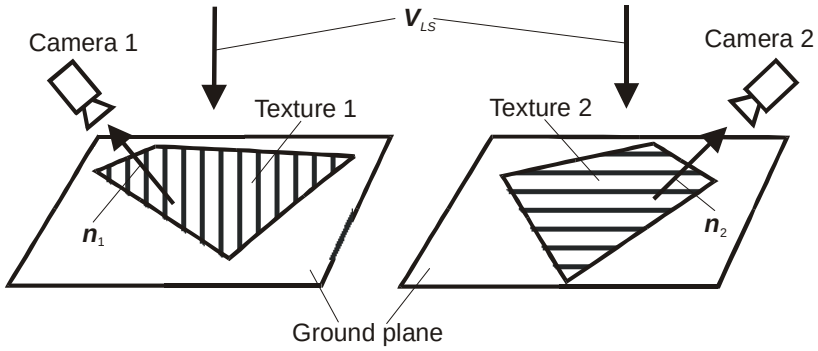


Abb. 21 Geometriedefinitionen für View-Dependent Texture Mapping

Soll nun eine virtuelle Ansicht aus einer bestimmten Richtung ($\boldsymbol{v}_{VIEW}$ in Abb. 22) erzeugt werden, so wird für jede einzelne Textur ein Gewichtungsfaktor berechnet. Dieser Faktor ist abhängig von dem Winkel zwischen der Richtung der zu erzeugenden virtuellen Ansicht und der Kamerarichtung. Dann wird eine gewichtete Interpolation der Texturen vorgenommen. Dies stellt sicher, dass virtuelle Ansichten an den originalen Kamerapositionen mit diesen übereinstimmen. Zwischenansichten werden gewichtet interpoliert. Je näher man an einer originalen Kameraposition ist, desto stärker die Gewichtung.

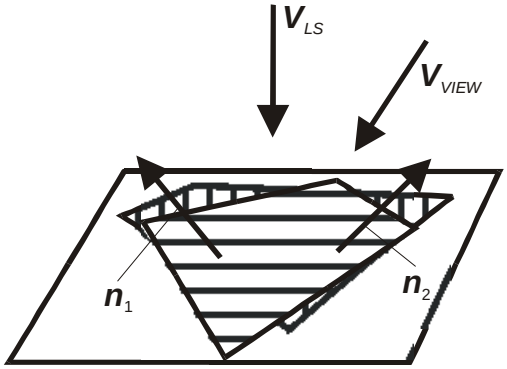


Abb. 22 Erzeugung einer virtuellen Ansicht

Abb. 23 veranschaulich die Berechnung der Gewichtung. θ_i bezeichnen die Winkel zwischen der virtuellen Blickrichtung und den Kamerarichtungen C_i. Zunächst werden individuelle Gewichte w_i berechnet nach:

$$w_i = \begin{cases} \dfrac{\cos\theta_i}{1-\cos\theta_i}, & \cos\theta_i \neq 1 \\ Float\_Max, & \cos\theta_i = 1 \end{cases}.$$

D.h. falls der Winkel zu null wird (virtuelle Ansicht in Richtung einer originalen Kamera) ergibt sich ein sehr großer Wert. Demgegenüber können alle anderen Gewichte vernachlässigt werden. Damit jedoch keine Überbelichtung geschieht, werden die Gewichte normiert, sodass sie in der Summe immer eins ergeben:

$$a_i = \frac{w_i}{\sum_{\forall i} w_i}.$$

Mit dieser Art der Interpolation wird eine möglichst realistische Erzeugung von virtuellen Ansichten und damit eine realistische Abbildung bei einer virtuellen Kamerafahrt durch die Szene ermöglicht.

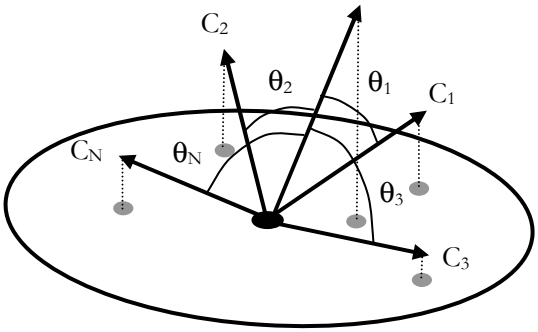


Abb. 23 Unstructured Lumigraph Rendering

Abb. 24 zeigt Beispiele von erzeugten Ansichten. An originalen Kamerapositionen ergibt sich das original Kamerabild. Dazwischen wird graduell mit angepassten Gewichtungen interpoliert.

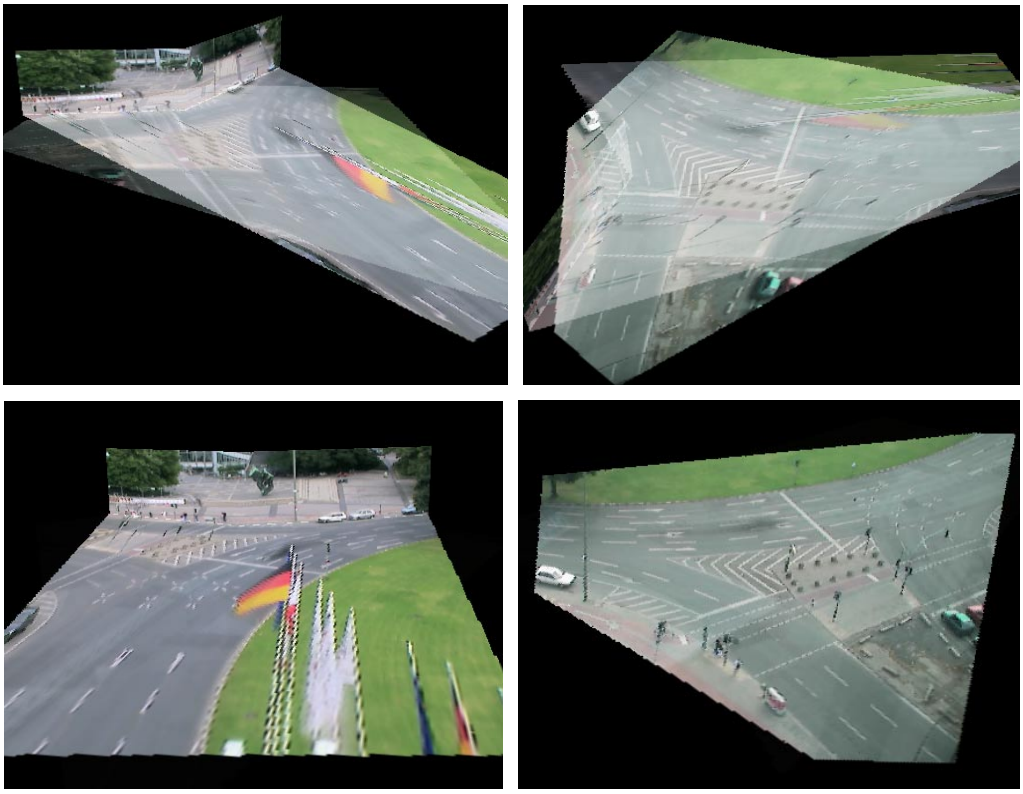


Abb. 24 Erzeugte Ansichten; oben: interpoliert; unten: an originalen Kamerapositionen

2.2.6. Rekonstruktion dynamischer Objekte

Die Rekonstruktion bewegter Objekte geht von der Segmentierung aus, die in Abb. 16 veranschaulicht ist. Die einzelnen Objekte werden über die Zeit verfolgt, wobei ein robuster Kalman-Filter Algorithmus eingesetzt wird, der statistische Bewegungsmodelle verwendet. Dieses 2D Tracking wird für alle Kameras parallel durchgeführt. Die einzelnen 2D Ansichten, die zu einem bewegten 3D Objekt gehören, werden dann zu einem 3D Objekt in Beziehung gesetzt. Wie in Abb. 25 gezeigt, können die segmentierten 2D Ansichten mit Hilfe der Kalibrierungsdaten in das 3D Modell projiziert werden. Zur eindeutigen Zuordnung werden die Schwerpunkte der bestimmten 2D Konturen auf die Grundfläche projiziert. Dabei liegen die Durchstosspunkte der Strahlen, die zu einem Objekt gehören, eng beieinander (aufgrund von Ungenauigkeiten bei Segmentierung und Tracking ergeben sich Abweichungen). Damit kann eine eindeutige Zuordnung der Ansichten zu einem 3D Objekt vorgenommen werden (Datenfusion). Die Position des 3D Objekts (in x und y) in der 3D Szene ergibt sich als Mittelwert der Durchstoßpunkte. Als Höhe wird z = 0 festgelegt, da angenommen werden kann, dass sich die Objekte auf der Grundfläche befinden (ansonsten würden sie fliegen). Schließlich wird auch die 3D Bewegung der Objekte verfolgt, d.h. die 3D Trajektorien werden bestimmt, wobei wiederum Spezielle Kalman-Filter und Bewegungsmodelle verwendet werden.

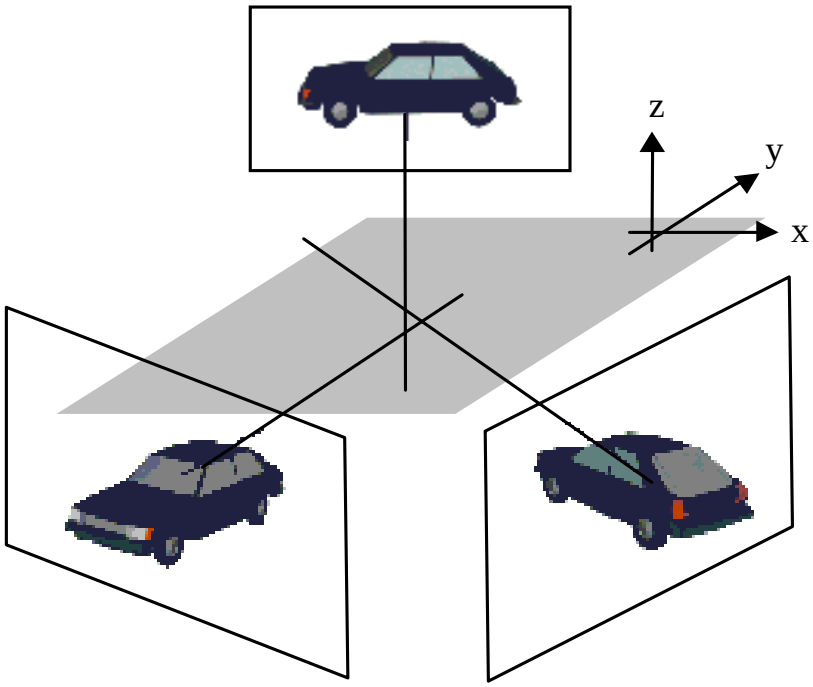


Abb. 25 Datenfusion der einzelnen Ansichten zu einem 3D Objekt und Positionierung in der 3D Szene

Zur Visualisierung der bewegten Objekte im Modell werden vordefinierte Drahtgittermodelle verwendet. Dies ist in Abb. 26 veranschaulicht. Ein 3D Modell eines Autos wird anhand der segmentierten Ansichten in seiner Größe skaliert, in dem ermittelten 3D Punkt im Gesamtmodell positioniert und anhand der Trajektorie in Fahrtrichtung ausgerichtet. Die Originaltexturen werden auf das Drahtgittermodell gemappt. Über die Zeit wird das Objekt entsprechend der ermittelten 3D Trajektorie im Modell bewegt.

Auch andere Arten von Objekten können auf diese Art modelliert werden. Fußgänger und Radfahrer werden z.B. durch senkrechte Ebenen im 3D Raum dargestellt, die in Größe und Form angepasst und mit Originaltextur versehen sind. Für LKW und Busse müssen entsprechende 3D Objekte verwendet werden. Hierzu wären eine automatische Erkennung der Art der Objekte und eine entsprechende Auswahl geeigneter Modelle aus einer vordefinierten Datenbank notwendig. Dieser Arbeitspunkt konnte bis zum Projektende noch nicht vollendet werden. Es werden wie beschrieben vordefinierte Modelle für Autos, Fußgänger und Radfahrer verwendet. Das Konzept zur automatischen Auswahl ist entwickelt worden und müsste nur noch umgesetzt werden.

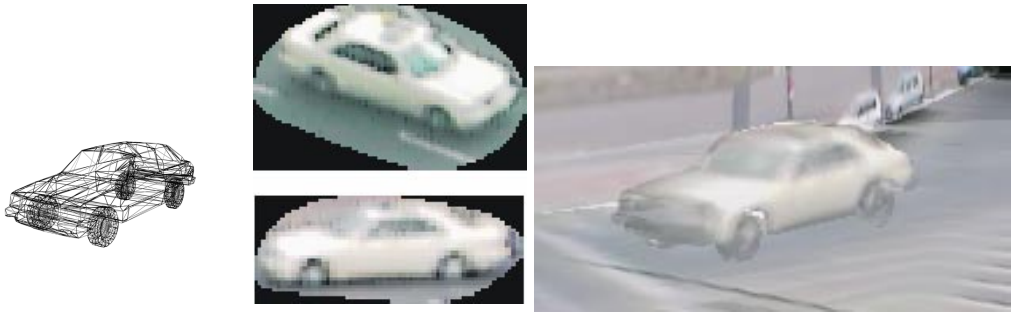


Abb. 26 3D Rekonstruktion eines Fahrzeugs aus einem 3D Modell und segmentierten Ansichten

2.2.7. Integration im Gesamtmodell

Statische und dynamische Teile werden schließlich in einem Gesamtmodell integriert. Abb. 27 zeigt einige virtuelle Ansichten einer Szene vom Ernst-Reuter-Platz, die mit 2 Kameras aufgezeichnet wurde.

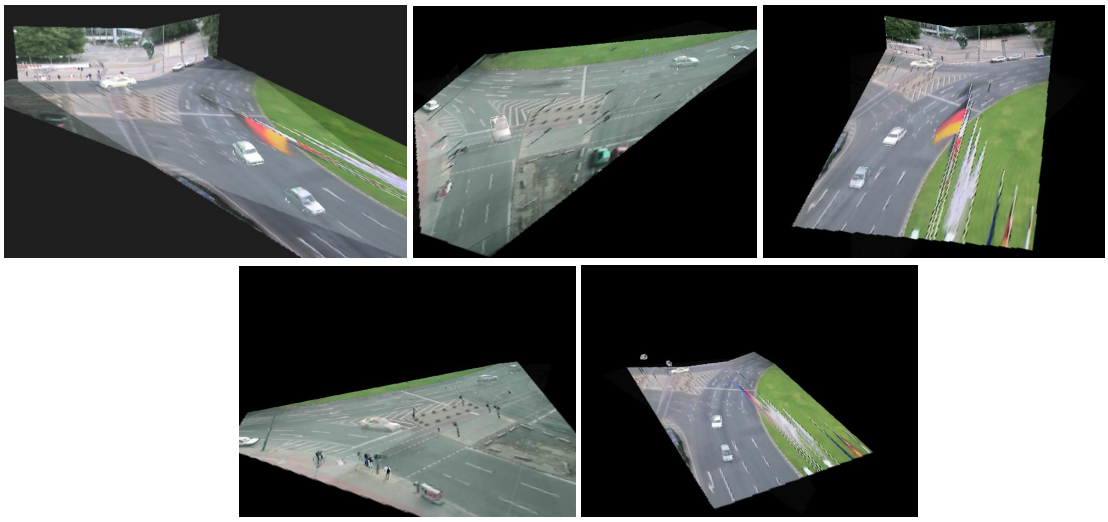


Abb. 27 Virtuelle Ansichten der rekonstruierten 3D Szene mit bewegten 3D Objekten

Die Bildwiederholrate ist im OIS-Projekt mit 5 Hz relativ gering. Dies reicht für das primäre Ziel der Verkehrdatenextraktion aus und ist dabei gering genug, um eine vertretbare Komplexität des Systems zu gewährleisten. Für eine ruckelfreie Animation des 3D Modells ist ein solcher Wert allerdings viel zu gering. Typischerweise erfolgt das Rendering mit 60 Hz und eine flüssige Bewegung zu gewährleisten. Aufgrund der Kenntnis der 3D Trajektorien der bewegten Objekte können aber auch zeitliche Zwischenansichten erzeugt werden. Damit wird die Renderingfrequenz unabhängig von der Wiederholrate der Eingabebilder, d.h. die Szene kann mit einer höheren Rate angezeigt werden. Für jeden Renderingzeitpunkt wird die Position aller 3D Objekt anhand der 3D Trajektorien interpoliert. Die Objekte werden entsprechend bewegt und dargestellt.

2.2.8. Implementierung, Test und Schlussfolgerungen

Während der Projektlaufzeit wurden bis auf die automatische Auswahl geeigneter 3D Modelle alle Teile 3D Rekonstruktionssystems implementiert und getestet. Die Module zur Segmentierung und zum 2D Tracking wurden in den Stand-alone-Demonstrator des FHG-HHI integriert und im Livebetrieb getestet. Dieser Teil wurde bzw. wird auf verschiedenen Events (Lange Nacht der Wissenschaften, Internationale Funkausstellung, CEBIT) einem breiten Publikum als Livesegmentierer präsentiert. Die anderen Module wurden bisher noch nicht im Livebetrieb getestet, obwohl die Integration von der Softwareseite aufgrund des durchgeführten Softwareengineering kein Problem darstellen wird.

Die weiteren Module wurden anhand von aufgezeichneten Testsequenzen getestet und optimiert. Dabei konnte die Funktionalität nachgewiesen werden. Bezüglich der Robustheit bleiben jedoch noch einige Probleme. Die Segmentierung ist konzeptionell immer eine Schätzung, die immer mit einer Restfehlerwahrscheinlichkeit behaftet ist. Ziel kann es nur sein, diese durch geeignete Verfahren und deren Optimierung zu minimieren. In den Experimenten sind aus verschiedenen Gründen immer wieder Segmentierungsfehler aufgetreten.

Bei enger Bewegung von Fahrzeugen, die farblich sehr ähnlich waren, wurden diese häufig zu einem Objekt zusammengefasst (Clustering). Diesem Effekt könnte durch eine gemeinsame Auswertung der Information aller Kameras entgegengewirkt werden, da Cluster häufig in anderen Ansichten mit unterschiedlichem Blickwinkel als getrennte Objekte erscheinen. Auch Schattenwurf kann zu ungenauer Erkennung der Objektgrenzen führen. Hierzu wurde ein geeigneter Algorithmus entwickelt, der die Segmentierung in einem andern Farbraum (HSV anstelle RGB) durchführt. Bei plötzlicher Änderung der Lichtverhältnisse (z.B. Wolke schiebt sich vor die Sonne) wird die Segmentierung instabil. Dem wird durch ein adpatives Verfahren entgegengewirkt, das die Segmentierung automatisch auf die neuen Lichtverhältnisse einstellt. Weitere Probleme, die sich aus dem Outdoor-Szenario ergeben (z.B. Regen, Nebel, Schnee, Dunkelheit) konnten bisher nicht berücksichtigt werden.

Daher ist zu folgern, dass das entwickelte System zu einer vollautomatischen 3D Rekonstruktion bei Forderung von absoluter Fehlerfreiheit zurzeit nicht geeignet ist. Trotzdem hat es für den Einsatz im OIS-Gesamtsystem einen hohen Wert.

Zum einen kann die Statistik der auftretenden Fehler durch ausgiebige Tests gemessen werden. Es können Kenngrößen wie Fehlerwahrscheinlichkeit, Varianz, etc. angegeben werden, z.B. auch für verschiedene Wetterverhältnisse. Bei einer Anwendung des Systems zur automatischen Verkehrsanalyse (z.B. Abbiegeverhalten) können diese Kenngrößen einbezogen werden. Die ermittelten Daten müssten dann mit Confidencewerten versehen werden, die sich aus der Fehlerstatistik ergeben.

Bei der Visualisierung führen Segmentierungsfehler zu fehlerhafter Darstellung. Dies kann bei manchen Anwendungen tolerierbar sein, wenn es z.B. um die reine Visualisierung geht. Dann ist auch eine vollautomatische Implementierung möglich. In anderen Fällen kann die Fehlerfreiheit durch interaktive, d.h. benutzerassistierte Verarbeitung gewährleistet werden. Das Szenario hierzu wären Offline-Anwendungen, bei denen mit aufgezeichneten Videodaten gearbeitet wird. Bei Fehlern könnte ein Be-

nutzer eingreifen und entsprechende Korrekturen vornehmen. Ein solcher Aufwand ist in manchen Fällen sicher gewährleistet (z.B. Gerichtsverhandlungen).

Bezüglich der im Projektantrag angegebenen einzelnen Arbeitspunkte für die Zwischenbildinterpolation (siehe 1.1.2) haben sich wie oben begründet einige Änderungen ergeben. Diese ermöglichten es, das globale Projektziel der freien Navigation über die Kreuzung trotz der speziellen, sehr ungünstigen Gegebenheiten im OIS-Gesamtprojekt (wenige Kameras mit sehr unterschiedlicher Ausrichtung) zu erreichen. Im Folgenden werden die erzielten Ergebnisse in Bezug zu den Arbeitspunkten in 1.1.2 gesetzt.

2.2.8.1. Entwicklung von Disparitätsgestützten Verfahren zur Zwischenbildinterpolation (Arbeitspaket 3311)

Wie beschrieben musste der disparitätsgestützte Ansatz verworfen werden. Der stattdessen neu entwickelte modellbasierte Ansatz unter Ausnutzung von Vorwissen über die Szene ermöglichte trotzdem die Erfüllung dieses wesentlichen Arbeitspunktes. Es konnte gezeigt werden, dass auf diese Weise auch bei sehr ungünstigen Kamerakonfigurationen die Interpolation beliebiger Zwischenbilder möglich ist.

2.2.8.2. Navigation über die Kreuzung (Arbeitspaket 3312)

Die freie Navigation über die Kreuzung konnte demonstriert werden. Die Verknüpfung mit MPEG-4-BIFS konnte nicht durchgeführt werden, da die notwendige Technologie (View-dependent Texture Mapping) noch nicht von MPEG-4 unterstützt wird. Jedoch ist wie beschrieben diese Technologie aus dem Projekt heraus in eine Erweiterung des Standards eingebracht worden, womit in Zukunft eine standardkonforme Repräsentation der Daten ermöglicht wird.

2.2.8.3. Entwicklung von geeigneten Codierverfahren (Arbeitspaket 3313)

Im neuen Verfahren werden keine Videosequenzen übertragen, sondern dynamische 3D Objekte mit Texturen. Die Codierverfahren hierfür werden von MPEG-4 bereits unterstützt.

2.2.8.4. Entwicklung von geeigneten Algorithmen zur Verkehrsdatenextraktion aus den oben genannten Datengrundlagen (Arbeitspaket 3314)

Die modellbasierte Beschreibung ermöglicht wie beschrieben die direkte Ableitung von 3D Information der bewegten Objekte (3D Position, 3D Trajektorien, Geschwindigkeiten).

2.2.8.5. Umsetzung der Algorithmen in echtzeitfähige Software (Arbeitspaket 3315)

Dieser Arbeitspunkt wurde mit dem neuen Ansatz erfüllt.

2.2.8.6. Planung und Durchführungen der Tests im Labor und Feldversuch (Arbeitspaket 3316)

Dieser Arbeitspunkt wurde mit dem neuen Ansatz erfüllt.

2.2.9. Verwertung

Die 3D Rekonstruktion stellt eine Software dar, die für eine Verkehrszentrale vorgesehen ist. Sie kann in verschiedener Weise konfiguriert und somit in verschiedenen Gesamtpaketen vermarktet werden. Die Funktionalität steht zwar nicht im Mittelpunkt des OIS-Projekts, stellt aber einen bedeutenden Zusatz dar, der eine Abgrenzung (Added Value) von existierenden Systemen zulässt.

Die entwickelten Konzepte und Algorithmen stellen eine Erweiterung bisher verfügbarer Technologie dar und sind dem wissenschaftlichen Fachpublikum auf verschiedenen bedeutenden Fachkonferenzen (IEEE ICME, WIAMIS) präsentiert worden. Die entwickelten Methoden zur blickwinkeladaptiven Texturinterpolation wurden erfolgreich in einen neuen MPEG Standard zur Computergrafik (AFX) eingebracht, was ggf. eine weltweite Verbreitung der in OIS entwickelten Technologie bedeutet. Außerdem wurde das System einer breiten Öffentlichkeit auf bedeutenden Messen präsentiert (IFA, CEBIT).

2.3. Mitarbeit in MPEG 3DAV

Die Abteilung Bildsignalverarbeitung des FHG-HHI ist bereits seit vielen Jahren maßgeblich an der Entwicklung verschiedener MPEG und ITU Standards beteiligt (MPEG-4, MPEG-7, H.264/AVC). Verschiedene Mitarbeiter haben Arbeitsgruppen geleitet. Eine Vielzahl von Vorschlägen aus der Abteilung ist in verschiedene Standards aufgenommen worden. Weiterhin wurde maßgeblich an der technischen Entwicklung in vergleichenden Experimenten sowie der Formulierung von Standardtext mitgewirkt. Auch die Verantwortung für die Entwicklung der Referenzsoftware für verschiedene Standards lag bzw. liegt beim FHG-HHI.

Das Engagement in MPEG ist im Rahmen verschiedener Projekte erbracht worden. Der Nutzen für das Institute und die Projekte ist vielschichtig. Dies können z.B. direkte Lizenzeinnahmen sein, wenn es gelingt, eigene Patente in einen Standard einzubringen, der weltweite Verwendung findet (siehe z.B. MP3 bei Fraunhofer IIS). Wichtig ist jedoch insbesondere auch der Aufbau von entsprechendem Know-how über die Standards und deren Anwendung, sowie die Entwicklung von entsprechender Software und Systemen. Auf diese Weise konnte sich das FHG-HHI z.B. als weltweites Kompetenzzentrum für MPEG-4, MPEG-7 und H.264/AVC etablieren. Dies beinhaltet auch den Aufbau von intensiven weltweiten Kontakten zu potentiellen Partnern und Kunden. Eine Vielzahl von Projekten und Industriekooperationen konnte auf Basis dieser Expertise und der Kontakte eingeworben werden. Das OIS Projekt selbst ist ein sehr gutes Beispiel dafür. Hier brachte das FHG-HHI seine Expertise auf dem Gebiet der MPEG-4 Codierung ein, um den AP 3200 zur Videokommunikation zu realisieren. Ähnliche Projekte wurden und werden mit verschiedenen deutschen Unternehmen realisiert (z.B. Siemens, Bosch, Deutsche Telekom, Thomson Multimedia, Sciworx).

Unter dem Namen 3DAV ist im Rahmen von MPEG zeitgleich mit dem Start von OIS eine neue Initiative zur Standardisierung von 3D-Video und -Audio gestartet worden. Es sollen neue AV-Formate entwickelt werden, die über die heute verfügbaren klassischen 2D Formate mit vordefiniertem Standpunkt hinausgehen. Ein wesentliches

Merkmal hierbei ist die Interaktivität, d.h. der Zuschauer soll in gewissen Grenzen seinen Standpunkt und/oder Blickwinkel frei wählen können. Hierzu zählen Multiple-View-Video (mehrere Kameras), Rundumvideo sowie Stereo-Video. Die Entwicklung genau solcher Technologie stellte den Kern des OIS Arbeitspakets 3310 dar, das vom FHG-HHI bearbeitet wurde.

Die Arbeitsgruppe hat sich daher von Beginn an federführend an 3DAV beteiligt [18]-[51]. Dies geschah zum Teil in Kooperation mit dem EU-Projekt ATTEST an dem die Abteilung ebenfalls beteiligt war und in dem Verfahren zu 3D-TV entwickelt wurden. Herr Dr. Smolic ist Chairman der 3DAV-Gruppe in MPEG, Editor des „Applications and Requirements for 3DAV“-Dokuments [20], das die Anforderungen eines solchen neuen Standards definiert und Editor des „Report on 3DAV Exploration“-Dokuments [21], das die Ergebnisse der Arbeiten zusammenfasst.

Herr Smolic hat maßgeblich an der Definition und Beschreibung des Arbeitsgebietes (Interactive 3D Video and Audio), der Anwendungsszenarien sowie der abgeleiteten Requirements mitgewirkt. Dabei hat er auch die Interessen von OIS vertreten und verwirklicht. Dies bedeutet, dass die Technologie, die in AP 3310 entwickelt wurde, zu den Anwendungsszenarien und Requirements passt. Weiterhin wurden in 3DAV experimentellen Untersuchungen durchgeführt, die das Ziel hatten, potentielle Technologie im technischen Detail zu evaluieren [19]. Nach Abschluss der Initiierungsphase (Exploration) werden nun im Rahmen von 3DAV konkrete Standardisierungsarbeiten begonnen [18].

Für das FHG-HHI und OIS hat dieses Engagement in 3DAV einigen Nutzen gebracht. Durch die intensive und regelmäßige Zusammenarbeit mit weltweit führenden Wissenschaftlern in der 3D Szenenrekonstruktion konnten wir uns ein umfassendes Know-how zur Realisierung des Arbeitspaketes 3310 aneignen. Als wichtige Kooperationspartner sind dabei z.B. die Eidgenössische Technische Hochschule (ETH) Zürich, das Max-Planck-Institut für Informatik (MPI) Saarbrücken, Mitsubishi Electric Research Labs (MERL) Cambridge, MA, USA, STMicroelectronics, CA, USA, NTT Japan, SEGA Japan, Electronics and Telecommunications Research Institute (ETRI) Korea, University of Nagoya Japan, University of Kyoto Japan, Hanyang University Korea, France Telecom und die TU Ilmenau zu nennen.

Dies war insbesondere wichtig, da das ursprüngliche Antragskonzept, eine disparitätsgestützte Zwischenbildinterpolation durchzuführen, sich für das gegebene Szenen- und Kamerasetup an einer Verkehrkreuzung als ungeeignet erwiesen hat. Der stattdessen verfolgte Ansatz einer Unterteilung der Verkehrsszene in statische und dynamische Anteile und deren getrennte Modellierung durch Multitextureobjekte steht in engem Zusammenhang zu Vorschlägen, die in 3DAV untersucht werden.

Umgekehrt konnten die Ergebnisse aus OIS in MPEG eingebracht werden. Die in Abschnitt 2.2.5 beschriebenen Methoden zum View-dependent Texture Mapping wurden für eine aktuelle Erweiterung des MPEG-4 Computergrafik Standards Animation Framework eXtension (AFX) vorgeschlagen und akzeptiert. Dieses Ergebnis aus OIS wird damit im neuen Standard weltweite Verbreitung finden.

Für die Zukunft ergeben sich aus dem im Rahmen von 3DAV erarbeiteten unfassenden Know-how im Bereich 3D Video zahlreiche Perspektiven für aufbauende Projek-

te und Industriekooperationen. Hierfür ist auch die Rolle von Herrn Smolic als Chairman und Editor von Bedeutung, die dem OIS-Projekt als auch dem Institut eine hohe internationale Reputation und Sichtbarkeit verschafft hat, die sich z.B. in eingeladenen Veröffentlichungen in namhaften Fachzeitschriften [2], [3] und Vorträgen [55]-[63] widerspiegelt.

Auf dem Emailreflektor von 3DAV sind über 250 Teilnehmer registriert, darunter Vertreter sehr vieler namhaften Firmen und Institute im Bereich Multimedia und Computergraphik. An den Treffen der Arbeitsgruppe nehmen jeweils 30-40 Vertreter von Firmen und Instituten aus aller Welt teil und auch in den Plenarsitzungen von MPEG besteht ein großes Interesse an den Arbeiten in 3DAV. Im Rahmen der Arbeiten konnten darüber hinaus sehr viele interessante Kontakte zu Vertretern von Firmen und Instituten aus aller Welt geknüpft werden.

Neben den Aktivitäten in 3DAV wurden weitere relevante Entwicklungen in MPEG (Joint Video Team, AHG on Scalable Video Coding) verfolgt, da das FHG-HHI in OIS auch die Aufgaben der MPEG-Videocodierung und –Übertragung übernommen hat (AP 3200). Zur erfolgreichen Durchführung dieser Arbeiten war es notwendig, auch weiterhin an den Entwicklungen in MPEG (Video und Systems) teilzunehmen und auf dem neuesten Stand zu bleiben.

Zusammenfassend kann festgehalten werden, dass sich die Mitarbeit in MPEG 3DAV für den Projekterfolg von OIS und für potentielle aufbauende Vorhaben als äußerst fruchtbar und wichtig erwiesen hat.

2.4. Veröffentlichungen

Im Folgenden sind alle im Zusammenhang mit OIS und aufbauenden Arbeiten entstandenen Veröffentlichungen aufgeführt.

2.4.1. Artikel in Fachzeitschriften

[1] K. Mueller, A. Smolic, M. Droese, P. Voigt, and T. Wiegand, 3D Reconstruction of a Dynamic Environment with Fully Calibrated Background for Traffic Scenes, accepted for publication in IEEE Trans. on CSVT.

[2] A. Smolic, and P. Kauff, Interactive 3D Video Representation and Coding Technologies, Invited Paper, accepted for publication in Proceedings of the IEEE, Special Issue on Advances in Video Coding and Delivery, scheduled December 2004.

[3] A. Smolic, and D. McCutchen, 3DAV Exploration of Video-Based Rendering Technology in MPEG, Invited Paper, IEEE Trans. on CSVT, Special Issue on Immersive Communications, Vol. 14, No. 9, pp. 348-356, March 2004.

[4] A. Smolic, C. Grünheit, and T. Wiegand, Interaktives Streaming von hochaufgelösten 360°-Panoramen, Invited Paper, FKT, Fachzeitschrift für Fernsehen, Film und Elektronische Medien, January 2003.

2.4.2. Artikel für Konferenzen

[5] K. Mueller, A. Smolic, P. Merkle, B. Kaspar, P. Eisert, and T. Wiegand, 3D Reconstruction of Natural Scenes with View-Adaptive Multi-Texturing, Submitted to: 3DPVT'04, International Symposium on 3D Data Processing, Visualization, and Transmission, Thessaloniki, Greece, September 6.-9. 2004.

[6] A. Smolic, K. Mueller, P. Merkle, T. Rein, Y. Vatis, P. Eisert, and T. Wiegand, Representation, Coding, and Rendering of 3D Video Objects with MPEG-4, Submitted to: MMSP'04, IEEE International Workshop on Multimedia Signal Processing, Siena, Italy, Sept. 29-Oct 1. 2004.

[7] S. Wuermlin, A. Smolic, et al., Image-space Free Viewpoint Video, Submitted to: ACM Multimedia 2004, New York, NY, USA, October 10.-16. 2004.

[8] A. Smolic, K. Mueller, P. Merkle, T. Rein, P. Eisert, and T. Wiegand, Free Viewpoint Video Extraction, Representation, Coding, and Rendering, To appear: ICIP 2003, IEEE International Conference on Image Processing, Singapore, Singapore, October 24.-27. 2004.

[9] K. Mueller, A. Smolic, B. Kaspar, P. Merkle, T. Rein, P. Eisert, and T. Wiegand, Octree Voxel Modeling with Multi-view Texturing in Cultural Heritage Scenarios, Proc. WIAMIS 2004, 5th European Workshop on Image Analysis for Multimedia Interactive Services, Lisbon, Portugal, April 21.-23. 2004.

[10] A. Smolic, and D. McCutchen, MPEG 3DAV - Video-based Rendering for Interactive TV Applications, Proc. 10. Dortmunder Fernsehseminar, ITG/FKTG-Fachtagung, Dortmund, Germany, September 2003.

[11] K. Müller, A. Smolic, M. Droese, P. Voigt, and T. Wiegand, Multi-Texture Modeling of 3D Traffic Scenes, Proc. ICME 2003, IEEE International Conference on Multimedia & Expo, Baltimore, MD, USA, July 6.-9. 2003.

[12] K. Müller, A. Smolic, M. Droese, P. Voigt, and T. Wiegand, 3D Modeling of Traffic Scenes using Multi-texture Surfaces, Proc. PCS 2003, Picture Coding Symposium, St. Malo, France, April 23.-25. 2003.

[13] A. Smolic, and D. McCutchen, Efficient Representation and Coding of Omnidirectional Video Using MPEG-4, Proc. WIAMIS 2003, 4th European Workshop on Image Analysis for Multimedia Interactive Services, London, UK, April 9.-11. 2003.

[14] A. Smolic, K. Müller, M. Droese, P. Voigt, and T. Wiegand, Multiple View Video Streaming and 3D-Scene Reconstruction for Traffic Surveillance, Proc. WIAMIS 2003, 4th European Workshop on Image Analysis for Multimedia Interactive Services, London, UK, April 9.-11. 2003.

[15] C. Grünheit, A. Smolic and T. Wiegand, Efficient Representation and Interactive Streaming of High-Resolution Panoramic Views, Proc. ICIP2002, IEEE International Conference on Image Processing, Rochester, NY, USA, September 22.-25. 2002.

[16] A. Smolic, Robust Generation of 360° Panoramic Views from Consumer Video Sequences, Proc. VIPromCom2002, International Symposium on Video / Image Processing and Multimedia Communications, Zadar, Croatia, June 16.-19. 2002.

[17] C. Grünheit, A. Smolic and T. Wiegand, Interaktives Streaming von hochaufgelösten 360°-Panoramen, Proc. 20. FKTG Jahrestagung, Zürich, Suisse, June 10.-13. 2002.

2.4.3. MPEG Beiträge

[18] ISO/IEC JTC1/SC29/WG11, "Call for Comments on 3DAV", Doc. N6051, Gold Coast, Australia, October 2003.

[19] ISO/IEC JTC1/SC29/WG11, "Description of Exploration Experiments in 3DAV", Doc. N5959, Gold Coast, Australia, October 2003.

[20] ISO/IEC JTC1/SC29/WG11, "Applications and Requirements for 3DAV", Doc. N5877, Trondheim, Norway, July 2003.

[21] ISO/IEC JTC1/SC29/WG11, "Report on 3DAV Exploration", Doc. N5878, Trondheim, Norway, July 2003.

[22] A. Smolic, K. Mueller, P. Merkle, T. Rein, Y. Vatis, P. Eisert, and T. Wiegand, Results for EE2 on Model Reconstruction Free Viewpoint Video, ISO/IEC JTC1/SC29/WG11, MPEG04/M10676, Munich, Germany, March 2004.

[23] A. Smolic, S. Heymann, K. Mueller, and Y. Guo, Demonstration of a MPEG-4 Compliant System for Omni-directional Video, ISO/IEC JTC1/SC29/WG11, MPEG04/M10674, Munich, Germany, March 2004.

[24] A. Smolic, Response to the Call for Comments on 3DAV", ISO/IEC JTC1/SC29/WG11, MPEG04/M10673, Munich, Germany, March 2004.

[25] A. Smolic and H. Kimata, AHG on 3DAV Coding", ISO/IEC JTC1/SC29/WG11, MPEG04/M10479, Munich, Germany, March 2004.

[26] A. Smolic and H. Kimata, AHG on 3DAV Coding, ISO/IEC JTC1/SC29/WG11, MPEG03/M10218, Waikoloa, HI, USA, December 2003.

[27] A. Smolic, K. Mueller, and P. Merkle, Preliminary Results on EE2 Using Octree Reconstuction and View-dependent Texture Mapping, ISO/IEC JTC1/SC29/WG11, MPEG03/M10343, Waikoloa, HI, USA, December 2003.

[28] K. Müller, A. Smolic, and T. Rein, Preliminary Results on Core Experiments on View-Dependent Multi-Texturing for MPEG-4 AFX, ISO/IEC JTC1/SC29/WG11, MPEG03/M10344, Waikoloa, HI, USA, December 2003.

[29] K. Mueller and A. Smolic, Study on View-Dependent Multitexturing for MPEG-4 AFX, ISO/IEC JTC1/SC29/WG11, MPEG03/M10152, Gold Coast, Australia, October 2003.

[30] A. Smolic and H. Kimata, AHG on 3DAV Coding, ISO/IEC JTC1/SC29/WG11, MPEG03/M9954, Gold Coast, Australia, October 2003.

[31] C. Fehn, P. Kauff and A. Smolic, Broadcast Transmission of 3D-TV Data, ISO/IEC JTC1/SC29/WG11, MPEG03/M9875, Trondheim, Norway, July 2003.

[32] A. Smolic and D. McCutchen, Requirement for Random Access and View-dependent Partial Decoding and Rendering in 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG03/M9839, Trondheim, Norway, July 2003.

[33] Aljoscha Smolic, Karsten Mueller, Michael Droese, Birgit Kaspar, Philipp Merkle, Peter Eisert and Thomas Wiegand, Multi-texture Surfaces for View-dependent Rendering in Free Viewpoint Video and Graphics, ISO/IEC JTC1/SC29/WG11, MPEG03/M9837, Trondheim, Norway, July 2003.

[34] A. Smolic and H. Kimata, AHG on 3DAV Coding, ISO/IEC JTC1/SC29/WG11, MPEG03/M9635, Trondheim, Norway, July 2003.

[35] C. Fehn, K. Schueuer, P. Kauff and A. Smolic, Coding Results for EE4 in MPEG 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG03/M9561, Pattaya, Thailand, March 2003.

[36] C. Fehn, K. Schueuer, P. Kauff and A. Smolic, Meta-Data Requirements for EE4 in MPEG 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG03/M9559, Pattaya, Thailand, March 2003.

[37] A. Smolic and H. Kimata, AHG on 3DAV Coding, ISO/IEC JTC1/SC29/WG11, MPEG03/M9371, Pattaya, Thailand, March 2003.

[38] A. Smolic, D. McCutchen, Requirement for very high resolution video in 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG02/M9256, Awaji, Japan, December 2002.

[39] C. Fehn, K. Schueuer, I. Feldmann, P. Kauff and A. Smolic, Distribution of AT-TEST Test Sequences for EE4 in MPEG 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG02/M9219, Awaji, Japan, December 2002.

[40] A. Smolic, R.Yamashita, AHG on 3DAV Coding, ISO/IEC JTC1/SC29/WG11, MPEG02/M9097, Awaji, Japan, December 2002.

[41] K. Schueuer, C. Fehn, I. Feldmann, P. Kauff, A. Smolic, Preliminary results on disparity coding with in-loop median filter, ISO/IEC JTC1/SC29/WG11, MPEG02/M9017, Shanghai, China, October 2002.

[42] C. Fehn, K. Schueuer, I. Feldmann, P. Kauff, A. Smolic, Proposed experimental conditions for EE4 in MPEG 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG02/M9016, Shanghai, China, October 2002.

[43] A. Smolic, M. Droese, Results of EE1 on Usage of 3D Mesh Objects for Omni-directional Video, ISO/IEC JTC1/SC29/WG11, MPEG02/M8927, Shanghai, China, October 2002.

[44] A. Smolic, R. Yamashita, AHG on 3DAV Coding, ISO/IEC JTC1/SC29/WG11, MPEG02/M8785, Shanghai, China, October 2002.

[45] A. Smolic, R. Yamashita, Report of Ad hoc Group on 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG02/M8679, Klagenfurt, Austria, July 2002.

[46] David McCutchen, A. Smolic, Spherical Efficiency in Global Photography Mapping, ISO/IEC JTC1/SC29/WG11, MPEG02/M8678, Klagenfurt, Austria, July 2002.

[47] K. Schüür, C. Fehn, P. Kauff, A. Smolic, About the Impact of Disparity Coding on Novel View Synthesis, ISO/IEC JTC1/SC29/WG11, MPEG02/M8676, Klagenfurt, Austria, July 2002.

[48] K. Müller, A. Smolic, Study of 3D Coordinate System and Sensor Parameters for 3DAV, ISO/IEC JTC1/SC29/WG11, MPEG02/M8536, Klagenfurt, Austria, July 2002.

[49] A. Smolic et al (AHG on 3D Video), Draft Requirements for 3D Video, ISO/IEC JTC1/SC29/WG11, MPEG02/M8428, Fairfax, VA, USA, May 2002.

[50] C. Fehn and A. Smolic, Study of some MPEG Tools Related to 3D-Video, ISO/IEC JTC1/SC29/WG11, MPEG02/M8423, Fairfax, VA, USA, May 2002.

[51] A. Smolic, C. Gruenheit and T. Wiegand, Efficient Representation and Interactive Streaming of High-Resolution Panoramic Views using MPEG-4 BIFS, ISO/IEC JTC1/SC29/WG11, MPEG02/M8305, Fairfax, VA, USA, May 2002.

2.4.4. Beiträge zu Ausstellungen

[52] 3D Szenen-Rekonstruktion für interaktive Medien, CEBIT'2004, Hannover, Germany, March 2004.

[53] TraVis 3D – Multiview Video und 3D Rekonstruktion zur Verkehrsüberwachung, IFA'2003, Internationale Funkausstellung, Berlin, Germany, August/September 2003.

[54] Multiview Verkehrsanalyse und 3D Szenenrekonstruktion, Lange Nacht der Wissenschaften, Fraunhofer HHI, Berlin, Germany, June 2003.

2.4.5. Eingeladene Vorträge

[55] A. Smolic, 3D iMedia - 3D Reconstruction and Image-based Rendering for Interactive Media, Colloquium, Hanyang University, Seoul, Korea, April 2004.

[56] A. Smolic, 3D iMedia - 3D Reconstruction and Image-based Rendering for Interactive Media, Colloquium, Information and Communication University (ICU), Daejon, Korea, April 2004.

[57] A. Smolic, 3D iMedia - 3D Reconstruction and Image-based Rendering for Interactive Media, Colloquium, Electronics and Telecommunications Research Institute (ETRI), Daejon, Korea, April 2004.

[58] A. Smolic, 3D iMedia - 3D Reconstruction and Image-based Rendering for Interactive Media, Colloquium, Eidgenössische Technische Hochschule Zürich (ETH), Zürich, Switzerland, April 2004.

[59] A. Smolic, 3D iMedia - 3D Reconstruction and Image-based Rendering for Interactive Media, Colloquium, Technische Universität Ilmenau, Ilmenau, Germany, March 2004.

[60] A. Smolic, Advanced Video Coding Using Video Analysis and 3D Reconstruction and Image-based Rendering for Interactive Media, Colloquium, Mitsubishi Electric Research Labs (MERL), Cambridge, MA, USA, March 2004.

[61] A. Smolic, Video-Based Rendering und 3D Rekonstruktion für Interaktive Medien, Wiesbadener Medien Symposium #3, Codierung Audiovisueller Daten, Fachhochschule Wiesbaden, Germany, October 2003.

[62] A. Smolic, Multiview Videostreaming und 3D Szenenrekonstruktion für Interaktive Multimediadienste, IFA'2003, Internationale Funkausstellung, Berlin, Germany, August 2003.

[63] A. Smolic, MPEG 3DAV – Standardization of 3D Audio and Video, Invited talk, Workshop on Immersive Communication and Broadcast Systems, Fraunhofer HHI, Berlin, Germany, January 2003.

2.4.6. Studien- und Diplomarbeiten

[64] P. Voigt, Entwicklung und Untersuchung eines Multiclients für die Videoüberwachung mit mehreren Kameras, Diplomarbeit an der TU-Berlin, Fakultät für Elektrotechnik und Informatik, Fachgebiet Nachrichtenübertragung, Oktober 2003.

[65] B. Kaspar, Vergleich und Bewertung verschiedener Verfahren zur 3D Objektrekonstruktion aus Multiview-Kameraaufnahmen, Diplomarbeit an der Fachhochschule Köln, Fachbereich Photoingenieurwesen und Medientechnik, Juni 2003.

[66] M. Dröse, Entwicklung von Verfahren zur Segmentierung, Verfolgung und 3D-Rekonstruktion von bewegten Objekten in Verkehrsszenen mit mehreren Ansichten, Diplomarbeit an der TU-Berlin, Fakultät für Elektrotechnik und Informatik, Fachgebiet Nachrichtenübertragung, Januar 2003.

[67] P. Voigt, Client/Server Architektur für die Videoübertragung und -codierung mit mehreren Kameras, Studienarbeit an der TU-Berlin, Fakultät für Elektrotechnik und Informatik, Fachgebiet Nachrichtenübertragung, September 2002.

TEIL III ANLAGEN

Entwicklung und Untersuchung eines Multiclients für die Videoüberwachung mit mehreren Kameras

Patrick Voigt

Diplomarbeit
an der Technischen Universität Berlin
Institut für Telekommunikationssysteme,
Fachgebiet Nachrichtenübertragung

bei Prof. Dr.-Ing. Thomas Sikora

Betreuer :

Dipl.-Ing. Sila Ekmekci	**TU Berlin**
Dipl.-Ing. Karsten Müller	**Heinrich-Hertz-Institut**

Berlin, Oktober 2003

Inhaltsverzeichnis

1 Einleitung

1.1 Szenario

Aufgrund der steigenden Leistungsfähigkeit digitaler Übertragungstechnik haben sich die Einsatzmöglichkeiten von Videoüberwachungssystemen zum Schutz öffentlicher oder privater Objekte in den letzten Jahren stark verbessert. Bisheriger Höhepunkt ist das Vorhaben englischer Entwickler, einzelne bisher noch autarke Überwachungssysteme der Londoner City und in den öffentlichen Verkehrsmitteln zusammenzuschließen, wodurch die Verfolgung einer Person durch die gesamte Stadt möglich wäre. Obwohl die rechtliche Lage in einigen Ländern schwierig ist, werden zunehmend auch Möglichkeiten der automatischen Personenerkennung, sowie der Verhaltenserkennung und Gefahrenabschätzung erforscht.
Ein weiteres Anwendungsgebiet ist die Verkehrsüberwachung. Hier geht es nicht nur um die Verfolgung einzelner Objekte, sondern auch um die Erfassung eines Zustandes von Straßen oder Kreuzungen, um auf Basis der erfassten Daten Ampeln und andere Signalanlagen optimaler steuern zu können. Das Interesse an leistungsfähigeren Verkehrssteuerungssystemen ist groß, denn trotz ausgebauten U-Bahn-, S-Bahn- und Busnetzen ist das Auto immer noch das am meisten genutzte Verkehrsmittel.
Neben der Erfassung statistischer Daten über den Verkehrsfluß spielt die visuelle Überwachung eine wichtige Rolle. Mit Hilfe eines Netzes von Kameras, welche an wichtigen Verkehrsknoten, bekannten Stau- oder Unfallschwerpunkten installiert sind, kann in einer Leitstelle das Geschehen beobachtet und auf Ereignisse schneller reagiert werden. Dabei ist der Einsatz digitaler Technik wichtig, da diese durch die Entwicklung immer billigerer integrierter Schaltkreise die Kosten niedrig halten kann. Darüber hinaus gibt es deutschlandweit bereits zahlreiche lokale, regionale und überregionale Datennetze, welche zur Übertragung der digitalen Daten dienen können. Eine länderübergreifende Übertragung ist natürlich ebenfalls möglich.
Außer dem Aspekt der Übertragung ist auch die Frage der Verarbeitung und Darstellung zu betrachten. Die noch häufig in einfachen Sicherheitssystemen verwendete analoge Technik hat dabei einige Nachteile. Zum einen verbrauchen die zur Anzeige benötigten Röhrenmonitore viel Platz und deutlich mehr Strom verbunden mit höherer Wärmeabgabe. Zudem kann ohne vorgeschaltete Module pro Monitor nur ein Kamerabild angezeigt werden. Hardware zum Aufnehmen der Bilder und zugehörige Speichermedien benötigen zusätzlichen Platz und verursachen nicht zu unterschätzende Kosten.
Dem gegenüber steht die Bearbeitung an Computern und die Anzeige auf LCD-Bildschirmen. Bei digitaler Übertragung von Videodaten und statistischen Informationen ist es kostengünstiger, Empfang und Auswertung beider Datenströme auf einem System zu vereinen, welches auf Standard-PC-Komponenten basiert.
Die einfachere Bedienung von Computertechnik und andere Vorteile haben dazu geführt, das bereits viele professionelle allerdings auch teure Überwachungssysteme eingesetzt werden. So werden zum Beispiel PC-gestützte Systeme mit Anschlußmöglichkeiten für mehrere Kameras sowie integriertem DSP für die Videobearbeitung angeboten. Die Übertragung kann z.B. über den Ethernetanschluß oder

ISDN erfolgen. Die Bilddaten werden aus Kostengründen zumeist mit einfachen Verfahren komprimiert (JPEG, M-JPEG). Auch MPEG-2-Hardware-Encoder werden zunehmend eingesetzt. Die zur Zeit modernsten Codecs MPEG-4/H.26L (H.264) sind jedoch auf Grund der hohen Anforderungen an die Hardware nur wenig verbreitet. Weltweit wird jedoch an der Entwicklung schnellerer Algorithmen gearbeitet. Die Leistungsfähigkeit dieser Systeme muss weiter verbessert werden, um einen Durchbruch auf dem Massenmarkt zu erreichen. Die Verwendung standardisierter Übertragungsprotokolle kann dabei auch die Entwicklungskosten senken. Bisher werden diese von den Anbietern jedoch kaum genutzt. Zudem sind die Codierverfahren vieler Hersteller nicht kompatibel zu denen der Konkurrenz.

1.2 Überblick

Die in dieser Diplomarbeit entwickelte und untersuchte Anwendung wurde für das beschriebene Einsatzgebiet erstellt. Sie ermöglicht den parallelen Empfang mehrerer Videoströme auf einem Standard-PC. Die Videoströme werden von einer entsprechenden Serveranwendung erstellt, welche unkomprimierte Bilddaten im RGB- und YUV-Format verarbeiten kann.

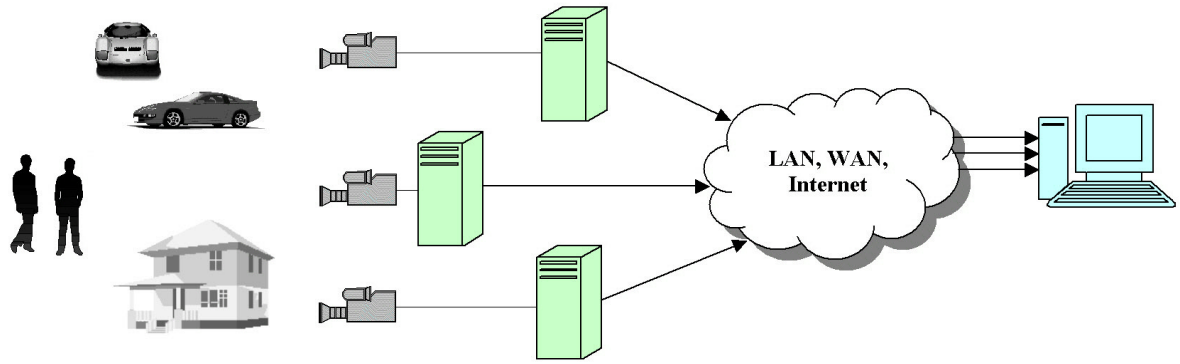


Abb. 1.1 : Parallele Übertragung digitaler Videoströme

Auf Seite des Servers ist neben der Kamera-Hardware ein System für die Verarbeitung der Bilddaten erforderlich. Dies wird durch einen PC vorgenommen. Pro Computer ist eine Firewire-Kamera vom Typ Sony DFW-V500 angeschlossen. Diese liefert Videobilder bis zu einer Auflösung von 640x480x30Hz.

Um die Bitraten der Videoströme möglichst gering zu halten, kommt der Videocodierstandard MPEG-4 zum Einsatz. Die Übertragung kann mit Hilfe der Standardprotokolle RTSP und RTP über beliebige, mit der notwendigen Bandbreite ausgestattete IP-Netze erfolgen. Diese liegen in vielen Fällen bereits vor (z.B. Firmennetze), sodass sich die Installationskosten auf Server- und Clienthardware beschränken. Aber auch wenn noch keine Netzinfrastruktur vorhanden ist, kann auf aufwendige Installation verzichtet werden. Aufgrund von Weiterentwicklungen und Preissenkungen bei wireless LAN Produkten, bietet sich unter bestimmten Bedingungen der Einsatz dieser Netzwerke an. Darum wurde im Rahmen der Diplomarbeit die Möglichkeiten der aktuell verfügbaren WLAN-Technologie bezüglich der Übertragung von Echtzeitvideodaten untersucht.

Die Empfangsplattform, auf welcher die Client-Anwendung läuft, besteht ebenfalls aus einem normalen PC mit Netzwerkanschluß. Aufgrund der wesentlich weniger aufwendigen Dekodierung genügt ein PC zum Empfang mehrerer Videoströme.
Da in vielen Fällen die Hardware für die Videoserver nur schwer zugänglich ist (Masten, Gebäudeaußenwand etc.) oder die Menge an Kameras eine Wartung vor Ort zu kostenintensiv machen würde, ist das Server-Client-System mit der Möglichkeit der Fernsteuerung ausgestattet. Diese erla ubt die Fernkonfiguration der Server durch den Empfänger. Neben der manuellen Steuerung von Bildformaten oder Bitraten ist die Clientanwendung auch in der Lage, die Gesamtbitrate des Netzes automatisch unter einem vorgegebenen Wert zu halten, um Störungen durch Netzüberlastung vorzubeugen. Da aber vor allem Funknetze sehr störanfällig sind, bietet der Client ebenfalls die Möglichkeit, Datenverluste automatisch auszuwerten und mit einer Neuregelung der Bitraten der beteiligten Videoströme zu reagieren.

1.3 Gliederung und Terminologie

Die Diplomarbeit ist in 6 Kapitel gegliedert. Kapitel 1 enthält eine Einführung zum Thema und einen Überblick über die Arbeit. Zum besseren Verständnis werden dann in Kapitel 2 zunächst die relevanten Standards und Protokolle beschrieben. Der Schwerpunkt liegt dabei auf der Vorstellung des Netzstandards 802.11b-Wireless LAN. In Kapitel 3 wird das Gesamtübertragungssystem vorgestellt. Dieses enthält auch die bereits genannte Serveranwendung.
Auf softwarespezifische Details der implementierten Module wird in Kapitel 4 eingegangen. Des Weiteren liefert dieses Kapitel eine Kurzbeschreibung der beiden Testanwendungen. Insgesamt richtet sich Kapitel 4 besonders an Softwareentwickler, welche die implementierten Module benutzen wollen.
In Kapitel 5 werden dann die Untersuchungen des Test-WLANs beschrieben und ausgewertet. Die Ergebnisse führen zu einer Erweiterung der Software zum Zwecke der besseren Anpassung der Codierung und Übertragung an die Netzwerkeigenschaften. Darin ist auch die Entwicklung eines weiteren Modules zur automatischen Steuerung der Videostreams enthalten.
Kapitel 6 enthält schließlich eine Zusammenfassung der Arbeit sowie einen Ausblick.
Während in Kapitel 5 nur ein repräsentativer Teil der Untersuchungsergebnisse vorgestellt wird, sind ausführlichere Meßergebnisse im Anhang aufgeführt. Dieser enthält zudem ein Literatur- und Abbildungsverzeichnis, eine Beschreibung der Interfacefunktionen des implementierten Client-Modules, sowie eine Beschreibung der Bedienoberfläche einer menügeführten Anwendung.

Die in der Diplomarbeit verwendeten englischsprachigen Begriffe entstammen dem Fachwortschatz der Elektrotechnik/Nachrichtentechnik und Informatik. Obwohl es zumeist deutsche Übersetzungen gibt, werden diese auch in der deutschen Fachwelt kaum benutzt. Vielmehr haben sich Begriffe wie “Frame“ oder “Streaming“ durchgesetzt, da der weltweite Informationsaustausch besonders in technischen Bereichen auf Englisch geführt wird. Darum werden englischsprachige Fachbegriffe in dieser Arbeit ohne besondere Kennzeichnung verwendet.

2 Standards und Protokolle

Die folgenden Kapitel geben einen Überblick über die Standards und Protokolle, welche bei der Implementierung der Software verwendet wurden. Nach einer kurzen Einführung in den Videoteil des MPEG-4-Standards, sowie in das Real-Time Transport Protocol RTP beschäftigt sich Kapitel 2.4. ausführlicher mit dem Standard 802.11b für Wireless LAN, da dieser lokale Netzwerktyp neben Ethernet als zweite Versuchsplattform für das entwickelte Server-Client-System diente.

2.1 MPEG4 - VideoCodec

Der für die Videoverarbeitung eingesetzte Codec folgt den Spezifikationen des MPEG-4 Advanced Simple Profile (ISO/IEC 14496-2). Er wurde am Heinrich-Hertz-Institut implementiert und unterstützt rechteckige Videoformate. Eine genaue Beschreibung ist in [15] enthalten.
Grundlage für die Videocodierung ist wie auch im Vorgänger-Standard MPEG-2 der Hybridcoder. Er nutzt die Ähnlichkeit aufeinanderfolgender Bilder aus, um die Bitrate zu senken. Bewegungen von Objekten im Vergleich zum Vorgängerbild werden geschätzt und kompensiert. Die Güte der Schätzung und Kompensation ist ein wesentliches Merkmal eines Encoders. Je besser sie ist, desto geringer ist die Energie des Differenzbildes. Dieses Bild, also der Schätzfehler, wird einer Transformation unterzogen (DCT), welche die Signalenergie in möglichst wenigen Koeffizienten bündeln soll. Die Koeffizienten werden dann quantisiert und nach einer VLC-Tabelle verlustlos codiert. Neben den mit dieser Methode berechneten prädiktiven Bildern (P-Frames) wird in bestimmten Abständen ein Bild rein intracodiert. Dieses sogenannte I-Frame wird dabei direkt der Transformation unterworfen. Da die Energie dieses Signales deutlich größer ist als beim Schätzfehlersignal, ist zur Übertragung mit gleicher Qualität eine höhere Bitrate notwendig. Der Vorteil ist aber, das sich Bildstörungen durch Übertragungsfehler nach Eintreffen eines I-Frames nicht weiter auswirken. Sinnvolle Werte für I-Frame-Raten liegen je nach Bildmaterial etwa bei ½ bis 2 Sekunden.

Die wichtigsten Features des verwendeten Codecs sind :

- Unterstützung beliebiger rechteckiger Bildformate, sofern die Seitenlängen Vielfache der Makroblockgröße (16x16 Pixel) betragen
- Kodierung von I und P–Bildern mit fester, voreingestellter Intra-Frame-Periode.
  Es werden keine B-Bilder unterstützt.
- Dekodierung beliebiger I-P-Bildfolgen
  Eine Neueinstellung der Bitrate des Servers (Neustart mit I-Frame) oder ein Verlust von Frames hat zur Folge, dass das nächste Frame nicht mit dem Erwarteten übereinstimmt. Der Decoder kann dies jedoch problemlos verarbeiten.
- Kodierung mit fester, einstellbarer Bitrate in einem weiten Bereich
  Die Bitrate wird intern optimal auf die I-P-Bildsequenz aufgeteilt. Die ursprünglichen Einstellungen für diese Aufteilung wurden nach Durchlauf einiger Testsequenzen im Sourcecode zugunsten der P-Bilder verändert, um die subjektive Qualität zu erhöhen und das Größenverhältnis zwischen I- und P-Frames zu verringern. Im Gegensatz zu Audiodaten

dauert die Übertragung von codierten Videodaten unterschiedlich lang, was zu einer ungleichmäßigen Anzeige führt.

- Freie Wahl der Framerate.
  Dabei gilt jedoch die Einschränkung, dass sie aus dem Bereich der natürlichen Zahlen kommen muss. In der Praxis ist dies jedoch kein Hindernis für die Benutzung anderer Bildraten (z.B. 12,5 Hz), sofern der Client nicht auf exakte Werte aus den übertragenen Headerinformationen angewiesen ist.
- Halbpixelgenauigkeit bei der Bewegungskompensation
  Dies stellt einen Kompromiß aus Geschwindigkeit und Qualität dar. Eine Genauigkeit auf ¼-Pixel würde zwar zu einer besseren Qualität führen, der damit verbundene höhere Rechenaufwand führt aber auch zum Verlust der benötigten Echtzeitfähigkeit auf langsameren Systemen.
- Rekursive schnelle Bewegungsschätzung (fast motion compensation)
  Der am HHI entwickelte Algorithmus verwendet die Bewegungsvektoren aus dem letzten Bild und der beiden benachbarten Makroblöcke zur groben Schätzung der Bewegung des aktuellen Makroblockes. In zwei weiteren Stufen wird die Schätzung dann noch mit Pixel- und Halbpixelgenauigkeit verfeinert. Bei dieser Suchmethode besteht natürlich die Möglichkeit, nur lokale Minima des Differenzfehlers zu finden. Es überwiegt aber der Vorteil der höheren Geschwindigkeit. Der Encoder kann somit auch auf langsamen PC-Systemen eingesetzt werden.
- Unterstützung des MMX-Befehlsatzes aktueller Prozessoren.
  Sowohl Encoder als auch Decoder benutzen für bestimmte rechenintensive Algorithmen, wie z.B. der DCT oder IDCT, den von der Intel Corporation entwickelten MMX-Befehlsatz. Mit Hilfe dieser Beschleunigung benötigt z.B. der Decoder nur ca. 3 ms auf einen 2 GHz P4 für die Bearbeitung eines CIF-Bildes oder ca. 12 ms für ein VGA-Bild. Diese Geschwindigkeit ermöglicht erst den Echtzeit-Empfang mehrerer großformatiger Videoströme auf einem PC.
- Encoder und Decoder arbeiten mit planaren YC_bC_r 4:2:0 Bildern als Eingangs- bzw. Ausgangsdaten. Der vom Encoder produzierte MPEG4-Elementardatenstrom beginnt mit dem Initial Object Descriptor, in diesem Fall der VOL-Header ([5], 6.2.3). Dieser wird bei Aufnahme der Verbindung gesondert übertragen und dient der Initialisierung des Decoders. Dem IOD folgen dann einzelne Access Units, in diesem Fall VOPs bestehend aus VOP-Header und kodierten Bilddaten ([5], 6.2.5). Soll der Stream als Datei gespeichert werden, so müssen die benötigten Header (z.B. für AVI-Datei etc.) selbst erstellt werden.

Der verwendete Encoder stellt eine Kompromiß aus Geschwindigkeit und Qualität dar. Beim vorliegenden Szenario ist trotz der schnellen Bewegungsschätzung eine hohe Qualität schon bei niedrigen Bitraten möglich, da sich oft nur wenige Objekte vor einem starren Hintergrund bewegen. Erreichbare PSNR-Werte wurden anhand einer Testsequenz vom Ernst-Reuter-Platz in Berlin in Abhängigkeit verschiedener Parameter berechnet ([18])

2.2 RTSP/RTP – Protokolle zur Videoübertragung

Die beiden zur Übertragung eingesetzten Protokolle RTSP (real-time streaming protocol) und RTP (real-time transport protocol) arbeiten auf der Anwendungsschicht des ISO/OSI-Schichtenmodells. RTSP dient dabei der Instanzierung und Steuerung des Streamings. Über eine TCP- oder auch UDP-Verbindung werden Textnachrichten zwischen Server und Client ausgetauscht. Die Syntax und Bedeutung der Nachrichten ist in RFC2326 [13] beschrieben. In Bezug auf die Steuerung der Streams bietet RTSP die gängigen Funktionen eines Videoplayers, wie "Start", "Stop", "Pause" oder "Record". Die für das Live-Streaming verwendete Steuerung beschränkt sich auf das Starten und Stoppen. Der erste Schritt vor der eigentlichen Übertragung ist die Abfrage von Medieninformationen vom Server durch den Client. RTSP unterstützt auch die Kontrolle mehrerer Tracks. So werden z.B. häufig ein Audio- und ein Videotrack benötigt.
Während RTSP für die Kontrolle der Videostreams verwendet wird, dient RTP [14] der eigentlichen Übertragung. Das Protokoll spezifiziert dabei die Struktur von einzelnen RTP-Datenpaketen. Da pro RTP-Paket nur maximal 1460 Bytes Daten als Zuladung zugelassen sind, müssen größere Dateneinheiten wie Video-Access Units zunächst zerlegt werden. Der Header eines RTP-Paketes enthält jedoch Informationen, welche ein Zusammensetzung der Pakete auf Empfängerseite ermöglichen. Jedes Paket erhält zunächst eine 16-Bit Sequenznummer. Diese Nummern beginnen mit einem zufälligen Wert und werden fortlaufend mitgezählt. Anhand dieser Nummern sind typische Netzwerk-Übertragungsfehler wie Verlust, Duplizierung oder Reordering einfach zu erkennen. RTP selbst bietet aber keine Funktionen zum Umgang mit erkannten Fehlern. Eine Sortierung vertauschter Pakete oder eine Neuanforderung verlorener Pakete muss durch andere, übergeordnete Mechanismen vorgenommen werden.
Des Weiteren enthält der RTP-Header einen 32-Bit Timestamp und ein Marker-Bit. Jede Access Unit besitzt einen eigenen Timestamp, welcher in der Regel fortlaufend größer wird. Zumindest sollte er sich bei jeder Access Unit ändern, damit der Empfänger leicht neu eingetroffene oder fehlende AUs erkennen kann. Zudem gibt das Marker-Bit an, ob das letzte Paket einer AU eingetroffen ist. Der Empfänger kann dann die Nutzdaten aller zusammengehörenden Pakete wieder vereinen und weiterverarbeiten.

2.3 TCP/IP und UDP

Zur Übertragung der Videodaten wird das einfache, schnelle Transportprotokoll UDP ([10]) eingesetzt. Der Overhead ist im Gegensatz zu TCP ([9]) geringer. Dafür wird auf Sicherungsmechanismen verzichtet. Verlorene UDP-Pakete werden z.B. nicht neu angefordert, sodass Anwendungen nicht von einem vollständigen Datenempfang ausgehen können. Zur Steuerung der Streams (RTSP, Remote Control) wird aber TCP eingesetzt, da es hier auf eine sichere Übertragung ankommt.
Den beiden Transportprotokollen untergeordnet ist das Internet-Protokoll ([1], [8]). Dieses Protokoll der Schicht 3 des ISO/OSI-Modelles ist seit seiner Einführung in den 80er Jahren zum weltweiten Standard geworden. Es verbindet zahlreiche lokale und globale Netze unterschiedlicher Topologie miteinander. Wichtigstes Merkmal ist eine 4-Byte-Adresse (IPv4) für jeden im Netz befindlichen Host. Die IP-Adresse wird von zentralen Stellen vergeben. Sie dient dem Routing von Datenpaketen innerhalb der Netze und muss daher einmalig sein.

2.4 Wireless LAN

Da im Rahmen der Diplomarbeit das Videostreamingsystem speziell auf wireless LAN getestet werden soll, wird in diesem Kapitel auf die Eigenschaften dieses Netzwerktyps eingegangen. Es wurde ein 11 MBit-WLAN von Compaq gemäß dem Standard 802.11b verwendet. Der Vorgänger 802.11 wurde 1997 von der IEEE verabschiedet und charakterisiert wie die anderen Standards der 802-Familie die beiden unteren Schichten des ISO/OSI-Modelles, also den Data Link Layer und den Physical Layer. Aufgrund des Übertragungsmediums Luft waren bei der Ausarbeitung des Standards eine Reihe neuer Probleme zu bewältigen. Im Gegensatz zu kabelbasierten Netzen gibt es mehr Möglichkeiten der Beeinflussung und Störung. Darum mußten bei den Tests eine Reihe von Einschränkungen und Vereinfachungen vorgenommen werden. Näheres dazu in den nächsten Abschnitten und im Kapitel 5.
Grundlage der heutigen WLAN-Technologie ist der 802.11–Standard, welcher Übertragungsraten bis 2 MBits/s erlaubt. Da schon bald erkannt wurde, das dies nicht für die gängigen Aufgaben eines lokalen Netzes reicht, entwickelte ein Konsortium aus Firmen WiFi (wireless fidelity), eine auf 802.11 aufbauende WLAN-Technologie mit Bandbreiten von 5,5 und 11 MBits/s. Diese dann 1999 unter der Bezeichnung 802.11b standardisierte Technologie begründete den Durchbruch von WLAN bei Firmen- und Privatkunden. Die meisten zur Zeit installierten WLANs basieren auf diesem Standard.
Komponenten, welche nach dem ebenfalls entwickelten Standard 802.11a arbeiten, stellen sogar eine Bandbreite von theoretischen 54 MBits/s zur Verfügung. Dieser Standard ist aber nicht kompatibel zu 802.11b. Lange Zeit waren die Produkte auch deutlich teurer, sodass sie sich nicht durchsetzen konnten. Trotzdem wurde der Wunsch nach noch höherer Bandbreite erkannt und im Jahr 2003 der neueste Standard 802.11g mit ebenfalls 54 MBits/s aber vorhandener Kompatibilität verabschiedet. Die ersten Produkte sind bereits auf dem Markt und sorgen für steigende Verbreitung von wireless LANs.

2.4.1 Physical Layer

Der Standard für WLAN beinhaltet die Übertragung mittels Infrarot-Licht und durch Radiowellen. Für Letztere steht der lizenzfreie Frequenzbereich zwischen 2,4 und 2,484 GHz (ISM-Band) zur Verfügung. Größere Entfernungen, eine höhere Bandbreite, sowie die Übertragung durch optische Hindernisse (Mauern etc.) sind nur mit Radiowellen möglich, sodass die darauf basierenden WLAN-Systeme immer mehr Verbreitung finden und oft Ethernet-Netze ersetzen oder zumindest ergänzen. Ein Nachteil gegenüber der leitungsgebundenen Übertragung ist allerdings die Verringerung der Feldstärke von Elektromagnetischen Wellen proportional zum Abstand (E~1/r). Die Leistungs-flußdichte S_r (Leistung pro Fläche) und damit die Empfangsleistung ist gar reziprok-proportional zum Abstandsquadrat. Weiterhin besitzt jede Antenne, deren Form vom isotropen Kugelstrahler abweicht, eine mehr oder weniger ausgeprägte Richtcharakteristik. D.h. der Großteil der Energie wird z.B. nur in eine Richtung (Beispiel siehe Abb. 2.1d) abgestrahlt.

$$P_E = P_S \left(\frac{\lambda_0}{4 \pi r} \right)^2 G_{iso1} * G_{iso2} \qquad (2.1)$$

$$\text{Gewinn einer Antenne } G_{iso} = \frac{\dfrac{\text{Leistungsdichte in Hauptstrahlrichtung}}{\text{gesamte abgestrahlte Leistung}}}{\dfrac{\text{Leistungsdichte des isotropenKugelstrahlers}}{\text{gesamte abgestrahlte Leistung des Kugelstrahlers}}} \qquad (2.2)$$

Beispiel : G_{iso} eines $\lambda/2$-Dipols = 1,64

Für zusätzliche Dämpfung sorgen Hindernisse im Ausbreitungsweg (Fresnelzone), wie Mauern, Personen oder Büroinventar. Die 1. Fresnelzone (Abb. 2.1.a) ist ein Rotationsellipsoid um zwei Antennen mit dem Abstand r und hat einen Durchmesser von $d = \sqrt{r * \lambda_0}$. Mit λ_0 = 12,5cm (2.4 GHz) und r = 6m ergibt sich z.B. d = 0,87m. Objekte in dieser Zone verringern die verfügbare Empfangsleistung, da sie selber als Antennen wirken und Energie absorbieren.
Die Empfangsleistung von WLAN-Modulen hängt daher sehr stark vom Aufstellungsort und der Ausrichtung ab. Der negative Einfluß von stationären oder temporären Hindernissen spielte z.T. auch bei der Durchführung der Messungen eine Rolle (Kapitel 5.).
Die von den Herstellern genannten Sendeentfernungen bezeichnen oft nur Maximalwerte bei optimalen Bedingungen. Da jede Umgebung jedoch andere Bedingungen bietet, ist der Aufstellungsort von WLAN-Geräten vom jeweiligen Netzadministrator sorgfältig zu planen. Einige Hersteller bieten zur Vereinfachung der Planung die Möglichkeit, Sende- und Störleistung zu messen. Die Meßwerte können mit entsprechender Software aus den WLAN-Modulen gelesen und angezeigt werden. Ist z.B. die Störleistung hoch, so kann von einem weiteren Netz in der unmittelbaren Umgebung ausgegangen werden. Zur Verhinderung von Interferenzen zwischen benachbarten Netzen müssen in diesem Fall nichtüberlappende Frequenzbereiche gewählt werden. Nur drei der 13 zur Verfügung stehenden Kanäle haben diese Eigenschaft.

Entscheidend für die Signalqualität ist letztendlich aber der Signal-Rausch-Abstand, gemessen in dB. Er gibt das Verhältnis der Signalleistung zur Störleistung an. Bei der Anzeige der Verbindungsqualität wird grob in "sehr gut", "gut" , "ausreichend" und "schlecht" unterschieden. Die zugehörigen Werte für den SNR sind dabei herstellerabhängig.

Leider bedeutet ein guter Empfang nicht immer eine hohe mögliche Übertragungsrate, da es in Gebäuden aber auch im Freien zu Signalverfälschungen durch Mehrwegeempfang kommen kann. Dabei treffen verschieden verzögerte Signalteile gleichzeitig am Empfänger ein und überlagern sich. Der Bitstrom kann unter Umständen nicht mehr korrekt demoduliert werden. In WLAN-Geräten werden deshalb sogenannte RAKE-Receiver ([1]) eingesetzt, welche sich an die Verzögerungen mehrfach empfangener Signale bis zu einer gewissen Grenze anpassen können.

Die folgende Abbildung zeigt die wichtigsten Störeinflüsse und ein Antennenbeispiel:

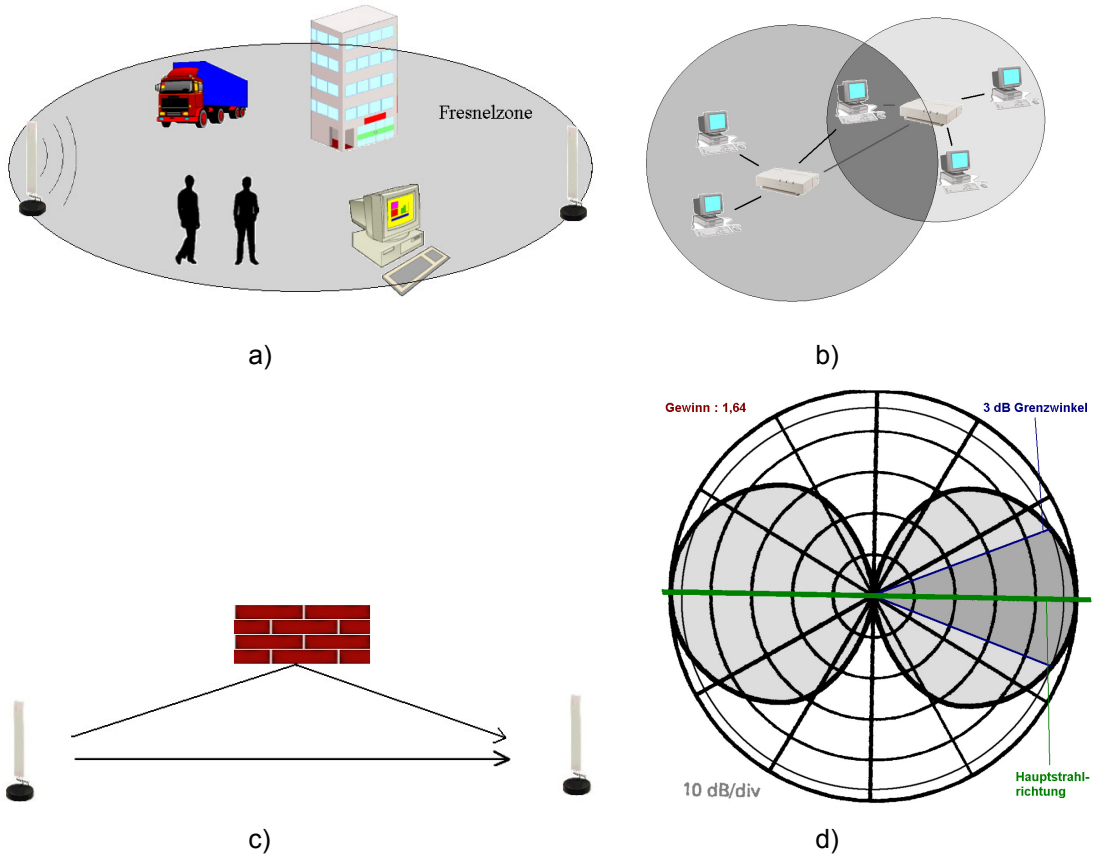


Abb. 2.1 : a) Zusätzliche Dämpfung b) Interferenzen c) Mehrwegeempfang d) Richtcharakteristik (λ/2-Dipol, Quelle [1])

Die Übertragung der Binärsignale erfolgt phasenmoduliert (phase shift keying). Im 1 MBits/s-Modus wird die Trägerfrequenz zwischen den Phasen 0 oder Π umgetastet (difference binary phase shift keying - DBPSK), im 2 MBits/s-Modus werden jeweils 2 Bits zu einem DiBit zusammengefasst und mittels differential quadrature phase shift keying (DQPSK) übertragen ([7]).

Das Problem von gewollten oder ungewollten Störungen wird durch zwei im Standard festgelegte Spreiz-Spektrum-Verfahren vermindert: Frequency Hopping (FHSS) und Direct Sequence Spread

Spectrum (DSSS). Das Prinzip ist bei beiden Verfahren gleich, die Verteilung der Sendeleistung auf ein größeres Spektrum. Beim ersten Verfahren wird die Trägerfrequenz nach einem festen Muster gewechselt. Arbeitet ein anderes Netz in der Nähe, so kommt es nur zu Teilüberlappungen der Sendungen und somit auch nur zu Teilstörungen. Auch das Abhören des Datenverkehrs wird erschwert, wenn das Frequenzmuster geheim bleibt. Beim zweiten Verfahren moduliert das binäre Datensignal ein höherfrequentes Signal, die sogenannte Chip-Sequenz, welche dann wiederum die Trägerfrequenz moduliert. Auch für dieses System gelten die genannten Vorteile. In den kommerziell genutzten Systemen wird jedoch eine bekannte Sequenz (11-Chip-Barker-Sequenz : +1,-1,+1,+1,-1,+1,+1,+1,-1,-1,-1) benutzt, damit WLAN-Geräte verschiedener Hersteller zusammenarbeiten können. Diese Chip-Sequenz ist annähernd gleichanteilsfrei und gut geeignet für Korrelationsanalysen zur Verminderung der Fehleranfälligkeit durch Mehrwegeempfang. Die Chip-Rate ist 11 MChips/s, die Symbolrate (1 Bit oder 1 Di-Bit) ist 1 MS/s. Somit ergeben sich Datenraten von 1 MBits/s und 2 MBits/s bei gleicher Chiprate. Bei Differenz-BPSK und -QPSK wird die Chip-Sequenz nicht geändert, wenn eine “1“ übertragen wird, sie wird invertiert bei einer “0“ :

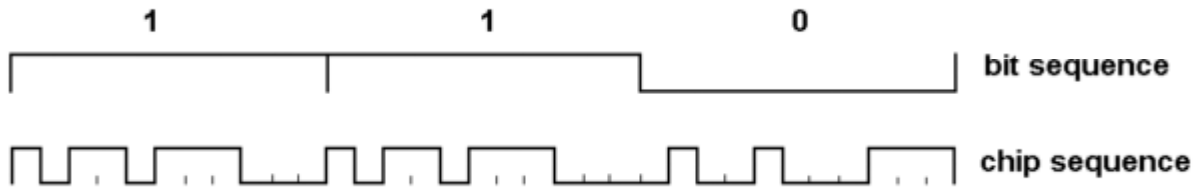


Abb. 2.2 : DSSS mit 11-Chip-Barker-Code (1 MBits/s-Modus)

2.4.2 MAC-Layer

Die MAC-Schicht ist ein Teil von Schicht 2 des ISO/OSI-Modelles und beschreibt Zugriffsmechanismen auf den Übertragungskanal. Dies ist von Bedeutung, wenn mehrere Sender Zugriff auf den gleichen Kanal haben. Dieser Zugriff muss natürlich geregelt werden. Die zur Verfügung stehende Bandbreite soll dabei optimal und fair ausgenutzt werden. Bei Ethernet erfolgt die Steuerung mittels CSMA/CD, ein Protokoll aus der ALOHA-Familie. CD steht für “collision detection“. Kabelgebundene Systeme haben den Vorteil, dass auch während der Übertragung eine Überlagerung mit einem zweiten Signal sofort erkannt und die Übertragung abgebrochen werden kann. Für WLAN ist diese Lösung aber nicht praktikabel, da dazu ein zweites Modul zum Abhören des Kanals während des Sendens benötigt wird und zudem die Detektion einer Erhöhung der Signalleistung aufgrund der großen Dynamik (> 50 dB) nur schwer möglich ist. Darum beschränkt man sich auf Kollisionsvermeidung (CSMA/CA).
Der für wireless LANs entwickelte Mechanismus heißt DCF (distributed coordination function). Er ist die Grundlage der auf dem Markt vorhandenen Geräte, d.h. jedes Gerät muss DCF unterstützen. Wie die Bezeichnung schon ausdrückt, wird der Zugriff dezentral geregelt. Dies funktioniert sowohl für WLANs mit zentralem Access Point (infrastructured LAN) als auch für Adhoc-Netze. Details zur DCF

findet man in zahlreicher Literatur zum Thema Kommunikationsnetze ([12]). Darum sollen im Folgenden nur einige wesentliche Punkte genannt werden:
Jede sendebereite Station hört den Kanal vorher ab. Dies kann durch direkte Detektion des Signales oder durch Überprüfen des Empfangspuffers geschehen. Wenn der Kanal bereits belegt ist, zieht sich die Station für eine bestimmte, zufällige Zeit zurück. Das Intervall für die Bestimmung dieser Zeit variiert, hat aber auch eine Obergrenze. Wenn der Kanal hingegen frei ist, beginnt die Station nach einer weiteren kurzen Wartezeit mit dem Senden.
Beim Auslesen des Empfangspuffers kann die Station den Header des fremden Paketes auswerten. Im Header der MAC-Datenpakete (MPDU) ist die Länge, bzw. Übertragungsdauer des Paketes enthalten, sodass jede Station die Länge der weiteren Wartedauer bestimmen kann. Zur Übertragungsdauer zählen auch Pausenzeiten und die Dauer der Quittung vom Empfänger an den Sender. Erst nach Abwarten der fremden Übertragung prüft die Station erneut den Kanal.
Die DCF kann ergänzt werden mit RTS/CTS-Handshaking. Dies ist besonders dann sinnvoll, wenn nicht jede Station in Reichweite aller anderen Stationen ist. In einem solchen Fall kommt es zu einem deutlichen Performanceeinbruch durch zahlreiche Kollisionen, da die Stationen nicht erkennen können, ob eine andere gerade sendet. Bei RTS/CTS erfolgt vor dem Senden der MPDU daher zuerst eine Anfrage (“request to send“), welche vom Empfänger mit einem “clear to send“ beantwortet wird. Die CTS-Pakete werden auch von den anderen Stationen empfangen. Sie enthalten Informationen über die Dauer des folgenden Paketes. Durch den zusätzlichen Overhead bei RTS/CTS sinkt die Nettobandbreite des WLANs jedoch. Allerdings werden Kollisionen der langen Datenpakete vermindert und somit die Bandbreite effektiver ausgenutzt. Kollisionen der recht kurzen RTS/CTS-Pakete verursachen weniger Zeitverluste.

Die DCF funktioniert für normalen Datenverkehr (TCP) und nicht zu vielen beteiligten Stationen recht gut. Sie bereitet aber gerade bei Echtzeit-Daten einige Probleme. Die Zusicherung von Bandbreite und maximalen Verzögerungszeiten für einzelne Stationen ist nicht möglich. Kurze Delays verbunden mit minimalem Pufferaufwand ist aber für viele Echtzeitanwendungen von großer Bedeutung. Ebenfalls nicht möglich ist die Zusicherung einer bestimmten Paketfehlerrate. Diese ist nicht nur abhängig von der Signal- und Störleistung (-> BER) sondern auch von der Kollisionswahrscheinlichkeit. Alle drei Werte können großen Schwankungen unterworfen sein. Anwendungen, deren Daten mit dem unsicheren Transportprotokoll UDP übertragen werden, müssen sich auf möglicherweise große Paketverluste einstellen.
Um WLAN auch für Echtzeit-Audio- und Videostreaming interessant zu machen, ist im Standard eine weitere Zugriffssteuerung enthalten. Bei der PCF (point coordination function) regelt eine Station zentral den Zugriff. Sie weist allen anderen Stationen Übertragungszeiten zu. Diese Funktion wird sinnvollerweise in einem Access Point integriert, weshalb die PCF nicht für Adhoc-Netze geeignet ist. Gegenüber dem Vorteil der zugesicherten Bandbreite steht der Nachteil des größeren Signalisierungs-Overheads. Bei hoher Kanalauslastung ist die PCF aber effektiver als die DCF ([6]).
Obwohl im Standard vorgesehen, wird die zentrale Steuerung durch PCF nicht von jedem Hersteller implementiert, da die teureren Bauteile im harten Wettbewerb Nachteile bedeuten. Die meisten Geräte

bieten jedoch RTS/CTS-Signalisierun. Der Benutzer kann zum Beispiel per Treiberzugriff zwischen Deaktivierung, voller und bedingter Aktivierung wählen. Bei letzterer wird die Paketgröße vorgegeben, ab der RTS/CTS verwendet werden soll. Das Handshaking ist bei kleinen Paketen eher hinderlich als nützlich, da die zusätzliche Signalisierung Bandbreite kostet, obwohl die Kollisionswahrscheinlichkeit kleiner Pakete nur gering ist.
Nach der Übertragung eines Paketes quittiert der Empfänger den korrekten Empfang mit einer kurzen Bestätigung. Trifft diese nicht nach der berechneten Zeit beim Sender ein, so liegt ein Übertragungsfehler vor. In diesem Fall wird das unquittierte Paket zumeist noch einmal übertragen. Die Anzahl der Wiederholungen kann bei einigen Geräten vom Anwender vorgegeben werden. Bei anderen ist sie jedoch fest eingestellt. Zusätzlich wird eine Auto-Fallback-Lösung angeboten. Dabei wird bei ausbleibender Bestätigung von einer schlechten Signalleistung bzw. einem kleinen SNR ausgegangen und das Paket in einem anderen Modus mit geringerer Datenrate wiederholt. Leider berücksichtigt dieser Mechanismus nicht die Verluste durch Kollisionen. In solchen Fällen mit wäre es sinnvoller, die Pakete mit höchstmöglicher Datenrate zu wiederholen.
Bestätigungen, festgelegte Wartezeiten, Wiederholungen, Fallback-Mechanismen, MAC-Header etc. sorgen dafür, dass die Netto-Übertragungsrate mit ca. 40-60% deutlich unter der angegebenen Netzbandbreite liegt. Liegen einzelne Stationen so weit voneinander entfernt, dass sie sich nicht gegenseitig hören können (hidden terminals), so ist die Performance des Netzes noch deutlich schlechter.

2.4.3 Weiterentwicklung zum 802.11b-Standard

In der Verabschiedung des 802.11b Standards 1999 gab es zwei wichtige Neuerungen, um zu höheren Bandbreiten zu gelangen. Zunächst wird für die Bandspreizung nur noch DSSS und als Modulationsverfahren nur DQPSK benutzt. Der zur Modulation der Frequenz benutzte Chip-Code wird nun aber auch zur Codierung selbst verwendet. Statt des festen 11-Chip-Barker-Codes wird ein Set aus 64 komplexen (vierwertigen) 8-Chip-Codes zur Codierung benutzt. Dieses Set wurde aus 4^8 möglichen Codewörtern unter der Maßgabe ausgewählt, dass diese Codes näherungsweise orthogonal sind. Mit den 64 Codewörtern können 6 von 8 Bits moduliert werden, die übrigen 2 modulieren mittels DQPSK das gesamte Codewort. Pro Symbol werden also 8 Bit Daten übertragen. Die Symbolrate wurde auf 1,375 MS/s angehoben. Daraus ergibt sich eine Bandbreite von theoretisch 11 Mbits/s.
Das Verfahren wird als Complementary Code Keying – CCK bezeichnet. Neben der Datenrate von 11 Mbits/s gibt es auch einen Modus mit 5,5 Mbits/s. In diesem Modus wird zur Modulation von 2 Bits ein Set von 4 komplementären Codewörtern verwendet. Zwei weitere Bits modulieren diese wiederum mittels DQPSK wodurch eine Datenrate von 1,375 x 4 Mbits/s erreicht wird.
In Gegensatz zu 802.11 ist die Bitrate gleich der Chiprate. Die Zusammenarbeit zwischen älteren und neuen Systemen wird aber durch die Beibehaltung der Chiprate (11 MChips/s) vereinfacht.

3 Softwarekonzept

3.1 Zielplattform und Entwicklungsumgebung

Die implementierte Software wurde für 32-Bit Windows-Systeme entwickelt. Getestet wurde sie unter Windows 2000 und Windows XP. Ein Einsatz unter Linux ist nicht möglich. Zur Erstellung wurde Microsoft Visual C++ in der Version 6.0 benutzt. Neben der direkten Benutzung der Win32-API wurde von der Microsoft-Bibliothek MFC, sowie der am Heinrich-Hertz-Institut entwickelten Klassen-Bibliothek MSysLib Gebrauch gemacht. Aus der MFC wurden speziell Fenster- und Dialogklassen verwendet, aus der MSysLib insbesondere Thread- und Socketklassen.
Ein großer Vorteil des verwendeten Betriebssystemes ist die Unterstützung von Multitasking und Multithreading. Ist ein Programm in mehrere Threads aufgeteilt, so wird diesen vom Betriebssystem abwechselnd Prozessorzeit zugewiesen. Die Threads laufen quasi parallel. Der typische Fall ist die Unterteilung einer Anwendung in einen sogenannten User-Interface- und einen Workerthread. Der erste übernimmt die Interaktion mit dem Benutzer. Der zweite kann die gewünschten Berechnungen ausführen, ohne die Benutzerinteraktion zu blockieren. Bei Multiprozessorsystemen oder dem neuen Pentium 4 mit Hyperthreading-Technologie ist damit ein Performancegewinn möglich, da die Threads vom Betriebssystem auf beide logischen oder realen Prozessoren verteilt werden.
Die entwickelte Software läuft jedoch auch auf Single-Prozessor-Systemen. Hier kann sie allerdings nicht vom Multithreaddesign profitieren.

3.2 Server-Client-System

Wie in Abb. 3.1 zu sehen ist, handelt es sich aus Sicht der Übertragung um ein Multi-Server-Single-Client System. Von einer bereits vorhandenen Serveranwendung ([18]) wird auf Anforderung des Clients ein Videostream versendet. Da der Client mehrere Streams gleichzeitig empfangen kann, wird er nachfolgend als Multi-Videoclient (abgekürzt MVC) bezeichnet. Auf seine Komponenten wird in Kapitel 4. genauer eingegangen.

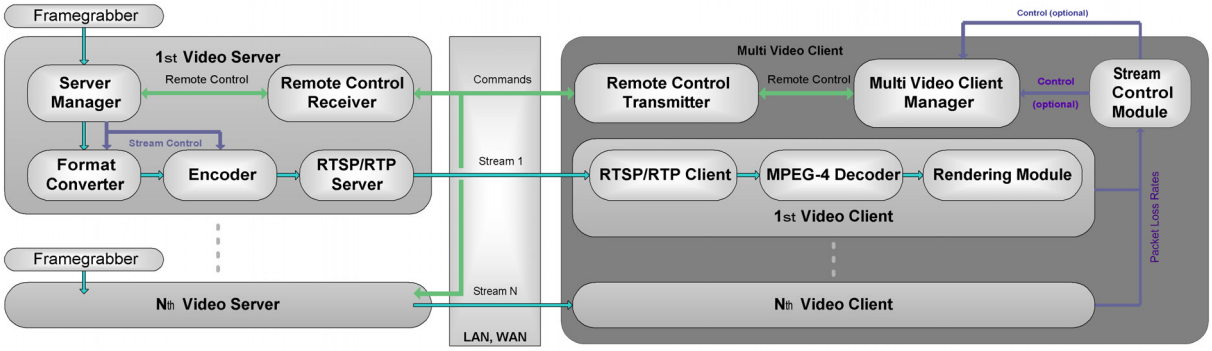


Abb. 3.1 : Surveillance Mode für Videoüberwachung

Abb. 3.1 zeigt den Standardmodus (surveillance mode) des Multi-Videoclients. Wird er zur rein visuellen Überwachung verwendet, so sind alle internen Module sowie die Videofenster aktiv. Die Bilder werden entweder in interne oder extern erzeugte Fenster gerendert.

Wieviele Streams der Client verarbeiten kann, hängt von der verwendeten Hardware und den Videoformaten ab. Neueste PCs sind in der Lage, mehrere Streams in VGA- oder TV-Auflösung zu verarbeiten.
Die dargestellte Serveranwendung ist das Gegenstück zum MVC. Sie enthält Module zum Erzeugen und Versenden eines Videostreams unter Verwendung der beschriebenen Protokolle und Standards, sowie ein Modul zum Empfang der Steuerkommandos. Der Server kann verschiedenartiges Bildmaterial automatisch verarbeiten. Die Aquisition der Bilddaten ist funktionell ausgegliedert. Der Framegrabber ist somit vom Anwender bereitzustellen. Bisher ist der Server nur in der Lage, einen Stream pro Encoder zu versenden. Denkbar ist aber auch ein Multicast-Betrieb. Nach Änderungen am RTSP/RTP-Server wäre es dann z.B. möglich, eine Kreuzung von mehreren Leitstellen aus zu überwachen oder neben der Überwachung den Live-Videostream auch Serviceanbietern für Verkehrsnachrichten (TV, Internet) bereitzustellen.

Die folgende Abbildung zeigt den zweiten Modus (“streaming mode“) :

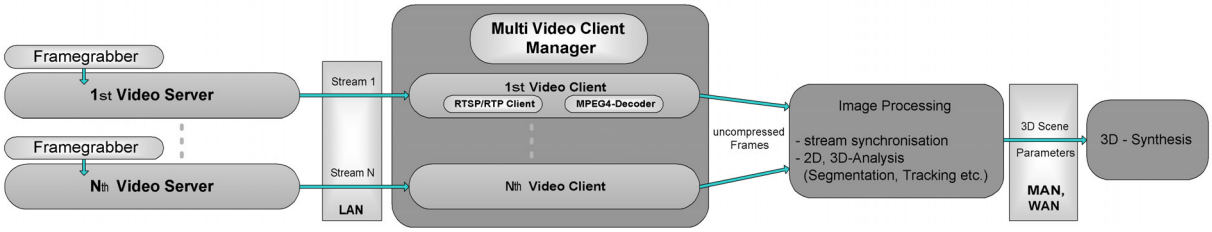


Abb. 3.2 : Streaming Mode für Empfang und Bearbeitung

In diesem Modus wird der MVC nur zum Empfang und Dekodieren der Videoströme verwendet. Alle nicht dafür benötigten Module sind deaktiviert, bzw. in ihrer Funktion eingeschränkt. Anstatt die dekodierten Bilder in einem Fenster anzuzeigen, wird einer externen Anwendung eine Schnittstelle zum Empfang der Bilder freigegeben. Die Anwendung kann die Bilder vom Typ YC_bC_r 4:2:0 dann für eigene Zwecke verwenden. Im gezeigten Beispiel werden anhand der Bilddaten von einer Kreuzung statistische Verkehrsdaten gewonnen und eine künstliche 3D-Szene generiert. Eine weitere Anwendungsmöglichkeit innerhalb eines Überwachungsszenarios wäre die Erkennung und Verfolgung bestimmter Personen im Bild oder der Einsatz von einfachen Algorithmen zur Bewegungsdetektion.
Zur Synchronisation der Streams und zur Erfassung verlorener Bilder kann dabei der vom Server gelieferte Timestamp verwendet werden. Eine interne Synchronisation erfolgt nicht. Da es aufgrund von Netzwerkproblemen zu Verspätungen oder Verlust von Paketen kommen kann, sind größere Verzögerungen von einer Sekunde und mehr möglich. Das externe Modul zur Weiterverarbeitung muss also einen geeigneten Mechanismus zur Pufferung und Synchronisation mehrerer Streams bereitstellen.
Der jeweils benötigte Modus wird beim Initialisieren des MVC vom Benutzer festgelegt und ist dann bis zum Beenden des MVC-Modules gültig. Alle in dieser Zeit gestarteten Streams werden gemäß den aktuellen Moduseinstellungen behandelt.

4 Implementierung

In den folgenden Kapiteln wird auf den internen Aufbau des MVCs eingegangen. Außerdem werden zwei Anwendungen, eine Test- sowie eine ausgereifte Anwendung mit grafischer Bedienoberfläche vorgestellt. Alle Interfacefunktionen sind in Anhang C ausführlich beschrieben.

4.1 Das Hauptmodul – Der MVC-Manager

Der MVC-Manager ist die Zentrale des MVC. Zu seinen Aufgaben gehört die Initialisierung des RC- und des Stream-Kontrollmodules, die Ausführung der Funktionen des Benutzerinterfaces, das Absetzen von Nachrichten an das Remote Control Modul und die Steuerung und Verwaltung der einzelnen Videoclients. Der MVC-Manager wird aus diesem Grund als erstes gestartet und zuletzt geschlossen.
Zur Verwaltung der einzelnen Streams werden alle relevanten Informationen, wie z.B. Server-IP oder Servername in einer Tabelle gespeichert. Die Videoclients sind zwecks eindeutiger Unterscheidung mit einer 32-Bit ID gekennzeichnet, welche vom Benutzer vorgegeben wird. Bei allen externen und internen Aktionen wird diese Nummer angegeben.
Die Funktionen des MVC-Managers sind in der Klasse “MultiVideoClientMng“ gekapselt. Theoretisch können mehrere Instanzen dieser Klasse erzeugt, also mehrere Manager erstellt werden. Das von der Bibliothek bereitgestellte Interface benutzt nur eine Instanz.
Abbildung 4.1. zeigt die zentrale Rolle des MVC-Managers:

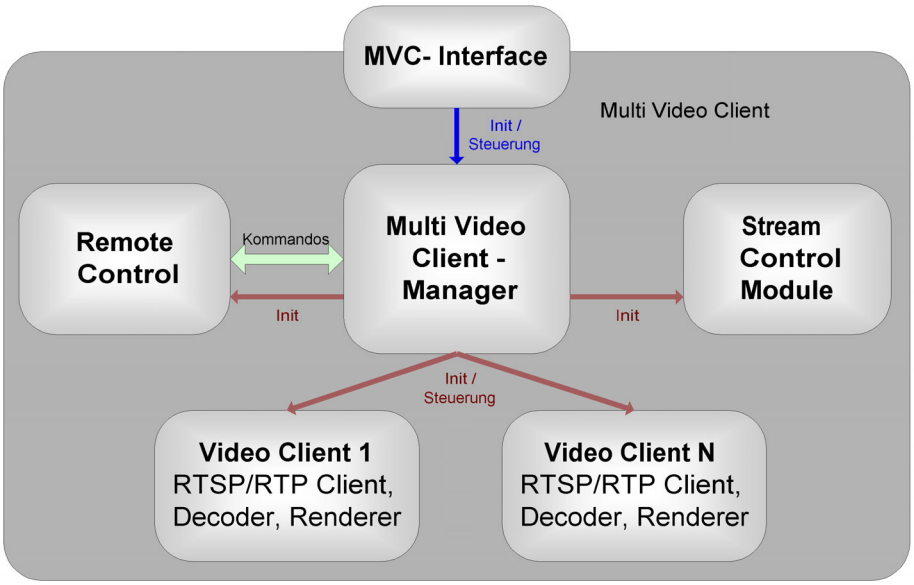


Abb. 4.1 : MVC-Manager

4.2 Remote Control

Die Fernsteuerung beinhaltet Funktionen zum Absetzen von Nachrichten an einen vorgegebenen Videoserver sowie zum Empfang und zur Auswertung der vom Server gesendeten Bestätigung. Eine Nachricht besteht aus einem Kommando und zwei 32-Bit Parametern. Diese drei Komponenten werden gemäß der festgelegten Syntax zu einem String zusammengesetzt, ggf. verschlüsselt und

dann versendet. Die Bedeutung der zwei Parameter ist abhängig vom Kommando. So enthält z.B. der erste Parameter für das Kommando "SETACTUALBITRATE" die gewünschte Bitrate in [kbits/s].
Für die Verbindung zum Server muss die Server-IP und der Port bekannt sein, an welchem der Server ständig auf die Verbindungaufnahme wartet. Das benutzte Übertragungsprotokoll ist TCP. Nach dem Verbindungsaufbau und dem Versenden der Nachricht wartet der Videoclient eine gewisse, vom Benutzer einstellbare Zeit auf die Antwort vom Server. Diese Antwort folgt der gleichen Syntax wie die Nachricht. Sie enthält also das vorher gesendete Kommando und zwei Parameter, von denen der erste "1" oder "0" ist. Diese beiden Zustände geben Auskunft, ob die Nachricht vom Server erfolgreich verarbeitet wurde oder nicht. Der zweite Parameter enthält in einigen Fällen weitere Informationen. So wird er z.B. benutzt, um vom Server abgefragte Daten zu übermitteln.
Der Verbindungsaufbau erfolgt vor der Versendung des ersten Kommandos. Die Verbindung bleibt bis zur Beendigung des RC-Modules oder des Videoservers bestehen. Wie lange der Client auf die Antwort vom Server warten soll, ist natürlich abhängig von der Netzlast und den gewünschten Aktionen des Servers. In vielen Fällen ist eine Dauer von unter einer Sekunde vollkommen ausreichend. Kommt es wiederholt zu Timeout-Fehlern, so sollte dieser Wert erhöht werden.
Damit der Austausch von Nachrichten mit den Servern nicht andere Programmabläufe blockiert, arbeitet das Modul in einem separaten Thread. Soll eine Nachricht versendet werden, so wird diese gespeichert, ein Flag gesetzt und die aufrufende Funktion kann sofort zurückkehren. Ist die Antwort eingetroffen, so wird diese ebenfalls gespeichert. Das Modul (MVC-Manager), welches auf die Antwort wartet, muss also in gewissen Abständen nachschauen (Polling).
Es kann immer nur eine Nachricht verarbeitet werden. Es gibt keine Warteschlange, in der mehrere Nachrichten eingereiht werden können. Dies ist einfacher zu handhaben und auch völlig ausreichend, da die Steuerung von Servern oder das Abfragen von Serverinformationen nur gelegentlich notwendig wird. Zudem ist bei Einsatz einer Queue die Bearbeitungsdauer einer neuen Nachricht von der Anzahl bereits wartender Nachrichten abhängig und kann somit schwerer vorherbestimmt werden. Wie aus Abb. 4.2 zu ersehen ist, können Nachrichten aber aus zwei Richtungen initiiert werden, von der externen Anwendung bzw. dem Benutzer und vom Stream-Kontrollmodul. Da es dadurch zu Überschneidungen kommen kann, müssen beide Parteien Strategien zur Kollisionsvermeidung verwenden.

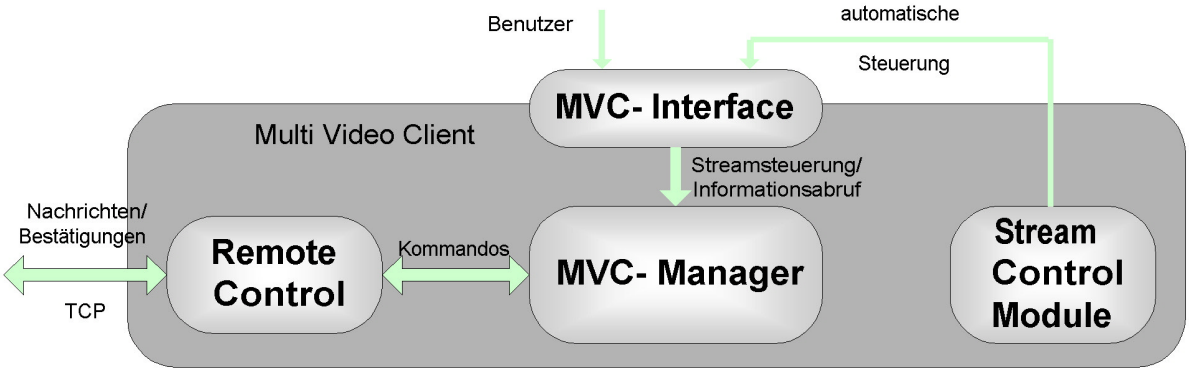


Abb. 4.2 : manuelle und automatische Fernsteuerung der Videoserver

Um die Fernsteuerung ggf. in anderen Anwendungen wiederverwenden zu können, wurde sie in einer Klasse gekapselt und in der statischen Bibliothek "CommandModuleLibStatic.lib"ausgelagert.

4.3 Videoclient

Zur Verarbeitung eines Streams wird je ein RTSP/RTP-Client, Decoder und Renderer benötigt. Diese drei Module sind funktionell zu einem Single Video Client (SVC) zusammengefasst. Alle Informationen über die drei Module und den Stream werden vom MVC-Manager verwaltet. Ein SVC wird immer als Ganzes betrachtet, seine Module werden komplett gestartet und beendet. Dies vereinfacht die Implementierung, da das An- und Abmelden einzelner Module bei ihren Nachbarn entfällt. Zudem betreffen die meisten Ereignisse wie z.B. das Ändern eines Videoformates den gesamten SVC.
Die nachfolgend beschriebenen Module sind über Interface-Klassen miteinander verbunden. Diese enthalten die Funktionen zur Datenübergabe.

4.3.1 RTSP/RTP Client

Der RTSP/RTP-Client übernimmt zunächst den Aufbau und die Steuerung der Streaming-Session. Dazu wird das Signalisierungsprotokoll RTSP verwendet. Der erste Schritt ist der Aufbau einer TCP-Verbindung an einem festgelegten Port des Videoservers. Ist der Port nicht bekannt, so kann er über den RC-Kanal abgefragt werden. Die Abfrage und das Einstellen des Ports werden vom Benutzer vor bzw. bei der Initialisierung vorgenommen. Nach Öffnen der Verbindung werden diverse RTSP-Nachrichten zum Setup und Starten des Streams zwischen Client und Server ausgetauscht. Inhalt und Form dieser Nachrichten folgen dem RTSP-Standard ([13]) und dem Session Description Protocol SDP ([2]). Die TCP-Verbindung bleibt während der ganzen Session bestehen. Da jedoch nach dem Starten des Streams oft kein weiterer Bedarf an Nachrichtenaustausch besteht, wird in Abständen von ca. 30 Sekunden eine Nachricht (“OPTIONS“-Methode) versendet, um die Verbindung offen zu halten.
Wichtigster Schritt vor dem Starten des Streams ist die Vorbereitung zweier weiterer Sockets. Es werden zwei benachbarte Ports für eine TCP- und eine UDP-Verbindung bereitgestellt. Der TCP-Kanal dient normalerweise dem Austausch von Daten mit Hilfe des Real-Time Control Protocols RTCP. Dieses Protokoll wird aber in der bestehenden RTP-Client-Implementierung nicht verwendet. Am zweiten Socket werden die RTP-Pakete empfangen. Er wird als nichtblockierender Socket und mit einer Empfangspuffergröße von 65 KByte initialisiert. Der Datenfluß ist unidirektional.
Zur Implementierung des RTSP/RTP-Clients wurde die RTP-Bibliothek von Lucent sowie eine am HHI entwickelte RTSP-Bibliothek verwendet. Insbesondere Letztere wurde jedoch gründlich überarbeitet, da sie nicht den Anforderungen entsprach :

- Die Funktionen der RTSP-Lib wurden in Klassen gekapselt. Das Interface wird durch die Klasse RTSPSimpleClient gestellt. Gemeinsame Funktionen für RTSP-Server und Client wurden in einer weiteren Bibliothek ausgelagert. Dazu zählt u.a. das Interface zur RTP-Lib. Die Abläufe der RTSP-Signalisierung und der RTP-Datenempfang bleiben vor dem Benutzer verborgen.
- Die RTSP/RTP-Funktionen (speziell der Datenempfang) laufen nun in einem separaten Thread. Für jeden Client wird also ein neuer Thread angelegt. Die Bibliotheken wurden darum auch in Hinblick auf Thread-Sicherheit überarbeitet. Obwohl die Performance durch mehrere

Threads nicht unbedingt steigt, so ist die Verwendung von einem Thread pro Client sinnvoll, da Störungen, Verzögerungen oder Blockaden in einem Stream nicht die anderen Streams beeinflussen. Z.B. kann die RTSP-Signalisierung über TCP theoretisch beliebig lange dauern. Netzprobleme, langsame oder nicht erreichbare Videoserver verzögern den Aufbau einer Session. Dies darf in einer Überwachungsanwendung nicht zu Störungen der anderen Bilddaten führen. Insbesondere das Überlaufen der Empfangspuffer, wenn die Pakete gerade nicht abgeholt werden können, führt zum Verlust von Paketen und damit möglicherweise zum Verlust des Frames.

- Der RTP-Server wurde ebenfalls überarbeitet. Zu den wichtigsten Neuerungen zählt die paketweise Pufferung der Streams. In der alten Version wurde mehrere Frames gepuffert, was einen verschwenderischen Speicherplatzbedarf zu Folge hatte da die kleineren P-Frames nur einen Bruchteil eines Framebuffers füllten. Weiterhin mißt der Server nun die Bitrate und kann optional eine Kurzzeitbitraten-Steuerung (Kurzzeit-Loadbalancing) aktivieren. Dabei werden die im Allgemeinen überdurchschnittlich großen I-Frames stückweise versendet. Der Vorteil ist der gleichmäßigere Empfang der Daten bei höheren Netzbandbreiten. Große I-Frames können bei schnellen Netzen u.U. den Empfangspuffer des Clients überfluten (Größe 65 KByte), was Paketverluste zur Folge hat. Dies wurde beim Streaming von 3 Streams größer 2 MBits/s über ein 100 Mbit-switched LAN beobachtet, wobei die Prozessorauslastung des Client-PCs nur mittelgroß war. Nachteil ist die zusätzliche künstliche Verzögerung von möglicherweise ½-1 Sekunde.

Da sich im Verlauf der WLAN-Tests (Kap.5) zeigte, das die Fehleranfälligkeit von WLAN unakzeptabel hoch ist, wurde ein System zur wiederholten Paketübertragung implementiert. Dazu war es notwendig, die gesamte Datenpufferung beim Server und Client zu überarbeiten. Ursprünglich existierte beim Client ein 8fach Paketpuffer, welcher nur Reordering kompensieren konnte und ein Framebuffer zum Zusammensetzen der Pakete. Abb. 4.3 zeigt das neue System :

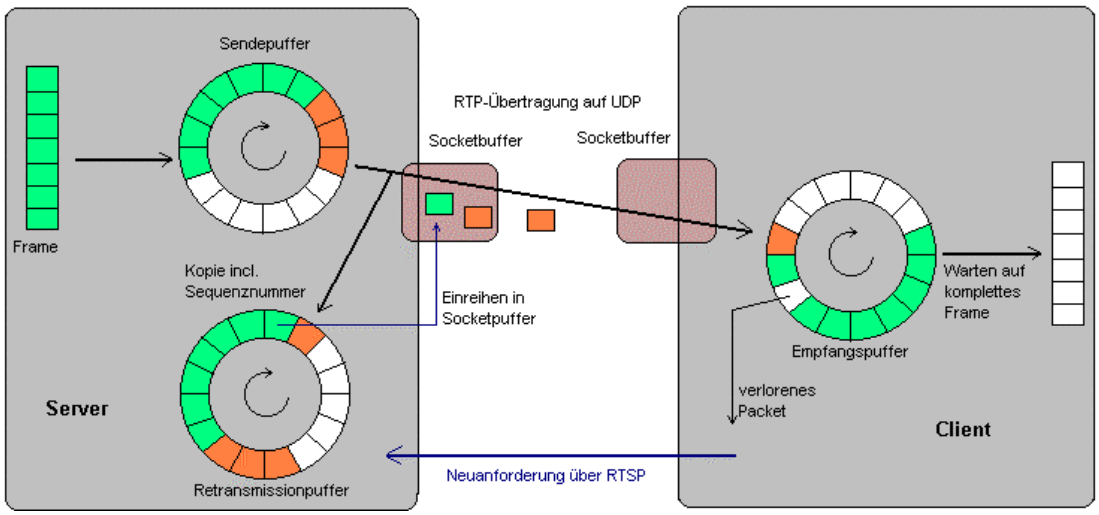


Abb. 4.3 : Paketwiederholung bei gestörter Übertragung

Wie in der Abbildung zu sehen, wird das vom Encoder kommende Frame in Pakete zerlegt und in die nächsten freien RTP-Sendepuffer kopiert. Die Puffer besitzen eine Größe von 1460 Bytes, was dem maximalen Payload nach dem RTP-Standard entspricht. Sind nicht genügend Puffer frei, so gehen Frames verloren. Dies hat größere Auswirkungen auf die empfängerseitige Anzeige als bei Paketverlusten auf dem Netz. Zunächst werden natürlich nachfolgende P-Frames gestört. Darüber hinaus kann der Verlust eines ganzen Frames nicht durch Sequenznummernanalyse beim RTP-Client erkannt werden. Dies kann nur durch Auswertung geeigneter Timestamps beim Decoder (evtl. auch beim RTP-Client) erfolgen. D.h. die Timestamps im RTP-Paketheader müssen von außen, also z.B. beim Encoder erzeugt und zum Decoder durchgereicht werden. Der implementierte RTP-Client nutzt die Timestamps nur, um den Beginn eines neuen Frames festzustellen.
Vom Sendepuffer aus gelangen die Pakete in den Socketpuffer (Socketfunktion der Win32-API). Außerdem wird eine Kopie der Pakete samt RTP-Sequenznummer in einen weiteren Puffer geladen und verbleibt dort solange, bis sie von späteren Paketen überschrieben wird. Die Dauer des Aufenthaltes der Kopie hängt von der Größe des Retransmission-Puffers und der Bitrate ab. Sie beträgt aber mindestens 1-2 Sekunden. Wird vom Client eine Anforderung zur Wiederholung gesendet, so wird das Paket mit der gewünschten Sequenznummer herausgesucht und in den Socketpuffer eingereiht. Die Wartezeit des Clients hängt also auch vom Füllstand des Socketpuffers ab.
Die Anforderung von Paketwiederholungen erfolgt über eine neue, nicht im Standard definierte RTSP-Nachricht (“RETRANSMISSION“). Diese enthält als Parameter die Sequenznummer des ersten und letzten vermißten Paketes und ist somit eine kumulative negative Bestätigung, nachfolgend CNACK genannt. CNACKs werden vom Client sofort gesendet, wenn die empfangene Sequenznummer größer als die erwartete ist, wobei die Beschränkung der Nummern auf 16 Bit beachtet wird. Das neue Paket wird dann in den Empfangspuffer eingereiht. Für vermißte Pakete werden Puffer freigelassen, wobei diese mit der kalkulierten Sequenznummer und einem Flag markiert werden. Kommen die zugehörigen Pakete beim zweiten oder einem späteren Mal an, so können sie an der freien Stelle einsortiert werden.
Parallel zum Empfang der Pakete überprüft der Client ständig, ob alle Pakete des nächsten Frames da sind. Das letzte Paket wird anhand eines Markerbits im RTP-Header erkannt. Zusätzlich wird nach dem VOP-Startcode gesucht, um das erste Paket eines Frames zu detektieren. Sind alle Pakete zusammen, so werden sie in einen Framepuffer kopiert und samt Nebeninformation (Timestamp, Bildgröße) an den Decoder übergeben. Vermißte Pakete verbleiben solange im Puffer, bis ein vorgegebener Timeoutwert T_{total} abgelaufen ist. Dann werden sie und alle zum selben Frame gehörenden Pakete gelöscht. Die maximale Wartezeit wird vom Benutzer vorgegeben. Er gibt auch die Round-Trip-Time RTT für eine Übertragungswiederholung vor. Dadurch wird gleichzeitig die Anzahl an Wiederholungen festgelegt : $n = T_{total} / RTT$. Ist RTT also kleiner als T_{total} , so wiederholt der Client sein NACK nach Ablauf der RTT, gemessen ab dem letzten (C)NACK. Der dadurch möglicherweise auftretende Mehrfachempfang eines Paketes stellt jedoch kein Problem dar. Die Größe des Empfangspuffers beträgt 1000 Pakete. Somit können z.B. bei einer Bitrate von 2000 kbits/s ca. 6 Sekunden gespeichert werden. Dies ist mehr als ausreichend, da die maximale Wartezeit

zumeist deutlich kleiner eingestellt wird (≤1 sec). Der Beobachter will in der Regel das sehen, was aktuell beim überwachten Objekt geschieht.
Für Benutzer, welche die Netzperformance und die Geschwindigkeit der Server-/Client-Software testen wollen, wurde ein einfacher Paketverlust-Simulator implementiert. Dieser verwirft eintreffende Pakete nach einem binären Markov-Modell mit den beiden Zuständen “Verlust“<->“Empfang“.
Abb. 4.4 zeigt das Modell mit den Wahrscheinlichkeiten :

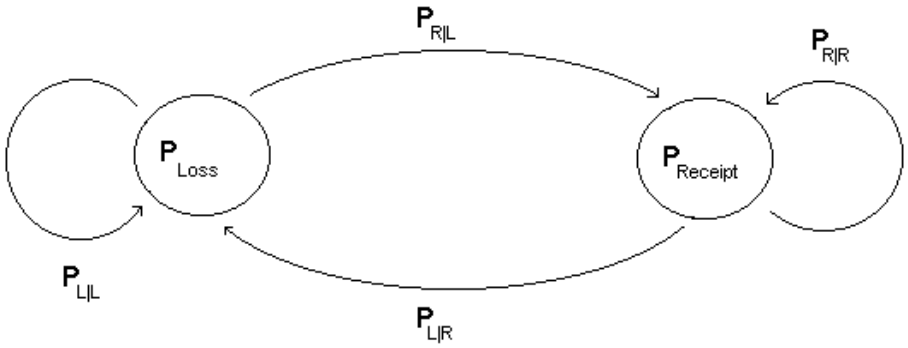


Abb. 4.4 : Binäres Markov Modell für Verlustsimulation

Vom Benutzer werden P_{Loss} und $P_{L|L}$ festgelegt. Alle weiteren Wahrscheinlichkeiten lassen sich aus diesen beiden mit Hilfe der Verbundwahrscheinlichkeiten berechnen :

$$P_{Loss} = P_{Loss} * P_{L\,|\,L} + P_{Re\,ceipt} * P_{L\,|\,R} \quad \text{mit} \quad P_{Re\,ceipt} = 1 - P_{Loss} \qquad \text{(4.1.a) / (4.1.b)}$$

$$P_{L\,|\,L} = 1 - P_{R\,|\,L} \quad \text{und} \quad P_{R\,|\,R} = 1 - P_{L\,|\,R} \qquad \text{(4.2.a) / (4.2.b)}$$

somit ist: $P_{L\,|\,R} = \frac{(1 - P_{L\,|\,L}) * P_{Loss}}{1 - P_{Loss}}$ bzw. $P_{R\,|\,R} = 1 - \frac{(1 - P_{L\,|\,L}) * P_{Loss}}{1 - P_{Loss}}$ (4.3.a) / (4.3.b)

Burst-Fehler können mit dem Modell auf einfache Weise durch Erhöhung von $P_{L|L}$ simuliert werden. Den Übergang von einem Zustand in den nächsten regelt ein Pseudo-Zufallszahlengenerator. Bei Kurzzeitmessungen werden darum die Verlustraten nicht immer genau eingehalten. Das Ergebnis der Simulationen ist ein Überblick über die Verzögerungszeiten des Server-Client-Systems. Für jede Wiederholung wird die Dauer vom Programm gemessen und kann in einem Konsolenfenster (DOS-Fenster) ausgegeben werden. Am Ende wird die mittlere Wartezeit berechnet. Auf dieser Basis lassen sich die optimalen Werte für RTT und T_{total} genauer abschätzen.

Um die Performance der Übertragung auswerten zu können, sammelt der RTP-Client statistische Daten. Zunächst werden die empfangenen und vermißten Pakete gezählt, außerdem die Anzahl an wiederholt übertragenen und letztendlich empfangenen Paketen. Die Differenz ergibt die Anzahl an verlorenen Paketen. Darüber hinaus wird für jedes Paket die Anzahl an NACKs und die Wartezeit erfasst. Ein Teil dieser Daten kann extern abgefragt werden. Dazu zählt auch die gemessene Bitrate, welche sich aus empfangenen Bytes und der aktuellen Zeitdauer ergibt und nicht unbedingt genau mit der beim Encoder vorgegebenen übereinstimmen muss. Die Messung der Bitrate erfolgt als Durchschnitt über die Gesamtzeit. Kurzzeitschwankungen werden also nicht erfasst. Der Benutzer kann jedoch jederzeit einen Neustart der Messungen veranlassen.

4.3.2 MPEG4-Decoder

Im Kapitel 2.1 wurden bereits die wichtigsten Eigenschaften des MPEG4-Codecs vorgestellt. Dieses Kapitel geht nun näher auf die Implementierungsdetails des Decoders ein.
Der Kern des Decoders basiert auf einer am HHI entwickelten C-Bibliothek. Zugriff auf die Bibliothek ist über eine Interface-Klasse möglich. Das Decodermodul selbst ist in einer Thread-Klasse gekapselt. Initialisierung und Steuerung des Threads erfolgen durch den MVC-Manager. Zur Übergabe von Daten, also dem Initial Object Descriptor (IOD/VOL-Header) und den Video Object Planes (VOPs/Frames), dient die Interfaceklasse “DecoderIf“. Die dort übergebenen Frames werden zunächst in einem internen Ringpuffer gespeichert. Für jedes Frame wird eine Zeit vorgegeben, zu der es dekodiert und gerendert werden soll. Im einfachsten Fall ist diese gleich der Ankunftzeit. Da die Frames jedoch nicht regelmäßig eintreffen, kommt es so zu einer ungleichmäßige Anzeige. In Kapitel 4.4 wird ein Verfahren zur Verhinderung dieses Effektes beschrieben.
Da bei einem Framebuffer aufgrund der kleinen P-Frames viel Speicherplatz verschwendet wird, wurde statt eines Mehrfach-Framebuffers ein Paket-Ringpuffer erstellt. Er kann maximal 1500 Pakete der Größe 1460 Bytes aufnehmen. Letztere Größe wurde in Anlehnung an den RTP-Client gewählt. Die hohe Anzahl an pufferbaren Paketen ist notwendig, um auch bei hohen Bitraten mehrere gleichzeitig übergebene Frames aufnehmen zu können. Dies tritt immer dann auf, wenn sich beim RTP-Client mehrere Frames aufgrund verspäteter Pakete angesammelt haben.
Der Ausgangspuffer besitzt das Format YC_bC_r 4:2:0. Es ist nur ein Ausgangspuffer notwendig, da er im Wesentlichen nur zur Übergabe des dekodierten Bildes an den Renderer dient. Zusätzlich ist es jedoch auch möglich, ihn extern auszulesen. Der Benutzer stellt bei Aufruf der entsprechenden Funktion einen weiteren Puffer bereit und erhält dann eine Kopie des letzten dekodierten Bildes, sowie Zusatzinformationen wie Bildgröße und Servertimestamp.
Die bei fehlerhafter Übertragung auftretenden Paketverluste führen zu Verlusten ganzer Frames, da keine Korrektur-Codes verwendet werden und die Aufteilung der Frames auf RTP-Pakete unabhängig vom Inhalt erfolgt. Wenn der Stream durch fehlende Pakete unterbrochen ist, kann sich der Decoder erst wieder mit dem nächsten Frame synchronisieren. Die Auswirkungen von Frameverlusten hängen vom Bildinhalt ab. Bei bewegten Objekten wirkt sich die Dekodierung von P-Frames auf Basis falscher Bewegungsvektoren mitunter sichtbar störend aus. Eine höhere I-Frame-Rate kann hier subjektiv Verbesserung bringen, der durchschnittliche PSNR-Wert als objektives Qualitätsmaß wird dadurch jedoch gesenkt [18].
Um Frameverluste sicher erkennen zu können, gibt es zwei Möglichkeiten. Zum einen bietet der VOP-Header einen Zeitstempel (vop_time_increment, siehe [5]), welcher gleichmäßig hochgezählt wird (mit wrap around). Weiterhin kann der 32-Bit-Timestamp im RTP-Header ausgewertet werden, welcher theoretisch eine höhere Genauigkeit bietet. Dieser wird vom Server allerdings zur Übertragung beliebiger Zeitinformation verwendet, deren Auflösung möglicherweise zu klein für eine Auswertung ist. In der aktuellen Version des RTP-Servers werden darum die obersten 5 Bits des externen Timestamps durch einen eigenen Counter (0-31) ersetzt. Nur die unteren 27 Bit bleiben zur Übertragung von Datum oder Uhrzeit. Es können auch Frameverluste beim Videoserver

miteinbezogen werden, z.B. bei Überlauf des Encoder- oder RTP-Puffers. Denn neben der Detektion von Frameverlusten aufgrund fehlerhafter Übertragung soll der Counter auch zur Resynchronisation der eintreffenden Frames dienen (Kap. 4.4).
Eines der wichtigen Features einer Überwachungsanwendung ist die Aufnahme der Bilddaten, um sie z.B. später auswerten zu können. Videodaten können in komprimierter und unkomprimierter Form gespeichert werden. Beide Bilddaten liegen am Decoder vor. Eine schnelle Festplatte (SCSI) wäre in der Lage, YUV-Bilder in Echtzeit zu speichern. Der Vorteil wäre, das zum Betrachten kein spezieller Decoder/Player notwendig ist und der Benutzer direkten Zugriff auf die Einzelbilder hat. Der hohe Speicherplatzbedarf und der Bedarf nach paralleler Aufnahme mehrerer Streams führten aber zu der Entscheidung, den MPEG4-Elementardatenstrom direkt zu speichern. Dieser besteht nur aus dem Initial Object Descriptor (VOL-Header) und den VOPs, das AVI- oder MPEG4-Fileformat wird nicht unterstützt. Der Stream kann z.B. mit einem einfachen Player des Heinrich-Hertz-Instituts abgespielt und auch als YUV-File abgespeichert werden. Der Dateiname wird automatisch aus Kamera- bzw. Servername und dem aktuellen Servertimestamp generiert. Das Starten und Stoppen der Aufnahme erfolgt durch den Benutzer. Nach dem Starten muss der Decoder jedoch erst auf das nächste I-Frame warten, da vorherige P-Frames nicht dekodiert werden könnten. Bei Frameverlusten werden in den gespeicherten Stream keine Frames eingefügt, da Verluste gemäß dem Standard auch aus den Zeitstempeln im VOP-Header erkennbar sind.

4.3.3 Renderer

Das letzte Modul in der Übertragungskette zeigt das Bild auf dem Monitor an. Unter dem Windows-Betriebssystem gibt es dafür den Fenster- und den Vollbildmodus. Ersterer ist am einfachsten zu handhaben. Für jedes Bild muss ein Fenster bereitgestellt werden. Zur Erzeugung von Fenstern bietet die Win32-Api eine Reihe von Funktionen. Die wichtigsten davon wurden in einer einfachen Klasse (RendererWindow) gekapselt. Von der MFC-Fensterklasse CWnd wurde nicht Gebrauch gemacht. Ein Fenster wird zunächst durch eine globale Fensterklasse charakterisiert. Diese ist nicht zu verwechseln mit den genannten Klassen wie RendererWindow oder CWnd. Sie enthält vielmehr den Typ, Eigenschaften und den Verweis auf eine globale Fensterprozedur. Die Fensterprozedur dient der Verarbeitung von Nachrichten, welche an die auf der globalen Fensterklasse basierenden Fenster verschickt wurden. Unter Windows wird die Kommunikation zwischen Fenstern mittels Nachrichten geregelt ([17]). Diese bestehen aus einer Nachrichten-ID und zwei Integer-Parametern. Ein typisches Beispiel ist die Nachricht WM_RBUTTONDOWN, welche besagt, das mit der rechten Maustaste in das Fenster geklickt wurde. Die Parameter enthalten dann z.B. die Koordinaten des Mausklicks. Die Anwendung ist für die Abfrage und Bearbeitung eintreffender Nachrichten selbst verantwortlich, wobei zum Teil schon eine Standardreaktion per Default eingestellt ist (zum Beispiel Schließen des Fensters bei Klick auf den Close-Button). Um Fenster voneinander unterscheiden zu können, bekommt jedes bei der Erzeugung einen eindeutigen Fensterhandle zugewiesen. Dieser wird bei Funktionen zur Steuerung von Fenstern benötigt.

Der Multivideoclient unterstützt drei Modi zum Rendern in ein Fenster. Im Standardmodus (Singlemode) wird intern ein Fenster pro Stream erzeugt. Der Handle dieses Fensters wird per Windows-Nachricht der Hauptanwendung bekanntgegeben, damit diese das Fenster bei Bedarf kontrollieren kann. Dieser Modus ist der einzige, bei dem vom MVC in der Fensterüberschrift zusätzliche Informationen zum Stream angezeigt werden. Dazu zählt der Servername und die Serverzeit. Bei den beiden anderen Modi muss dies durch die Hauptanwendung vorgenommen werden.
Im zweiten Modus werden mehrere Streams in ein gemeinsames, extern erzeugtes Fenster gerendert. Der Anwendung gibt dabei die Zeile und Spalte für jeden Stream vor. Alle Videos haben dieselbe Bildgröße, unabhängig von ihrer tatsächlichen Auflösung.
Die größte Freiheit bei der Gestaltung der Bedienoberfläche bietet aber der dritte Modus. Auch für diesen muss das Fenster extern erzeugt werden. Die Anwendung kann dann die genaue Position in Pixelkoordinaten für das Video vorgeben. Sie muss sich bei Änderungen der Fenstergröße aber auch um die Einstellung der neuen Videokoordinaten kümmern.

Abb. 4.5 : Renderer-Modi (Single-, Multi- und Free Mode)

Zum Anzeigen eines Bildes auf dem Monitor gibt es ebenfalls drei Möglichkeiten. Die Standardvariante ist das Zeichnen des Bildes mit Hilfe der GDI-Funktionen der Win32-API. Dies ist grafikkartenunabhängig aber auch am langsamsten. Die beiden anderen Möglichkeiten betreffen die von modernen Grafikkarten bereitgestellten Schnittstellen OpenGL und das von Microsoft entwickelte und entsprechend weit verbreitete DirectX. DirectX beinhaltet Features zum Bearbeiten von 2D- und 3D-Grafik, Sound und vielfältigen Eingabegeräten (Joystick, Gamepad etc.). Die 2D-Funktionen sind unter der Bezeichnung DirectDraw zusammengefaßt und werden vom aktuellen Renderer als einzige der drei Varianten unterstützt. Die 2D-Grafikobjekte werden durch sogenannte DirectDraw-Surfaces (Oberflächen) repräsentiert, welche einen Speicherbereich im Grafikkartenspeicher/Systemspeicher sowie Angaben zum Typ und gewisse Features enthalten ([17]).
Zunächst gibt es eine primäre Oberfläche im sichtbaren Bereich der Grafikkarte, d.h. diese Oberfläche repräsentiert den Monitorinhalt. Weiterhin gibt es Hintergrund-Oberflächen, in denen das Bild aufgebaut wird. Hier können z.B. zur Vorbearbeitung mehrere Objekte hineinkopiert oder gezeichnet werden. Erst nach Fertigstellung des ganzen Bildes im Hintergrundspeicher wird dieser in den Vordergrundspeicher kopiert. Durch die hohe Kopiergeschwindigkeit erreicht man eine flüssige Darstellung am Bildschirm.

In der vorliegenden Renderer-Version gibt es eine primäre DirectDraw-Oberfläche, welche vom MVC-Manager zentral erzeugt und verwaltet wird. Darüber hinaus wird für jeden Stream durch den Renderer eine Hintergrundoberfläche der Größe Bildbreite x Bildhöhe erzeugt. Diese Oberfläche kann vom Typ RGBA oder auch UYVY sein, sofern die Grafikkarte die automatische Farbkonvertierung UYVY->RGBA unterstützt. Der Vorteil einer UYVY-Oberfläche ist zum einen die schnellere Konvertierung der Bildes durch den Grafikkartenprozessor. Zum anderen muss nur die halbe Datenmenge (16 Bit pro Pixel) über den AGP-Bus transportiert werden. Unterstützt die Grafikkarte dies jedoch nicht, so legt der Renderer automatisch eine RGBA-Oberfläche (32 Bit pro Pixel) an und die Formatkonvertierung YUV420->RGBA wird vom Prozessor selbst durchgeführt. Dazu wurden Konverterroutinen in C++, MMX und SSE2 implementiert. SSE2 ist dabei ca. doppelt so schnell wie C++, wird aber nur von Pentium 4 Prozessoren unterstützt.
Neben der Aufnahme des Streams durch den Decoder bietet der Renderer die Aufnahme (Capturing) von Einzelbildern, z.B. zwecks Verwendung als Beweismaterial oder zum genauen Betrachten von Details. Es können Einzelbilder im Microsoft-Bitmap-Format mit einer Pixeltiefe von 32 Bit gespeichert werden. Zum Betrachten dieses Standardformats ist jedes Bildbearbeitungsprogramm geeignet. Nachteile sind der hohe Platzbedarf (z.B. > 1 MByte/Bild) und die längere Speicherdauer (u.U. mehrere 100 ms), welche eine Aufnahme von Bildfolgen in Echtzeit schwierig macht.
Der Name der Bitmap-Datei setzt sich aus Servername und Servertimestamp zusammen, um eine spätere Zuordnung der Bilder zu vereinfachen. Das Capturen von Bildern kann durch zwei Vorgänge ausgelöst werden. Bei einem Klick mit der rechten Maustaste in ein internes Videofenster wird automatisch das aktuelle YUV-Frame in einen Nebenpuffer kopiert. Die Anwendung kann dann durch Aufruf einer Funktion des MVC-Interfaces das Speichern auf Festplatte veranlassen. Zudem kann die Funktion auch ohne vorherigen Mausklick aufgerufen werden, was z.B. bei der Verwendung externer Videofenster notwendig ist.

4.3.4 Framesynchronisation

Ein bei der Übertragung von Videodaten auftretendes Problem sind die unterschiedlichen Übertragungszeiten der einzelnen Bilder von der Quelle zur Senke. Dazu zählt nicht nur die reine Übertragung über das Netz, sondern auch die Vorverarbeitung beim Server und Aufbereitung beim Client. Da es sich um eine unidirektionale Übertragung handelt, ist die Gesamtverzögerung weniger problematisch, sofern sie nicht größere Ausmaße von einigen Sekunden oder mehr annimmt. Vielmehr geht es aber um die Abhängigkeit der einzelnen Verzögerungen und somit auch der Gesamtverzögerung von einigen, teilweise nicht beeinflussbaren Faktoren, wie z.B. Bildinhalt und der Art der Frames. Die Schwankung der Übertragungszeiten führt zu einer ungleichmäßigen Anzeige der Bilder auf dem Monitor, was sehr störend wirken kann.

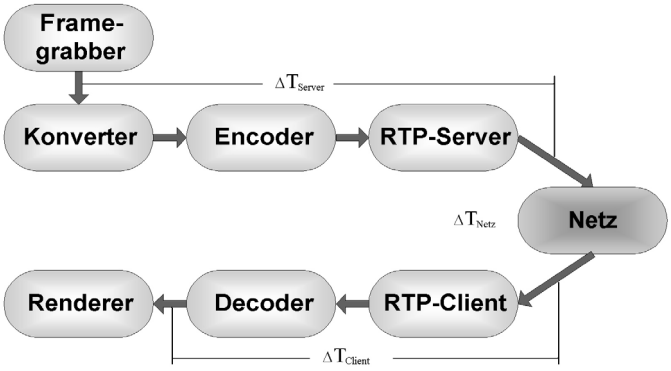


Abb. 4.6 : Verzögerungen bei der Videoübertragung

Die Abbildung 4.6 zeigt die drei wesentlichen Abschnitte der Übertragung. Dabei wird angenommen, das die Bilder in regelmäßigen Zeitabständen T vom Framegrabber kommen. Außerdem wird die Verzögerung des Renderers vernachläßigt, da sie nur gering (ca. 1-2 ms bei modernen Grafikkarten) und auch kaum Schwankungen unterworfen ist.

Auf Seiten des Servers bleiben noch 3 Fehlerquellen. Da das externe Bildformat vom internen entkoppelt ist, muss der Konverter ggf. eine Umwandlung des Bildtyps nach YC_bC_r 4:2:0 und eine räumliche Unterabtastung durchführen, letztere möglicher-weise mit aufwendiger Tiefpassfilterung. Auch wenn man davon ausgehen kann, das die Konvertierung für jedes Bild annähernd die selbe Zeit in Anspruch nimmt, so führt die dafür benötigte Zeit zusammen mit der Kodierzeit aus anderen Gründen zu Schwankungen der Bearbeitungsdauer. Konverter und Encoder laufen zusammen in einem separaten Thread mit eigenem Eingangs-Ringpuffer und zeitlich entkoppelt vom Framegrabber und RTP-Server. Da das Betriebssystem die Prozessorzeit aber den Threads der Reihe nach zuteilt, kann es zu unregelmäßigen Unterbrechungen des Konvertier- und Kodiervorganges kommen, wenn andere Threads gerade Rechenzeit beanspruchen. Gleiches gilt auch für den Thread des RTP-Servers. Auch hier kann die Versendung der RTP-Pakete nach Ablauf des Zeitquantums (z.B. 10 ms für Single-Prozessor-Systeme) unterbrochen werden bis der RTP-Thread wieder an der Reihe ist.

Beim Encoder selbst sind im Wesentlichen die unterschiedlichen Kodierzeiten für I- und P-Bilder sowie die Abhängigkeit der Kodierzeit bei P-Bildern von der Bewegungskomplexität als weitere Gründe zu nennen.

Auf Seiten des Clients gelten ähnliche Ursachen. Auch hier verursacht das Multithread-Design Schwankungen der Streamverarbeitung. Außerdem werden ebenfalls verschiedene Zeiten für die Dekodierung von I- und P-Bildern benötigt.

Das für die Übertragung verwendete Netz kann aber unter Umständen für die größten Verzögerungen und Schwankungen verantwortlich sein. Zum einen benötigen I-Frames im Normalfall mehr Zeit als P-Frames (Größenverhältnis z.B. 10:1). Zum anderen werden über das Netz möglicherweise noch andere Daten übertragen, sodass es zu kurzen oder längeren Staus, Kollisionen oder verschiedenen Routen bei größeren Netzen (WAN, WWW) kommen kann.

Die folgenden Abbildungen sollen die Problematik verdeutlichen. Sie zeigen Häufigkeitsverteilungen der Zeitdauer zwischen zwei Bildern, also die Schwankungen der Periode. In Anhang B sind weitere Meßergebnisse dargestellt.

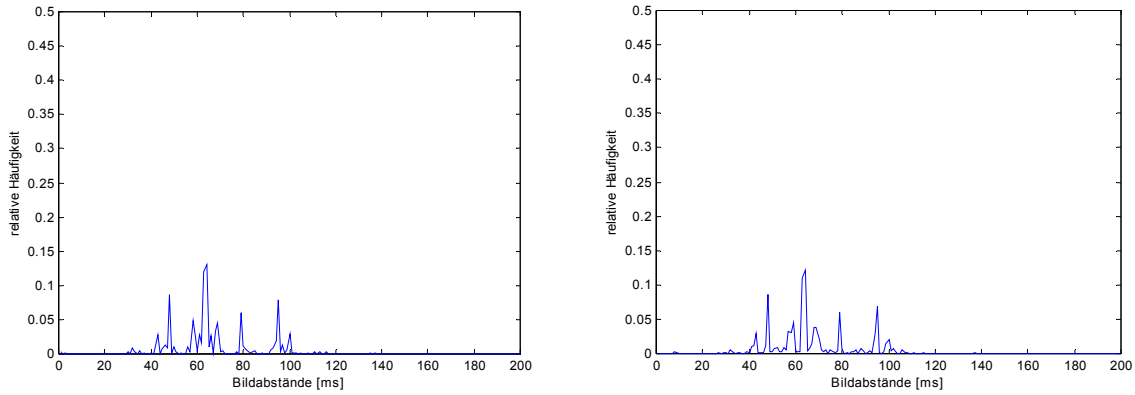


Abb. 4.7 : Verteilung der Bildabstände beim a) RTP-Server b) Renderer (Client)
Einzelstream 320x240x15Hz (66ms) unsynchronisiert :

Man erkennt starke Schwankungen schon auf der Serverseite. Für die Peaks und die breite Streuung von maximal ca. ± 35 ms ist vor allem das Multithreaddesign verantwortlich.
Um eine gleichmäßige Anzeige der Bilder zu erreichen, muss an geeigneter Stelle ein Ringpuffer eingebaut werden, aus dem zu berechneten Zeiten mit dem Abstand T ein Bild entnommen wird. Der optimale Ort für diese Synchronisation wäre der Renderer beim Client, da bis zur eigentlichen Anzeige nur noch kurze Zeit vergeht. Der Nachteil hier ist aber der große Speicherbedarf. Pro Bild werden B x H x 1,5 Bytes (für YC_bC_r 4:2:0) benötigt. Darum wurde die Synchronisation zwischen RTP-Client und Decoder verlegt, welche beide in separaten Threads laufen. An dieser Stelle liegen die Bilder noch im komprimierten Zustand vor.
Es wird wie folgt vorgegangen:

- Speicherung des einkommenden Frames im nächsten freien Decoder-Puffer (Paketringpuffer)
- Auswertung und Aufbereitung des mitgelieferten Serverzeitstempels, Detektion verlorener Frames
- Berechnung des Dekodierzeitstempels aus aufbereiteter Serverzeit und aktueller lokaler Zeit
- Der Decoderthread prüft ständig, ob der Zeitstempel des nächsten zu dekodierenden Frames abgelaufen ist. Wenn ja, dann wird es kopiert, die Paketpuffer freigegeben und das Bild wird dekodiert und gerendert.

Zur Berechnung des Dekodierzeitstempels wird das Least-Square-Verfahren benutzt. Es wird die Gerade gesucht, für welche die Summe der quadratischen Fehler zu den unregelmäßigen lokalen Zeitstempeln am kleinsten ist.

Abb. 4.8 zeigt ein fiktives Beispiel :

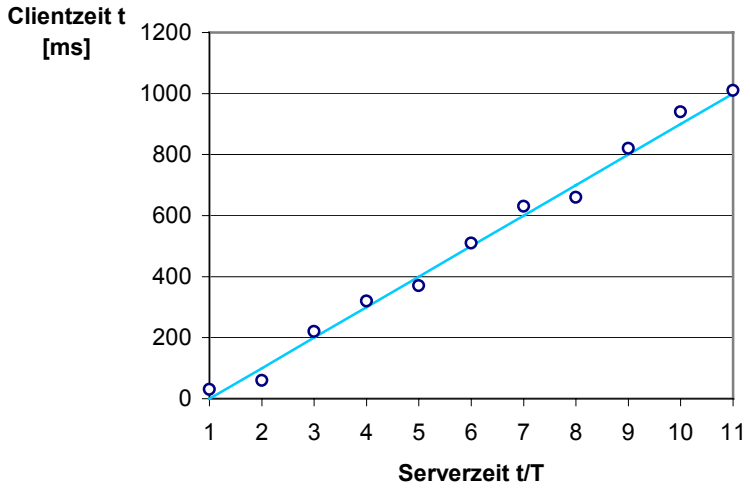


Abb. 4.8 : unregelmäßige Frame-Ankunftszeiten und optimaler Verlauf (Gerade)

Die Geradengleichung lautet demnach :

$$t_{Decode} = a * t_{Server} + b \qquad (4.4)$$

t_{Decode} ist der zu berechnende Zeitstempel, bei dem das neue Frame dekodiert werden soll, t_{Server} ist der gleichmäßig wachsende Serverzeitstempel. Da aber nur der lokale Zeitstempel t_{Client}, also die aktuelle Zeit, zur Verfügung steht, müssen a und b mittels Least-Square-Verfahren berechnet werden. Die Summe des quadratischen Fehlers von t_{Decode}–t_{Client} ist :

$$\sum_{i=1}^{N} (\Delta t)^2 = \sum_{i=1}^{N} (a * t_{Server} + b - t_{Client})^2 \qquad (4.5)$$

Zur Berechnung des Minimums werden die partiellen Ableitungen nach a und b bestimmt und zu 0 gesetzt. Diese liefern ein Gleichungssystem, welches in Matrixschreibweise wie folgt aussieht :

$$\begin{pmatrix} \sum_{i=1}^{N} t_{Client} * t_{Server} \\ \sum_{i=1}^{N} t_{Client} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} t_{Server}^2 & \sum_{i=1}^{N} t_{Server} \\ \sum_{i=1}^{N} t_{Server} & \sum_{i=1}^{N} 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \qquad (4.6)$$

Der gesuchte Vektor $(a\ b)^T$ ergibt sich also nach Berechnung der inversen 2x2 Matrix. Mit den so ermittelten Werten kann der Dekodierzeitpunkt bestimmt werden. Dieser kann allerdings auch in der Vergangenheit liegen, d.h. der berechnete optimale Zeitpunkt ist bereits überschritten. Das Frame muss dann sofort gerendert werden. Die Differenz zur optimalen Zeit wird als Offset bei allen nachfolgenden Frames dazuaddiert. Weiterhin sind auch die verschiedenen Dekodierzeiten für I- und P-Frames zu beachten. Darum werden diese Zeiten bei jedem Frame gemessen und fließen in die

Berechnung mit ein. Der Dekodierzeitstempel ergibt sich dann wie folgt :

$$t_{Decode,\,I} = a * t_{Server} + b + t_{offset} + t_P \qquad (4.7.a)$$

$$t_{Decode,\,P} = a * t_{Server} + b + t_{offset} + t_I \qquad (4.7.b)$$

wobei t_P und t_I für die mittlere Dekodierdauer für P- und I-Frames stehen.
Abb. 4.9 zeigt das Ergebnis der Synchronisation. Die Bilder werden in fast gleichmäßigen Abständen gerendert:

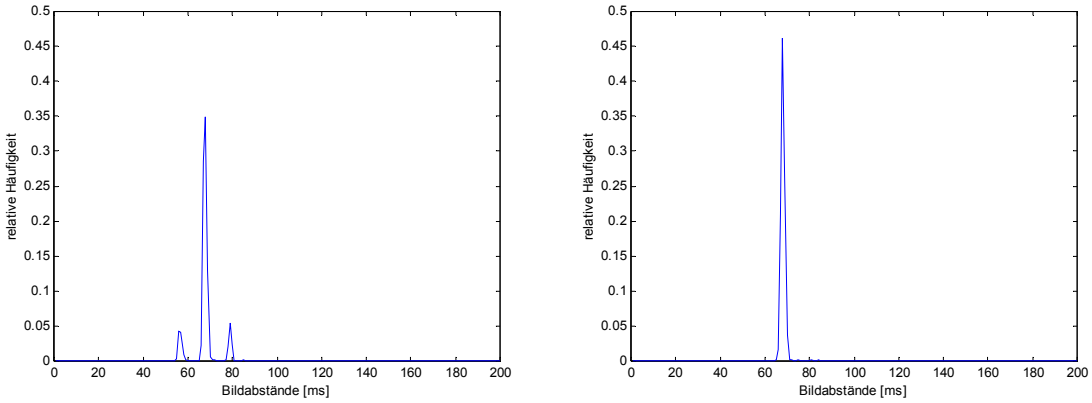


Abb. 4.9 : Verteilung der Bildabstände beim Renderer (f =15Hz)

a) ohne Einberechnung t_P , t_I b) mit Einberechnung t_P , t_I

4.4 Stream Control

Neben der manuellen Kontrolle der Streams durch den Benutzer ist auch eine interne automatische Steuerung möglich. Das Stream Control Module ist das Ergebnis von Untersuchungen zum Streaming über Wireless LAN und anderer Vorgaben. Es kann prinzipiell auch für Ethernet eingesetzt werden. Das interne Modul wird vom MVC-Manager verwaltet. Die Initialisierung und Aktivierung erfolgt aber durch den Benutzer. Dieser kann die automatische Steuerung beliebig an– und ausschalten. Bei der Initialisierung muss dem Modul u.a. der Name einer Datei mit Netzwerkkonfigurationsdaten übergeben werden. Aus dieser Datei wird eine interne Liste mit Informationen über alle separaten Netzwerke, sowie allen in den Netzen befindlichen Servern erstellt. Sie muss folgende Daten enthalten :

- mindestens eine Netzwerk-ID
  Die Videoserver können in Gruppen zusammengefaßt werden (z.B. alle Server einer Kreuzung oder einer Etage). Die Steuerung erfolgt dann für jede Gruppe separat. Für jede Gruppe muss eine eindeutige ID vergeben werden. Ein Beispiel mit zwei Subnetzen ist in Abbildung 4.10 zu sehen.
- Angaben über die maximalen Bandbreiten (=Gesamtbitraten) für jedes Subnetz
  Die maximale Gesamtbitrate ist abhängig von der Anzahl offener Streams je Gruppe. Die Werte müssen also für jede mögliche Anzahl von Streams definiert sein. Zur Ermittlung der Werte müssen nach der Installation des Netzes Versuche durchgeführt werden. Dazu werden z.B. N Streams gestartet und die Bitraten solange erhöht, bis Frameverluste bei Servern oder

hohe Paketverluste auftreten. Die einzutragenen Bitraten ergeben sich dann aus diesen Grenzwerten abzüglich eines Sicherheitsabstandes.

- eine Liste mit Server-IDs
  Die IDs müssen mit denen übereinstimmen, welche auch zum Starten der Streams benutzt werden. Jeder Server hat nur eine eindeutige ID. Weiterhin muss für jeden Server die Netzwerkgruppe und eine Priorität zwischen 1 (niedrig) und 3 (hoch) vergeben werden. Diese Prioritäten geben an, wie stark die Bitrate der einzelnen Server gesenkt werden darf. Besonders sicherheitsrelevante Videostreams behalten so bei Netzproblemen ihre Datenrate, während die Bitrate und demzufolge die Qualität weniger wichtiger Streams gesenkt wird.

Im Anhang E ist ein Beispiel für eine Konfigurationsdatei angegeben. Die Endung ist per Default "*.ncf". Es handelt sich aber um eine einfache Textdatei. Die beiden Hauptteile sind durch Anfangs- und Endmarkierungen getrennt.

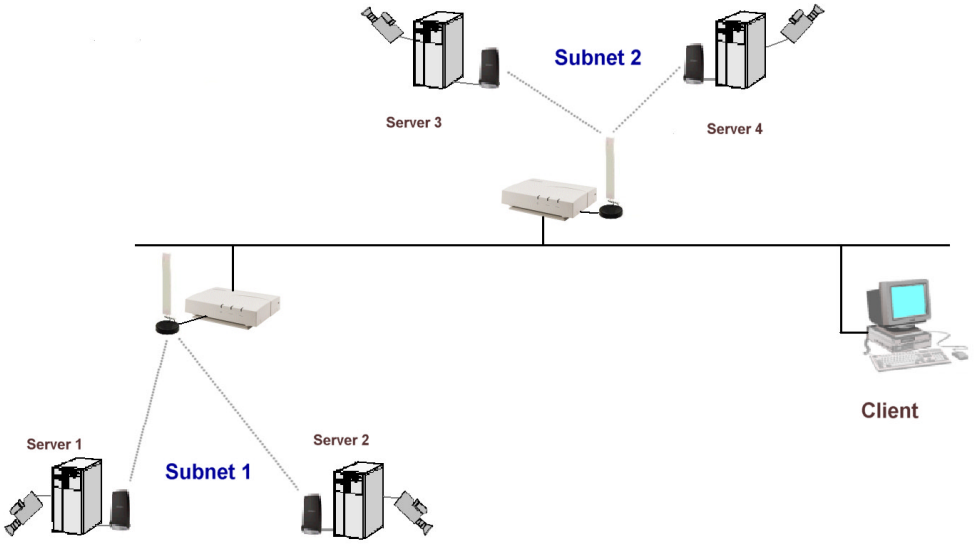


Abb. 4.10 : Unterteilung eines Überwachungsnetzes

Das Stream Control Module wurde als Klasse implementiert. Außerdem gibt es die Interfaceklasse "StreamControlIf", deren Pointer den einzelnen RTPClient- und Decoderthreads übergeben wird. Diese können das Kontrollmodul darüber mit statistischen Daten versorgen. Der Aktualisierungszeitraum beträgt zwischen 10 und 60 Sekunden in Schritten von 10 sec. Die aus der Konfigurationsdatei erzeugte Liste mit allen Servern enthält Einträge für die letzten und die aktuellen Frameverlustraten. Da das Kontrollmodul nicht in einem eigenen Thread läuft, muss die Taktung von außen erfolgen. Dies geschieht mit Übergabe der Werte für verlorene Frames vom letzten Videoclient des Subnetzes. Bei diesem Ereignis werden die zu diesem Zeitpunkt vorliegenden statistischen Daten ausgewertet und Entscheidungen bzgl. der Neuregelung von Bitraten oder Videoprofilen getroffen.
Theoretisch wäre es möglich, die Steuerung der Bitraten/Videoprofile direkt intern vorzunehmen. Dies hätte aber bedeutende Nachteile. Die externe Anwendung bzw. der Benutzer würde davon keine Kenntnis erhalten. Es muss aber davon ausgegangen werden, dass die Anwendung ebenfalls eine Liste mit aktuellen Informationen über Videoprofile, Bitraten und Verlustraten verwaltet oder eventuell

neben der manuellen Kontrolle durch den Benutzer sogar ein externes automatisches Kontrollmodul implementiert wurde. Eine externe Kontrolle oder Datenbank würde also kurzzeitig oder auch längerfristig nicht auf dem neuesten Stand sein und möglicherweise andere Entscheidungen zur Streamsteuerung treffen. Zur Verhinderung möglicherweise gegensätzlicher Steuerungsbefehle wurde der indirekte Weg gewählt. Das Kontrollmodul teilt seine Steuerkommandos der externen Anwendung über eine Windows-Nachricht mit. Dazu wird dem Modul bei der Initialisierung der Handle des Hauptanwendungsfensters und die ID der Kontrollnachricht übergeben. Die Nachricht besteht aus der Nachrichten-ID, sowie zwei 32-Bit-Integer-Parametern, von denen der erste die ID des zu steuernden Streams enthält. Der zweite enthält in den oberen 16-Bit das neue Videoprofil oder in den unteren die neue Bitrate in [kbits/s]. Soll einer der Werte nicht geändert werden, so wird er auf 0 gesetzt. Nach Erhalt der Nachricht kann die Hauptanwendung über die entsprechende Interface-Funktion des MVC (siehe Anhang C) die Änderungen am Stream veranlassen. Somit ist gewährleistet, dass die Anwendung eine eigene Datenbank rechtzeitig aktualisieren und den Benutzer informieren kann. Die interne Liste mit den Clientdaten im Kontrollmodul wird automatisch auf den neuesten Stand gebracht.
Die Verwendung von Windows-Nachrichten hat allerdings auch einen Nachteil. Die damit verknüpfte Aktion wird von einem Fensterthread der Hauptanwendung durchgeführt. Es ist aber möglich, das während des Ablaufes anderslautende Befehle von einem Nebenthread kommen. Dies kann z.B. ein Menü zur Einstellung der Bitraten durch den Benutzer sein. Neben der Abgleichung der unterschiedlichen Streamdaten zwischen beiden externen Threads ist es ebenfalls wichtig zu klären, wie die Threads mit der blockierten internen Fernsteuerung umgehen. Diese kann nicht mehrere Kommandos auf einmal bearbeiten. Solange die Antwort auf ein Kommando vom Server nicht eingetroffen oder es zu einem Timeout gekommen ist, nimmt sie keine neuen Befehle entgegen. Dies kann unter Umständen Sekunden dauern. Der Umgang mit dieser Blockade bleibt der Hauptanwendung überlassen. Mögliche Strategien sind :

- vorübergehendes Abschalten der internen Steuerung bei Aktivierung von externer Steuerung
- externe Steuerung durch Versendung einer Nachricht an das Hauptanwendungsfenster. Da die Hauptanwendung die Nachrichten von beiden Seiten nur nacheinander abarbeiten kann, sind Kollisionen ausgeschlossen.
- Verwendung von Backoff-Zeiten nach einem bestimmten Algorithmus, z.B. ähnlich der MAC-Zugriffssteuerung CSMA/CD bei Ethernet. Der Thread, welcher die Fernsteuerung blockiert vorfindet, zieht sich für einige Zeit zurück und versucht es dann noch einmal.

In der implementierten Anwendung TraViS (siehe Kapitel 4.5.1) wird die erste Methode verwendet. Bei Öffnen eines Dialoges, in dem der Benutzer Videoprofil und Bitrate einstellen kann, wird die interne Steuerung zeitweilig deaktiviert. Das Modul erhält zwar weiterhin die statistischen Daten der Videoclients, unterläßt aber die Auswertung. Im umgekehrten Fall kann kein Steuerungsdialog geöffnet werden, wenn das Kontrollmodul gerade die Bitraten aufgrund von Netzstörungen anpasst. Zur Signalisierung des Endes der automatischen Steuerung versendet das SCM eine letzte Kontrollnachricht an die Hauptanwendung bei der aber die beiden zugehörigen Parameter 0 sind.

4.5 Implementierte Anwendungen für den MVC

Das Programm MultiVideoClientTest.exe ist eine einfache Konsolenanwendung zur Überprüfung der korrekten Funktionsweise des Multivideoclients. Außerdem wurde dieses Programm auch für die Untersuchungen der Streamingfähigkeiten des Test-WLANs verwendet. Es kann bis zu vier Clients starten und gibt im Konsolenfenster die Debug-Informationen und statistischen Daten des MVCs aus. Neben dem Konsolenfenster wird ein weiteres Fenster erstellt, welches als Ersatz für das Hauptanwendungsfenster einer Win32-Anwendung dient. Die empfangenen Videostreams können auch zusammen in dieses Fenster gerendert werden.
Die Anwendung TraViS (Traffic Video Surveillance) ist eine auf der Microsoft Foundation Class basierende Anwendung mit grafischer Benutzeroberfläche. Die MFC-Bibliothek stellt die dafür benötigten Dialogklassen zur Verfügung. TraViS kann eine Vielzahl von Videostreams verwalten. Die Daten der einzelnen Server, wie ID oder IP-Adresse werden aus einer einfach editierbaren Datei geladen. Zentrale Aufgabe des Programmes ist die menügeführte Steuerung der Streams.
Eine ausführliche Dokumentation mit Beschreibung aller Menüs ist in Anhang D.

5 Untersuchung des Systemes auf 802.11b Wireless LAN

5.1 Vorbereitung

5.1.1 WLAN-Ausrüstung

Zum Aufbau des Test-WLANs wurden Komponenten und Software des Herstellers Compaq (jetzt HP) verwendet. Es wurde ein sogenanntes Infrastructured WLAN aufgebaut, in dessen Zentrum ein Access Point WL410 steht. Dieser besitzt einen 10 MBit-Ethernetanschluß zur Verbindung mit einem leitungsgebundenen Netz. Somit wird die Maximalbandbreite von WLAN bereits eingeschränkt. Der Anschluß des Access Points erfolgte direkt am PC mit der Clientanwendung. Von diesem PC aus kann das WLAN auch konfiguriert werden. Um die Wahrscheinlichkeit von Interferenzen mit anderen eventuell in der Nähe befindlichen Netzen zu senken, wurde auf einen anderen Kanal umgeschaltet.
Trotz dieser Maßnahme war es während der gesamten Tests wichtig, mit Hilfe eines Herstellertools die Störleistung zu beobachten, um bei Aktivierung weiterer Netze sofort auf einen ungestörten Kanal auszuweichen.
Die WEP-Verschlüsselung wurde deaktiviert, da die Sicherheit von 128 Bit-RC4-Codes nur gering ist. Die große Menge von übertragenen Videodaten ermöglicht es zur Zeit noch relativ einfach anhand der mitgeschnittenen Daten den Code zu entschlüsseln. Ein Einsatz von WLAN für sicherheitsrelevante Überwachungstechnik ist bis zur Verbesserung der Verschlüsselungsalgorithmen also nicht anzuraten.

Abb. 5.1 : WLAN-Equipment (Access Point, externe Antenne, USB-Adapter)

Die Verbindung mit dem WLAN übernimmt eine im Access Point eingebaute PCMCIA-Karte vom Typ WL110. Diese für Laptops entwickelte Karte besitzt auch einen Anschluß für eine externe Antenne. Die beiden Vorteile einer solchen Antenne ist die vergrößerte Reichweite (ca. 1.5fach) sowie die Möglichkeit der festen Installation bei optimaler Ausrichtung außerhalb der störenden Abschirmung durch PC, Laptop oder Personen. Die senkrechte Ausrichtung der Antenne ermöglicht den Betrieb anderer WLAN-Adapter in der Hauptstrahlrichtung (2.4.1).
Die bis zu vier als Videoserver betriebenen PCs wurden mit einem USB-WLAN-Adapter vom Typ WL215 ausgestattet. Der Nachteil von ebenfalls erhältlichen PCI-Karten (WL120) ist der feste Einbauort an der Rückseite des PCs, dessen Gehäuse die Signale deutlich abschirmt.
Die USB-Adapter dagegen besitzen zwei integrierte Antennen zur Erhöhung der Reichweite und sind variabel aufstellbar. Alle Komponenten besitzen eine maximale Sendeleistung von 15dBm. Im 11-MBits/s-Modus können Laufzeitunterschiede bis 65ns (frame error rate < 1%) kompensiert werden. Bei einem doppelt empfangenen Signal z.B. entspricht dies einer Wegdifferenz von 19,5m.

Wie bereits in Kapitel 2.4.1 angesprochen, wird die Empfangsqualität in vier Kategorien eingeteilt : “excellent“ (sehr gut), “good“ , “marginal“ und “poor“. Es gab allerdings keine Informationen darüber, welche Signal-Rausch-Abstände für diese Grobeinteilung gelten. Möglicherweise fließen in die Bewertung noch andere Größen, wie Paketverlustraten oder Absolutwerte für Signal- und Rauschleistung ein. Bei den Messungen wurde für die beste Kategorie “excellent“ ein Grenzwert von ca. 25dB beobachtet, für “good“ gilt ein Bereich von etwa 15-25 dB.

5.1.2 Testumgebung

Die Tests wurden innerhalb des HHI-Bürogebäudes in Alt-Moabit durchgeführt. Die Etagen sind jeweils mit mehreren kleinen Büros, getrennt durch massive Mauern ausgestattet. In den Büros befindet sich z.T. zahlreiche Hardware und Mobiliar. Diese Umgebung hat für das Test-WLAN folgende Auswirkungen :

- Die Reichweite hält sich in Grenzen. Ein Empfang ist nur durch maximal zwei Räume auf derselben Etage oder einen Raum ober- bzw. unterhalb des Senders möglich. Dies wurde mit einem Notebook getestet
- Die Stärke des Datensignales schwankt stark ([11]). Dies hat bei großem SNR nur geringe Auswirkungen auf die Übertragung, bei ohnehin kleiner Empfangsleistung kann es dagegen Zusammenbrüche der Verbindung kommen.
- Mehrwegeempfang stellt im Vergleich zu der hohen Dämpfung nur ein geringes Problem dar. Die Laufzeitunterschiede sind nur gering bzw. Signale mit längerer Wegstrecke kommen nur stark gedämpft an. (Dies gilt nicht für Übertragungen über mehrere Räume!)
- Interferenzen mit anderen WLANs sind sehr unwahrscheinlich. Diese müßten sich schon in benachbarten Räumen befinden, um einen störenden Einfluß zu haben. Für diesen Fall reicht das Ausweichen auf einen anderen Kanal normalerweise aus.
- Die Testbedingungen sind veränderlich. Es treten sowohl kurzfristige (Personenbewegungen) als auch langfristige (Personenanwesenheit, Mobiliarbewegung) Veränderungen auf. Die für Outdoor-Tests bedeutsamen Wetterbedingungen spielen dagegen keine Rolle.

Die Durchführung der Tests erfolgte in einem Raum der Größe 6m x 4,80m. In einer Ecke befand sich der Client-PC mit dem Access Point+Antenne. Direkt daneben (ca. 1m) ein Server-PC. Zwei der anderen Server-PCs wurden in den beiden gegenüberliegenden Ecken des Raumes aufgestellt, der letzte etwas näher zum Client. Für die Messungen mit 2 und 3 Servern kamen Server-PCs an unterschiedlichen Standorten zum Einsatz.
Die Antenne des AP und die USB-Adapter wurden erhöht aufgestellt, um eine größtenteils freie Sichtverbindung (Fresnelzone) untereinander zu ermöglichen. Dies traf jedoch nicht zu jeder Zeit zu.
Wie zu erwarten, waren die Signal-Rausch-Abstände bei allen Sendern sehr hoch. Allerdings bezieht sich dieses Prädikat auf einen relativ weiten Bereich der Empfangsleistung. So wurden SNR-Werte zwischen ca. 40 und 60 dB gemessen, je nach Größe der Abschirmung durch Personen, PCs oder

Stühle und der Ausrichtung der Doppel-Antennen der USB-Module. Die Antennen befanden sich näherungsweise auf gleicher Höhe und somit in Hauptabstrahlrichtung zueinander. Die Störleistung lag während der Tests gleichmäßig auf einem sehr niedrigen Niveau von ca. -90 dBm. Das folgende Bild zeigt die Raumaufteilung :

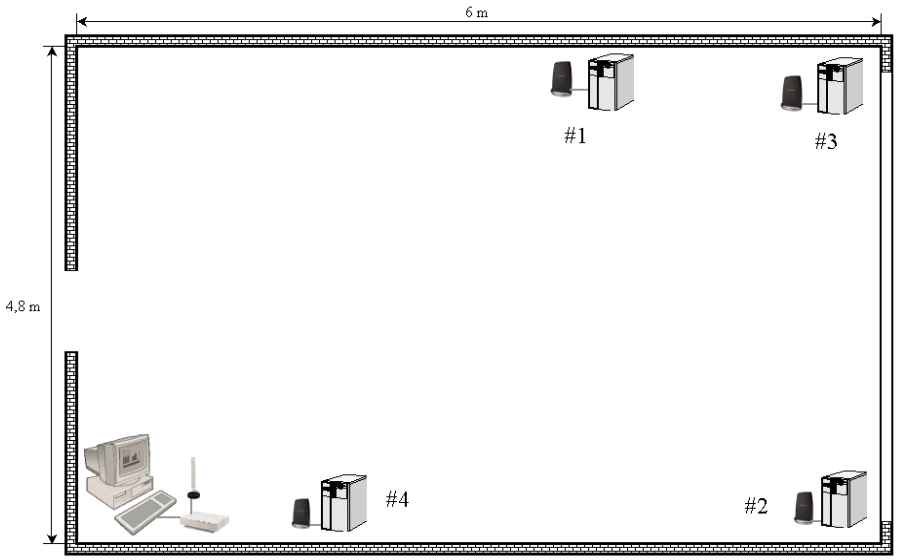


Abb. 5.2 : Aufbau des Wireless LAN

5.2 Messungen

Zur Durchführung der Messungen wurden zunächst fertige MPEG4-Datenströme erzeugt. Dies war notwendig, um alle Testreihen mit den selben Daten durchführen zu können. Bei Einsatz der Kameras und Übertragung von Livebildern hätte jeder Stream eine andere Verteilung der Frame- und Paketgrößen gehabt. Darum war es sinnvoll, erst mit vorgefertigten Streams zu arbeiten und zum Schluß die Ergebnisse und das darauf angepasste Server-Client-System mit Livedaten zu testen.
Es wurden zwei YUV-Sequenzen der Straße Alt-Moabit aufgenommen und mit Bitraten von 500 kbits/s bis 3000 kbits/s in Schritten von 100 kbits/s kodiert :

Stream 1 : 640x460 @ 15Hz
4500 Frames
Dauer : 5 min
I-Frame-Periode : 15
Stream 2 : 640x480 @ 15 Hz
9000 Frames
Dauer : 10 min
I-Frame-Periode : 15

Abb. 5.3 : Testdaten Alt-Moabit (Stream 1)

Die Abbildung 5.4 zeigt für Teststream #1 die Häufigkeitsverteilung der Anzahl an RTP-Paketen pro Frame. Es sind zwei Maxima zu erkennen. Das erste umfaßt alle P-Frames, welche deutlich weniger

Pakete benötigen als I-Frames. Das zweite Maximum (I-Frames) befindet sich etwa bei dem 3 bis 5fachen des Ersten. Die Abbildung zeigt in gewisser Form die Wirkungsweise des Encoders. Er verteilt die Bitrate so auf I- und P-Frames, das die beiden Maxima nicht zu weit auseinander liegen. Dies hat praktische Bedeutung für Bildqualität und Übertragungsdauer. Die Übertragungsdauer von I-Frames ist dadurch nicht extrem höher ist als die von P-Frames. Die Bilder kommen gleichmäßiger beim Empfänger an, was die Synchronisation (Kapitel 4.3.4) erleichtert und die Belastung des Netzwerkes über der Zeit besser verteilt.

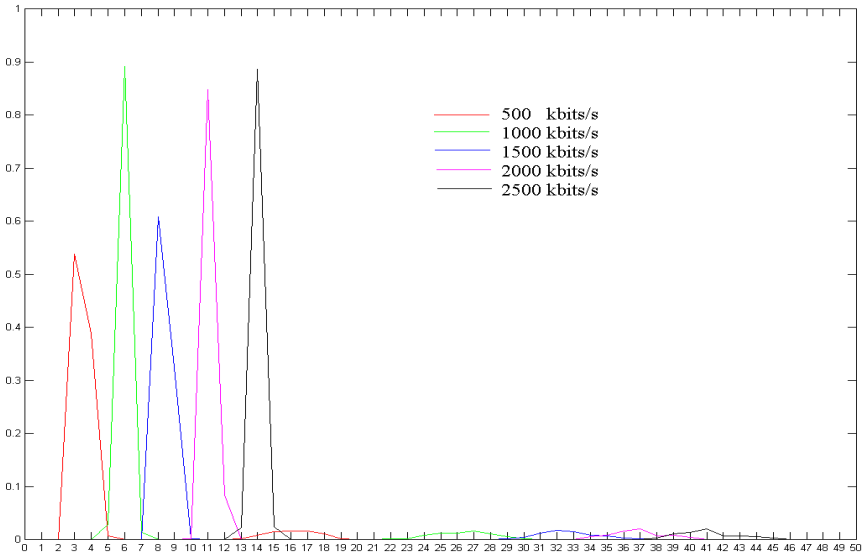


Abb. 5.4 : relative Häufigkeit von RTP-Paketen pro Frame

Die Messungen wurden bei jedem Aufbau mehrmals durchgeführt (i.A. 3fach). Zwischen den einzelnen Durchläufen lagen Pausen von wenigen Minuten bis zu Tagen. Von den dadurch verursachten Änderungen der Übertragungsbedingungen sind Veränderungen in der Anwesenheit von Personen und die Verschiebung von Mobiliar im Testraum und der Umgebung zu nennen, welche die Ausbreitung der Funkwellen beeinflußten. Die Auswirkungen waren besonders bei hohen Bitraten zu bemerken, wenn sich das WLAN an der Grenze seiner Übertragungskapazität befand.

5.3 Auswertung

Nachfolgend werden alle Testreihen beschrieben. Dazu werden jeweils ausgesuchte Ergebnisse in Diagrammform präsentiert und Schlußfolgerungen in Bezug auf Server- und Clientarchitektur gezogen. Ein Teil der Änderungen an Server und Client wurden sofort umgesetzt und gingen in spätere Messungen bereits mit ein.

Begonnen wurde mit nur einem Server. Im gesamten Bereich von 500 – 3000 kbits/s traten erwartungsgemäß keine Paketverluste auf. Aufgrund fehlender interner und externer Störeinflüsse, der sehr kleinen Kanalbitfehlerrate und der Möglichkeit der wiederholter Übertragung auf MAC-Ebene ist die Wahrscheinlichkeit für Paketverluste vernachlässigbar gering.

Im Gegensatz zu Fast Ethernet führt die geringere Bandbreite jedoch zu einer unregelmäßigen Bildanzeige. Während die kurzen P-Frames sehr schnell übertragen werden, kommt es bei den deutlich größeren I-Frames zu einer Streckung der Übertragung auf der unteren Schichten. Um trotzdem ein gleichmäßiges Rendern beim Client zu erreichen, muss der in Kapitel 4.3.4 beschriebene Synchronisations-Mechanismus aktiviert werden. Die dafür notwendige Eingangspuffergröße und die Gesamtverzögerung steigt mit zunehmender Bitrate.

Die ersten auswertbaren Beobachtungen wurden bei der Verwendung zweier Server in ca. gleichen Entfernungen zum Client gemacht. Die Server wurden in kurzen Abständen von weniger als 100ms gestartet. Dadurch fiel die Übertragung der regelmäßig auftretenden I-Frames beider Streams zeitlich zusammen. Da es auf MAC-Ebene zu einer Vermischung der Pakete kommt, werden beide I-Frames zusätzlich verzögert und die schon beschriebene unregelmäßige Bildanzeige verschlechtert sich weiter.
Durch die Möglichkeit von Kollisionen verlief die Übertragung nun auch nicht mehr problemlos. Vielmehr traten bereits bei geringen Bitraten vereinzelte Paketverluste auf, welche jedoch die Verluste ganzer Frames und teils starke Störungen nachfolgender P-Frames zur Folge hatten. Abb. 5.5 zeigt eine der Meßreihen. Um die zum Teil sehr große Dynamik der Fehlerraten zu zeigen, sind neben dem Mittelwerten auch Maxima und Minima dargestellt :

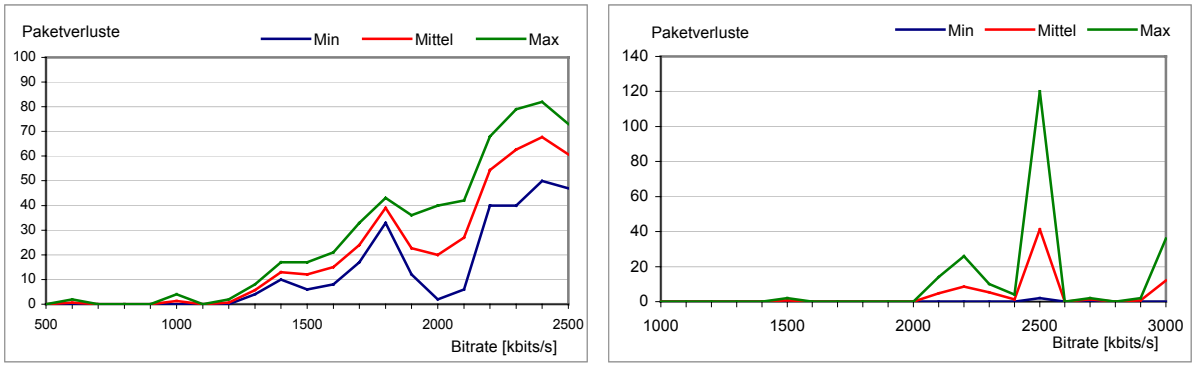


Abb. 5.5 : Paketverluste bei 2 Servern mit alten und neuen Treibern (Server #1)

In beiden Diagrammen sind die Absolutwerte dargestellt. Obwohl die relativen Verluste annähernd gleich blieben, sind die Absolutwerte entscheidend, da bereits einzelne Paketverluste zu Störungen führen. Der Verlauf der Frameverluste folgt daher näherungsweise der Kurve für Paketverluste. (siehe Kapitel A.1 und A.2 im Anhang). Bei der ersten Messung zeigten sich störende Paketverluste schon bei mittleren Bitraten. Bei 3 oder 4 Servern hätte dies bedeutet, das pro Stream nur eine Bitrate von einigen hundert kbits/s zur Verfügung steht. Da erfahrungsgemäß die Performance eines Netzes stark von den verwendeten Treibern abhängt, wurden versuchsweise bei allen Komponenten die neuesten Treiber (09/2002 statt 06/2001) installiert. Diese beinhalteten auch eine Erneuerung der Firmware in den USB-Adaptern sowie dem Access Point. Erneute Messungen zeigten danach deutlich bessere Ergebnisse (Abb. 5.5 rechts, Anhang A.1). Die Angaben des Herstellers geben aber keine Information darüber, welche Verbesserung für die Leistungssteigerung verantwortlich ist.

Bei einer einzelnen Messung mit 2500 kbits/s kam es zu einem Performanceeinbruch. Auch wenn die Gründe hierfür nicht genau spezifiziert werden können, zeigt dies aber die Schwierigkeiten bei der Vorhersage des Netzverhaltens. Wie auch bei späteren Tests beobachtet werden konnte, kann schon der Aufenthalt von Personen zwischen Sender und Empfänger bei hohen Bitraten kurzzeitig zu großen Verlusten führen. Insgesamt zeigten die ersten Messungen mit 2 Servern jedoch nur geringe Paketverluste bis zur Meßgrenze von 2 x 3000 kbits/s. In späteren Messungen wurden teilweise sogar gar keine Paketverluste beobachtet.

In einem weiteren Test wurde die Anfälligkeit des Systemes bei Signal-Rausch-Abständen kleiner als 30 dB untersucht. Zur Abschirmung wurde der Access Point mit einer Doppelschicht Aluminiumpapier umwickelt und die externe Antenne demontiert. Wie bei einem Faradayschen Käfig ist das Innere eines solchen Raumes theoretisch feldfrei. Im Versuch gelang es, die Signalstärke um ca. 30 dB zu senken. Die durch die USB-Adapter gemessenen SNR-Werte bewegten sich danach im Bereich von 20-30 dB und unterlagen teils starken Schwankungen. Der Abstand zum Client betrug für beide Server ca. 6m. Die Ergebnisse sind in folgender Tabelle aufgeführt:

	Server 1				Server 2			
Bitrate	Paketverluste		defekte Frames		Paketverluste		defekte Frames	
kbits/s	1.	2.	1.	2.	1.	2.	1.	2.
500	0	133	0	59	26	12	13	6
600	4	0	2	0	0	2	0	1
800	0	4	0	2	6	0	3	0
1000	0	0	0	0	12	426	6	194
1200	0	0	0	0	8	564	4	253

Tab. 5.1 : Paketverluste bei schlechter bis mittlerer Verbindung

Es zeigte sich, dass es auch bei geringen Bitraten zu großen Paketverlusten kommen kann. Besonders Hindernisse im Übertragungsweg spielten dabei eine große Rolle. Sie führten z.T. auch zu kurzzeitigen Verbindungsabrissen. Bei einer zweiten Meßreihe wurden die Server nahe an den Client gestellt (ca. 1m), sodass keine Personen oder Gegenstände die Übertragung stören konnten. Erwartungsgemäß kam es dadurch zu recht stabilen SNR-Werten und kaum Paketverlusten. Für das Design von WLAN-Überwachungs-Netzen bedeutet dies, das eine Übertragung auch bei geringer Signalleistung (≈20dB) möglich ist, solange es nicht zu weiteren kurzzeitigen Absenkungen durch äußere Störeinflüsse oder Veränderungen der Antennenausrichtung kommt. Ein hoher Montageort bei Indoor-Netzen und die feste, optimale Installation der Antennen ist also zwingend notwendig.
Bei einigen Probemessungen im Bereich von 500-1100 kbits/s mit noch besser abgeschirmten Client (SNR<20 dB) wurden stark schwankende Verlustraten gemessen. Teilweise kam es sogar zum Verbindungsabbruch ohne offensichtliche Störquellen.

Nach Aufnahme der ersten Messungen stellt sich die Frage, ob die beobachteten Störungen noch akzeptabel sind. Da die Anwendung für Überwachungszwecke und möglicherweise als erste Stufe eines Systemes zur Erfassung statistischer Verkehrsdaten eingesetzt werden soll, müssen hierbei strenge Maßstäbe angesetzt werden. Danach sollte die vollständige Übertragung des Bildmaterials

wichtiger sein als die dabei möglicherweise entstehenden größeren Verzögerungen. Bei einer Observation eines Geländes ist es z.B. weniger entscheidend, ob die Bilder 1-2 Sekunden später beim Wachpersonal eintreffen. Vielmehr dürfen möglichst keine Bilder fehlen oder gestört sein, damit eindringende Personen nicht zufälligerweise übersehen werden.
Aus diesem Grunde wurde das bereits in Kapitel 4.3.1 beschriebene System zur Paketwiederholung implementiert. Damit können fast alle Paketverluste aufgefangen werden, sofern noch genügend Bandbreite für die negativen Bestätigungen (NACKs) und die wiederholten Pakete zur Verfügung steht. Die dadurch allerdings verursachten Verzögerungen sind u.a. vom Netz, den Bitraten und den eingestellten Werten für Round-Trip-Time abhängig. Mit Hilfe der ebenfalls implementierten Paketverlustsimulation wurden für das Test-WLAN folgende Werte bestimmt.

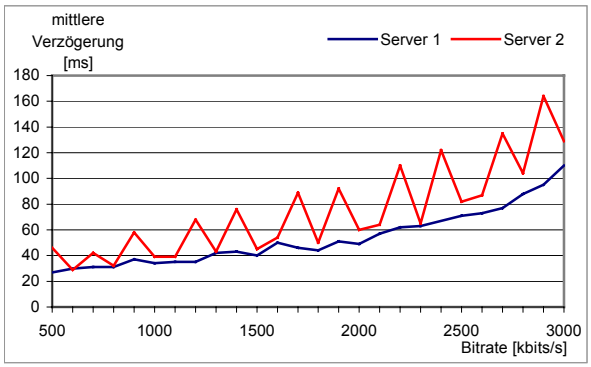


Abb. 5.6 : Verzögerungszeiten im WLAN bei Paketverlusten

Die Verzögerungszeit ergibt sich aus der Differenz der Zeiten zwischen der ersten Aufforderung zur Wiederholung (1. NACK) und dem Empfang des möglicherweise mehrmals wiederholten Paketes. Sie steigt bei zunehmender Netzbelastung an, liegt aber noch in für eine Überwachungsanwendung akzeptablen Bereichen. Wie u.a. in Abb. A.4 im Anhang zu sehen ist, können bis auf vereinzelte Pakete alle empfangen werden. Läßt man noch mehr Verzögerung zu, so wäre ein (fast) vollständiger Empfang in einem weiten Bitratenbereich möglich.

Die Ergebnisse der Messungen mit 2 Servern zeigen, das der maximale Datendurchsatz des WLANs noch nicht erreicht wurde. Untersuchungen ([19]) haben ergeben, das bei zunehmender Netzbelastung der Durchsatz bis zu einem Maximum näherungsweise linear steigt und im weiteren Verlauf konstant bleibt. Für eine UDP-Echtzeit-Übertragung bedeutet dies, das ab diesem Punkt der Sendepuffer überläuft. Im vorliegenden RTP-Server ist dies ein dem Socket-Puffer vorgeschalteter Ringpuffer. Wenn nicht mehr genügend Paketpuffer frei sind, muss das aktuelle Frame an dieser Stelle komplett verworfen werden. Neben den durch Paketverluste defekten Frames müssen also auch diese beachtet werden. Der Verlust dieser Frames wird nicht im RTP-Client erkannt, da hier nur die RTP-Sequenznummern ausgewertet werden, nicht jedoch die Zeitstempel. Der Vorteil von Verlusten durch Pufferüberlauf im Gegensatz zu Netzverlusten ist jedoch, das dies bereits vom Server erkannt wird. Theoretisch könnte dieser also schon durch Bitratenreduzierung gegensteuern. Allerdings haben die Server im Gegensatz zum Client keinen Überblick über die anderen Server. Aus

diesem Grunde wurde nur ein einfacher Mechanismus implementiert, welcher das Netz vor Überlastung schützen und die für den Betrachter sichtbaren Störungen der Videos vermindern soll:

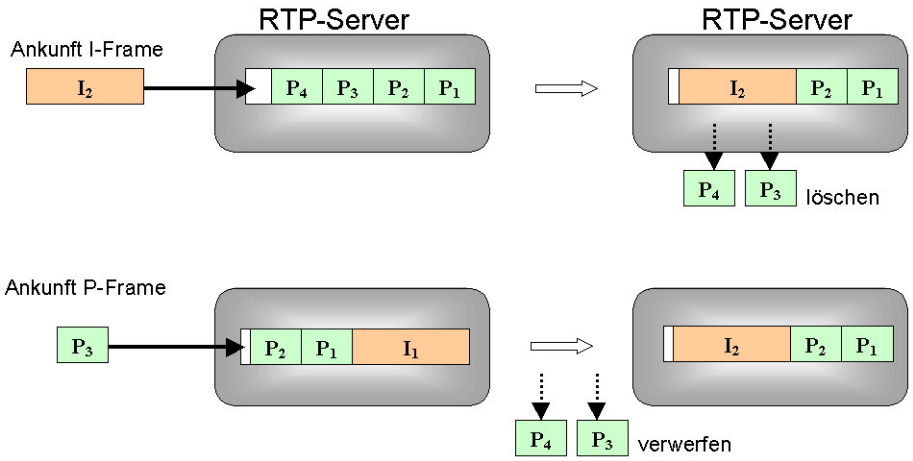


Abb. 5.7 : Verwerfen von Frames bei Pufferüberlauf

Reicht die Anzahl an freien Paketpuffern nicht mehr für das aktuelle Frame, so müssen Frames verworfen werden. Ist das aktuelle Frame ein I-Frame, so werden die letzten, bereits gespeicherten, P-Frames gelöscht, bis genügend Platz frei ist. Handelt es sich jedoch um ein P-Frame, so wird es selbst und alle nachfolgenden P-Frames verworfen. Somit werden im Überlastfall unter Umständen nur die wichtigen I-Frames und ggf. einige nachfolgende P-Frames übertragen. Die echte Bitrate der betroffenen Server wird kurzzeitig gesenkt. Der Client erhält aber trotzdem durch Auswertung der Zeitstempel Kenntnisse über diese Verluste und kann entsprechend reagieren.

Die ersten Meßreihen mit 3 Servern wurden vorerst jedoch ohne diese Steuerung aufgenommen.
Wie in Abb. 5.8 und Anhang A.3) zu sehen ist, kommt es ab ca. 3x 1500 kbits/s zu steigenden Paketverlusten. Bei der Messung wurde weiterhin beobachtet, dass die Streams ab ca. 3x2100 kbits/s nicht mehr mit der eingestellten Rate von 15 Hz versendet werden. Hier ist also die Sättigung erreicht, das Netz kann nicht mehr als ca. 6300 kbits/s übertragen. Vor weiteren Messungen wurden die Server allerdings so umgestellt, dass sie zwangsweise mit 15 Hz senden, auch wenn dies serverseitige Frameverluste bedeuten kann.

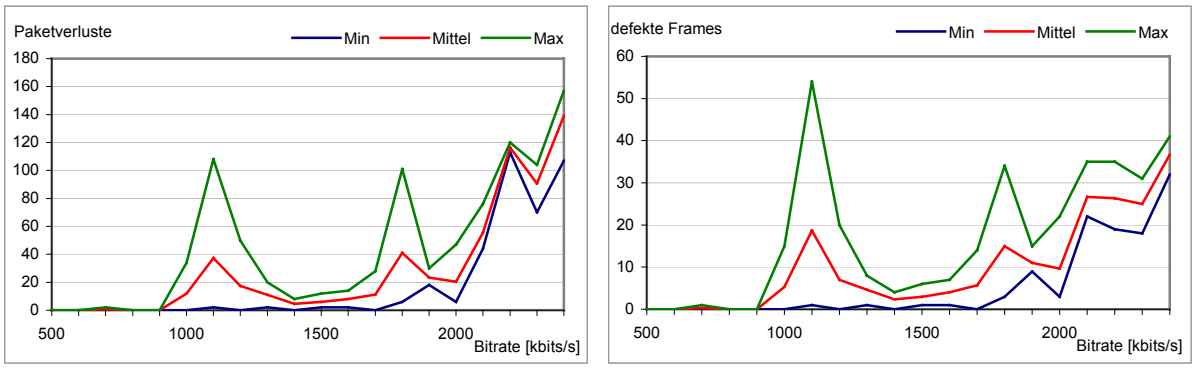


Abb. 5.8 : Paketverluste und defekte/verlorene Frames bei Server #3

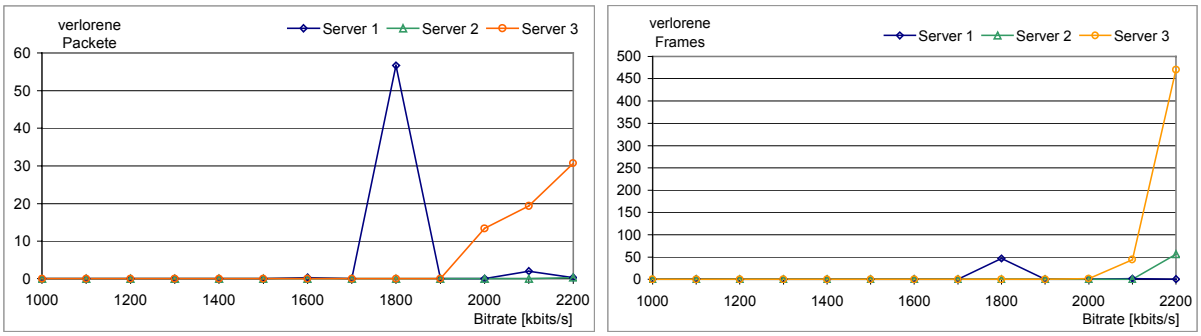


Abb. 5.9 : Verlorene Pakete/Frames bei 3 Servern und aktivierter Paketwiederholung

Die weiteren Versuche wurden zunächst ohne die beschriebene Steuerung der Serververluste in Abhängigkeit vom Frametyp durchgeführt. In Abb. 5.9 ist das Ergebnis zu sehen. Hier wurden vermißte Pakete zudem neu angefordert. Das linke Diagram zeigt die endgültig verlorenen Pakete nach zweimaliger Wiederholung an. Wie auch in Anhang A.4) zu erkennen ist, treten die Verluste nicht gleichmäßig bei allem Servern auf. Vielmehr gibt es Benachteiligungen bestimmter Server. In diesem Fall ist es Server 3, bei dem nach Erreichen des Sättigungspunktes verstärkt Frames durch Pufferüberlauf auf Serverseite verloren gehen. Die anderen beiden Server können hingegen mit voller Bitrate weitersenden. Für die Steuerung aller Server durch den Client ist dies von Bedeutung. Der Client steht vor der Wahl, ob er in diesem Fall nur die Bitrate des betroffenen Servers oder aller Server zurückfährt. Diese Entscheidung ist nicht einfach, denn zunächst muss der Client zwischen den beiden Ursachen für hohe Frameverluste unterscheiden: Frameverluste durch Paketverluste oder durch Pufferüberlauf beim Server. Er muss also die statistischen Daten auswerten. Liegt die Frameverlustrate über der Paketverlustrate, so gehen auf jeden Fall schon Frames beim Server verloren. Dieser Fall tritt in der Praxis aber kaum auf. Vielmehr kann es in diesen Bitratenbereichen zu hohen Paket- und Frameverlusten kommen. Wie in Abb. 5.9 zu sehen ist, muss dies ebenfalls nicht alle Server gleichermaßen betreffen. Eine Entscheidung, welche Ursachen für verlorene Frames verantwortlich sind, ist allerdings einfach zu treffen, wenn man beim RTP-Client die verlorenen Frames mitzählt. Diese Zählung erfolgt beim Löschen von Paketen, welche aufgrund fehlender Pakete nicht zu einem VOP zusammengefasst werden können. Durch Auswertung der Timestamps kann die genaue Anzahl der gelöschten VOPs ermittelt werden, sofern mindestens ein Paket pro VOP empfangen wurde. Ein Vergleich mit der Anzahl fehlender VOPs beim Decoder, welcher die Gesamtverluste erkennt, ergibt die Zahl der beim Server verworfenen Frames.

Zur genauen Auswertung der Verluste kann der Client auch die Frameverluste des Servers über den RC-Kanal (Kap. 4.2) abfragen. Die Differenz zwischen den gezählten Verlusten auf Client- und Serverseite ergibt die Anzahl an durch Paketverluste defekten Frames. Der Vorteil dieser Methode ist, dass alle Serververluste erfasst werden. Der Nachteil ist allerdings die problematische Abfrage über eine TCP-Verbindung im Falle eines überlasteten Netzes. Die zur Auswertung benötigten Daten können mitunter nicht rechtzeitig ermittelt werden. Daher wurde für das Stream Control Modul die erste Variante gewählt (Kapitel 5.4).

In weiteren Messungen mit drei Servern wurde die Sättigungsgrenze des WLANs genauer bestimmt. In den ersten Versuchen wurde die Gesamtbitrate in Schritten von 300 kbits/s erhöht. Nun werden 2 Server auf 2500 kbits/s fest eingestellt und nur der Dritte schrittweise erhöht. Die Ergebnisse basieren also auf einer Genauigkeit von 100 kbits/s und unterstreichen die bisherigen Auswertungen:

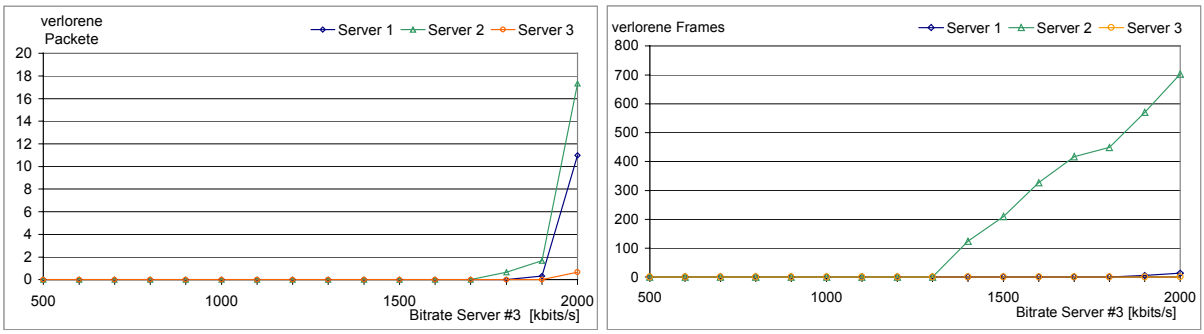


Abb. 5.10 : Verluste bei zwei festen und einem variablen Server (#3)

Auch bei diesen Messungen wurde die Paketwiederholung auf Anwendungsebene aktiviert. Das Ergebnis ist eine fehlerfreie Übertragung bis 1300 kbits/s (Gesamtbitrate : 6300 kbits/s). Ab diesem Wert traten Frameverluste durch Pufferüberlauf bei Server 2 auf. Aber erst bei höheren Bitraten konnten auch Paketverluste nicht mehr verhindert werden. Nach Abschluß eines Einzeltests wurden vom Client auch die gemessenen Bitraten ausgegeben. Eine Addition dieser Werte ergab einen maximalen Datendurchsatz von 6300 – 6400 kbits/s.
Die vollständigen Ergebnisse sind in Anhang A.5) zu finden. Dort sind auch Meßreihen zur mittleren Paketverzögerung aufgeführt. Diese liegt zwangsläufig immer unterhalb der vordefinierten Grenze von 500 ms, da die verspäteten Pakete nicht mehr in die Berechnung eingehen. Sie nähert sich daher nur mit zunehmender Bitrate der Grenze an, ohne sie zu erreichen.

In den letzten Untersuchungen wurden alle 4 Server betrieben. Es zeigte sich eine deutlich höhere Fehleranfälligkeit des Systemes im Vergleich zu den vorherigen Messungen. Die maximale Gesamtbitrate liegt mit ca. 5600 kbits/s unter der mit 3 Servern. Von da an traten erhöhte Frameverluste bei einem oder mehreren Servern auf. Die höhere Kollisionswahrscheinlichkeit verursacht deutlich mehr vermißte Pakete schon bei niedrigen Bitraten. Diese können nur z.T. aufgefangen werden. Wie zu beobachten war, ist auch die Anfälligkeit gegenüber Fremdstörungen (Personen) größer. Die Ergebnisse im Anhang A.6) zeigen, das eine Übertragung mit wenigen Verlusten (siehe Minimumwerte) bis in die Nähe der Sättigung möglich ist. In anderen Fällen traten jedoch auch schon bei niedrigen Bitraten Verluste auf. Wie in der Abb. 5.11 zu sehen ist, kommt es bei zwei der vier Server zu deutlichen Störungen in Einzelfällen, während bei den anderen die Verlustraten über einen weiten Bereich sehr niedrig sind. Die Verluste bei diesen Servern könnten aber gesenkt werden, wenn man die betroffenen Pakete öfter wiederholen würde.

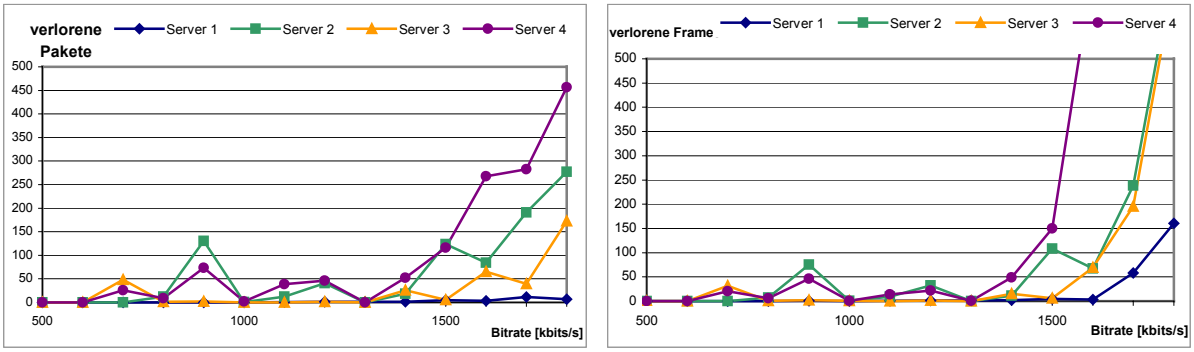


Abb. 5.11 : Paket-/ Frameverluste bei 4 Servern

Des Weiteren könnten Optimierungen in den MAC-Algorithmen weitere Performanceverbesserungen bringen. Besonders die Anzahl an Retransmissions auf MAC-Ebene und der in Kapitel 2.4.2 erwähnte Auto-Fallback-Mechanismus spielen dabei eine wichtige Rolle. Wiederholungen von unbestätigten MAC-PDUs können schneller erfolgen als die Anforderung auf Anwendungsebene, bei der zusätzlich noch Bandbreite und Rechenzeit für die Paketanforderung benötigt wird. Die maximale Anzahl an MPDU-Wiederholungen liegt bei den meisten WLANs bei 4 und ist selten vom Benutzer einstellbar. Hinzu kommt noch, dass die zweite und weitere Wiederholungen in einem langsameren Übertragungsmodus erfolgen. Das WLAN schaltet vom 11-Mbit-Modus auf 5,5, 2 und ggf. 1 MBits/s zurück, da es von einer schlechten Signalleistung oder Problemen durch Mehrwegeempfang ausgeht. In diesen Fällen ist die Übertragung mit niedrigerer Bitrate sinnvoll. Durch eine andere Signalmodulation steigen die Chancen auf einen korrekten Empfang. Zum Beispiel liegt beim Mehrwegeempfang die Grenze für 1 % Paketverluste im 11-MBit-Modus bei 65 ns Laufzeitdifferenz, während sie im 5,5 MBit-Modus bei 225 ns liegt (gültig für das verwendete WLAN-Equipment).
Beim Zurückschalten des Übertragungsmodus‘ wird die Möglichkeit von Kollisionen aber nicht in Betracht gezogen, obwohl eine Auswertung theoretisch denkbar wäre. Dazu müßten die Geräte ständig des Netz analysieren und zudem untereinander Informationen über SNR-Werte, Stationen im Netz, verlorene Pakete usw. austauschen, was allerdings Bandbreite kostet und zudem teure Erweiterungen der Hardware erforderlich macht. Eine praktikable Zwischenlösung wäre die Möglichkeit der Deaktivierung des Fallback-Mechanismus‘ durch den Benutzer.

5.4 Strategien für die Streamkontrolle

Nach den Untersuchungen des WLANs folgt die Umsetzung der Ergebnisse in geeignete Strategien zur Kontrolle der Videostreams. Dazu muss festgelegt werden, wie der MultiVideoClient auf bestimmte Ereignisse reagieren soll. Die Auswirkung dieser Ereignisse können einen Wechsel des MVC von einem Zustand in einen anderen bedeuten. Ziel der Messungen und Auswertungen war es, für das WLAN eine begrenzte Menge an möglichen Zuständen zu definieren. Im einfachsten Fall gibt es davon zwei : "Verlustfrei" und "Verlustbehaftet". Im ersten Zustand läuft die Übertragung fehlerfrei, im zweiten kommt es zu mehr oder weniger großen Verlusten. Die Grenze zwischen den beiden Zuständen wäre die Sättigungsgrenze, ab der bereits beim Server Frameverluste auftreten. Bis zu

diesem Punkt werden alle Paketverluste durch Wiederholung aufgefangen. Dieses Modell wäre leicht zu handhaben. Leider ist dies bei steigenden Verlustwahrscheinlichkeiten aufgrund höherer Bitraten nur zu erreichen, indem man größere Verzögerungszeiten zuläßt. An der Sättigungsgrenze geht die Verzögerung dann gegen unendlich, da es keine Übertragungskapazität für Paketwiederholungen mehr gibt.
Nach Auswertung der WLAN-Tests und Überarbeitung des Server-Client-Systemes wurden darum drei Zustände definiert. In der folgenden Abbildung sind diese dargestellt:

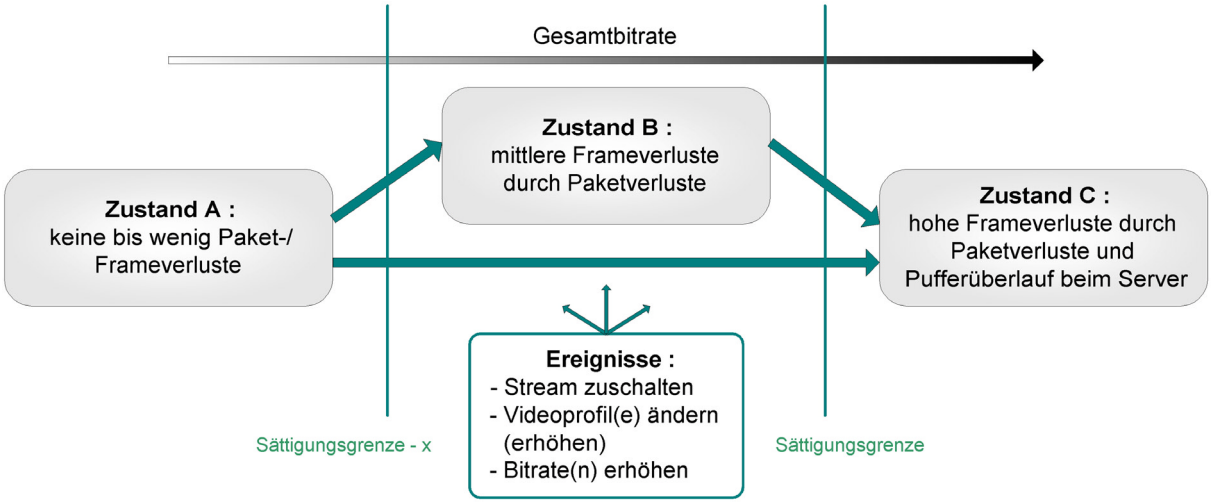


Abb. 5.12 : Verschlechterung der Übertragung bei steigenden Bitraten

Im Zustand A können durch wiederholte Übertragung der Pakete auf Anwendungsebene bis auf wenige Ausnahmen alle Pakete empfangen werden. Die durch die Wiederholung entstehenden Verzögerungen sind akzeptabel. Dieser Zustand gilt für einen weiten Bitratenbereich. Im Zustand B kommt es zu stark steigenden Verlusten, da viele vermißte Pakete nicht mehr rechtzeitig wiederholt werden können. Der Bitratenbereich für B ist relativ eng. Er kann durch Zulassen größerer Verzögerungszeiten bis nahe 0 verkleinert werden.
Im letzten Zustand C kommt es schon bei einem oder mehr Servern zum Stau. Dort müssen Frames verworfen werden. Außerdem können nur wenige der vermißten Pakete wiederholt werden, da es kaum Bandbreite dafür gibt.
Der Wechsel in höhere Zustände erfolgt bei steigender Gesamtbitrate oder Störungen im Netz. Ziel des Clients muss es sein, das System stabil im Zustand A zu halten. Bei Benutzersteuerung kann der Client dies im Voraus berechnen, bei Störereignissen kann er allerdings nur reagieren.

Die Kontrolle der Streams wird mit Hilfe des Stream Control Modules (SCM) durchgeführt. Beim Design des Modules wurden die Ereignisse in die beiden Kategorien “extern“ und “intern“ unterteilt. Zur ersten Gruppe gehören alle Aktionen, die von der externen Anwendung, bzw. dem Benutzer, kommen und Änderungen der Bitrate eines Streams zur Folge haben. Das sind demnach Starten/Stoppen eines Streams, Änderung des Videoprofiles oder der Bitrate eines laufenden

Streams. Das Ziel einer Streamsteuerung ist in diesen Fällen die Überschreitung einer vorgegebenen maximalen Gesamtbitrate für das Netz und somit das Auftreten von Frameverlusten zu verhindern.
Zur zweiten Kategorie gehören alle Störungen des Netzes. Die dadurch verursachten Bildstörungen können vom Client nicht verhindert werden. Das Stream Control Modul muss Paket- und/oder Frameverlustraten überwachen, auswerten und mit einer geeigneten Reduzierung der Bitraten reagieren.

In der folgenden Tabelle sind die Reaktionen der Streamkontrolle auf Benutzervorgaben aufgeführt. Die Bitraten der Streams werden dabei in Abhängigkeit einer vorgegebenen Priorität geregelt. Bitraten von Streams der Priorität 3 dürfen durch das Kontrollmodul weder gesenkt noch erhöht werden. Bei Priorität 2 darf die Bitrate um maximal N% gegenüber der definierten Profilbitrate (siehe Anhang F) gesenkt werden. N ist dabei ein konfigurierbarer Wert (Anhang E) im Bereich zwischen 1 und 25. Streams der Priorität 1 dürfen maximal um 2N% gesenkt werden. Eine Bitratenerhöhung bei Nichtauslastung des Netzes ist generell nur bis zur Profilbitrate möglich.
Bei den Tests wurde N=10% eingestellt. Bei der Benutzung größerer Werte muss man die Qualitätsminderung der Videobilder beachten. Durch verstärkte Blockartefakte ist eine Objekterkennung oder Bewegungsdetektion eventuell nicht mehr möglich. Letztendlich ist die Grenze für N von der eingestellten Profilbitrate und den Benutzeranforderungen abhängig.
Die Vergabe von Prioritäten soll es dem Administrator eines Überwachungsnetzes bei beschränkter Bandbreite ermöglichen, in wichtige und weniger wichtige Videos zu unterteilen.

Benutzeraktion	**Reaktion des MultiVideoClients / SCM**
- Starten eines neuen Streams	Berechnen der neuen Gesamtbitrate Bei Überschreitung der maximalen Gesamtbitrate wird überprüft, ob durch Bitratenreduzierung der anderen Streams sowie des neuen Streams die Grenze eingehalten werden kann. Ist dies der Fall, so werden alle möglichen Bitraten gesenkt und der neue Stream gestartet.
- Stoppen eines Stream	Der Stream wird gestoppt. Die freigewordene Bitrate wird nicht auf die anderen Streams verteilt.
- Wechsel eines Videoprofiles in ein anderes mit höherer Bitrate - direkte Erhöhung der Bitrate bei einem laufenden Stream	Überprüfung und Änderung der Bitraten wie beim Starten eines neuen Streams, allerdings mit dem Unterschied, das nur die Bitraten der anderen Streams geändert werden dürfen
- Wechsel eines Videoprofiles in ein anderes mit geringerer Bitrate - direkte Senkung der Bitrate bei einem laufenden Stream	Das Videoprofil bzw. die Bitrate wird geändert. Die freigewordene Bitrate wird auf die anderen Streams der Priorität 1 und 2 verteilt, wobei deren Profilbitrate nicht überschritten wird. Die Verteilung kann durch den Benutzer unterbunden werden.

Tab. 5.2 : Reaktionen des MVC auf Benutzeraktionen bei aktiviertem Kontrollmodul

In einem Zwischenmodus wird die neue Gesamtbitrate durch das Kontrollmodul zwar geprüft, die Bitraten der Streams bleiben aber unverändert. Bei einem drohenden Überschreiten der Obergrenze wird die Aktion des Benutzers mit einer Fehlermeldung abgebrochen, d.h. ein neuer Stream kann z.B. nicht gestartet werden. Auch im Falle einer Bitratensenkung eines Streams bleiben die anderen unverändert.
Bei deaktiviertem Kontrollmodul erfolgt keine Überprüfung. Dies ist aber nur dann sinnvoll, wenn ein schnelles Netz mit genügend Bandbreite zur Verfügung steht.

Der zweite Aspekt der Streamsteuerung betrifft die Frameverluste. Bei einem wireless LAN kommt es beim Betrieb mehrerer Server unweigerlich zu Paketverlusten. Können diese jedoch durch wiederholte Übertragung auf nahe Null gesenkt werden, so stellt dies kein Problem dar. Die dadurch auftretenden Verzögerungen muss der Client nur durch geeignete Pufferung ausgleichen. Wie die Untersuchungen zeigten, kann es jedoch zu zwei möglichen Ereignissen kommen, welche eine Reaktion durch den Client erforderlich machen.
Zum einen muss die vorkonfigurierte Maximalbitrate nicht mit der tatsächlichen übereinstimmen. Beim Zuschalten der Streams kommt es dann zu hohen Paket- und Frameverlusten, da das Netz trotz präventiver Kontrolle durch das SCM überlastet wird (Zustand B oder C). Ursachen für eine falsche Bitratengrenze können z.B. ungenaue Messungen oder eine Veränderung von Umgebungsbedingungen (Wetter, Hindernisse, Antennenausrichtung, Störsignale) sein. In einem solchen Fall reagiert der Client wie folgt :

- Zählung der Frameverluste von RTP-Client und Decoder und regelmäßige (n x 10sec) Weitergabe der Daten an das Stream Control Modul
  Paketverluste spielen keine Rolle. Die Frameverluste werden getrennt in Stauverluste beim Server (L_{Server})und Netzverluste durch verlorene Pakete (L_{Netz}). Letztere Anzahl wird beim RTP-Client erfasst, der Decoder zählt die Gesamtverluste :

$$L_{Total} = L_{Decoder} = L_{Netz} + L_{Server} \qquad \text{mit } L_{Netz} = L_{RTPClient} \tag{5.1}$$

$$\rightarrow \qquad L_{Server} = L_{Decoder} - L_{RTPClient} \tag{5.2}$$

- Auswertung der Daten durch das SCM
  Liegen die Verluste über einem Grenzwert (gemessen in Verluste pro Zeiteinheit) so müssen die Bitraten der Streams gesenkt werden. Dazu werden die Gesamtbitraten prozentual gesenkt und für alle Streams eine neue Bitrate nach den bereits beschriebenen Prioritätskriterien berechnet. Die für die aktuelle Anzahl an laufenden Streams gültige maximale Gesamtbitrate kann aus den gemessenen Datenmengen der RTP-Clients berechnet werden. Diese übermitteln neben den Verlustraten auch die empfangene Streambitrate, welche kleiner oder gleich der Encoderbitrate ist. Eine Addition aller Streams ergibt dann den aktuellen Datendurchsatz des Netzes. Dieser wird aber nur hinzugezogen, wenn sich das Netz im Überlastfall befindet (Zustand C). Anhang A.7) bis A.9) zeigen Verläufe der Kurzzeitbitratenmessungen.
- Senkung der Bitraten je nach Zustand
  Die Grenze zwischen Zustand B und C wurde nach Auswertung der Meßergebnisse festgelegt. Überwiegen die Frameverluste auf dem Netz, so wird vom Zustand B ausgegangen,

andernfalls vom Zustand C. Die Messungen zeigen, dass der Bitratenbereich für B durch die aktivierte Paketwiederholung sehr klein sein kann. In einer Messung (Tab. 5.3) mit 3 Servern unter optimalen (störungsfreien) Übertragungsbedingungen war er nahe Null. Bei vier Servern ist dies jedoch schwerer einzuschätzen, da Frameverluste über einen weiten Bitratenbereich auftraten. Reduziert man diese Verluste durch mehr Paketwiederholungen, so muss man nur den Bereich nahe der Sättigungsgrenze betrachten. Hier wurde ein Bereich von ca. 800 kbits/s (4x200 kbits/s) bestimmt. Bezogen auf den maximalen Datendurchsatz für 4 Server von etwa 6 MBits/s sind das ca. 10-15%.

Auf Basis dieser Messung wurde eine Senkung der Gesamtbitrate um 10% festgelegt, wenn sich das Netz im Zustand B befindet. Auch die Grenzwerte der Gesamtbitraten werden um diesem Wert gesenkt, um eine Rückkehr in den fehlerbehafteten Zustand durch Benutzeraktionen zu verhindern. Für Zustand C wird der gemessene Datendurchsatz als Grundlage genommen. Er stellt die Grenze zwischen B und C dar, d.h. die Gesamtbitrate muss mindestens auf diesen Wert abzüglich ca. 10% gesenkt werden.

- Änderung der Videoprofile bei starker Bitratensenkung

  Bei starker Überlastung kann es vorkommen, das die geforderte Senkung der Gesamtbitrate nicht durch direkte Senkung der Einzelbitraten erreicht werden kann, da dazu die untere Bitratengrenze einzelner Streams unterschritten werden muss. In diesem Fall ist eine Änderung des Videoprofiles dieser Streams notwendig. Hierbei steht im Vordergrund, das die Bildqualität erhalten bleiben soll. Bei zu niedriger Bitrate kommt es zu starken Blockartefakten, welche eine Auswertung durch Betrachter oder Algorithmen erschweren. Die Umschaltung auf ein anderes Videoformat mit höherer I-Frame-Periode, geringerer zeitlicher oder räumlicher Auflösung ist darum sinnvoller. Ab wann aber eine Umschaltung erfolgt, kann der Benutzer durch Konfiguration der Prioritäten und Grenzwerte indirekt steuern.

	Messung 1						**Messung 2**					
Bitrate	**Server 1**		**Server 2**		**Server 3**		**Server 1**		**Server 2**		**Server 3**	
[kbits/s]	Netz	Server	Netz	Server	Netz	Server	Netz	Server	Netz	Server	Netz	Server
1900	0	0	0	0	0	0	0	0	0	0	0	0
2000	0	0	0	0	0	0	0	0	0	0	0	0
2100	0	0	0	1	0	0	0	0	0	0	0	12
2200	0	483	0	0	0	183	0	165	718	18	0	233
2300	0	301	0	438	0	438	Verbindungsabriß Server #2 (keine Daten)					

Tab. 5.3 : Frameverluste bei 3 Servern bei optimalen Bedingungen (Datendurchsatz ≤ 6300 kbits/s)

Neben dauerhaften Störungen durch Überschreiten von Bitratengrenzen, sind auch kurzfristige Beeinflussungen möglich. Dazu zählt z.B. die Senkung der Signalleistung oder Mehrwegeempfang durch temporäre Hindernisse. Der Client kann diesen Fall nur durch Langzeitauswertung ab dem Zeitpunkt der ersten Störung erkennen. Da er aber sofort reagieren muss, ist eine Unterscheidung in kurzfristige oder dauerhafte Störungen nur schwer möglich. Wie zu beobachten war, treten in diesen Fällen besonders Frameverluste durch Paketverluste auf dem Netz auf. Der Client geht also vom Zustand B aus, d.h. er reagiert mit einer Bitratensenkung um einen festgelegten Wert.

5.5 Test des Kontrollmodules

Nach Implementierung der automatischen Steuerung wurde das System auf Fast Ethernet und WLAN getestet. Zunächst durchlief das Kontrollmodul einige Tests zur automatischen Bitratenanpassung bei manuellen Änderungen der Bitrate oder des Videoprofiles. Außerdem wurde das Verhalten beim Starten und Stoppen einzelner Streams überprüft. Frameverluste flossen nicht in die Steuerung ein.
Die Konfiguration des Subnetzes enthielt folgende Daten :

Maximale Bitratenänderung pro Prioritätsstufe : 10%
maximale Anzahl an Streams : 3
Prioritäten der Streams : 1 (niedrig)
Default-Bitrate der Streams (Videoprofil 1) : 1300 kbits/s
maximale Gesamtbitrate bei 3 Streams : 3900 kbits/s

Zur Protokollierung aller Biratenänderungen erzeugt die Anwendung eine Log-Datei. Das Ergebnis einer ersten Untersuchung zeigt Abb. 5.13. Nach Start aller drei Streams wurden die Bitraten um die Profilbitrate herum verändert. Wie zu erkennen ist, übersteigt die Gesamtbitrate nie die festgelegte Grenze. Zum Beispiel wurde bei Server 1 zum Zeitpunkt 75 die Bitrate auf 120% erhöht (1560 kbits/s). Die Bitraten der beiden anderen Streams wurden vorher automatisch auf 90% (1170 kbits/s) heruntergeregelt. Zum Zeitpunkt 91 wurde Stream 2 auf 1040 kbits/s geändert. Da Stream 1 zu diesem Zeitpunkt schon über der Profilbitrate lag, erhöhte sich nur die Bitrate von Stream 3. Die Gesamtbitrate sank an diesem Punkt durch die Begrenzung von Stream 1 auf 3770 kbits/s.

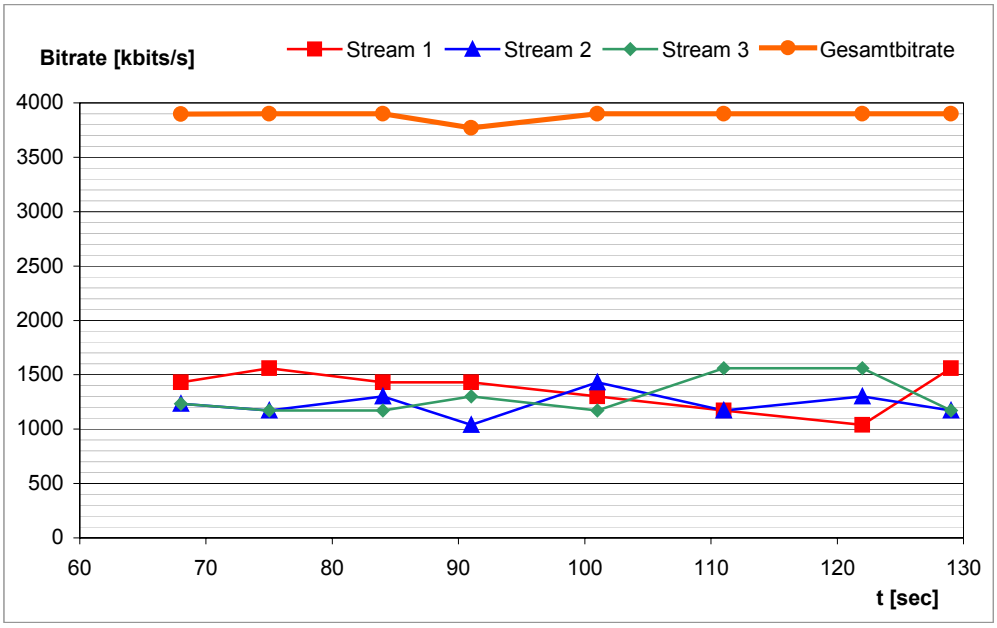


Abb. 5.13 : Ausbalancierung bei Bitratenänderungen

Bei einer drastischen Erhöhung eines Streams auf über 140% verweigerte der Client erwartungsgemäß die Änderung, da dazu die beiden anderen Streams die Grenze von 80% unterschritten hätten.

In den zweiten Versuchen wurde das Verhalten beim Start eines Streams und Änderungen der Videoprofile untersucht. Die Versuche beinhalteten aber auch eine direkte Bitratenänderung. Die maximale Gesamtbitrate bei drei Streams betrug hier nur 3700 kbits/s. Der Start eines dritten Streams mit 1300 kbits/s hätte also einen Überlauf bedeutet. Wie in Abb. 5.14. allerdings zu sehen ist, bewirkt das Starten von Stream 3 zum Zeitpunkt 46 eine Rückregelung aller Streams (auf 1232 kbits/s).
Bei einer Änderung des Videoprofiles von Stream 2 bei Zeitpunkt 55 mit einer neuen Bitrate von 1000 kbits/s, konnten die Bitraten der beiden anderen Streams wieder auf die Default-Rate erhöht werden. Die Gesamtbitrate betrug dann aber nur noch 3600 kbits/s. Die folgende Änderung des Profiles bei Server 3 bewirkte eine deutliche Absenkung der Gesamtbitrate, da die beiden anderen Bitraten nicht weiter erhöht werden konnten.

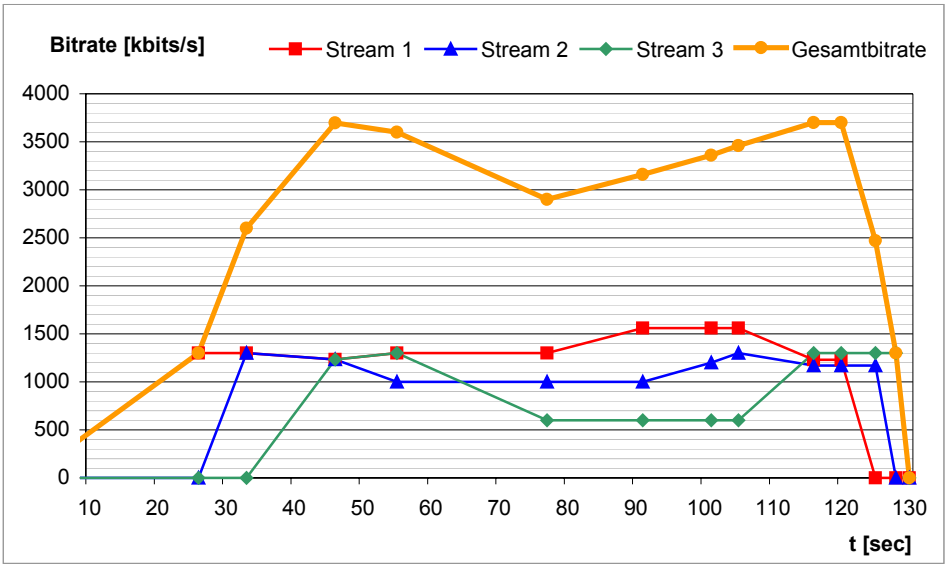


Abb. 5.14 : Ausbalancierung bei Streamstart, Videoprofil- und Bitratenänderung

Nach Implementierung der automatischen Verlustauswertung und Streamsteuerung mußte zunächst die Meßgenauigkeit der Kurzzeitbitratenmessung des RTP-Clients untersucht werden. Die Kurzzeitbitraten fließen in die Steuerung mit ein, wenn die Verluste eines Servers die Frameverluste auf dem Netz übersteigen (Zustand C). Die Zeiträume der Messungen betrugen jeweils 10 und 20 Sekunden. Dies sind auch die Abstände, in denen das Stream Control Module die Analyse durchführt. Die Ergebnisse im Anhang A.7) zeigen, das bei nicht ausgelastetem Netz die Kurzzeitbitraten sehr nahe am Mittelwert liegen. Dies gilt sowohl für Fast Ethernet als auch für Wireless LAN. Im Überlastfall für WLAN hingegen, bei dem ein Teil der beteiligten Servern Frames verworfen hat, kam es zu größeren Schwankungen.
Die Unterschiede zwischen den Streams sind dabei aber deutlich. Während z.B. die Standard-abweichung bei Server #3 (T = 10sec, B = 2200 kbits/s) nur 19,15 kbits/s beträgt, ist sie bei Server #1 56,33 kbits/s. Es fällt auf, dass sie mit größerer Abweichung des Streams von der vorgegebenen Bitrate ebenfalls ansteigt. Die Schwankungen sind aber nicht so stark, dass eine Einbeziehung der gemessenen Bitraten in die Analyse des Kontrollmodules zu ungeeigneten Ergebnissen führen würde. Diese Gefahr besteht nur, wenn die Konfiguration eine zu kleine Bitratensenkung (<< 10%) vorsieht.

In der aktuellen Version beträgt der Abstand der maximalen Gesamtbitrate zur berechneten Sättigungsgrenze 10%. Die in den Versuchen gemessenen Standardabweichungen bewegen sich hingegen im Bereich von 0 bis 5 % bezogen auf die jeweiligen Mittelwerte.

Vor den letzten Tests des Systemes auf Wireless LAN wurde zunächst die Funktionsweise der Streamkontrolle unter simulierten Fehlern auf Fast Ethernet untersucht. Beteiligt waren dabei 2 Streams, deren Prioritäten variierten. Durch Vorgabe verschiedener Frameverlustraten und gemessenen Bitraten wurden die beiden Zustände B und C eingestellt und die Reaktion der Streamkontrolle beobachtet. Es kam wie erwartet zur automatischen Absenkung der Bitraten durch direkte Bitratenänderung oder Änderung des Videoprofiles. Die Ergebnisse sind im Anhang A.8) tabellarisch aufgeführt. Die Tabellen enthalten Angaben zu Streambitraten, Profilen sowie den Bitratengrenzen des Subnetzes. Betrachtet werden die beiden Zeitpunkte vor und nach dem Auftreten der simulierten Frameverluste.

Die abschließenden Tests befassen sich mit dem Verhalten des Clients auf dem wireless LAN. Sie wurden mit 3 und 4 Servern durchgeführt. Die Bitraten bei 3 Servern betrug 2200 kbits/s, um die in früheren Messungen erhaltene Kapazitätsgrenze von ca. 6300 kbits/s zu überschreiten. Die Prioritäten aller 3 Streams wurden zunächst auf 1 (niedrig) eingestellt. Die Einstellung der maximalen Paketwiederholung auf 3 und der Round Trip Time auf 250 ms führte zu einer geringen Zahl verlorener Pakete. Der Bitratenbereich für den Zustand B war also sehr eng. Die weiteren Kenndaten für die ersten Messreihen sind :

Aktualisierungs-Periode des Kontrollmodules	: 10 sec
Bitratenabsenkung für Priorität 2 / 1	: -10% / -20 % unter der Profilbitrate
Prioritäten der Streams	: alle 1 (niedrig)
Signalstärke des WLANs	: excellent (40-60 dB)
Grenzwerte für verlorene Frames Server/Netz	: 3 / 3 Frames in 10 sec

Die maximalen Gesamtbitraten für 3 und 4 laufende Streams wurden mit 6600 und 6000 kbits/s höher eingestellt als die tatsächliche Grenze, da sonst das Kontrollmodul die Bitraten bereits beim Starten der Streams automatisch gesenkt hätte.

Die Ergebnisse der Tests sind in Anhang A.9) in Tabellen- und Diagrammform aufgeführt. Sie zeigen den Zeitverlauf der Bitraten und Frameverlusten. Nach dem Start oder zu einem späteren Zeitpunkt kam es zu Frameverlusten, welche die Grenzwerte überstiegen. Das Kontrollmodul reagierte daraufhin mit einer Absenkung der Bitraten in Abhängigkeit von den gemessenen Bitraten und den Streamprioritäten. Ein Beispiel zeigt Abb. 5.15. Hier kann die vorgegebene Bitrate von 2200 kbits/s bei Stream 1 und 2 nicht eingehalten werden. Der Server verwirft bereits einen Teil der Frames. Nach 17 Sekunden reagiert der Client ausgehend von den gemessenen Einzelbitraten mit einer Absenkung der Gesamtbitrate auf 5558 kbits/s. Da alle Streams die selbe Priorität besitzen, werden sie auch gleichermaßen gesenkt. Da die Senkung der Bitraten oder die Änderung der Videoprofile innerhalb

der nächsten Messzeitraumes stattfindet, können die in dieser Zeit gemessenen Bitraten und Verluste nicht für die nächste Auswertung verwendet werden. Das Streamkontrollmodul muss nach einer Steuerung mindestens zwei Runden, in diesem Fall also 2x10 Sekunden, warten bis es wieder verwertbare Daten bekommt. Wie in der Abbildung zu sehen ist, reicht dies aber nicht immer aus. Nach Absenkung der Bitraten sind die Sendepuffer einzelner Server immer noch überfüllt und können z.T. nicht schnell genug verarbeitet werden. Es kommt zu weiteren Verlusten in den nachfolgenden 20 Sekunden. Dies führt zu einer weiteren Absenkung der Bitraten nach 37 Sekunden auf 3x1640 kbits/s. Diese Absenkung wäre aber nicht notwendig gewesen, da die Frameverluste bereits rückläufig waren. Die Latenzzeit des Kontrollmodules wurde daraufhin auf 20 Sekunden erhöht. Nach einer Steuerung der Streams wartet das Modul nun also länger (3x10sec), bis es neue Auswertungen vornimmt.

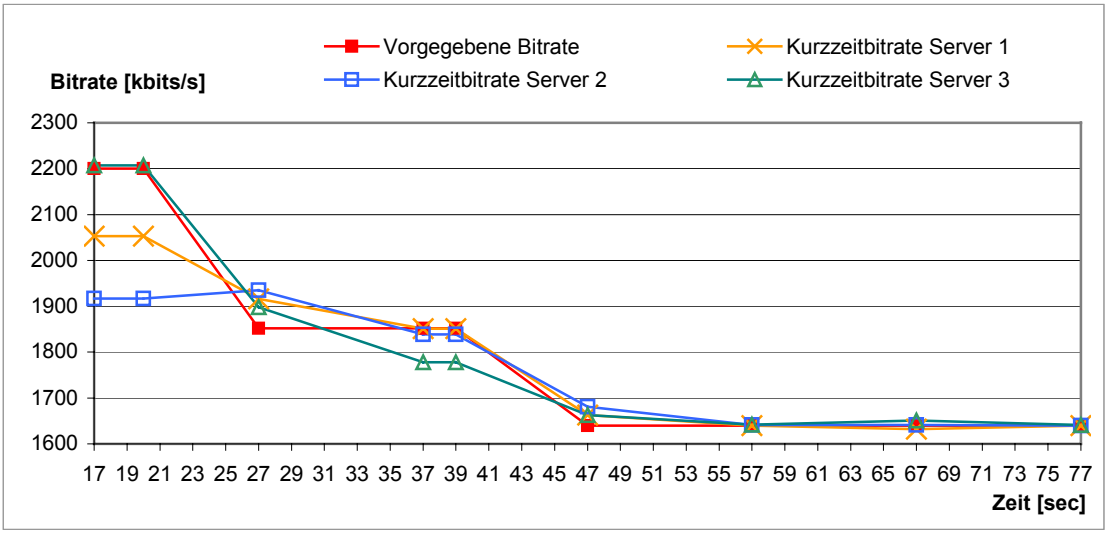


Abb. 5.15 : Anpassung der Bitraten bei Frameverlusten bei 3 Servern auf WLAN

Während bei den Tests mit 3 Servern hauptsächlich die Serververluste eine Rolle spielten, kam es bei 4 Servern teilweise auch zu hohen Verlusten von Paketen auf dem Netz. Bei den Untersuchungen wurden die Bitraten der Streams auf 1300, 1500 und 1700 kbits/s eingestellt. Die Kapazitätsgrenze des Netzes beträgt etwa 5600 kbits/s. Im Gegensatz zu 3 Servern ist der Bitratenbereich für Zustand B mit einigen hundert kbits/s aber größer. Wie die ersten Untersuchungen zeigten, kann es auch bei mittleren Bitraten zu vereinzelten Frameverlusten kommen. Um eine automatische Steuerung in diesen Fällen zu verhindern, dürfen die Grenzwerte für Frameverluste nicht zu niedrig, die Anzahl an zugelassenen Paketwiederholungen sollten dagegen so hoch wie möglich sein. Grenzwerte von 1 oder 2 Frames pro Intervall von 10 sec sind nicht angemessen. Es zeigte sich aber, dass auch ein recht hoher Wert von 5 verlorenen Frames/10sec nicht unbedingt ausreicht. Wie in Abbildung 5.16 zu sehen ist, wird die Bitrate der 4 Servern kurz nach dem Start aufgrund hoher Frameverluste von 1700 kbits/s auf 1348 kbits/s reduziert. 60 Sekunden später kam es aber bei Stream 4 zu weiteren Verlusten, welche zu einer erneuten Absenkung aller Bitraten um den vorgegebenen Wert von 10% führte. In den weiteren Minuten wurden dann keine Verluste mehr beobachtet.

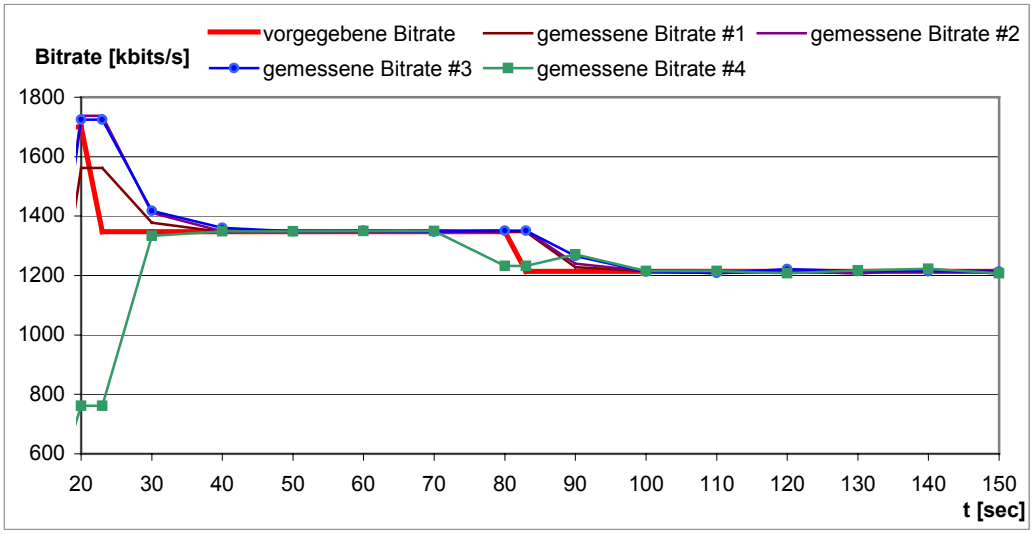


Abb. 5.16 : Bitratenanpassung bei 4 Servern auf WLAN

Bei einem letzten Test mit unterschiedlichen Streamprioritäten kam es wie gewünscht zu einer differenzierten Steuerung. Während die Bitrate von Stream 3 mit hoher Priorität unverändert auf 1700 kbits/s blieb, änderten sich die Bitraten von Stream 2 (mittlere Priorität) auf 1359 kbits/s, sowie von Stream 1 und 4 (niedrige Priorität) auf 1170 kbits/s. Die Gesamtbitrate betrug somit 5400 kbits/s.

Die Untersuchungen des MultiVideoClients auf dem wireless LAN demonstrieren die Funktionsfähigkeit des integrierten Modules zur automatischen Streamkontrolle. Der Client kann bei auftretenden Frameverlusten innerhalb von maximal 10 Sekunden reagieren. Die Bitraten oder Videoprofile der beteiligten Streams werden in Abhängigkeit ihrer Priorität angepasst. Um das Modul optimal konfigurieren zu können, sind allerdings Messungen zur Bestimmung der verfügbaren Bandbreite notwendig. Diese Bandbreite sollte nach Möglichkeit konstant bleiben. Ein Einsatz der Streamkontrolle über größere Netze (WAN, WWW) ist aufgrund der zum Teil stark schwankenden Bandbreite nicht sinnvoll. Allerdings ist die Unterteilung eines Überwachungsnetzes in beliebig viele Subnetze möglich. Diese können einzeln konfiguriert und gesteuert werden.
Die automatische Bitratensteuerung bei Frameverlusten beschränkt sich auf eine Absenkung. Das Kontrollmodul kann keine Nichtauslastung des Netzes erkennen und die Bitraten erhöhen. Dies muss der Benutzer durchführen.
Die zweite Aufgabe des Kontrollmodules ist die Überprüfung der Gesamtbitrate und Einhaltung von definierten Grenzwerten bei manueller Steuerung der Streams durch den Anwender. Auch hier zeigte sich, dass das Modul sicher funktioniert. Je nach Art der manuellen Steuerung wurden die Bitraten der beteiligten Streams gemäß ihrer Priorität angepasst. Bei Unterschreitung der Grenzwerte ist auch eine automatische Erhöhung bis maximal zur Profilbitrate möglich.
Ein Einsatz des MultiVideoClients auf Ethernet oder Fast Ethernet ist weniger problematisch, da es hier kaum zu Paketverlusten kommt. Der für das WLAN eingeführte Zustand B im Grenzbereich der Netto-Bandbreite ist somit nicht vorhanden. Trotzdem ist auch hier eine Begrenzung und Steuerung der Gesamtbitrate sinnvoll, wenn das Netz noch von anderen Teilnehmern benutzt wird.

6 Zusammenfassung

6.1 Inhalt

In der vorliegenden Diplomarbeit wurde eine Anwendung zum parallelen Empfang mehrerer Videoströme vorgestellt. Die Videos werden von einer Serveranwendung mit integriertem echtzeitfähigen MPEG-4 Encoder erzeugt und unter Verwendung der Standardprotokolle RTSP und RTP über ein IP-Netz übertragen. Das Server-Client-System ist für die Verwendung auf lokalen Netzen optimiert. Dazu wurde es zunächst auf Fast-Ethernet und weiterhin auf 802.11b Wireless LAN ausführlich getestet. Die speziellen Tests auf einem kabellosen Versuchsnetz mit vier Servern und einem Client dienten der Untersuchung des Verhaltens von WLAN während der Übertragung mehrerer Echtzeitvideoströme mit hohen Bitraten. Dabei wurde festgestellt, dass ein Empfang mit nur wenigen Frameverlusten bis nahe der Kapazitätsgrenze des Netzes möglich ist. Dafür müssen aber verschiedene Bedingungen erfüllt sein. Der Signal-Rausch-Abstand des Funknetzes sollte 25-30 dB nicht unterschreiten. Bei niedrigerem SNR reichen unter Umständen schon geringe Störungen aus, um die Übertragung nachhaltig zu behindern. Des Weiteren muss das Übertragungssystem Paketverluste kompensieren können, da diese in einem Funknetzwerk auch schon bei niedrigen Bitraten auftreten und zum Teil starke Bildartefakte verursachen. Zu diesem Zweck wurde in den Server ein Zwischenpuffer für bereits gesendete Pakete integriert. Stellt der Client anhand der Sequenznummern der RTP-Pakete einen Verlust fest, so kann er die fehlenden Pakete vom Server neu anfordern. Die Anzahl an Wiederholungen und die Wartezeiten kann der Benutzer festlegen. Er muss dabei berücksichtigen, dass die Dauer für eine Wiederholung mit der Netzbelastung ansteigt. Mit Hilfe des implementierten Systemes zur Paketwiederholung wurden für das Test-WLAN hohe Grenzwerte für die Gesamtbitrate erreicht. Bei 4 beteiligten Servern sind es etwa 5600 kbits/s, bei 3 und 2 Servern steigt dieser Wert auf ca. 6300 kbits/s. Beide Werte liegen aber noch deutlich unter der theoretischen Bandbreite für 802.11b WLAN von 11 Mbits/s, da außer den Nutzdaten noch Bandbreite für Headerinformationen, Paketbestätigungen auf MAC-Ebene, wiederholte Pakete usw. benötigt wird. Bei Überschreitung der Bitratengrenze kommt es schon bei einem oder mehreren Servern zu Frameverlusten durch Überlaufen des Sendepuffers. Zur Minderung der störenden Auswirkungen auf die empfängerseitige Anzeige erfolgt die Löschung von Frames beim Server selektiv. Die wichtigen I-Frames werden z.B. bevorzugt behandelt.

Um die Überschreitung des maximalen Datendurchsatzes eines Netzes zu verhindern, wurde ein Modul mit verschiedenen Steuerungsmechanismen implementiert. Dieses Modul regelt bei drohender Überlast, z.B. durch Zuschalten eines neuen Streams, die Bitraten der Streams so herunter, dass der Grenzwert nicht überschritten wird. Die notwendigen Daten für die Steuerung werden über eine Konfigurationsdatei geladen. Darin sind z.B. Grenzwerte, die Zuordnung der Server zu Subnetzen oder Serverprioritäten enthalten. Letztere legen fest, wie weit die Bitraten eines Streams relativ zum Ausgangswert gesenkt werden darf.

Das Kontrollmodul ist weiterhin für die Überwachung der Frameverluste zuständig. Dazu übermitteln die einzelnen Videoclients in regelmäßigen Abständen statistische Daten, wie gemessene Bitrate und Frameverlustraten an das Modul. Überschreiten die Verluste einen vorgegebenen Grenzwert, so

reagiert das Modul mit Senkung der Bitraten und Anpassung der Grenzwerte. In einigen Simulationen und Versuchen mit Fast Ethernet und WLAN wurde diese Steuerung erfolgreich getestet.

Der “MultiVideoClient“ liegt als Softwaremodul in Form einer dynamischen Bibliothek vor. Zur Benutzung des Modules müssen Softwareentwickler eine geeignete Anwendung implementieren, welche über die Interfacefunktionen des MVC auf dessen Features zugreifen kann. Die Anwendung sollte auch ein Fenster bereitstellen, welches Nachrichten vom Client empfangen und bearbeiten kann. In dieses Fenster können bei Bedarf auch die Videos gerendert werden. Neben einer einfachen Testanwendung wurde die menü- und dialogbasierte Anwendung “TraViS“ implementiert, welche dem Benutzer einfachen Zugriff auf das MVC-Modul bietet.
Der MVC wurde für 32-Bit-Windows-Systeme konzipiert und unter Win2K und WinXP getestet. Sein Multithreaddesign erlaubt Performance-Steigerungen auf Mehrprozessorsystemen oder dem neuen P4 mit Hyperthreading. Verschiedene Algorithmen im integrierten Decoder und Renderer benutzen zur Steigerung der Geschwindigkeit die Befehlssätze MMX und SSE2 moderner Prozessoren. So konnten im Labor und auf mehreren öffentlichen Präsentationen 4 Streams mit einer Auflösung von 640x480 und einer Gesamtbildrate von 60 fps auf einem Dual-Prozessor-System empfangen werden.

6.2 Anwendung und Ausblick

Das System, bestehend aus Server- und Clientanwendung, wurde für den speziellen Fall der Videoüberwachung entwickelt. Da es auf konventioneller PC-Technik läuft, kann damit ein kostengünstiges Überwachungssystem aus Standardkomponenten aufgebaut werden. Auf Serverseite genügt aufgrund des schnellen Encoders ein einfaches PC-System mit ca. 1 GHz Prozessortakt und Netz-Anschluß zur Codierung und Versendung der Videostreams. In Fällen, wo nur wenig Platz zur Verfügung steht, kann ein sogenannter Barebone-PC eingesetzt werden, ein vollständig ausgerüsteter Computer im speziellen Gehäuseformat von nur 30x20x20 cm. Der Empfang mehrerer großformatiger Streams ist ebenfalls mit einem Standard-PC möglich. Das Bildmaterial kann dabei für eine spätere Auswertung als unkomprimiertes Einzelbild oder MPEG4-Stream aufgenommen werden.
Da die empfangenen Bilder auch im Format YC_bC_r 4:2:0 abgerufen werden können, ist weiterhin eine automatische Überwachung mit Hilfe von Algorithmen zur Bewegungsdetektion, Objekt- oder auch Gesichtserkennung denkbar.
Der Einsatz des Systemes auf Wireless LAN ermöglicht einen kostengünstigen Aufbau eines Überwachungsnetzes. Allerdings ist dies zur Zeit noch nicht anzuraten, da die aktuelle standardisierte WEP-Verschlüsselung nicht sicher ist. Zudem sind die möglichen Störeinflüsse auf das Funknetz im Vorfeld genau zu klären. Hohe Dämpfung, Mehrwegeempfang oder andere Funknetze senken die zur Verfügung stehende Bandbreite. Die Weiterentwicklung von wireless LAN zum 802.11g Standard mit 54 Mbits/s Bandbreite verspricht aber neue Einsatzmöglichkeiten mit mehr Videoströmen bzw. höheren Bitraten.
Zur zusätzlichen Sicherheit wäre die Implementierung von Verschlüsselungsalgorithmen auf Anwendungsebene sinnvoll. Hierfür stehen eine Reihe von sicheren Verfahren zur Verfügung (RS4, PGP etc.).

Verzeichnisse

Literaturverzeichnis

[1] Jerry D. Gibson et al, "The Communication Handbook – Second Edition", CRCPress, 2002

[2] Jürgen Goebel, "Kommunikationstechnik", Hüthig Verlag Heidelberg, 1999

[3] M. Handley, V. Jacobson, “SDP : Session Description Protocol“, RFC 2327, April 1998

[4] ISO/IEC JTC1/SC29/WG11, “Overview of the MPEG4-Standard“, 07/98

[5] ISO/IEC JTC1/SC29/WG11, "Information Technology- Generic Coding of Audio-Visual Objects - Part 2 : Visual", Document no. N2502, Atlantic City, October 1998

[6] A. Köpsel, J.P.Ebert, A. Wolisz, “A Performance Comparison of Point and Distributed Coordination Function of an IEEE 802.11 WLAN in the Presence of Real-Time Requirements“, Berlin 2000

[7] Peter Noll, “Nachrichtenübertragung II”, Berlin, 1999

[8] Jon Postel, “Internet Protocol - DARPA Internet Program Protocol Specification”, RFC791, September 1981

[9] Jon Postel, “Transmission Control Protocol - DARPA Internet Program Protocol Specification”, RFC793, September 1981

[10] Jon Postel, "User Datagram Protocol", RFC 768, August 1980

[11] B. Rathke, M. Schläger, A. Wolisz, “Systematic Measurement of TCP-Performance over Wireless LANs”, Berlin, 1998

[12] Asunción Santamaría, Francisco J. López-Hernández, "Wireless LAN - Standards and Applications", Artech House, 2001

[13] H. Schulzrinne, “Real Time Streaming Protocol (RTSP)“, RFC2326, April 1998

[14] H. Schulzrinne, “RTP : A Transport Protocol for Real-Time Applications“, RFC1889, January 1996

[15] Benno Stabernack, Christian Stoffers, "Documentation MPEG-4 Video Encoder”, March 2001

[16] William Stallings, ”Data and Computer Communications“, 6th Edition, Prentice Hall, 2000

[17] Viktor Toth, Dirk Louis, „Visual C++ 6 Kompendium”, Markt & Technik, 1999

[18] Patrick Voigt, “Entwicklung eines steuerbaren Videoservers zur Übertragung von MPEG-4 Echtzeitvideoströmen”, Studienarbeit, Berlin, September 2002

[19] J. Weinmiller, M. Schläger, A. Festag, A. Wolisz, "Performance Study of Access Control in Wireless LANs - IEEE 802.11 DFWMAC and ETSI RES 10 HIPERLAN", Berlin, 1997

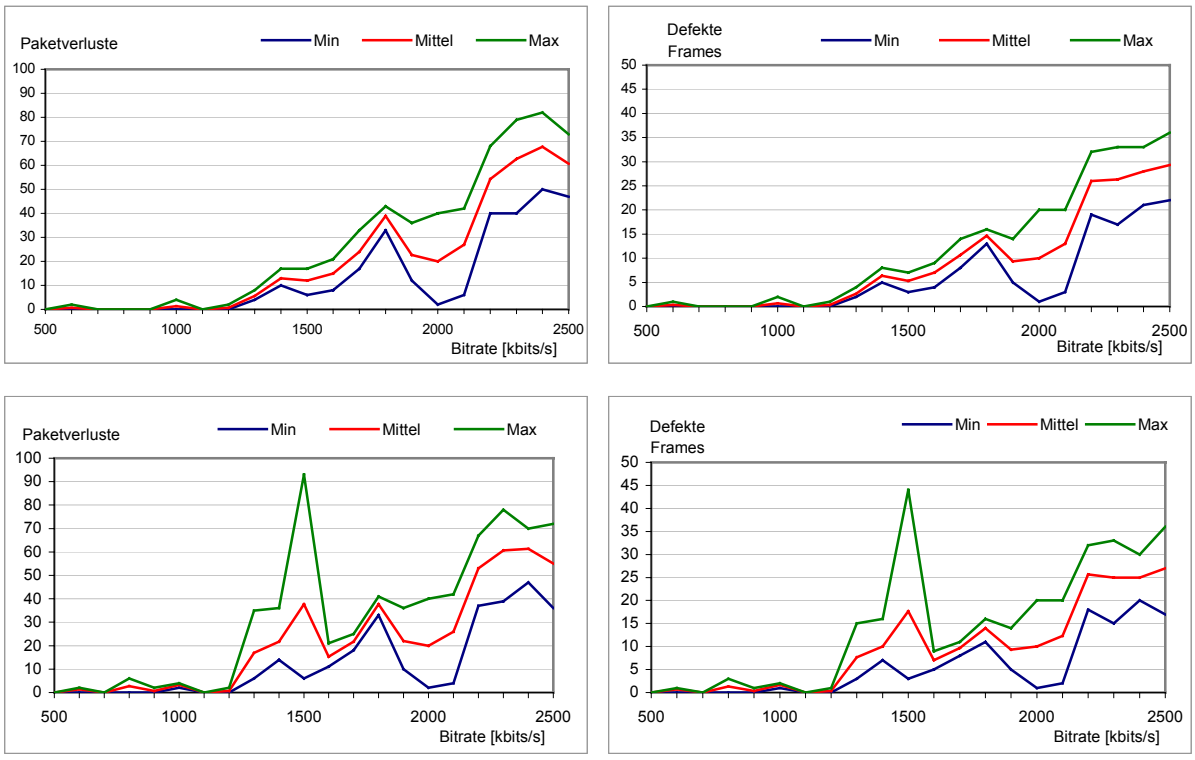
Abbildungsverzeichnis

Tabellenverzeichnis

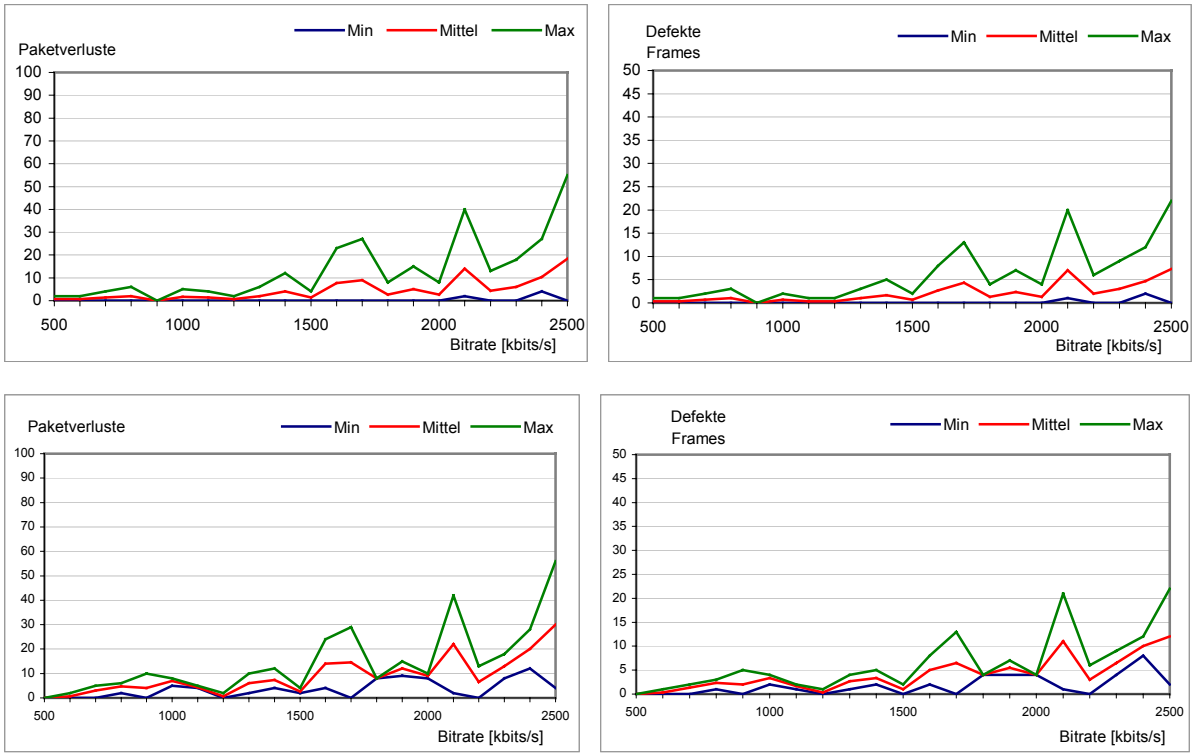
Anhang

Anhang A – Meßergebnisse der Tests auf WLAN (/Fast Ethernet)

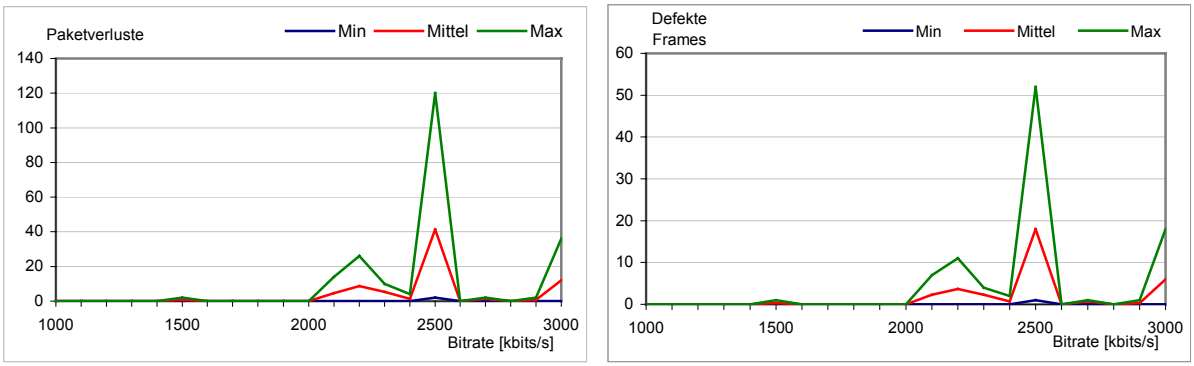
A.1) 2 Server, ohne Paketwiederholung, SNR > 40dB (sehr gut)



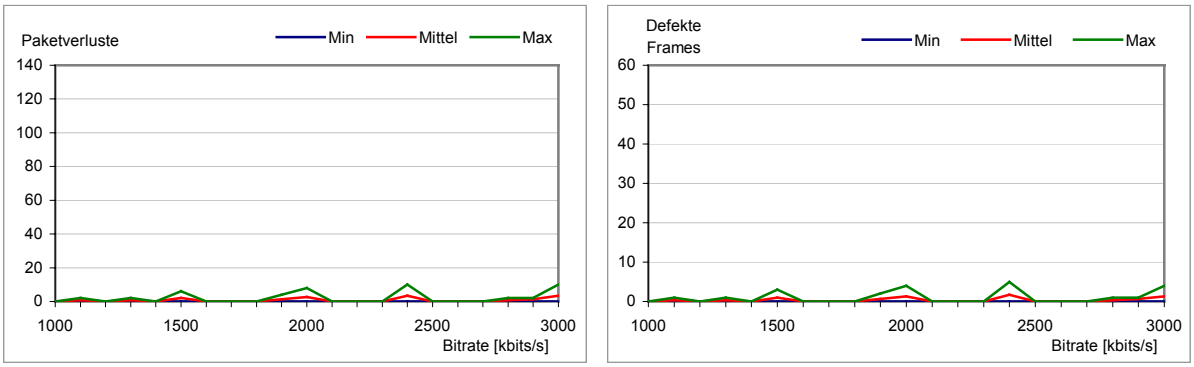
Server 1 (oben) und Server 2 (unten)
Abb. A.1 : Paket-/Frameverluste bei 2 Servern (alte Treiber)



Server 1 (oben) und Server 2 (unten)
Abb. A.2 : Paket-/Frameverluste bei 2 Servern mit Startverzögerung
(250 ms für Server #2)



Server 1



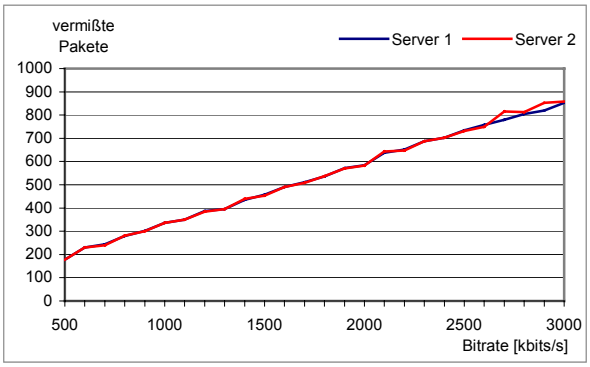
Server 2

Abb. A.3 : Paket-/Frameverluste bei zwei Servern (neueste Treiber)

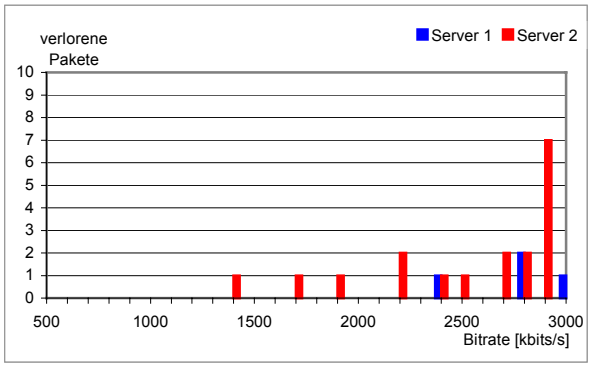
A.2) 2 Server mit aktivierter Paketwiederholung

a.) Bei einer Meßreihe mit zwei Servern wurden keine Paketverluste beobachtet.
b.) Nachfolgende Werte wurden für aktivierte Paket-Verlust-Simulation gemessen

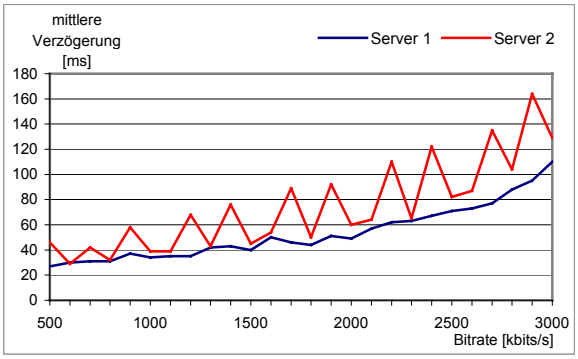
P_{Lost} = 0.01 und $P_{Lost|Lost}$ = 0.01 Timeouts : T_{RT} = 250 ms und T_{Total} = 500 ms



a) vermißte Pakete nach 1. Übertragung



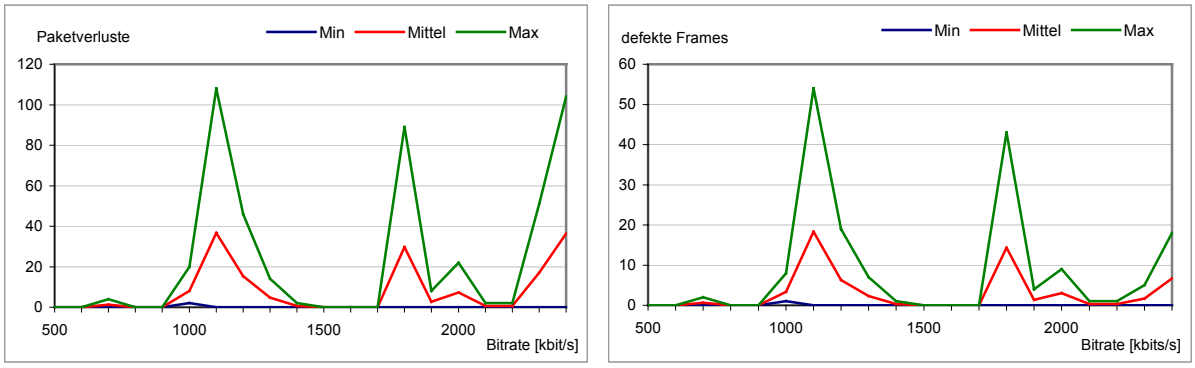
b) Verlorene Pakete nach wiederholter Übertragung



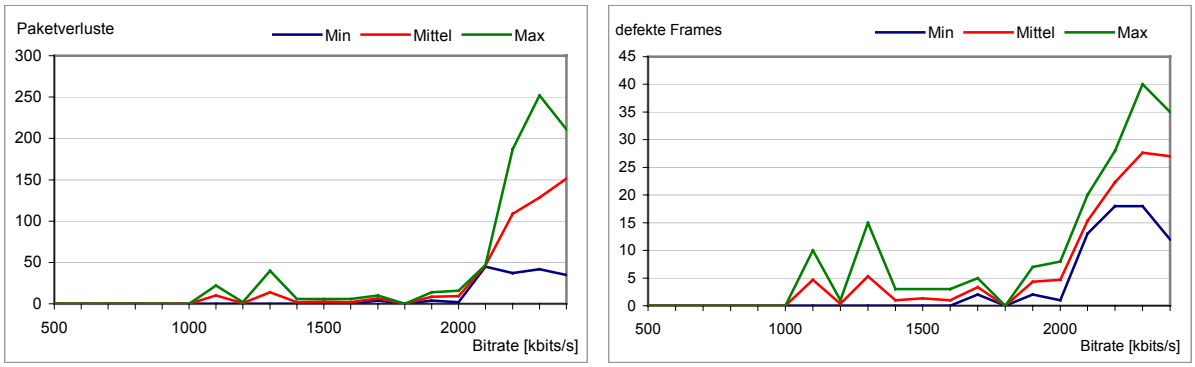
c) mittlere Wartezeit bei wiederholter Übertragung

Abb. A.4 : Paketverlustsimulation mit zwei Servern auf WLAN

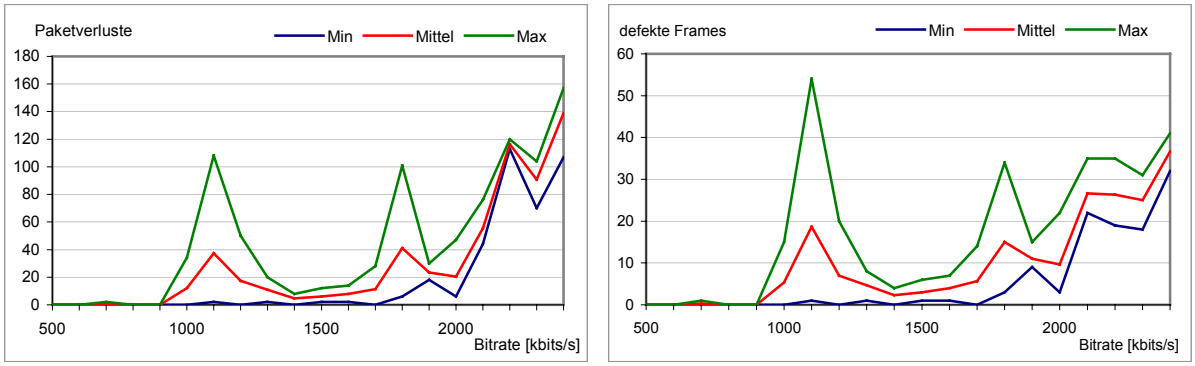
A.3) Drei Server ohne aktivierte Paketwiederholung, SNR > 40 dB (“excellent“)



Server 1



Server 2



Server 3

Abb. A.5 : Paketverluste/Frameverluste bei 3 Servern (ohne Paketwiederholung)

A.4) 3 Server mit aktivierter Paketwiederholung

Timeouts : T_{RT} = 250ms und T_{Total} = 500ms -> max. 2 Wiederholungen

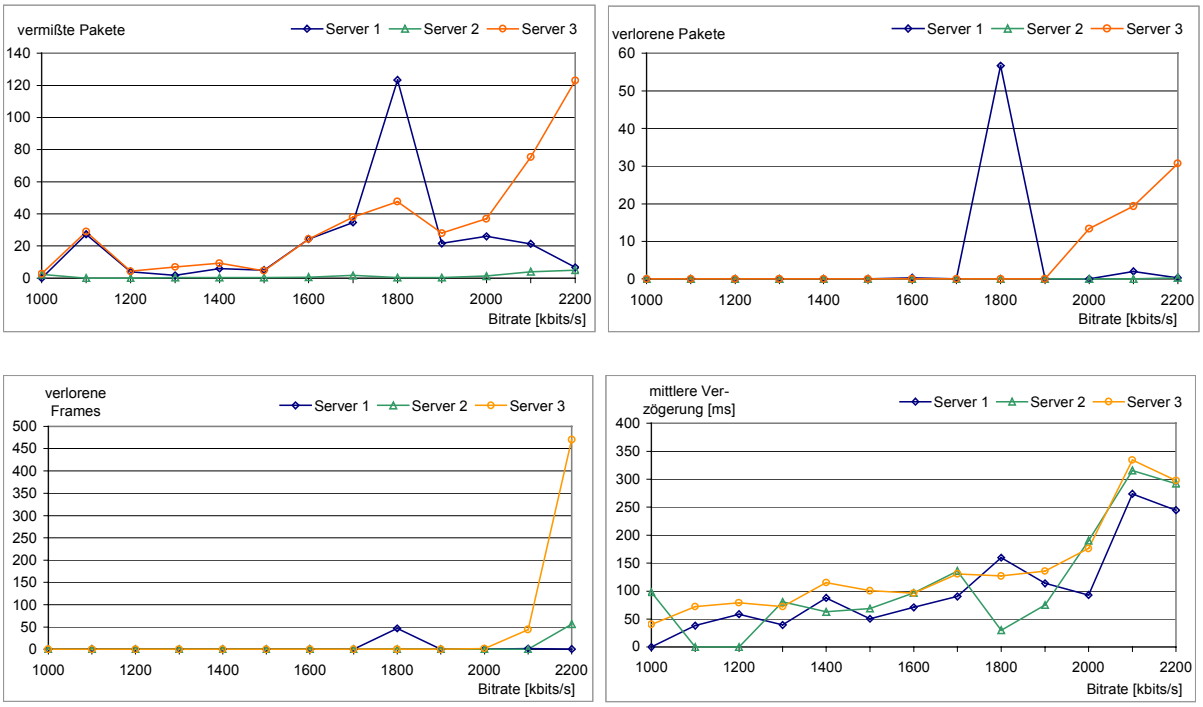


Abb. A.6 : Paketverluste/Frameverluste bei 3 Servern (mit Paketwiederholung)

A.5) 3 Server mit aktivierter Paketwiederholung

Server #1 und #2 fest auf 2500 kbits/s
Server #3 variabel von 500 – 2000 kbits/s
gemessene maximale Gesamtbitrate : ca. 6300 kbits/s

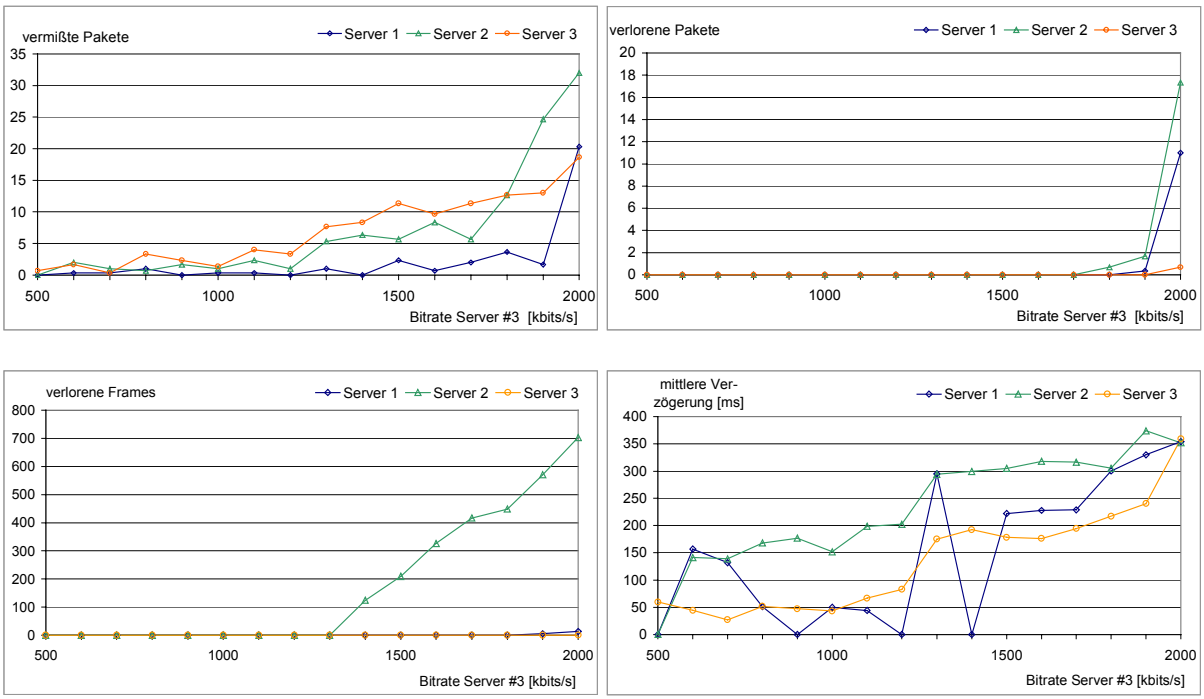


Abb. A.7 : Paketverluste/Frameverluste bei 3 Servern (2 Server fest)

A.6) 4 Server mit aktivierter Paketwiederholung

Timeouts : T_{RT} = 250ms und T_{Total} = 500ms -> max. 2 Wiederholungen

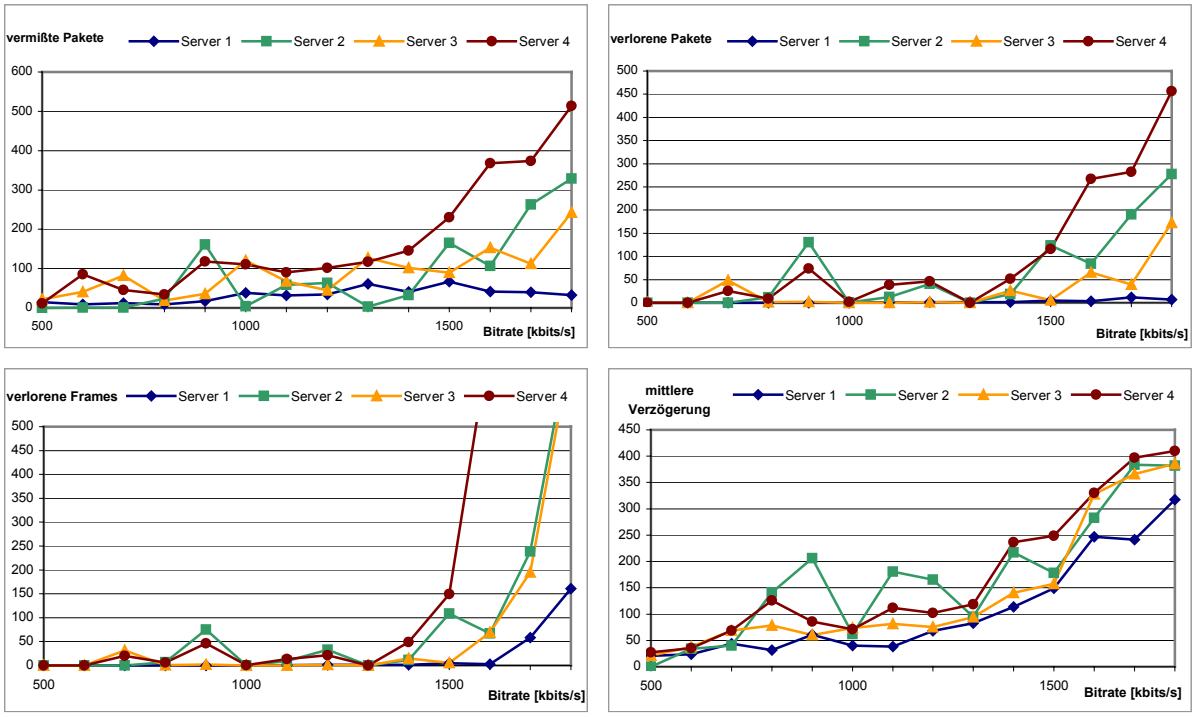


Abb. A.8 : Paketverluste/Frameverluste im Vergleich bei 4 Servern (Mittelwerte)

Server #1 Server #2

Server #3 Server #4

Abb. A.9 : Einzelwerte der Paketverluste bei 4 Servern

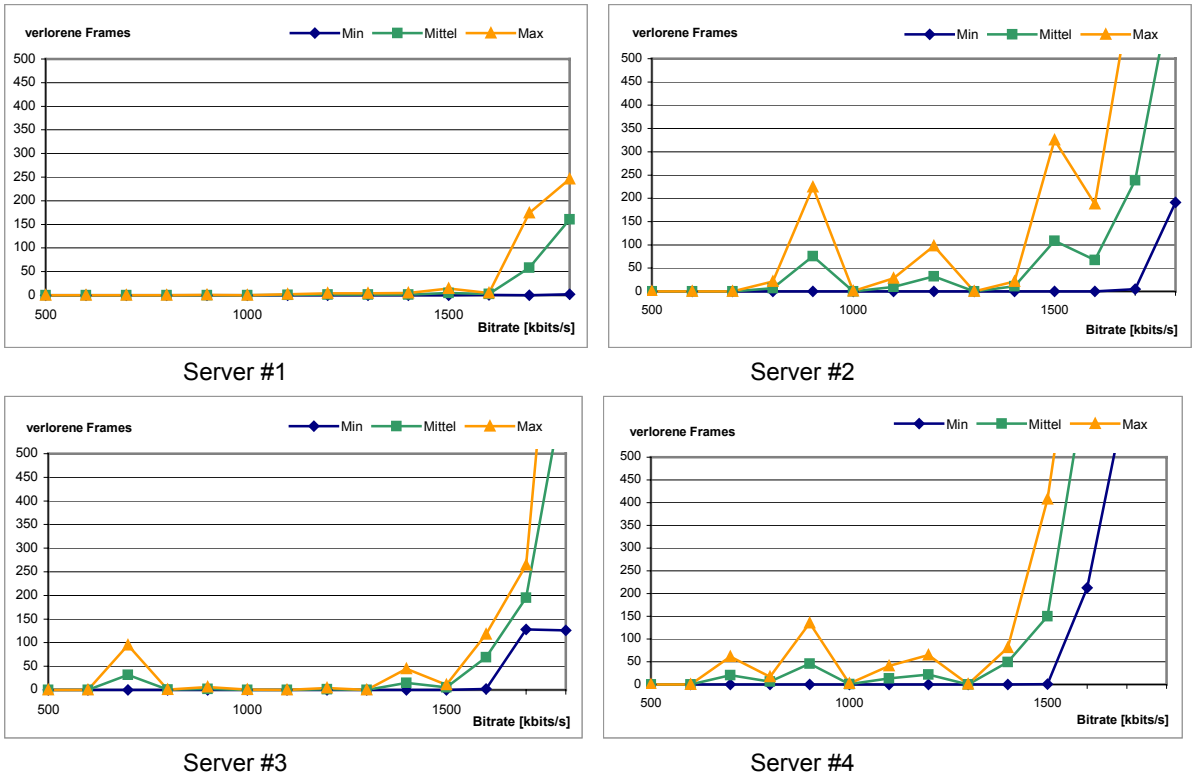


Abb. A.10 : Einzelwerte der Frameverluste bei 4 Servern

A.7) Kurzzeitbitratenmessung des RTP-Clients

A.7.1) Fast Ethernet, 2 Server, Refresh Period: t_1 = 10 sec / t_2 = 20 sec

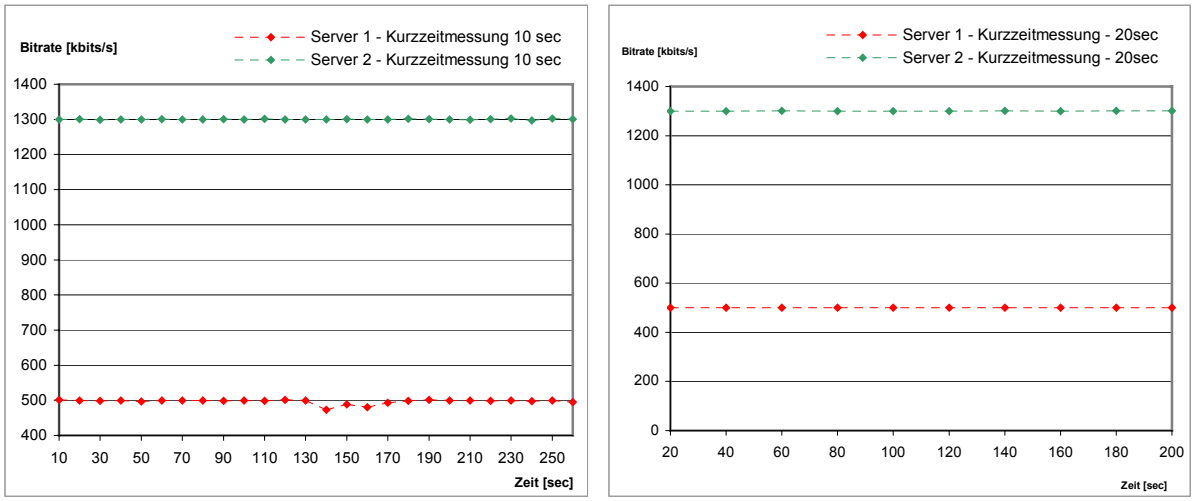


Abb. A.11 : Kurzzeitbitratenmessung bei 2 Servern (Fast Ethernet, $T_{refresh}$ =10sec/20sec)

A.7.2) WLAN, 3 Server a 2200 kbits/s, Refresh Rate 10/20ms

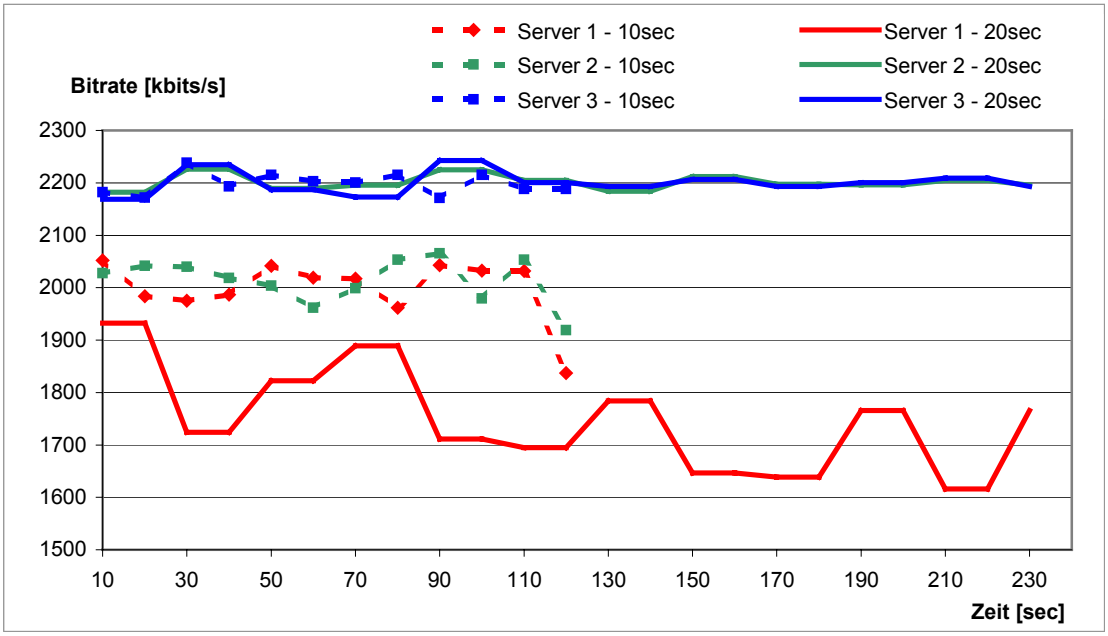


Abb. A.12 : Kurzzeitbitratenmessung bei 3 Servern auf WLAN ($T_{refresh}$ =10sec/20sec)

Mittelwerte der gemessenen Bitraten [kbits/s]:

	10 sec				**20 sec**			
	Server 1	Server 2	Server 3	Gesamt	Server 1	Server 2	Server 3	Gesamt
Mittelwerte	1999	2014	2198	6210	1748	2201	2201	6150
Std.-Abw.	56,33	41,44	19,15		93,94	13,64	20,64	

Tab. A.1 : Mittelwerte und Standardabweichungen bei Kurzzeitbitratenmessung

A.8) Fehlersimulation zum Test des Kontrollmodules

2 Server auf Ethernet
Die Änderungen sind jeweils hervorgehoben

Zustand C
Prioritäten Server 1 1
Server 2 1

Zeit	Aktuelle Bitraten			gemessene Bitraten			Profiles		Maximale Bitraten		
	Server 1	Server 2	Gesamt	Server 1	Server 2	Gesamt	Server 1	Server 2	1 Server	2 Server	3 Server
Start	1300	1300	2600	0	0	0	1	1	3000	2600	2200
12 sec	1000	1000	2000	1150	1150	2300	**3**	**3**	2388	**2069**	1751

Zustand C
Prioritäten Server 1 1
Server 2 2

Zeit	Aktuelle Bitraten			gemessene Bitraten			Profiles		Maximale Bitraten		
	Server 1	Server 2	Gesamt	Server 1	Server 2	Gesamt	Server 1	Server 2	1 Server	2 Server	3 Server
Start	1300	1300	2600	0	0	0	1	1	3000	2600	2200
16 sec	**1042**	**1171**	2213	1230	1230	2460	1	1	2554	**2213**	1873

Zustand C
Prioritäten Server 1 1
Server 2 2

Zeit	Aktuelle Bitraten			gemessene Bitraten			Profiles		Maximale Bitraten		
	Server 1	Server 2	Gesamt	Server 1	Server 2	Gesamt	Server 1	Server 2	1 Server	2 Server	3 Server
Start	1300	1300	2600	0	0	0	1	1	3000	2600	2200
14 sec	1000	**1170**	2170	1206	1206	2412	**3**	1	2504	**2170**	1836

Zustand B
Prioritäten Server 1 1
Server 2 2

Zeit	Aktuelle Bitraten			gemessene Bitraten			Profiles		Maximale Bitraten		
	Server 1	Server 2	Gesamt	Server 1	Server 2	Gesamt	Server 1	Server 2	1 Server	2 Server	3 Server
Start	1300	1300	2600	0	0	0	1	1	3000	2600	2200
11 sec	**1126**	**1213**	2339	1100	1100	2200	1	1	2699	**2339**	1979

Zustand B
Prioritäten Server 1 2
Server 2 2

Zeit	Aktuelle Bitraten			gemessene Bitraten			Profiles		Maximale Bitraten		
	Server 1	Server 2	Gesamt	Server 1	Server 2	Gesamt	Server 1	Server 2	1 Server	2 Server	3 Server
Start	1300	1300	2600	0	0	0	1	1	3000	2600	2200
14 sec	1000	1300	2300	1100	1100	2200	**3**	1	2699	**2339**	1979

Tab. A.2 : Reaktionssimulation des Streamkontrollmodules

A.9) Live-Tests auf WLAN mit 3 und 4 Servern

A.9.1) 3 Server mit überhöhter Bitrate (2200 kbits/s), Paketwiederholungen 3, RTT = 250ms, Refresh Rate 10s

Zeit	Bitraten [kbits/s]				gemessene Bitraten				Frameverluste Server		
sec	1	2	3	Gesamt	1	2	3	Gesamt	1	2	3
1	2200	2200	2200	6600	0	0	0	0	0	0	0
14	2200	2200	2200	6600	2076	1994	2151	6221	**7**	**12**	0
17	1866	1866	1866	**5598**	2076	1994	2151	6221	7	12	0
24	1866	1866	1866	5598	1958	1958	2016	5932	1	5	0
34	1866	1866	1866	5598	1865	1866	1864	5595	0	0	0

Tab. A.3 : Änderung der Bitraten bei Serververlusten

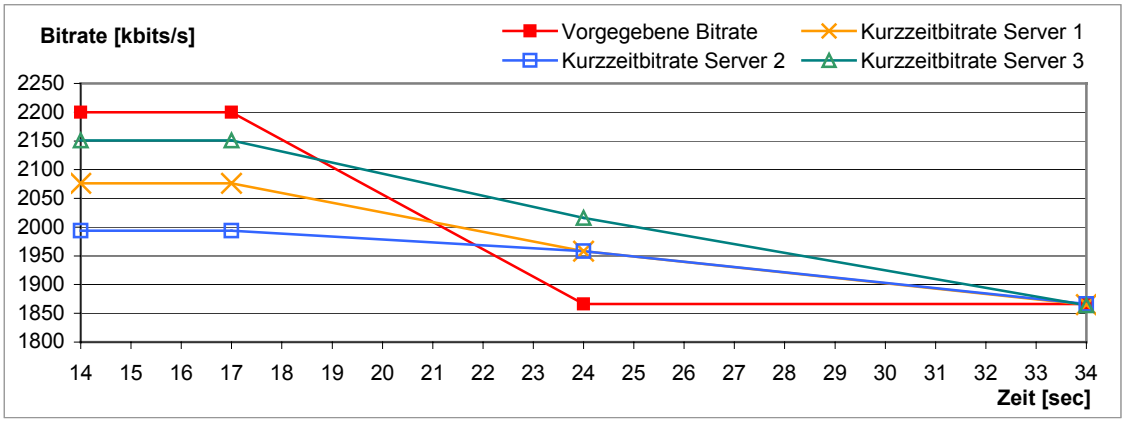


Abb. A.13 : Bitratenverlauf bei Netzüberlastung

Zeit	Bitraten [kbits/s]				gemessene Bitraten				Frameverluste Server		
sec	1	2	3	Gesamt	1	2	3	Gesamt	1	2	3
1	2200	2200	2200	6600	0	0	0	0	0	0	0
17	2200	2200	2200	6600	2053	1917	2207	6177	**11**	**18**	**2**
20	1852	1852	1852	**5556**	2053	1917	2207	6177	11	18	2
27	1852	1852	1852	5556	1916	1935	1898	5749	3	4	6
37	1852	1852	1852	5556	1851	1839	1778	5468	0	0	**5**
39	1640	1640	1640	**4920**	1851	1839	1778	5468	0	0	5
47	1640	1640	1640	4920	1663	1681	1663	5007	0	0	1
57	1640	1640	1640	4920	1640	1641	1642	4923	0	0	1
67	1640	1640	1640	4920	1632	1641	1651	4924	0	0	0
77	1640	1640	1640	4920	1640	1640	1641	4921	0	0	0

Tab. A.4 : Zweifache Änderung der Bitraten bei Serververlusten

A.9.2) 4 Server mit verschiedenen Bitraten, Paketwiederholungen 3, RTT = 250ms, Refresh Time 10s

Zeit	eingestellte Bitraten [kbits/s]				gemessene Bitraten				Netzverluste
sec	1	2	3	4	1	2	3	4	Server 4
3	1300	1300	1300	1300	0	0	0	0	0
21	1300	1300	1300	1300	1300	1293	1304	1301	0
31	1300	1300	1300	1300	1300	1300	1300	1300	0
41	1300	1300	1300	1300	1300	1300	1302	1300	0
51	1300	1300	1300	1300	1300	1301	1294	1300	0
61	1300	1300	1300	1300	1302	1299	1301	1300	0
71	1300	1300	1300	1300	1300	1302	1300	1246	0
81	1300	1300	1300	1300	1300	1302	1303	1246	**10**
84	**1214**	**1214**	**1214**	**1214**	1300	1302	1303	1309	0
91	1214	1214	1214	1214	1300	1227	1246	1309	0
101	1214	1214	1214	1214	1215	1215	1220	1213	0
111	1214	1214	1214	1214	1214	1220	1211	1202	0
121	1214	1214	1214	1214	1215	1210	1212	1214	0

Tab. A.5 : Änderung der Bitraten bei Netzverlusten (4x1300 kbits/s)

Zeit	eingestellte Bitraten [kbits/s]				gemessene Bitraten				Netzverluste
sec	1	2	3	4	1	2	3	4	Server 4
3	1500	1500	1500	1500	0	0	0	0	0
14	1500	1500	1500	1500	1493	1499	1496	1493	0
174	1500	1500	1500	1500	1501	1501	1498	1504	0
184	1500	1500	1500	1500	1502	1492	1493	1440	**9**
187	**1348**	**1348**	**1348**	**1348**	1502	1492	1493	1440	9
194	1348	1348	1348	1348	1381	1396	1402	1448	0
204	1348	1348	1348	1348	1349	1348	1348	1350	0
224	1348	1348	1348	1348	1348	1349	1355	1341	0
244	1348	1348	1348	1348	1341	1348	1342	1358	0
264	1348	1348	1348	1348	1350	1349	1343	1359	0
284	1348	1348	1348	1348	1341	1348	1341	1358	0
304	1348	1348	1348	1348	1348	1349	1356	1349	0

Tab. A.6 : Änderung der Bitraten bei Netzverlusten (4x1500 kbits/s)

Zeit	eingestellte Bitraten				gemessene Bitraten				Netzverluste		Serververluste
sec	1	2	3	4	1	2	3	4	3	4	2
3	1700	1700	1700	1700	0	0	0	0	0	0	0
21	1700	1700	1700	1700	1562	1737	1725	762	2	**41**	**3**
24	**1348**	**1348**	**1348**	**1348**	1562	1737	1725	762	2	41	3
31	1348	1348	1348	1348	1378	1411	1418	1334	0	5	0
41	1348	1348	1348	1348	1346	1349	1361	1349	0	0	0
51	1348	1348	1348	1348	1351	1349	1349	1349	0	0	0
61	1348	1348	1348	1348	1348	1348	1351	1350	0	0	0
71	1348	1348	1348	1348	1351	1349	1348	1350	0	0	0
81	1348	1348	1348	1348	1347	1350	1351	1232	0	**6**	0
84	**1214**	**1214**	**1214**	**1214**	1347	1350	1351	1232	0	6	0
91	1214	1214	1214	1214	1228	1240	1266	1271	0	0	0
101	1214	1214	1214	1214	1215	1215	1213	1215	0	0	0
121	1214	1214	1214	1214	1216	1215	1222	1208	0	0	0
141	1214	1214	1214	1214	1216	1222	1214	1223	0	0	0
161	1214	1214	1214	1214	1216	1214	1216	1215	0	0	0
181	1214	1214	1214	1214	1215	1214	1213	1223	0	0	0
201	1214	1214	1214	1214	1224	1215	1208	1209	0	0	0

Tab. A.7 : Zweifache Änderung der Bitraten bei Netz- und Serververlusten (4x1700 kbits/s)

A.9.3) 4 Server mit verschiedenen Bitraten, Paketwiederholungen 3, RTT = 250ms,
Refresh Time 20s
verschiedene Prioritäten

Prioritäten : niedrig (1)

Zeit	eingestellte Bitraten				gemessene Bitraten				Netzverluste	Serververluste		
sec	1	2	3	4	1	2	3	4	4	1	2	4
3	1700	1700	1700	1700	0	0	0	0	0	0	0	0
34	1700	1700	1700	1700	1610	1682	1701	1251	**8**	**13**	**5**	**111**
39	**1348**	**1348**	**1348**	**1348**	1610	1682	1701	1251	8	13	5	111
54	1348	1348	1348	1348	1403	1417	1430	1405	1	1	0	13
74	1348	1348	1348	1348	1348	1348	1346	1351	0	0	0	0
94	1348	1348	1348	1348	1348	1344	1349	1344	0	0	0	0
114	1348	1348	1348	1348	1347	1350	1351	1351	0	0	0	0
134	1348	1348	1348	1348	1348	1347	1350	1348	0	0	0	0
154	1348	1348	1348	1348	1348	1349	1347	1348	0	0	0	0
174	1348	1348	1348	1348	1347	1347	1350	1344	0	0	0	0

Tab. A.8 : Änderung der Bitraten bei Netz- und Serververlusten (4x1700 kbits/s)

Prioritäten : Server 1 -> niedrig (1)
Server 2 -> mittel (2)
Server 3 -> hoch (3)
Server 4 -> niedrig (1)

Zeit	eingestellte Bitraten					gemessene Bitraten					Netzverluste		Serververluste		
sec	1	2	3	4	ges.	1	2	3	4	ges.	1	4	1	2	4
3	1700	1700	1700	1700	6800	0	0	0	0	0	0	0	0	0	0
24	1700	1700	1700	1700	6800	1605	1634	1687	1314	6240	1	**3**	**9**	**4**	**63**
28	**1170**	**1359**	**1700**	**1170**	5399	1605	1634	1687	1314	6240	1	3	9	4	63
44	1170	1359	1700	1170	5399	1225	1436	1713	1262	5636	0	0	4	4	10
64	1170	1359	1700	1170	5399	1170	1360	1706	1174	5410	0	0	0	0	0
84	1170	1359	1700	1170	5399	1170	1355	1694	1171	5390	0	0	0	0	0
104	1170	1359	1700	1170	5399	1170	1361	1705	1170	5406	0	0	0	0	0
124	1170	1359	1700	1170	5399	1170	1359	1698	1167	5394	0	0	0	0	0

Tab. A.9 : Änderung der Bitraten bei verschiedenen Streamprioritäten

Anhang B – Least Square Framesynchronisation

Teststreams jeweils mit 2000 Frames : Bildabstände beim Server (links) und Client (rechts)

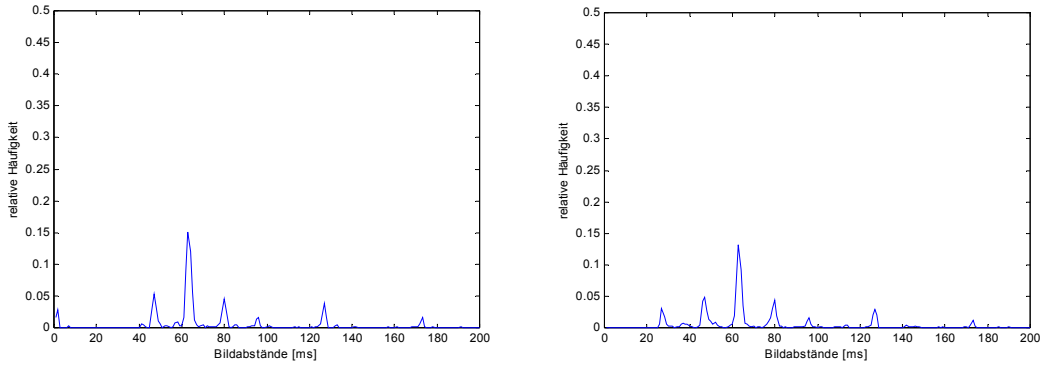


Abb. B.1 : 640x480x15Hz, Einzelstream, unsynchronisiert

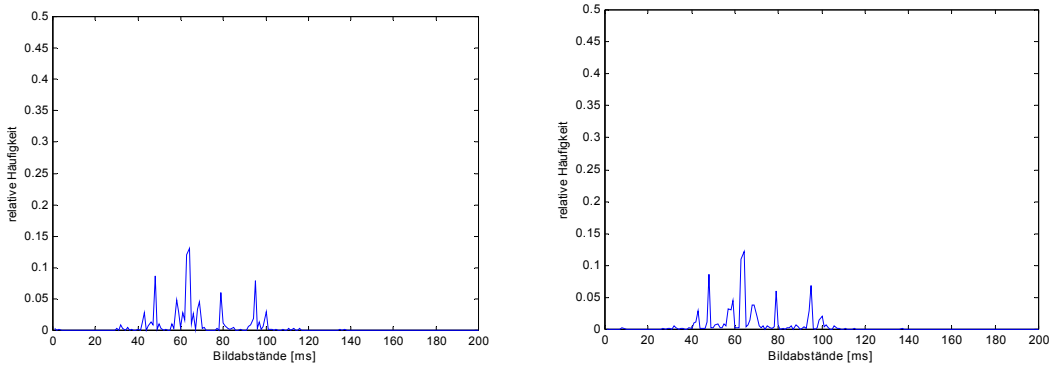
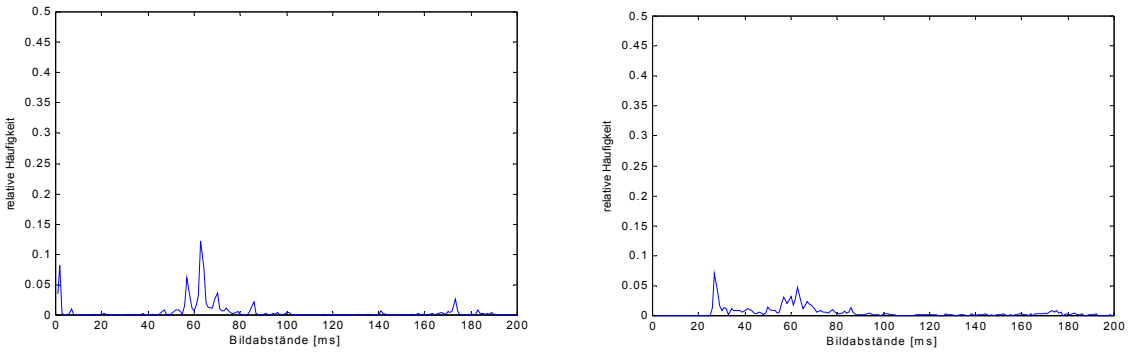
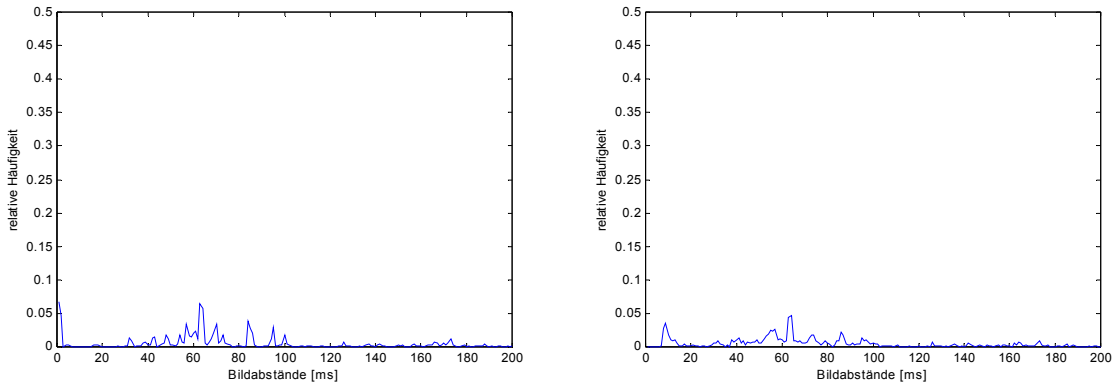


Abb. B.2 : 320x240x15Hz, Einzelstream, unsynchronisiert



640x480x15Hz



320x240x15Hz

Abb. B.3 : Doppelstream unsynchronisiert, 2000 Frames

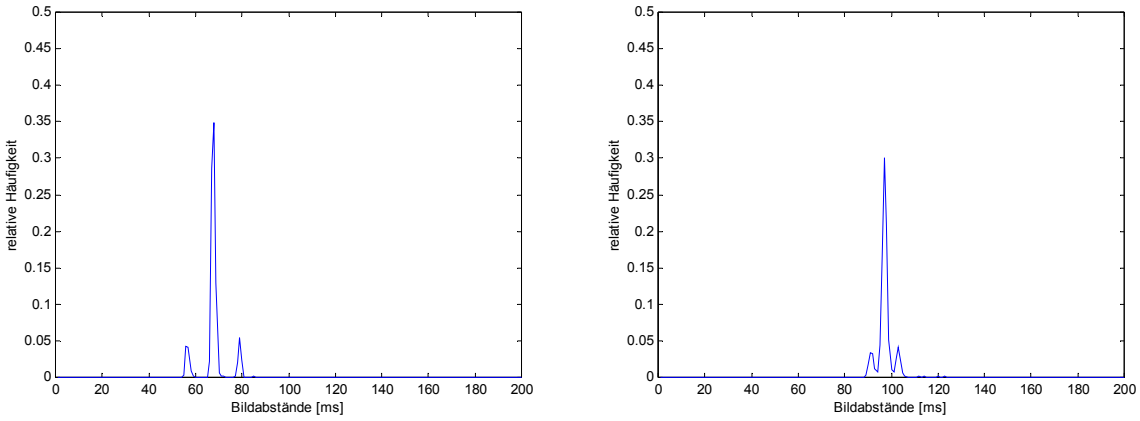


Abb. B.4 : Einzelstream, LS-synchronisiert ohne I-P-Frame-Sync
(a) 640x480x15 Hz : Mittelwert 66,84 ms , Standardabweichung 5,4 ms
(b) 640x480x10 Hz : Mittelwert : 96,3 ms, Standardabweichung 4,8 ms

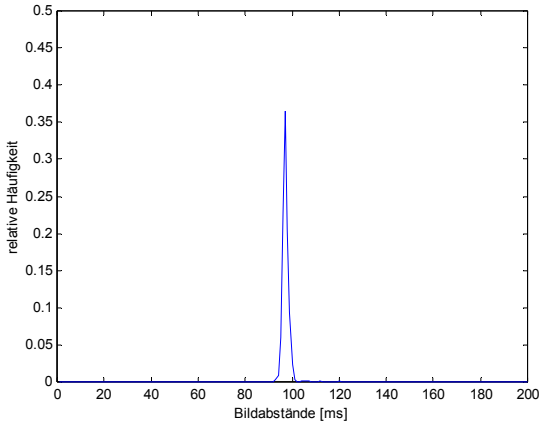


Abb. B.5 : Einzelstream, LS-synchronisiert mit I-P-Frame-Sync
640x480x 10 Hz : Mittelwert 96,15 ms/ Standardabweichung 2,84 ms/ mit IP-Frame-Sync

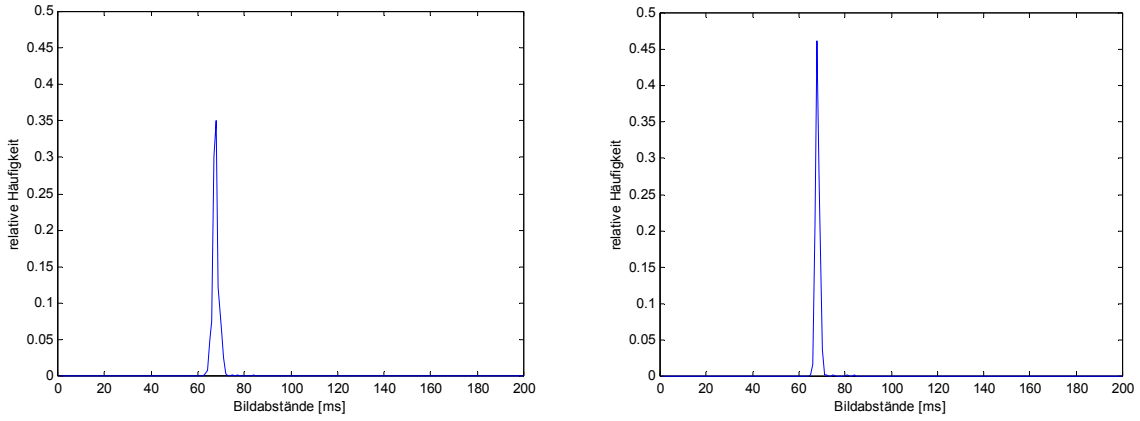


Abb. B.6 : Einzelstream, LS-synchronisiert
(a) 320x240x15 Hz : Mittelwert 66,75 ms / Standardabweichung 2,3 ms / ohne IP-Frame-Sync
(b) 320x240x15 Hz : Mittelwert : 67,15 ms / Standardabweichung 2,07 ms / mit IP-Frame-Sync

Anhang C – MultiVideoClient Interface

C.1) Einleitung

Der MultiVideoClient (MVC) ist ein als DLL implementiertes Softwaremodul zum Empfang mehrerer Videoströme, welche in Form von MPEG-4-Elementardatenströmen von einem Videoserver versendet werden. Diese Ströme werden mit Hilfe des Standardprotokolls RTP übertragen. Der MVC empfängt die RTP-Pakete, setzt sie zu sogenannten Access Units (VOPs) zusammen und dekodiert sie. Die dekodierten Bilder werden in einem Fenster angezeigt.
Entwickelt wurde der MVC für Win32-Betriebssysteme. Getestet wurde er auf Win2K und WinXP.
Nachfolgender Dokumentation liegt die Version 1.1, Stand 15.10.2003 zugrunde.

C.2) Interner Aufbau

Wie in Abb. C.1 zu sehen ist, gibt es ein zentrales Modul, den MVC-Manager. Dieser erstellt und steuert die einzelnen VideoClients (SingleVideoClient -> SVC). Die SVCs wiederum bestehen aus einem RTSP/RTP-Client, einem MPEG-4-Decoder und einem Renderer.

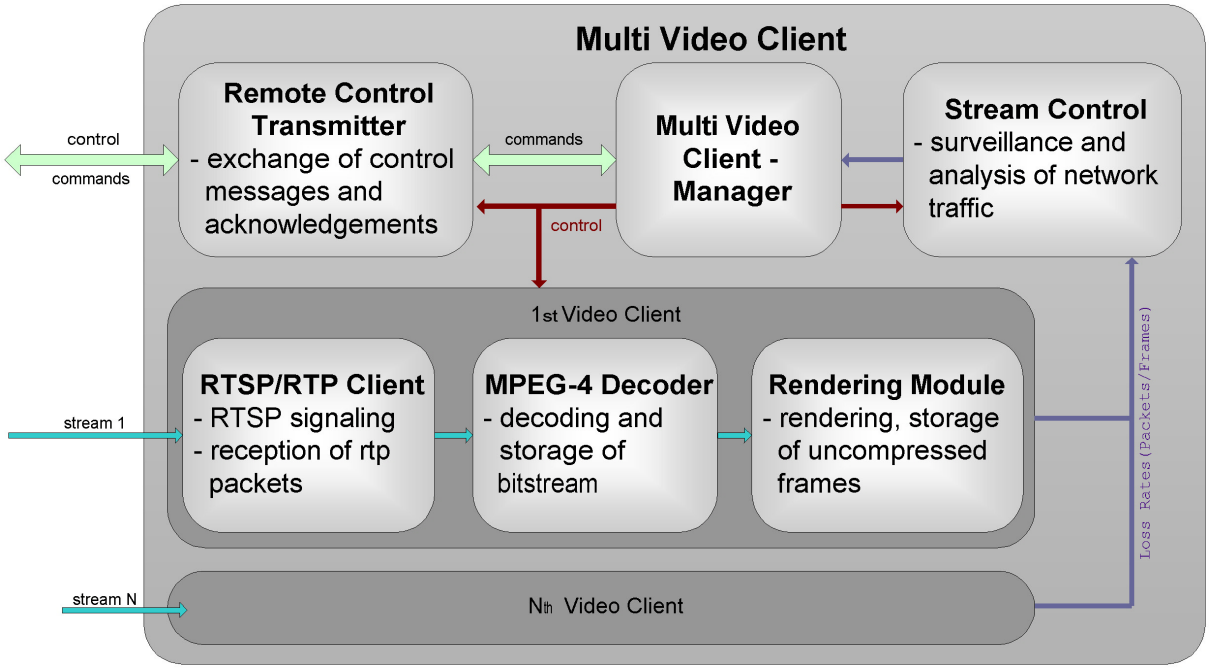


Abb. C.1 : Aufbau MultiVideoClient

Das Kommandomodul (Remote Control Transmitter) dient dem Absetzen von Befehlen an die Serveranwendung. Hiermit kann z.B. der Videostrom gesteuert oder Informationen abgefragt werden.
Wie schon erwähnt, wird der MVC als DLL geliefert. Der Benutzer muss also eine geeignete Hauptanwendung implementieren. Dies kann eine menügeführte oder eine einfache Win32-Konsolenanwendung sein.

Um die DLL benutzen zu können, muss die Headerdatei "MultiVideoClient.h" eingebunden und die Bibliothek "MultiVideoClient.lib" (bzw. "MultiVideoClientd.lib" für die Debug-Version) zum Projekt gelinkt werden. Die DLLs "MultiVideoClient.dll" (bzw. "MultiVideoClientd.dll"), und "MSysLib.dll" ("MSysLibd.dll") müssen an geeigneter Stelle zur Verfügung stehen, z.B. im Verzeichnis "C:\WINNT\System32\".

C.3) Beschreibung der Schnittstelle zum MVC-Modul

Die Headerdatei "MultiVideoClient.h" enthält alle relevanten Konstanten, Datentypen und Funktionen. Die Rückgabewerte der Funktionen sind vom Typ Integer, "0" bedeutet in der Regel Erfolg, Rückgabewerte < 0 sind Fehler. Nachfolgend werden alle Funktionen und Datentypen beschrieben. Bei fast allen Funktion ist eine Server-ID anzugeben. Jedem Videoserver im Netz muss eine eindeutige Nummer zugeteilt werden. Die Nummern sind von der Hauptanwendung zu verwalten.
Weiterhin wird bei einigen Funktionen ein Timeout-Wert (uint32 uiTimeout) benötigt. Dieser muss immer dann vorgegeben werden, wenn es um Austausch von Informationen mit einem Server geht. Er gibt die maximale Zeit an, wie lange der Client auf die Anwort vom Server warten soll. Der Timeout-Wert muss im Bereich [100...10000 ms] liegen und ein Vielfaches von 100 sein.

int MVC_InitMultiVideoClient(bool bStreamingMode);

Diese Funktion wird einmalig am Programmbeginn aufgerufen. Es werden interne Module und Datenstrukturen angelegt. Mit dem Parameter kann der Modus des MVC eingestellt werden. Im “Surveillance Mode“ (bStreamingMode = false) werden alle Streams gerendert. Im “Streaming Mode“ hingegen wird der MVC als reiner Empfänger eingesetzt. Die dekodierten Bilder werden in einem nach außen zugänglichen Puffer bereitgestellt, von wo sie abgeholt und weiterverwendet werden können.

int MVC_UnInitMultiVideoClient();

Diese Funktion wird einmalig am Programmende aufgerufen. Interne Strukturen werden gelöscht und Speicher wieder freigegeben. Alle Streams müssen zu diesem Zeitpunkt heruntergefahren worden sein .

Struct **SessionParameters**

Im Rtsp/Rtp-Sprachgebrauch wird ein Streaming-Vorgang als "Session" bezeichnet.
Dieser Begriff wurde für die ganze Übertragung übernommen. Daher die Bezeichnung "SessionParameters". Dieses Struct nimmt alle für eine Session relevanten Daten auf und wird in der Funktion MVC_InitVideoClient(...) übergeben.

uiServerID	eine eindeutige Nummer (ID) für eine Session. Die IDs müssen durch den Benutzer festgelegt und verwaltet werden.
strServerIP	IP-Adresse des Videoservers in Dot-Notation ("1.2.3.4")

usServerPort	RTSP-Port des Videoservers, muss zwischen 30200 und 30299 liegen.
strContent	Streaming Content als String, muss z. Z. "live.mp4" sein !
strServerName	Bezeichnung des Videoservers, z.B. "Kamera 1" oder "Schloßallee – Kamera 1" etc. Der Name dient der Unterscheidung der Streams und erscheint u.a. im Videofenster.
usRCPort	Serverport für die Fernsteuerung (Remote Control)
hMainWnd	Handle des bzw. eines Hauptanwendungsfensters. An dieses Fenster werden unten beschriebene Nachrichten gesendet. In der zugehörigen Fensterfunktion können diese Nachrichten abgefangen und bearbeitet werden. Als Handle kann für jede Session ein anderes Fenster angegeben werden. Eines für alle Sessions ist in der Regel aber ausreichend. Wird keine Kommunikation MVC -> Anwendung gewünscht, so kann hier 0 angegeben werden.
hExternalVideoWnd	Hier kann der Benutzer einen Fensterhandle vorgeben. Die Bilder werden dann in dieses Fenster gerendert, ein internes Videofenster wird nicht erstellt. Soll ein internes Fenster verwendet werden, so ist hier 0 anzugeben!
uiVideoPosition	wird das Video in ein externes Fenster gerendert, so kann hier die Position festgelegt werden. Dazu sollte die Funktion MVC_GetVideoPos(BYTE uiColumns, BYTE iRows, BYTE uiHorPos, BYTE uiVerPos) verwendet werden. Die ersten beiden Parameter geben die Anzahl an Videos an, welche in das Fenster gerendert werden. Die letzten beiden die Position des Videos an. Zum Beispiel werden bei MVC_GetVideoPos(2,2,0,0) 2x2 = 4 Videos in ein Fenster gerendert und das aktuelle erscheint oben links an der Position (0,0).
rVideoRegion	statt uiVideoPosition kann auch eine feste Region zum Rendern des Videos im externen Fenster vorgegeben werden. Bei : rVideoRegion = MVC_GetRectStruct(10,10,320,240); wird das Video an der Stelle (10,10) bis (330,250) innerhalb des Fensters gerendert. Somit ist der Anwender bei der Gestaltung der grafischen Bedienoberfläche sehr flexibel.

Die folgenden Typen sind IDs für Windows-Nachrichten. Sie dienen der Kommunikation MVC->Hauptanwendung. D.h. der MVC versendet bei bestimmten Ereignissen diese Nachrichten an das Fenster, welches durch hMainWnd gegeben ist. Dies ist aber optional. Wird keine Kommunikation gewünscht, so kann hier 0 angegeben werden. Als Parameter werden die beiden Windowsprogrammierern bekannten Integerwerte wParam und lParam mitgeliefert. wParam ist dabei immer die Server-ID und dient somit der Zuordnung der Nachricht zu einem VideoClient (also dem Absender). lParam hat verschiedene Bedeutungen, je nach Nachricht.

Die Nachrichten können z.B. mit der WinAPI-Funktion RegisterWindowMessage (LPCTSTR lpString) registriert werden und sind dann systemweit einmalig :

uiCreationMsg	lParam -> (Int32)hWnd wird lParam mit HWND gecastet, so erhält man den Handle des Fensters des Videoclients. Die Nachricht wird kurz nach Erzeugung des Videofensters versendet. Die Hauptanwendung erhält somit Kenntnis über alle internen Videofenster (für Steuerungszwecke).
uiNotificationMsg	Diese Nachricht wird versendet, wenn der Benutzer mit der rechten Maustaste in ein internes Videofenster klickt. lParam enthält die Mauskoordinaten. x und y können mit GET_X_LPARAM(lParam) und GET_Y_LParam(lParam) extrahiert werden. An diesen Koordinaten kann dann die Hauptanwendung z.B. ein Bedienmenü öffnen.
uiCloseMsg	Diese Nachricht wird versendet, wenn der Benutzer auf den Close-Button des internen Videofensters (kleines Kreuz oben rechts) klickt. Das Videofenster wird daraufhin nicht(!) geschlossen. Vielmehr wird die Anwendung über den Wunsch des Benutzers informiert, die Session zu beenden. Sie kann daraufhin die Session herunterfahren und somit des Fenster schließen.
uiConnClosedMsg	Im Gegensatz zur vorigen Nachricht wird diese Nachricht gesendet, wenn der Stream serverseitig beendet wurde. D.h. es kommen keine Videodaten mehr (4 Sekunden). Dies kann natürlich auch bei akuten Netzproblemen der Fall sein. Die Hauptanwendung sollte mit einer Beendigung der Session reagieren.

int MVC_InitVideoClient(SessionParameters Sessionparams);

Mit dieser Funktion wird eine Streaming-Session initialisiert. Diese Funktion wird pro Session nur einmal aufgerufen. Als Parameter wird o. g. Struct mit Daten der Session übergeben.

int MVC_StartStreaming(uint32 uiServerID, bool bCheckBitrates,
bool bAutomaticBRControl);

Diese Funktion startet den Streamingvorgang des Clients, welcher durch die übergebene ID (uiServerID) repräsentiert wird. Die Funktion darf erst nach MVC_InitVideoClient() aufgerufen werden. bCheckBitrates gibt vor, ob der MVC die Gesamtbitrate des Netzes überprüfen soll.
Wird diese mit dem neuen Stream überschritten, so kann er nicht gestartet werden. Ist bAutomaticBRControl jedoch aktiviert, so wird noch überprüft, ob die Gesamtbitrate durch Herunterregelung der anderen Streams eingehalten werden kann. Im positiven Fall werden die Bitraten der anderen Streams heruntergefahren (je nach Priorität) und der neue Stream kann gestartet werden.

int MVC_StopStreaming(uint32 uiServerID, bool bAutomaticBRControl);

Mit dieser Funktion wird das Streaming beendet.

Ist bAutomaticBRControl gesetzt, so überprüft der MVC, ob er die freigesetzte Bitrate anderen Streams zuteilen kann. Diese werden dann maximal bis zu ihrer Profilbitrate erhöht.

int MVC_UnInitVideoClient(uint32 uiServerID);

Zum Schluß wird der SVC heruntergefahren, d.h. interne Module werden beendet und Datenstrukturen gelöscht. Vorher muss das Streaming mit MVC_StopStreaming() beendet worden sein.

int MVC_GetRTPVideoStatistics(uint32 uiServerID, uint32* puiReceivedPackets,
uint32* puiMissingPackets,
uint32* puiRetransmittedPackets,
uint32* puiLostPackets);

Rückgabewerte dieser Funktion sind statistische Daten. Es wird für einen Client die Gesamtzahl an empfangenen und vermißten Paketen ermittelt. Wurde für den Client der Retransmission-Mechanismus aktiviert, so werden vermißte Pakets von Server neu angefordert und somit noch einmal oder mehrmals übertragen. Der letzte Wert enthält die Anzahl der trotz wiederholter Übertragung verlorenen Pakete.

int MVC_SaveCapturedFrame(uint32 uiServerID);

Bei einem Klick mit der rechten Maustaste in ein Videofenster wird das zu diesem Zeitpunkt angezeigte Bild zwischengepuffert. Es kann nun als 24-Bit Bitmap (*.bmp) gespeichert werden. Der Name der Bitmapdatei wird automatisch aus Server-Name und Server-Timestamp zusammengesetzt. Die Funktion kann aber zu jeden beliebigen Zeitpunkt aufgerufen werden. Es wird dann das Frame gespeichert, welches sich gerade im Puffer befindet.

int MVC_RecordVideoStream(uint32 uiServerID, bool bRecord);

Mit dieser Funktion kann eine Aufzeichnung des Videostreams gestartet (bRecord = true) und gestoppt (bRecord = false) werden. Der Videostream wird als *.mp4 gespeichert. Der Dateiname wird automatisch generiert. Zu beachten ist, dass der Stream nicht sofort aufgenommen wird, sondern erst nach Eintreffen des nächsten I-Frames. Im internen Videofenster erscheint für die Dauer der Aufnahme ein "[R]". Der mp4-Datenstrom kann mit einem geeigneten Player abgespielt werden (z.B. Mp4Play -> stoffers, stabernack@hhi.de).

int MVC_ChangeWindowSize(uint32 uiServerID, uint32 uiSize);

Hiermit kann die Größe des Videofensters manipuliert werden. Für uiSize sind Werte zwischen 25 und 400 möglich. Diese Angabe ist in Prozent. Die Bezugsgröße für die Prozentangabe ist die Standardbildgröße des Streams.

int MVC_GetActualBitrate(uint32 uiServerID, uint32* puiBitrate);

Jeder Videoclient mißt intern die Bitrate des Videostreams. Diese kann bei Bedarf abgefragt werden. Die Bitrate ist ein über der Zeit gemittelter Wert. Kurzzeitänderungen werden nicht erfaßt.

int MVC_ResetBitrateMeasurement(uint32 uiServerID);

Hiermit wird die interne Messung der Bitrate neu gestartet. Dies ist z.B. nach Umschaltung der Bitrate beim Encoder erforderlich.

int MVC_GetActualFramerate(uint32 uiServerID, float* pfFramerate);

Jeder Videoclient mißt intern die Bildrate des Videostreams. Diese kann bei Bedarf abgefragt werden.

int MVC_ResetFramerateMeasurement(uint32 uiServerID);

Hiermit wird die interne Messung der Bildrate neu gestartet.

int MVC_GetServerStatistics(uint32 uiServerID,
uint32* puiLostUncompressedFrames,
uint32* puiLostEncodedFrames,
uint32 uiTimeout);

Bei Aufruf dieser Funktion werden vom Server statistische Daten abgefragt. Eine ständige Erhöhung der Anzahl an verlorenen Frames deutet i.A. auf schlechte Prozessor- und/oder Netzperformance hin. Die Funktion kann nur für bereits laufende Streams aufgerufen werden.

int MVC_GetServerBitrate(uint32 uiServerID, string strServerIP, uint32 uiServerPort
uint32* puiBitrate, uint32 uiTimeout)

Mit dieser Funktion kann die aktuelle Bitrate des im Server integrierten MPEG-4-Encoders abgefragt werden. Dazu werden IP-Adresse und Port für die Fernsteuerung des Servers benötigt. Die Funktion kann zu einem beliebigen Zeitpunkt aufgerufen werden.

int MVC_SetServerBitrate(uint32 uiServerID, string strServerIP, uint32 uiServerPort
uint32 uiBitrate, uint32 uiTimeout,
bool bAutomaticBRControl)

Mit dieser Funktion kann die aktuelle Bitrate des MPEG-4-Encoders neu eingestellt werden. Der Encoder wird kurz gestoppt und mit neuer Bitrate wieder gestartet. Der Client kann aber weiterlaufen. Zur Versendung des Kommandos an den Server wird dessen IP und Port benötigt. Ist die interne Streamkontrolle aktiviert, so überprüft der MVC bei einer Erhöhung der Bitrate, ob die maximale Gesamtbitrate noch eingehalten wird. Ist dies nicht der Fall, so kann die Bitrate nicht geändert werden, es sei denn bAutomaticControl ist aktiviert. In diesem Fall wird zusätzlich überprüft, ob durch Senkung der anderen Bitraten die aktuelle erhöht werden

kann. Im Gegenzug können bei einer Senkung der aktuellen Bitrate evtl. die anderen Bitraten erhöht werden.

int MVC_GetServerProfile(uint32 uiServerID, string strServerIP, uint32 uiServerPort uint32* puiProfile, uint32 uiTimeout)

Mit dieser Funktion kann das aktuelle Videoprofil des Servers abgefragt werden.

int MVC_SetServerProfile(uint32 uiServerID, string strServerIP, uint32 uiServerPort uint32 uiProfile, uint32 uiTimeout, bool bCheckBitrates, bool bAutomaticBRControl)

Mit dieser Funktion wird das Videoprofil des Servers geändert. Das neue Profil muss kompatibel zum Format der unkomprimierten Bilder sein. Da der Server zur Änderung des Profiles neu gestartet wird, muss der Client vorher gestoppt und nach erfolgreicher Änderung wieder neu gestartet werden. Ist bCheckBitrates aktiviert, so überprüft der MVC, ob durch die Änderung des Profiles die maximale Gesamtbitrate überschritten wird (Stream Control Modul muss aktiviert sein). In diesem Fall gibt bAutomaticBRControl an, ob bei Überschreitung der maximalen Bitrate die anderen Bitraten gesenkt werden sollen. Die maximale Senkung beträgt –2N% bei Streams mit Priorität 1, -N% bei Priorität 2 und 0% bei Priorität 3, jeweils bezogen auf die Profilbitrate. N kann in einer Konfigurationsdatei vorgegeben werden (Anhang E). Im Gegenzug können die Bitraten der anderen Streams auch erhöht werden, sollte das neue Profil eine geringere Bitrate haben (Erhöhung maximal bis zur Profilbitrate).

int MVC_GetActualFrame(uint32 uiServerID, char* pchFrameBuffer, uint32* puiWidth, uint32* puiHeight, uint32* puiTimestamp)

Diese Funktion dient dem direkten Empfang der dekodierten Bilder im Format YC_bC_r 4:2:0. Sie wird vor allem im “streaming mode“ verwendet. In diesem Modus werden die Bilder nicht in einem Fenster angezeigt. Vielmehr können sie zwecks Weiterverarbeitung in einen vom Benutzer bereitgestellten externen Puffer (pchFrameBuffer) kopiert werden. Bei Aufruf der Funktion müssen weiterhin Pointer für Breite, Höhe und Timestamp des kopierten Frames bereitgestellt werden. Die Länge des YUV-Frames beträgt dann Breite x Höhe x 1.5. Der externe Puffer muss natürlich eine ausreichende Größe besitzen. Der empfangene Timestamp ist die Serverzeit und kann je nach Serveranwendung ein anderes Format haben. Anhand dieses Timestamps ist aber in der Regel eine Erkennung verlorener Frames möglich. Um die Erkennung zu unterstützen, werden vom RTP-Server die oberen 5-Bit des 32-Bit Timestamps mit einer fortlaufenden Nummer belegt. Die unteren 27 Bit enthalten die vom Framegrabber vorgegebene Zeitinformation.

Jedes Frame kann nur einmal abgeholt werden. Somit kann die Funktion auch bei ungenügender Kenntnis der Framerate sicher verwendet werden.

int MVC_SetVideoPosition(uint32 uiServerID, uint32 uiVideoPosition)

Hiermit kann die Position des Videos innerhalb des externen Fensters neu gesetzt werden. Der 32-Bit-Wert uiVideoPosition enthält folgende Werte :

Byte 3 (MSB) -> horizontale Anzahl an Videos im Fenster : 1..M

Byte 2 -> vertikale Anzahl an Videos : 1..N

Byte 1 -> horizontale Position : 0..M-1

Byte 0 -> vertikale Position : 0..N-1

int MVC_SetVideoRegion(uint32 uiServerID, RECT rVideoRegion)

Ändert die Region des Videos innerhalb des externen Fensters relativ zur oberen linken Ecke. Im Gegensatz zur Videoposition ist die Videoregion frei (pixelgenau) wählbar.

uint32 MVC_GetVideoPos(BYTE uiColumns, BYTE uiRows, BYTE uiHorPos, BYTE uiVerPos)

Eine Hilfsfunktion zum Berechnen des Parameters uiVideoPosition im Struct SessionParameters und des zweiten Parameters der Funktion MVC_SetVideoPosition(..)

RECT MVC_GetRectStruct (long lLeft, long lTop, uint32 uiWidth, uint32 uiHeight)

Eine Hilfsfunktion zum Setzen des Parameters rVideoRegion im Struct SessionParameters. Es werden die Koordinaten der linken, oberen Ecke sowie die Regionenbreite und –höhe benötigt.

int MVC_GetDecoderStatistics(uint32 uiServerID, uint32* puiReceivedVOPs, uint32* puiLostVOPs)

Mit dieser Funktion werden Daten vom Decoder abgefragt. puiReceivedVOPs enthält die Anzahl an korrekt empfangenen Frames, puiLostVOPs die Anzahl der verlorenen. Die Gesamtzahl der versendeten Frames ergibt sich somit durch Addition beider Werte.

int MVC_EnableRTPRetransmission(uint32 uiServerID, bool bEnable, uint32 uiRTTimeout, uint32 uiTotalTimeout)

Mit dieser Funktion kann der Retransmission-Mechanismus im RTP-Client aktiviert/ deaktiviert werden. Wird anhand der Sequenznummer eines Paketes ein Verlust vorangegangener Pakete festgestellt, so werden diese vom Server neu angefordert. Auf diese Weise können Paketverluste drastisch verringert werden. uiRTTimeout gibt die Anzahl an Millisekunden an, die der Client auf eine wiederholte Sendung warten soll (Round Trip Time). uiTotalTimeout gibt, wie lange der Client maximal warten soll, bis er das vermißte Paket aufgibt. Es gilt :

uiRTTimeout ≤ uiTotalTimeout ≤ 1000

maximale Anzahl an Retransmissions = uiTotalTimeout/uiRTTimeout

int MVC_EnablePacketLossSimulation (uint32 uiServerID, bool bEnable, double fPLost, double fPLostLost)

Da kein Netz eine fehlerfreie Übertragung garantieren kann, dient diese Funktion der Simulation von Paketverlusten. Dadurch kann das Streamingsystem auf seine Fehleranfälligkeit und Fähigkeiten zur schnellen wiederholten Übertragung (Delay/RTT) getestet werden. Der RTP-Client simuliert Paketverluste mit Hilfe eines binären Markov-Modells mit den beiden Zuständen Verlust <-> kein Verlust. Als Parameter müssen zwei Wahrscheinlichkeiten vorgegeben werden. Beide liegen zwischen 0 und 1 :

fPLost gibt die Verlustrate der Pakete vor (Beispiel : 0.01 -> jedes hundertste Paket)

fPLostLost gibt die Wahrscheinlichkeit für einen Paketverlust an, wenn das vorangegangene Paket bereits "verloren" wurde. Ein kleiner Wert verursacht somit vor allem Einzelverluste, ein großer führt zu Burstfehlern.

int MVC_SetTimeStampResolution(uint32 uiServerID, uint32 uiTSR)

Da in der Kopfzeile der Videofenster die Serverzeit angezeigt wird, muss der Client wissen, in welchem Format, bzw. in welcher zeitlichen Auflösung der Timestamp ankommt. Mit uiTSR wird diese beim Client eingestellt. Der Wertebereich geht von 1 und 60000 Millisekunden.

Bsp. : 1000 bedeutet, dass der Servertimestamp in Sekunden gezählt wird.

Der Defaultwert nach dem Start eines Clients ist immer : 1000 ms = 1 Sekunde.

int MVC_EnableStreamControl(HWND hMainWnd, uint32 uiStreamControlMsg, string strConfigFile, uint32 uiControlRate, bool bEnable)

Mit dieser Funktion wird die interne Streamkontrolle aktiviert/deaktiviert. Die Aktivierung kann nur erfolgen, wenn kein Stream läuft. Zur Ermittlung aller maximalen Bitraten und weiterer Daten wird der Name einer Netzwerkkonfigurationsdatei benötigt (siehe Anhang E).

Eine Aufgabe des Stream Control Modules ist die Überprüfung und ggf. Regelung aller Bitraten, wenn der Benutzer die Bitrate oder das Profil eines Streams ändert oder einen Stream startet/stoppt.

Des Weiteren erhält das interne Modul statistische Daten der Streams. Bei Frameverlusten können die Bitraten oder Videoprofile der beteiligten Streams heruntergefahren werden. Dazu sendet der MVC eine Nachricht (uiStreamControlMsg) an die Hauptanwendung (hMainWnd). Der erste Parameter (lParam) ist die ID des Streams, der geändert werden soll. Der zweite 32-Bit-Parameter (wParam) enthält das neue Profil (HIWORD) und/oder die neue Bitrate (LOWORD). Die Hauptanwendung kann dann mit diesen beiden Werten die Funktion MVC_ControlStream() aufrufen. Sind die Parameter aber 0, so signalisiert das SCM damit, dass vorläufig keine weiteren Kontrollnachrichten kommen. Es ist sinnvoll, bei Eintreffen von Streamkontrollnachrichten die externe Steuerung durch den Benutzer zu deaktivieren.

uiControlRate gibt vor, in welchen Zeitabständen (sec) die Verlustraten dem Kontrollmodul mitgeteilt werden. Mit dieser Taktrate erfolgt dann auch die automatische Steuerung.

int MVC_ControlStream(uint32 uiServerID, uint32 uiStreamControl)

Mit dieser Funktion werden Profil oder Bitrate eines Streams ohne interne Überprüfung durch das Kontrollmodul geändert. Der Aufruf der Funktion sollte nach Empfang einer Nachricht vom Stream-Kontrollmodul erfolgen. Die oberen 16 Bit von uiStreamControl enthalten die Nummer des neuen Profiles, die unteren die neue Bitrate in kbits/s. Die Funktion wird mit den beiden Parametern wParam und lParam der Nachricht uiStreamControlMsg aufgerufen.

Die folgenden Funktionen dienen der Steuerung des Servers oder der Abfrage von Serverinformation. Die ersten Parameter sind jeweils die ServerID, die IP-Adresse und der Remote Control Port.
Weitere Werte dienen dem Setzen oder dem Empfang von Serverparametern.

int MVC_KillServer(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32 uiTimeout)

Beendet die Serveranwendung. Ein eventuell laufender Stream wird vorher gestoppt.

int MVC_StartServer(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32 uiTimeout)

Startet die Servermodule.

int MVC_StopServer(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32 uiTimeout)

Stoppt die Servermodule. Ein eventuell laufender Stream wird vorher gestoppt.

int MVC_LockServer(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32 uiTimeout)

Sperrt den Server gegenüber allen nachfolgenden Fernsteuerkommandos. Diese Funktion kann also nur einmal aufgerufen werden. Ein Entsperren ist nicht möglich.

int MVC_CheckAvailability(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32* puiAvailability, uint32 uiTimeout)

Prüft beim Server, ob dieser für die Übertragung bereit ist. Rückgabewerte für *puiAvailability sind:
0 – Server ist aus bzw. nicht erreichbar
1 – Server ist bereit, aber es kommen keine Bilder vom Framegrabber
2 – ein oder mehrere Module des Servers laufen nicht
3 – Server ist bereit

int MVC_SetRTSPPort(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32 uiRTSPPort, uint32 uiTimeout)

Ändert den RTSP-Port beim Server.

int MVC_GetRTSPPort(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32* puiRTSPPort, uint32 uiTimeout)

Ermittelt den RTSP-Port des Servers.

int MVC_GetUserDefinedProfile(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32* puiWidth, uint32* puiHeight, uint32* puiFramerate, uint32* puiBitrate, uint32* puiIFP, uint32 uiTimeout)

Ermittelt die Einstellungen des benutzerdefinierten Profiles beim Server. Dieses wird benutzt, wenn die Bilddaten nicht mit einem vordefinierten Format übertragen werden können.

int MVC_SetFilter(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32 uiFilter, uint32 uiTimeout)

Stellt einen Filtertyp beim Server ein. TP-Filter werden von Server beim Unterabtasten von Bildern verwendet, um die Spiegelungsfehler zu verringern. Gültige Werte sind :
0 = kein Filter, 1 = Haar-Filter, 2 = Dreiecksfilter, 3 = steilflankiges 7-Tap-Filter

int MVC_Zoom(uint32 uiServerID, string strServerIP, uint32 uiServerPort, uint32 uiLeft, uiTop, uint32 uiTimeout)

Veranlaßt den Server, einen 2fach Zoom durchzuführen und nur den durch die obere, linke Koordinate festgelegten Bildausschnitt zu übertragen.

C.4) Zusammenfassung

Das beschriebene Interface des MultiVideoClients bietet eine Vielzahl von Funktionen zur Erstellung eigener Multiview-Videostreaming-Anwendungen. Neben den Hauptfunktionen zum Starten und Steuern der Streams können zudem wichtige statistische Daten überwacht werden. Zur Verwendung des MVC in einem Überwachungssystem besteht die Möglichkeit, Einzelbilder oder das komplette Video zu speichern.
Die intern erstellten Fenster basieren auf der Win32-API. Die Größe des Fensters entspricht zunächst der Bildgröße + Fensterrahmen, kann aber fast stufenlos geändert werden. Die Überschrift der Fenster ist der bei der Initialisierung übergebene Name sowie der Servertimestamp und ggf. weitere Informationen. Die Videos können optional auch in ein oder mehrere externe Fenster gerendert werden. Diese sind durch die Hauptanwendung zu erstellen und zu verwalten. Der Vorteil ist dabei die freie Gestaltung grafischer Bedienoberflächen.

Version : 1.1 Stand 10/2003
Autor : Patrick Voigt

Anhang D – TraViS

D.1) Einleitung

Das Programm TraViS (Traffic Video Surveillance) ist eine menügeführte Anwendung zum parallelen Empfang und zur Anzeige mehrerer Videostreams. Erstellt wurde das Programm als MFC-Dialog-Anwendung. Es setzt auf den als DLL verfügbaren MultiVideoClient auf und bietet einfachen Zugriff auf dessen Features. Zum Starten sind neben der Datei "TraViS.exe" die beiden Dlls "MultiVideoClient.dll" und "MSysLib.dll" erforderlich. Alle drei Dateien gibt es auch als Debug-Version in der Form "*d.*".
Im Folgenden wird die Bedienung des Programmes Schritt für Schritt erklärt.

D.2) Das Hauptmenü und dessen Untermenüs

Nach Starten der Datei TraViS.exe erscheint zunächst folgendes zentrales Menü :

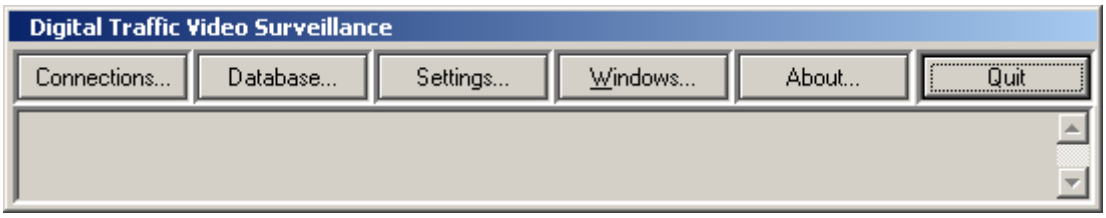


Abb. D.1 : Hauptmenü

Dieses Menü enthält neben diversen Buttons auch ein Textfeld, in welchem während des Programmablaufes wichtige Statusmeldungen ausgegeben werden. Die aktuellste steht an oberster Stelle.

Zunächst einmal benötigt TraViS eine Datenbank mit allen relevanten Daten, die zur Verbindungsaufnahme mit den Kameraservern benötigt werden. Dazu wird über "Database..." ein neuer Dialog geöffnet, welcher die Möglichkeit zum Einladen einer Datenbank aus einer Datei bietet :

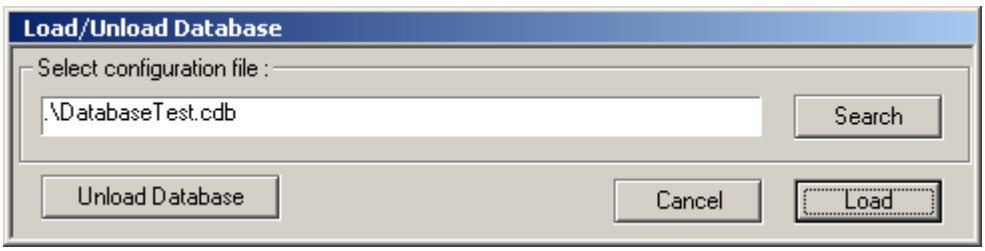


Abb. D.2 : Menü zum Laden einer Datenbank

Es wird eine Textdatei geladen, welche folgenden Aufbau haben muss :

Überschrift :***SERVER DATABASE***
unsigned int : Server-ID
string : Servername / -Bezeichnung
string : Server-IP in Dot-Form (z.B. "1.2.3.4")
unsigned int : Port des Servers für Remote Control (zwischen 30120 und 30199)
unsigned int : RTSP-Port des Servers (zwischen 30200 und 30299)
nächster Datensatz beginnend mit der ID

Alle Einzeldaten sind durch <Enter> getrennt, nach dem letzten Datensatz darf maximal noch ein Enterzeichen stehen. Die Endung der Datei kann beliebig sein, die Standardendung ist "*.cdb".
Hier ein Beispiel mit 2 Datensätzen :

```
***SERVER DATABASE***
1
Erste Kamera
172.19.50.166
30120
30200
2
Zweite Kamera
172.19.50.185
30120
30201
```

Alle eingetragenen IDs müssen einmalig sein, keine ID darf doppelt vergeben werden. TraViS benötigt diese eindeutigen Nummern u.a. für den internen Datenaustausch. Der Servername erscheint auf allen Videofenstern und diversen anderen videospezifischen Menüs. Somit können diese leicht identifiziert bzw. zugeordnet werden. Aus diesem Grunde ist es sinnvoll, kurze und prägnante Bezeichnungen zu vergeben. Werden beim Einlesen der Datei Fehler festgestellt, so wird der Vorgang abgebrochen und es erscheint eine entsprechende Meldung im Log-Fenster. Die bis dahin korrekt eingelesenen Datensätze sind aber gespeichert. Nach Laden einer Datei können weiteren Dateien geladen werden. Die darin enthaltenen Datensätze dürfen aber keine ID besitzen, welche bereits bekannt ist.
Des Weiteren kann eine geladene Datenbank auch wieder gelöscht werden. Dies ist allerdings nur möglich, wenn kein Videostream läuft.
Wurden erfolgreich Datensätze geladen, so können über den Menüpunkt "Connections..." nun Videostreams gestartet werden. Es erscheint folgendes Menü :

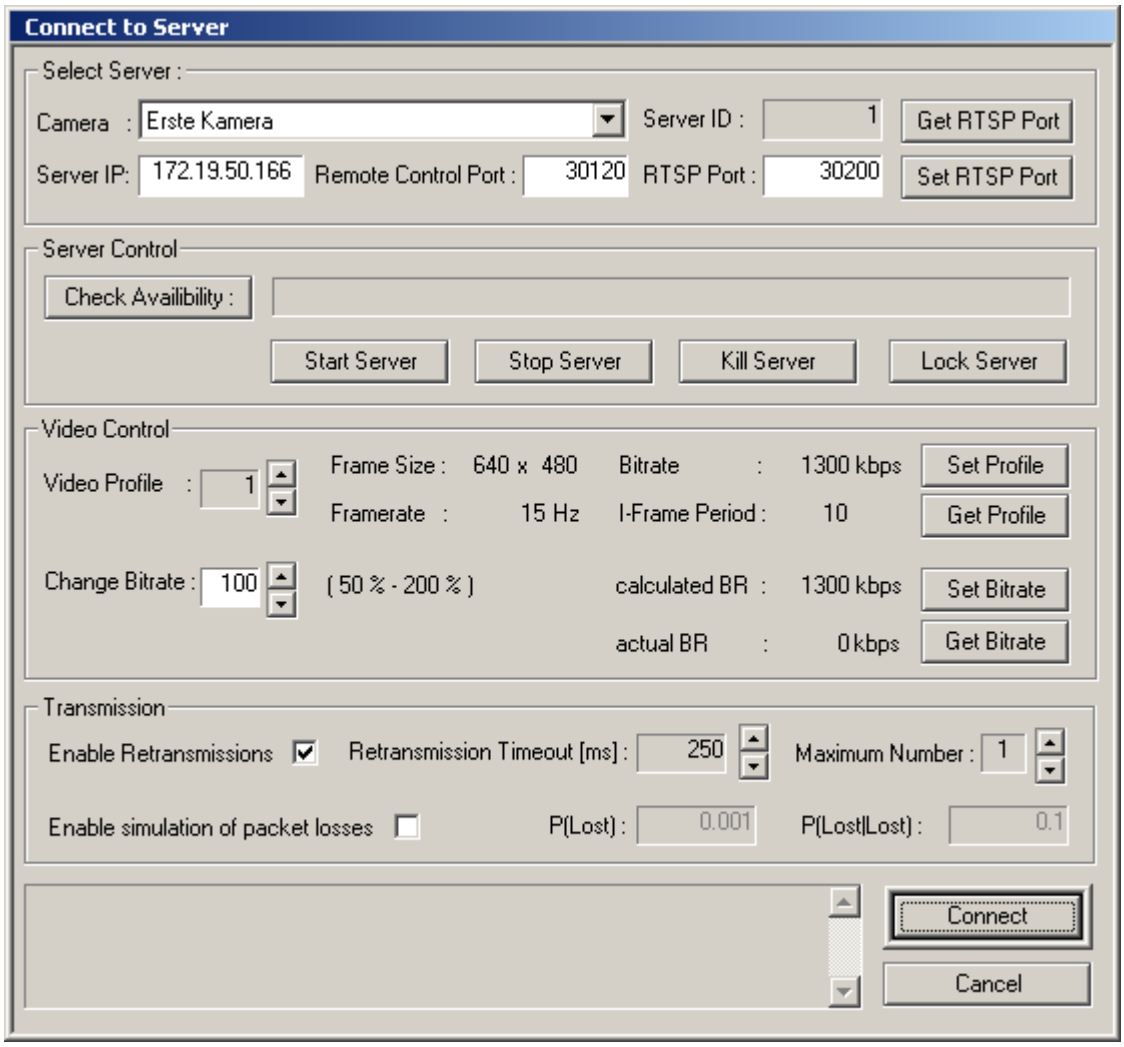


Abb. D.3 : Menü zum Öffnen und Steuern einer Verbindung

Unter "Select Server" gibt es zunächst eine Liste mit allen möglichen Verbindungen, welche in der Datenbank eingetragen sind. Hier wählt man den gewünschten Videoserver aus und es werden automatisch alle relevanten Daten in den benachbarten Feldern angezeigt. Bis auf die ID kann der Benutzer diese Daten bei Bedarf ändern. Außerdem kann vom Server über den Remote Control Kanal der aktuelle RTSP-Port abgefragt oder auch eingestellt werden, falls dieser nicht mit dem Datenbankeintrag übereinstimmt. Sind alle Einstellungen korrekt, so kann mittels "Connect" der Verbindungsaufbau erfolgen. Steht die Verbindung aber bereits, so kann an dieser Stelle auch ein Abbruch veranlaßt werden. In diesem Fall ändert sich der "Connect"-Button zu "Disconnect".

Neben der Verbindungsaufnahme bietet das Menü auch diverse Einstellmöglichkeiten des Videoservers und des Videostreams. Unter "Server Control" kann ein Server gestoppt, gestartet oder geschlossen werden. Letzteres bedeutet serverseitig ein Beenden des Prozesses! Darum sollte dies mit Vorsicht benutzt werden. Des Weiteren kann die Verfügbarkeit des Servers abgefragt werden. Wurde ein Server z.B. gestoppt, so ist die Verfügbarkeit negativ, d.h. es kann keine Videoverbindung aufgenommen werden. Mit "Lock Server" wird die Fernbedienung serverseitig abgeschaltet. Er nimmt somit keine Befehle oder Anfragen des Clients mehr entgegen. Ist der Server zu diesem Zeitpunkt allerdings verfügbar, so kann zumindest noch ein Videostream gestartet werden. "Lock Server" ist

somit mit Vorsicht zu gebrauchen, denn viele Features von TraViS sind danach nicht mehr verfügbar! Unter "Video Control" können Parameter des Videostreams noch vor der Verbindungsaufnahme geändert werden. Die Grobeinstellung erfolgt durch Auswahl eines Videoprofiles. Diese Profile bieten voreingestellte Werte für Auflösung, Bildrate, Bitrate und Intraframe-Periode. Mit "Set Profile" wird der Server angewiesen, auf das gewünschte Profil umzuschalten. Dies gelingt allerdings nur, wenn die unkomprimierten Eingangsbilder beim Server auch mit dem gewünschten Profil kompatibel sind. So können z.B. 320x240-Frames nicht in einen 352x288-Stream codiert werden. "Mit "Get Profile" und "Get Bitrate" werden das aktuelle Profil und die aktuelle Bitrate des Servers abgefragt. Eine Feineinstellung kann mit "Set Bitrate" vorgenommen werden. Der Einstellbereich liegt bei 50 – 200 % bezogen auf die im Profil vordefinierte Bitrate.

Im vierten Abschnitt geht es um Kanalfehler. Da kein Netz eine fehlerfreie Übertragung garantiert, kann es zu Paketverlusten kommen. Die betroffenen Frames müssen verworfen werden und nachfolgende P-Frames sind ebenfalls gestört. Eine erneute Anforderung von verlorenen Paketen führt aber meist zu einer deutlichen Verbesserung. Dazu werden geeignete Werte für Timeout (Round Trip Time) und Anzahl der Wiederholungen eingestellt.

Um auch bei annähernd fehlerfreiem Kanal die eingestellten Werte überprüfen zu können, wurde ein Simulator für Paketverluste implementiert. Dazu sind zwei Wahrscheinlichkeiten anzugeben : Die Verlustwahrscheinlichkeit P_{Lost} und die bedingte Wahrscheinlichkeit $P_{Lost|Lost}$ (= Verlustwahrscheinlichkeit wenn vorheriges Paket bereits verloren wurde).

Zu allen Aktionen im Menü werden Statusmeldungen im Log-Fenster ausgegeben. Weiterhin werden in der Datei "ClientCommandLog.txt" alle an die Server versendeten Kontroll-Kommandos sowie die Bestätigungen der Server mitprotokolliert.

Der Menüpunkt "Settings..." bietet die Möglichkeit, diverse globale Einstellungen vorzunehmen. In der aktuellen Version öffnet sich folgendes Menü :

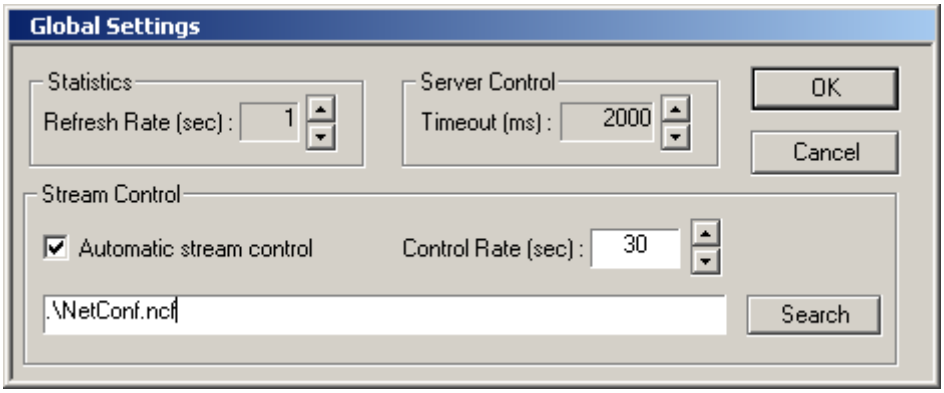


Abb. D.4 : Menü für globale Einstellungen

Die "Refresh Rate" gibt die Periode an, mit welcher statistische Videodaten (z.B. aktuelle Bitrate) und Paketverlustraten erfasst und angezeigt werden sollen. Der Timeoutwert unter "Server Control" gibt an, wie lange der Client bei Benutzung der Server-Fernsteuerung auf die Antwort des Servers warten soll. Bei größeren Netzen sollte dieser Wert z.B. höher angesetzt werden.

Das integrierte Modul zur Streamkontrolle kann im unteren Teil aktiviert/deaktiviert werden. Bei Aktivierung ist eine Datei anzugeben, welche alle Konfigurationsdaten für das Netz beinhaltet. Eine

Beispieldatei ist in Anhang E. Die Control Rate gibt an, in welchen Zeitabständen Paket- und Frameverluste ausgewertet werden sollen. Auftretende Verluste führen dann i.A. zu einer automatischen Regelung der Bitraten der laufenden Streams. Des Weiteren prüft das Kontrollmodul bei Aktivierung, ob bei Bitratenänderungen an einem Stream auch andere Streams geändert werden können oder müssen, um eine Überschreitung der festgelegten maximalen Gesamtbitrate zu verhindern.
Mit dem Menüpunkt "Windows..." können bestimmte Eigenschaften aller Videofenster zusammen gesteuert werden. Die gewünschte Aktion wird einfach markiert und bestätigt :

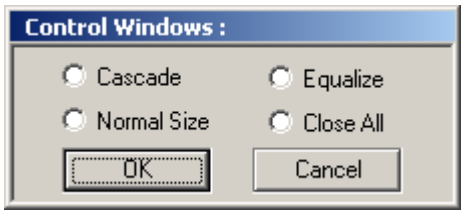


Abb. D.5 : Kontrollmenü für Videofenster

"Cascade" ordnet alle Videofenster hintereinander an. Es wird mit dem ersten gestarteten Stream begonnen. Mit "Normal Size" werden alle Videofenster auf Standardgröße zurückgesetzt. Die Standardgröße ist die Bildgröße, mit welcher der Videostream codiert wurde. Mit "Equalize" werden alle Videofenster auf die Standardgröße des kleinsten Videostreams gesetzt. "Close All" schließlich beendet alle Streams und schließt alle Fenster.
Die Bedeutungen der beiden letzten Buttons des Hauptmenüs "About" und "Quit" sind selbsterklärend.

D.3) Videospezifische Fenster und Menüs

Nach Starten eines Streams werden die Bilder empfangen, dekodiert und angezeigt. Dazu wird intern ein Fenster erzeugt, welches die Größe der eintreffenden Bilder hat. Dies kann einen Moment dauern, da das Fenster erst nach dem ersten dekodierten I-Frame erstellt werden kann. Es sieht dann z.B. folgendermaßen aus :

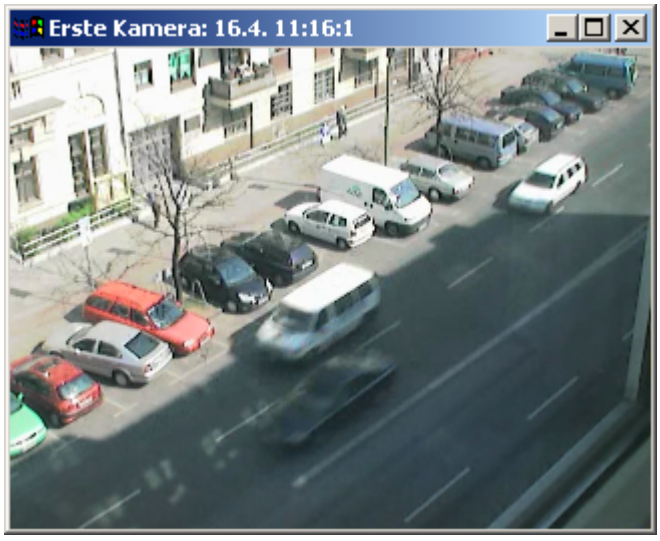


Abb. D.6 : Videofenster

Das Fenster kann standardmäßig maximiert und minimiert werden. Ein Druck auf den Close-Button oben rechts bewirkt ein Stoppen des Streams und ein Schließen des Fensters.
Zur Identifizierung wird die Serverbezeichnung aus der Datenbank angezeigt. Außerdem wird der vom Server mitgelieferte Zeitstempel als Uhrzeit angezeigt.

Klickt der Benutzer mit der rechten Maustaste in das Fenster, so öffnet sich folgendes Menü :

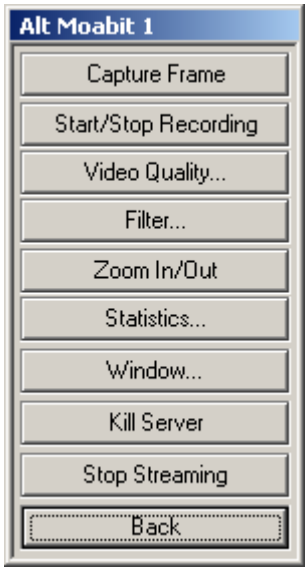


Abb. D.7 : Kontrollmenü für einen Stream

Mit "Capture Frame" wird ein Einzelbild als 24-Bit-Bitmap (*.bmp) gespeichert. Der Dateiname wird automatisch aus Server- bzw. Kamerabezeichnung und Datum/Uhrzeit zusammengesetzt. Gespeichert wird das Bild, welches zum Zeitpunkt des rechten Mausklicks im Speicher war. Dieses wurde intern gepuffert.
"Start/Stop Recording" startet und beendet die Aufnahme eines Videostreams als MPEG-4-Datei (*.mp4, nur Elementardatenstrom). Während der Aufnahme erscheint das Symbol "[R]" in der Überschrift des Videofensters. Der Name der MPEG4-Datei wird ebenfalls automatisch erzeugt. Die Datei wird im aktuellen Verzeichnis der Anwendung angelegt. Der Beginn der Aufnahme erfolgt erst, wenn das nächste I-Frame eingetroffen ist. Dies kann je nach eingestellter I-Frame-Periode unter Umständen 1-2 Sekunden dauern. Die Anzahl an gleichzeitig aufnehmbaren Streams hängt im Wesentlichen von der Festplattengeschwindigkeit ab.

Bei Auswahl des Menüpunktes "Video Quality..." öffnet sich ein weiterer Dialog :

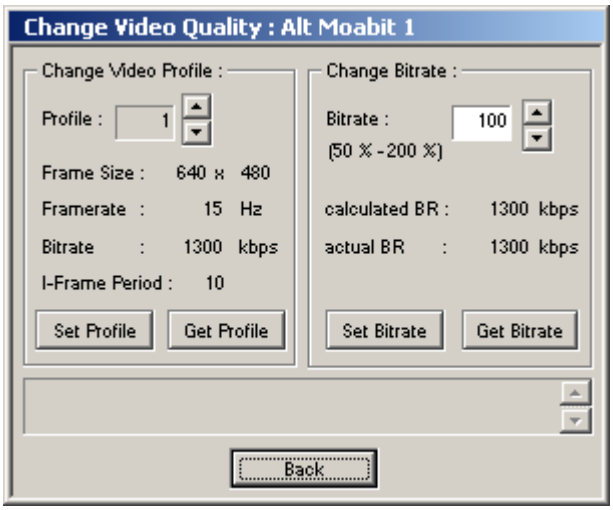


Abb. D.8 : Änderung des Videostreams

Vorher wird allerdings vom Server automatisch das aktuelle Videoprofil und die Bitrate abgefragt.
Mit diesem Menü kann nun der Videostream in zwei Stufen manuell verändert werden. Zunächst erfolgt eine grobe Einstellung der Bitrate durch Wahl eines anderen kompatiblen Videoprofiles. Dies geschieht durch Auswahl des Profiles im linken Teil des Menüs und Betätigen des Buttons "Set Profile". Die Videoprofile sind nach ihrer Bitrate geordnet. Mit "Get Profile" kann jederzeit das aktuelle Profil vom Server abgefragt werden. In einem weiteren Schritt wird die Bitrate direkt geändert. Eingestellt wird ein relativer Wert in Bezug zur Profil-Standardbitrate. Der Wertebereich liegt zwischen 50 und 200 %. Fehler und Erfolgsmeldungen der getätigten Aktionen werden in einem Logfenster im unteren Bereich des Dialoges angezeigt.

Ist die Bildgröße des Videostreams kleiner als die Größe der Frames der Videoquelle (Kamera, TV-Karte, Datei), so muss der Server eine Unterabtastung der Bilder durchführen. Dies führt ohne Einsatz von Filtern zu Aliasing (Spiegelung im Frequenzbereich) und somit zu einer Verringerung der Bildqualität. Zur Minderung dieses Effektes wurden im Server drei Tiefpaßfilter implementiert. Diese können bei Bedarf aktiviert werden. Das entsprechende Menü wird über “Filter...“ geöffnet:

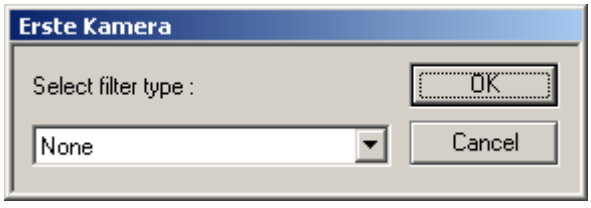


Abb. D.9 : Auswahl eines Tiefpaß-Filters

Der Benutzer hat die Wahl zwischen einem einfachen Summenfilter (Haar-Filter), einem 3-Tap-Filter mit relativ starker Filterwirkung (Dreiecksfilter), sowie einem qualitativ hochwertigen 7-Tap-Filter. Letzteres erfordert aber eine hohe Leistung des Serverprozessors.

Die Filterkoeffizienten der drei Filter sind :

{0.5, 0.5} bzw. {1/3, 1/3, 1/3} bzw. {1/4, 1/4, 1/4, 1/4}

{0.25, 0.5, 0.25}

{-0.045636, -0.028772, 0.295636, 0.557544, 0.295636, -0.028772, -0.045636}

Der erste Filter ist für Unterabtastungen von 2:1 , 3:1 und 4:1 nutzbar, die letzten beiden werden nur bei der am häufigsten verwendeten 2:1 Unterabtastung verwendet.

Für den Spezialfall der 2:1 Bildverkleinerung wurde im Server noch ein anderer Modus implementiert. Während im Normalfall das ganze Bild unterabgetastet und übertragen wird, wird im Zoom-Modus nur ein Bildausschnitt codiert. Dieser Ausschnitt beträgt ein Viertel des Bildes. Der Mittelpunkt des Bereiches, welcher übertragen werden soll, wird durch die Mausposition während des rechten Mausklicks in das Videobild festgelegt. Ein Klick auf den Menüpunkt "Zoom In/Out" aktiviert bzw. deaktiviert dann den Zoom-Modus. Weitere Möglichkeiten des serverseitigen Zooms sind geplant.

Ein Klick auf den Menüpunkt "Statistics..." öffnet eine Anzeige von statistischen Daten. Diese werden mit einer bestimmten Rate aktualisiert (standardmäßig 1 Sekunde). Im unteren Teil findet man die Client-Daten. Unter anderem werden Bitrate und Bildrate vom Client gemessen. Diese müssen nicht genau mit den im Videoprofil vorgegebenen Daten übereinstimmen, sondern richten sich u.a. nach den Vorgaben der Bildquelle und der Genauigkeit des Encoders.

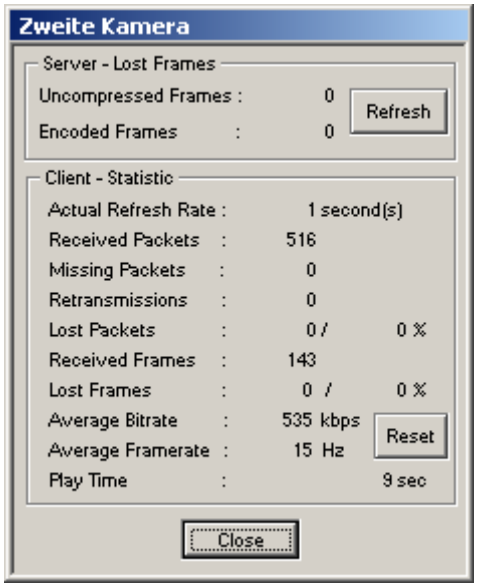


Abb. D.10 : Anzeige von statistischen Daten des Servers und Clients

Im oberen Teil wird die Leistungsfähigkeit des Servers angezeigt. Dazu muss der Button "Refresh" gedrückt werden. Gehen beim Server unkomprimierte Bilder verloren, so ist der Encoder zu langsam. Mit anderen Worten, die Leistung des Serverrechners ist zeitweise oder dauerhaft nicht ausreichend. Ähnliches gilt für die verlorenen codierten Frames, wobei hier noch die Sendegeschwindigkeit (Netzwerkbandbreite) eine große Rolle spielt.
Im unteren Teil werden Statistiken bezogen auf RTP Pakete und Frames angezeigt.

"Missing Packets" gibt die Anzahl an Paketen an, die bei der ersten Übertragung verlorengegangen sind, "Lost Packets" die Anzahl der vermißten Pakete, welche trotz wiederholter Übertragung bis zum endgültigen Timeout nicht eingetroffen sind. "Lost Frames" gibt an, wieviele Frames verworfen werden mußten, da eines oder mehrere Pakete verloren gegangen ist.
Das Menü wird solange angezeigt, bis der "Close"-Button gedrückt oder das zugehörige Videofenster geschlossen wird.
Neben der Möglichkeit das Fenster zu maximieren, kann die Fenstergröße auch stufenlos geändert werden. Der Menüpunkt "Window..." öffnet dazu folgenden Dialog :

Abb. D.11 : Ändern der Fenstergröße

Die Videofenstergröße kann zunächst stufenweise zwischen ¼x und 4x verändert werden. Ist dies zu ungenau, so kann der Benutzer im Nebenfeld auch eine Fenstergröße relativ zur Standardgröße einstellen. Die Angaben sind relativ zur Standardbildgröße. Im unteren Feld kann aus einer Liste die zeitliche Auflösung der Servertimestamps gewählt werden. Dies dient der korrekten Interpretation und Anzeige der übertragenen Serverzeit in der Fensterüberschrift.

Der Menüpunkt " Kill Server" beendet die komplette Serveranwendung per Remote Control (siehe auch Menü "Connect to Server").
Und schließlich kann auch mit "Stop Streaming" die Verbindung beendet werden.

D.4) Zusammenfassung und Ausblick

Die derzeitige Version V1.1 des Programmes TraViS bietet eine benutzerfreundliche menügeführte Bedienoberfläche für den Zugriff auf die Features des MultiVideoClients.
Es können mehrere Videostreams empfangen werden. Die Konfiguration der Videoserver erfolgt manuell, eine interne automatische Anpassung der Bitrate oder der Videoprofile ist optional verfügbar.
Das interne Kontrollmodul ist in der Lage Bitraten und Videoprofile der Streams bei Netzwerkproblemen automatisch zu regeln.

Version : 1.1 Stand 10/2003
Autor : Patrick Voigt

Anhang E – Network Configuration File

Nachfolgend ein Beispiel für eine Datei zur Konfiguration des Kontroll-Modules.
Die Server sind auf zwei lokale Netze verteilt. Die maximale Anzahl an Servern beträgt jeweils 4.

```
***NETWORK CONFIGURATION FILE***
***NETWORK LIST START***
NET# : 1
PRIORITY_INTERVAL : 10
MAX_LOSTFRAMES_SERVER : 3
MAX_LOSTFRAMES_NET : 3
MAX_BITRATE_1 : 6600
MAX_BITRATE_2 : 6500
MAX_BITRATE_3 : 6300
MAX_BITRATE_4 : 5600
NET# : 2
PRIORITY_INTERVAL : 5
MAX_LOSTFRAMES_SERVER : 2
MAX_LOSTFRAMES_NET : 2
MAX_BITRATE_1 : 8000
MAX_BITRATE_2 : 7000
MAX_BITRATE_3 : 6000
MAX_BITRATE_4 : 6000
***NETWORK LIST END***
***SERVER LIST START***
ServerID : 1
NetworkID : 1
Priority : 2
ServerID : 2
NetworkID : 1
Priority : 1
ServerID : 3
NetworkID : 1
Priority : 1
ServerID : 4
NetworkID : 1
Priority : 1
ServerID : 5
NetworkID : 2
Priority : 2
ServerID : 6
NetworkID : 2
Priority : 2
***SERVER LIST END***
```

Anhang F – Videoprofile

Profil	Bildbreite	Bildhöhe	Framerate[Hz]	Bitrate [kbits/s]	IFrame-Periode
0	Benutzerdefiniert				
1	640	480	15	1300	10
2	640	480	5	600	5
3	640	480	15	1000	15
4	640	480	5	500	10
5	320	240	15	500	10
6	320	240	5	250	5
7	320	240	15	400	15
8	320	240	5	200	10
9	352	288	25	800	12
10	352	288	10	400	10
11	176	144	25	200	12
12	176	144	10	100	5
13	704	576	25	2000	12
14	720	576	25	2000	12
15	512	384	5	500	5
16	1024	768	5	2000	5

Danksagung

Die Anfertigung dieser Diplomarbeit wurde mir durch Aufnahme einer Tätigkeit als studentischer Mitarbeiter in der Abteilung Image Processing am Heinrich-Hertz-Institut ermöglicht.
Ich bedanke mich besonders bei meinen beiden Betreuern des Instituts Dr. Aljoscha Smolić und Dipl.-Ing. Karsten Müller für ihre große Unterstützung in den zwei erfolgreichen Jahren. In dieser Zeit habe ich viel wertvolles Wissen im Bereich der Videocodierung, Videoübertragung, 3D-Synthese und allgemein der Softwareentwicklung erworben. Die Arbeit im Projektteam hat mir sehr viel Spaß gemacht. Ich hoffe, sie können die Ergebnisse meiner Studien- und Diplomarbeit für neue Projekte weiterverwenden.

Weiterhin möchte ich den Kollegen Christian Stoffers und Jens Guether danken, welche mir Software für meine Arbeiten zur Verfügung gestellt und meine vielen Fragen dazu bereitwillig beantwortet haben.

Prof. Dr.-Ing. Thomas Sikora und Dipl.-Ing. Sila Ekmekci danke ich für die Bereitstellung des Themas und die fachliche Unterstützung in der Zeit meines Hauptstudiums. Bei Frau Boldin bedanke ich mich für die kompetente Beratung und Hilfe in Fragen der Organisation.

Ein großes Dankeschön geht an meine Familie, insbesondere meine Eltern. Sie haben mir die Aufnahme des Studiums der Elektrotechnik ermöglicht und mich in dieser Zeit tatkräftig unterstützt.

Ich wünsche Karsten Müller, Sila Ekmekci und meiner Schwester viel Erfolg bei der Anfertigung ihrer Doktorarbeiten.

Patrick Voigt

Eidesstattliche Erklärung

Hiermit versichere ich an Eides Statt, dass ich die vorliegende Diplomarbeit selbstständig und eigenhändig angefertigt habe.

Berlin, den 15. Oktober 2003

_____

Patrick Voigt

Entwicklung eines steuerbaren Videoservers zur Übertragung von MPEG-4 Echtzeitvideoströmen

Patrick Voigt

Studienarbeit
an der Technischen Universität Berlin
Institut für Telekommunikationssysteme,
Fachgebiet Nachrichtenübertragung

bei Prof. Dr.-Ing. Thomas Sikora

Betreuer :

Dipl.-Ing. Sila Ekmekci **TU Berlin**
Dipl.-Ing. Karsten Müller **Heinrich-Hertz-Institut**

Berlin, September 2002

Inhaltsverzeichnis

1 Einleitung

1.1 Digitale Videoüberwachung

Aufgrund des weltweit steigenden Verkehrsaufkommens wird in vielen Ländern ständig nach neuen Lösungen zur Kontrolle des Verkehrsstromes gesucht. Hauptproblempunkte sind dabei der Verkehr in Metropolen sowie auf starkfrequentierten Hauptverkehrsadern. Die bisher am häufigsten verwendeten Systeme basieren auf Induktionszählschleifen im Straßenboden, welche aber nicht den Gesamtzustand auf einer Straße erfassen können. Der Einsatz von Videoüberwachung als Beitrag zur Lösung der Probleme ist nicht neu, ausgereifte Systeme sind jedoch nur schwer zu realisieren. In einigen japanischen Großstädten z.B. existieren Systeme zur allgemeinen Beobachtung wichtiger Verkehrsknotenpunkte in einem Leitzentrum. Das Hauptproblem der großflächigen Videoüberwachung sind die enormen Datenmengen, welche übertragen, verarbeitet und gegebenenfalls angezeigt werden müssen. Eine Verbesserung kann nun durch Einsatz von leistungsfähigen digitalen Videocodierstandards wie H.26x oder MPEG-4 in Zusammenhang mit der Verwendung von breitbandigen Netzen erreicht werden.
Ein wichtiger Punkt dabei ist die Frage der zentralen oder dezentralen Verarbeitung. So kann eine Auswertung aller Informationen entweder vor Ort, d.h. an der Straße, oder in einem Leitzentrum erfolgen. Da aber zur Codierung der aufgenommenen Bilder bereits Hardware vor Ort benötigt wird, könnte diese günstigerweise auch für die Vorverarbeitung des Bildmaterials verwendet werden. Die Leitstelle kann sich dann auf die Auswertung der gewonnenen Informationen, sowie auf Steuerungsaufgaben konzentrieren. Durch diese Trennung der Aufgabengebiete wird desweiteren auch die Störanfälligkeit des Gesamtsystemes verringert.
Ein weiterer Vorteil dieses Systemes ist die parallele Übertragung von Verkehrsdaten und Videodaten, wobei durch geeignete Priorisierung beider Datenströme die vorhandene Netzkapazität optimal ausgenutzt werden kann.

In dieser Studienarbeit wird nun eine Anwendung präsentiert, welche für den Einsatz der Videoüberwachung geeignet ist. Die zu implementierende Anwendung soll die Aufnahme, Codierung und Versendung eines Videostreams über ein IP-Netz ermöglichen. Da die Software auch auf schwer zugänglicher Hardware eingesetzt werden soll, muß die Steuerung im Wesentlichen über einen Rückkanal auf dem Netz erfolgen. Zu Steuerungsaufgaben gehören z.B. das Starten oder Beenden der Anwendung, von einzelnen Modulen oder Änderungen am Videostream selbst (Auflösung, Framerate).

Im vorliegenden Szenario, in dem ein Verkehrsknotenpunkt überwacht werden soll, kommen bis zu vier Kamerarechner zum Einsatz. An diesen sind über Firewire-Ports CCD-Kameras angeschlossen, welche für den Tageslicht- und Nachteinsatz (Infrarot) geeignet sind. Die Kamerarechner übernehmen dabei die Funktion von Servern, welche auf Anforderung des Clients, dem zentralen Kreuzungsrechner, Videodaten in Form von MPEG4-Elementardatenströmen versenden. Anwendungen auf dem Zentralrechner können diese Daten entgegen nehmen und sie weiter verwerten (Anzeigen, Speichern etc.).

Die Übertragung der Daten kann z.B. über Ethernet oder Wireless LAN erfolgen. Vor- und Nachteile beider Technologien werden in Kapitel 3 angesprochen.

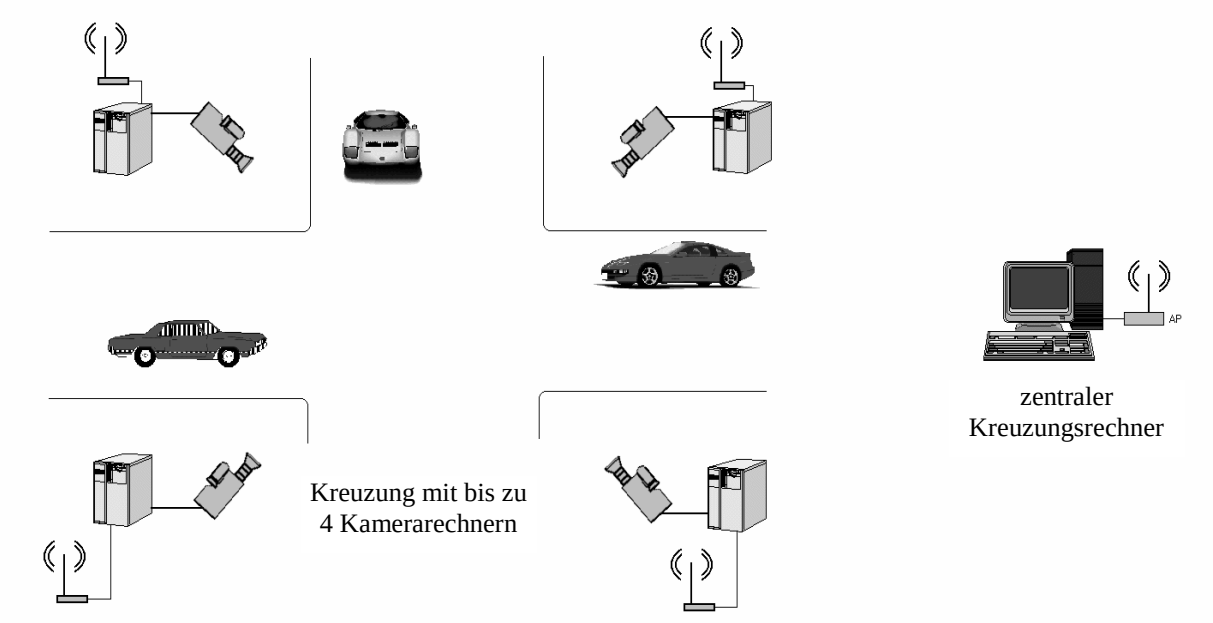


Abb. 1.1 : Gesamtsystem für Verkehrsdatenerfassung und Videoüberwachung (WLAN)

Die in der Studienarbeit verwendeten Begriffe für die zu implementierende Software lauten “Anwendung“ (bzw. “Server1394“) und “Videoserver“. Der Videoserver ist dabei ein Modul, welches die Codierung und Versendung leistet. Die Anwendung selbst bedient sich dieses Modules und übernimmt dazu die Bildakquisition über Firewire und das Benutzerinterface in Form eines Menüs. Die Trennung dieser Funktionen ermöglicht einen universellen Einsatz der Software für unterschiedliche Anwendungsfälle.

1.2 Überblick

Die Studienarbeit ist in 6 Kapitel gegliedert. Kapitel 1 enthält ein Vorwort zum Thema und einen Überblick. In Kapitel 2 wird eine Einleitung zur Struktur der implementierten Anwendung gegeben. Kapitel 3 geht dann zunächst auf die für das Projekt relevanten Standards und Technologien ein. Kapitel 4 beschreibt detailierter die Funktionsweise der einzelnen Module der Anwendung sowie die verwendeten Klassen, ohne jedoch auf jede einzelne Funktion etc. einzugehen. Eine Auswertung der Software z.B. in Bezug auf Bedienbarkeit und Verwendbarkeit erfolgt in Kapitel 5. Kapitel 6 enthält schließlich eine Zusammenfassung. Die für den Benutzer wichtigen Interfaceklassen, sowie die verwendeten Videoprofile und Steuerbefehle sind im Anhang beschrieben. Außerdem sind im Anhang F die Ergebnisse der Untersuchung zweier Testsequenzen dargestellt.
Im Rahmen der Ausführungen wird der Begriff “Frame“ gleichbedeutend mit einem Einzelbild eines Videostreams verwendet. Ferner sind alle Begriffe, die dem Sourcecode entnommen wurden z.B. Klassennamen oder Variablen *kursiv* gedruckt.

2 Softwarekonzept des Videoservers

2.1 Zielplattform und Entwicklungsumgebung

Zielplattform für den Videoserver ist Microsoft Windows 2000, ein 32-Bit-Betriebssystem mit NT-Kernel. Es unterstützt Multitasking und Multithreading. Die verwendete Programmiersprache ist Visual C++ in der Version 6.0, welche in die Entwicklungsumgebung Microsoft Developer Studio eingebettet ist. Die Schnittstelle zum Betriebssystem wird von Microsoft in Form der Win32 API bereitgestellt, welche aber nur zum Teil direkt genutzt wurde (z.B. für Dateiausgabe). Wesentliche Teile des Videoservermoduls wurden unter Zuhilfenahme der am HHI entwickelten Programmbibliothek MSysLib geschrieben, welche eine Sammlung von Klassen für die API zum Betriebssystem enthält.
Für die Testanwendungen wurde zudem von der Microsoft Foundation Class MFC Gebrauch gemacht. Aus dieser Bibliothek wurden speziell Fenster- und Dialogklassen für die Benutzerschnittstelle verwendet.

2.2 Modulares System durch Multithreadprogrammierung

Der Videoserver ist modular aufgebaut, d.h. die Aufgabengebiete sind in abgeschlossenen Einheiten zusammengefaßt. Bis auf die Interfaceklasse und dem Hauptmodul sind alle Module sogenannte Worker-Threads. Einmal gestartet führen diese Threads Berechnungen aus, bis sie von außen oder innen beendet werden.
Multithreadprogrammierung hat den Vorteil, daß mehrere Aufgaben “gleichzeitig“ bearbeitet werden können. Die notwendige Prozessorzeit wird dabei vom Betriebssystem (bzw. dem sogenannten Scheduler) aufgeteilt. So kann zum Beispiel eine Anwendung , die aus einem Workerthread und einem Interfacethread besteht, eine Berechnung durchführen und parallel dazu auf Benutzereingaben reagieren. Das Verhalten von Threads läßt sich zudem durch Zuweisung einer Prioritätsstufe steuern.
Der Nachteil einer Multithreadanwendung ist die teilweise aufwendige Synchronisation der parallel laufenden Threads, welche natürlich mit der Anzahl der Threads noch zunimmt. So kann in unserem Beispiel der Encoder erst dann mit der Codierung beginnen, wenn ihm mitgeteilt wurde, das ein neues Frame eingetroffen ist.

In der vorliegenden Anwendung erfolgt die intermodulare Kommunikation im Wesentlichen durch Aufrufen einer Memberfunktion der Zielthreadklasse. Dazu muß vorallem sichergestellt werden, das der Zielthread zum Zeitpunkt des Aufrufens auch existiert. Dieses erfordert besondere Sorgfalt beim Hoch- und Runterfahren einzelner Threads. Außerdem müssen prinzipiell die Kommunikationswege durchdacht sein, um ein gegenseitiges Aufrufen (Deadlock) zweier oder mehrerer Funktionen zu vermeiden. In unserem Fall wird dies jedoch durch den Umstand erleichtert, das die interne Übertragung der Frames nur in eine Richtung erfolgt. Mehr dazu im Kapitel 4.

Eine andere Variante wäre der Einsatz von Windows-Nachrichten. Diese werden entweder über die zentrale Nachrichtenwarteschlange von Windows an einen Fensterhandle oder direkt an die Fensterfunktion eines Fenster gesendet und sind überwachbar. Allerdings muß

dazu in jedem Thread ersteinmal ein Fenster mit einer zugehörigen Fensterfunktion erstellt werden. Außerdem können Windows-Nachrichten auch von anderen (Sabotage-) Programmen an die Anwendung gesendet werden. Aus diesen Gründen wurde auf die Verwendung von Nachrichten verzichtet, obwohl dies die für Windowsprogramme übliche Form der Kommunikation ist.

Die dritte Möglichkeit zur Synchronisation ist der Einsatz von Flagvariablen. Hier ergibt sich aber wiederum ein anderer Nachteil von Multithreadprogrammierung. Da die Threads abwechselnd vom Prozessor bearbeitet werden, kann der augenblickliche Zustand von Variablen bzw. Puffern nicht immer korrekt definiert werden. Abhilfe schafft hier die Verwendung von Synchronisierungsobjekten der Win32-API. Sogenannte Critical Sections schützen z.B. einen Teil eines Codes oder eine Resource des Threads vor dem Zugriff durch andere Threads. Dieser ist für die Dauer der Ausführung gesperrt. Somit ergibt sich die Möglichkeit, den Zugriff mehrerer Threads auf eine gemeinsame Resource (Flagvariable, Puffer etc.) zu regeln.

In der Übertragungsstrecke Grabber->Encoder->RTPServer (siehe Abb. 2.1 in Kapitel 2.3) besitzt jedes Modul eigene interne Puffer. Die Übergabe von Daten erfolgt durch Aufruf einer Memberfunktion der nächstfolgenden Threadklasse, d.h. der Grabber ruft eine Funktion des Encoders auf und der Encoder eine Funktion des RTP-Servers. Innerhalb dieser Funktion werden die als Parameter übergebenen Daten (Pointer auf Datenpufferklasse) in den klasseninternen Puffer kopiert, unter Einsatz eines Critical Sections-Objektes eine Flagvariable gesetzt und die Funktion beendet. Somit ist auch sichergestellt, das mit Hilfe des Rückgabewertes der Funktion der Quellthread über eventuell bei der Übergabe aufgetretene Fehler informiert wird. Eine Schwierigkeit bei dieser Verfahrensweise ist die Übergabe der Pointer auf die benachbarten Threadklassen, damit ein Thread Zugriff auf die benötigte Übergabefunktion erlangen kann. Da alle drei Threads beliebig hoch und runter gefahren werden können, müssen Funktionen zum An- und Abmelden eines Modules bei seinen Nachbarn implementiert werden. Das Starten und Stoppen von einzelnen Modulen ist jedoch eher zu Testzwecken und Fehlersuche gedacht. In der praktischen Anwendung macht nur das vollständige Starten oder Anhalten aller Module Sinn, da der Videoserver ja bereits bei einem heruntergefahrenem Modul nicht mehr funktionstüchtig ist.

2.3 Gesamtsystem

Die Abbildung 2.1 zeigt das Gesamtkonzept des Videoservers und der Anwendung :

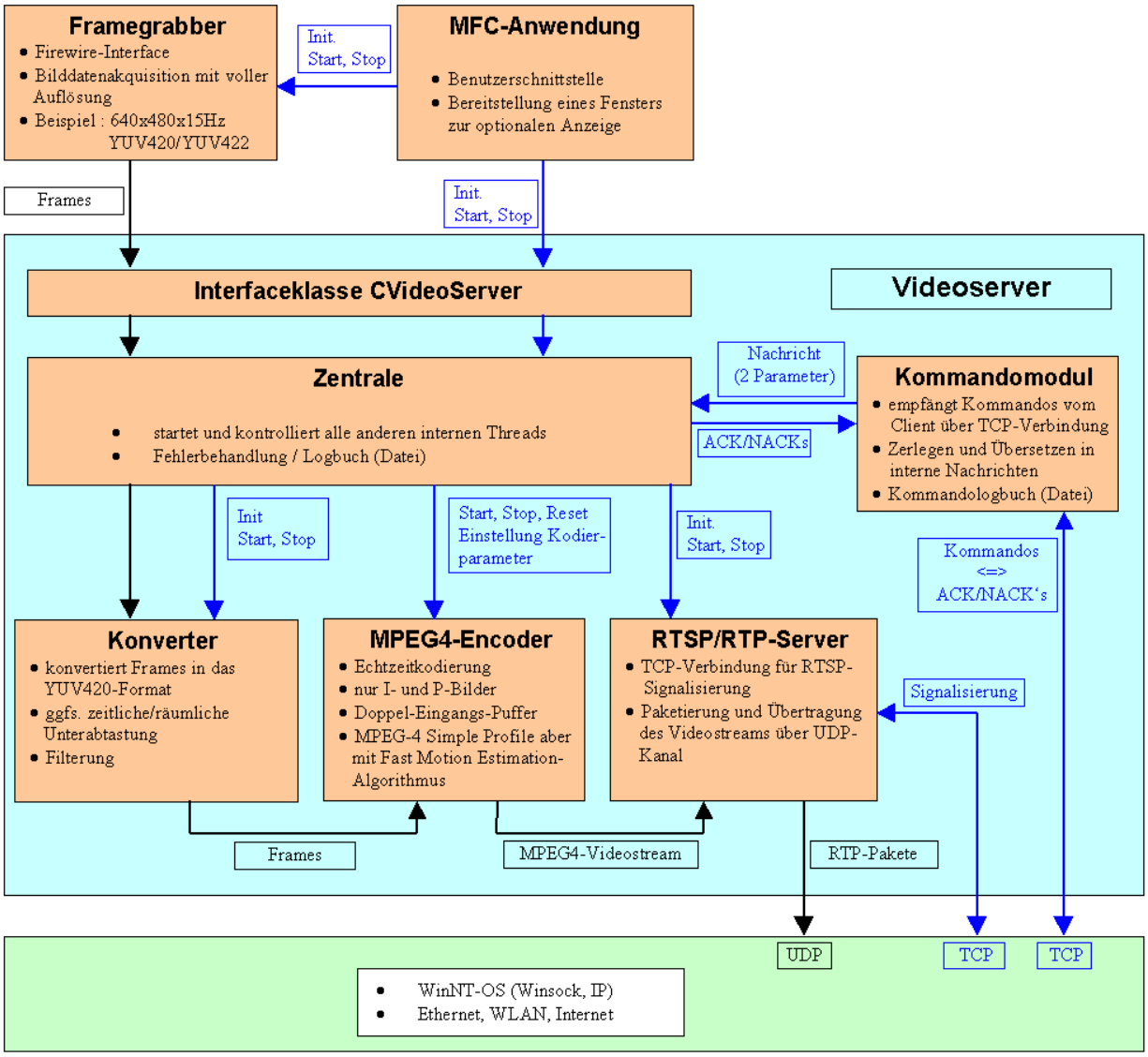


Abb. 2.1 : Softwarekonzept des Videoservers

Wie in der Abbildung zu sehen ist, besteht die Anwendung aus folgenden Modulen :

Testanwendung ("Server1394") :

- MFC-Anwendungsgerüst
- Framegrabber

Modul und Bibliothek “Videoserver“:

- Hauptmodul (Zentrale)
- Kommandomodul
- Konverter
- MPEG4-Encoder
- RTSP/RTP-Server

Dieses modulare Konzept bietet den Vorteil, das Änderungen an einem Modul wenig oder gar keinen Programmieraufwand für die anderen Module bedeutet. So kann z.B. der Framegrabber einfach an eine USB-Kamera oder TV-Karte angepaßt werden. Den nachfolgenden Konverter betrifft dies nur insofern, als das er das neue Bildformat kennen muß. Eine anderes Szenario wäre der Einsatz einer neuen Version des MPEG4-Encoders. Diese könnte ohne große Mühe eingebaut werden.
Durch Auslagern der Bilddatenverarbeitung in eine Bibliothek ist außerdem die Wiederverwendbarkeit für andere Projekte/Themengebiete gewährleistet.

Das Interface ist bewußt einfach gehalten. Es enthält Funktionalitäten bzgl. Initialisierung, Starten und Stoppen des Videoservers sowie zur Übergabe von Frames. Optional sind einige Möglichkeiten zur direkten, serverseitge Kontrolle verfügbar.
Die Kontrolle des Videoservers erfolgt aber hauptsächlich durch Kommandos über einen Rückkanal. Dazu ist eine Reihe von Kommandos samt Parameter spezifiziert worden, welche in Anhang D aufgelistet sind.

3 Technologien

Bevor auf die einzelnen Module eingegangen werden kann, sind zunächst eine Reihe von Basistechnologien und Standards kurz vorzustellen. Dies soll dem leichteren Verständnis für die folgenden Beschreibungen der Software und den später zu diskutierenden Beschränkungen und Einsatzmöglichkeiten des Servers dienen. Die Reihenfolge der Themen dieses Kapitels orientiert sich am Schichtenmodell des ISO-OSI-Referenzmodells. Es wird mit den unteren Schichten begonnen (Physical Layer...) und mit der obersten Schicht (Application-Layer) geendet. Informationen zum Referenzmodell findet man in zahlreicher Literatur ([1]).

3.1 Das Kamerainterface : Firewire bzw. der IEEE1394-Standard

Unter dem Namen “Firewire“ entwickelte in den 90er Jahren Apple ein neues digitales I/O System mit serieller Busarchitektur für Peer-to-Peer Datenübertragung. Die IEEE nahm dieses System unter IEEE1394 im Jahr 1995 als Industriestandard auf. Seitdem wurde um diesen Bus herum eine Reihe von Hard- und Software entwickelt, vorwiegend im Multimediabereich und für die Datenspeicherung.
Gegenüber dem im Low-Cost Bereich angesiedelten USB (V1.1) bietet Firewire eine höhere garantierte Datenübertragungsrate von bis zu 400 MBits/s, wobei bereits eine neue Version mit 1,2 GBits/s in Entwicklung ist. Mit diesen Raten läßt sich auch die Übertragung von wenig oder unkomprimierten hochauflösenden Videodaten realisieren. Aus diesem Grunde bietet jeder derzeit produzierte Digital-Camcorder ein entsprechendes Interface. Darüber hinaus gibt es eine Reihe von digitalen Videokameras, welche ihre Daten vornehmlich im YUV-Format in Echtzeit an angeschlossene Computer übertragen. Der Vorteil solcher Kameras liegt im wenig verfälschten Videomaterial, welches frei von Kodierartefakten ist.
Firewire-Peripheriegeräte werden über ein 4-Pin oder 6-Pin-Kabel angeschlossen. 4 Pins dienen dabei für die Hin- und Rückleitung, die weiteren 2 Pins stellen eine Stromversorgung für Geräte ohne eigenen Stromanschluß zur Verfügung.
Ein großer Vorteil von Firewire ist zudem, daß Geräte bei aktivem Bus angeschlossen oder entfernt werden können.

3.2 Local Area Networks : Ethernet vs. Wireless-LAN

Die beiden für das Projekt interessantesten LAN-Technologien sind Inhalt des IEEE 802-Standards. Das IEEE 802 Komitee befaßt sich unter anderem mit Zugriffsmethoden und Signaling für Ethernet (Standard 802.3) und Wireless LAN (Standard 802.11). Bezogen auf das ISO-OSI-Referenzmodell betreffen diese Standards den Physical Layer und den Link Layer, welcher zudem funktionell in MAC-Layer (Medium Access Control) und LLC (Logical Link Control) unterteilt wird.
Der Standard für Ethernet umfaßt dabei unter anderem die gebräuchlichste Zugriffsmethode CSMA/CD (Carrier Sense Multiple Access/Collision Detection), sowie Spezifikationen für Ethernet (1/10 Mbps), Fast-Ethernet (100 Mbps) und Gigabit Ethernet. Unter CSMA/CD

versteht man eine spezielle Methode für den geregelten Zugriff mehrerer Hosts auf ein gemeinsames Übertragungsmedium. Dabei wird der Kanal vor dem Versenden von Datenpaketen abgehört (Carrier Sense). Ist er gerade belegt, so zieht sich der Host für eine stochastisch bestimmte Zeit zurück und versucht es dann erneut. Auch das gleichzeitige Versenden zweier oder mehrerer Pakete wird anhand der Signalstärke auf dem Kanal erkannt (Collision Detection). Mit steigender Kanalbelastung steigt auch die Wahrscheinlichkeit für Kollisionen.
Die gebräuchlichste Netztopologie für Ethernet ist die Stern-Topologie. Die Hosts werden an einen zentralen Hub (shared LAN) oder einen Switch (switched LAN) angeschlossen. Ein Switch sorgt für eine höhere Netzkapazität, da kommunizierende Host-Paare voneinander entkoppelt sind. Da im vorliegenden Szenario mehrere Server Daten an einen Client senden, ist es aber in diesem Fall egal, ob ein Switch oder Hub verwendet wird. Vorteil der Sterntopologie ist die einfache Erweiterbarkeit des Netzes. Sofern ein Gebäude ausreichend verkabelt ist, sind nur Konfigurationsmaßnahmen am Switch und an den betroffenden Computern notwendig.

Im Vergleich zu Ethernet bietet Wireless-LAN Möglichkeiten zur noch einfacheren, schnelleren Vernetzung von Computern. So können zum Beispiel mehrere PC's für Konferenzzwecke zu einem Ad-Hoc-Netzwerk zusammen geschlossen werden. Aber auch die Anbindung von mobilen Hosts an ein bestehendes verkabeltes LAN kann über ein Kontroll-Modul, dem sogenannten Access Point, realisiert werden. Der Zugriff auf das Übertragungsmedium wird durch den Algorithmus DFWMAC (Distributed Foundation Wireless MAC) geregelt, welcher die verteilte, dezentrale Zugriffsmethode CSMA beinhaltet, aber auch die Möglichkeit zur zentralen Steuerung bietet. Eine Kollisionserkennung wie bei Ethernet ist aber nicht vorgesehen, da dies nur schwer zu realisieren wäre.
Beide WLAN-Topologien sind für die Verkehrsüberwachung hochinteressant, da eine Verkabelung von Kreuzungen umständlich und teuer ist. Nachteile sind allerdings u.A. die geringere Bandbreite sowie die Abhängigkeit der Bandbreite von Umgebungseinflüssen und Entfernungen. Zudem besteht die Gefahr der Beeinflussung des WLANs durch fremde WLAN-Zellen und die Möglichkeit des Abhörens oder gar der Manipulation der übertragenen Daten.
Die Bandbreite von WLANs deckt den Bereich von ca. 1 Mbps für Infrarotübertragung bis hin zu 20 Mbps für Spreizsprektrum- sowie Mikrowellen-LANs ab. Letztere arbeiten im Frequenzband 902-928 MHz und im Gigahertz-Bereich. Die zur Zeit erhältlichen WLANs haben eine Bandbreite von theoretisch bis zu 11 Mbps bei einer Frequenz von ca. 2,4 GHz.
In Abb. 3.1 sind die beiden für das Projekt interessanten LAN-Topologien zusammen abgebildet. Der Anschluss eines WLANs an ein Backbone Wired LAN ist besonders dann erforderlich, wenn die vom Client empfangenen Videodaten noch weiter versendet werden sollen, z.B. über das Internet an eine Leitstelle.
Zum Anschluß eines PC's an ein WLAN wird i.A. eine PCMCIA- oder eine PCI-Karte verwendet. Vorteilhafter in Bezug auf die Reichweite sind aber externe USB-Module, welche auch eine Antenne (symbolisch in Abb. 3.1) integriert haben.

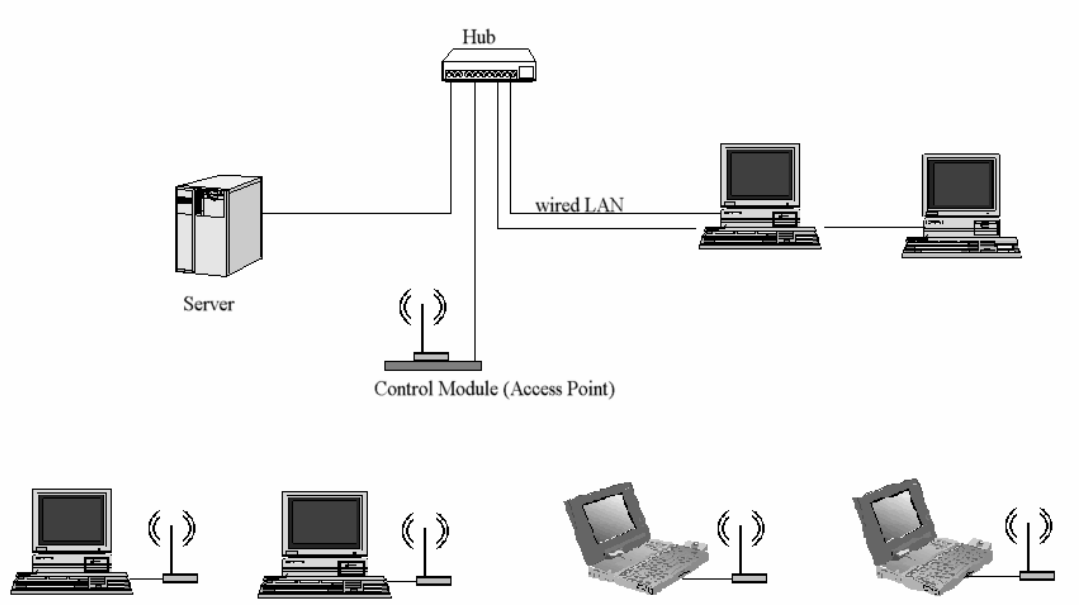


Abb. 3.1 : Kombination aus WLAN und Ethernet (Sternverkabelung)

3.3 Übertragungsprotokolle : TCP/UDP und IP

Das Internetprotokoll, entstanden Anfang der 80er Jahre im Rahmen von militärischen Forschungsprojekten über neue sichere Kommunikationsnetze (ARPANET), hat sich zum führenden Kommunikationsprotokoll zur weltweiten Verknüpfung lokaler und regionaler Netze entwickelt. Literatur zu diesem Thema, sowie zu den auf IP aufsetzenden Protokollen der Transportschicht TCP (Transmission Control Protocol) und UDP (User Datagram Protocol) findet sich reichlich. Stellvertretend sei hier auf [1] sowie die RFC's 791 ([2]), 793 ([3]) und 768 (UDP) verwiesen.
IP ist ein Protokoll der Vermittlungsschicht und dient zur Übertragung von Datenpaketen, sog. Datagrams. Diese bestehen aus einem Header und einem Segment aus dem eigentlichen Datenstrom (Payload). Der Aufbau des Headers gibt bereits Auskunft über die verfügbaren Dienste des Protokolls. So befinden sich in ihm u.a. die Sender- und Empfängeradresse, Time to Live-Information, Type of Service-Information oder eine Checksumme für den Header.
Die beiden genannten Adressen identifizieren jeweils genau einen Host im weltweiten Netz, sie sind somit einmalig (und müssen es auch sein). Da die bisher 32-Bit langen Adressen aufgrund des enormen Wachstums des Internets nicht mehr ausreichen, ist eine erweiterte Adresse Bestandteil neuer IP-Spezifikationen (IPv6-Header). Die TtL-Information verhindert ein unendlich langes Herumirren eines Paketes im Netz.
Das Internet Protokoll selbst bietet keine Funktionalität bzgl. Übertragungssicherheit, Flußkontrolle oder Paketnummerierung. Dies bleibt übergeordneten Protokollen vorbehalten.

Das Transmission Control Protocol ist eng mit dem Internet Protokoll verbunden. Es setzt auf IP auf und bietet eine kontrollierte Übertragung von Datenpaketen. So enthält der Header eine Sequenznummer zum Wiederordnen der Pakete auf Empfängerseite oder eine Checksumme zum Erkennen defekter Pakete. Das Protokoll arbeitet zudem mit

Bestätigungen. Jedes Paket oder auch mehrere Pakete zusammen können dem Sender bestätigt werden. Außerdem ist die Anzahl an gleichzeitig versendbaren Paketen beschränkt, wobei diese Beschränkung wiederum variabel ist. Diese sogenannte “sliding window“-Methode dient der Stau- bzw. Flußkontrolle.

Während TCP zur Realisierung einer weitgehend sicheren Übertragung von zeitunkritischen Daten genutzt wird, bietet UDP nicht die genannten Funktionalitäten. Der Header eines UDP-Datagrams besitzt zwar eine Checksumme aus der mit hoher Wahrscheinlichkeit Fehler erkannt werden können, eine Rückmeldemöglichkeit im Fehlerfall an den Sender besteht jedoch nicht. Der Vorteil liegt aber in der schnelleren Übertragung von Daten, von denen z.B. Echtzeit-Fähigkeiten erwartet werden. Die Handhabung von defekten Paketen oder Paketverlusten bleibt der Anwendungsschicht überlassen.

Da IP zwar die Adressierung eines Host bietet, ein Host jedoch mehrere Übertragungen vornehmen kann, wird für eine TCP/UDP-Verbindung ein Portpaar bereitgestellt. Dieses wird bei der Verbindungsaufnahme zugewiesen und ist bis zum Abbau der Verbindung gültig. Die Portnummern (Sender- und Empfängerport) sind auch Bestandteil der TCP- / UDP-Header. Die Netzwerkschnittstelle bestehend aus IP-Adresse und Portnummer wird als Socket bezeichnet.

3.4 Protokolle für die Videoübertragung : RTSP/RTP

In die Anwendung ist ein am HHI entwickelter RTSP/RTP-Server integriert. Dieser übernimmt die Packetierung und Versendung des Videostreams. Zuerst muß der Client Verbindung mit dem Server aufnehmen und ihm mitteilen, was er von ihm erwartet. Dies ist in unserem Fall die Übertragung eines MPEG4-Livestreams und zwar ausschließlich eines Videotracks.

Nachfolgend sind einige Informationen zu den verwendeten Protokollen aufgeführt. Näheres kann z.B. in [4] und [5] nachgelesen werden.
Das Real-Time Transport Protocol (RTP) bietet die Übertragung von Echtzeit-Daten wie z.B. Audio oder Video an. Es setzt dabei zumeist auf UDP/IP auf. Der zweite Bestandteil des Protokolls ist das RTP Control Protocol (RTCP), welches i. A. auf einem zweiten Kanal (typischerweise TCP) parallel arbeitet. Mittels regelmäßig ausgesendeter Kontrollpakete können so Informationen über Qualität der Übertragung oder über die Teilnehmer der Sitzung verbreitet werden.
Die von RTP und RTCP benutzten Ports liegen nebeneinander, ersterer sollte eine gerade Nummer haben.
RTP-Pakete bestehen wie auch in anderen Protokollen üblich aus Header und Payload. Der Header beinhaltet wichtige Informationen, wie source ID, timestamp, sequence number oder payload type. Die Sequenznummer dient z.B. zum Ordnen der Pakete auf der Clientseite, der Typ der Ladung gibt an, ob es sich um Audio, Video etc. handelt. Die genannten Informationen werden u.a. benötigt, um beim Empfänger zwei oder mehrere Streams zu

synchronisieren. Die Längenbeschränkung für RTP-Pakete resultiert aus den Beschränkungen der unterliegenden Protokolle, der RTP-Header selbst besitzt kein Feld für eine Längeninformation.
RTP bietet sowohl unicast- als auch multicast-Funktionalität, sofern die unteren Protokolle dies ebenfalls unterstützen, und ist somit z.B. für Audio- oder Videokonferenzen geeignet.

Auf der Anwendungsschicht arbeitet das Real Time Streaming Protocol (RTSP). Dieses Protokoll dient zur Instanzierung und Kontrolle der eigentlichen Datenübertragung, welche dann mittels RTP abgewickelt werden kann. Es arbeitet mit dem Austausch von Textnachrichten, bestehend aus US-ASCII-Zeichen. Diese Nachrichten sind z.B. Anforderungen an den Server oder auch Client, welche an erster Stelle eine Methodenbezeichnung enthalten. Sie dienen z.B. zur Medienbescheibung (“DESCRIBE”-Methode) oder zur Ansteuerung des Servers mit den für Videogeräte typischen Kommandos Play, Pause oder Record (“PLAY”, “PAUSE” etc. -Methode), wobei in geforderte, empfohlene und optionale Methoden unterschieden wird. Das bedeutet, daß nicht jeder implementierte RTSP-Server alle im Standard aufgeführten Methoden unterstützt.
Der Vorgang der Initialisierung, der Übertragung und des Herunterfahrens wird in der RTSP-Terminologie als “Session” bezeichnet.
Obwohl i.A. für die Nachrichtenübertragung eine ständige TCP-Verbindung geschaffen wird, welche für die Dauer einer Session bestehen bleibt, kann auch im verbindungslosen Modus mittels UDP gesendet werden. In diesem Fall ist als Senderichtung allerdings nur Client->Server möglich, während im erstgenannten Fall auch der Server Nachrichten senden kann (z.B. “ANNOUNCE”-Methode). Der Default-Port für eine Verbindungsaufnahme ist der Port 554.
Weitere Details über den integrierten RTSP/RTP-Server sind in Kapitel 4.6 beschrieben.

3.5 Videocodierung : Der MPEG4-Standard

Der MPEG-4 Standard als Weiterentwicklung der Standards MPEG-1 und MPEG-2 wurde von der Motion Picture Experts Group 1999 in der ersten und 2000 in der zweiten Version verabschiedet, unterliegt aber noch ständigen Erweiterungen.
Im Gegensatz zu den Vorgängern beschäftigt er sich nicht nur mit der Verbesserung von Videocodierverfahren, sondern beinhaltet auch Möglichkeiten der Szenenkomposition oder der Formcodierung.
Eine Szene besteht dabei aus Audio-Visuellen Objekten, deren Anordnung und Zusammenwirken im BIFS (Binary Format for Scenes) deklariert ist. Anhand dieser Informationen wird die Szene empfängerseitig durch den Kompositor zusammengefügt.
Für die zu implementierende Anwendung ist besonders der Kernpunkt der Codierung von Videoobjekten interessant. Die Techniken, welche MPEG-4 hier einsetzt, sind in sogenannten Profiles spezifiziert und basieren auf den Techniken der Vorgängerstandards. Prinzipiell arbeitet MPEG-4 wie seine Vorgänger und die H.26x-Standards mit Bewegungsschätzung, anschließender Kompensation und Transformationscodierung (DCT) der entstehenden Differenzbilder. Die DCT-Koeffizienten werden dann mittels VLC codiert.

Für tiefergehende Informationen sei auf einschlägige Literatur oder die Veröffentlichungen der WG 11 der Moving Picture Experts Group ([7], [8]) verwiesen.
Der benutzte Encoder ist konform zum MPEG-4 Advanced Simple Profile (ISO/IEC 14496-2), unterstützt allerdings nicht alle darin enthaltenen Features.

Von den vielen Funktionalitäten des Advanced Simple Profiles werden im Folgenden einige für das Projekt wichtige genannt. Außerdem wird auf die Beschränkungen des verwendeten Encoders hingewiesen :

1.) räumliche und zeitliche Skalierbarkeit
    Dies bietet mehr Freiheiten in Bezug auf Bildauflösung und Bildrate. Der verwendete MPEG4-Encoder arbeitet allerdings nur mit Breiten und Höhen, welche Vielfache der Makroblockgröße (16) sind. Die Bildrate ist zudem auf ganze Zahlen beschränkt.
2.) Halbpixel-Bewegungskompensation (keine ¼-Pel-Kompensation)
    Dies ermöglicht eine genauere Schätzung der Makroblockverschiebungen als bei pixelgenauer Kompensation, bedeutet aber auch einen höheren Rechenaufwand (Interpolation). Letztendlich führt dies zu einem Differenzbild, dessen DCT-Koeffizienten zu einem größeren Teil nahe Null liegen und somit nicht übertragen werden müssen.
3.) flexible Verwendung von I- und P- Bildern
    B-Bilder werden aufgrund des höheren Aufwandes nicht berechnet, da die Anwendung auch auf langsamer Hardware echtzeitfähig sein soll und zudem möglicherweise wenig Speicherplatz für Bildspeicher zur Verfügung steht.
    Die Rate für I-Bilder ist frei wählbar. Eine kleine Intraperiode bedeutet z.B. bei Übertragungsfehlern ein schnelleres Zurücksetzen der gestörten Bilder, kostet aber auch mehr Bitrate bzw. senkt den PSNR-Wert bei fehlerfreier Übertragung.
    Theoretisch ist auch eine reine Intraframecodierung machbar.
4.) Freie Skalierung der Bitrate
    Die zur Verfügung stehende Bitrate wird nach einem Optimierungsalgorithmus auf die I- und P-Frames aufgeteilt.

Um die Echtzeitfähigkeit des Encoders auch auf langsamer Hardware zu ermöglichen, arbeitet der Encoder zudem mit einem am HHI entwickelten schnellen Algorithmus für die Bewegungsschätzung, da hier ein großer Teil der Rechenzeit benötigt wird.
Der FME-Algorithmus (Fast Motion Estimation) beinhaltet drei Stufen, die blockrekursive, die pixelrekursive Stufe sowie eine Stufe zur Verfeinerung des Ergebnisses auf Halbpixelgenauigkeit. Bei der blockrekursiven Stufe werden bis zu drei Bewegungsvektoren herangezogen und die resultierenden Differenzbilder miteinander verglichen. Genauer gesagt betrachtet man die Bewegungsvektoren des oberen und des linken/rechten Makroblockes und zieht außerdem den Vektor für diesen Block aus dem letzten Bild hinzu. Der beste Vektor ergibt dann den Ausgangspunkt für die zweite Stufe, in der nach der Methode des optischen Flusses mit Pixelgenauigkeit nach einem noch besseren Bewegungsvektor gesucht wird.

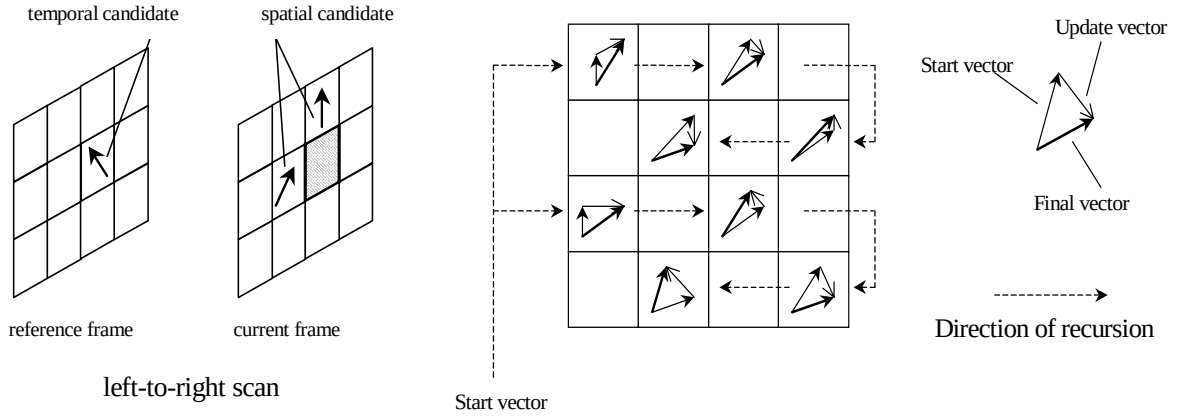


Abb. 3.2 : a) blockrekursive Stufe b) pixelrekursive Stufe des FME-Algorithmus (Quelle [9])

In der letzten Stufe werden dann noch die acht benachbarten Halbpixel betrachtet.
In einer zweiten Version wurden die ersten beiden Stufen direkt auf hochgetastete Bilder angewendet, um die Halbpixelgenauigkeit direkt zu erreichen.
Für genauere Informationen zur Funktionsweise des integrierten MPEG4-Encoders sei auf die Beschreibung bzw. die Dokumentation der Autoren verwiesen ([9]) .

4 Implementierung der Library und der Testanwendung

Im folgenden Kapitel werden die Module der Bibliothek “VideoServer” sowie die zugehörige MFC-Testanwendung “Server1394” beschrieben. Auf das Interface zur Bibliothek wird dabei nur kurz eingegangen, eine umfangreichere Beschreibung ist in Anhang A zu finden.

4.1 Das Interface

Das Interface ist in der Klasse *CVideoServer* gekapselt. Sie enthält einige für die Initialisierung wichtige Membervariablen, einen Zeiger auf die Klasse *CCentralModule* (das Hauptmodul bzw. die Zentrale), sowie Funktionen zum Starten, Stoppen und zur Übergabe von Frames. Die zugehörige Headerdatei “VideoServer.h” muß zusammen mit der Bibliothek MSysLib und der Bibliothek für die Containerklasse *CStreamBuffer* (siehe Anhang B) in die eigene Anwendung eingebunden werden. Nach Initialisierung und Starten des Videoservers muß die Anwendung letztgenannte Klasse mit den gegrabbten Frames sowie Nebeninformationen füllen und an das Interfacemodul übergeben.

4.2 Das Hauptmodul (Zentrale)

4.2.1 Funktionsbeschreibung

Dieses Modul wird zusammen mit dem Kommandomodul zuerst gestartet. Es ist sozusagen die Schaltzentrale. Von hieraus werden alle anderen Module (Threads) gestartet, gesteuert und gestoppt. Außerdem führt dieses Modul ein Logbuch in Form einer Textdatei über wichtige Ereignisse.
Wesentliche Steuerbefehle werden allerdings vom Client über die Kommandostrecke (siehe Abbildung 4.1) gesendet. Dieser Rückkanal über das IP-Netz ermöglicht die Fernsteuerung des Servers und ist daher ein wesentlicher Teil des Videoservers.
Das Hauptmodul ist auch für die Initialisierung aller globalen Variablen, Klassen sowie der Nachrichten für die Kommunikation zwischen Kommandomodul und Hauptmodul verantwortlich. Letztere sind in der Klasse *CUserMessages* (*UserMessages.h* und *UserMessages.cpp*) gekapselt.

4.2.2 Klassen

Das Hauptmodul wird durch die Klasse *CCentralModule* repräsentiert. Die o.g. Interfaceklasse enthält einen Zeiger auf diese Klasse, um sie zu initialisieren und zu starten. Die Klasse *CCentralModule* wiederum enthält Zeiger auf die Klassen der weiteren Module : *CCommandThread*, *CConverterThread*, *CEncoderThread* und *CRTPServerThread*. Beim Starten des Hauptmodules wird immer auch das Kommandomodul gestartet, ohne das der Videoserver nicht funktionsfähig wäre. In dieser Konstellation befindet sich der Videoserver im Wartemodus, er kann noch keine Frames verarbeiten aber schon Befehle vom Client entgegennehmen. Diese könnten zum Beispiel das Starten der anderen Module beinhalten.

Der Videoserver kann aber auch gleich im Vollmodus gestartet werden. Voraussetzung für das Funktionieren ist dann unter anderem die Wahl eines gültigen Videoprofiles (siehe 4.6. und Anhang E). Die Profile sind in der Klasse *CProfiles* gekapselt und dienen der Steuerung des Konverters und des Encoders. In ihr sind diverse Einstellungen bzw. Informationen über den Videostrom enthalten.

Die Klasse *CUserMessages* schließlich kapselt eine Reihe von Nachrichten-IDs, sowie Funktionen zum Umgang mit diesen Nachrichten. Sie wird vom Hauptmodul initialisiert und dem Kommandomodul bei dessen Start übergeben. Das Kommandomodul kann somit diese Nachrichten beim Aufruf der Funktion *CCentralModule::SendCommand* zusammen mit zwei Parametern verwenden. Die Funktion *SendCommand* “empfängt“ diese Nachrichten, führt eine mit der Nachricht verknüpfte Memberfunktion des Hauptmodules aus und verwendet die Funktion *SendAnswer* des Kommandomodules zum Senden der Bestätigung an den Client.

4.3 Das Kommandomodul

4.3.1 Funktionsbeschreibung

Das Kommandomodul muß server- und clientseitig implementiert werden, mit Anpassungen an die jeweilige Aufgabe. In diesem Fall wurde es in den Videoserver integriert, während auf Seite des Clients eine eigenständige Anwendung die Kommunikation mit dem Server übernimmt. Diese Anwendung dient vorallem Testzwecken. "Per Hand‘‘ können Steuerbefehle an den Server abgesetzt werden, während der Empfang des Videostreams parallel dazu von der am HHI entwickelten Anwendung MP4Play.exe übernommen wird.
In einer noch zu entwickelnden Clientanwendung sollen später beide Anwendungen zusammengefaßt werden.
Nachfolgend ist der Nachrichtenfluß der Kommandostrecke dargestellt. Zu beachten ist dabei, daß das Kommandomodul mit dem Hauptmodul über die gemeinsame Klasse *CUserMessages* verbunden ist. Diese anwendungsspezifischen, selbstdefinierten Nachrichten (nicht zu verwechseln mit den Windows-Nachrichten) verhindern in diesem Fall eine einfache Trennung der beiden Module. Im Anhang D gibt es eine Liste aller Kommandos mit den zugehörigen möglichen Parametern, sowie einer Kurzbeschreibung.

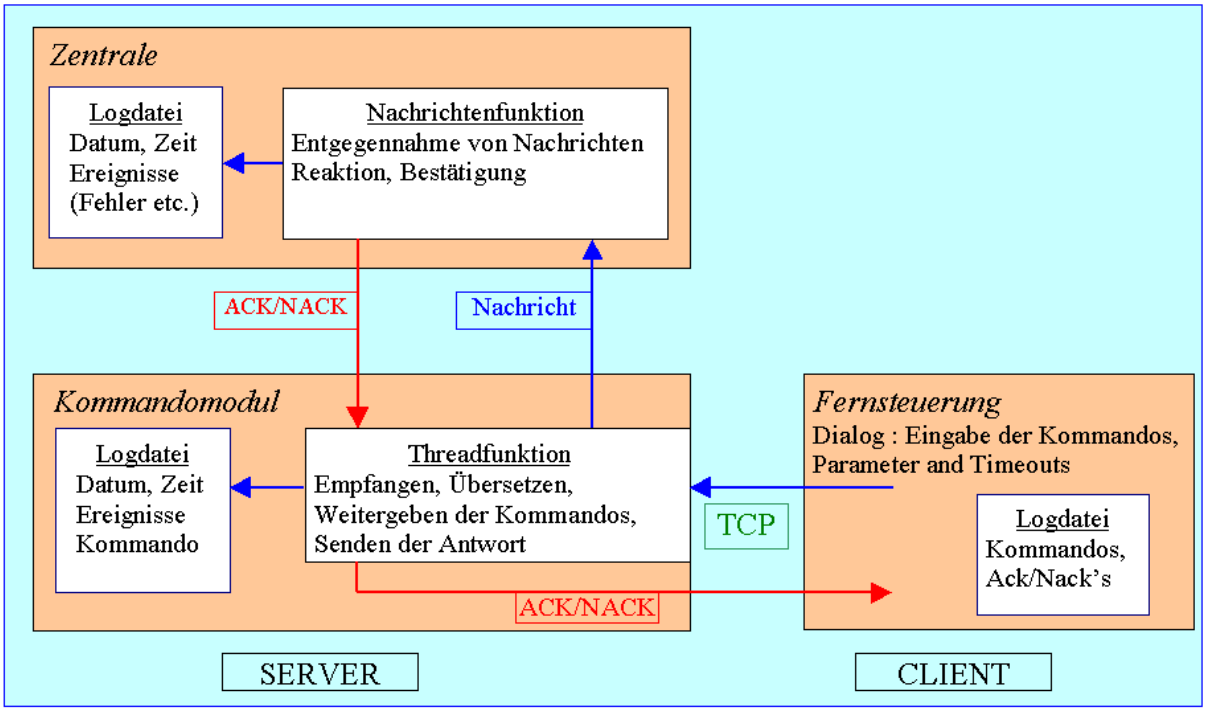


Abb. 4.1 : Kommandostrecke

Die Steuerkommandos des Clients werden in Form von US-ASCII-Strings übertragen und enthalten den Befehl sowie zwei Parameter. Nicht jeder Befehl benötigt zusätzliche Parameter, in diesem Fall werden die nicht benötigten Parameter zu Null gesetzt, bzw. vom Server nicht weiter beachtet.
Die Bestätigung des Servers auf jedes Kommando erfolgt ebenfalls in Form von Strings, welche den eben empfangenen Befehl enthalten und bei denen der erste Parameter als 1 (positive Bestätigung = ACK) oder 0 (negative Bestätigung = NACK) gesetzt wird. Der zweite Parameter wird teilweise ebenfalls verwendet und dient zum Beispiel der Übertragung von angeforderten Informationen.
Diese Hin- und Rückübertragung in Form von Strings erfordert server- und clientseitig einen einfachen Parser, welcher auch die Validierung und Übersetzung der Kommandos übernehmen muß. Nach Identifizierung des Kommandos erfolgt eine Meldung an das Hauptmodul, welches daraufhin die notwendigen Schritte ausführt und den positiven oder negativen Abschluß wieder an das Kommandomodul zurückmeldet. Dieses sendet dann eine Bestätigung an den Client über die noch offene TCP-Verbindung.
Der Kommandofluß wird auf beiden Seiten in einer Textdatei mitprotokolliert, sodaß im Fehlerfall die Logbücher miteinander verglichen werden können. Dazu ist es erforderlich, die Uhren aller beteiligten Computer möglichst zu synchronisieren, um die Ereignisse besser nachvollziehen zu können.

Noch zu erwähnen sind die clientseitigen, kommandospezifischen Timeouts, welche dem Benutzer nach einer gewissen Zeit das Ausbleiben der Serverbestätigung melden. In einer späteren Projektstufe kann daraufhin eine automatische oder semiautomatische Reaktion der Clientanwendung erfolgen, bisher muß dies allerdings der Benutzer selbst tun. Zu beachten ist dabei auch, daß der Client nach diesem Timeout die Verbindung beendet,

sodaß auch der Server einen Fehler meldet, da er seine eventuell verspätete Bestätigung nicht mehr los wird. Der Befehl kann also trotzdem bearbeitet worden sein. Dies wäre ein Fall, welcher mittels der Logbücher leicht erkannt werden kann.
Der Abbruch der TCP-Verbindung nach jedem Kommando ist nicht notwendig aber sinnvoll, da die Clientanwendung beliebige bzw. beliebig viele Server steuern können soll. Die Handhabung vieler ständiger Verbindungen wäre deutlich schwieriger und in unserem Anwendungsfall überflüssig, da Kommandos in der Regel nur gelegentlich übertragen werden müssen. Die TCP-Kanäle wären somit größtenteils ungenutzt. Die Umstellung auf dauerhafte Verbindungen wäre aber einfach zu realisieren.

4.3.2 Klassen

Zur Kapselung der Funktionalitäten des Kommandomodules wurde die Threadklasse *CCommandThread* eingerichtet. Es handelt sich hierbei um einen einfachen Workerthread, welcher von der MSysLib-Klasse *Thread* abgeleitet ist und der einmal gestartet die Memberfunktion *run()* ausführt. Diese enthält wiederum eine Schleife, welche an einem TCP-Port auf ankommende Verbindungsanfragen eines Clients wartet. Trifft diese ein, so wird eine Verbindung aufgebaut und der Kommandostring empfangen. Der String wird anschließend in die Nachricht und die beiden Parameter aufgeteilt. Dazu wurde die Klasse *CCommandParser* erstellt, welche Funktionen zum Analysieren und Zusammensetzen der Kommandostrings kapselt. Im Anhang C ist diese Klasse vorgestellt. Sie kann in Form einer statisch gelinkten Bibliothek in eigene (Client-) Anwendungen integriert werden.
Stimmt die eingetroffene Nachricht mit einer der Nachrichtenklasse *CUserMessages* überein, so wird dessen ID zusammen mit den beiden Parametern an das Hauptmodul übergeben und anschließend die Antwort wieder als String an den Client zurückgesendet.
Zum Umgang mit der TCP-Verbindung, also dem Erstellen und Verwalten von Sockets werden die MSysLib-Klassen *SocketManager* und *Socket* verwendet.

4.4. Das Konvertermodul

4.4.1 Funktionsbeschreibung

Da der nachfolgende Encoder mit einem festen Bildformat arbeitet, muß sichergestellt werden, das dieses auch bei ihm ankommt. Dazu dient der Konverter. Ursprünglich war nur die Implementierung von Routinen zur Größenkonvertierung, genauer gesagt zur Unterabtastung, vorgesehen. Um die Wiederverwendbarkeit des Videoservers jedoch zu steigern, kann nun auch eine Typenkonvertierung vorgenommen werden. Eine zeitliche Unterabtastung n:1 ist ebenfalls möglich.
Die Übergabe eines Frames in Form eines Zeigers auf die Klasse *CStreamBuffer* vom Hauptmodul zum Konverter erfolgt über die Funktion *SendFrame* des Konvertermoduls. Dort wird es ein erstes Mal überprüft und in einen modulinternen Puffer kopiert. Hierbei erfolgt auch gleich die Anpassung der externen an die interne Bildrate. Liegt das neue Frame nicht im YUV420-Format vor, so wird zunächst eine Typenkonvertierung durchgeführt. Als Eingangsmaterial können dabei YUV422, YUV444 oder RGB24-Bilder verwendet werden. Erstere müssen in planarer Form (YYY...UUU...VVV...), letztere können auch interleaved

(RGBRGBRGB...) vorliegen. Das Ergebnis der Typenkonvertierung oder des einfachen Kopierens befindet sich in einem Zwischenpuffer (Backbuffer), welcher dann der Größenkonvertierung zugeführt wird. Diese Stufe beinhaltet Funktionen zum Unterabtasten der Bilder, sollten sie in einem größeren Format als notwendig vorliegen. Die Bildgröße muß dabei in Breite und Höhe ein geradzahliges Vielfaches der Zielgröße sein. Möglich sind Unterabtastungen von 2:1, 3:1 oder 4:1 in beiden Dimensionen, wobei horizontale und vertikale Unterabtastung getrennt behandelt werden.
Das resultierende Bild muß auf Grund der Einschränkungen des verwendeten Encoders außerdem wieder genau in Makroblöcke (16x16 Pixel) zerlegbar sein. Eine Hochtastung ist nicht möglich und auch nicht sinnvoll, da dies auch beim Empfänger erfolgen kann.

Ein wichtiger Punkt bei Unterabtastung ist das Auftreten von Alaising-Fehlern. Sie werden auch Spiegelungsfehler genannt, da durch das periodische Auftreten des Spektrums nach der Modulation des Signales mit einer Deltaimpulsfolge halber (drittel, viertel) Kreisfrequenz Bereiche in das Grundspektrum hineingespiegelt werden. Dies betrifft somit vorallem die hohen Frequenzbereiche, d.h. es kann z.B. zum Auftreten hochfrequenter Muster kommen. Abgesehen von der möglichen visuellen Störung wird dadurch auch die Effizienz des Encoders gesenkt, da dieser an Frequenzgänge mit geringen hochfrequenten Anteilen angepaßt ist.
Statt der einfachen Unterabtastung werden aus diesem Grunde auch Möglichkeiten der Vorfilterung zur Verfügung gestellt. Es stehen drei FIR-Filter zur Auswahl. Für das erste Filter (Summenfilter) berechnen sich die Zielpixel pro Dimension nach :

$$y(n) = \sum_{q=0}^{Q} \frac{1}{Q+1} X_{(n-q)} \quad \text{mit } Q = 1;2;3 \tag{4.1}$$

Die Filterkoeffizienten sind also { ½ , ½ } ; { ⅓ , ⅓ , ⅓ } ; { ¼ , ¼ ; ¼ , ¼ }. Im Gegensatz zu Gleichung 4.1 werden sie allerdings in der Regel so verschoben, daß sie um den Nullpunkt herum liegen, wodurch das Filter aber nicht mehr kausal ist. Für eine ungerade Anzahl an Koeffizienten ergibt sich dann die Gleichung für ein diskretes Halteglied höherer Ordnung ([10]) :

$$y(n) = \frac{1}{2M+1} \sum_{k=-M}^{M} X_{(n+k)} \tag{4.2}$$

Dieses Filter ist einfach zu implementieren und recht schnell, da es nur eine Mittelwertbildung über benachbarte Pixel durchführt. Es verbessert die Bildqualität schon sichtbar, allerdings ist die Übertragungsfunktion $H_1(j\Omega) = \cos(\Omega/2) * e^{-j\Omega/2}$ (für Q=1) noch weit von einem idealen Tiefpaß entfernt. Auch für Q=2 und Q=3 ist das Tiefpaßverhalten nur befriedigend. Die Impulsantwort des Summenfilters ist ein diskreter Rechteckimpuls und hat als Amplitudengang einen cosinusförmigen Verlauf. Dies ist in Abbildung 4.2 bei den ersten drei Amplitudengängen erkennbar.
Speziell für 2:1 Unterabtastungen steht ein weiterer, symetrischer Filter 2. Ordnung (3-Tap-Dreiecksfilter) zur Verfügung. Dieser hat einen flachen Amplitudengang. Damit dämpft er aber auch niedrige und mittlere Frequenzen deutlich.

Eine höherwertigere, allerdings aufwendigere Variante ist die Vorfilterung mit einem 7-Tap-Filter. Der benutzte Filter von Marc Antonini et.al. ([11]) hat nachfolgende Koeffizienten, woraus sich der Amplitudengang laut Abbildung 4.2 ergibt. Die Filterung erfolgt erst horizontal dann vertikal.

Koeffizienten 3-Tap-Dreiecksfilter : $b_{-1} = b_1 = 0{,}25$
$b_0 = 0{,}5$
Koeffizienten 7-Tap-Filter: $b_{-3} = b_3 = -0{,}045636$
$b_{-2} = b_2 = -0{,}028772$
$b_{-1} = b_1 = 0{,}295636$
$b_0 = 0{,}557544$

Eigenschaften : lineare Phase, stabil, nicht kausal

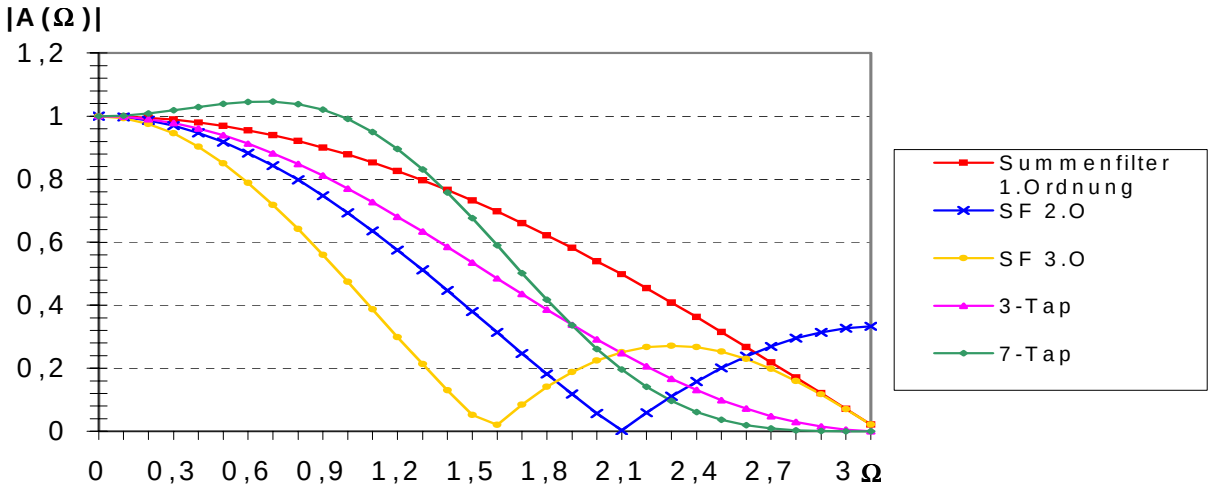


Abb. 4.2 : Amplitudengänge verschiedener Filter

Der 7-Tap-Filter wird vom Konverter ebenfalls nur für 2:1 Unterabtastungen benutzt. Die Koeffizienten folgen im Gegensatz zu den anderen Filtern dem Verlauf einer si-Funktion, dessen Fouriertransformierte den idealen, rechteckförmigen Tiefpass ergibt. Mit mehr Koeffizienten (9-Tap, 11-Tap usw.) könnte der si-Verlauf noch genauer approximiert werden, allerdings auf Kosten des Rechenaufwandes.

Wurden alle Konvertierungsstufen erfolgreich durchlaufen, so wird das Bild dem Encodermodul übergeben.
Da jedes Frame Informationen über den Inhalt mitgeliefert bekommt, ist der Konverter in der Lage, einkommende Frames unabhängig vom Vorgänger zu bearbeiten. Dies bedeutet konkret, daß letztendlich beliebige (aber konvertierbare) Frames dem Videoserver übergeben werden können, ohne diesen bei Änderungen am Format neu starten zu müssen. Die internen Puffer passen sich neuen Framegrößen oder -typen automatisch an. In der Praxis wird dieser Fall aber nur selten auftreten. Denkbar wäre hier z.B. die Übertragung von Bildsequenzen verschiedener Kameras über einen Videokanal im Zeitmultiplex.

4.4.2 Klassen

Abbildung 4.3 zeigt die Funktionsweise des Konverters im Zusammenhang mit den verwendeten Klassen :

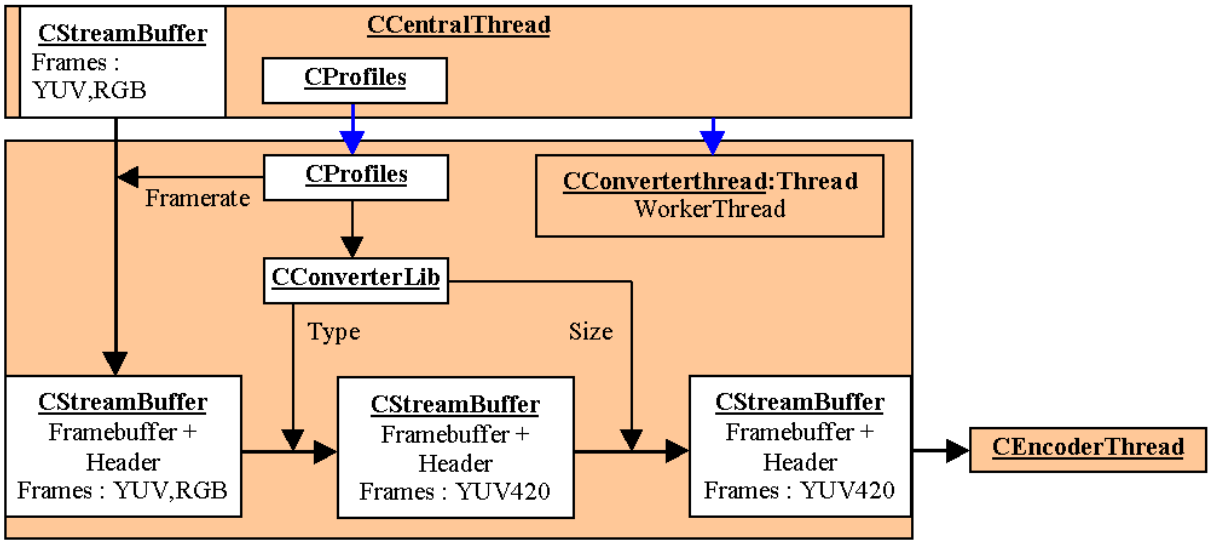


Abb. 4.3 : Aufbau des Konverters

Der Konverter ist in der Threadklasse *CConverterThread* gekapselt, welche von der MSysLib-Klasse *Thread* abgeleitet ist. Die Übergabefunktion für die eintreffenden Frames ist *SendFrame*, hier wird das neue Bild in einen internen Puffer kopiert. Die vom Hauptmodul initialisierte Klasse *CProfiles* gibt die Einstellungen vor, welche für die diversen Konvertierungsstufen benötigt werden. Sie können nur vor dem Starten des Konverters geändert werden.
Die Typen- und Größenkonvertierung sowie das Weiterversenden erfolgt in der *run*-Funktion der Threadklasse. Dabei kommt die Bibliothek “Converterlib.lib“ mit der Klasse *CConverterLib* zum Einsatz, welche die eigentlichen Konvertierungsfunktionen beinhaltet. Kommt es im Verlauf der Operationen zu einem Fehler, z.B. wenn die Größe nicht konvertierbar ist, so wird das Frame verworfen. In einem Zähler, welcher auch vom Client mit dem Kommando GETLOSTFRAMES abgefragt werden kann, wird die Anzahl an verlorenen Frames festgehalten.

4.5 Das Encodermodul

4.5.1 Funktionsbeschreibung

Dieses Modul übernimmt die Codierung der eintreffenden Frames in einen MPEG4-konformen Bitstream. Es werden dabei reine VOP’s (video object planes) erstellt, bestehend aus VOP-Header und -Rumpf. Entsprechend der eingestellten I-Frame-Periode wird in regelmäßigen Abständen ein Bild rein intracodiert. Die Bilder dazwischen sind P-Bilder, für die gemäß der Funktionsweise eines Hybridcoders eine Bewegungsschätzung und -kompensation durchgeführt wird (siehe Kapitel 3.5). Dadurch wird der Bedarf an Bitrate für

diese Bilder gesenkt. Der am HHI entwickelte MPEG4-Codec verteilt die zu Verfügung stehende Bitrate nach einem Optimierungsalgorithmus ([9]) auf die I- und P-Frames.
Die zu codierenden Frames müssen in einem bestimmten Format vorliegen. Es sind nur YUV420-Frames erlaubt, welche zudem eine Breite und Höhe haben müssen, welche durch die Makroblockgröße 16 teilbar ist. Ränder darüberhinaus sind also nicht erlaubt. Für die Bildrate gilt, daß sie aus dem Integer-Zahlenbereich kommen muß. Bei anderen Werten, wie 12,5 Hz oder 7,5 Hz kann ein gerundeter Wert angegeben werden.
Am Eingang des Modules nimmt ein Doppelpuffer die Frames auf. Dieser Doppelpuffer dient zur kurzzeitigen Überbrückung von Performanceeinbrüchen, welche zum Beispiel beim Starten von anderen Programmen oder Modulen entstehen können, wenn die Rechnerleistung ohnehin im Grenzbereich liegt. Beim Testen des Videoservers wurde zudem festgestellt, daß bei viel Bewegung oder einem schnellen Kameraschwenk der Encoder stark belastet wird und einige Frames verwerfen mußte. Um dies zumindest teilweise aufzufangen, wird ein zweiter Eingangspuffer bereitgestellt. Ein noch größerer Puffer (Ringpuffer) ist denkbar, braucht aber weiteren Speicherplatz und kann eine größere oder ständige Überlastung des Videoservers z.B. auf Grund langsamer Hardware auch nicht auffangen.

4.5.2 Klassen

Wie in Abb. 4.4 zu sehen, ist das Encodermodul in der Threadklasse *CEncoderThread* gekapselt. Sie ist rund um den Encoderkern, welcher durch die Klasse *CVideoEnc* repräsentiert wird, aufgebaut. Diese Klasse ist Bestandteil der Bibliothek “VideoEnc.lib“.

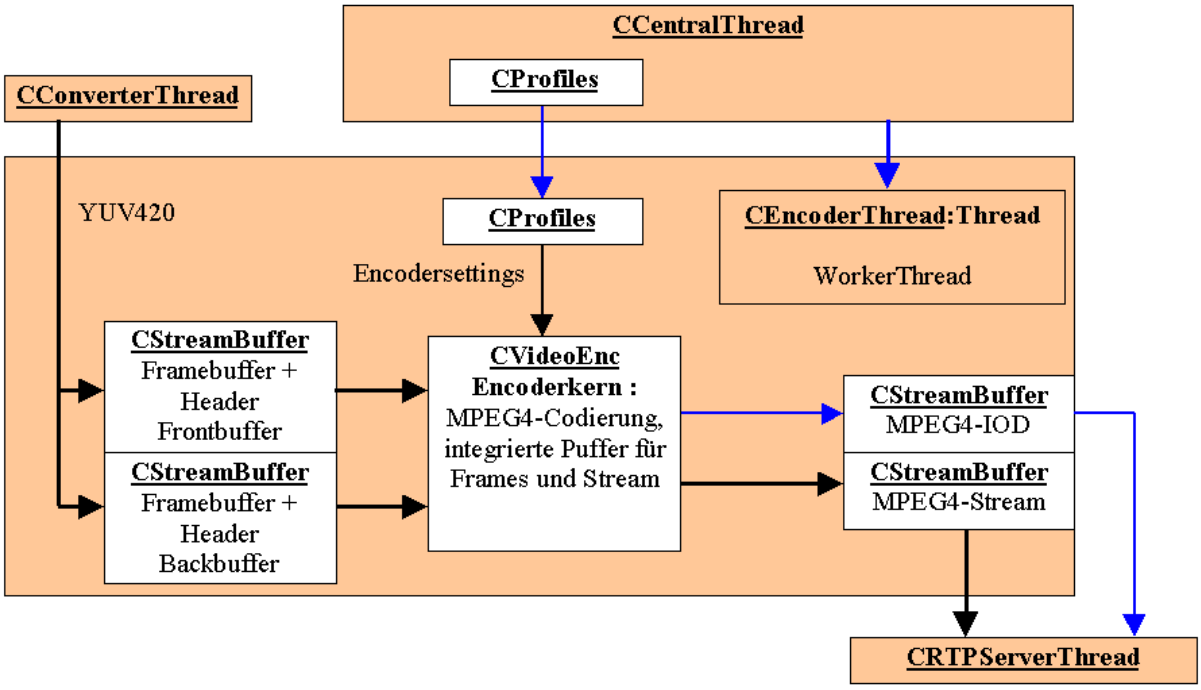


Abb. 4.4 : Aufbau des Encodermodules

Die Encodereinstellungen werden vor dem Start über die vom Hauptmodul initialisierte Klasse *CProfiles* definiert. Sie können nur bei abgeschaltetem Encodermodul geändert werden.
Der Datenstrom erfolgt wieder über die Klasse *CStreamBuffer*, welche eingangsseitig YUV420-Frames liefert und am Ausgang zur Übermittlung des Initial Object Descriptors (IOD) und des codierten Frames an den RTSP/RTP-Server genutzt wird. Die konverterseitige Eintrittsfunktion ist *Sendframe(CStreamBuffer* cStrBuf)* in der das eintreffende Frame in den Frontbuffer oder, wenn dieser noch belegt ist, in den Backbuffer kopiert wird. Der Encoderkern befindet sich in der *run()*-Funktion der Threadklasse. Zur Übermittlung des codierten Frames an die Eintrittsfunktion des nächsten Modules dient ein Pointer auf dessen Threadklasse. Beim Startvorgang des Encodermodules wird dieser Pointer initialisiert, sofern der RTP-Server schon hochgefahren wurde, ansonsten muß er später nach dessen Start initialisiert werden. Wichtig ist in diesem Zusammenhang, daß der IOD noch vor dem ersten Frame, also gleich nach Übergabe des Pointers auf den RTSP/RTP-Server, erstellt und diesem übermittelt werden muß.

4.6. Der RTSP/RTP-Server

4.6.1 Funktionsbeschreibung

Kernstück dieses Modules ist der integrierte RTSP/RTP-Server, welcher am Heinrich Hertz Institut entwickelt wurde und im Form einer Bibliothek bereitgestellt wird. Zur Übertragung der Daten wird RTP benutzt. Die Funktionen hierfür bietet die an der Columbia Universität entwickelte RTPLib. Das zugehörige Kontrollprotokoll RTCP ist nicht implementiert.
Für den RTSP-Server steht ein BSD- und Win32-kompatibler TCP/IP-Stack zur Verfügung. Er ist unter Win32, Linux und MacOS getestet, wobei nur erstere Funktionalität in dieser Anwendung genutzt wird. Weiterhin ist der Server in der Lage, sowohl MPEG-4 und Quicktimedateien als auch Livedatenströme zu handhaben. Einzelheiten zum Serverkern können der Dokumentation [6] entnommen werden.
Nach dem Hochfahren des RTPServer-Threads wird zuerst der Initial Object Deskriptor vom Encoder angefordert. Wurde dieser Thread noch nicht gestartet, so muß dieser Schritt bei dessen Start nachgeholt werden. Der Deskriptor ([8]) hat eine Länge von einigen Bytes und enthält Informationen über den folgenden Stream. Diese Informationen müssen dem Client vor dem Beginn des Streamings mitgeteilt werden. Der IOD wird also im Serverkern gespeichert bis eine Verbindungsanfrage eines Clients eintrifft.
Nach Aufbau einer Verbindung und Initialisierung des Streams (DESCRIBE-, SETUP-Methode) wird vom Client eine PLAY-Nachricht gesendet (siehe [4]) und damit das Streaming der vom Encoder eintreffenden VOP's eingeleitet. Der RTP-Server paketiert die VOP's und versendet sie über einen UDP-Kanal. Der Client ist für das Ordnen und Zusammensetzen der Pakete gemäß den Sequenznummern im RTP-Header verantwortlich. Aufgrund der schon erwähnten Beschränkungen von UDP (siehe Kapitel 3.2) können verlorengegangene Pakete nicht wiederangefordert werden. Das Streaming bzw. die RTSP-Session kann von Seiten des Clients (TEARDOWN-Methode) oder vom Server beendet werden.

4.6.2 Klassen

Abbildung 4.5 zeigt den Aufbau des RTSP/RTP-Servermodules.

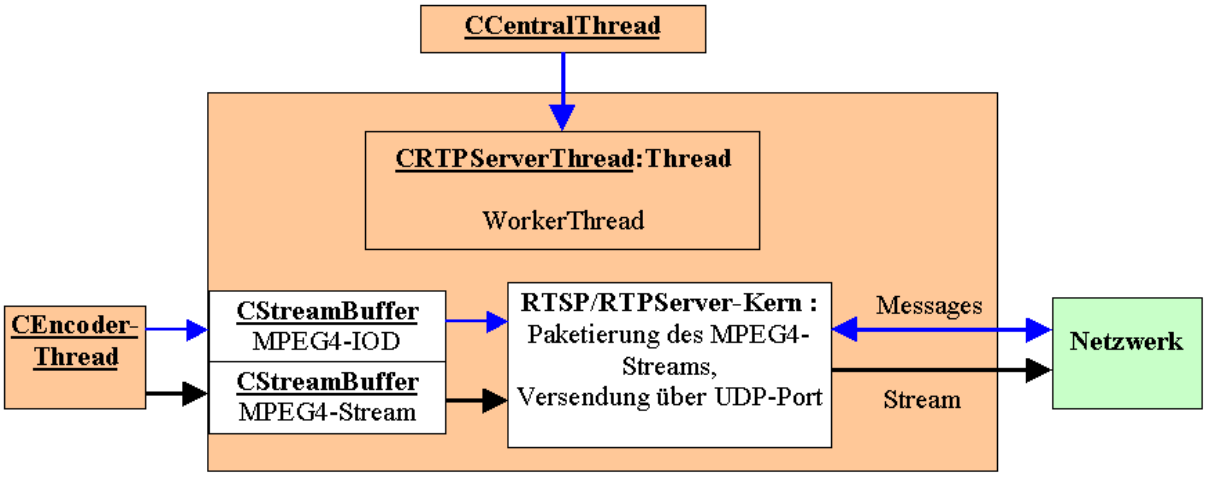


Abb. 4.5 : Aufbau des RTSP/RTP-Servermodules

Der bereits erwähnte Kern, sowie diverse Puffer sind in einer Threadklasse (*CRTPServerThread*) gekapselt. Der Name bezieht sich auf das eigentliche Streaming mittels RTP. Die Threadklasse enthält Funktionen zum Initialisieren, Starten und Stoppen des Kernes, sowie zur Verwaltung der Puffer. Der Kern selbst legt intern einen weiteren Thread pro Session an und besitzt ebenfalls Funktionen zum Empfang und Puffern eintreffender VOP's. Die um ihn herum gebaute Threadklasse dient darum in erster Linie der besseren Handhabung sowie der Vereinheitlichung des modularen Aufbaus des Videoservers.

4.7 Integrierte Videoprofile

Um die Netzlast durch den Videostream an gegebene oder veränderliche Netzbandbreiten anzupassen, sind in den Videoserver mehrere sogenannte Profile integriert, welche durch die Klasse *CProfiles* gekapselt werden. Diese enthalten die Größen Bildbreite, Bildhöhe, Bildrate, Bitrate und I-Frame-Periode. Ausgehend von der höchsten Auflösung der verwendeten Kamera 640x480x15Hz wurden Bildgröße und Bitrate stufenweise heruntergesetzt. Es ergeben sich die vordefinierten Profile 1-8 mit Bitraten von 1200 kbit/s bis hinunter zu 200 kBit/s für schmalbandige Systeme. Die weiteren Bildformate beziehen sich auf das häufig verwendete Format CIF, außerdem QCIF, 4CIF, ITU-R 601. sowie Super-VGA für hochauflösende Videos. Für eine Echtzeitcodierung hochauflösender Videos mit hoher Bildrate ist allerdings entsprechende Hardware notwendig. Es ist zu erwarten, das dies in naher Zukunft auch auf Standard–PC's realisiert werden kann.
Im Profil 0 kann der Benutzer im Rahmen der Möglichkeiten der verwendeten Kamera bzw. abhängig vom verwendeten Bildmaterial die Einstellungen auch selbst vornehmen.
Wird das aktuelle Profil nicht beim Startvorgang, sondern über die Kommandostrecke festgelegt, so muß der Server ggfs. heruntergefahren und mit neuen Einstellungen wieder

hochgefahren werden. Der Client muß ebenfalls stoppen und den neuen Videostream über den im RTSP/RTP-Server definierten TCP-Port erneut anfordern. Darum ist es auch nicht zulässig, Änderungen am Profil 0 bei laufendem Videoserver vorzunehmen, wenn dieses Profil aktuell ist.
Eine Auflistung der integrierten Profile ist in Anhang E zu finden. Die Bitraten wurden ausgehend von der Bandbreite von Ethernet/WLAN, dem geschätzten Bedarf an Bandbreite durch parallel laufende Anwendungen (3D-Modellierung, Verkehrsdatenübertragung) und schließlich nach Untersuchungen der ausgewählten Testsequenzen (siehe Kapitel 5.) gewählt. Sollte sich die Notwendigkeit für Korrekturen auf Grund anderer Anforderungen ergeben, so ist dies einfach zu realisieren. Um die jeweils optimale Bitrate herauszufinden, kann sich der Benutzer dazu des einstellbaren Profiles 0 bedienen.

Die gewählte Periode für I-Frames basiert auf einer Abwägung verschiedener Einflüsse aufgrund des Einsatzgebietes. Es stehen für jede Auflösung und Bildrate zwei I-Frame-Perioden verbunden mit unterschiedlichen Bitraten zu Verfügung. Während beim qualitativ hochwärtigsten Profil 1 alle halbe Sekunde ein I-Frame gesendet wird, ist diese Zeitdauer beim benachbarten Profil 3 auf eine Sekunde erhöht, während die Bitrate reduziert werden konnte. Letztere Einstellung hat allerdings Nachteile bei fehlerhaft übertragenen Datenpaketen, welche länger anhaltende Bildstörungen verursachen können.
Steht jedoch ein Netz mit genügend Bandbreite zur Verfügung, so kann die I-Frame-Periode auch weiter heruntergesetzt werden. Eine häufigere Verwendung von I-Frames vermindert die Prozessorbelastung, da die für I-Frames wegfallende Bewegungsschätzung und Bewegungskompensation einen nicht unerheblichen Rechenanteil am Kodierprozeß haben.

4.8 Testanwendung

4.8.1 Die MFC-Anwendung “Server1394.exe“

Die MFC-Anwendung benutzt die Bibliothek VideoServer und dient zum Testen sowie zum Betrieb eines Demonstrationsnetzes. Sie wurde mit dem MFC-Anwendungsassistenten erstellt und bietet ein Menü und ein Fenster zum Anzeigen der Frames. Der Name “Server1394“ bezieht sich auf den Firewire-Framegrabber.
Zusätzlich zu den Steuermöglichkeiten, welche durch die Fernsteuerung gegeben sind, werden in der MFC-Anwendung weitere Dialoge für Benutzereingaben bereitgestellt. Über die Menüleiste der Anwendung können diese aufgerufen werden. Sie dienen in erster Linie der schnellen Vor-Ort-Konfiguration des Videoservers. Unter dem Menüpunkt “Camera Settings“ hat der Benutzer ferner die Möglichkeit, eine Kalibrierung der Kamera (Farbe, Schärfe etc.) durchzuführen. Dies funktioniert aber nur bei laufendem Grabber.
Unter dem Menüpunkt “Remote Control“ kann der Port des Kommandomodules festgelegt werden. Außerdem kann das Kommandomodul komplett gegenüber fremden Hosts gesperrt werden.

4.8.1 Der Framegrabber

Das Framegrabbermodul ist eine eigenständige Threadklasse, die bei Bedarf einfach modifiziert oder ausgetauscht werden kann. Die Klasse enthält als Kern die Bibliothek “1394Camera“ (1394Camera.h , *.lib, *.dll) welche am Robotics Institute der Carnegie Mellon University entwickelt wurde und einfachen Zugriff auf den ebenfalls dort entwickelten Firewiretreiber bietet. Um diesen Kern herum wurde die Threadklasse *CGrabberThread* entwickelt, um Initialisierung, Start, Stop etc. besser handhaben und die Vorteile des Multithreadings ausnutzen zu können.
Je nach Kameratyp können Frames mit verschiedener Auflösung und Framerate akquiriert werden. Die Auflösungen entsprechen dabei den gängigen VGA-Formaten, also 640x480, 800x600 etc. Die im Rahmen der Studienarbeit benutzte Kamera Sony DFW-V500 bietet Frames bis zur Größe 640x480 mit 30 Hz Bildwiederholrate im asynchronen Grabmodus, bzw. 15 Hz im synchronen Modus. Dies ist für das Projekt mehr als ausreichend, da auf dem Server noch andere rechenaufwendige Anwendungen laufen sollen und damit voraussichtlich die Grenzen heutiger Computer (Prozessortakt 2-3 GHz) erreicht werden.
Höher aufgelöste Bilder sind mit anderen Kameratypen möglich, dies geht jedoch auf Kosten der Bildrate, da die Übertragungsrate von Firewire auf derzeit maximal 400 MBit/s begrenzt ist.
Das Format der gegrabbten Frames ist YUV444, YUV422 oder YUV411 und zwar interleaved, d.h. die Frames müssen vor der Codierung noch in YUV420 (planar) konvertiert werden.

4.8.2 Die Fernsteuerung

Zur Fernsteuerung des Videoservers dient die Anwendung “RemoteControl“. Mir ihr können theoretisch beliebig viele Server angesteuert werden. Die Oberfläche besteht aus einem mit dem Dialogeditor erstellten Standarddialog (abgeleitet von der MFC-Klasse *CDialog*) und enthält einige Bedienfelder zur Auswahl des Servers, des Kommandos, der Parameter und zum Einstellen eines Timeouts in Millisekunden. Nach dieser Auswahl wird per Button “Send“ das Kommando zusammengesetzt und eine TCP-Verbindung mit dem Server aufgebaut. Über diese Verbindung wird das Kommando gesendet und eine bestimmte Zeit auf die Antwort gewartet. Diese wird dann in ihre Bestandteile zerlegt und der Inhalt im Ausgabefenster angezeigt. Alle Ausgaben werden zudem in einer Logdatei erfasst.
Für das Zusammensetzen und Zerlegen von Kommandos wird die statisch gelinkte Bibliothek “Commandparser.lib“ mit der Klasse *CCommandParser* benutzt, welche auch im Videoserver Verwendung fand.
Da die Fernsteuerung eine MFC-Anwendung ist, sind zu deren Ausführung einige MFC-spezifische DLL’s notwendig.

5. Auswertung

5.1 Untersuchungen der Videoprofile

Da die für WLAN zu erwartende Netzbandbreite im Gegensatz zu Fast-Ethernet deutlich geringer ist, muß untersucht werden, welche Bitraten und I-Frame-Perioden für die vordefinierten Videoprofile günstig sind. Dazu wurden PSNR-Messungen für zwei typische Verkehrsvideosequenzen durchgeführt. Die Sequenzen wurden aus zwei Perspektiven am Ernst-Reuter-Platz in relativ großem Abstand mit einer feststehenden Kamera aufgenommen. Ausschnitte aus den Szenen sowie die kompletten Untersuchungsergebnisse sind in Anhang F enthalten.

5.1.1 Wahl geeigneter Bitraten und Intraframeperioden

Einer der wichtigsten Fragen bei digitaler Codierung ist, bei welcher Bitrate man eine gewünschte Qualität erreicht. Oder anders herum, wie weit man mit der Bitrate heruntergehen kann, ohne einen bestimmten Qualitätslevel zu unterschreiten. Die Mathematik dieser Problematik ist unter der Bezeichnung “Rate-Distortion-Theory“ zusammengefaßt. Unter “Rate“ verstehen wir in unserem Fall die Bitrate, für die “Distortion“ (Verzerrung) wird oft der mittlere quadratische Fehler bzw. für logarithmische Darstellung das PSNR-Maß herangezogen. Das PSNR-Maß ist für 8-Bit-Werte wie folgt definiert :

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\text{mse}} \quad \text{in dB} \qquad \text{mit mse -> mean-squared-error} \qquad (5.1)$$

Für die Berechnung des mittleren quadratischen Fehlers werden die Bilder vor und nach der Codierung-Decodierung herangezogen. Generell gilt : Mit einer höheren Bitrate wird ein höherer PSNR-Wert erzielt. Der Verlauf ist aber abhängig vom Bildmaterial. Darum geben nachfolgende Ergebnisse auch nur einen Anhaltspunkt auf die möglichen Bitraten für den Fall einer feststehenden Kamera, ohne Schwenks, Zooms, raschen Helligkeits-veränderungen, Szenenwechseln und mit langsamer Bewegung der Objekte. Aufgrund dieser Einschränkungen wird eine hohe Qualität schon bei niedrigen Bitraten erwartet.
Die herangezogenen Testsequenzen wurden mit einer Sony DFW-V500 im unkomprimierten YUV411-Format aufgenommen. Sie zeigen eine typische Verkehrsszene mit ruhendem und fließendem Verkehr, sowie mit wenigen und vielen Objekten. Die Länge der Sequenzen beträgt jeweils 500 Frames. Es wurde der PSNR-Verlauf in Abhängigkeit von der Bitrate und der Intraframeperiode (IFP) aufgenommen. Somit ergeben sich 5 Kurvenverläufe für IFP’s von 3, 5, 7, 10 und 15 Frames :

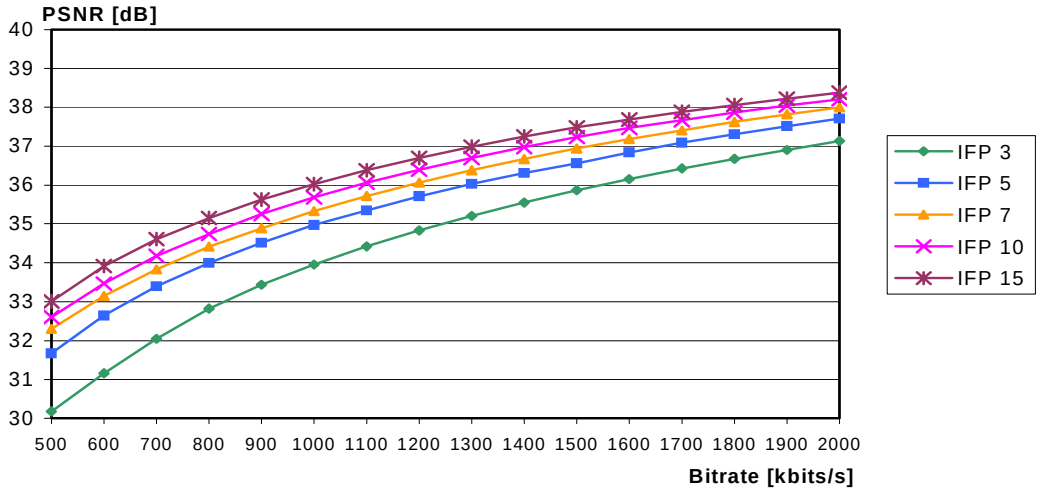


Abb. 5.1 : RDF für Testsequenz ErnstReuterPlatz A

Für dieselbe Testsequenz wurden auch PSNR-Verläufe für 5 Hz, sowie für die gefilterte und unterabgetastete Version aufgenommen. Diese sind in Anhang F aufgeführt.
Zu erkennen ist, das für größere IFP's weniger Bitrate benötigt wird. Nimmt man z.B. ein PSNR von 36 dB als Bezugspunkt, so liegen zwischen IFP 5 und 15 immerhin 300 kbits/s, die eingespart werden könnten. Da in den Kurvenverläufen aber noch nicht der Einfluß von Kanalfehlern enthalten ist, sollte keine zu große IFP gewählt werden, da fehlerhafte Blöcke dann für lange Zeit nachwirken.

5.2.1 Einfluß der Filterung bei Unterabtastung

Da der Encoder durch den zwischengeschalteten Konverter unabhängig vom Framegrabber arbeitet, ist es möglich, daß er mit geringerer Bildauflösung arbeiten soll. Der Konverter paßt die Bildgröße dann automatisch an. Da die durch Unterabtastung entstehenden Alaisingfehler die Bildqualität verschlechtern, wurde noch die Auswirkung der Vorfilterung mit den implementierten Filtern untersucht. Dazu wurde die bereits erwähnte Testsequenz ErnstReuterPlatz_A einer Filterung und 2:1 Unterabtastung unterworfen und dann die RDF für eine feste Intraframeperiode aufgenommen. Es ist dabei zu beachten, daß für die Berechnung des PSNR die Testsequenz vor und nach der Codierung-Decodierung herangezogen wurden, nicht wie üblich die Bilder zwischen Quelle und Senke (Abb. 5.2). Dazu hätte man auf Decoderseite geeignete (z.B. orthogonale) Filter benötigt. Somit kann nur eine Aussage über die Auswirkung auf die Codierung selbst getätigt werden, nicht über die Gesamtübertragungsstrecke :

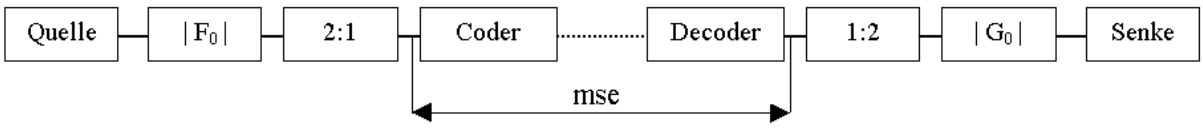


Abb. 5.2 : Übertragungsstrecke mit Unterabtastung

Die Ergebnisse der Messungen sind in Abb. 5.3 dargestellt. Man erkennt die unterschiedlichen Auswirkungen auf die Codiereffizienz des Encoders. Prinzipiell erreicht

man mit jeder Tiefpaßfilterung eine Verbesserung. Während aber das steilflankige 7-Tap-Filter speziell der Dämpfung hoher Frequenzen und somit der Verminderung von Alaisingfehler dient, dämpft das flache 3-Tap-Filter auch tiefere und mittlere Frequenzen und verursacht dadurch eine gewisse Unschärfe auch im unterabgetasteten Bild. Dies führt zu höheren PSNR-Werten, allerdings nur bezogen auf die in Abb. 5.2 gezeigten Meßpunkte. Das 7-Tap-Filter dagegen erhält diese Frequenzen bzw. verstärkt sie noch. (Überschwinger im Amplitudengang). Somit muß der Encoder die zur Verfügung stehende Bitrate für mehr Frequenzanteile, bzw. DCT-Koeffizienten verwenden. Ein optisch besseres Bild entsteht aber bei höherer Bitrate, wenn die nach der Filterung erhalten gebliebenen Details auch in entsprechender Qualität übertragen werden können.

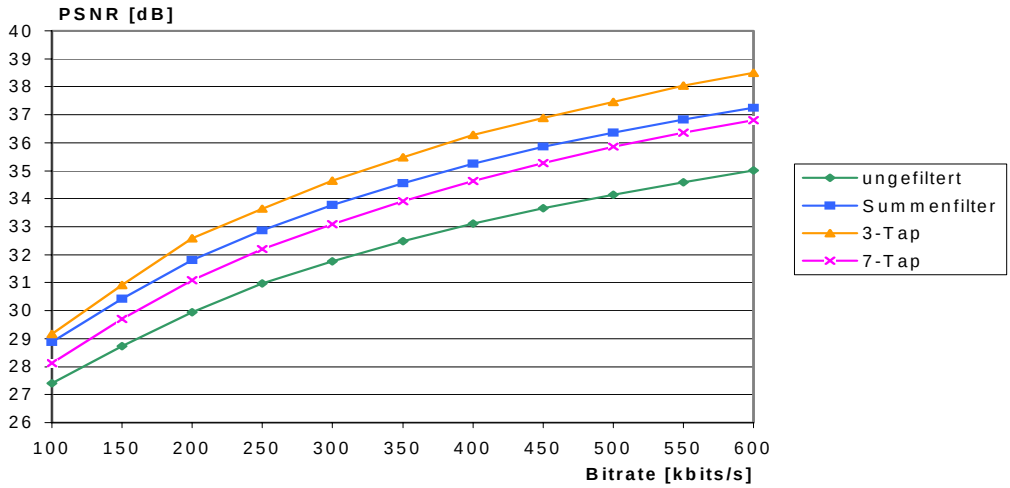


Abb. 5.3 : RDF für verschiedene Vorfilter

5.2 Verwendungsmöglichkeiten des Videoservers

Obwohl die Anwendung für den Fall der Kreuzungsüberwachung entwickelt wurde, sind aber auch diverse andere Einsatzmöglichkeiten denkbar. Diese sind allerdings an das Vorhandensein eines WinNT IP-Netzes gekoppelt. Denkbar wäre z.B. die Verwendung des Videoservers für die Gebäude- oder Raumüberwachung oder die Aufnahme eines Objektes mit mehreren Kameras. Die Anzahl der ansteuerbaren Videoserver wird letztendlich nur durch die Fähigkeiten der Hard- und Software der Clientseite und der Kapazität des Netzes und des Clientrechners begrenzt. Durch die einzelne Konfigurierbarkeit der Server kann dabei die Netzbandbreite auf wichtige und unwichtige Videoströme verschieden aufgeteilt werden. Für die Beobachtung wichtiger Objekte kann so z.B. eine höhere Auflösung verbunden mit einer höheren Bitrate gewählt werden, während unwichtige Bereiche nur mit geringerer Bildqualität überwacht werden. Ebenso kann je nach beobachteten Objekt die Bitrate für höhere Auflösung (ruhende Objekte) oder höhere Bildrate (sich bewegende Objekte) verwendet werden.
Da die API auch Funktionen zum Initialisieren der Videoservers bietet, ist dieses Modul aber auch zum einfachen, ungesteuerten Streamen von Videosequenzen (Live oder gespeichert) geeignet. Das Kommandomodul sollte dabei möglichst gegenüber fremden Hosts gesperrt werden.

6. Zusammenfassung

6.1 Inhalt

In der vorliegenden Studienarbeit wurde ein steuerbarer Videoserver zur Codierung und Übertragung von Echtzeitvideodaten über IP-basierte Netze beschrieben. Implementiert wurden neben dem eigentlichen Servermodul, welches im Form einer dynamisch gelinkten Bibliothek zur Verfügung gestellt wird, eine menügeführte Testanwendung und eine Anwendung zum Absetzen von Steuerbefehlen. Darüberhinaus steht eine Bibliothek mit Konvertierungs- und Filterfunktionen sowie weitere zur API gehörende Bibliotheken zur Verfügung.
In der Studienarbeit wurde nach einer Einführung in die den Videoserver betreffenden Technologien und Standards die Funktionsweise der einzelnen Module des Servers detailiert beschrieben, sowie Angaben über die benutzten Klassen gemacht. Ferner wurde auf die vordefinierten Videoprofile des Servers eingegangen. Diese decken die zur Zeit gängigen Bildgrößen und Bildraten ab. Zum Abschluß des zentralen Kapitels 4 wurde noch die Bedienung der MFC-Testanwendung und der Fernsteuerung erklärt. In der dann folgenden Auswertung wurden zwei Verkehrsaufnahmen untersucht, um Anhaltspunkte für die Wahl geeigneter Kodierparameter zu bekommen. Außerdem wurden die Auswirkungen der Benutzung von Tiefpaßfiltern bei Unterabtastung auf die Kodiereffizienz des Encoders untersucht.
Der Anhang der Studienarbeit enthält schließlich detailierte Beschreibungen der den Videoserver betreffenden Bibliotheken einschließlich der Interfaceklasse. Anwender, welche den Videoserver nutzen wollen, können diese in die eigene Anwendung einbinden. Der Server kann dabei generell zur Kodierung und Übertragung unkomprimierten Videomaterials benutzt werden. Anpassungen des Videostreams an eigene Anforderungen sind möglich. Im Anhang sind weiterhin die vollständigen Untersuchungsergebnisse der Testsequenzen aufgeführt.

6.2 Ausblick

Bisher wurde der Videoserver nur auf switched Fast-Ethernet verwendet. Es soll weiterhin ein Einsatz auf Wireless LAN erfolgen, welches allerdings zu Testzwecken vorerst nur gebäudeintern benutzt wird. Desweiteren wird noch zu untersuchen sein, inwieweit die Übertragungskapazität von WLAN im Außeneinsatz, d.h. im Einsatz an Kreuzungen verschiedener Größe, von Umgebungseinflüssen beeinflußt wird. Auf Grund der ermittelten Erkenntnisse könnte eine Anpassung der Videoprofile notwendig werden.
Und schließlich sei noch auf Sicherheitsaspekte hingewiesen, die bei Überwachungsaufgaben eine wichtige Rolle spielen. Um in diesem Fall einen Mißbrauch des Videoservers zu verhindern, ist eine geeignete Verschlüsselung sowohl der Steuerkommandos als auch der Videodaten selbst erforderlich.

Literaturverzeichnis

[1] William Stallings, ”Data and Computer Communications“, 6th Edition, Prentice Hall, 2000

[2] Jon Postel, “Internet Protocol - DARPA Internet Program Protocol Specification”, RFC791, September 1981

[3] Jon Postel, “Transmission Control Protocol - DARPA Internet Program Protocol Specification ”, RFC793, September 1981

[4] H. Schulzrinne, “Real Time Streaming Protocol (RTSP)“, RFC2326, April 1998

[5] H. Schulzrinne, “RTP : A Transport Protocol for Real-Time Applications“, RFC1889, January 1996

[6] Benno Stabernack, “RTSP Server Implementierung“, Dokumentation, Februar 2002

[7] ISO/IEC JTC1/SC29/WG11, “Overview of the MPEG4-Standard“, 07/98

[8] ISO/IEC JTC1/SC29/WG11, "Information Technology- Generic Coding of Audio-Visual Objects - Part 2 : Visual", Document no. N2502, Atlantic City, October 1998

[9] Benno Stabernack, Christian Stoffers, „Documentation MPEG-4 Video Encoder”, March 2001

[10] P. Noll, “Signale und Systeme”, Berlin 1998

[11] M. Antonini, M. Barlaud, P.Mathieu, I. Daubechies, “Image Coding Using Wavelet Transform”, IEEE Transaction on Image Processing, vol. 1,no. 2, pp 205-220, Apr. 1992

[12] Viktor Toth, Dirk Louis, „Visual C++ 6 Kompendium”, Markt & Technik, 1999

Abbildungsverzeichnis

Anhang A – Dokumentation der Klasse CVideoServer

A. 1 Inhaltsbeschreibung

Die Klasse *CVideoServer* ist das Interface zum Videoserver. Über sie wird das Servermodul initialisiert, gestartet, mit Frames versorgt und wieder gestoppt. Die Klasse wird in einer dynamisch gelinkten Bibliothek bereitgestellt. Zur Benutzung werden folgende Dateien benötigt :

VideoServer.h
VideoServer.lib oder VideoServer_dbg.lib
VideoServer.dll oder VideoServer_dbg.dll

Die Debugversion unterscheidet sich von der Releaseversion im Wesentlichen dadurch, das bei Problemen oder Fehlbedienungen Fehlermeldungen der Form "Assertion failed! ... " in einem Fenster ausgegeben werden, während in der Releaseversion die Funktionen einen Fehlercode zurückgeben. Außerdem ist die Debugversion wie üblich langsamer.

A. 2 Membervariablen und Konstanten

Die Klasse enthält folgende, als *protected* deklarierte Membervariablen :

Datentyp	Bezeichnung	Beschreibung
Bool	m_bInterfaceInit	interne Flagvariable
Bool	m_bServerStarted	interne Flagvariable
Bool	m_bStartAll	Server im Warte- oder Vollmodus starten
UInt32	m_RC_TCPPort	gewünschter Port für die Fernbedienung
UInt32	m_DefaultProfile	gewünschtes Profil für den Start
UserProfile	m_UserProfile	selbstdefiniertes Videoprofil

Die 4 letztgenannten Variablen dienen zur Erfassung von Benutzervorgaben.
Außerdem wird noch die als *public* deklarierte Variable *m_bShowImage* vom Typ *Bool* bereitgestellt. Diese wird vom Videoservermodul gesetzt. Genauer gesagt wird ihr Wert über die Fernsteuerung mittels des Kommandos "SHOWIMAGE" festgelegt. Sie kann die Anwendung über den Wunsch des Clients informieren, ob das Frame serverseitig auf dem Bildschirm angezeigt werden soll oder nicht. Die Serveranwendung kann (muß aber nicht) diese Variable regelmäßig auslesen und darauf reagieren.
UserProfile hat folgende Struktur :

```
struct UserProfile
{
UInt32 Width, Height, Framerate, Bitrate, IFramePeriod;
Bool    ProfileInitialized;
        }
```

Hiermit kann der Benutzer ein eigenes Videoprofil definieren.

A. 3 Memberfunktionen

Die Memberfunktionen sind als *public* deklariert. Der Rückgabewert jeder Funktion ist vom Typ *ErrVal* (Integerwert), welcher in der MSysLib definiert ist und die beiden Werte *m_nOK* und *m_nERR* der Klasse *Err* enthalten kann.

- *ErrVal CVideoServer::InitVideoServer(UInt32 RC_TCPPort, UInt32 DefaultProfile, Bool bStartAll)*

Mit dieser Funktion werden einige für den Start wichtige Werte gesetzt. *RC_TCPPort* enthält den gewünschten Port, an den die Fernsteuerung auf der Clientseite Kommandos absetzen kann. Der Port muß ≥ 30120 sein. Ist der eingestellte Port bereits belegt, so wird erst beim Aufruf der Startfunktion ein Fehler zurückgeliefert.
Der zweite Parameter gibt das Profil vor, welches zuerst verwendet werden soll, bis vom Client ein anderslautendes Kommando kommt. Ist das Profil nicht vorhanden, so kommt es beim Starten ebenfalls zu einer Fehlermeldung.
Der letzte Parameter legt schließlich das Startverhalten fest. Wird für *bStartAll "false"* übergeben, so wird der Videoserver zuerst im Wartemodus gestartet. Er kann schon Kommandos vom Client empfangen, die Module Konverter, Encoder und RTP-Server wurden aber noch nicht gestartet, sodaß noch keine Frames versendet werden können. Im Gegensatz dazu veranlaßt *"true"* den vollständigen Start, sodaß der Client bereits den RTP-Server kontaktieren und Frames empfangen kann. Dieser Modus in Verbindung mit einem voreingestellten Profil ist die einfachste Möglichkeit, den Videoserver zur Übertragung von Livebildern zu benutzen.

- *ErrVal CVideoServer::StartVideoServer()*

Diese Funktion veranlaßt den Start des Servers mit o.g. Einstellungen. *InitVideoServer()* muß also vorher erfolgreich aufgerufen worden sein. Bei erfolgreichem Start wird Err::m_nOK zurückgeliefert, ansonsten *Err::m_nERR*. Wie schon beschrieben, startet der Videoserver im Warte- oder Vollmodus.
Die einzelnen Schritte der Startsequenz werden in einer Datei mitgeschrieben, ebenso die wesentlichen Fehler, falls diese aufgetreten sind.

- *ErrVal CVideoServer::SendFrame(CStreamBuffer* Frame)*

Mit dieser Funktion werden Frames an den Videoserver übergeben. Die Funktion wird dabei mit einem Pointer auf die Klasse *CStreamBuffer* aufgerufen, welche in Anhang B beschrieben wird. Wesentlicher Inhalt dieser Klasse ist ein Pointer auf einen Puffer im Speicher sowie Membervariablen, welche Informationen über den Puffer enthalten (Größe, Typ etc.). Die Klasse muss also vor dem Aufruf mit den entsprechenden Werten gefüllt werden. Da der Videoserver über interne Konvertierungsroutinen verfügt, besteht eine gewisse Freiheit was Größe und Typ der Frames (RGB oder YUV) angeht.

Über die Interfaceklasse und dem Hauptmodul des Videoservers gelangen die Frames zum Konvertermodul. Dort werden sie zum ersten Mal überprüft und dann in einen internen Puffer kopiert. Danach kehrt die Funktion mit *Err::m_nOK* bzw. *Err::m_nERR* bei einem Fehler zurück. Ein Fehler kann z.B. sein, daß das Bild nicht konvertierbar ist oder allgemein ein ungültiges Format besitzt. Weiterhin kann es vorkommen, das der Arbeitsspeicher zum Kopieren nicht mehr ausreicht. Die internen Puffer werden erst beim ersten Framedurchlauf mit der richtigen Größe angelegt. Reicht der Arbeitsspeicher dann nicht aus, so können die ankommenden Frames nicht verarbeitet werden. In der Debugversion wird daraufhin dann die besagte "Assertion..." – Fehlermeldung ausgegeben, während in der Releaseversion die Frames nur verworfen werden. Über ihren Verlust kann sich der Client über ein spezielles Kommando (siehe Anhang D) informieren.

- *ErrVal CVideoServer::StopVideoServer()*

Mit dieser Funktion wird der Videoserver gestoppt. Es werden alle Module gestoppt, zerstört und die internen Puffer gelöscht. Nun erst kann der Videoserver mit neuen Werten initialisiert und erneut gestartet werden. Ein Aufruf der Funktion *InitVideoServer* während des Betriebes ist also nicht zulässig.

- *ErrVal CVideoServer::SetUserDefinedProfile(UInt32 Width, UInt32 Height, UInt32 Framerate , UInt32 Bitrate, UInt32 IFramePeriod)*

Mit dieser Funktion kann der Benutzer ein eigenes Videoprofil anlegen. Dabei werden die Einstellungen des vom Videoservers benutzten Profiles 0 bei dessen Start überschrieben. Die Funktion *SetUserDefinedProfile* muß also vor der Startfunktion aufgerufen werden. In ihr werden die übergebenen Parameter auch überprüft. Zur Aktivierung des eigenen Videoprofiles muß der Parameter 2 der Funktion *InitVideoServer* auf 0 gesetzt werden.

- *ErrVal CVideoServer::EnableCommandEncryption(UInt8[16] Key, Bool bEnable)*

Mit dieser Funktion wird die Entschlüsselung und Verschlüsselung der Kommandos aktiviert oder deaktiviert. Dies sollte nur erfolgen, wenn dies beim Client ebenfalls eingestellt wird. Der übergebene Schlüssel dient im Aktivierungsfall, d.h. *bEnable* = *true*, zur Entschlüsselung aller danach empfangenen Kommandos sowie zur Verschlüsselung der Antworten. Er muß mit dem Schlüssel des Clients übereinstimmen.

- *ErrVal CVideoServer::LockVideoServer(Bool bLock)*

Mit dieser Funktion wird die Entgegennahme von Steuerbefehlen eines Clients erlaubt oder gesperrt.

A. 4 Hinweise zum Umgang

Eine umfangreiche Beschreibung zum Innenleben des Videoservers und dessen Funktionsweise ist in Kapitel 4. der Studienarbeit enthalten.
Aufgrund der genannten Unterschiede zwischen der Debug- und Releaseversion empfiehlt es sich, zuerst die (langsamere) Debugversion in die eigene Anwendung einzubinden, um die korrekte Bedienung und Funktionsweise sicherzustellen. Kommt es dabei zu den beschriebenen Fehlermeldungen, während die Releaseversion scheinbar "sauber" läuft, so liegt das mit hoher Wahrscheinlichkeit an "Bedienungsfehlern" (z.B. falsches Frameformat etc.) .

Nachfolgend noch ein Anwendungsbeispiel für die Benutzung der Klasse *CVideoServer* :

```
#include <MSysLib.h>
#include <VideoServer.h>
#include <StreamBuffer.h>

CStreamBuffer     cStrBuf;
CVideoServer      cServer;

Int Testfunktion()
{
        UInt32 i;

        // Framebuffer initialisieren (Fehlerüberprüfung fehlt hier ausnahmsweise)
        cStrBuf.CreateFrameBuffer(640*480*3);
        cStrBuf.SetBufferHeader(NULL, 640, 480, 640*480*3, 10, FRAMETYPE_RGB24);

        //Videoserver initialisieren (Vollmodus)
        if ( Err::m_nOK != cServer.InitVideoServer( 30120 , 1, true) ) return –1;

        //Videoserver starten
        if ( Err::m_nOK != cServer.StartVideoServer() ) return –1;

        //1000 Frames senden
        for (i = 0; i<1000; i++)
                {
                //Bufferklasse füllen => Frame grabben
                ...

                //Frame senden
                if ( Err::m_nOK != cServer.SendFrame( &cStrBuf ) ) break;
                }

        //Videoserver stoppen
        cServer.StopVideoServer();

        //Framebuffer löschen (Fehlerüberprüfung fehlt hier)
        cStrBuf.DeleteFrameBuffer();
        cStrBuf.ResetBufferHeader();

return 0;
}
```

Anhang B – Dokumentation der Klasse CStreamBuffer

B. 1 Inhaltsbeschreibung

Die Klasse *CStreamBuffer* dient dem Austausch von Daten zwischen den Modulen Konverter, Encoder und RTP-Server. Sie enthält einen Datenpuffer für Frames bzw. Streams, Headerinformationen über den Inhalt, sowie Funktionen zum Umgang mit diesem Puffer. Sie wird aber nicht nur zum Austausch zwischen den Modulen, sondern auch modulintern benutzt, z.B. bei Framekonvertierungen. Die Notwendigkeit dieser Dokumentation ergibt sich aber aus der Tatsache, daß diese Klasse auch zum Interface der Server-Library gehört. Sie wird benutzt, um dem Videoserver Frames zu übergeben, die er dann codieren und versenden soll. Aus diesen Grund wurde die Klasse in eine statisch gelinkte Bibliothek gebunden, welche mit der Headerdatei in eigene Programme eingebunden werden kann. Die Dateinamen sind “StreamBuffer.h” und “StreamBuffer.lib” bzw. “StreamBuffer_dbg.lib” für die Debugversion.

B. 2 Membervariablen und Konstanten

Die Klasse enthält folgende, als *public* deklarierte Membervariablen :

Datentyp	Bezeichnung	Beschreibung
UInt8*	m_FrameBuffer	Pointer auf den Datenpuffer
UInt32	m_FrameWidth	Breite des Bildes in Pixeln
UInt32	m_FrameHeight	Höhe des Bildes in Pixeln
UInt32	m_BufferLenght	Länge des aktuellen Puffers in Bytes
UInt32	m_Timestamp	Timestampinformation (optional)
UInt32	m_Framerate	Bildrate des Videostreams in Hz
UInt32	m_UserDefined	eigene Information
std::string	m_FrameType	Typ des Pufferinhalts (string)

Für den Inhalt des Puffers sind folgende Typen vorgesehen (m_FrameType) :

Bezeichnung (#define)	Inhalt (string)	Beschreibung
FRAMETYPE_YUV420	"YUV420"	YYYYYYYY...UU...VV...
FRAMETYPE_YUV422	"YUV422"	YYYYYYYY...UUUU...VVVV...
FRAMETYPE_YUV444	"YUV444"	YYYY...UUUU...VVVV...
FRAMETYPE_RGB24P	"RGB24PLANAR"	RRRR...GGGG...BBBB...
FRAMETYPE_RGB24I	"RGB24INTERLEAVED"	RGBRGBRGBRGB...
FRAMETYPE_MPEG4	"MPEG4STREAM"	video object plane
FRAMETYPE_IOD	"MPEG4IOD"	initial object descriptor

Der integrierte MPEG4-Encoder arbeitet mit YUV420-Frames. Aus diesem Grunde wurden diverse Konvertierungsroutinen implementiert, welche eine Typenkonvertierung, z.B. YUV444 -> YUV420 oder RGB24 -> YUV420, vornehmen (im Konverter-Modul).
Für die Typenkonstanten wurden Strings und keine Zahlen verwendet, um eine schnellere und klare Ausgabe dieser Information (für Testzwecke etc.) zu ermöglichen.

Die weiteren Konstanten sind :

Bezeichnung (#define)	Inhalt (UInt)	Beschreibung
FRAME_MAXSIZE	1280*960*3	maximale Größe eines Datenpuffers (Bytes)
MPEG4_MAXSIZE	256000	max. Größe eines kodierten Frames (Bytes)
FRAMERATE_MAX	60	max. Framerate (Hz)
IOD_MAXSIZE	100	max. Größe des IODs (Bytes)

Die hier definierte maximale Puffergröße bezieht sich auf ein RGB24- oder YUV444-Frame.

B.3 Memberfunktionen

Die Memberfunktionen sind alle als *public* deklariert. Sie ermöglichen den Umgang mit dem Puffer. Dabei wurde der Umgang mit dem Header von dem eigentlichen Datenpuffer getrennt, um dem Anwender die Möglichkeit zu geben, einen bereits anderweitig erstellten Puffer weiter zu verwenden statt einen neuen anzulegen.
Der Rückgabewert jeder Funktion ist vom Typ *ErrVal*, welcher in der MSysLib definiert ist und die beiden Werte *m_nOK* und *m_nERR* der Klasse *Err* enthalten kann.

- *ErrVal CStreamBuffer::CreateFrameBuffer(UInt32 BufferSize)*

  Diese Funktion erstellt einen Puffer der Größe *BufferSize.* Die Membervariable *m_Framebuffer* enthält dann diesen Puffer und *m_BufferLenght* enthält die Länge.

- *ErrVal CStreamBuffer::DeleteFrameBuffer()*

  Diese Funktion löscht einen Datenpuffer aus dem Speicher, setzt *m_Framebuffer* zu NULL und *m_BufferLenght* zu 0. Alle anderen Headerinformationen werden nicht verändert.

- *ErrVal CStreamBuffer::SetBufferHeader(UInt8* BufferAddress,UInt32 Width, UInt32 Height, UInt32 Lenght, UInt32 Framerate, std::string Type)*

  Diese Funktion füllt den Header mit Informationen. Wird bzw. wurde der Datenpuffer mit *CreateFrameBuffer* erzeugt, so ist für *BufferAddress* NULL anzugeben, ansonsten wird hier der Pointer auf einen bereits anderweitig erzeugten Puffer übergeben. Dies kann vorallem dann genutzt werden, wenn der Speicherplatz begrenzt ist. Als *Type* können die unter B.2 aufgeführten Datentypen angegeben werden.
  Bei der Verwendung von MPEG4-Daten (Stream oder IOD) ist die Angabe der Bildbreite *Width* und Bildhöhe *Height* optional, die Angabe der Bildrate *Framerate* ist für die Benutzung des Videoservers jedoch zwingend vorgeschrieben.

- *ErrVal CStreamBuffer::ResetBufferHeader()*

Diese Funktion setzt alle Headerinformationen zurück, z.B. auf 0, sodaß der Puffernicht mehr sinnvoll verwendet werden kann. Der Puffer selbst wird aber nicht gelöscht! Dies erfolgt über die o.g. Funktion *DeleteFrameBuffer()*.

- *ErrVal CStreamBuffer::CompareBufferHeader(UInt32 Width, UInt32 Height, UInt32 Length, UInt32 Framerate, std::string Type)*

Diese Funktion vergleicht den Headerinhalt mit den übergebenen Parametern. Bei Gleichheit wird *Err::m_nOK* zurückgegeben sonst *Err::m_nERR.*

- *ErrVal CStreamBuffer::CopyFrameBuffer(CStreamBuffer cSourceBuf, UInt8 AdjustLenght)*

Diese Funktion kopiert den Inhalt des übergebenen Puffers sowie die zugehörigen Headerinformationen. Ist der Zieldatenpuffer noch nicht angelegt, so wird es nun getan. Die Funktion kann somit auch einfach als Vervielfältiger genutzt werden. Ist der Zieldatenpuffer jedoch schon angelegt, so wird seine Größe überprüft. Stimmt sie nicht mit der Quelle überein, so wird der Zielpuffer gelöscht und neu angelegt. Letzteres geschieht aber in Abhängigkeit vom Parameter *AdjustLenght* :

0 - Zielpuffer nicht neu anlegen (wenn Puffer verschieden lang -> Fehler)
1 - neu anlegen wenn Zielpuffer > QuellBuffer
2 - neu anlegen wenn Zielpuffer < QuellBuffer
3 - neu anlegen wenn Zielpuffer ≠ QuellBuffer

B. 4 Hinweise zum Umgang

Bei der Benutzung der Memberfunktionen ist zu beachten, das sie die übergebenen Parameter auf sinnvollen Inhalt hin überprüfen. Dazu gehören vorallem Überprüfungen hinsichtlich überschrittener Grenzen (siehe B.2) und der erlaubten Typen.
So kann z.B. *CreateFrameBuffer* keine Datenpuffer größer als *FRAME_MAXSIZE* erstellen. Besonders wichtig ist auch die Begrenzung der Bildbreite und Bildhöhe auf Vielfache der Makroblockgröße 16 !
Diese Beschränkungen können umgangen werden, indem die Membervariablen direkt gesetzt werden. Für die Benutzung des Videoservers macht dies in den meisten Fällen jedoch keinen Sinn.

Nachfolgend noch ein Anwendungsbeispiel für die Benutzung der Klasse *CStreamBuffer* :

```
#include <MSysLib.h>
#include <StreamBuffer.h>

CStreamBuffer      cBuffer,
                   cBufferCopy;

Int Testfunktion()
{
//Puffer der Größe 640x480x1.5 erzeugen
if ( Err::m_nOK != cBuffer.CreateFrameBuffer(460800) ) return –1;

//Headerinformationen setzen (Adresse behalten da Puffer schon erzeugt)
if (Err::m_nOK != cBuffer.SetBufferHeader( NULL, 640, 480, 460800, 10, FRAMETYPE_YUV420)
        {
        cBuffer.DeleteFrameBuffer();
        return –1;
        }
//Puffer mit einer Farbe füllen
memset(cBuffer.m_FrameBuffer, 128, 460800);

//1:1 Kopie erstellen
if ( Err::m_nOK != cBufferCopy.CopyFrameBuffer( cBuffer, 3 )
        {
        cBuffer.DeleteFrameBuffer();
        return –1;
        }
//Puffer löschen und Headerinformationen auf 0 setzen
cBuffer.DeleteFrameBuffer();
cBuffer.ResetBufferHeader();

//Kopie verwenden
...
//Kopie löschen
cBufferCopy.DeleteFrameBuffer();
cBufferCopy.ResetBufferHeader();

return 0;
}
```

Anhang C – Dokumentation der Klasse CCommandParser

C. 1 Inhaltsbeschreibung

Die Klasse *CCommandParser* enthält einige Funktionen, welche die Zusammensetzung und Zerlegung der über eine TCP-Verbindung übertragenen Kommandos ermöglichen. Die Klasse wird in einer statisch gelinkten Bibliothek bereitgestellt mit den Dateibezeichnungen “CommandParser.h“ und “CommandParser.lib“ bzw. “CommandParser_dbg.lib“ und kann in der serverseitigen und clientseitigen Anwendung verwendet werden.
Anhang D enthält eine Übersicht über alle Kommandos, welche derzeit im Videoserver unterstützt werden. Die Kommandos bestehen aus US-ASCII-Strings, welche eine Nachricht sowie zwei Parameter im Klartext enthalten. Der über den Kanal übertragene String kann dabei nicht nur das Kommando, sondern auch andere Zeichen davor oder danach enthalten. Dies hat zurzeit noch keine weitere Bedeutung, kann aber z.B. später für Verschlüsselungs-algorithmen interessant werden. Terminiert wird der String mit den Zeichen ’\n’ oder ’\0’. Die maximale Länge beträgt COMMAND_MAXSIZE (siehe C. 2). Gesendet und empfangen wird der String mit den zugehörigen Socket-Funktionen (z.B. *send(), recv()*).
Das in dem String enthaltene Kommando hat die Form: “Nachricht|Parameter1|Parameter2“ und muß mit eckigen Klammern abgegrenzt sein : “[Nachricht|Parameter1|Parameter2]“ .
Als Parameter können ausschließlich 32-Bit unsigned integer übertragen werden.

C. 2 Membervariablen und Konstanten

Die Klasse *CCommandParser* enhält nur die Konstante COMMAND_MAXSIZE (256 Bytes), welche die maximale Länge des Strings angibt, was auch gleichzeitig der maximalen Länge des Kommandos entspricht.

C. 3 Memberfunktionen

Die Memberfunktionen sind als *public* deklariert. Der Rückgabewert jeder Funktion ist vom Typ *ErrVal* (Integerwert), welcher in der MSysLib definiert ist und die beiden Werte *m_nOK* und *m_nERR* der Klasse *Err* enthalten kann.

- *ErrVal CCommandParser::GetCommand(std::string recvbuffer, std::string* command, UInt32 SearchRange)*

  Diese Funktion sucht aus einem String der maximalen Länge COMMAND_MAXSIZE ein Kommando mit o.g. Format heraus. In *recvbuffer* wird der String übergeben, *command* erhält dann das Ergebnis, wenn ein korrektes gefunden wurde. *SearchRange* enthält die Reichweite in der im String gesucht werden soll. Sie ist üblicherweise gleich der Länge des Strings.
  Zu beachten ist, das nach eckigen Klammern gesucht wird, d.h. sie dürfen im String nur einmal und in der richtigen Reihenfolge auftreten. Das Ergebnis ist von diesen Klammern jedoch befreit und hat die Form : “Nachricht|Parameter1|Parameter2“

- ErrVal CCommandParser::GetMessage(std::string command, std::string message)*

Wurde das Kommando mit *GetCommand* aus einem beliebigen String extrahiert, so kann nun der erste Teil, die Nachricht, herausgefiltert werden. Dies ist der String bis zum ersten Trennzeichen. Er wird in *message* zurückgeliefert.

*- ErrVal CCommandParser::GetParams(std::string command, UInt32 *P1, UInt32 *P2)*

Der nächste Schritt ist die Extraktion der beiden Integer-Parameter. Sie liegen als Strings vor, müssen also herausgelöst und in Zahlen umgewandelt werden. Das Ergebnis wird in *P1* und *P2* zurückgeliefert.

- ErrVal CCommandParser::CreateCommand(std::string message, UInt32 Param1, UInt32 Param2, std::string command)*

Im Gegensatz zu den anderen Funktionen, welche die Zerlegung zum Inhalt hatten, wird diese Funktion benutzt, um aus den drei Bestandteilen ein versandfähiges Kommando zusammenzusetzen. Übergeben werden die drei Teile *message*, *Param1* und *Param2*, das fertige Kommando wird in *command* zurückgeliefert und besitzt als letztes Zeichen '\n'.

C. 4 Hinweise zum Umgang

Nachfolgend ein Anwendungsbeispiel für die Benutzung der Klasse *CCommandParser* :

```
#include <MSysLib.h>
#include <CommandParser.h>

CCommandParser cParse;
std::string     Kommandostring,
                Kommando,
                Message;
UInt32          Parameter1,
                Parameter2;

Int Testfunktion()
{
//Kommando erzeugen :Kommandostring ->  "[STARTALL|0|0]"
if ( Err::m_nOK != cParse.CreateCommand( "STARTALL", 0, 0, &Kommandostring ) )
        return –1;

//Kommando zerlegen :Kommando -> "STARTALL|0|0"
if ( Err::m_nOK !=
     cParse.GetCommand( Kommandostring, &Kommando, Kommandostring.length() ) ) return –1;

//Nachricht separieren : Message->"STARTALL"
if ( Err::m_nOK != cParse.GetMessage( Kommando, &Message) ) return –1;

//Parameter ermitteln : Parameter1 -> 0 , Parameter2 -> 0
if ( Err::m_nOK != cParse.GetParams( Kommando, &Parameter1, &Parameter2) ) return –1;

return 0;
}
```

Kommando (String)	P1	P2	Timeout	Kurzbeschreibung	Antwort vom Server (Parameter 1 und 2)
STARTALL	0	0	3000	startet Server im Vollmodus	P1 : 1/0 (Ack / Nack)
STOPALL	0	0	3000	versetzt Server in den Wartemodus	P1 : 1/0 (Ack / Nack)
STARTCONVERTER	0	0	1000	startet den Konverter (für Testzwecke)	P1 : 1/0 (Ack / Nack)
STARTENCODER	0	0	1000	startet den Encoder (für Testzwecke)	P1 : 1/0 (Ack / Nack)
STARTSERVER	0	0	1000	startet den RTP-Server (für Testzwecke)	P1 : 1/0 (Ack / Nack), P2 : aktueller TCP-Port
STOPCONVERTER	0	0	1000	stoppt den Konverter	P1 : 1/0 (Ack / Nack=Konverter läuft gar nicht)
STOPENCODER	0	0	1000	stoppt den Encoder	P1 : 1/0 (Ack / Nack=Encoder läuft gar nicht)
STOPSERVER	0	0	1000	stoppt den RTP-Server	P1 : 1/0 (Ack / Nack=RTP-Server läuft nicht)
SETPROFILE	0..16	0	3000	stellt das aktuelle Profil ein, ggfs. Restart	P1 : 1/0 (Ack / Nack)
GETPROFILE	0	0	1000	ermittelt aktuelles Profil vom Server	P2 : 0..16
SETACTUALBITRATE	50..3000	0	1000	ändert die Bitrate des aktuellen Profiles (temporär)	P1 : 1/0 (Ack / Nack)
GETACTUALBITRATE	0	0	1000	ermittelt die akuelle Bitrate	P2 : aktuelle Bitrate in [kbits/s]
SETFRAMESIZE	Breite	Höhe	1000	stellt Bildgröße beim Profil 0 ein	P1 : 1/0 (Ack / Nack= 0 ist aktuelles Profil)
SETFRAMERATE	1...30	0	1000	stellt Bildrate beim Profil 0 ein	P1 : 1/0 (Ack / Nack= 0 ist aktuelles Profil)
SETBITRATE	[kBit/s]	0	1000	stellt Bitrate beim Profil 0 ein	P1 : 1/0 (Ack / Nack= 0 ist aktuelles Profil)
SETIFRAMEPERIOD	1...30	0	1000	stellt Periode der I-Frames beim Profil 0 ein	P1 : 1/0 (Ack / Nack= 0 ist aktuelles Profil)
GETFRAMESIZE	0	0	1000	ermittelt eingestellte Bildgröße beim Profil 0	P2 : HIWORD=Breite / LOWORD=Höhe
GETFRAMERATE	0	0	1000	ermittelt eingestellte Bildrate beim Profil 0	P2 : Framerate [Hz]
GETBITRATE	0	0	1000	ermittelt eingestellte Bitrate beim Profil 0	P2 : Bitrate [kBit/s]
GETIFRAMEPERIOD	0	0	1000	ermittelt eingestellte I-Frame-Periode	P2 : Intraframe Periode [Frames]
ACTIVATEFILTER	0..3	0	1000	(0) kein TP, (1) Summen-, (2) 3-Tap, (3) 7-Tap-Filter	P1 : 1/0 (Ack / Nack)
GETRTSPPORT	0	0	1000	ermittelt TCP-Port des RTSP-Servers	P1 : 1/0 (Ack or Nack), P2 : TCP-Port
GETLOSTFRAMES	0	0	1000	ermittelt Anzahl an übersprungenen Frames	P2 : Anzahl verlorener Frames
SHOWFRAME	1/0	0	1000	informiert Serveranwendung (setzt Flagvariable)	P1 : 1/0 (Ack or Nack)
QUITPROGRAM	0	0	1000	beendet Anwendung/Prozess (Kill !)	P1 : 1/0 (Ack or Nack=wird nicht benutzt)

Anhang E – Übersicht Videoprofile

Profil	Bildbreite	Bildhöhe	Framerate[Hz]	Bitrate [kbits/s]	IFrame-Periode
0	Benutzerdefiniert				
1	640	480	15	1200	7
2	640	480	5	600	3
3	640	480	15	1000	15
4	640	480	5	500	5
5	320	240	15	500	7
6	320	240	5	250	3
7	320	240	15	400	15
8	320	240	5	200	5
9	352	288	25	800	12
10	352	288	10	400	5
11	176	144	25	200	12
12	176	144	10	100	5
13	704	576	25	2000	12
14	720	576	25	2000	12
15	512	384	5	500	5
16	1024	768	5	2000	5

Anhang F - Auswertungen

F.1 Rate-Distortion-Funktionen für verschiedene Bildparameter

Abb. F.1 : Testsequenzen ErnstReuterPlatz Ansicht A und B, 500 Frames 640x480, YUV420

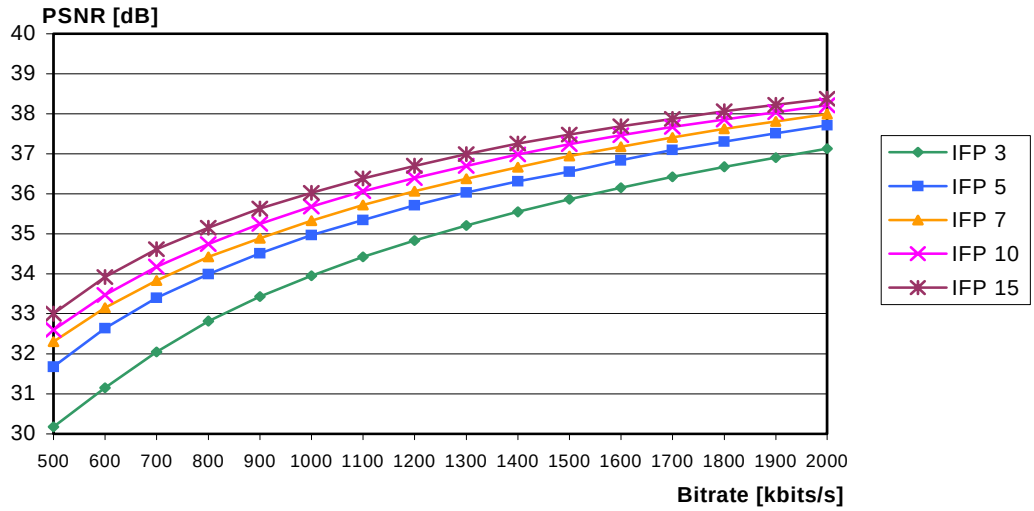


Abb. F.2 : RDF für Testsequenz ErnstReuterPlatz A, @15 Hz

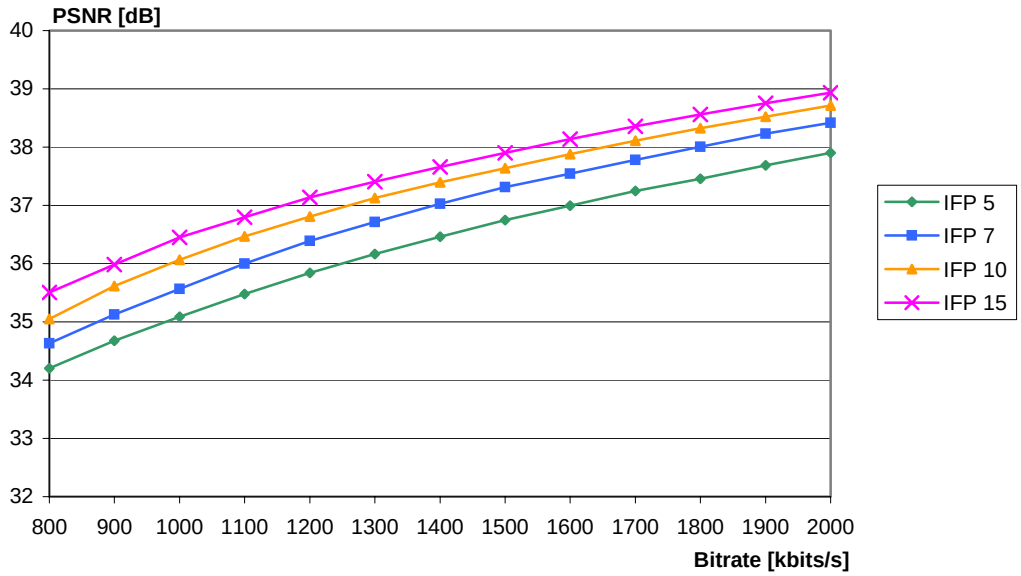


Abb. F.3 : RDF für Testsequenz ErnstReuterPlatz B, @15 Hz

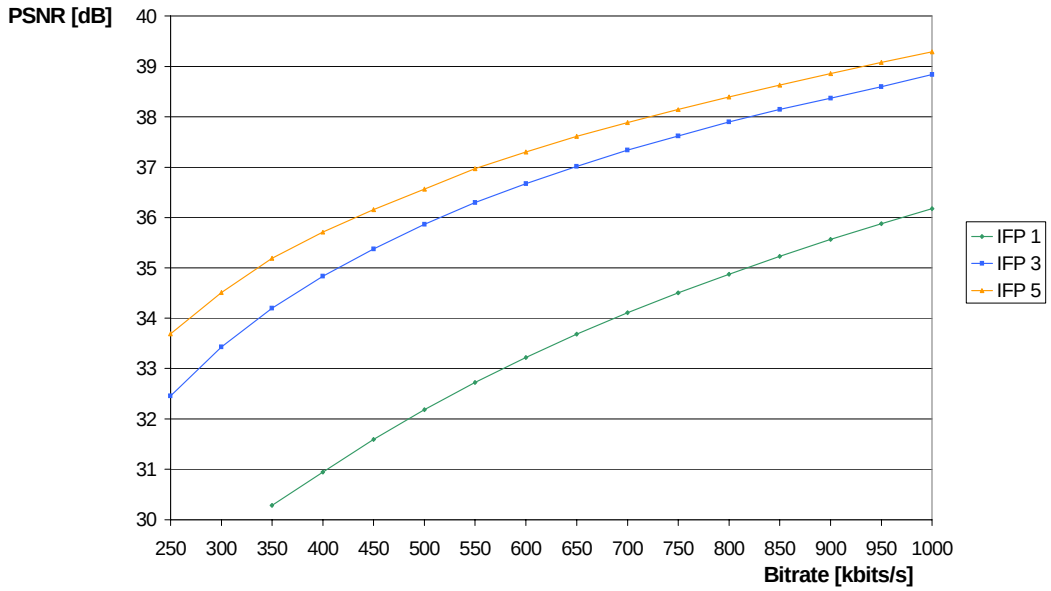


Abb. F.4 : RDF für Testsequenz ErnstReuterPlatz A @5 Hz

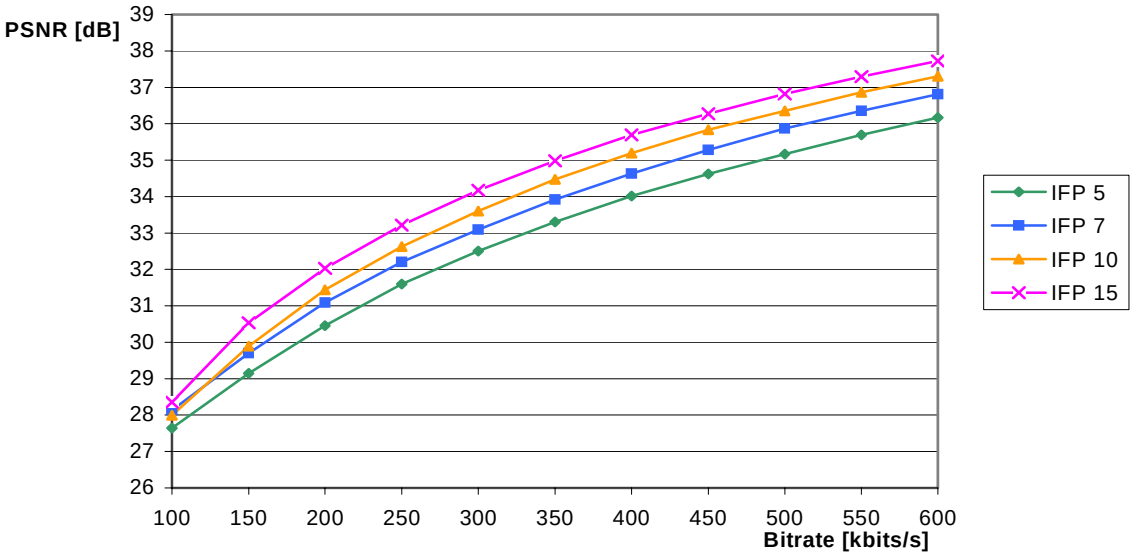


Abb. F.5 : RDF für Testsequenz ErnstReuterPlatz A, 15 Hz, 320x240, vorgefiltert mit 7-Tap

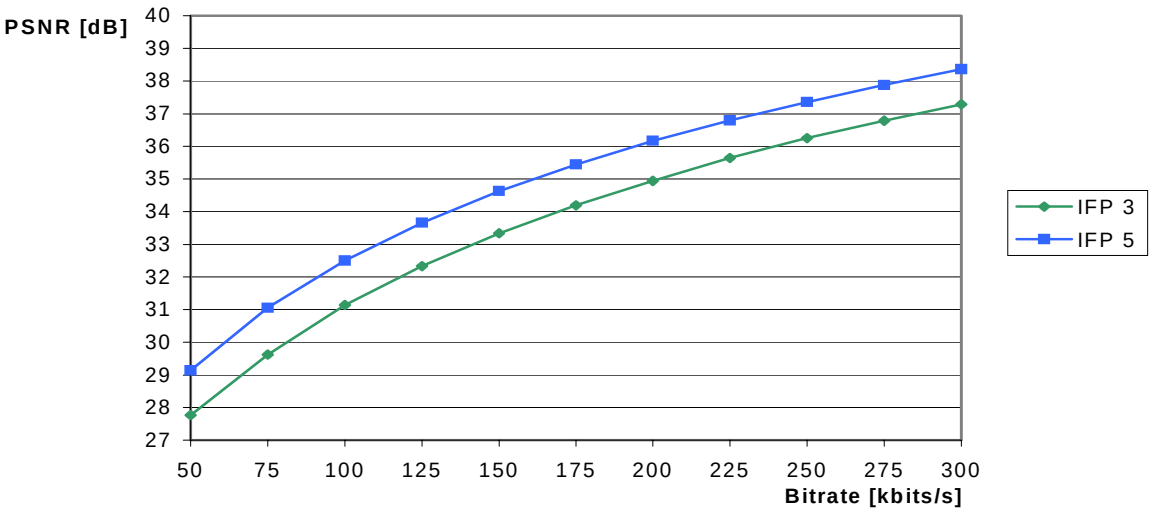


Abb. F.6 : RDF für Testsequenz ErnstReuterPlatz A, 5 Hz, 320x240, vorgefiltert mit 7-Tap

„Entwicklung von Verfahren zur Segmentierung, Verfolgung und 3D-Rekonstruktion von bewegten Objekten in Verkehrsszenen mit mehreren Ansichten“

Diplomarbeit

Von
Michael Dröse
Matrikelnummer: 182574

Hochschullehrer:	Prof. Dr.-Ing. Thomas Sikora
Betreuer (TUB):	Dipl.-Ing. Sila Ekmekci
Betreuer (HHI):	Dr.-Ing. Aljoscha Smolić

Institut für Telekommunikationssysteme

Fachgebiet Nachrichtenübertragung (Elektrotechnik)
Technische Universität Berlin, Januar 2003

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Um eine qualitative Aussage über das Verkehrsaufkommen und das Verhalten der Verkehrsteilnehmer treffen zu können, werden elektronische Systeme benötigt, die den Verkehr analysieren und auswerten. Einfache Systeme, wie induktionsbasierte Systeme, registrieren nur die Anzahl der Fahrzeuge pro Zeit in einem vorher festgelegten Bereich. Für eine Analyse reicht das in den meisten Fällen nicht aus. Aus diesem Grund wird in dieser Diplomarbeit ein visuelles System zur Beobachtung von Verkehrsszenen vorgestellt. Ein solches System könnte beispielsweise Verkehrsteilnehmer in Lkw, Pkw, Motorrad, Fahrrad oder Fußgänger klassifizieren und das Verhalten der Verkehrsteilnehmer analysieren (z.B. Spurwechsel). In Grenzen könnte sogar der Typ des Fahrzeugs ermittelt werden (Limousine, Kleinbus,...). Ein visuelles System bietet zudem noch die Möglichkeit einer 3D Rekonstruktion der Szene. Dem Benutzer wird somit die Möglichkeit gegeben in einer animierten 3D Szene frei zu navigieren. Die Eingabe der Daten in das System erfolgt über mehrere Kameras. Das System wurde im Rahmen eines Projekts des Heinrich-Hertz-Instituts Berlin entwickelt und implementiert.

1.2 Grundprinzip

1.2.1 Übersicht

Eine Visualisierung einer Verkehrszene lässt sich in zwei getrennte Arbeitspunkte zerlegen. Der erste Teil beschäftigt sich mit der 3D Rekonstruktion und Animation der dynamischen Objekte, d.h. den Verkehrsteilnehmern. Der zweite Teil behandelt die Modellierung der statischen Objekte Fahrbahn, Häuser, usw. Die hier vorliegende Diplomarbeit bearbeitet den ersten Teil. Dazu gehören die Untersuchung und Entwicklung von Verfahren zur Objektextraktion, Objektverfolgung und 3D Rekonstruktion.
Die Arbeit lässt sich in vier getrennte Arbeitspunkte gliedern, die aufeinander aufbauen. Der erste Teil behandelt die Extraktion der Vordergrundobjekte aus dem Bildmaterial. Der zweite Teil beschäftigt sich mit der Objektverfolgung in den einzelnen Kameraansichten. Der dritte Teil behandelt die Zuordnung von Objekten aus den einzelnen Kameraansichten zu globalen Objekten und berechnet für jedes globale Objekt die 3D Position. Wichtig ist hier vor allem, dass die Videodaten aus den einzelnen Kameraansichten synchron verarbeitet werden. Die Bewegungsparameter werden wieder von einem linearen Kalman-Filter modelliert. Und schließlich der letzte Teil, die 3D Rekonstruktion und Animierung der Objekte. Aufgrund der geringen Anzahl von Kameras, ist es unmöglich generische Ansätze, wie Disparitätenschätzung, voxel-basierte Verfahren oder light-field Verfahren, zu verwenden.

Die Informationen zur 3D Rekonstruktion ist bei einem solchen Aufbau viel geringer. Um trotzdem eine 3D Rekonstruktion durchführen zu können, muss diese Informationslücke geschlossen werden. Das hier entwickelte Verfahren benutzt dazu eine Datenbasis aus 3D Modellen. Abb. 1 zeigt den Aufbau des Gesamtsystems um einen zu observierenden Verkehrspunkt.

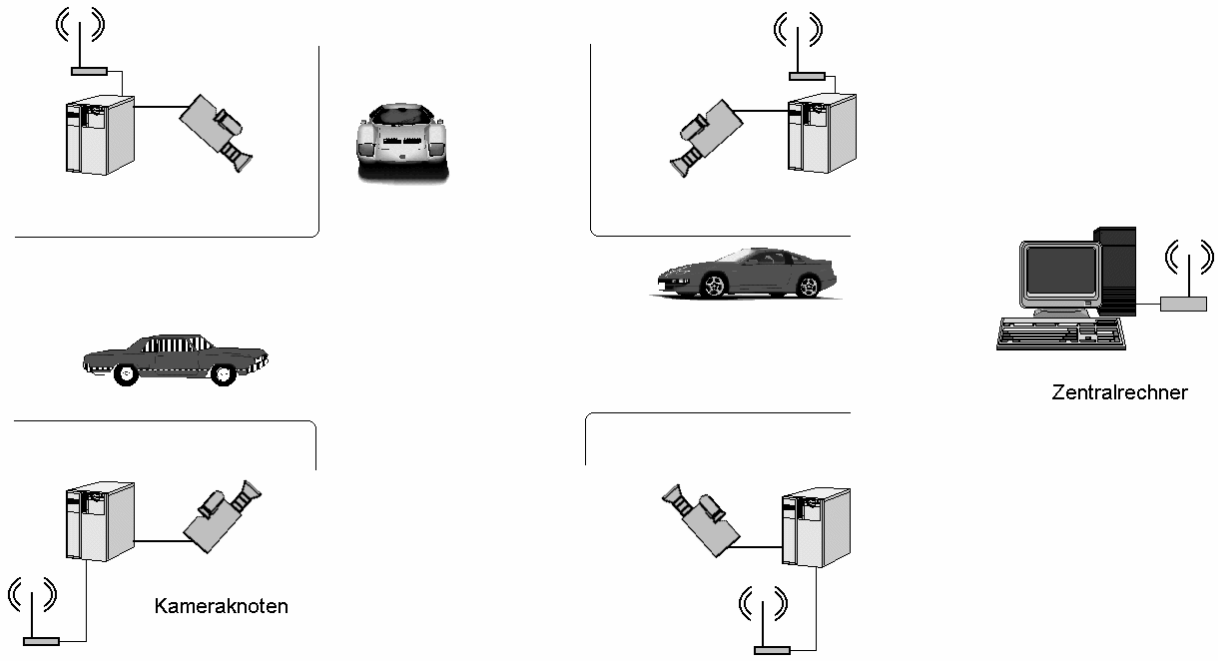


Abb. 1: Aufbau des Gesamtsystems

Die Kameras sind konvex um den zu observierenden Verkehrspunkt angeordnet. Alle Kameraparameter, sowohl intrinsische als auch extrinsische, sind vor der Inbetriebnahme des Systems bekannt und ändern sich während des Betriebs nicht. Die Kameras sind folglich vollkalibriert.
An den Kameraknoten wird eine Objektsegmentierung, Objektverfolgung und eine Kodierung der Bilder nach MPEG-4 durchgeführt. Der Zentralrechner dekodiert den Videostream, vereint die Objektverfolgung der einzelnen Kameraknoten und generiert daraus eine 3D Szene. Die Übermittlung der Daten, das beinhaltet den Videostream und die verfolgten Objekte von den Kameraknoten zum Zentralrechner erfolgt über Wireless LAN. Die Verwendung von B-Splines hat zudem den Vorteil, dass die Datenmenge, die zum Zentralrechner übertragen wird, je nach Auslastung variiert werden kann.

1.2.2 Aufbau der Implementierung

In den einzelnen Kameraansichten wird zuerst eine Objektsegmentierung und eine Objektverfolgung in 2D durchgeführt (siehe Abb. 2). Die Objektsegmentierung entspricht einem

Hintergrundschätzverfahren. Dieses Verfahren berücksichtigt Beleuchtungseinflüsse. Die extrahierten Regionen werden konnektiert und zu konvexen Konturen gewandelt. Für die Objektverfolgung in 2D werden die konvexen Konturen durch B-Splines approximiert. Zwei lineare Kalman-Filter modellieren die Bewegungsparameter und die Form der Kontur. Die verfolgten Objekte aus den einzelnen Ansichten werden im Anschluss globalen Objekten zugeordnet. Die Zuordnung erfolgt über die Position der Objekte in den Kamerabildern und der zuvor bestimmten Homographien zwischen den Kameraansichten.

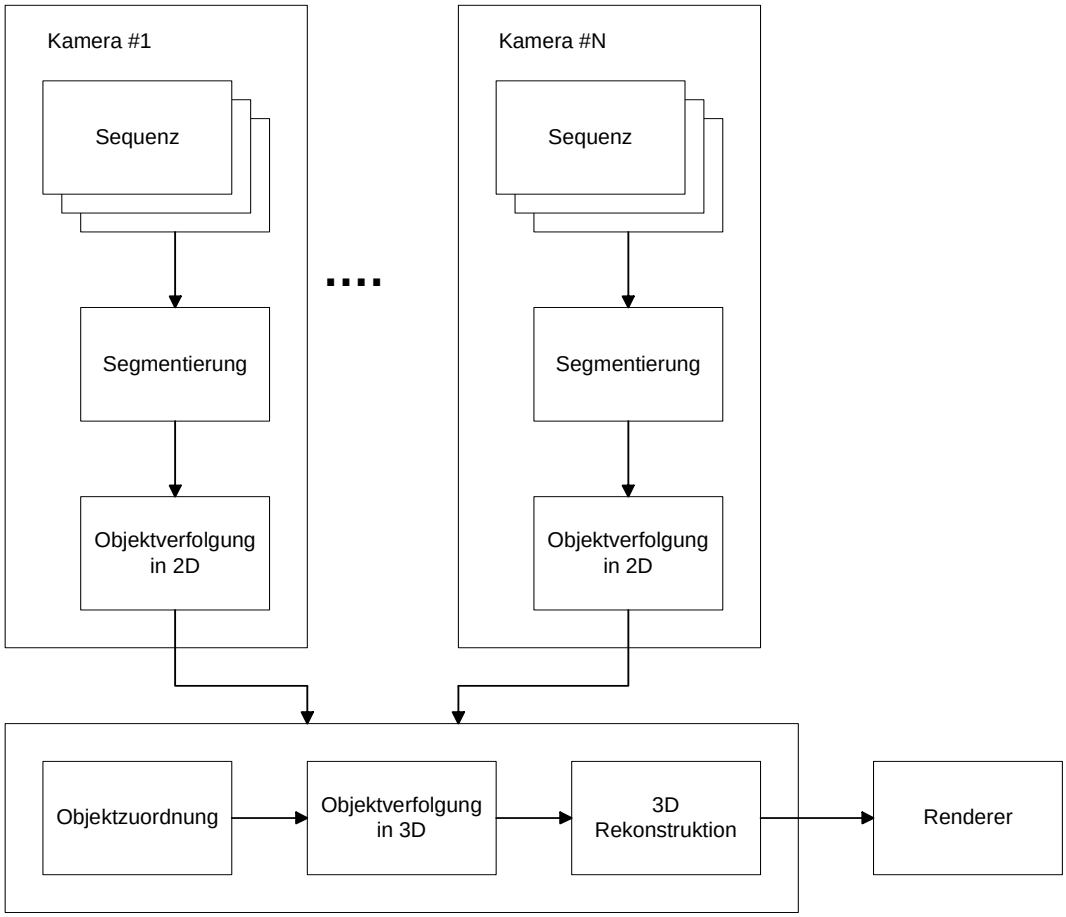


Abb. 2: Blockdiagramm der Implementierung

Aus der Zuordnung, den Konturen in den einzelnen Ansichten und den Kameramatrizen lassen sich die 3D Positionen der globalen Objekte berechnen. Die Berechnung erfolgt über die Zentren der Konturen in den einzelnen Ansichten und einem „Least-Square“-Verfahren. Die Bewegungsparameter werden wieder mit einem linearen Kalman-Filter modelliert. Für die Visualisierung der Verkehrsszene werden die Texturen aus den einzelnen Ansichten extrahiert. Die 3D Rekonstruktion besteht aus Positionierung und Orientierung des 3D Modells, sowie der Rückprojektion der extrahierten Texturen auf das 3D Modell. Die Animation erfolgt durch Aktualisierung der 3D Position und der Orientierung des 3D Modells. Schließlich wird die Szene über den Renderer visualisiert.

1.3 Aufbau der Arbeit

Die Diplomarbeit ist so aufgebaut, dass zuerst der Stand der Technik in Kapitel 2 vorgestellt wird. Kapitel 3 erläutert die theoretischen Grundlagen und die verwendeten Verfahren dieser Diplomarbeit. In Kapitel 4 werden die implementierten Verfahren mit Bezug auf die Theorie beschrieben. Kapitel 5 beschäftigt sich mit der Analyse der Implementierung anhand von künstlichen Daten und realen Verkehrssequenzen. Und schließlich Kapitel 6 mit der Zusammenfassung der Ergebnisse und dem Ausblick auf Ergänzungen zu den bisherigen Arbeiten.

2 Stand der Technik

In dem folgenden Kapitel werden Arbeiten zur Objektextraktion, Objektverfolgung und 3D Rekonstruktion vorgestellt, die dem aktuellen Stand der Technik entsprechen. Die Verfahren werden bewertet und klassifiziert nach ihren Einsatzgebieten. Abschnitt 2.1 stellt zwei prinzipielle Verfahren der Objektextraktion vor, die in der Objektverfolgung verwendet werden. In Abschnitt 2.2 werden Arbeiten auf dem Gebiet der Objektverfolgung in 2D vorgestellt. Abschnitt 2.3 stellt Arbeiten zur Objektverfolgung in mehreren Kameras mit gemeinsamem Blickfeld vor. Der letzte Abschnitt 2.4 dieses Kapitels erläutert den Stand der Technik auf dem Gebiet der 3D Rekonstruktion.

2.1 Segmentierung

Unter Segmentierung versteht man die Extraktion von Regionen aus einem Bild. Eine Region beschreibt eine Fläche im Bild. Aus ihr lassen sich geometrische Merkmale wie Schwerpunkt und Orientierung bestimmen. Zudem können aus der Kombination von Regionen mit Bildern Farb- bzw. Texturinformationen gewonnen werden. Der folgende Abschnitt stellt zwei prinzipielle Verfahren der Segmentierung vor, die in der Objektverfolgung eingesetzt werden. Das erste Verfahren wird dazu verwendet, ausschließlich bewegte Objekte zu extrahieren. Dieses Verfahren wird in der Literatur auch als „Segmentation by Motion“ bezeichnet. Das zweite Verfahren hingegen extrahiert Vordergrundobjekte von einem bekannten Hintergrund. Die Betonung liegt hier auf Vordergrund. Die Methode des optischen Flusses wird in [17] erläutert. Die Ergebnisse sind dem „Segmentation by Motion“-Ansatz ähnlich - allerdings ist der Rechenaufwand höher.

2.1.1 Segmentierung bewegter Objekte (Segmentation by Motion)

Ein einfaches Verfahren, dass in [13] verwendet wird, ist die Methode der Differenzbildung. Jedes Pixel des aktuellen Bildes wird von dem Vorgänger subtrahiert. Das erhaltene Bild wird mit einem Schwellwertverfahren in eine Binärmaske überführt. Eine Konnektierung der Maske liefert die Vordergrund- bzw. Hintergrundregionen. Die folgenden Abbildungen 3 und 4 veranschaulichen das Prinzip dieses Verfahrens. In Abb. 3 sind zwei aufeinanderfolgende Bilder der Sequenz „Ernst-Reuter-Platz“ zu sehen. Die Differenz der beiden Bilder ist in dem linken Bild von Abb. 4 zu sehen. Im rechten Bild der Abb. 4 ist schließlich das Endergebnis nach Anwendung eines Schwellwertverfahrens und der Konnektierung zu sehen. Eine Konnektierung vereint benachbarte Regionen zu einer Region.

Abb. 3: Zwei aufeinanderfolgende Bilder der Sequenz „Ernst-Reuter-Platz“

Abb. 4: Differenzbild (*links*), Schwellwertverfahren und Konnektierung (*rechts*)

Da immer zwei aufeinanderfolgende Bilder subtrahiert werden, ist der Einfluss der Beleuchtung gering. Der Nachteil dieses Verfahrens liegt in der Beschränkung auf bewegte Objekte. Für den Einsatz zur Verkehrsbeobachtung ist es deshalb nur in Spezialfällen sinnvoll.

2.1.2 Segmentierung von Vordergrundobjekten

Das zweite Verfahren unterliegt nicht der Beschränkung auf bewegte Objekte. Es ist vom Prinzip dem ersten Verfahren ähnlich, auch hier wird wieder eine Differenzbildung durchgeführt. Ist das Hintergrundbild der Szene bekannt, die sogenannte „leere Szene“, dann kann durch Bildung der Differenz des Hintergrundbildes mit dem aktuellen Bild der Sequenz und anschließendem Schwellwertverfahren eine Binärmaske erstellt werden. Die Konnektierung liefert wieder die extrahierten Regionen, diesmal allerdings die Vordergrundobjekte. Als Veranschaulichung dieses Verfahrens seien die folgenden Abb. 5 und 6 gegeben.

Abb. 5: Hintergrundbild (*links*), Bild aus der Sequenz „Ernst-Reuter-Platz“ (*rechts*)

Das Hintergrundbild (siehe Abb. 5 links) wurde aus der Sequenz „Ernst-Reuter-Platz“ extrahiert. Dafür wurde ein Verfahren nach [19] verwendet.

Abb. 6: Differenzbild (*links*), Schwellwertverfahren und Konnektierung (*rechts*)

Im linken Bild der Abb. 6 ist die Differenz vom Hintergrundbild zu erkennen. Das rechte Bild zeigt die extrahierten Vordergrundregionen nach Anwendung des Schwellwertverfahrens und der Konnektierung. Der Vorteil dieses Verfahrens ist, dass auch momentan statische Objekte extrahiert werden. Der Nachteil ist aber die starke Abhängigkeit von äußeren Einflüssen. Äußere Einflüsse sind Änderungen der Beleuchtung und der Umgebung. Eine Änderung der Beleuchtung hat zur Folge, dass sich die Differenz der Intensitäten verringert oder erhöht. Die anschließende Segmentierung mit einem Schwellwertverfahren liefert dann unter Umständen eine falsche Maske.

2.1.3 Gegenüberstellung der vorgestellten Verfahren

Die beiden Verfahren der Differenzbildung sind einfach und effektiv implementierbar. Sie liefern aber beide schlechte Ergebnisse für die Objektverfolgung in realen Szenen. Das erste Verfahren eignet sich ausschließlich zur Bewegungssegmentierung. Das zweite Verfahren

eignet sich für die Extraktion von allen Vordergrundregionen, sofern ein Hintergrundbild zur Verfügung steht. In [9] wurde eine Verfahren zur Segmentierung vorgestellt, dass die äußeren Einflüsse durch ein Kalman-Filter-Formalismus modelliert. Das Filter entscheidet unter Einbeziehung der Dynamik für jedes Pixel, ob es zum Vordergrund oder Hintergrund gehört. Das Verfahren wurde in einer Verkehrsbeobachtung eingesetzt. Ein ähnlicher Ansatz wurde in [12] verwendet. Auch in dieser Diplomarbeit wurde ein solcher Ansatz gewählt.

2.2 Objektverfolgung in 2D

Es gibt viele wissenschaftliche Arbeiten auf dem Gebiet der Objektverfolgung, speziell auch im Hinblick auf die Verkehrsbeobachtung. Es ist zu beachten, dass eine Kamera eine 2D Abbildung einer 3D Szene liefert. Man hat somit eine Dimension verloren. Es handelt sich also nicht mehr um 3D Objekte, sondern um Flächen, welche durch Projektion entstanden. Es gilt also ein hinreichend genaues Modell zu finden, dass diesen Sachverhalt repräsentiert. Je nach Problemstellung variiert hier der Ansatz. Im folgenden werden 3 Arbeiten auf dem Gebiet der Objektverfolgung vorgestellt.
In der Arbeit von [7] wird von einer optimalen Anordnung der Kamera ausgegangen. Die Kamera blickt direkt von oben auf die Fahrbahn. Dadurch kommen echte Objektverdeckung gar nicht zustande, sondern es kommt höchstens zu partiellen Verdeckungen. Damit wurde das größte Problem der Objektverfolgung gelöst. In jedem Zeitschritt wird ein Assoziationsgraph der Objektinteraktionen aufgebaut. Aus den einzelnen Assoziationen lassen sich dann die Spezialfälle ableiten, wie Objektverdeckungen und Ende von Objektverdeckungen. Die Bewegungsparameter werden von einem linearen Kalman-Filter modelliert. Zudem wurde eine Gruppierung der Regionen zu Objekten eingesetzt, um Segmentierungsfehler wie das Aufbrechen eines Objektes in mehrere Regionen entgegenzuwirken. Durch die Einschränkung der Kameraposition ist dieses Verfahren für eine allgemeine Verkehrsbeobachtung nicht geeignet.
In [11] wurden für die Objektverfolgung 3D Modelle benutzt. Durch Vergleich der 3D Modelle mit den Konturen können die Bewegungsparameter bestimmt werden. Der Vorteil dieses Ansatzes ist mit Sicherheit die Genauigkeit, da es sich in der Realität ja auch um einen 3D Raum handelt. Als Nachteil hat sich allerdings die große Anzahl von 3D Modellen herausgestellt, die benötigt werden um alle Fahrzeuge genau verfolgen zu können. Ein Einsatz für eine Echtzeitapplikation in einer Verkehrsbeobachtung ist deshalb nicht möglich.
In [12] wird ein Konturtracker vorgestellt, der zur Verkehrsbeobachtung eingesetzt wird. Die Konturen werden dabei durch uniforme kubische B-Splines approximiert. Zwei lineare Kalman-Filter modellieren die Bewegungsparameter und Form der Kontur. Die Bewegung der Konturen wird durch ein affine Transformationen beschrieben. Die Anordnung der Kamera und die zu observierende Szene ermöglichte es ein robustes Verfahren zu entwickeln, dass auch die Problematik der Objektverdeckung in Grenzen löste. Dieser Ansatz der Konturverfolgung wurde auch in dieser Diplomarbeit verwendet.

2.3 Objektverfolgung in mehreren Kameraansichten

In [2] wird eine Arbeit zur Objektverfolgung in mehreren Ansichten vorgestellt. Es wird von 2 vollkalibrierten Kameras mit gemeinsamen Blickfeld ausgegangen. Die Idee besteht darin, die extrahierten Regionen aus den einzelnen Kameraansichten unter Verwendung der 2D Homographie globalen Objekten zuzuordnen. Anschließend wird die 3D Position berechnet und die Bewegungsparameter der Objekte mit linearen Kalman-Filtern verfolgt. Es konnte gezeigt werden, dass Objektverdeckungen, die bei nur einer Kamera auftreten, mit diesem Ansatz aufgelöst werden.
In der Arbeit von [14] werden die Objekte zuerst in den einzelnen Kameraansichten verfolgt. Dazu erfolgt der Vergleich der Objekte zwischen den Frames durch Aufstellen der Farbhistogramme. Ein einfaches „Best-Match“-Verfahren liefert die Korrespondenzen zwischen den Frames. Um die Ergebnisse der einzelnen Kameraansichten zu vereinigen, wird eine Datenfusion über die Position durchgeführt. Ein „Least-Square“-Verfahren wird verwendet um die 3D Position zu berechnen. Anschließend modelliert ein lineares Kalman-Filter die Trajektorie der Objekte in 3D.
Ein ähnlicher Ansatz wurde auch in dieser Diplomarbeit gewählt. Hier wird auch zuerst eine Objektverfolgung in den einzelnen Ansichten durchgeführt und anschließend die Daten unter Verwendung einer homographischen Abbildung zusammengeführt. Die Trajektorie der 3D Position wird mit einem Kalman-Filter modelliert.

2.4 3D Rekonstruktion

Eine sehr interessante Arbeit zur 3D Rekonstruktion ist in [10] zu finden. Hier wird aus einer Folge von Bildern ein 3D Modell der Szene erzeugt. Der Vorteil dieses Verfahrens ist, dass kein Wissen der Szene und der Kameras benötigt werden. Es wird zuerst die Position und die intrinsischen Parameter der Kamera berechnet (RANSAC). Danach werden Korrespondenzen zwischen den Bildpaaren gesucht, wobei ein Bildpaar als Stereoskopie betrachtet werden kann. Als nächstes wird eine Tiefenkarte der Szene aus den Korrespondenzen berechnet. Die Tiefenkarte wird trianguliert und man erhält somit ein 3D Modell der Szene. Schließlich werden die Bilder auf das 3D Modell projiziert.
In [4] wird ein voxel-basiertes Verfahren zur 3D Rekonstruktion mit Light-Fields vorgestellt. Ein Light-Field repräsentiert eine 4D Funktion, die die Lichtausbreitung in einer Szene mit fester Beleuchtung charakterisiert. Das Verfahren hat den Vorteil, dass das Suchen von Korrespondenzen zwischen den einzelnen Bildern entfällt (voxel-basiertes Verfahren) und das unterschiedliche Beleuchtungen berücksichtigt werden. Für jedes Voxel der Szene wird eine Hypothese anhand der Light-Fields aufgestellt. Durch Rückprojektion der sichtbaren Voxel werden die Voxel entfernt, die nicht konsistent mit den Light-Fields sind.
Die beiden hier vorgestellten Verfahren beruhen auf der Annahme, dass die aufeinanderfolgenden Bilder einem stereoskopischen Bilderpaar entsprechen. Die Anordnung

der Kameras in dieser Diplomarbeit erfüllt diese Bedingung nicht. Aus diesem Grund konnten diese Verfahren nicht für die 3D Rekonstruktion verwendet werden. Das hier entwickelte Verfahren geht von bekannten 3D Modellen aus. Die Rekonstruktion besteht aus der Positionierung und Orientierung des 3D Modells, sowie der Projektion der extrahierten Texturen auf das 3D Modell.

3 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen der verwendeten Verfahren gegeben. Der Abschnitt 3.1 dieses Kapitels behandelt die Theorie des Kalman-Filters. Kalman-Filter finden Verwendung in der Segmentierung als auch in der Objektverfolgung. Abschnitt 3.2 liefert die theoretischen Grundlagen zu der Segmentierung mit einem Kalman-Filter-Formalismus. Abschnitt 3.3 erläutert die Grundlagen der B-Splines. B-Splines werden für die Approximation der Konturen benötigt, um so eine konstante Anzahl von Eckpunkten für die Konturverfolgung zu erhalten. Es folgt in Abschnitt 3.4 eine Beschreibung der 2D Homographie. Sie bildet die Grundlage für die Zuordnung der Objekte aus den einzelnen Kameraansichten. Abschnitt 3.5 erläutert die theoretischen Grundlagen der Kamerageometrie. Diese werden für die Objektverfolgung in 3D und die modell-basierte 3D Rekonstruktion der dynamischen Objekte benötigt.

3.1 Kalman-Filter

3.1.1 Definition des Systemmodells

Das Kalman-Filter ist ein mathematisches Modell zur optimalen Schätzung eines Zustands eines Prozesses. Die optimale Schätzung erfolgt im Sinne des kleinsten quadratischen Fehlers. Das zeitdiskrete Systemmodell wird durch die folgenden vektoriellen Differenzengleichungen beschrieben [6][15] [22]:

$$\mathbf{x}_{k+1} = \mathbf{\Lambda}_{k+1}\mathbf{x}_k + \mathbf{\Gamma}_{k+1}\mathbf{u}_{k+1} + \mathbf{w}_{k+1} \tag{1}$$

$$\mathbf{z}_k = \mathbf{\Phi}_k\mathbf{x}_k + \mathbf{v}_k . \tag{2}$$

Die erste Differenzengleichung beschreibt den Zustand $\mathbf{x} \in \Re^n$ eines zeitdiskreten Prozesses. Die zweite Gleichung formuliert die Messung bzw. Beobachtung $\mathbf{z} \in \Re^m$ dieses Prozesses.

Die unabhängigen Zufallsvariablen $\mathbf{w}_k$ und $\mathbf{v}_k$ werden als normalverteilt angenommen:

$$p(\mathbf{w}_k) \sim N(0, \mathbf{\Omega}_k) \tag{3}$$

$$p(\mathbf{v}_k) \sim N(0, \mathbf{\Xi}_k) . \tag{4}$$

Nur die Statistiken sind von $\mathbf{w}_k$ und $\mathbf{v}_k$ bekannt, welche durch die Kovarianzmatrizen $\mathbf{\Omega}_k$ und $\mathbf{\Xi}_k$ beschrieben werden. $\mathbf{\Omega}_k$ beschreibt den Systemfehler und $\mathbf{\Xi}_k$ den Messfehler. Aus Gleichung 1 ergibt sich, dass $\mathbf{\Lambda}_k$ eine Matrix ist vom Typ $n \times n$. $\mathbf{\Lambda}_k$ und $\mathbf{\Phi}_k$ beschreiben das physikalische Modell. Außerdem ist noch ein weiterer Term in Gleichung 1 zu erkennen.

Dieser lässt eine externe Steuerung des Filters zu, wobei $\mathbf{\Gamma}_k$ vom Typ $n \times l$ ist und $\mathbf{u}_k \in \Re^l$ der deterministische Input. Aus Gleichung 1 folgt, dass $\mathbf{\Phi}_k$ eine quadratische Matrix vom Typ $m \times m$ ist. Der Zustand $\mathbf{x} \in \Re^n$ ist direkt unzugänglich und kann nur über die Beobachtung $\mathbf{z} \in \Re^m$ und der Kenntnis der Statistik geschätzt werden.

3.1.2 Definition des Kalman-Filters

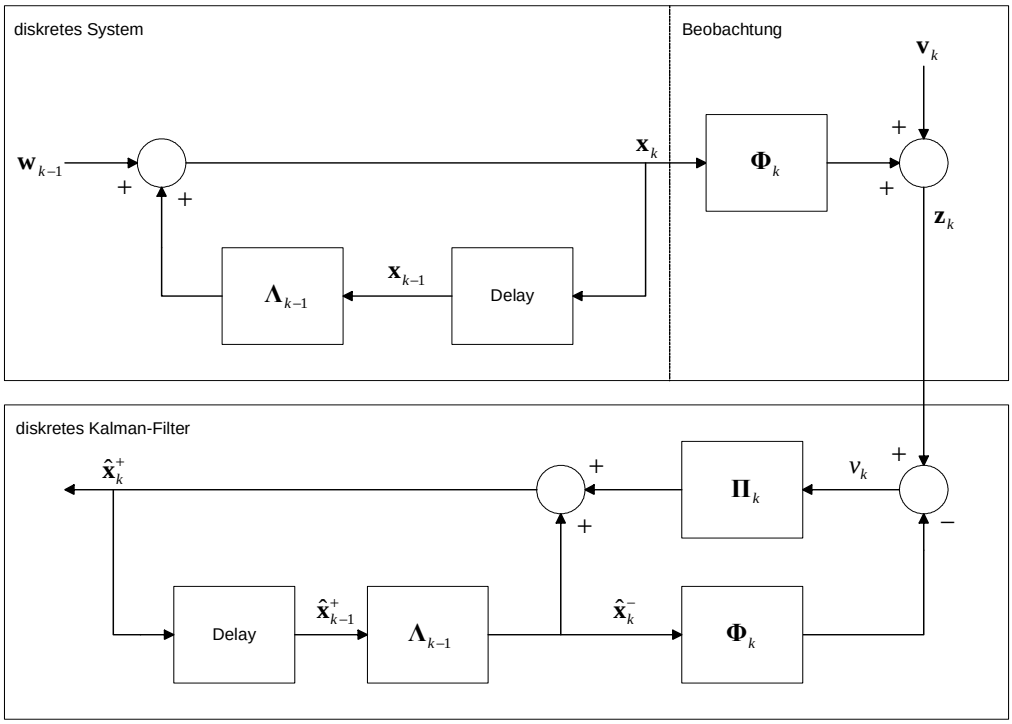


Abb. 7: Systemmodell (*oberer Kasten*), diskretes Kalman-Filter (*unterer Kasten*)

Der obere Teil der Abb. 7 zeigt den von Außen unzugänglichen Teil des Prozesses. Dieser lässt sich noch in das diskrete System und die Beobachtung aufspalten. Gleichungen 1 und 2 definieren genau diesen Sachverhalt mathematisch.
Im unteren Teil ist das diskrete Kalman-Filter zu erkennen. Ihm liegt nur die Messung $\mathbf{z}_k$ und die Statistik der Fehler als Eingabe vor. Das Filter wird durch folgende 5 Gleichungen beschrieben [6][15][22]:

$$\hat{\mathbf{x}}_k^- = \mathbf{\Lambda}_k \hat{\mathbf{x}}_{k-1}^+ + \mathbf{\Gamma}_k \mathbf{u}_k \tag{5}$$

$$\mathbf{\Theta}_k^- = \mathbf{\Lambda}_k \mathbf{\Theta}_{k-1}^+ \mathbf{\Lambda}_k^T + \mathbf{\Omega}_{k-1} \tag{6}$$

$$\mathbf{\Pi}_k = \frac{\mathbf{\Theta}_k^- \mathbf{\Phi}_k^T}{\mathbf{\Phi}_k \mathbf{\Theta}_k^- \mathbf{\Phi}_k^T + \mathbf{\Xi}_k} \tag{7}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{\Pi}_k (\mathbf{z}_k - \mathbf{\Phi}_k \hat{\mathbf{x}}_k^-) \tag{8}$$

$$\boldsymbol{\Theta}_k^+ = (\mathbf{E} - \boldsymbol{\Pi}_k \boldsymbol{\Phi}_k)\boldsymbol{\Theta}_k^- \tag{9}$$

Die beiden Gleichung 5 und 6 bilden die Extrapolation des Zustands bzw. der Fehlerkovarianz. Das Minuszeichen an der Zustandsvariablen bedeutet, dass sich der Zustand unmittelbar vor der Beobachtung befindet. Die Schätzung des Zustands erfolgt in Gleichung 8, wobei das Kalman-Gain $\boldsymbol{\Pi}_k$ die Differenz der Messung und Extrapolation gewichtet.
Die Schätzung erfolgt also durch Korrektur der Prädiktion. Das Pluszeichen zeigt an, dass sich der Zustand unmittelbar nach der Messung befindet. $\boldsymbol{\Theta}_k$ lässt sich allgemein als die Fehlerkovarianz der Schätzung interpretieren.

3.1.3 Der Kalman-Filter-Algorithmus

Ein Zyklus des Kalman-Filters lässt sich in zwei Operationen zerlegen. Die erste Operation ist die Prädiktion („Time Update“) und die zweite Operation die Korrektur („Measurement Update“) [22] (siehe Abb. 8).

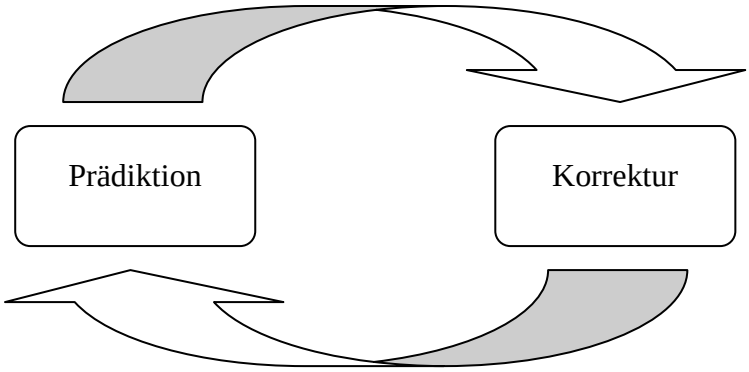


Abb. 8: Kalman-Filter-Zyklus

Die Prädiktion ergibt sich aus den Gleichungen der Extrapolation 5 und 6.

$$\hat{\mathbf{x}}_k^- = \boldsymbol{\Lambda}_k \hat{\mathbf{x}}_{k-1}^+ + \boldsymbol{\Gamma}_k \mathbf{u}_k$$

$$\boldsymbol{\Theta}_k^- = \boldsymbol{\Lambda}_k \boldsymbol{\Theta}_{k-1}^+ \boldsymbol{\Lambda}_k^T + \boldsymbol{\Omega}_{k-1}$$

Der Zustand wird also erst mal in die Zukunft prädiziert. Im Anschluss an die Prädiktion erfolgt eine Korrektur durch Gleichungen 7, 8 und 9:

$$\boldsymbol{\Pi}_k = \frac{\boldsymbol{\Theta}_k^- \boldsymbol{\Phi}_k^T}{\boldsymbol{\Phi}_k \boldsymbol{\Theta}_k^- \boldsymbol{\Phi}_k^T + \boldsymbol{\Xi}_k} \ .$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{\Pi}_k(\mathbf{z}_k - \mathbf{\Phi}_k\hat{\mathbf{x}}_k^-)$$

$$\mathbf{\Theta}_k^+ = (\mathbf{E} - \mathbf{\Pi}_k\mathbf{\Phi}_k)\mathbf{\Theta}_k^-$$

Die Initialisierung des Filters erfordert einen Anfangswert des Zustands $\hat{\mathbf{x}}_0^+$ und einen Anfangswert für die Kovarianzmatrix $\mathbf{P}_0^+$.

3.1.4 Eigenschaften des Kalman-Filters

Da das Kalman-Gain $\mathbf{\Pi}_k$ nur von den Größen $\mathbf{\Lambda}_k, \mathbf{\Phi}_k, \mathbf{\Omega}_k$ und $\mathbf{\Xi}_k$ abhängt, ergibt sich für den Fall das diese Größen über die Zeit konstant bleiben, auch ein konstantes $\mathbf{\Pi}$. Das Filter befindet sich im stationären Zustand und arbeitet wie ein Wiener-Filter mit $\mathbf{\Pi}$ als Verstärkung [6].

Betrachtet man Gleichung 7 so fällt auf, dass das Kalman-Gain für sehr kleine $\mathbf{\Xi}_k$ gegen $\mathbf{\Phi}_k^{-1}$ strebt und somit den Term $\mathbf{z}_k - \mathbf{\Phi}_k\hat{\mathbf{x}}_k^-$ stärker gewichtet:

$$\lim_{\mathbf{\Xi}_k \to \mathbf{0}} \mathbf{\Pi}_k = \mathbf{\Phi}^{-1}. \tag{10}$$

Der umgekehrte Fall ergibt sich für sehr kleine Werte der Kovarianz des Schätzfehlers $\mathbf{\Theta}_k^-$. Die Schätzung erfolgt dann nur noch aus der Prädiktion:

$$\lim_{\mathbf{\Theta}_k \to \mathbf{0}} \mathbf{\Pi}_k = \mathbf{0}. \tag{11}$$

3.2 Segmentierung mit einem Kalman-Filter-Formalismus

Im folgenden Abschnitt wird die Segmentierung von Vordergrund- bzw. Hintergrundregionen nach [9] beschrieben. Es handelt sich dabei um ein Hintergrundschätzverfahren. Die Schätzung des Zustands erfolgt mit einem Kalman-Filter-Formalismus. Jedes Pixel wird mit der geschätzten Intensität verglichen. Weicht die Differenz zu stark von einem konstanten Schwellwert ab, so wird das Pixel als Vordergrund detektiert.

3.2.1 Definition

Der verwendete Kalman-Filter-Formalismus verwendet nur die Gleichungen 5 und 8 des linearen Kalman-Filters:

$$\hat{\mathbf{x}}_k^- = \mathbf{\Lambda}\hat{\mathbf{x}}_{k-1}^+ \tag{12}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{\Pi}(\mathbf{z}_k - \mathbf{\Phi}\hat{\mathbf{x}}_k^-)\,. \tag{13}$$

Es wird von einem fehlerfreiem System und einer fehlerfreien Beobachtung ausgegangen. Es handelt sich also um einen sehr vereinfachten Kalman-Filter. Im folgenden wird nur die Dynamik eines einzelnen Pixels betrachtet. Soll der Algorithmus auf Bilder der Größe $N \times M$ angewendet werden, so muss für jedes Pixel ein solcher Kalman-Filter initialisiert werden.
Der Systemzustand $\mathbf{x}_k$ wird durch die Intensität s und der ersten diskreten Ableitung $\dot{s}$ der Intensität nach der Zeit gebildet. Die Beobachtung $\mathbf{z}_k$ erfolgt über die Intensität s_k des Pixels, da die Kamera nur die Intensität des Pixels liefert. Für die Gleichung 13 der Schätzung ergibt sich nach [9]:

$$\hat{\mathbf{x}}_k^+ = \begin{bmatrix} \hat{s}_k^+ \\ \hat{\dot{s}}_k^+ \end{bmatrix} = \begin{bmatrix} \hat{s}_k^- \\ \hat{\dot{s}}_k^- \end{bmatrix} + \mathbf{\Pi} \cdot \left(s_k - \mathbf{\Phi} \cdot \begin{bmatrix} \hat{s}_k^- \\ \hat{\dot{s}}_k^- \end{bmatrix} \right) \tag{14}$$

mit der konstanten Beobachtungsmatrix

$$\mathbf{\Phi} = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{15}$$

Für die Gleichung 12 der Extrapolation ergibt sich:

$$\begin{bmatrix} \hat{s}_k^- \\ \hat{\dot{s}}_k^- \end{bmatrix} = \mathbf{\Lambda} \begin{bmatrix} \hat{s}_k^+ \\ \hat{\dot{s}}_k^+ \end{bmatrix} \tag{16}$$

mit der konstanten Systemmatrix

$$\mathbf{\Lambda} = \begin{bmatrix} 1 & a_1 \\ 0 & a_2 \end{bmatrix}. \tag{17}$$

Das Kalman-Gain aus Gleichung 14:

$$\mathbf{\Pi} = \begin{bmatrix} \kappa_{1,k} \\ \kappa_{2,k} \end{bmatrix} = \begin{bmatrix} \kappa_k \\ \kappa_k \end{bmatrix} \tag{18}$$

wird auf

$$\kappa = \alpha \cdot m_{k-1} + \beta \cdot (1 - m_{k-1}) \tag{19}$$

gesetzt, wobei m_k eine binäre Maske darstellt:

$$m_{k-1} = \begin{cases} 1, \text{falls } d_{k-1} \geq th_{BgEstimator} \\ 0, \text{sonst} \end{cases} \tag{20}$$

mit

$$d_{k-1} = \left| s_{k-1} - \hat{s}_{k-1}^{+} \right|. \tag{21}$$

Die 1 steht in der binären Maske für ein Vordergrundpixel. Der Parameter $th_{BgEstimator}$ gibt den Schwellwert für die Vordergrund- bzw. Hintergrunddetektion an. Ist die Differenz der Beobachtung und der Schätzung größer als der konstante Schwellwert $th_{BgEstimator}$, so wird das Pixel als Vordergrund erkannt und das Kalman-Gain auf α gesetzt. Im anderen Fall wird das Pixel als Hintergrund erkannt und das Kalman-Gain auf β gesetzt. Die Konstanten α und β lassen sich als Adaptionsgeschwindigkeiten interpretieren.

3.3 Uniforme kubische B-Splines

Die Aufgabe der kubischen B-Splines besteht darin die segmentierten Regionen durch eine geschlossene Kurve zu beschreiben. Sie dienen als Eingabe für die Objektverfolgung in den einzelnen Kameraansichten. Zuerst wird die allgemeine Definition der uniformen kubischen B-Splines gegeben [1]. Im Anschluss wird ein Verfahren vorgestellt, dass geschlossene Kurven durch B-Splines approximiert.

3.3.1 Definition

Ein uniformes kubisches B-Spline ist durch seine vier kubischen Segmente definiert ($u \in \Re$):

$$
\begin{aligned}
b_{-0}(u) &= \frac{1}{6}u^3 \\
b_{-1}(u) &= \frac{1}{6}(1+3u+3u^2-3u^3) \\
b_{-2}(u) &= \frac{1}{6}(4-6u^2+3u^3) \\
b_{-3}(u) &= \frac{1}{6}(1-3u+3u^2-u^3)\,.
\end{aligned}
\tag{22}
$$

Jede Funktion ist für ein bestimmtes Intervall $u_i \le u \le u_{i+1}$ definiert. Die Abb. 9 veranschaulicht die kubischen Segmente im Intervall $u_i \le u \le u_{i+4}$.

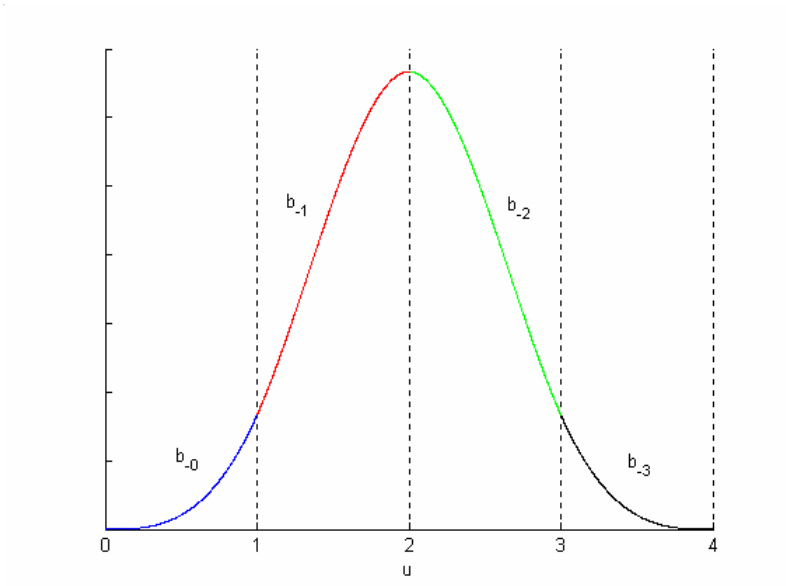


Abb. 9: Kubische Segmente $B_i(u)$

Die Basisfunktion $B_0^4(u)$ besteht aus diesen vier Segmenten. Dabei ist zu beachten das $B_0^4(u)$ zu Null wird für $u \le u_i \wedge u \ge u_{i+4}$. Jede weitere Basisfunktion $B_j^4(u)$ berechnet sich aus einer Verschiebung von $B_0^4(u)$ (siehe Abb. 10).

$$B_j^4(u) = B_0^4(u - j) \tag{23}$$

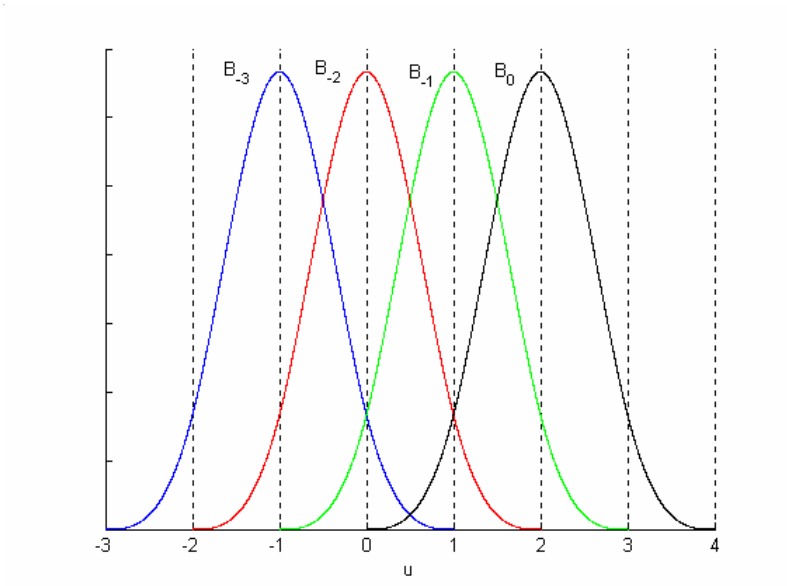


Abb. 10: Die ersten 4 Basisfunktionen

Der Begriff „Uniform“ bedeutet, dass die Abstände der Parametrisierung alle gleich sind.

$$u_{i+1} - u_i = \Delta u = const$$

Eine Kurve Q ist dann allein durch seine Kontrollpunkte V_j bestimmt.

$$Q(u) = \begin{pmatrix} X(u) \\ Y(u) \end{pmatrix} = \sum_{j=0}^{M-1} V_j \cdot B_j^4(u) = \sum_{j=0}^{M-1} \begin{pmatrix} X_j \\ Y_j \end{pmatrix} \cdot B_j^4(u) \tag{24}$$

mit M - Anzahl der Kontrollpunkte

3.3.2 Invarianz gegenüber Translation, Rotation und Skalierung

Eine Verschiebung der Kontrollpunkte ändert nicht die Form der Kurve. Beschreibt t einen Vektor und V_j die Kontrollpunkte, dann folgt:

$$Q_t(u) = \sum_i (V_i + t) \cdot B_i^4(u) = \sum_i V_i \cdot B_i^4(u) + t \cdot \sum_i B_i^4(u). \tag{25}$$

Da nach [1]

$$\sum_i B_i^4(u) = 1 \tag{26}$$

gilt, folgt

$$Q_t(u) = \sum_i V_i \cdot B_i^4(u) + t = Q(u) + t\,. \tag{27}$$

Auch die Rotation der Kontrollpunkte ändert nicht die Form der Kurve. Die Rotation wird durch eine Matrix R beschrieben. Es folgt der Beweis der Invarianz gegenüber Rotation.

$$\begin{aligned} Q_r(u) &= \sum_i (R \cdot V_i) B_i^4(u) = R \cdot \sum_i V_i B_i^4 \\ &= R \cdot Q(u) \end{aligned} \tag{28}$$

Da auch die Skalierung durch eine Matrix beschrieben wird, verläuft der Beweis für die Invarianz analog.

3.3.3 Approximation einer offenen Kurve

Es sollen N Punkte ρ_i durch M Kontrollpunkte V_j eines B-Splines approximiert werden ($M \leq N$). Die Kurve muss folglich aus $M-3$ Segmenten besteht, da jedes Segment durch vier Kontrollpunkte bestimmt wird.

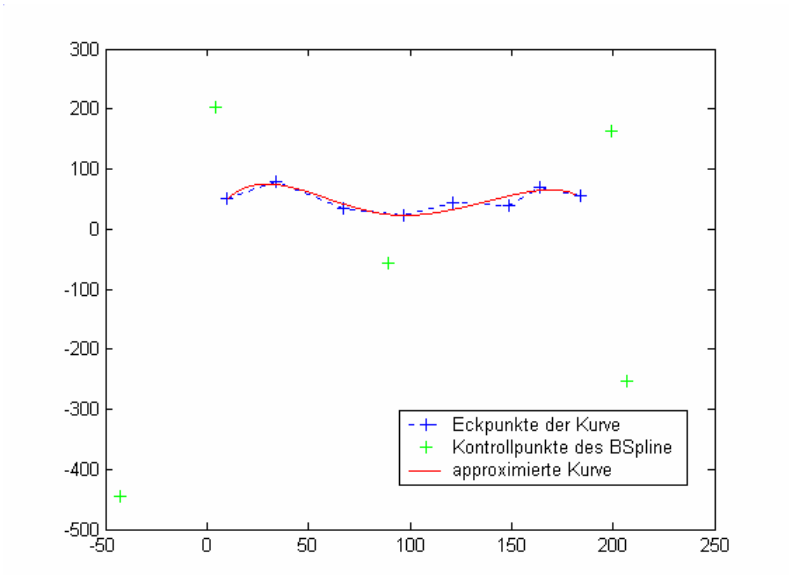


Abb. 11: B-Spline Approximation einer offenen Kurve

Gegeben sind die Punkte ρ_i der Kurve, die approximiert werden soll und die Basisfunktion für uniforme kubische B-Splines. Da die Kurve nicht genau durch ρ_i läuft, sondern nur angenähert, muss der Fehler der dabei entsteht minimiert werden (Optimierung im Sinne kleinster quadratischer Fehler).

$$\sum_{i=0}^{p-1} \left| Q(U_i) - \rho_i \right|^2 = \sum_{i=0}^{p-1} \left[(X(U_i) - x_i)^2 + (Y(U_i) - y_i)^2 \right] = R \tag{29}$$

U_i - Parameterwert der dem i-ten Punkt zugewiesen wurde

Da Gleichung 29 quadratisch ist, erhält man das Minimum nach der ersten Ableitung.

$$\frac{\partial}{\partial Y_l} R = \sum_{j=0}^{m-1} \left[\sum_{i=0}^{p-1} B_j^4(U_i) \cdot B_l^4(U_i) \right] \cdot Y_j - \sum_{i=0}^{p-1} y_i \cdot B_l^4(U_i) = 0 \tag{30}$$

Die Berechnung von X_j erfolgt analog.

Die Parametrisierung erfolgt nach der „chord-length“ Methode. Die „chord-length“ Methode beachtet die Geometrie der Kurve.

$S = \sum_{i=1}^{p-1} \left| \rho_i - \rho_{i-1} \right|$ - Gesamtlänge der Segmente

Der erste Wert der Parametrisierung wird fest auf 3 gesetzt, da 3 der erste Wert ist, für den die ersten 4 Basisfunktionen ungleich Null sind:

$$U_0 = 3$$
$$U_{i+1} = U_i + (m-3)\frac{\left| \rho_{i+1} - \rho_i \right|}{S} \tag{31}$$

mit

$$0 <= i < n-1 .$$

Die Berechnung der Kontrollpunkte reduziert sich auf die Lösung des Gleichungssystems in Gleichung 30. Die Lösung erhält man beispielsweise mit der LR-Zerlegung der Matrix.

3.3.4 Approximation einer geschlossenen Kurve

Für die Berechnung einer geschlossenen Kurve ergibt sich aus der Stetigkeit folgende Bedingung:

$$\begin{aligned} V_0 &= V_{M-3} \wedge \\ V_1 &= V_{M-2} \wedge \\ V_2 &= V_{M-1} \end{aligned} \tag{32}$$

Das bedeutet also, dass die letzten 3 Kontrollpunkte mit den ersten drei Kontrollpunkten übereinstimmen müssen. Mit dieser Bedingung wird Gleichung 30 zu:

$$\frac{\partial}{\partial Y_l} R = \sum_{j=0}^{m-4} \left[\sum_{i=0}^{p-1} \hat{B}_j^4(U_i) \cdot \hat{B}_l^4(U_i) \right] \cdot Y_j - \sum_{i=0}^{p-1} y_i \cdot \hat{B}_l^4(U_i) = 0 \tag{33}$$

mit $$\hat{B}_j^4(U_i) = \begin{cases} B_j^4(U_i) + B_{j+M-3}^4, 0 \le j \le 2 \\ 0, sonst \end{cases} \tag{34}$$

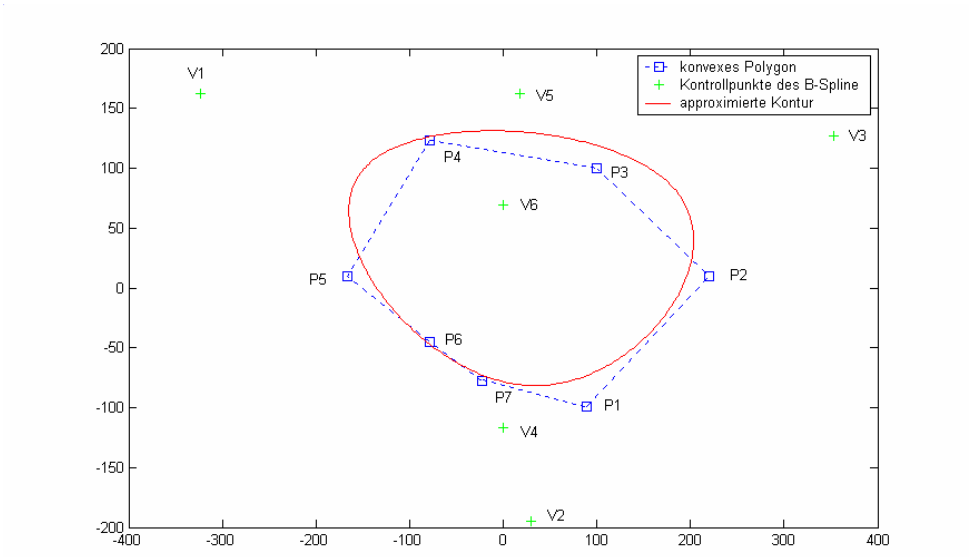


Abb. 12: B-Spline Approximation einer geschlossenen Kurve

Das Beispiel (Abb. 12) zeigt eine Approximation einer konvexen Kontur. Die Kontur besteht aus 7 Eckpunkten und soll durch 6 „einfache“ Kontrollpunkte approximiert werden. Die Anzahl der tatsächlichen Kontrollpunkte beträgt $M = 9$, da die Kontrollpunkte $\{V_1, V_2, V_3\}$ wegen Bedingung 32 doppelt vorkommen.

3.4 2D Homographie

Die Aufgabe der 2D Homographie in dieser Diplomarbeit besteht darin, die Objekte aus den einzelnen Kameraansichten globalen Objekten zuzuordnen. Dieser Abschnitt liefert die Definition und die Grundlagen für die Berechnung der 2D Homographie [8].

3.4.1 Definition

Ist eine Menge von Punkten $\mathbf{p}_i \in \aleph^2$ und die Menge der entsprechenden Punkte $\mathbf{p}_i' \in \aleph^2$ gegeben, so ist $\mathbf{H}$ die projektive Transformation, die die Punkte $\mathbf{p}_i$ nach $\mathbf{p}_i'$ abbildet.

$$\mathbf{p}_i' = \mathbf{H}\mathbf{p}_i \tag{35}$$

$\aleph^2$ bezeichnet die projektive Ebene. Für die Homographie $\mathbf{H}$ ergibt sich aus dieser Definition eine 3×3 Matrix.

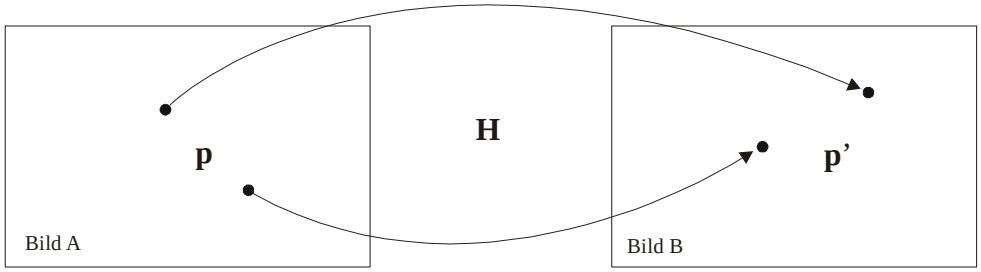


Abb. 13: Veranschaulichung der projektive Transformation $\mathbf{H}$

Abb. 13 zeigt zwei Bilder mit zwei korrespondierenden Punktepaaren. Die Homographie $\mathbf{H}$ beschreibt eindeutig die Abbildung der Punkte von Bild A nach Bild B.

3.4.2 Berechnung der Homographie

Die Berechnung der Homographie $\mathbf{H}$ muss so erfolgen, dass für alle Punkte die Gleichung 35 gilt. Es stellt sich die Frage wie viele korrespondierende Punktepaare $\mathbf{p}_i \leftrightarrow \mathbf{p}_i'$ benötigt werden um die Homographie $\mathbf{H}$ zu berechnen. Eine allgemeine 3×3 Matrix besitzt 9 Freiheitsgrade, da aber h_{33} für die projektive Ebene einer willkürlichen Gesamtskalierung entspricht, ergeben sich für $\mathbf{H}$ genau 8 Freiheitsgrade. Jeder Punkt in $\aleph^2$ besitzt 2 Freiheitsgrade, da der dritte Wert des Punktes wieder eine willkürliche Skalierung darstellt. Für die Lösung von $\mathbf{p}_i' = \mathbf{H}\mathbf{p}_i$ werden also 4 korrespondierende Punktepaare $\mathbf{p}_i \leftrightarrow \mathbf{p}_i'$ benötigt.

In den seltensten Fällen hat man die genauen Korrespondenzen. Üblicherweise sind die Punkte mit einem Fehler überlagert aufgrund ungenauer Messungen. Es handelt sich bei ungenauen Koordinaten um eine Schätzung der Homographie, welche durch iterative Verfahren verbessert werden kann [8].

3.4.3 DLT-Algorithmus zur Berechnung der Homographie

Der „Direct Linear Transformation"-Algorithmus [8] bestimmt aus mindestens 4 korrespondierenden Punktepaaren $\mathbf{p}_i \leftrightarrow \mathbf{p}_i'$ die projektive Transformation $\mathbf{H}$. Da es sich um homogene Größen handelt, sind die Punkte $\mathbf{Hp}_i$ und $\mathbf{p}_i'$ nur in der Richtung identisch und unterscheiden sich um einem Skalierungsfaktor. Es ist deshalb einfacher Gleichung 35 durch:

$$\mathbf{p}_i' \times \mathbf{Hp}_i = \mathbf{0} \tag{35}$$

darzustellen.

Folgende Schreibweise wird eingeführt:

$$\mathbf{Hp}_i = \begin{pmatrix} \mathbf{h}^{1T}\mathbf{p}_i \\ \mathbf{h}^{2T}\mathbf{p}_i \\ \mathbf{h}^{3T}\mathbf{p}_i \end{pmatrix}, \tag{36}$$

wobei $\mathbf{h}^{jT}$ der j-ten Zeile der Matrix $\mathbf{H}$ entspricht.

Das Kreuzprodukt aus Gleichung 35 lässt sich dann als:

$$\mathbf{p}_i' \times \mathbf{Hp}_i = \begin{pmatrix} y_i'\mathbf{h}^{3T}\mathbf{p}_i - w_i'\mathbf{h}^{2T}\mathbf{p}_i \\ w_i'\mathbf{h}^{1T}\mathbf{p}_i - x_i'\mathbf{h}^{3T}\mathbf{p}_i \\ x_i'\mathbf{h}^{2T}\mathbf{p}_i - y_i'\mathbf{h}^{1T}\mathbf{p}_i \end{pmatrix} = \mathbf{0} \tag{37}$$

mit

$$\mathbf{p}_i' = \begin{pmatrix} x_i' \\ y_i' \\ w_i' \end{pmatrix} \tag{38}$$

schreiben.

Da $\mathbf{h}^{jT}\mathbf{p}_i = \mathbf{p}_i^T\mathbf{h}^j$ gilt, kann Gleichung 38 in die folgende Form überführt werden:

$$\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{p}_i^T & y_i'\mathbf{p}_i^T \\ w_i'\mathbf{p}_i^T & \mathbf{0}^T & -x_i'\mathbf{p}_i^T \\ -y_i'\mathbf{p}_i^T & x_i'\mathbf{p}_i^T & \mathbf{0}^T \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = \mathbf{0}. \tag{39}$$

Die dritte Zeile der Matrix aus Gleichung 39 kann weggelassen werden, da nur 2 Zeilen linear unabhängig sind. Die dritte Zeile lässt sich durch Linearkombination der ersten beiden Zeilen darstellen.

Gleichung 39 wird zu:

$$\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{p}_i^T & y_i'\mathbf{p}_i^T \\ w_i'\mathbf{p}_i^T & \mathbf{0}^T & -x_i'\mathbf{p}_i^T \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = \mathbf{0}$$

$$\mathbf{A}_i\mathbf{h} = \mathbf{0}. \tag{40}$$

Für jede Korrespondenz $\mathbf{p}_i \leftrightarrow \mathbf{p}_i'$ wird die Matrix $\mathbf{A}_i$ berechnet. Die Matrizen $\mathbf{A}_i$ werden zu einer Matrix $\mathbf{A}$ gestapelt. Da für $\mathbf{h}$ eine willkürliche Skalierung gilt, kann $\mathbf{h}$ auf 1 normiert werden.

$$\mathbf{Ah} = \mathbf{0} \tag{41}$$

Die Lösung eines solchen Gleichungssystems, mit einer Normierung auf 1 für $\mathbf{h}$, kann mit der SVD („Singular Value Decomposition“) erfolgen. Die Lösung erfolgt über die Berechnung des Eigenvektors mit dem kleinsten Eigenwert der Matrix $\mathbf{A}$ [18]. Die Zerlegung von $\mathbf{A}$ liefert:

$$\mathbf{A} = \mathbf{UDV}^T \tag{42}$$

Die Matrizen $\mathbf{U}$, $\mathbf{V}$ sind orthogonale Matrizen und $\mathbf{D}$ ist eine Diagonalmatrix mit nur positiven Einträgen, den Eigenwerten der Matrix $\mathbf{A}$. Sind die Werte absteigend in der Diagonalen von $\mathbf{D}$, so ist $\mathbf{h}$ die letzte Spalte von $\mathbf{V}$.

Die Homographie $\mathbf{H}$ ergibt sich aus $\mathbf{h}$ ($\mathbf{h}^{jT}$ der j-ten Zeile der Matrix $\mathbf{H}$).

Nach [8] ist das Ergebnis des DLT-Algorithmus nicht invariant gegenüber Ähnlichkeits-transformationen. Eine solche Transformation besteht nur aus Rotation, Skalierung und Translation. Das bedeutet, dass das Ergebnis des DLT-Algorithmus nicht unabhängig von der Lage der Punkte ist. Es ist deshalb wichtig die Daten vorher zu normalisieren. Dazu muss für beide Ebenen eine Ähnlichkeitstransformation bestimmt werden und zwar so, dass das Zentrum der Punkte bei (0,0) liegt und der durchschnittliche Abstand bei $\sqrt{2}$. Nach der Transformation der Punkte kann der DLT-Algorithmus angewendet werden. Als Ergebnis des

DLT-Algorithmus erhält man dann die vorläufige Homographie $\tilde{\mathbf{H}}$. Die eigentliche Homographie ergibt sich anschließend aus der Rücktransformation der vorläufigen Homographie $\tilde{\mathbf{H}}$.

$$\mathbf{H} = \mathbf{T}'^{-1}\tilde{\mathbf{H}}\mathbf{T} \tag{43}$$

mit

$\mathbf{T}$ - Ähnlichkeitstransformation für die Normalisierung der Punkte $\mathbf{p}_i$
$\mathbf{T}'$ - Ähnlichkeitstransformation für die Normalisierung der Punkte $\mathbf{p}_i'$.

3.5 Kamerageometrie

Die Kamerageometrie wird für die Berechnung der 3D Position und für die 3D Rekonstruktion benötigt. Abschnitt 3.5.1 erläutert das in dieser Arbeit verwendete Kameramodell und die damit verbundene Kameramatrix $\mathbf{P}$. Es sei hier noch angemerkt, dass die Projektion keine Linsenverzerrungen berücksichtigt. Abschnitt 3.5.2 liefert die theoretischen Grundlagen, wie die Kameramatrix $\mathbf{P}$ in die intrinsische bzw. extrinsische Matrix zerlegt wird. Abschnitt 3.5.3 zeigt wie die Kameramatrix $\mathbf{P}$ anhand von korrespondierenden Punktepaaren berechnet werden kann (DLT-Algorithmus). Abschnitt 3.5.4 behandelt die Rückprojektion von 2D Koordinaten nach einer Geraden in 3D. Die theoretischen Grundlagen wurden [8] entnommen.

3.5.1 Definition der Kameramatrix

Die Kameramatrix $\mathbf{P}$ ist eine 3×4 Matrix, die 3D Koordinaten nach 2D Koordinaten projiziert. $\mathbf{P}$ lässt sich in zwei Matrizen zerlegen. Die erste Matrix beschreibt die externe euklidische Transformation von 3D Weltkoordinaten nach 3D Kamerakoordinaten und wird durch eine 3×4 Matrix repräsentiert. Der Kameraursprung liegt nach der Transformation im Nullpunkt des Koordinatensystems und die z-Achse liegt auf der Blickrichtung der Kamera, also kolinear zur Normalen der Bildebene. Die Transformation entspricht einer Rotation $\mathbf{R}$ und einer Translation $\mathbf{C}$, wobei $\mathbf{C}$ der Kameraposition in Weltkoordinaten entspricht.
Die zweite Matrix beschreibt die intrinsischen Parameter und hängt von dem verwendeten Kameramodell ab. Die Matrix wird auch als Kalibrierungsmatrix $\mathbf{K}$ bezeichnet und ist vom Typ 3×3:

$$\mathbf{K} = \begin{bmatrix} \eta_x & \vartheta & x_0 \\ 0 & \eta_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{44}$$

Matrix 44 beschreibt die intrinsischen Parameter der endlichen projektiven Kamera. Das Bildseitenformat (Aspekt ratio) ist hier durch die Parameter η_x und η_y bezeichnet. η_x lässt sich weiterhin in:

$$\eta_x = f \cdot m_x \tag{45}$$

zerlegen. f gibt hierbei die Brennweite an und m_x die Anzahl der Pixel pro Bildkoordinate.

Die Parameter x_0 und y_0 geben den Schnittpunkt der z-Achse der Kamera durch die Bildebene in Pixel an. Auch diese Parameter lassen sich zerlegen in:

$$x_0 = m_x \cdot \delta_x . \tag{46}$$

δ_x bezeichnet den Schnittpunkt in Kamerakoordinaten. Für η_y und δ_y ist die Definition analog.
ϑ ist ein Scherungsparameter und ist im Normalfall immer Null. Ist $\vartheta \neq 0$, so bedeutet das, dass die x- und y-Achse nicht rechtwinklig zueinander stehen. Das ist im Falle eines CCD-Feldes eher unwahrscheinlich. $\vartheta \neq 0$ kann dann entstehen, wenn ein Bild von einem anderen Bild aufgenommen wurde [8].

Die Kameramatrix $\mathbf{P}$ wird dann zu:

$$\mathbf{P} = \mathbf{KR}[\mathbf{E} \mid -\mathbf{C}] \tag{47}$$

$$\mathbf{P} = \mathbf{K}[\mathbf{R} \mid -\mathbf{RC}] = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \tag{48}$$

mit $\mathbf{C}$ der Kameraposition als inhomogener Vektor und $\mathbf{E}$ einer 3×3 Einheitsmatrix.

Eine Vergrößerung (Zoom) geht durch ein Skalar θ mit in die Kalibrierungsmatrix $\mathbf{K}$ ein:

$$\mathbf{K}_{zoom} = \begin{bmatrix} \theta & 0 & 0 \\ 0 & \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{K}. \tag{49}$$

3.5.2 Dekomposition der Kameramatrix

Wie im ersten Abschnitt gezeigt wurde, besteht die Kameramatrix $\mathbf{P}$ aus intrinsischen und extrinsischen Parametern:

$$\mathbf{P} = \mathbf{K}[\mathbf{R} \mid -\mathbf{RC}] = [\mathbf{M} \mid -\mathbf{MC}]. \tag{50}$$

Die Berechnung der Kameraposition $\mathbf{C}$ aus der Kameramatrix $\mathbf{P}$ ist einfach:

$$\mathbf{C} = -\mathbf{M}^{-1}\mathbf{n}_4 \tag{51}$$

mit $\mathbf{n}_i$ ist die i-te Spalte von $\mathbf{P}$ und

$$\mathbf{M} = \left[\mathbf{n}_1 \mid \mathbf{n}_2 \mid \mathbf{n}_3\right]. \tag{52}$$

$\mathbf{M}$ lässt sich weiterhin in eine obere Dreiecksmatrix $\mathbf{K}$, welche die Kalibrierungsmatrix ist, und eine Rotationsmatrix $\mathbf{R}$ zerlegen. $\mathbf{R}$ gibt die Orientierung der Kamera an, d.h. die Blickrichtung:

$$\mathbf{M} = \mathbf{KR}. \tag{53}$$

Die Zerlegung der Matrix $\mathbf{M}$ erfolgt mit der QR-Methode [18].

3.5.3 Schätzung der Kameramatrix mit dem DLT-Algorithmus

Eine 3×4 Matrix besitzt im Allgemeinen 12 Freiheitsgrade. Für die Kameramatrix $\mathbf{P}$ ergeben sich 11 Freiheitsgrade, da p_{34} einer willkürlichen Gesamtskalierung entspricht. Aus diesem Grund benötigt die minimale Lösung 11 Gleichungen, welche sich aus 6 korrespondierenden Punktepaaren ergeben (siehe auch Abschnitt 3.4.2).

Die Kameramatrix $\mathbf{P}$ projiziert jede 3D Koordinate in die Bildebene:

$$\mathbf{p}_i = \mathbf{P}\mathbf{p}'_i \tag{54}$$

mit $\mathbf{p} \in \aleph^2$ und $\mathbf{p}' \in \aleph^3$.

Es folgt der gleiche Ansatz wie in Abschnitt 3.4.3 „DLT-Algorithmus zur Berechnung der Homographie“. Für jede $\mathbf{p}'_i \leftrightarrow \mathbf{p}_i$ lässt sich die Beziehung ausdrücken als:

$$\begin{bmatrix} \mathbf{0}^T & -w'_i\mathbf{p}'^T_i & y'_i\mathbf{p}'^T_i \\ w'_i\mathbf{p}'^T_i & \mathbf{0}^T & -x'_i\mathbf{p}'^T_i \\ -y'_i\mathbf{p}'^T_i & x'_i\mathbf{p}'^T_i & \mathbf{0}^T \end{bmatrix} \begin{pmatrix} \mathbf{n}^1 \\ \mathbf{n}^2 \\ \mathbf{n}^3 \end{pmatrix} = \mathbf{0} \tag{55}$$

wobei $\mathbf{n}^{iT}$ der i-ten Zeile von $\mathbf{P}$ entspricht. Die dritte Zeile der Matrix aus Gleichung 55 kann weggelassen werden, da nur 2 Zeilen linear unabhängig sind:

$$\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{p}_i'^T & y_i'\mathbf{p}_i'^T \\ w_i'\mathbf{p}_i'^T & \mathbf{0}^T & -x_i'\mathbf{p}_i'^T \end{bmatrix} \begin{pmatrix} \mathbf{n}^1 \\ \mathbf{n}^2 \\ \mathbf{n}^3 \end{pmatrix} = \mathbf{0}. \tag{56}$$

Für jede Korrespondenz $\mathbf{p}'_i \leftrightarrow \mathbf{p}_i$ wird die Matrix $\mathbf{A}_i$ berechnet. Die Matrizen $\mathbf{A}_i$ werden zu einer Matrix $\mathbf{A}$ gestapelt. Da für $\mathbf{n}$ eine willkürliche Skalierung gilt, kann $\mathbf{n}$ auf 1 normiert werden:

$$\mathbf{Ap} = \mathbf{0} \tag{57}$$

mit $\mathbf{n}$ als Vektor, der alle Einträge von $\mathbf{P}$ enthält. Die Lösung des Gleichungssystems erfolgt wieder durch Zerlegung der Matrix $\mathbf{A}$ mit der SVD.

Wie im Falle der Homographie müssen die Daten vor Anwendung des DLT-Algorithmus normalisiert werden. Die Bildkoordinaten werden, wie im Abschnitt 3.4.3 beschrieben, normalisiert (Matrix $\mathbf{T}$). Für die 3D Koordinaten erfolgt die Normalisierung auf einen durchschnittlichen Abstand von $\sqrt{3}$ (Matrix $\mathbf{U}$). Die vorläufige Kameramatrix $\tilde{\mathbf{P}}$ erhält man aus der letzten Spalte von $\mathbf{V}$ (siehe auch Abschnitt 3.4.3).
Nachdem die vorläufige Kameramatrix $\tilde{\mathbf{P}}$ berechnet wurde, kann eine lineare Schätzung durchgeführt werden um den geometrischen Fehler zu minimieren. Iterative Verfahren wie Levenberg-Marquardt werden ausführlich in [8] erläutert. Als geometrischer Fehler kann beispielsweise die Distanz von Projektion zu Bildkoordinaten genommen werden. Die entgültige Kameramatrix $\mathbf{P}$ erhält man durch Rücktransformation der vorläufigen Kameramatrix:

$$\mathbf{P} = \mathbf{T}^{-1}\tilde{\mathbf{P}}\mathbf{U} \tag{58}$$

mit

$\mathbf{T}$ - Ähnlichkeitstransformation für die Normalisierung der Punkte $\mathbf{p}_i$
$\mathbf{U}$ - Ähnlichkeitstransformation für die Normalisierung der Punkte $\mathbf{p}'_i$.

Wie im Abschnitt 3.5.2 gezeigt wurde, läßt sich die Kameramatrix in ihre Bestandteile $\mathbf{K}$, $\mathbf{R}$ und $\mathbf{C}$ zerlegen. Nachdem einmal die Kalibrierungsmatrix $\mathbf{K}$ bestimmt wurde, kann diese für weitere Aufnahmen verwendet werden. Die extrinsischen Parameter, die Orientierung $\mathbf{R}$ und die Kameraposition $\mathbf{C}$ müssen der neuen Anordnung entsprechend angepaßt werden. Sind Kameraposition, Blickrichtung und ein sogenannter Up-Vektor bekannt, so läßt sich $\mathbf{R}$ berechnen [21]. Der Up-Vektor gibt die Richtung an, die als oben in dem Koordinatensystem definiert wurde.

3.5.4 Rückprojektion von 2D nach 3D

Die Kameramatrix **P** projiziert 3D Koordinaten nach 2D Bildkoordinaten. Dabei geht ein Freiheitsgrad verloren. Die Rückprojektion entspricht also einer Geraden im 3D Raum. Der Ursprung dieser Geraden liegt in $\mathbf{PC} = \mathbf{0}$, also in der Kameraposition. Der zweite Punkt

$$\mathbf{D} = \left((\mathbf{M}^{-1}\tilde{\mathbf{p}})^T, 0\right)^T \tag{59}$$

ergibt sich durch Schnitt des rückprojizierten Punktes $\tilde{\mathbf{p}} \in \Re^3$ mit der Ebene im Unendlichen für endliche Kameras [8]. Die Gleichung der Geraden wird zu:

$$\mathbf{X}(\lambda) = \lambda \begin{pmatrix} -\mathbf{M}^{-1}\tilde{\mathbf{p}} \\ 0 \end{pmatrix} + \begin{pmatrix} -\mathbf{M}^{-1}\mathbf{n}_4 \\ 1 \end{pmatrix}. \tag{60}$$

4 Implementierung

Die Algorithmen wurden in der Programmiersprache C++ implementiert. C++ erlaubt die Verwendung von modernen Konzepten der Softwareentwicklung, wie objektorientierte Analyse und Design. Im Anhang C befinden sich die Klassendiagramme der einzelnen Module, welche mit dem Entwicklungswerkzeug Rational Rose erstellt wurden.
Eine Beschreibung der Parameter der implementierten Module befindet sich sich im Anhang B. Es wurden alle Algorithmen bis auf die Segmentierung, diese wurde dem Softwarepaket Halcon der Firma MVTec [16] entnommen, selbst implementiert.

4.1 Kalman-Filter

Da die Objektverfolgung auf einen Kalman-Filter basiert, musste zuerst ein solcher implementiert werden. Für die Implementierung des Kalman-Filters wurden die 5 Gleichungen des Filters direkt in die Programmiersprache übersetzt. Die Implementierung wurde allgemein gehalten, d.h. die selbe Implementierung kann für alle Aufgabenstellungen verwendet werden. Am aufwendigsten ist die Berechnung des Kalman-Gains $\mathbf{\Pi}_k$, da hier die Berechnung der Inversen des Nenners benötigt wird:

$$\mathbf{\Pi}_k = \frac{\mathbf{\Theta}_k^- \mathbf{\Phi}_k^T}{\mathbf{\Phi}_k \mathbf{\Theta}_k^- \mathbf{\Phi}_k^T + \mathbf{\Xi}_k} \quad . \tag{61}$$

Die Inverse des Nenners wurde nach [3] mit einer LR-Zerlegung ermittelt. Die Komplexität dieser Berechnung beträgt $O(n^3)$, wobei n für die Anzahl der Spalten bzw. Zeilen dieser quadratischen Matrix steht.

Der Aufruf des Filters erfolgt mit den Matrizen, Fehlerkovarianzen und Vektoren des Systems, der Beobachtung und dem deterministischen Input. Was die Genauigkeit angeht, so ist das Kalman-Filter auf den Zahlentyp „double“ (64 Bit) mit 15 Stellen beschränkt. Es sei noch angemerkt, dass die Angabe der Stellmatrix $\mathbf{\Gamma}$ und des deterministischen Inputs $\mathbf{u}$ in der Implementierung optional ist.

4.2 Objektsegmentierung

Im dritten Kapitel wurde schon das Prinzip der Segmentierung mit Kalman-Filtern erläutert. Das hier verwendete Verfahren [19] ist eine Erweiterung des Kalman-Filter basierten Algorithmus von [9]. Es wurde die Berechnung des Kalman-Gains geändert. Zudem wurde der konstante Schwellwert durch einen variablen Schwellwert ersetzt. Das Filter extrahiert Hintergrund- bzw. Vordergrundregionen eines Bildes für ein gegebenes Hintergrundbild.

Dieser Vorgang kann auch als Schätzung des Hintergrundes bezeichnet werden. Die Dynamik lässt äußere Einflüsse, wie Änderungen in der Beleuchtung und der Umgebung, zu. Die Implementierung wurde dem Softwarepaket Halcon [16] in der Version 6.0.1 der Firma MVTec entnommen. In den nächsten Abschnitten folgt eine Beschreibung dieser Implementierung. Alle Parameter, die zur Initialisierung und Aufrufs des Moduls notwendig sind, werden in Anhang B tabellarisch aufgelistet.

4.2.1 Modell der Dynamik

Nachteil in dem Ansatz von [9], ist die zu schnelle Adaption von Vordergrundregionen in den Hintergrund für große β (siehe Abschnitt 3.2), falls Objekte sich nicht kontinuierlich bewegen. Also auch zum Stillstand kommen können. Für eine allgemeine Verkehrsszene gilt diese Voraussetzung nicht. In der Arbeit von [19] wird dieses berücksichtigt – stillstehende Vordergrundobjekte werden nicht in den Hintergrund adaptiert. Die Erweiterung des Modells für den Kalman-Filter unterscheidet sich nur im Kalman-Gain.

Das Kalman-Gain

$$\mathbf{\Pi} = \begin{bmatrix} \kappa_{1,k} \\ \kappa_{2,k} \end{bmatrix} = \begin{bmatrix} \kappa_k \\ \kappa_k \end{bmatrix} \tag{62}$$

wird nach [19] folgendermaßen modelliert:

$$\kappa_k = \begin{cases} \alpha, \text{falls} & d_k' \geq th_k \vee d_k' < th_k \wedge d_k'' \geq th_k \\ \beta, \text{falls} & d_k' < th_k \wedge d_k'' < th_k \end{cases} \tag{63}$$

mit

$$d_k' = \left| s_k - \hat{s}_k^- \right| \tag{64}$$

$$d_k'' = \left| s_k - \hat{s}_k^{'} \right| \tag{65}$$

$$\hat{s}_k' = \hat{s}_k^- + \beta \cdot \left(s_k - \hat{s}_k^- \right). \tag{66}$$

Die Größen α und β geben Verstärkungsfaktoren an. Mit ihnen lässt sich die Adaptation des Hinter- und Vordergrunds beschleunigen bzw. verlangsamen. Ist der Schwellwert $th_{BgEstimator,k}$ kleiner als die Differenz aus Prädiktion und Beobachtung, dann gehört das Pixel zum Vordergrund und die Schätzung erfolgt mit α. Im anderen Fall wird eine Voraus-Schätzung $\hat{s}_k'$ nach (Gleichung 66) berechnet. Die Dynamik wird in diesem Ansatz stärker gewichtet als in [9]. Die Differenz aus Beobachtung und Voraus-Schätzung wird mit dem Schwellwert $th_{BgEstimator,k}$ verglichen. Ist der Schwellwert größer als die Differenz, dann wird der Pixel als Hintergrund erkannt und die Schätzung erfolgt mit β. Im anderen Fall wird das Pixel wieder

als Vordergrund detektiert und die Schätzung erfolgt wiederum mit α. Im Gegensatz zu dem Ansatz von [9] kann sich der Schwellwert über die Zeit ändern.

4.2.2 Kalman-Gain Modi

Der Hintergrundschätzer kann in zwei Modi betrieben werden, welche nur das Kalman-Gain betreffen. Der „fixed"-Modus gibt an, dass die Größen α und β konstant sind und in einem Wertebereich von 0.0 und 1.0 liegen. Dieser Modus ist speziell für die Detektion von Bewegung geeignet. Der zweite Modus („frame"-Modus) hat zur Folge das eine Tabelle der Kalman-Gains für alle 256 möglichen Grauwertdifferenzen erzeugt wird. Es wird dann über Größen α und β die Anzahl der Frames angegeben, die benötigt werden, um eine Abweichung zwischen Vorhersagewert und Messwert zu adaptieren. Im Gegensatz zu dem ersten Modus werden größere Differenzen schneller adaptiert. Für kleine Differenzen macht sich der Unterschied dieser beiden Modi nicht bemerkbar [19].

4.2.3 Variabler Schwellwert der Hintergrunddetektion

In Abschnitt 4.2.1 ist gezeigt worden, dass der Schwellwert sich über die Zeit ändern kann. Ist die Adaptierung des Schwellwerts aktiviert, so wird dieser aus der Streuung der Grauwerte und der vorgegebenen Basisschwelle bestimmt. Für jeden Zeitschritt wird die Verteilung der Grauwerte ausgewertet und der Basisschwelle hinzuaddiert.

$$th_{BgEstimator,k} = th_{BgEstimator,b} + v_{k-1} \tag{67}$$

v_k gibt die Varianz der Intensität zum Zeitpunkt k an. Genauere Details über die Berechnung der Varianz ist in [19] zu finden.
Die Anzahl der statistischen Datensätze, die zur Auswertung beitragen sollen, wird als Parameter mit angegeben. Da für verdeckte Bereiche keine genaue Aussage gemacht werden kann, wird die Hintergrundstatistik durch die Student's t-Distribution bestimmt [20]. Die Distribution wird durch den Parameter χ des Konfidenzintervalls bestimmt. Als letzter Parameter muss noch eine Zeitkonstante τ für eine e-Funktion angegeben werden, die die Schwelle bei Vordergrunddetektion anhebt. Das Anheben der Schwelle ist nötig, da im Falle einer Verdeckung keine Aussage über den Hintergrund getroffen werden kann. Des weiteren werden die Parameter des Hintergrundschätzers in [16] erklärt.

4.2.4 Nachverarbeitung der Segmentierung

Die Nachverarbeitung wird nach der Hintergrundschätzung durchgeführt. Die Tools wurden auch dem Softwarepaket Halcon entnommen. Die Nachverarbeitung schneidet das Ergebnis der Segmentierung mit der „Region of Interest“, diese entspricht dem relevanten Bereich, hier der Strasse. Diese muss als zusätzlicher Parameter angegeben werden und darf nicht leer sein. Anschließend werden die Regionen konnektiert. Die konnektierten Regionen werden nach der Größe der Fläche gefiltert – um so kleine Regionen auszusieben. Der Parameter für die Mindestfläche muss vorher festgelegt worden sein. Die restlichen Regionen werden zu konvexen Polygonen umgewandelt. Das nachfolgende Blockschaltbild Abb. 14 veranschaulicht noch mal die einzelnen Schritte dieses Moduls.

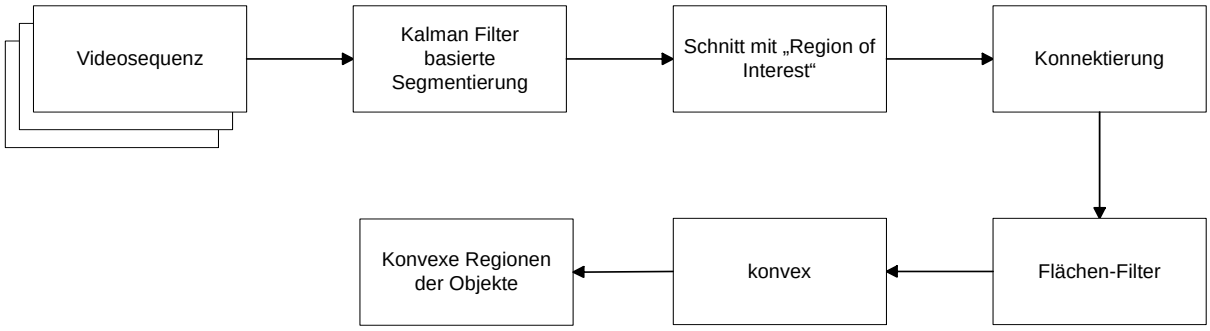


Abb. 14: Blockschaltbild der Segmentierung

4.3 Objektverfolgung mit Kalman-Filtern in 2D

Die Objektverfolgung basiert auf einem Ansatz aus [12]. Die von der Segmentierung kommenden konvexen Konturen werden für die Objektverfolgung durch B-Splines approximiert. Eine Approximation durch B-Splines bietet den Vorteil, dass die Anzahl der zu verfolgenden Eckpunkte konstant gehalten werden kann und die Anzahl je nach gewünschter Genauigkeit variiert werden kann.

4.3.1 Aufbau der Objektverfolgung in 2D

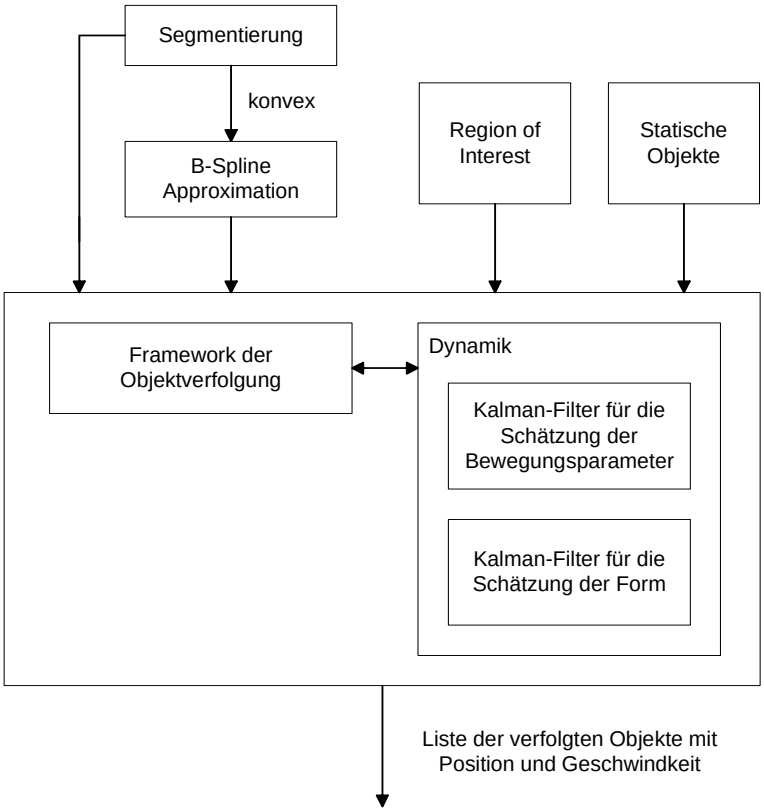


Abb. 15: Blockschaltbild der Objektverfolgung in 2D

Die Abb. 15 veranschaulicht die internen Module der Objektverfolgung in 2D. Als erstes werden die konvexen Polygone der Segmentierung durch ein geschlossenes B-Spline approximiert, wobei die Eckpunkte der konvexen Kontur approximiert werden. Da die Approximation mit konstanter Anzahl der Kontrollpunkte N arbeitet, gibt es eine Einschränkung - es können nur Konturen mit Anzahl M der Eckpunkte approximiert werden, wenn gilt $M \geq N - 3$.

Die Kontrollpunkte des B-Splines bilden den Eingangsvektor (Beobachtungsvektor) der beiden linearen Kalman-Filter. Das erste Filter wird für die Schätzung der

Bewegungsparameter verwendet, das zweite schätzt die Form der Kontur. Die Modellierung durch zwei Filter hat den Vorteil, dass lineare Kalman-Filter verwendet werden können [12]. Dieses ist einfacher zu implementieren, hat eine geringere Komplexität und ist stabiler als das erweiterte Kalman-Filter [6]. Die Bewegungsschätzung erfolgt nach einem physikalischen Modell konstanter Geschwindigkeit, d.h. es wird nur die erste Ableitung der Strecke nach der Zeit (Geschwindigkeit) betrachtet, um auch hier wieder nur das lineare Kalman-Filter benutzen zu können.
Für die Objektverfolgung wird noch eine „Region of Interest“ benötigt. Diese gibt an, welcher Bereich für die Verfolgung relevant ist. Für die Verkehrsbeobachtung ist das natürlich die Strasse bzw. Kreuzung. Außerdem können noch statische Vordergrundobjekte (bzw. bewegte Hintergrundobjekte) mit in der Verfolgung berücksichtigt werden. Statische Vordergrundobjekte sind Objekte, die nicht in die Verfolgung miteinbezogen werden sollen (wie z.B. Bäume oder Fahnenmasten). Als Ergebnis dieses Moduls erhält man eine Liste mit den verfolgten Objekten.

4.3.2 B-Spline Approximation geschlossener Kurven

Die Approximation einer geschlossenen Kurve durch ein kubisches B-Spline erfolgt durch Lösung des Gleichungssystems 33. Als Eckpunkte der zu approximierenden Kurve wurden die Eckpunkte der konvexen Kontur verwendet:

$$\frac{\partial}{\partial Y_l} R = \sum_{j=0}^{m-4} \left[\sum_{p=0}^{p-1} \hat{B}_j^4(U_i) \cdot \hat{B}_l^4(U_i) \right] \cdot Y_j - \sum_{i=0}^{p-1} y_i \cdot \hat{B}_l^4(U_i) = 0 \tag{68}$$

mit $$\hat{B}_j^4(U_i) = \begin{cases} B_j^4(U_i) + B_{j+M-3}^4, 0 \le j \le 2 \\ 0, sonst \end{cases}$$

m - Anzahl der Kontrollpunkte des B-Splines

p - Anzahl der Punkte der zu approximierenden Kurve.

Gleichungssystem 68 lässt sich in Matrixschreibweise als

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{D} \tag{69}$$

mit $$A_{jl} = \sum_{i=0}^{p-1} \hat{B}_j^4(U_i) \cdot \hat{B}_l^4(U_i) \tag{70}$$

$$D_l = \sum_{i=0}^{p-1} y_i \cdot \hat{B}_l^4(U_i) \quad \text{bzw. } x_i \text{ für die x-Komponente} \tag{71}$$

$0 <= j, l < m$ -3

schreiben.

Die Implementierung bestand darin den Vektor $\mathbf{U}$, die Matrix $\mathbf{A}$ und den Vektor $\mathbf{D}$ zu berechnen. Der Vektor $\mathbf{U}$ beinhaltet die Parametrisierung nach der „chord-length"-Methode. $\mathbf{A}$ und $\mathbf{D}$ ergeben sich aus den Gleichungen 70 und 71. Es ist zu beachten, dass der Vektor $\mathbf{U}$ und die Matrix $\mathbf{A}$ nur einmal für beide xy-Komponenten berechnet werden müssen.

Für die Berechung von $\mathbf{U}$ ergibt sich eine lineare Komplexität von $O(p)$, für $\mathbf{A}$ erhält man eine Komplexität von $O(m^2 p)$ und schließlich für $\mathbf{D}$ $O(mp)$.
Ein solches Gleichungssystem ist lösbar, wenn $\mathbf{A}$ regulär ist:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{E}.$$

Die Lösung erhält man effektiv mit der LR-Zerlegung der Matrix $\mathbf{A}$. Die Grundlagen und der Pseudo-Code für die Implementierung der LR-Zerlegung wurden [3] entnommen. Die Komplexität der LR-Zerlegung beträgt $O(n^3)$ mit n als Anzahl der Spalten bzw. Reihen der quadratischen Matrix $\mathbf{A}$. Die Berechnung von $\mathbf{D}$ weist eine Komplexität von $O(n^2)$ auf, so das sich für das Aufstellen und Lösen der Gleichung 69 eine kubische Komplexität ergibt.
Das Ergebnis der Approximation sind zwei Vektoren, jeweils einen für die x-Koordinate und einen für die y-Koordinaten der Kontrollpunkte. Diese dienen als Beobachtungsvektor für die Objektverfolgung im anschließenden Modul.

4.3.3 Bewegungsmodell der Objektverfolgung in 2D

Die Segmentierung liefert geschlossene, ebene Konturen von 3D Objekten, da die Kamera die 3D Objekte, also die Fahrzeuge, auf die Bildebene projiziert hat. Es gilt also ein Modell zu finden, dass solche Konturen und deren Bewegung hinreichend genau beschreibt. Die Schätzung des Zustands mit dem Kalman-Filter wird mit ansteigender Genauigkeit des Modells besser. Gute Resultate wurden in [12] mit dem affinen Bewegungsmodell erzielt, welches auch in dieser Arbeit verwendet wurde.

Ist eine Kontur mit seinen Eckpunkten $\tilde{\mathbf{p}} \in \Re^2$ gegeben, so lässt sich die affine Bewegung dieser Kontur folgendermaßen beschreiben [12]:

$$\mathbf{m}(\tilde{\mathbf{p}}) = \mathbf{N}(\tilde{\mathbf{p}} - \tilde{\mathbf{p}}_m) + \mathbf{t} \qquad (72)$$

mit $\mathbf{N}$ einer Rotations- und Skalierungsmatrix, $\tilde{\mathbf{p}}_m$ dem Mittelpunkt der Kontur und $\mathbf{t}$ dem Translationsvektor.

Die Matrix $\mathbf{N}$ kann noch vereinfacht werden, in dem angenommen wird, dass die Rotation sehr gering ist. Der Fehler der dadurch entsteht muss dann in dem Prozessrauschen berücksichtigt werden. Es bleibt also nur noch eine Skalierung in Matrix $\mathbf{N}$ übrig. Diese

ergibt sich aus der Bewegung der Objekte entlang der Kameraachse und kann deshalb durch einen skalaren Wert ersetzt werden.

Gleichung 72 wird dann zu:

$$\mathbf{m}(\tilde{\mathbf{p}}) = \xi(\tilde{\mathbf{p}} - \tilde{\mathbf{p}}_m) + \mathbf{t} \ . \tag{73}$$

4.3.4 Modellierung der Bewegungsdynamik

Die Dynamik der Objekte wird mit zwei Kalman-Filtern modelliert. Als Eingabe erhalten sie die Eckpunkte $\mathbf{Y}_k$ der Kontur. Das erste Filter schätzt die Bewegungsparameter des Objektes, d.h. Positionsänderung $\partial\mathbf{t}$ und Skalierung ξ. Diese sind nach Gleichung 73:

$$\mathbf{x}_{Motion,k} = \begin{bmatrix} \partial\mathbf{t}_k \\ \xi_k \end{bmatrix}. \tag{74}$$

Das zweite Filter schätzt die Form der Kontur, d.h. die Kontrollpunkte des B-Splines:

$$\mathbf{x}_{Shape,k} = \begin{bmatrix} \begin{bmatrix} x_{1,k} & y_{1,k} \end{bmatrix}^T \\ \begin{bmatrix} x_{2,k} & y_{2,k} \end{bmatrix}^T \\ \vdots \\ \begin{bmatrix} x_{n,k} & y_{n,k} \end{bmatrix}^T \end{bmatrix} \quad \text{mit } n \text{ - Anzahl der Kontrollpunkte.} \tag{75}$$

Die Systemgleichungen der Bewegungsparameter folgen aus dem affinen Bewegungsmodell Gleichung 73:

$$\mathbf{x}_{Motion,k+1} = \mathbf{\Lambda}_{Motion,k}\mathbf{x}_{Motion,k} + \mathbf{w}_k \tag{76}$$

$$\mathbf{z}_{Motion,k} = \mathbf{\Phi}_{Motion,k}\mathbf{x}_{Motion,k} + \mathbf{v}_k \tag{77}$$

mit

$$\mathbf{\Lambda}_{Motion,k} = \mathbf{E}_3 \tag{78}$$

$$\mathbf{z}_{Motion,k} = \mathbf{Y}_k - \mathbf{x}_{Shape,k-1} \tag{79}$$

$$\mathbf{\Phi}_{Motion,k} = \begin{bmatrix} \mathbf{E}_2 & \mathbf{x}_{Shape,1,k-1} - \mathbf{c}_{k-1} \\ \vdots & \vdots \\ \mathbf{E}_2 & \mathbf{x}_{Shape,n,k-1} - \mathbf{c}_{k-1} \end{bmatrix} \tag{80}$$

und $\mathbf{c}_k$ dem Zentrum der Kontrollpunkte. Es ist zu beachten das $\mathbf{z}_{Motion,k}$ die Änderung der Kontur beschreibt. $\mathbf{\Phi}_{Motion,k}$ beschreibt den Zusammenhang von Zustand zu Beobachtung. Die Fehlerstatistik wird von den konstanten Kovarianzmatrizen beschrieben:

$$\mathbf{\Omega}_{Motion} = \begin{bmatrix} \sigma_{p,x}^2 & 0 & 0 \\ 0 & \sigma_{p,y}^2 & 0 \\ 0 & 0 & \sigma_{p,\xi}^2 \end{bmatrix} \tag{81}$$

$$\mathbf{\Xi}_{Motion} = \begin{bmatrix} \sigma_{m,1}^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{m,n}^2 \end{bmatrix} \tag{82}$$

mit

$$\sigma_{m,Motion} = \sigma_{m,1} = \sigma_{m,2} = ... = \sigma_{m,n}.$$

Für die Schätzung der Kontrollpunkte folgt das Systemmodell aus Gleichung 73:

$$\mathbf{x}_{Shape,k+1} = \mathbf{\Lambda}_{Shape,k}\mathbf{x}_{Shape,k} + \mathbf{\Gamma}_{Shape,k}\mathbf{u}_{Shape,k} + \mathbf{w}_k \tag{83}$$

$$\mathbf{z}_{Shape,k} = \mathbf{\Phi}_k\mathbf{x}_{Shape,k} + \mathbf{v}_k \tag{84}$$

mit

$$\mathbf{\Lambda}_{Shape,k} = \mathbf{E}_{2n} \tag{85}$$

$$\mathbf{\Gamma}_{Shape,k} = \begin{bmatrix} \mathbf{E}_2 & \mathbf{x}_{Shape,1,k-1} - \mathbf{c}_{k-1} \\ \vdots & \vdots \\ \mathbf{E}_2 & \mathbf{x}_{Shape,n,k-1} - \mathbf{c}_{k-1} \end{bmatrix} \tag{86}$$

$$\mathbf{u}_{Shape,k} = \mathbf{x}_{Motion,k} \tag{87}$$

$$\mathbf{\Phi}_{Shape,k} = \mathbf{E}_{2n}. \tag{88}$$

Die Bewegungsparameter $\mathbf{x}_{Motion,k}$ bilden den deterministischen Input des zweiten Kalman-Filters. $\mathbf{\Lambda}_{Shape,k}$ und $\mathbf{\Phi}_{Shape,k}$ sind beide Einheitsmatrizen und haben somit keine ändernde Wirkung auf den Zustand. Der Zustand $\mathbf{x}_{Shape,k}$ wird direkt über den deterministischen Input gesteuert. Der Term $\mathbf{\Gamma}_{Shape,k}\mathbf{u}_{Shape,k}$ in Gleichung 83 addiert die Änderung zum geschätzten Zustand. Die Fehlerstatistik wird wiederum von den konstanten Kovarianzmatrizen beschrieben:

$$\mathbf{\Omega}_{Shape} = \begin{bmatrix} \sigma_{p,1}^2 & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & \sigma_{p,n}^2 \end{bmatrix} \tag{89}$$

$$\mathbf{\Xi}_{Shape} = \begin{bmatrix} \sigma_{m,1}^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{m,n}^2 \end{bmatrix} \tag{90}$$

mit

$$\sigma_{p,Shape} = \sigma_{p,1} = \sigma_{p,2} = ... = \sigma_{p,n}$$

$$\sigma_{m,Shape} = \sigma_{m,1} = \sigma_{m,2} = ... = \sigma_{m,n}.$$

Die Standardabweichungen $\sigma_{m,Motion}$ und $\sigma_{m,Shape}$ sind indentisch.

4.3.5 Initialisierung der Dynamik

Die Initialisierung kann nach dem zweiten Vorkommen des Objekts erfolgen, da ein Modell mit konstanten Geschwindigkeiten verwendet wird. Der Wert für $\partial\mathbf{t}_k$ ergibt sich aus der ersten diskreten Ableitung der ersten beiden Zentren der Konturen. Da die Änderung der Skalierung ξ_k sehr klein ist, kann diese mit 0 initialisiert werden.

$$\partial\mathbf{t}_k = \mathbf{c}_k - \mathbf{c}_{k-1} \tag{91}$$

$$\xi_k = 0 \tag{92}$$

Die Initialisierung der Kontrollpunkte erfolgt mit $\mathbf{Y}_k$:

$$\mathbf{x}_{Shape,k} = \mathbf{Y}_k. \tag{93}$$

4.3.6 Framework der Objektverfolgung

Für die Objektverfolgung werden zwei Listen verwaltet. Die erste Liste, wird im Folgenden nur noch als Tracklist bezeichnet und beinhaltet alle Objekte, die derzeit verfolgt werden. Die zweite Liste, im Folgenden nur noch Initlist, wird für die Initialisierung der Objekte benötigt. Bei Start der Objektverfolgung sind beide Listen leer. Der Algorithmus ist in Abb. 16 wiedergegeben.

- Für jedes Frame k der Sequenz:
  - Für jedes Objekt der Tracklist:
    1. Prädiziere die Kontur des Objekts
    2. Objektverfolgung siehe nächsten Abschnitt 4.3.7
  - Für jedes Objekt der Initlist:
    1. Vergleiche die Kontur des Objekts mit der Segmentierung (Intersektion der Regionen mit Schwellwert). Wird eine Korrespondenz gefunden, dann initialisiere ein neues Objekt und entferne die Korrespondenz aus der Segmentierung.
  - Lösche die Initlist und kopiere alle übriggebliebenen Regionen aus der Segmentierung in die Initlist.

Abb. 16: Algorithmus der Objektverfolgung (Listenverwaltung)

Nach dem Start der Objektverfolgung werden im ersten Zeitschritt die Regionen der Segmentierung in die Initlist kopiert. Im zweiten Zeitschritt beginnt dann die eigentliche Objektverfolgung, wie sie in Abb. 16 steht.

Um die Korrespondenzen von Frame k mit Frame $k-1$ zu finden, werden die Regionen geschnitten, die Flächen berechnet und die Relation bestimmt:

$$f = \frac{A(S_k \cap S_{k-1})}{A(S_{k-1})}. \tag{94}$$

f gibt das Verhältnis zum vorherigen Frame $k-1$ an. Dieser Faktor muss größer einem Schwellwert $th_{init,contourtrac\,\mathrm{ker}}$ bzw. $th_{track,contourtrac\,\mathrm{ker}}$ sein. Im Fall der Objektverfolgung wird die prädizierte Region verwendet anstelle von S_{k-1}.

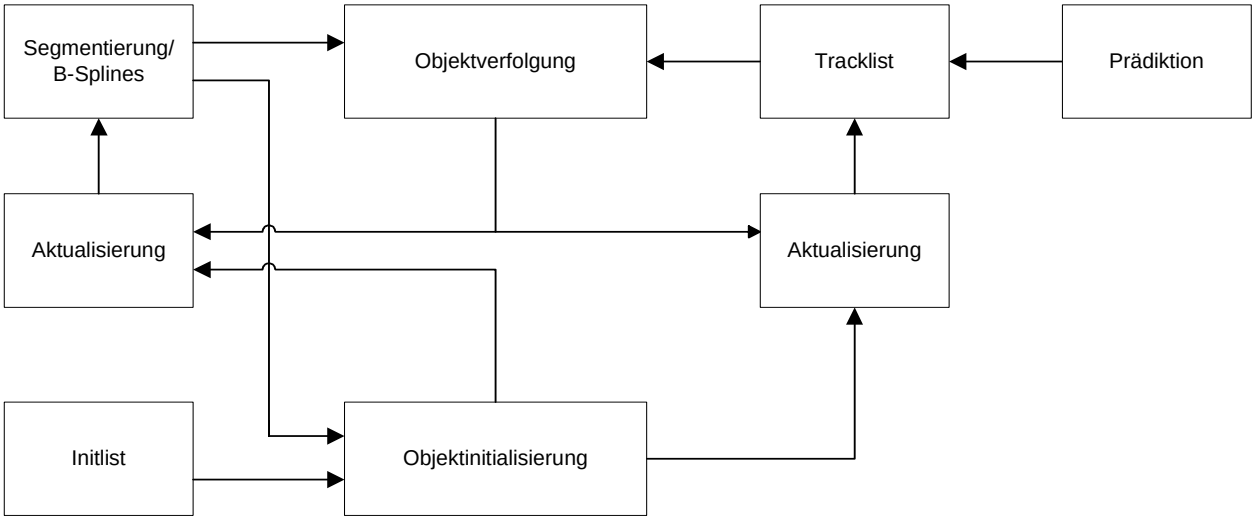


Abb. 17: Framework der Objektverfolgung in 2D

Die Abb. 17 verdeutlicht noch mal die Aktionen, die auf den Listen ausgeführt werden. Die Aktualisierung der „Segmentierung/B-Splines“ ist einfach eine Markierung des B-Splines. Somit wird verhindert, dass das selbe B-Spline mehrmals verwendet wird. Die zweite Aktion aktualisiert die Tracklist. Die Aktualisierung beinhaltet die Initialisierung des Kalman-Filters bzw. ein „Update“ auf die Kalman-Filter. Diese Aktionen werden immer dann ausgeführt, wenn eine Objektinitialisierung durchgeführt wurde bzw. ein Objekt verfolgt werden konnte.

Es müssen für beide Prozesse, der Verfolgung und der Initialisierung, immer zwei Listen miteinander verglichen werden. Somit erhält man eine Komplexität von $O(nm+nl)$ für dieses Framework, mit der Anzahl n der Regionen die von der Segmentierung detektiert wurden, m der Anzahl der Objekte in der Tracklist und l der Anzahl der Regionen in der Initlist.

4.3.7 Algorithmus der Objektverfolgung

Nachdem das Framework, also die Verwaltung der Listen, behandelt wurde, folgt in diesem Abschnitt die Objektverfolgung im Detail. Die Objektverfolgung lässt sich in mehrere Fälle aufspalten. Der erste Fall ist überprüft einfach nur, ob ein Objekt den relevanten Bereich (Region of Interest) verlässt. Der zweite Fall prüft, ob das Objekt ein statisches Objekt oder ein Objekt welches im Vordergrund steht, aber nicht verfolgt werden soll, schneidet. Der dritte Fall wiederholt den zweiten Fall, diesmal aber mit den anderen verfolgten Objekten. Der vierte Fall tritt auf, wenn das verfolgte Objekt mit der Segmentierung aktualisiert werden kann. Und schließlich der letzte Fall tritt ein, wenn das Objekt nicht aktualisiert werden konnte. Die folgende Abb. 18 veranschaulicht den Algorithmus und die Zustände.

- Für jedes Objekt der Tracklist:

  1. Überprüft, ob das Objekt den relevanten Bereich der Objektverfolgung verläßt? Ist dies der Fall, so kann das Objekt aus der Tracklist entfernt werden und mit dem nächsten Objekt verfahren werden.

  2. Schneidet das Objekt irgendein statisches Vordergrundobjekte, dann prädiziere die Trajektorie des Objekts, weise ihm den Status „SO“ zu und verfahre mit dem nächsten Objekt.

  3. Schneidet das Objekt irgendein anderes dynamisches Objekt, dann prädiziere die Trajektorie des Objekts, weise ihm den Status „DO“ zu und verfahre mit dem nächsten Objekt.

  4. Schneidet das Objekt irgendeine Region der Segmentierung, dann aktualisiere die Dynamik mit der gemessenen Kontur und weise als Status „TRACK“ zu. Verfahre mit dem nächsten Objekt.

  5. Wird keine Korrepondenz gefunden, dann markiere das Objekt mit Status „PHANTOM“.

Abb. 18: Algorithmus der Objektverfolgung

Vor jedem Aufruf der Objektverfolgung eines Zeitschritts, wird überprüft, wie lange das Objekt schon prädiziert wurde. Für die Zustände „SO“, „DO“, „PHANTOM“, gibt es Schwellwerte, die angeben, wie lange (d.h. wie viele Frames) ein Objekt prädiziert werden darf. Ist für ein Objekt eine dieser Schwellen überschritten, so wird das Objekt aus der Liste entfernt.

Die Liste der Objektverfolgung enthält nach jedem Zeitschritt den Zustand der Objekte, dieser beinhaltet die Position (Schwerpunkt) in Bildkoordinaten des Objekts (Kontur), sowie die Änderung der Position des Objekts und die Approximation der Kontur als B-Spline. Jedem Objekt wird bei der Initialisierung eine eindeutige Identifikation zugewiesen.

4.4 Objektverfolgung in mehreren Kameras

Die Objektverfolgung in 3D erhält als Eingabe nur die Ergebnisse der Objektverfolgung in den einzelnen Kameraansichten pro Zeitschritt. Die folgende Grafik verdeutlicht das Verfahren (siehe auch [2]).

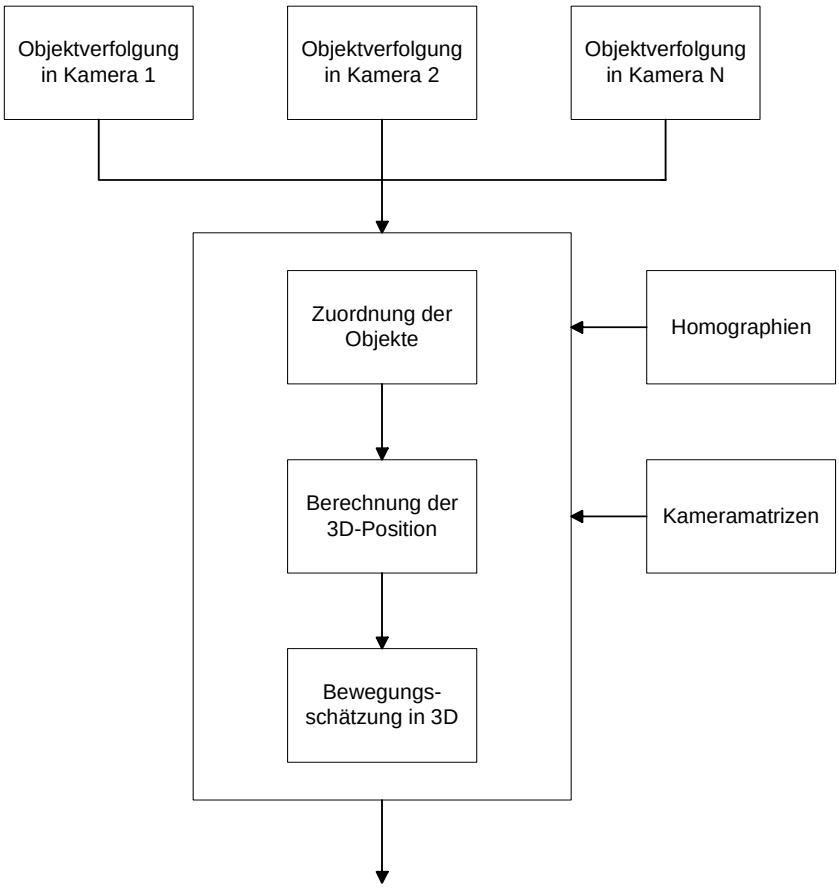


Abb. 19: Blockdiagramm der Objektverfolgung in 3D

Neben den Ergebnissen der Objektverfolgung in den einzelnen Kameraansichten, werden außerdem einige Matrizen benötigt, die vor dem Start bekannt sein müssen. Für die Zuordnung der Objekte werden die 2D Homographien zwischen den einzelnen Ansichten benötigt. Für die Berechnung der 3D Position müssen die Kameramatrizen bekannt sein. Abb. 19 zeigt, dass zuerst die Objekte aus den einzelnen Ansichten einem globalen Objekt zugeordnet werden. Danach erfolgt die Berechnung der 3D Position über ein „Least-Square"-Verfahren und schließlich die Bewegungsschätzung und Objektverfolgung in 3D.

4.4.1 Zuordnung der Objekte mit der 2D Homographie

Jedes Objekt aus den einzelnen Kameraansichten wird mit den Objekten aus den anderen Ansichten verglichen. Für den Vergleich werden die Fehlermatrizen $\mathbf{F}_{ij}$ aller Kamerapaare

berechnet. $\mathbf{H}_{ij}$ gibt die 2D Homographie von Kamera i nach Kamera j an. Für N Kameras erhält man folglich $N \cdot (N-1)$ Homographien. Die Fehlermatrizen werden nach folgender Rechenvorschrift berechnet:

$$\mathbf{F}_{ij} = \mathbf{F}_{ji} = \sum_{k=1}^{L_i} \sum_{l=1}^{L_j} \text{len}((\mathbf{H}_{ij} \cdot \mathbf{c}_{i,k}) - \mathbf{c}_{j,l}) + \text{len}((\mathbf{H}_{ji} \cdot \mathbf{c}_{j,l}) - \mathbf{c}_{i,k}) \tag{95}$$

mit

$0 <= i, j < N \wedge i \neq j$

len() - euklidische Länge des Vektors

$\mathbf{c}_i$ - Zentrum des Objektes aus Kamera i

L_i - Anzahl der Elemente in Liste i.

Die Fehlermatrix gibt den geometrischen Fehler an, der durch Projektion entsteht. Für die Projektion wird das Zentrum der Kontur verwendet. Die Berechnung einer Fehlermatrix hat also eine Komplexität von $O(L_i \cdot L_j)$ und eine Gesamtkomplexität von:

$$O(\frac{N}{2}(N-1) \cdot L \cdot L). \tag{96}$$

Nachdem die Fehlermatrizen berechnet wurden, erfolgt die Zuordnung der Objekte globalen (gemeinsamen) Objekten. Die Zuordnung der Objekte geschieht durch Suchen des kleinsten Fehlers in der Matrix. Eine Korrespondenz wird nur für den Fall erzeugt, falls der Fehler kleiner eines Schwellwertes th_{assign} ist und das gefundene Objekt auch auf das Objekt zurückzeigt. Die korrespondierenden Objekte werden in einer Liste gespeichert. Für die Zuordnung ergibt sich auch eine Komplexität von $O(L \cdot L)$.

4.4.2 Berechnung der 3D Position

Nachdem die Korrespondenzen aufgestellt wurden, wird die 3D Position berechnet. Die Berechnung der 3D Position geschieht durch Rückprojektion der Schwerpunkte der Konturen in Bildkoordinaten nach 3D Weltkoordinaten. Da die Projektion von 2D nach 3D eine Gerade ergibt, muss der Punkt gefunden werden, der den Abstand zu den Geraden minimiert (im Sinne des kleinsten quadratischen Fehlers). Die Anzahl der Geraden ergibt sich aus den gefundenen Korrespondenzen. Gibt es für ein Objekt gar keine Korrespondenz, so erfolgt die Berechnung der Position über die rückprojizierte Gerade des Schwerpunkts und der Flächennormalen der Ebene an dem unteren Rand der „Bounding Box“ [2]. Dafür wird die Homographie von Bild- nach Weltkoordinaten benötigt. Die folgende Abb. 20 veranschaulicht das Prinzip der Berechnung.

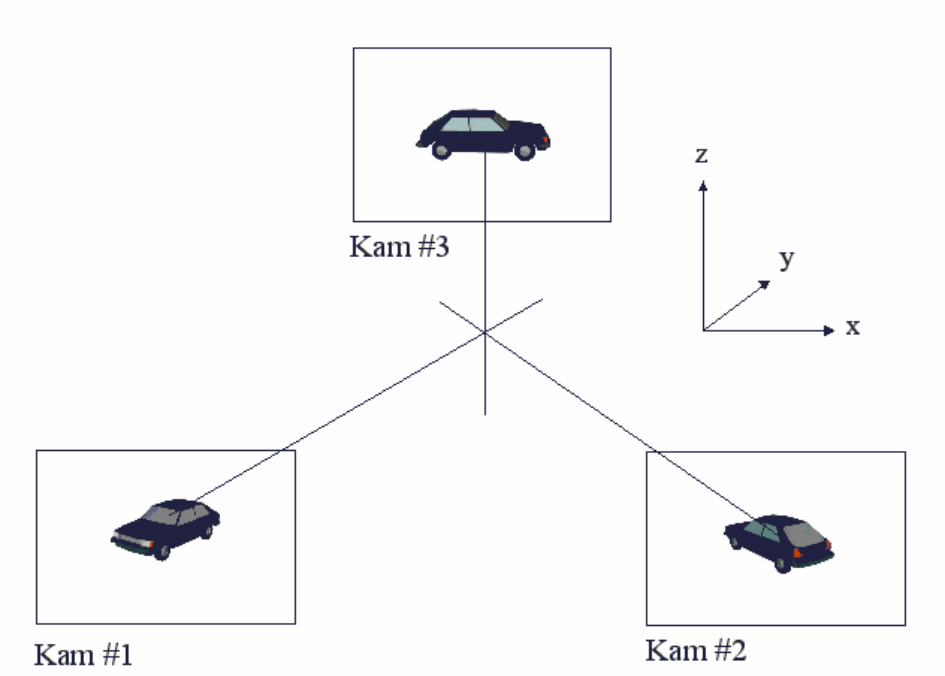


Abb. 20: Berechnung der 3D Position eines Objekts

Für jede Kameraansicht wird das Zentrum der Kontur in den 3D Raum rückprojiziert (siehe Abb. 20). Die Rückprojektion ergibt für jede Kameraansicht eine Gerade im 3D Raum. Der Schnittpunkt der 3 Geraden ergibt die 3D Position des Objekts. Da es sehr unwahrscheinlich ist, dass die Geraden sich genau in einem Punkt schneiden, wird der Punkt berechnet, der alle 3 Abstände zu den Geraden minimiert (im Sinne des kleinsten quadratischen Fehlers).

Die Berechnung der 3D Position erfolgt nach folgendem Algorithmus:

1. Stelle die Gleichungen der Geraden aus den gegebenen Kameramatrizen auf:

   $\mathbf{r}_i(\lambda) = \mathbf{a}_i + \lambda \mathbf{b}_i$. (siehe Abschnitt 3.5 „Kamerageometrie“)

2. Stelle für alle N Gleichungen der Geraden die Matrix **O** nach folgender Rechenvorschrift auf:

$$\mathbf{O} = \begin{bmatrix} \sum_{i=1}^{N} 1 - b_{ix}^{\,2} & \sum_{i=1}^{N} -b_{ix}b_{iy} & \sum_{i=1}^{N} -b_{ix}b_{iz} \\ \sum_{i=1}^{N} -b_{ix}b_{iy} & \sum_{i=1}^{N} 1 - b_{iy}^{\,2} & \sum_{i=1}^{N} -b_{iy}b_{iz} \\ \sum_{i=1}^{N} -b_{ix}b_{iz} & \sum_{i=1}^{N} -b_{iy}b_{iz} & \sum_{i=1}^{N} 1 - b_{iz}^{\,2} \end{bmatrix}$$

3. Stelle für alle N Gleichungen den Vektor $\mathbf{D}$ nach folgender Rechenvorschrift auf:

$$\mathbf{D} = \begin{bmatrix} \sum_{i=1}^{N} a_{ix} - b_{ix}\mathbf{a}_i \cdot \mathbf{b}_i \\ \sum_{i=1}^{N} a_{iy} - b_{iy}\mathbf{a}_i \cdot \mathbf{b}_i \\ \sum_{i=1}^{N} a_{iz} - b_{iz}\mathbf{a}_i \cdot \mathbf{b}_i \end{bmatrix}$$

4. Die Berechnung der 3D Position $\tilde{\mathbf{p}} \in \Re^3$ erfolgt durch Lösung folgenden Gleichungssystems:

$$\tilde{\mathbf{p}} = \mathbf{O}^{-1}\mathbf{D}\,.$$

4.4.3 Bewegungsschätzung in 3D

Die Bewegungsschätzung erfordert einen Kalman-Filter, wobei der Zustand nur aus der Position und der Änderung der Position besteht. Es handelt sich also wiederum um ein Bewegungsmodell konstanter Geschwindigkeit und das lineare Kalman-Filter kann verwendet werden:

$$\mathbf{x}_{Motion3D,k} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T . \tag{97}$$

Die Beobachtung des Prozesses erfolgt über die 3D Position des Objekts:

$$\mathbf{z}_{Motion3D,k} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{98}$$

Aus dem Bewegungsmodell und der Differenzengleichungen des Kalman-Filters:

$$\mathbf{x}_{Motion3D,k+1} = \mathbf{\Lambda}_{Motion3D,k}\mathbf{x}_{Motion3D,k} + \mathbf{w}_k \tag{99}$$

$$\mathbf{z}_{Motion3D,k} = \mathbf{\Phi}_k\mathbf{x}_{Motion3D,k} + \mathbf{v}_k \tag{100}$$

ergeben sich für $\mathbf{\Lambda}_{Motion3D,k}$ und $\mathbf{\Phi}_{Motion3D,k}$ einfache Matrizen:

$$\mathbf{\Lambda}_{Motion3D,k} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{101}$$

$$\mathbf{\Phi}_{Motion3D,k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{102}$$

Die Fehlerstatistik wird von den konstanten Kovarianzmatrizen beschrieben:

$$\mathbf{\Omega}_{Motion3D} = \begin{bmatrix} \sigma_p^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_p^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_p^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_p^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_p^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_p^2 \end{bmatrix} \tag{103}$$

$$\mathbf{\Xi}_{Motion3D} = \begin{bmatrix} \sigma_m^2 & 0 & 0 \\ 0 & \sigma_m^2 & 0 \\ 0 & 0 & \sigma_m^2 \end{bmatrix}. \tag{104}$$

4.4.4 Framework der Objektverfolgung in 3D

Das Framework ist identisch mit dem der Objektverfolgung in 2D (siehe Abschnitt 4.3.6). Es müssen wieder zwei Listen verwaltet werden, eine für die Initialisierung und die andere für die Objektverfolgung. Um ein neues Objekt initialisieren zu können, muss das Objekt zweimal vorgekommen sein, da auch hier ein Bewegungsmodell konstanter Geschwindigkeit verwendet wurde.
Das Finden von Korrespondenzen zwischen den Zeitschritten erfolgt über die euklidische Distanz der 3D Positionen. Für die Initialisierung und Objektverfolgung wird die geringste Distanz ermittelt. Ist diese Distanz kleiner als der Schwellwert $th_{multitrac\,ker,init}$ bzw. $th_{multitrac\,ker,track}$ so kann eine Aktualisierung der Objekte und Listen erfolgen.

4.5 3D Rekonstruktion

Die Visualisierung der dynamischen Objekte erfolgt durch Projektion der Texturen auf 3D Modelle. Die Auswahl der 3D Objekte aus einer Datenbank kann z.B. durch Vergleich der Konturen aus der Objektverfolgung in 2D erfolgen. Die Rekonstruktion verlangt eine volle Kalibrierung der Kameras. Außerdem ist bekannt, dass die Objekte sich ausschließlich auf einer gemeinsamen Ebene bewegen. Die Position und Ausrichtung der Objekte erhält man aus der Objektverfolgung in 3D.

4.5.1 Übersicht

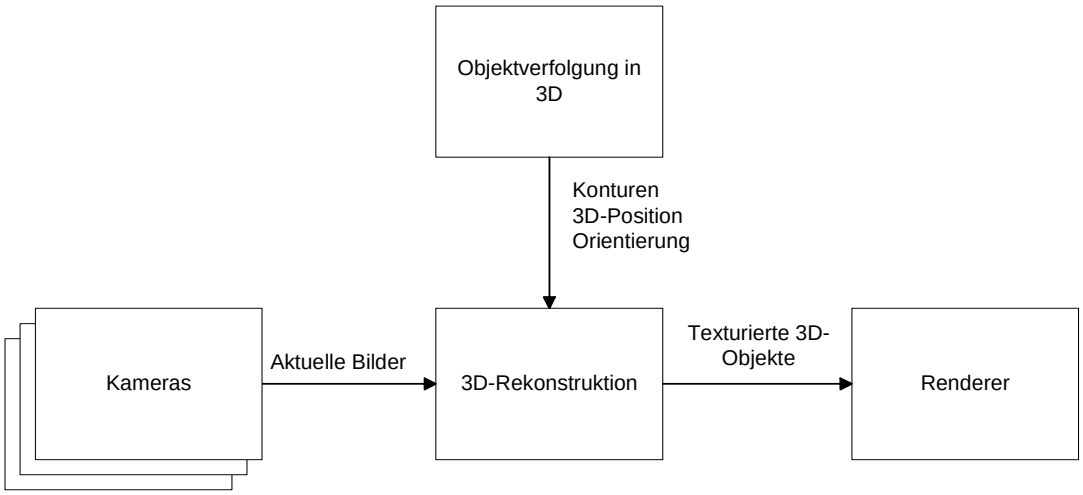


Abb. 21: Modul-Abhängigkeit der 3D Rekonstruktion

Das Blockdiagramm in Abb. 21 veranschaulicht die Abhängigkeiten des Moduls. Wichtig sind hier vor allem die Konturen der Objekte in den einzelnen Kameraansichten, die 3D Position der Objekte in Weltkoordinaten und die Ausrichtung (d.h. Rotation auf der Ebene) des Objekts. Die Kameras liefern die aktuellen Bilder der einzelnen Ansichten. Die Visualisierung der Szene erfolgt schließlich über den Renderer.

Für jedes neue Objekt aus der Objektverfolgung muss ein neues 3D Objekt erzeugt werden. Die Idee ist es, die extrahierten Texturen unter gleichen Bedingungen auf das 3D Modell zu projizieren. Die Projektion wird in diesem Fall zu einer einfacheren affinen Transformation. Es reicht, das 3D Modell zu positionieren, zu skalieren und zu orientieren. Im nächsten Schritt werden die sichtbaren Polygone des 3D Modells bestimmt. Durch Projektion der Texturen auf das 3D Modell ergeben sich die Texturkoordinaten.
Die Auswahl des 3D Modells, wird in dieser Diplomarbeit nicht betrachtet. Es wird für die Rekonstruktion angenommen, dass das Modell bekannt ist. Nachdem das 3D Objekt einmal initialisiert wurde, d.h. die Texturkoordinaten berechnet wurden, besteht die Aktualisierung des 3D Modells in der Szene nur noch aus Orientierung und Positionierung.

4.5.2 Initialisierung eines 3D Objekts

In Abb. 22 ist ein typisches 3D Modell eines Fahrzeugs aus der Datenbasis zu sehen:

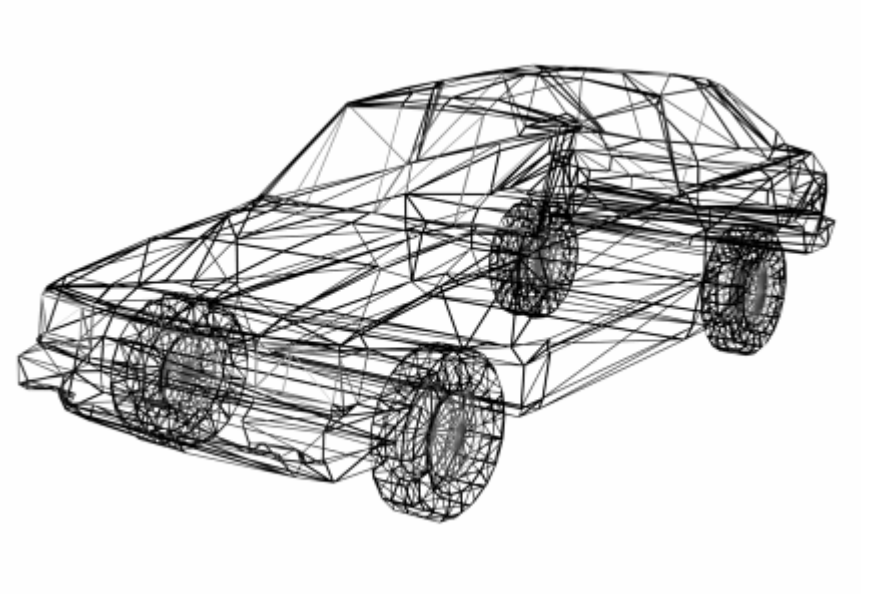


Abb. 22: 3D Modell eines Pkws

Der erste Schritt der Initialisierung besteht darin, die Skalierung des Objektes zu bestimmen. Das geschieht durch Vergleich der Projektion des 3D Modells nach Bildkoordinaten mit den Konturen der Segmentierung. Zuvor muss aber noch das 3D Modell in die Szene gesetzt und auf der Ebene ausgerichtet werden. Damit nur sichtbare Teile des 3D Modells projiziert werden, muss überprüft werden, ob die Fläche überhaupt für die Kamera sichtbar ist. Ein einfaches Verfahren das über die Flächennormale **N** und der Kameraposition **C** bestimmt, ob eine Fläche sichtbar ist, nennt sich Backface-Culling. Die Berechnung der Flächennormalen **N** ist in der nächsten Abb. 23 veranschaulicht.

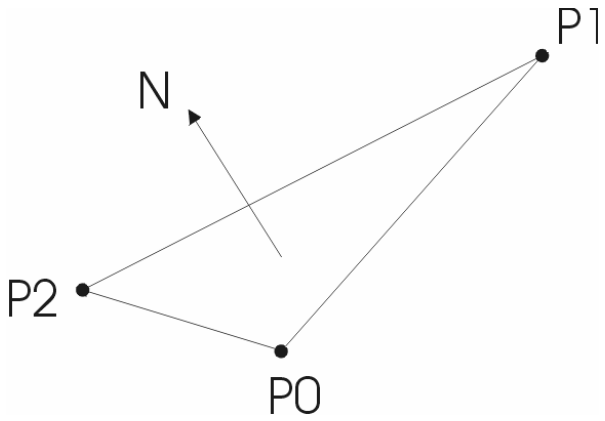


Abb. 23: Bestimmung der Flächennormale

Sind die Eckkoordinaten des Polygons gegen den Uhrzeigersinn angeordnet, also mathematisch positiv (wie in Abb. 23), so berechnet sich die Flächennormale **N** als:

$$\mathbf{N} = (\mathbf{P}_0 - \mathbf{P}_1) \times (\mathbf{P}_0 - \mathbf{P}_2) . \tag{105}$$

Aus der Flächennormalen **N** und der Kameraposition **C** lässt sich dann sehr einfach bestimmen, ob ein Objekt sichtbar ist. Dieses Verfahren ist standardmäßig auf den heutigen Grafikkarten implementiert um den Rennerprozess zu beschleunigen [21].

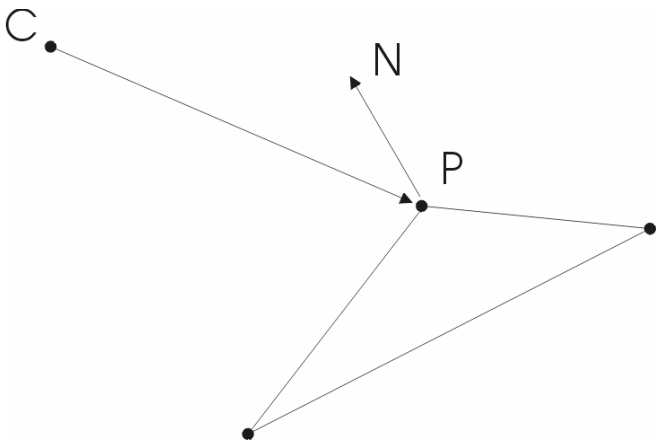


Abb. 24: Backface-Culling

Das Polygon ist sichtbar falls (siehe Abb. 24)

$$(\mathbf{p} - \mathbf{C}) \cdot \mathbf{N} < 0 \tag{106}$$

mit

$\mathbf{p} \in \Re^3$ gilt.

Ein Vergleich der projizierten Kontur mit der segmentierten Kontur liefert ein Skalar, dieser ist ausreichend, da das 3D Modell die Form bis auf eine Skalierung hinreichend beschreibt. Damit die beiden Konturen vergleichbar sind, muss von beiden die „Bounding Box“ bestimmt werden. Die Bounding Box ist das kleinste Rechteck, dass die Kontur vollständig umschließt. Die Ausrichtung des Rechtecks erfolgt an der x- bzw. y-Achse. Die Bounding Box ist so definiert, dass sie aus einem Schwerpunkt **c** und der Größe des Rechtecks besteht (siehe folgende Abb. 25).

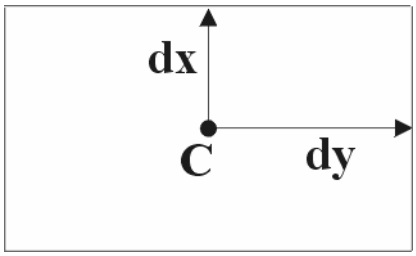


Abb. 25: Definition der Bounding Box

Für die Berechnung der Skalierung ergibt sich:

$$a_i = \frac{A(t_i)}{A(s_i)} \tag{107}$$

mit

- $A(x)$ ist die Fläche einer Bounding Box x
- t_i ist die Bounding Box der Objektverfolgung
- s_i ist die Bounding Box des projizierten 3D-Modells

Aufgrund von Segmentierungsfehlern und ungenaue Kameramatrizen können sich für mehrere Ansichten die Skalierungsfaktoren a_i unterscheiden. Der entgültige Faktor ergibt sich dann aus dem Mittelwert der Einzelfaktoren:

$$a = \frac{1}{N}\sum_{i=1}^{N} a_i \ . \tag{108}$$

Das Objekt wird auf den ermittelten Wert a skaliert. Die nächste Abb. 26 zeigt die Projektion des 3D Modells aus Abb. 22 mit Backface-Culling.

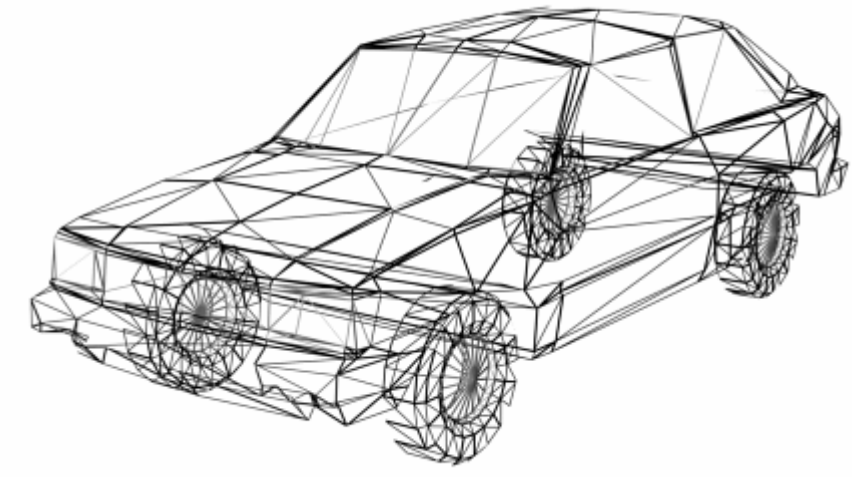


Abb. 26: Projektion mit Backface-Culling

Als nächster Schritt folgt die Bestimmung der Texturkoordinaten. Dazu wird wieder die Projektion des 3D Modells auf die Bildebene berechnet, um wieder die Bounding Box bestimmen zu können. Da durch die Mittelung der Skalierungsfaktoren, die Projektion kleiner bzw. größer als die Textur sein kann, wird die Textur auf die ermittelte Bounding Box gestaucht bzw. gestreckt.

Um die Textur skalieren zu können, werden die projizierten Koordinaten in den Ursprung verschoben (Subtraktion des Schwerpunkts der projizierten Bounding Box). Im Anschluss werden die Koordinaten auf die richtige Größe skaliert. Die Größe ergibt sich durch Division der segmentierten Bounding Box mit der projizierten Bounding Box. Die Koordinaten müssen dann nur noch zurück in das Zentrum der segmentierten Bounding Box geschoben werden. Entspricht die Größe der Textur genau der Größe der Bounding Box der Segmentierung, dann können die Koordinaten direkt übernommen werden. Ansonsten muss noch mal eine Skalierung auf die Größe der Textur erfolgen.

$\mathbf{p}$	-	projizierte Koordinate
$\mathbf{c}_p$	-	Schwerpunkt der projizierten Bounding Box
dx, dy	-	Größe der Bounding Box
w, h	-	Weite bzw. Höhe der Textur

Die Berechnung von u (u ist die Texturkoordinate in x-Richtung) lässt sich schreiben als:

$$u = \frac{(\mathbf{p} - \mathbf{c}_p)\frac{dx_s}{dx_p} + dx_s}{w} \tag{109}$$

Die Berechnung von v erfolgt analog.

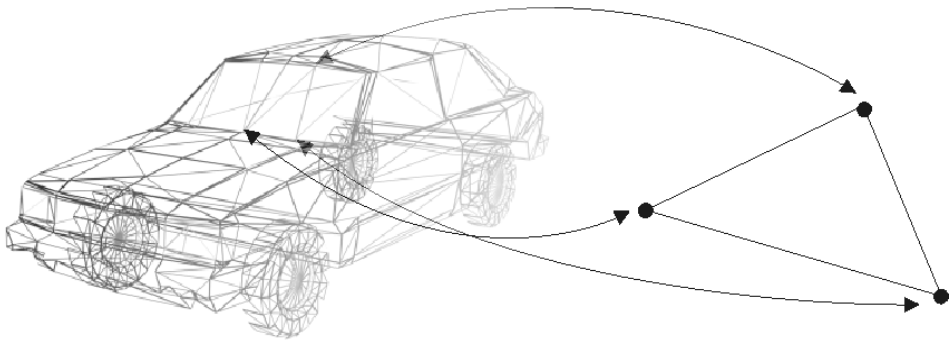


Abb. 27: Texturmapping

Abb. 27 veranschaulicht das Prinzip des Texturmappings. Für jedes Dreieck des 3D-Modells existieren drei Texturkoordinaten. Dadurch das nur 3 Punktkorrespondenzen zur Verfügung stehen, handelt es sich bei dem Texturmapping um eine affine Transformation.

In Abb. 28 ist noch mal das Gesamtverfahren der 3D Rekonstruktion als Blockschaltbild dargestellt.

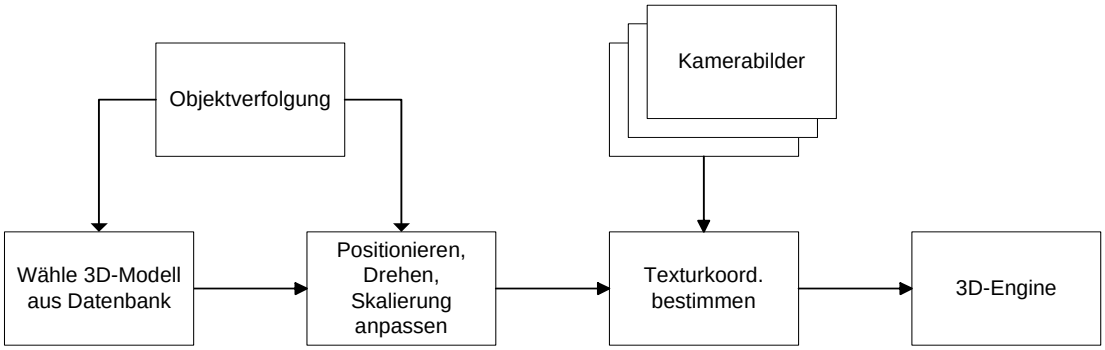


Abb. 28: Blockdiagramm der Initialisierung eines 3D-Objekts

Die Komplexität hängt von der Anzahl der Polygone des 3D Modells ab. Ist N die Anzahl der Polygone, so ergibt sich die Komplexität für die Initialisierung zu:

$$O(3N + 3N + 3N)\,. \tag{110}$$

Der erste Term steht für die Komplexität der Positionierung und Orientierung. Der zweite Term für die Komplexität der Skalierung des 3D Modells und der letzte Term für das Zuordnen der Texturkoordinaten. Es ist zu beachten, dass die Polygone hier durch Dreiecke repräsentiert werden.

4.5.3 Aktualisierung der Position und Orientierung des 3D Objekts

Nachdem das 3D Objekt initialisiert wurde, muss die Trajektorie des Objekts aktualisiert werden. Die neue Position des Objekts ergibt sich aus einer Translation um die Differenz zur alten Position. Die Orientierung berechnet sich auch aus der Differenz zur alten Position, unter der Annahme, dass die Fahrzeuge sich ausschließlich vorwärts bewegen.

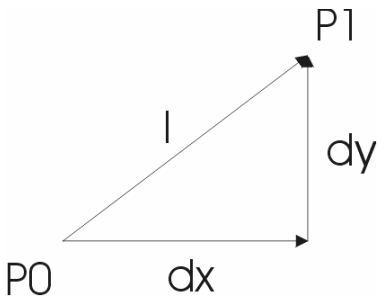


Abb. 29: Bestimmung der Orientierung auf der Ebene (z=0)

Abb. 29 verdeutlicht das Prinzip. Das Objekt hat sich von $\mathbf{P}_0$ nach $\mathbf{P}_1$ bewegt. Die Translation auf der Ebene ergibt sich dann aus dx und dy. Für die Rotation um die z-Achse im 3D Raum gilt:

$$\mathbf{R}_z = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{111}$$

mit α dem Winkel an $\mathbf{P}_0$. Nach Einsetzen der Definitionen des Sinus bzw. Kosinus ergibt sich:

$$\mathbf{R}_z = \begin{bmatrix} \frac{dx}{l} & -\frac{dy}{l} & 0 \\ \frac{dy}{l} & \frac{dx}{l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{mit } l = \sqrt{dx^2 + dy^2}\ . \tag{112}$$

5 Experimentelle Ergebnisse

Die einzelnen Module wurden getrennt anhand synthetischer bzw. realer Daten getestet. Die Verkehrsdaten wurden mit zwei Kameras (Sony DFW-V500) von mir und dem Kollegen Patrick Voigt aufgenommen. Beide Kameras verwendeten das 1/2“-Zoom-Objektiv H6Z810 der Firma Cosmicar/Pentax. Die Synchronisation wurde anschließend manuell durchgeführt, indem eine der beiden Sequenzen dementsprechend verzögert wurde. Die Sequenzen wurden mit einer Auflösung von 640x480 bei 7.5Hz aufgenommen und in YUV420 abgespeichert.

5.1 Kalman-Filter

Es wird in dem folgenden Abschnitt die Implementierung des Kalman-Filters auf Korrektheit geprüft. Für die Testreihe wurde ein einfaches 1-dimensionales Problem betrachtet. Es wurde überprüft ob die Spezifikation des Kalman-Filters mit der Implementierung übereinstimmt.

5.1.1 Untersuchung des Kalman-Filters anhand synthetisch erzeugter Daten

Es wird untersucht ob sich das Filter, den theoretischen Grundlagen entsprechend, für unterschiedliche Kovarianzmatrizen des Messfehlers $\boldsymbol{\Xi}$ richtig verhält.

Da hier nur der stationäre Fall interessiert sind die Matrizen $\boldsymbol{\Lambda}, \boldsymbol{\Phi}$ und $\boldsymbol{\Xi}$ konstant. Der Prozess ist fehlerfrei, da nur eine Konstante geschätzt werden soll. Des weiteren handelt es sich um einen 1-dimensionalen Prozess, dadurch werden die Matrizen durch ihre skalaren Korrespondenzen ersetzt. Die Systemgleichungen für die Versuche reduzieren sich auf:

$$x_{k+1} = \Lambda \cdot x_k \tag{113}$$

$$z_k = \Phi \cdot x_k + v_k \tag{114}$$

Die Skalare Λ und Φ haben beide den konstanten Wert 1. Die Zufallsvariable v ist durch die Konvarianzmatrix Ξ bestimmt.

Für die folgenden drei Versuche wurde eine Konstante $z = -1{,}6234$ mit einem weißen Rauschen überlagert. Das Rauschen hat eine Standardabweichung von $\sigma = 0{,}1$. Der Fehler der durch das Rauschen hinzugefügt wird, ist der Messfehler. Die Werte des Rauschens wurden einmal bestimmt und für jeden Versuch wiederverwendet, um einen Vergleich der Ergebnisse aufstellen zu können.
Das Filter benötigt einen Startzustand in Form einer Fehlerkovarianz Θ_0 der Prädiktion und eines Zustands X_0 der Schätzung. Θ_0 wurde auf den Wert 1 gesetzt. Es ist dabei egal welcher Wert genommen wird, er muss sich nur von 0 unterscheiden, da ansonsten die

Messung immer korrekt wäre. Die initiale Prädiktion X_0 erhielt den Wert 0 - auch hier spielt dieser Wert keine Rolle, da nur das Verhalten des Filters getestet werden soll.

5.1.2 Versuchsreihe

Der erste Versuch zeigt die Schätzung für $\Xi = \sigma^2 = 0{,}01$. Dieser Wert entspricht genau der Kovarianz des Fehlers und sollte somit das beste Ergebnis aufweisen.

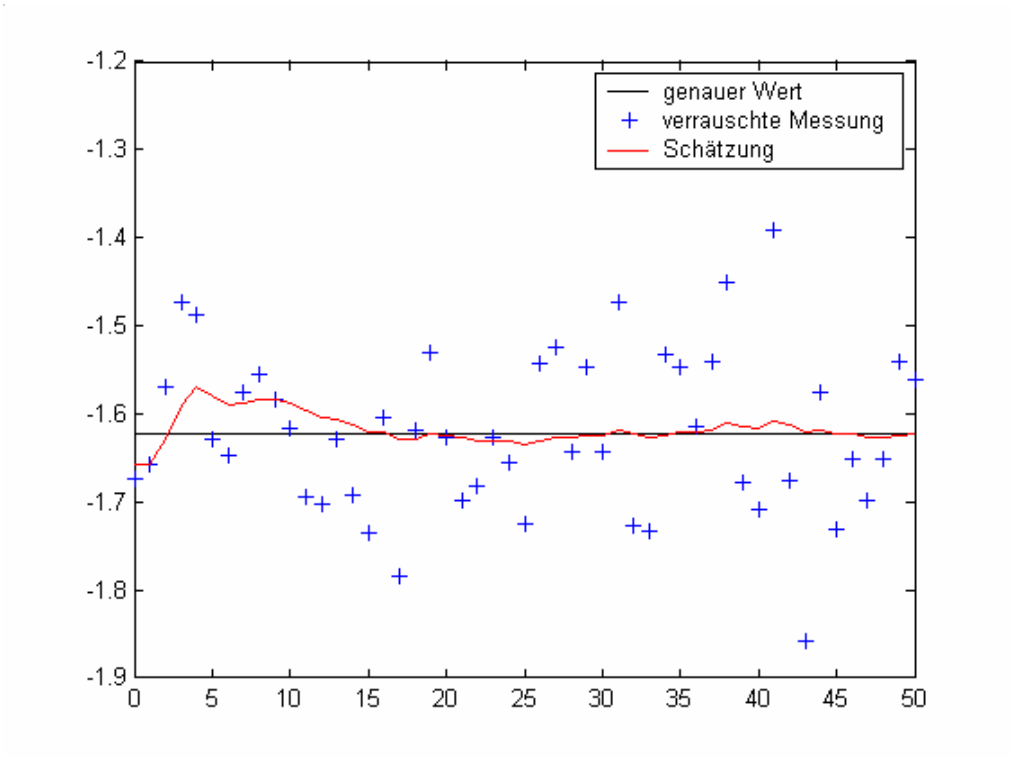


Abb. 30: Schätzung für Ξ =0,01

Wie in Abb. 30 zu erkennen ist, schwingt sich das Kalman-Filter schnell auf den genauen Wert für z ein. Für den zweiten Versuch wurde die Kovarianz des Messfehlers erhöht. Es wurde eine Kovarianz von $\Xi = 100 \cdot \sigma^2 = 1$ gewählt. Alle anderen Parameter des Kalman-Filters wurden nicht verändert. Nach den theoretischen Grundlagen sollte der Zustand x des Kalman-Filters langsamer auf die Messung reagieren. Die folgende Abb. 30 veranschaulicht dieses Verhalten des Filters.

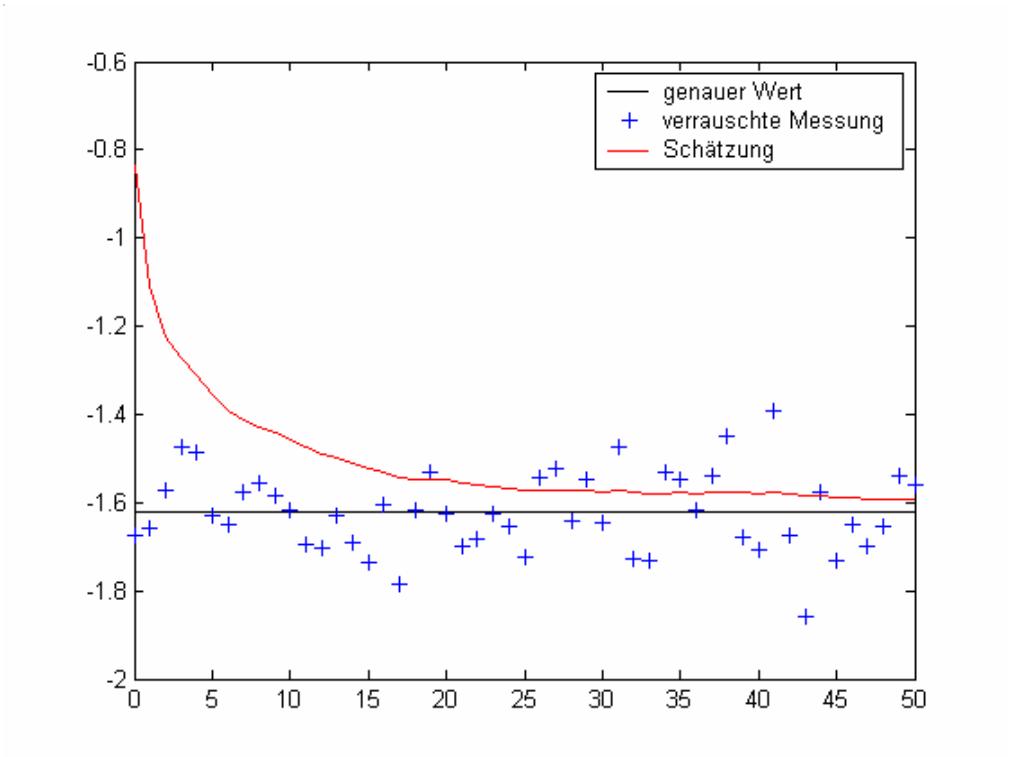


Abb. 31: Schätzung für Ξ =1

In Abb. 31 erkennt man, dass die Messwerte weniger gewichtet werden und der Zustand x mehr. Da der Anfangszustand X_0 auf 0 gesetzt wurde, strebt der geschätzte Zustand langsam von oben nach $z = -1{,}6234$. Der dritte Versuch wurde mit einer kleineren Kovarianz Ξ durchgeführt. Für die Kovarianz wurde $\Xi = \sigma^2 / 100 = 0{,}0001$ gewählt.

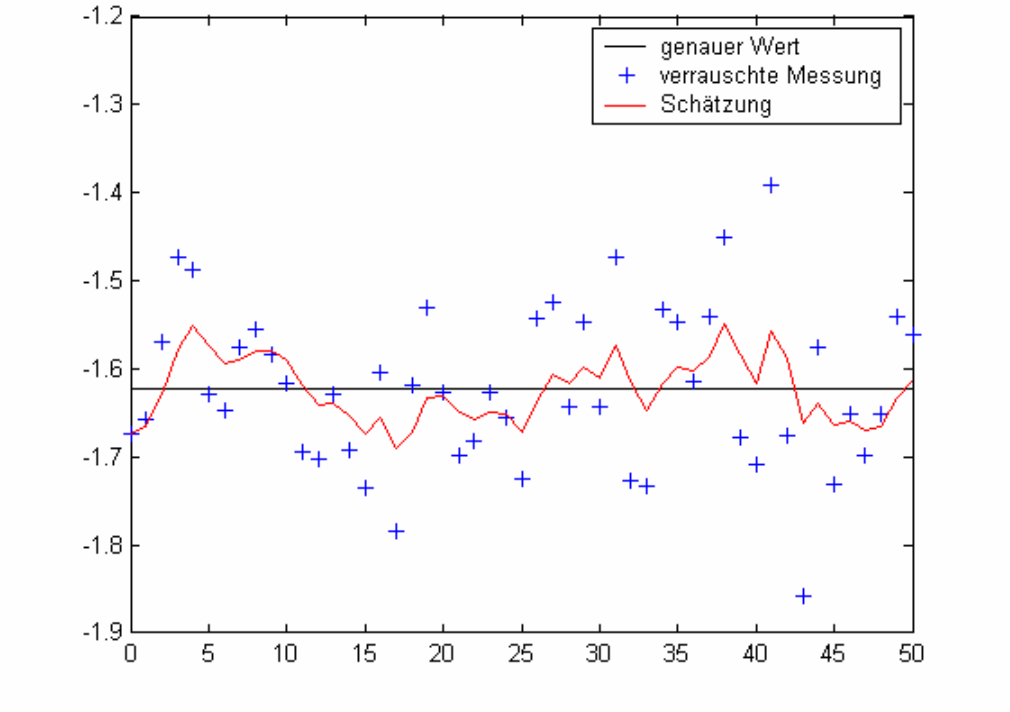


Abb. 32: Schätzung für Ξ =0,0001

Aus der Theorie zum Kalman-Filter lässt sich wieder das Verhalten vorhersagen. Diesmal wird sich das Filter schneller auf die Messungen einschwingen. Abb. 32 veranschaulicht das Verhalten des Kalman-Filters für eine kleine Kovarianz Ξ. Jede Aktualisierung des Filters durch die Messdaten hat einen starken Einfluss auf den geschätzten Zustand x. Dadurch ergibt sich ein Verlauf des geschätzten Zustands x wie ihn Abb. 32 zu sehen.

5.1.3 Ergebnis der Versuchsreihe

Die drei Abbildungen 30-32 zeigen wie sich der geschätzte Zustand x des Kalman-Filters verhält, wenn Ξ variiert. Das Filter weist die besten Ergebnisse auf, falls $\Xi = \sigma^2$ gilt. Durch Variation der Kovarianz Ξ des Messfehlers um den Faktor 100 konnte die Theorie bestätigt werden. Wurde $\Xi < \sigma^2$ gewählt, so wurde die Messung als wahrer angenommen, als sie in Wirklichkeit ist und die Kurve folgte vom Verlauf der gemessenen Werte. Im umgekehrten Fall $\Xi > \sigma^2$ wurde der Messung weniger vertraut und die Kurve näherte sich nur langsam dem genauen Wert von z an. Es lässt sich somit sagen, dass das Kalman-Filter korrekt arbeitet und kann deshalb für die folgenden Module verwendet werden.

5.2 Segmentierung

Es handelt sich hier um eine Implementierung des Softwarepakets Halcon der Firma MVTec [16]. Im folgenden werden reale Sequenzen für die Untersuchungen verwendet. Die Segmentierung hat die Aufgabe aus rohem Bildmaterial Objekte bzw. Konturen dieser Objekte zu extrahieren. Sie dienen als Eingabe für die Segmentierung aber auch für die 3D Rekonstruktion. Es ist deshalb sehr wichtig eine „gute“ Segmentierung zu haben. Im folgenden Abschnitt wird die Implementierung des Hintergrundschätzers des Softwarepakets Halcon auf seine Segmentierungsqualität in realen Sequenzen untersucht.

5.2.1 Initialisierung des Hintergrundschätzers

Die Initialisierung des Hintergrundschätzers und der Nachverarbeitung erfolgt mit den im Abschnitt 4.2 erläuterten Parametern (siehe auch Anhang B). In Anhang A sind die gewählten Parameter aufgelistet. Die Parameter wurden experimentell ermittelt und lieferten für den normalen Betrieb gute Ergebnisse. Der Hintergrundschätzer erhält als Startwert das Hintergrundbild, die sogenannte „leere Szene“. Die folgende Abb. 33 zeigt die leere Szene für die Sequenz „Ernst-Reuter-Platz“.

Abb. 33: Initialisierung des Hintergrundschätzers mit der leeren Szene

Das Hintergrundbild wurde auch mit dem Hintergrundschätzer bestimmt. Für die Bestimmung des Hintergrundes wurden die Parameter so gesetzt, dass die Segmentierung nur bewegte Objekte detektiert (siehe auch [16]). Für die Erstellung wurden 200 Frames der Sequenz benutzt. Wie zu erkennen ist, beinhaltet das Hintergrundbild noch Artefakte. Der Grund dafür sind Objekte, die sich während dieser Generierungsphase nur gering bewegt haben oder gar nicht. Eine größere Anzahl von Frames für die Filterung könnte das Ergebnis

verbessern. Die Schätzung des Hintergrundes erfolgt in den folgenden Abschnitten nur mit Grauwert-Bildern. Um die Segmentierung auf RGB-Bildern anzuwenden, muss der Hintergrundschätzer für jeden Kanal initialisiert werden.

5.2.2 Untersuchung an einer realen Verkehrsszene

Abb. 34: Frame #220 der Sequenz „Ernst-Reuter-Platz“

Nachdem der Hintergrundschätzer initialisiert wurde, erfolgt jetzt die Schätzung mit dem Frame #220 der Sequenz „Ernst-Reuter-Platz“ (siehe Abb. 34).

Abb. 35: Segmentierungsergebnis des Hintergrundschätzers (Frame #220)

Abb. 35 zeigt das reine Segmentierungsergebnis des Hintergrundschätzers ohne Nachverarbeitung. Der Hintergrundschätzer liefert als Ergebnis eine binäre Maske. Die weißen Punkte stellen die Vordergrundregionen dar. Aufgabe der Nachverarbeitung ist es nun aus der binären Maske Konturen der Vordergrundobjekte zu extrahieren. Als erstes werden die Regionen konnektiert. Die Konnektierung führt dazu, dass benachbarte Pixel zu einer Region verschmelzen. Jetzt lassen sich die kleinen Störungen herausfiltern, durch Anwendung des Flächen-Filters. Alle übriggebliebenen Regionen gehören zum Vordergrund und werden in konvexe Regionen konvertiert. Die folgende Abb. 36 zeigt den gleichen Frame #220 mit Nachverarbeitung.

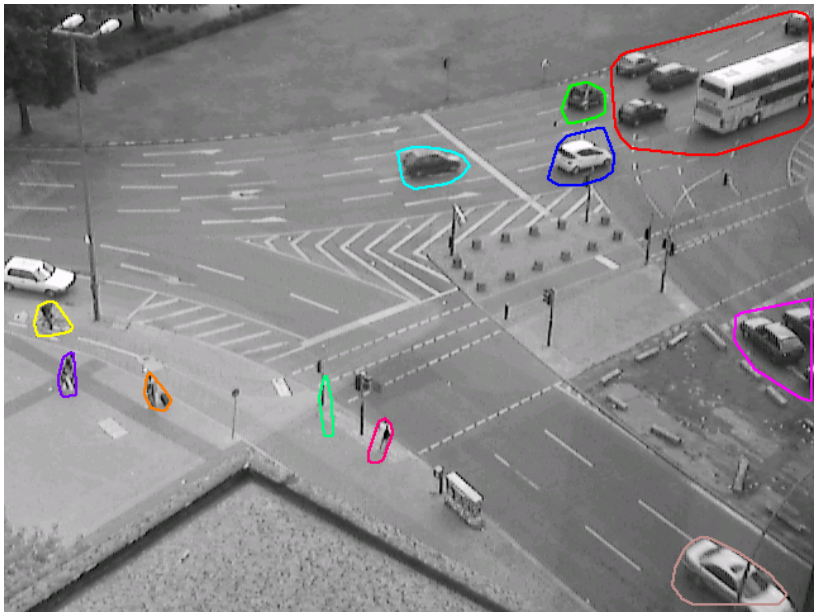


Abb. 36: Frame #220 mit Nachverarbeitung

Die folgende Abb. 37 zeigt die Objektsegmentierung nach weiteren 13 Frames.

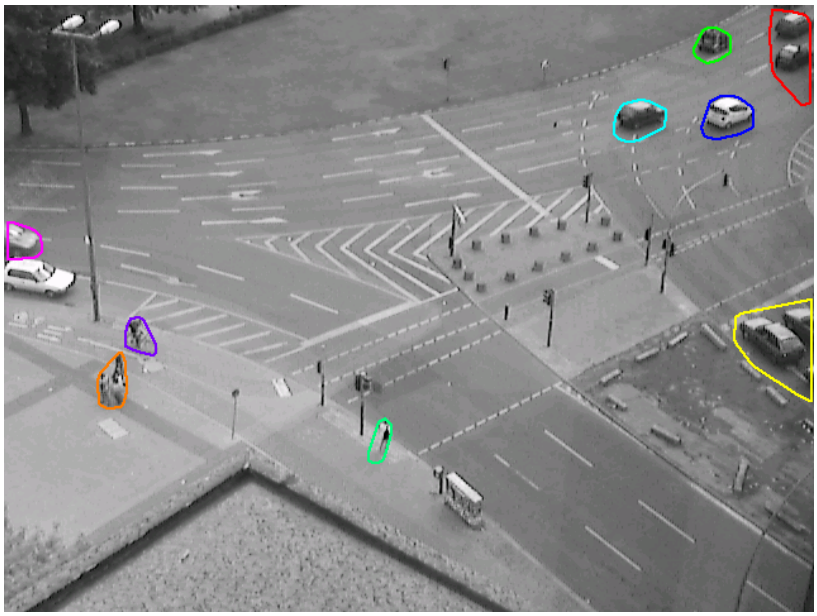


Abb. 37: Frame #233 mit Nachverarbeitung

In den Abb. 36 und 37 ist zu erkennen, dass durch Verdeckungen von Objekten untereinander und durch Schatten Fehler in den Konturen auftreten können. Diese Clusterbildung durch Okklusion lässt sich im Allgemeinen aber nicht vermeiden, wenn man den Standort und die Blickrichtung der Kameras willkürlich wählen will.

Die nächste Abb. 38 soll zeigen wie sich der Hintergrundschätzer verhält, wenn ein plötzliches Aufbrechen einer Wolkendecke zu einer starken Änderung der Lichtverhältnisse führt.

Abb. 38: Objektsegmentierung des Frames #1183 mit starker Änderung der Lichtverhältnisse

Die Fahrzeuge standen vor dem Aufbruch der Wolkendecke still und der Hintergrundschätzer war eingeschwungen. Dadurch das sich die Lichtverhältnisse geändert haben, und die Autos noch standen, konnte keine Aussage über den Hintergrund getroffen werden. Nachdem die Fahrzeuge losgefahren sind, wurden die freigelegten Regionen in den Vordergrund extrahiert. Das führte dann schließlich zu einer falschen binären Maske. In Abb. 38 ziehen die Fahrzeuge einen Schweif hinterher. Dieser Effekt ist auch an den fahrenden Pkws erkennbar, hier ist der Schweif aber kürzer. Durch Erhöhung des Verstärkungsfaktors für die Hintergrundadaption würde sich dieser Schweif verkürzen, mit dem Nachteil, dass stillstehende Fahrzeuge schneller in den Hintergrund adaptiert würden. Um vernünftige Ergebnisse zu erhalten muss also ein Kompromiss eingegangen werden. Die hier verwendeten Werte für die Parameter lieferten im normalen Betrieb gute Ergebnisse. Die nächste Abb. 39 veranschaulicht die Schätzung des Hintergrunds nach weiteren 10 Frames.

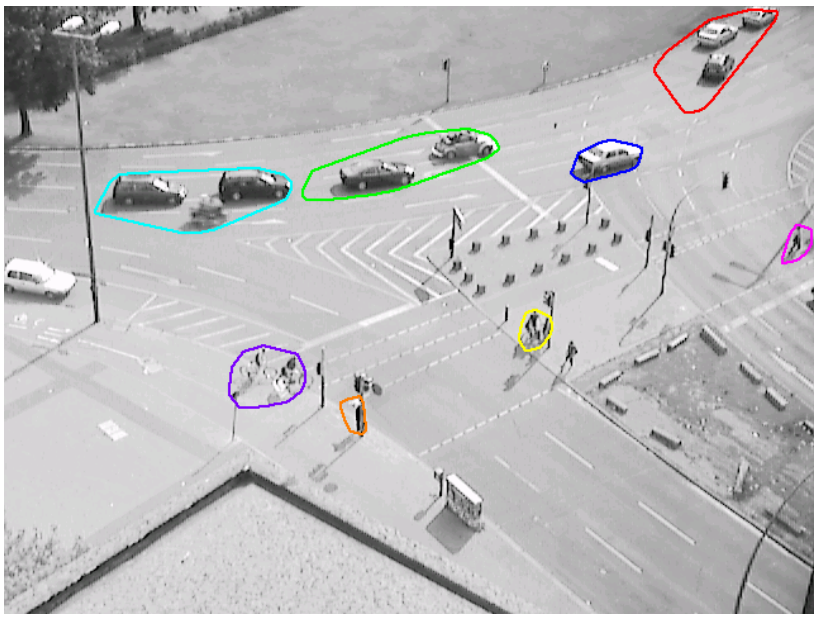


Abb. 39: Objektsegmentierung des Frames #1193, der Hintergrundschätzer schwingt sich ein

Abb. 39 zeigt, dass sich der Hintergrundschätzer langsam auf die Beleuchtungsdynamik eingeschwungen hat. Der Schweif der noch in Abb. 38 an dem weißen Fahrzeug war, ist verschwunden.

5.3 Approximation konvexer Konturen mit kubischen B-Splines

Die folgenden Abbildungen veranschaulichen die Approximation konvexer Konturen durch uniforme kubische B-Splines. Abb. 40 zeigt die konvexen Konturen der Objekte, wie sie von der Objektsegmentierung kommen.

Abb. 40: Konvexe Konturen der segmentierten Objekte des Frames #160

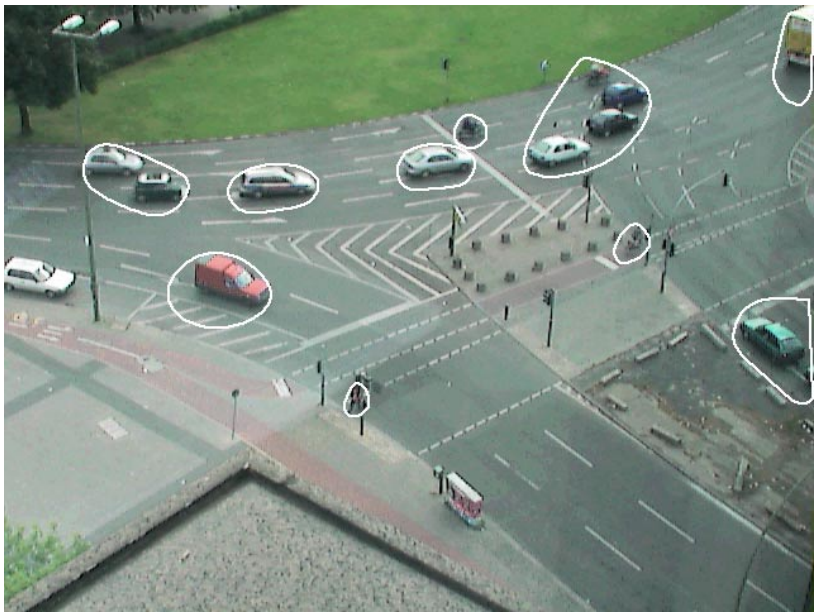


Abb. 41: Approximation durch B-Splines mit 9 Kontrollpunkten (Frame #160)

Die Abb. 41 zeigt die Approximation der Konturen. Man erkennt das die Konturen rund geworden sind. Eine Approximation hat natürlich zur Folge, dass die Konturen ungenauer werden.

Im nächsten Bild (Abb. 42) ist die minimale Anzahl von 6 Kontrollpunkten verwendet worden. Je größer die Anzahl der Kontrollpunkte, um so besser wird natürlich auch die Approximation. Andererseits werden aber auch die Objekte, die verfolgt werden können, durch die Bedingung $M - 3 \leq N$ beschränkt, da die Approximation über die Stützpunkte der konvexen Kontur erfolgt.

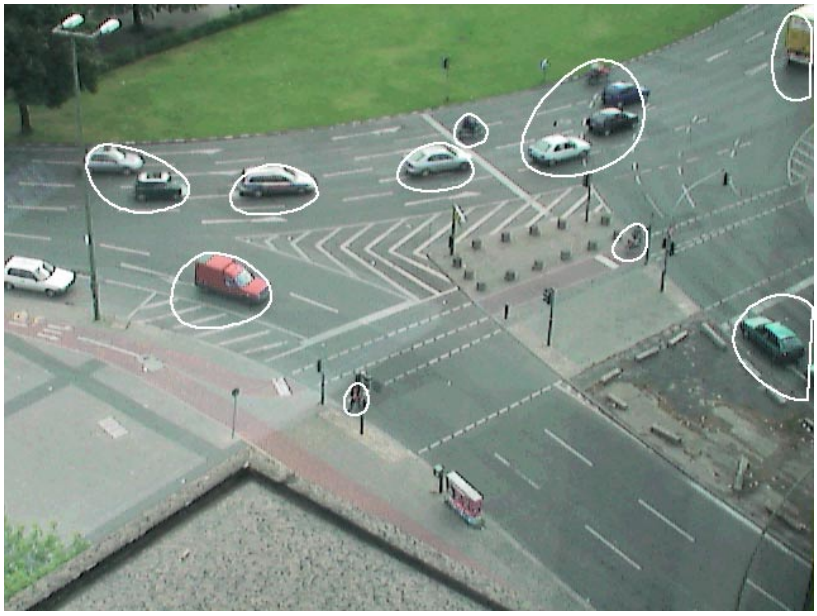


Abb. 42: Approximation durch B-Splines mit 6 Kontrollpunkten (Frame #160)

Abb. 43: Approximation durch B-Splines mit 12 Kontrollpunkten (Frame #160)

Abb. 43 zeigt die gleiche Szene noch mal mit 12 Kontrollpunkten. Es ist zu erkennen, dass die approximierten Konturen sich dem Original (Abb. 40) immer mehr angleichen.

5.4 Objektverfolgung mit Kalman Filtern in 2D

In Anhang A befinden sich alle Parameter die zur Initialisierung des Moduls notwendig sind. Alle Parameter wurden experimentell ermittelt und ergaben für die Sequenz „Ernst-Reuter-Platz“ gute Ergebnisse.

5.4.1 Untersuchung der Objektverfolgung anhand realer Verkehrsdaten

Die folgenden Abbildungen zeigen die Objektverfolgung über 20 Frames.

Abb. 44: Frame #168, erster Frame der Objektverfolgung

Die Abb. 44 zeigt die Objektverfolgung unmittelbar nach der Initialisierung. Es ist zu erkennen, dass nicht alle Objekte sauber initialisiert wurden. Durch Objektverdeckung bilden die Objekte #7 und #13 ein Cluster von jeweils zwei Fahrzeugen.

Abb. 45: Frame #173, der Cluster #13 konnte aufgelöst werden

In Abb. 45 ist zu sehen, dass sich der Cluster #13 getrennt hat. Es wurde ein neues Objekt erzeugt (#16).

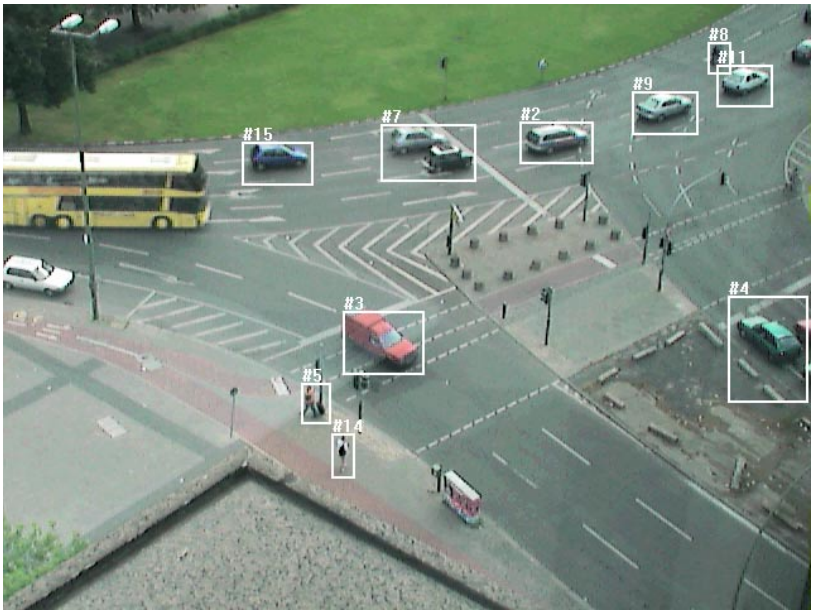


Abb. 46: Frame #178, Objektverfolgung in 2D

Abb. 46 zeigt die Verkehrsszene 5 Frames später. Der Cluster #7 konnte auch hier noch nicht aufgelöst werden. Die Auflösung eines Clusters verlangt, dass die Segmentierung separate Objekte extrahiert.

Abb. 47: Frame #183, Objektverfolgung in 2D

In Abb. 47 sieht man die Szene wiederum 5 Frames später. Ein neues Objekt ist hinzugekommen, ein Bus (#18). Der Cluster #7 konnte wiederum nicht aufgelöst werden.

Der Algorithmus liefert gute Ergebnisse, wie in den Abbildungen 44-47 zu sehen war. Es soll als nächstes Untersucht werden, wie sich der Objektverfolger verhält, wenn sich Objekte für kurze Zeit verdecken. In den nächsten Abbildungen ist eine dynamische Okklusion der Fußgänger zu erkennen. Der Objektverfolger wurde dazu neu initialisiert.

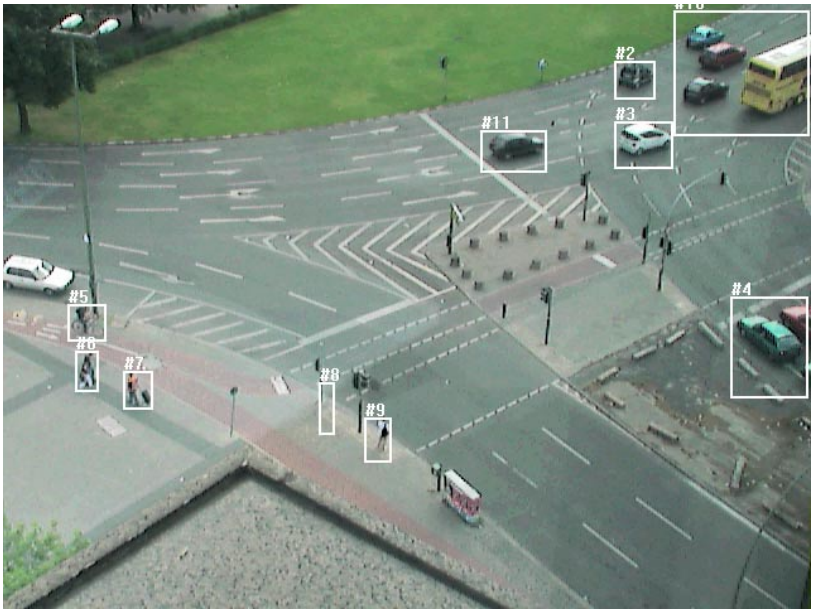


Abb. 48: Frame #225, Untersuchung bei dynamischer Okklusion

Abb. 48 zeigt den ersten Frame der Objektverfolgung nach Initialisierung. Durch einen Segmentierungsfehler ist ein falsches Objekt in die Verfolgung eingegangen (Objekt #8). Der Segmentierungsfehler entstand deshalb, weil die Zeit für die Einschwingphase des Hintergrundschätzers zu kurz gewählt war. Für die Untersuchung sind nur die Objekte #5, #6 und #7 interessant.

Abb. 49: Frame #230, die Objekte #5, #6 und #7 bilden eine dynamische Okklusion

Objekt #6 und Objekt #7 (links unten in Abb. 49) bilden eine dynamische Okklusion. Der Objektverfolger prädiziert die Trajektoren für diese Objekte während dieser Phase. Würde sich die Richtung der Objekte während dieser Phase ändern, dann wäre die Prädiktion falsch.

Abb. 50: Frame #235, die dynamische Okklusion ist beendet

In Abb. 50 ist Szene nach der dynamischen Okklusion dargestellt. Der Objektverfolger kann die Objekte #5, #6 und #7 durch gemessene Daten aktualisieren. Aus Abb. 48-50 wird der Vorteil des Kalman-Filters sehr deutlich. Kurzfristige Verdeckungen können durch Prädiktion übergangen werden.
Als nächstes soll noch untersucht werden, wie sich der Objektverfolger bei statischen Okklusionen verhält. Die nächsten Abb. 51-54 zeigen eine Okklusion von Objekt #13 mit den Fahnen im Vordergrund (roter Kasten). Befindet sich das Objekt hinter dem statischen Objekt, so wird die Trajektorie prädiziert. Da sich die Fahnen ständig im Wind bewegen, würden sie auch als Vordergrundobjekt detektiert werden. Objekt #13 würde mit einer falschen Kontur aktualisiert werden, welches eine falsche Prädiktion der Kontur zur Folge hätte.

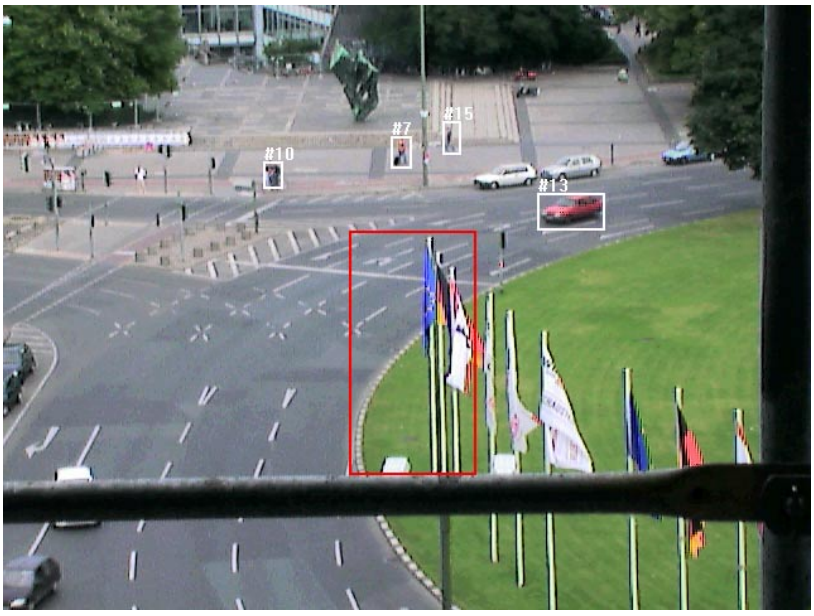


Abb. 51: Frame #265, Untersuchung einer statischen Okklusion, statisches Objekt (roter Kasten)

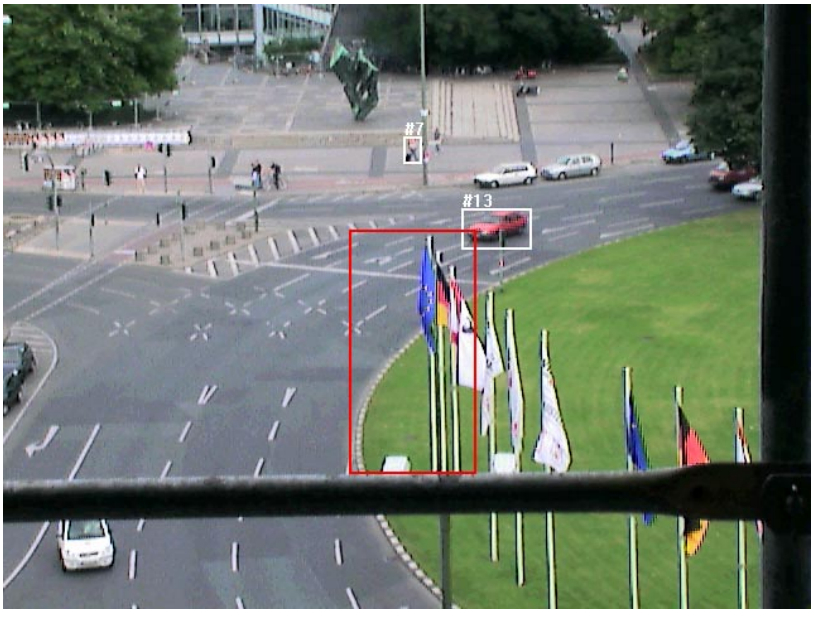


Abb. 52: Frame #270, Untersuchung einer statischen Okklusion

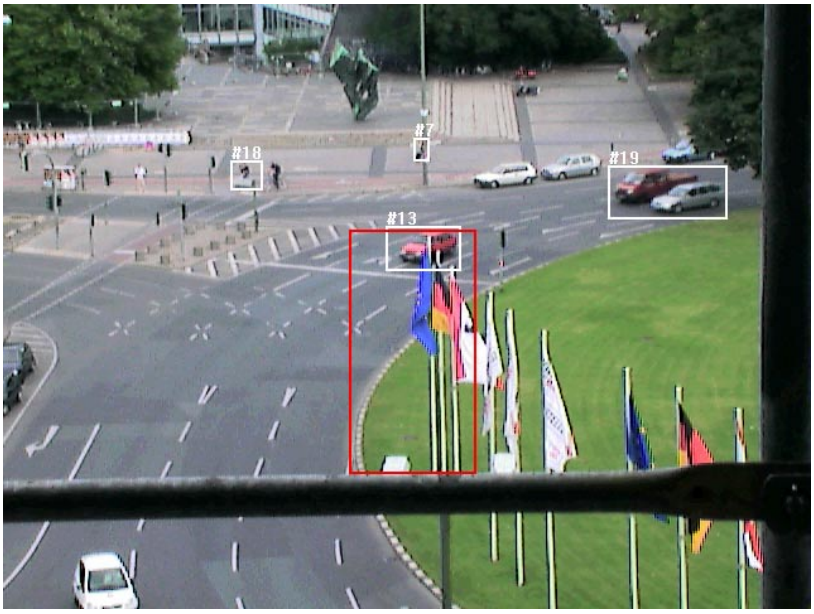


Abb. 53: Frame #275, Untersuchung einer statischen Okklusion

Abb. 54: Frame #285, Untersuchung einer statischen Okklusion

Auch in Abb. 51-54 wurde der Vorteil des Kalman-Filters gezeigt. Ein initialisiertes Objekt kann trotz Okklusion für einige Frames prädiziert werden.

Die Untersuchung hat gezeigt wie wichtig die Initialisierung für eine saubere Verfolgung ist. Sind die Objekte für einige Frames verfolgt worden, so kann eine Okklusion durch Prädiktion der Kontur übergangen werden. Da ein Bewegungsmodell mit konstanter Geschwindigkeit gewählt wurde, erfolgt die Prädiktion der Trajektorie entlang einer Geraden. Abhilfe könnte hier ein Bewegungsmodell konstanter Beschleunigung sein. Kurven könnten so besser modelliert werden.

5.5 Objektverfolgung in mehreren Kameraansichten

Die Untersuchung erfolgte wieder an der Sequenz „Ernst-Reuter-Platz“. Die Einstellungen der Segmentierung sind identisch mit den vorherigen. Für die Objektverfolgung in 3D werden synchrone Videodaten verlangt. Die Synchronisation erfolgte manuell. Die Parameter für die Initialisierung des Moduls befinden sich in Anhang A.

5.5.1 Untersuchung an realen Verkehrsdaten

Die folgenden Abbildungen veranschaulichen die Zuordnung der Objekte und die Verfolgung über die Zeit. Die Objekte wurden in beiden Ansichten mit den gleichen Nummern bezeichnet, um die Zuordnung aufzuzeigen.

Abb. 55: Frame #380, Kamera 1, Zuordnung der Objekte

In Abb. 55 und 56 stellen die Objektverfolgung in mehreren Kameraansichten unmittelbar nach der Initialisierung dar. Objekte die in einer Ansicht zu klein sind oder gerade verdeckt sind, werden in der anderen Ansicht erkannt, siehe Abb. 55 und 56. Objekt #3 ist während der Initialisierung hinter einem statischen Objekt und kann deshalb in Kamera 2 nicht initialisiert werden. Die Objekte #2, #4, #5 und #8 werden richtig zugeordnet. Wobei Objekt #8 eine Gruppe von Menschen darstellt. Es wird aber deutlich, dass die Anzahl der erkannten Objekte erhöht werden konnte, im Gegensatz zu einer Ansicht.

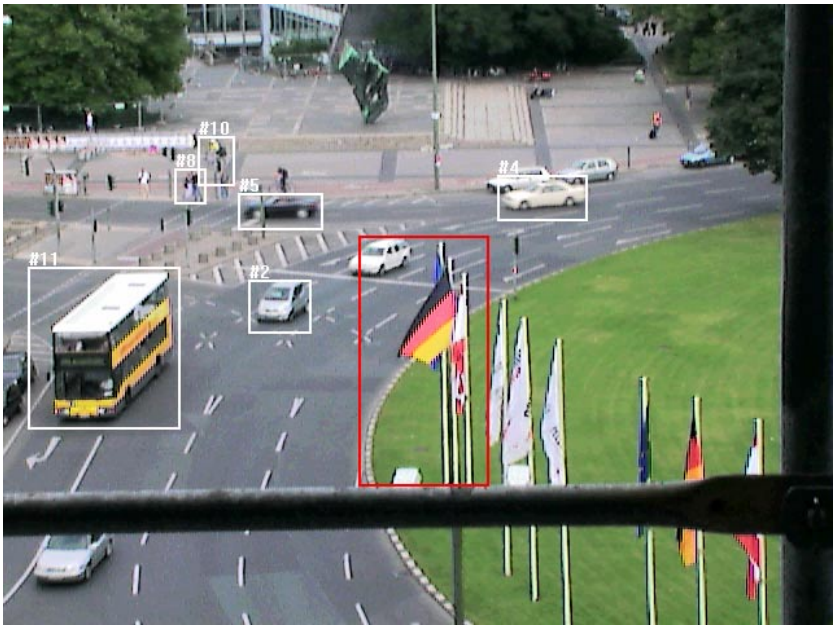


Abb. 56: Frame #380, Kamera 2, Zuordnung der Objekte

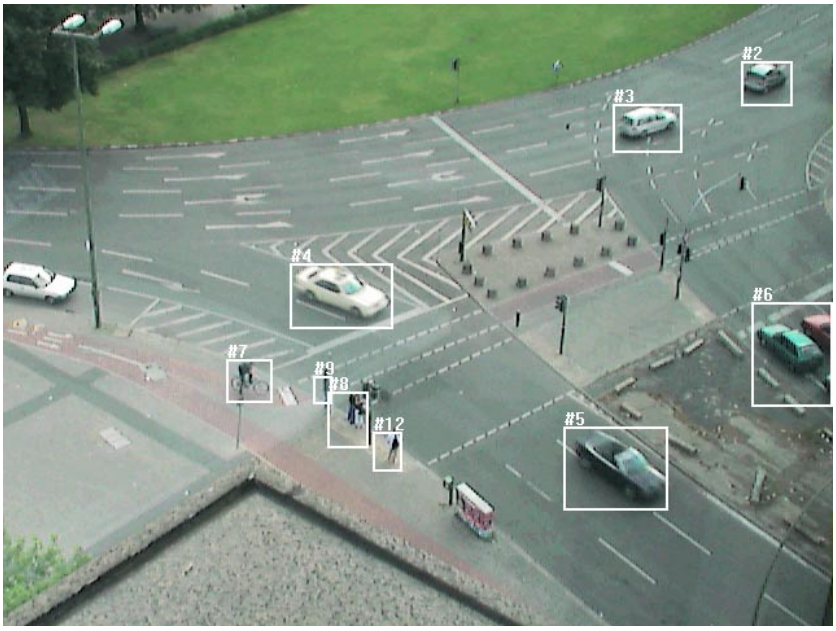


Abb. 57: Frame #400, Kamera 1, Objektverfolgung in 3D

Abb. 57 und 58 zeigen die Szene 20 Frames später. Das Objekt hinter den Fahnen konnte jetzt richtig dem Objekt #3 zugewiesen werden.

Abb. 58: Frame #400, Kamera 2, Objektverfolgung in 3D

Die Untersuchung hat gezeigt, dass die Zuordnung der Objekte zu globalen Objekten korrekt arbeitet. Objekte, die nur aus einer Ansicht sichtbar waren, wurden nachdem sie in der zweiten Ansicht sichtbar wurden, richtig zugewiesen. Durch die Datenfusion können so mehr Objekte verfolgt werden. Zudem wird die Qualität der Verfolgung erhöht.

3D Rekonstruktion

In diesem Abschnitt wird die 3D Rekonstruktion anhand realer Daten aus der Verkehrsszene „Ernst-Reuter-Platz“ untersucht. Die Berechnung der Kameramatrizen erfolgte über den vorgestellten Algorithmus (siehe Abschnitt 3.5) und vernachlässigen jegliche Linsenverzerrungen. Die verwendeten Kameramatrizen der Sequenz „Ernst-Reuter-Platz“ befinden sich im Anhang A. Für das folgende Beispiel wird als Objekt folgender Fahrzeugtyp benutzt (siehe Abb. 59).

Abb. 59: Gerendertes 3D Modell aus der Datenbasis

Die folgende Abb. 60 zeigt die beiden Frames mit dem roten Fahrzeug, dass rekonstruiert werden soll.

Abb. 60: Frames, die zur Rekonstruktion verwendet werden

In Abb. 60 erkennt man schon, dass die Größe der Texturen des Fahrzeugs sehr klein sind. Die Auflösung der Kamera ist 640x480. In der folgenden Abb. 61 sind noch mal beide Texturen separiert dargestellt und stark vergrößert.

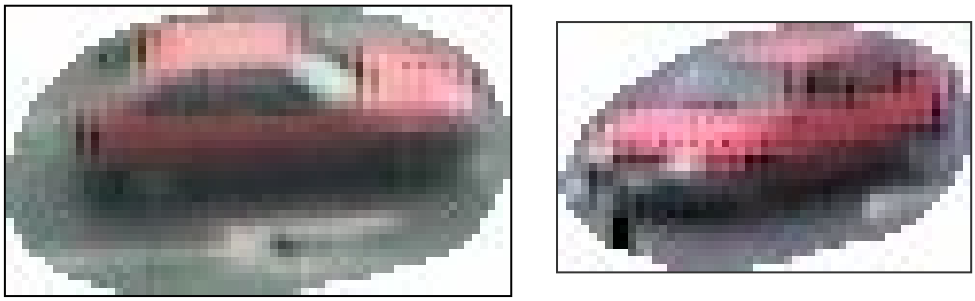


Abb. 61: Extrahierte Texturen (3-facheVergrößerung)

Bei dem verwendeten Grafikchipsatz handelt es sich um den GeForce2 Ultra von NVIDIA. Als Renderer wurde DirectX gewählt und so eingestellt, dass er ein Texturfilter bei Vergrößerung und Verkleinerung der Objekte anwendet. Zum Einsatz kommt ein anisotropischer Filter, da dieser projektive Verzerrungen berücksichtigt. Es reduziert Aliasing-Effekte und bewahrt die Schärfe in den Texturen. In [5] wird ein anisotropisches Verfahren beschrieben. Das genaue Verfahren, wie es auf der Grafikkarte implementiert wurde konnte nicht ermittelt werden.
Die Texturen eines Objekts werden mit einem α-Koeffizienten übereinandergeblendet, da die extrahierten Texturen sich überschneiden können. Man erkennt in Abb. 61, dass die Texturen einer unterschiedlichen Beleuchtung unterliegen. Es ist weiterhin zu erkennen, dass die Texturen nicht nur das Fahrzeug zeigen, sondern auch große Teile der Fahrbahn. Spiegelungen an der Scheibe lassen sich auch nicht vermeiden.

Abb. 62: Projektion der Textur aus Kam #1 auf das 3D Modell

Wie in Abb. 62 veranschaulicht wird, ist die Ausrichtung des Objekts, bzw. die Textur etwas verdreht. Das lässt sich auf die ungenaue Kameramatrix und Fehler in der Segmentierung zurückführen.

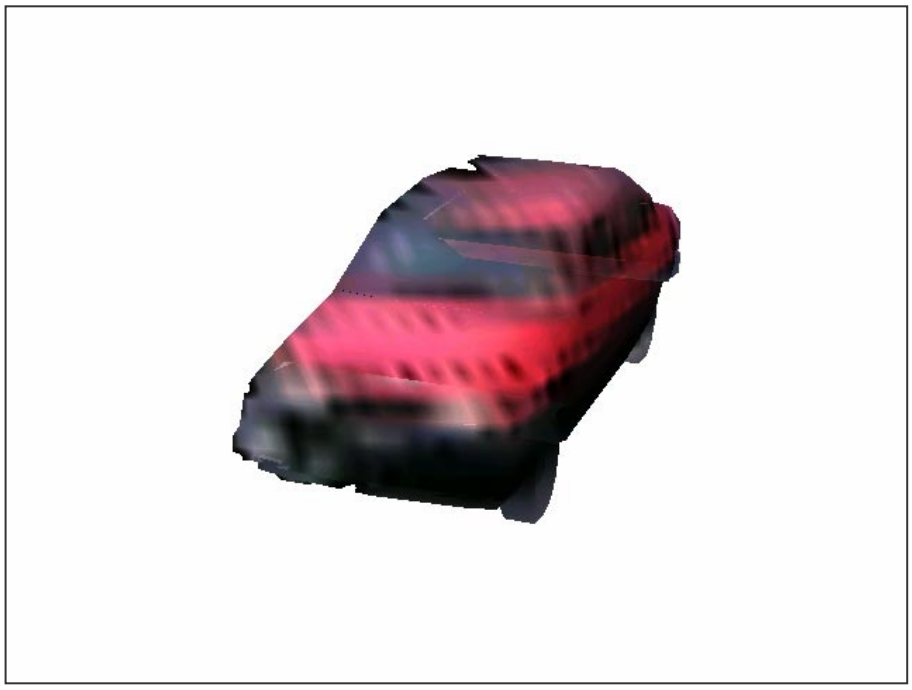


Abb. 63: Projektion der Textur aus Kam #2 auf das 3D Modell

In Abb. 63 wurde die Textur aus Kamera #2 auf das 3D Modell gemappt. Auch hier sind einige Ungenauigkeiten zu erkennen, welche auch auf besagtem Grund zurückzuführen sind.

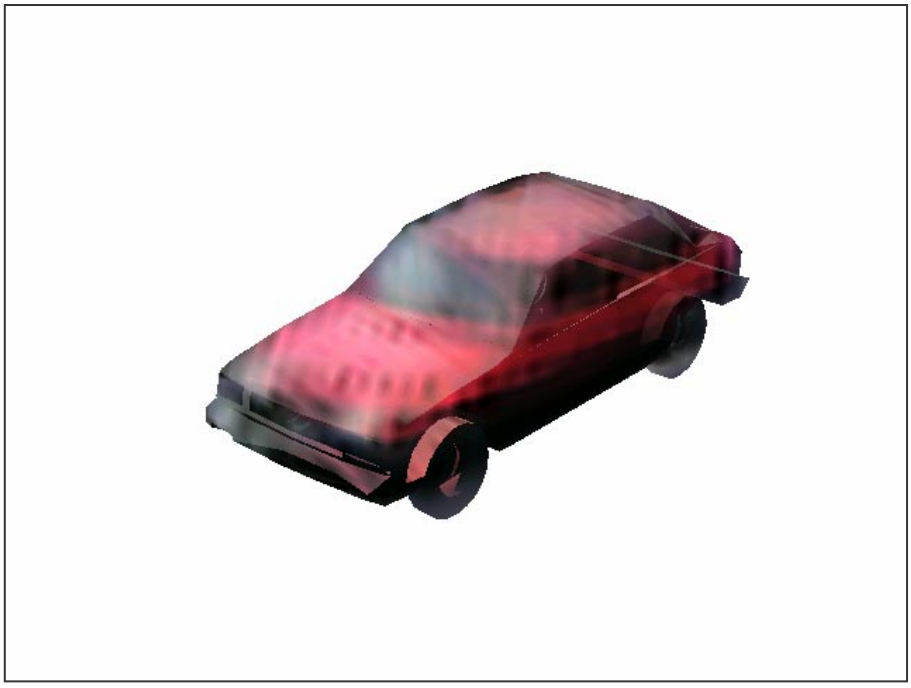


Abb. 64: Beide Texturen wurden auf das 3D Modell gemappt

In Abb. 64 sieht man das Ergebnis, wenn beide Texturen übereinander gemappt werden.

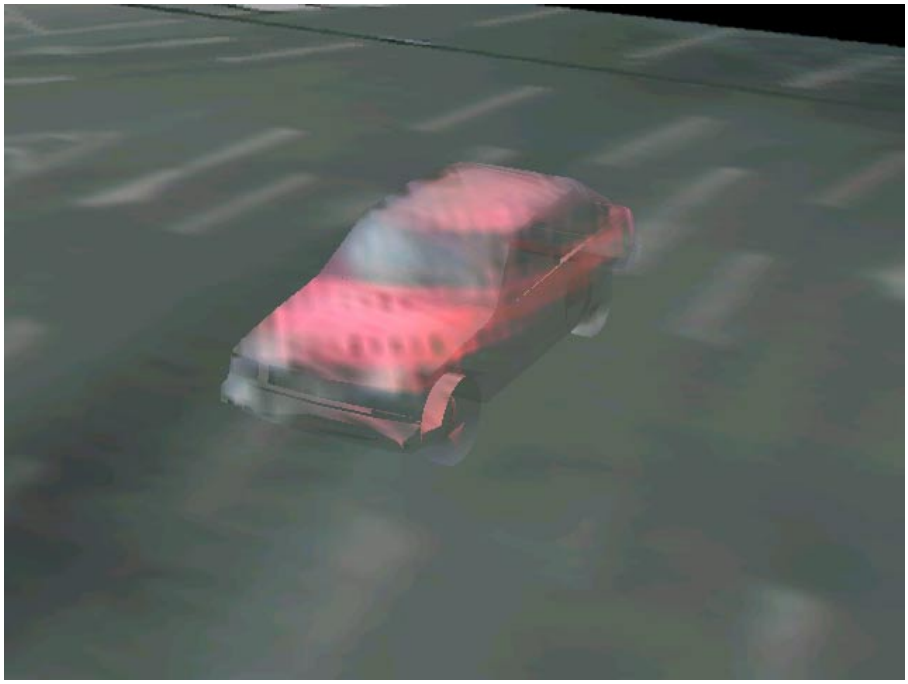


Abb. 65: 3D Modell nach Texturmapping auf die Ebene gesetzt

Abb. 65 zeigt das gleiche Objekt, wie aus Abb. 64, auf die Fahrbahn positioniert.

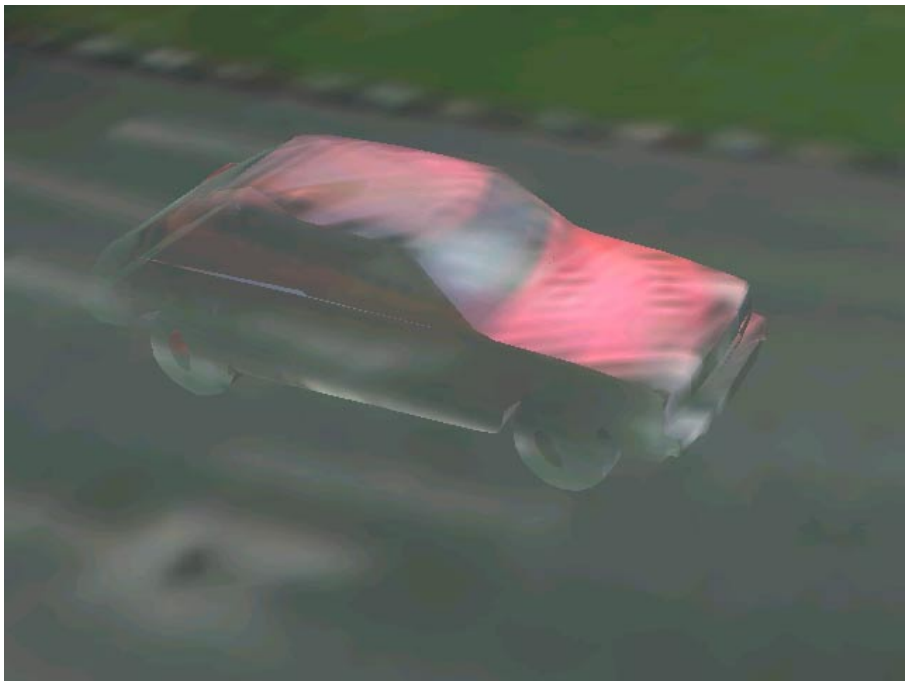


Abb. 66: 3D Objekt von der anderen Seite

Abb. 66 zeigt wiederum das gleiche Objekt, allerdings diesmal aus einer anderen Perspektive.

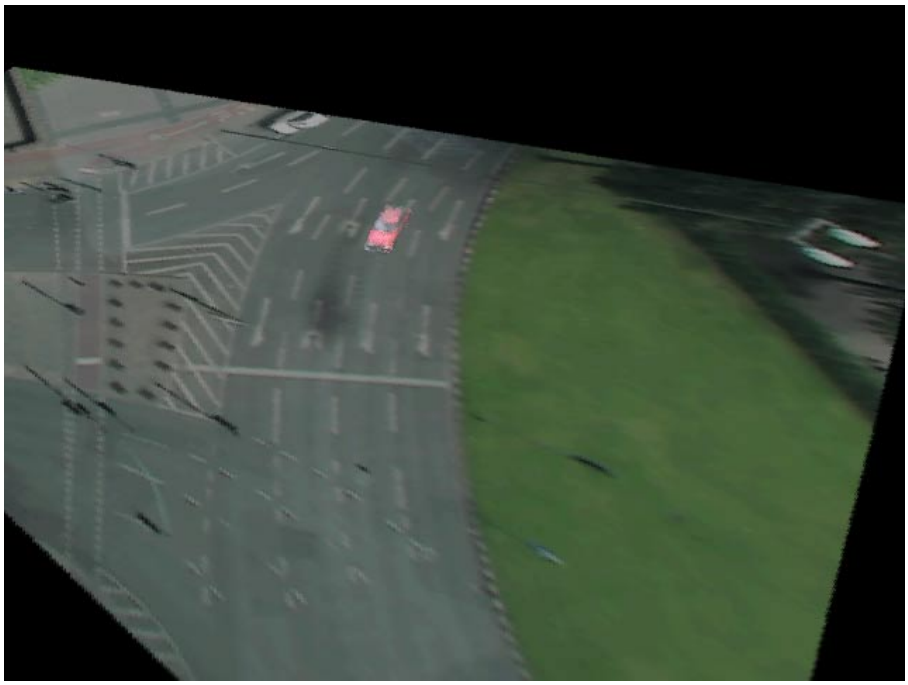


Abb. 67: Blick aus großer Entfernung

Abb. 67 stellt das Objekt aus großer Entfernung dar. Der Einfluss der ungenauen Rekonstruktion nimmt mit anwachsendem Abstand nach. Die Ebene wurde mit der Homographie von Bildkoordinaten nach Weltkoordinaten bei $z = 0$ erstellt. Verzerrungen

sind an Objekten, die aus der Ebene herausragen (Ampeln und Lampen), zu erkennen. Ein Vergleich der Abb. 67 mit Abb. 60 zeigt, dass auch die Position des Fahrzeugs korrekt ist.

Die folgenden Bilder (Abb. 68) zeigt die 3D Rekonstruktion weiterer Fahrzeuge.

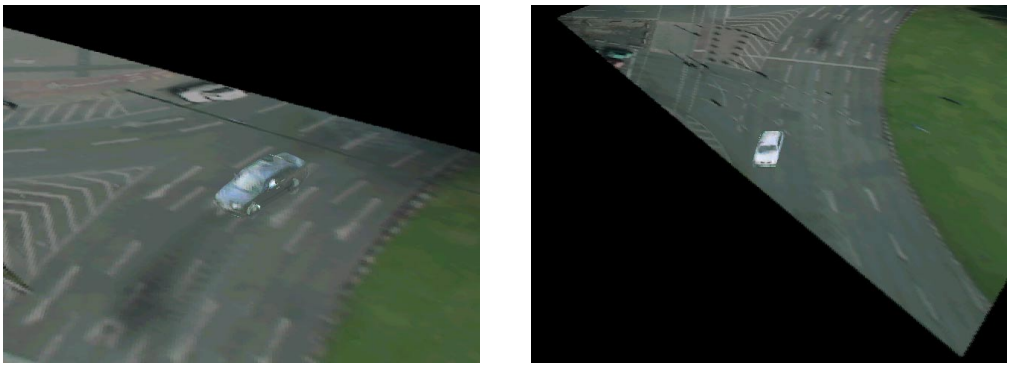


Abb. 68: 3D Rekonstruktion weiterer Fahrzeuge

Im linken Bild von Abb. 68 wurde ein anderes 3D Modell verwendet. Im folgenden soll untersucht werden, ob die Positionierung und die Orientierung korrekt ist. Dazu wurde ein Fahrzeug aus der Sequenz ausgewählt und explizit verfolgt. Die folgenden Abbildungen veranschaulichen, wie sich das 3D Objekt über die Fahrbahn bewegt (Abb. 69–71).

Abb. 69: Animation des Objekts anhand der 3D Position

Abb. 70: Animation des Objekts anhand der 3D Position

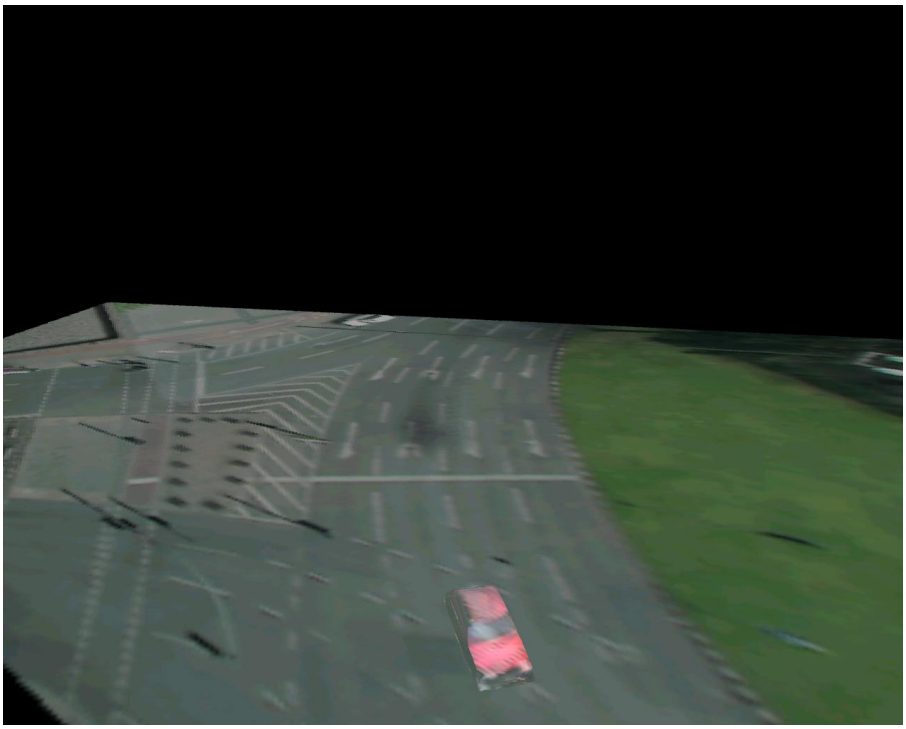


Abb. 71: Animation des Objekts anhand der 3D Position

Dem Benutzer soll die Möglichkeit gegeben werden frei in der 3D Szene zu navigieren. Die hier entwickelten Algorithmen zur 3D Rekonstruktion sollen in einer zukünftigen Applikation nebenläufig zur Navigation des Benutzers laufen.

5.6 Zusammenfassung der Ergebnisse

- Die Segmentierung ist sicher am problematischsten, da alle anderen Module von ihr abhängen. Das Problem, dass Schatten mit in den Vordergrund extrahiert werden, konnte mit diesem Ansatz nicht gelöst werden. Der äußere Einfluss durch Beleuchtungsänderungen, wie der plötzliche Aufbruch einer Wolke, liefert falsche Vordergrundobjekte, für den Fall, dass die Objekte während des Aufbruchs still standen und sich anschließend wieder bewegten. Nach einigen Sekunden hat sich der Hintergrundschätzer allerdings wieder eingeschwungen. Was die Komplexität betrifft, so ist der Aufwand mit Abstand am höchsten verglichen mit den anderen Modulen, da für jedes Pixel ein Kalman-Filter initialisiert und aktualisiert werden muss.

- Die Objektverfolgung in 2D hat wie erwartet Schwierigkeiten, falls die Form des Objekts in der Initialisierungsphase zu sehr schwankt oder Objekte zu einem Objekt verschmelzen. Das hat eine falsche Prädiktion zur Folge und kann im schlimmsten Fall dazu führen, dass der Kontakt zum Objekt verloren geht. Zu kleine Objekt können von diesem Ansatz nicht erfasst werden. Das ist auf die minimale Anzahl der Kontrollpunkte des B-Splines zurückzuführen, die die konvexen Konturen approximieren. Andererseits soll die Anzahl hoch gehalten werden, um so eine genaue Beschreibung der konvexen Kontur zur erhalten. Im Fall einer Verdeckung wird die Trajektorie im Sinne des Modells der konstanten Geschwindigkeiten richtig prädiziert. Ein großes Problem stellt die Gruppierung von Passanten dar. Die geringe Auflösung und die große Entfernung macht es fast unmöglich Passanten zu verfolgen. Problematisch sind auch sehr schnelle und kleine Objekte, die von der Initialisierung nicht erfasst werden. Die Komplexität ist eher gering und hängt nur von der Anzahl der extrahierten Objekte ab, welche normalerweise gering ist.

- Die Zuordnung und Objektverfolgung in 3D liefert gute Ergebnisse. Das einzige Problem, dass hier auftritt, ist die Zuordnung sehr kleiner Objekte in der Ferne, wie sie in Kamera #2 der Sequenz „Ernst-Reuter-Platz“ auftritt. Es kann vorkommen, dass das falsche Objekt zugewiesen wird. Die Berechnung der 3D Position über den Mittelpunkt der Konturen liefert gute Werte. Die meisten Objekte konnten richtig zugewiesen werden. Auch die Problematik der Objektverdeckung konnte so in Einzelfällen gelöst werden. Bei zwei Kameras ist dieser Effekt jedoch eher gering. Die Komplexität des Algorithmus steigt mit der Anzahl der aufgestellten Kameras und der Anzahl der verfolgten Objekte in den einzelnen Ansichten.

- Die 3D Rekonstruktion liefert gute Ergebnisse, falls das 3D Modell hinreichend der Form dem realen Objekt entspricht, das ist aber Vorbedingung für dieses Modul. Die Qualität des Mappings der Texturen steigt mit der Genauigkeit der segmentierten Regionen. Hier haben vor allem lange Schatten zu falschen Ergebnisse geführt. Durch das Übereinanderblenden der Texturen erhält man den Effekt, dass die Texturen nicht

in der Beleuchtung übereinstimmen. Der Aufwand für die 3D Rekonstruktion hängt nur von der Anzahl der Polygone des gewählten 3D Modells ab.

6 Zusammenfassung und Ausblick

Es wurde ein System entwickelt, welches mehrere Objekte aus N Kameraansichten verfolgen und Visualisieren kann. Da die Auswahl des 3D Modells nicht Bestandteil dieser Arbeit war, wird eine Vorauswahl eines 3D Modells manuell vorgenommen. Das Framework beinhaltet eine Segmentierung (Hintergrundschätzung) unter Verwendung eines Kalman-Filter-Formalismus, eine Konturverfolgung in Bildkoordinaten, eine Objektverfolgung in 3D (Weltkoordinaten) und eine modell-basierte 3D Rekonstruktion.

Der Kalman-Filter-Formalismus der Segmentierung modelliert gut die Dynamik des Hintergrundes in den Bereichen wo keine Verdeckungen durch Vordergrundobjekte stattfinden. Im Fall einer Verdeckung kann mit diesem Ansatz keine Aussage über den Hintergrund getroffen werden. Eine Idee um auch verdeckte Regionen zu schätzen, wäre die Umgebung mit in den Prozess einzubeziehen. Als Nachteil würde sich der drastische Anstieg der Komplexität ergeben. Außerdem wäre es sehr interessant zu sehen, wie sich das Ergebnis dieses Ansatzes unter Verwendung eines anderen Farbraums bzw. mehrerer Komponenten eines Farbraums verhält. Speziell im Hinblick auf die Modellierung von Schatten müssen weitere Farbräume untersucht werden. Auch in der Nachverarbeitung der Segmentierung müssen weitere Verfahren untersucht werden. So könnte beispielsweise eine Verbesserung der Konturen durch morphologische Operationen erzielt werden.

Die Konturverfolgung liefert korrekte Ergebnisse unter den folgenden Bedingungen. Während der Initialisierungsphase des Kalman-Filters darf keine Störung erfolgen. Erfolgte die Initialisierung korrekt, so ergab auch die Prädiktion im Falle einer Verdeckung mit anderen Objekten gute Ergebnisse. Wie gut die Initialisierung eines Objektes ist und damit die Prädiktion, lässt sich an zwei Faktoren knüpfen. Der erste Faktor beschreibt die Position und Blickrichtung der Kamera und der zweite Faktor hängt von der Qualität der Segmentierung ab. Clusterbildungen müssen während der Initialisierung vermieden werden, dies lässt sich nur durch den ersten Faktor realisieren. Eine weitere Einschränkung besteht außerdem in dem verwendeten Systemmodell. Die Prädiktion erfolgt entlang einer Geraden und somit können keine Kurven mit diesem Bewegungsmodell geschätzt werden. Mit einem Bewegungsmodell konstanter Beschleunigung lassen sich auch Kurven beschreiben. Eine Idee wäre es zum Beispiel beide Modelle zu verwenden. Falls das Objekt drei Zeitschritte existiert, so kann das komplexere Modell konstanter Beschleunigung die Schätzung übernehmen. Existiert ein Objekt allerdings nur zwei Zeitschritte, dann kann nur das einfache Modell konstanter Geschwindigkeit benutzt werden. Interessant wäre auch noch der Einbezug der Rotation in das Bewegungsmodell, gerade im Hinblick der Verwendung eines Bewegungsmodells mit konstanten Geschwindigkeiten. Im Einzelnen muss die Verfolgung von Personen untersucht werden, da Personen nicht als starre Objekte aufgefasst werden können. Im Besonderen muss die Gruppierung von Personen genauer betrachtet werden, da Personen sehr häufig miteinander interagieren. Die Einschränkung durch die Approximation durch B-Splines ist unschön. Die Approximation der Konturen durch deren Eckpunkte muss erweitert werden. Eine variable Anzahl von Eckpunkten wäre hier eine Lösung.

Die Untersuchung der Objektverfolgung in 3D ergab für den Fall, dass die Initialisierung der Objekte in der Konturverfolgung und das die berechneten Matrizen hinreichend genau waren, gute Ergebnisse. Die Zuordnung der Objekte über die 2D Homographie erfolgte in den meisten Fällen fehlerfrei. Nur bei der Zuordnung von Personen in der Ferne gab es falsche Ergebnisse. Das lässt sich aber auf die Gruppierung von Personen zurückführen. Die Berechnung der 3D Position über die Rückprojektion des Mittelpunktes der Kontur eines Objekte ergab auch korrekte Werte. In Einzelfällen konnten Objektverdeckungen durch diesen Ansatz aufgelöst werden - dies hängt aber wiederum sehr stark von der Position und Blickrichtung der Kamera ab. Wie im Fall der Konturverfolgung hängt die Güte der Verfolgung von der Initialisierung ab. Das Framework der Objektverfolgung wurde für N Kameras implementiert, obwohl die Testdaten nur mit zwei Kameras aufgenommen wurden. In den folgenden Untersuchungen muss ein Testszenario mit mehr als zwei Kameraansichten erstellt werden, um dies zu bestätigen.

Die 3D Rekonstruktion verschiedener Fahrzeuge hat gezeigt, das der Ansatz gute Ergebnisse erzielt. Die Visualisierung bietet die Möglichkeit in einer 3D Umgebung aus allen Perspektiven auf das Geschehen zu blicken. Problematisch waren hier nur Schatten in den extrahierten Texturen, die zu falschen Texturmappings führten. Durch das Übereinanderblenden der Texturen mit dem α-Faktor hat man den Einfluss der Beleuchtung vernachlässigt. Es muss deshalb eine weitere Studie auf diesem Gebiet erarbeitet werden. Die weiteren Arbeiten beschäftigen sich mit der Auswahl des 3D Modells. Die Idee ist es, über die segmentierten Konturen ein identisches 3D Modell aus einer Datenbasis zu ermitteln.

Anhang A

Für die zwei Sequenzen „Ernst-Reuter-Platz“ wurden folgende Parameter verwendet. Sie wurden gleichzeitig aus zwei unterschiedlichen Positionen aufgenommen. Alle Parameter wurden experimentell bestätigt. Die Erklärung der Parameter erfolgt in Anhang C.

Segmentierung

Für beide Sequenzen wurden folgende Parameter verwendet:

$a_1 = a_2 = 0{,}7$
GainMode = “fixed“
$\alpha = 0{,}001$
$\beta = 0{,}03$
AdaptMode = “on“
$th_{BgEstimator,b} = 8{,}0$
StatNum = 10
$\chi = 3{,}25$
$\tau = 15{,}0$
Area = 150.

Objektverfolgung in 2D

Die Initialisierung der Komponente unterschied sich nur in der RegionOfInterest und in den StaticObjects. Die RegionOfInterest war in beiden Kameras die Strasse und der Gehweg. In Kamera #2 wurde außerdem noch ein StaticObject gesetzt. Folgende Werte wurden benutzt:

$th_{init,contourtrac\,\mathrm{ker}} = 0{,}1$
$th_{static,contourtrac\,\mathrm{ker}} = 0{,}1$
$th_{dynamic,contourtrac\,\mathrm{ker}} = 0{,}1$
$th_{roi,contourtrac\,\mathrm{ker}} = 0{,}9$
$th_{track,contourtrac\,\mathrm{ker}} = 0{,}1$
NumPhantom = 8
NumOcclusionDyn = 10
NumOcclusionStat = 10
$\sigma^2_{e,x} = 50$
$\sigma^2_{e,y} = 50$
$\sigma^2_{e,\xi} = 50$

$\sigma^2_{p,x} = 3 \cdot 10^{-3}$
$\sigma^2_{p,y} = 3 \cdot 10^{-3}$
$\sigma^2_{p,\xi} = 10^{-8}$
$\sigma^2_{e,Shape} = 50$
$\sigma^2_{p,Shape} = 10^{-3}$
$\sigma^2_{m,Shape} = \sigma^2_{m,Motion} = 0{,}3$
NumBSpline = 9.

Objektverfolgung in 3D

Für die Sequenz „Ernst-Reuter-Platz wurden die folgenden Matrizen berechnet. Die erste Matrix ist die 2D-Homographie zwischen Kamera #1 und Kamera #2.

$$\mathbf{H}_{12} = \begin{bmatrix} -3{,}8366993e-001 & -2{,}9269928e-001 & 2{,}8498792e+002 \\ 2{,}4029575e-002 & 4{,}0381227e-002 & 6{,}3273164e+001 \\ -1{,}8099950e-004 & 9{,}0656009e-004 & 3{,}1996924e-001 \end{bmatrix}$$

Die Matrix wurde mit dem DLT-Algorithmus (siehe Abschnitt 3.4.3) in MatLab berechnet. Für die Berechnung wurden manuell mehrere (>4) Korrespondenzen in den Bildern bestimmt.

Für Kamera #1 und Kamera #2 wurde die Homographie für Bild- nach Weltkoordinaten **W** und die Kameramatrix **P** berechnet.

$$\mathbf{W}_1 = \begin{bmatrix} -4{,}7696330e+000 & 7{,}6562551e+001 & 2{,}7217584e+004 \\ -6{,}1714976e-001 & 1{,}0380230+001 & 3{,}6570247e+003 \\ -5{,}9712973e-005 & 9{,}6321005e-004 & 3{,}4268225e-001 \end{bmatrix}$$

$$\mathbf{W}_2 = \begin{bmatrix} -5{,}9630691e+000 & -6{,}7005997e+001 & -1{,}3327048e+004 \\ -7{,}8043364e-001 & -9{,}0422301e+000 & -1{,}8052196e+003 \\ -7{,}5072924e-005 & -8{,}4530991e-004 & -1{,}6747394e-001 \end{bmatrix}$$

$$\mathbf{P}_1 = \begin{bmatrix} -9{,}7093774e+002 & 6{,}5398959e+002 & -1{,}4976339e+002 & 7{,}0135083e+007 \\ 2{,}6303629e+002 & 2{,}4213476e+002 & -1{,}1033287e+003 & -2{,}3474135e+007 \\ -6{,}4417989e-001 & -5{,}9299173e-001 & -4{,}8310771e-001 & 5{,}7605635e+004 \end{bmatrix}$$

$$\mathbf{P}_2 = \begin{bmatrix} 5{,}8810285e+002 & -2{,}1437666e+003 & -6{,}3440661e+001 & -2{,}3689576e+007 \\ -2{,}1009315e+002 & -2{,}6156123e+001 & -2{,}2010135e+003 & 1{,}6996470e+007 \\ 9{,}7324952e-001 & 1{,}2116737e-001 & -1{,}9520203e-001 & -7{,}8445459e+004 \end{bmatrix}$$

Die Weltkoordinaten wurden mit Hilfe eines Lageplans bestimmt. Die Berechnung der Matrizen erfolgte mit dem DLT-Algorithmus (siehe Abschnitt 3.4.3 und 3.5.3) in MatLab. Folgende Werte wurden gewählt:

$\text{PlaneNormal} = [0 \quad 0 \quad 1]^T$
$\text{NumViews} = 2$
$th_{assign} = 50$
$th_{multitrac\,ker,track} = 10$
$th_{multitrac\,ker,init} = 5$
$\sigma_e^2 = 500$
$\sigma_p^2 = 1$
$\sigma_m^2 = 2.$

Anhang B

Im folgenden Kapitel werden noch mal alle Eingabeparameter der einzelnen Module der Übersichtlichkeit noch mal aufgelistet, die für die Initialisierung bzw. den Aufruf benötigt werden. Die genaue Beschreibung der Parameter befinden sich jeweils in den Abschnitten des Kapitels 4.

Segmentierung

Initialisierung:

a_1, a_2	- Systemmatrix d. Kalman-Filters
GainMode	- „fixed“ oder „frame“
α	- Kalman-Gain der Vordergrundadaption
β	- Kalman-Gain der Hintergrundadaption
AdaptMode	- variablen Schwellwert benutzen
$th_{BgEstimator,b}$	- Basisschwellwert [0..255]
StatNum	- Anz. d. stat. Datensätze z. Berechnung d. Schwellwerts
χ	- Konfidenzintervall der Student’t Distribution
τ	- Zeitkonstante der e-Funktion
Area	- Mindest-Größe einer Region [in Pixel]
BackgroundImage	- Hintergrundbild („leere Szene“)

Aufruf:

CurrentImage	- das aktuelle Bild

Rückgabe:

ForegroundRegions	- Feld der detektierten Vordergrundregionen

Objektverfolgung in 2D

Initialisierung:

RegionOfInterest	- die für die Objektverfolgung relevante Region
StaticObjects	- Liste statischer Objekte (Regionen)
$th_{init,contourtrac\,\mathrm{ker}}$	- min. Schwellwert für die Initialisierung
$th_{static,contourtrac\,\mathrm{ker}}$	- min. Schwellwert für den Schnitt mit stat. Objekt
$th_{dynamic,contourtrac\,\mathrm{ker}}$	- min. Schwellwert für den Schnitt mit dyn. Objekt
$th_{roi,contourtrac\,\mathrm{ker}}$	- min. Schwellwert für den Schnitt mit ROI
$th_{track,contourtrac\,\mathrm{ker}}$	- min. Schwellwert für den Schnitt mit Segmentierung
NumPhantom	- max. prädizierte Zeitschritte im Zustand Phantom

NumOcclusionDyn	- max. prädizierte Zeitschritte im Zustand DynamicOcc
NumOcclusionStat	- max. prädizierte Zeitschritte im Zustand StaticOcc
$\sigma^2_{e,x}$	- Startwert d. Varianz d. $\boldsymbol{\Theta}_{Motion}$: Translation x
$\sigma^2_{e,y}$	- Startwert d. Varianz d. $\boldsymbol{\Theta}_{Motion}$: Translation y
$\sigma^2_{e,\xi}$	- Startwert d. Varianz d. $\boldsymbol{\Theta}_{Motion}$: Skalierung ξ
$\sigma^2_{p,x}$	- Varianz d. Prozessfehlers d. Translation x
$\sigma^2_{p,y}$	- Varianz d. Prozessfehlers d. Translation y
$\sigma^2_{p,\xi}$	- Varianz d. Prozessfehlers d. Skalierung ξ
$\sigma^2_{m,Shape} = \sigma^2_{m,Motion}$	- Varianz d. Messfehlers
$\sigma^2_{p,Shape}$	- Varianz d. Prozessfehlers d. Form
$\sigma^2_{e,Shape}$	- Startwert d.Varianz d. Form
NumBSpline	- Anzahl der Kontrollpunkte der B-Splines

Aufruf:

CurrentRegions	- Array d. aktuellen Vordergrundregionen

Rückgabe:

TrackedObjects	- Liste d. verfolgten Objekte

Objektverfolgung in 3D

Initialisierung:

CameraMatrix	- Array d. Kameramatrizen
ImageHomographies	- 2D Homographien
WorldHomographies	- Homographien (Ebene zu Bild)
PlaneNormal	- Flächennormale d. Ebene
NumViews	- Anzahl d. Kameras
th_{assign}	- Schwellwert d. Zuordnung
$th_{multitrac\,\mathrm{ker},track}$	- Schwellwert d. Objektverfolgung in 3D
$th_{multitrac\,\mathrm{ker},init}$	- Schwellwert d. Initialisierung eines Objektes
σ^2_e	- Startwert d. Varianz d. Schätzung
σ^2_p	- Varianz d. Prozessfehlers
σ^2_m	- Varianz d. Messfehlers

Aufruf:

TrackedObjects2D	- Array d. Listen d. Objektverfolgung in 2D

Rückgabe:

TrackedObjects3D	- Liste d. verfolgten Objekte in 3D

3D Rekonstruktion

Initialisierung:

CameraMatrix	- Array d. Kameramatrizen
NumViews	- Anzahl d. Kameras

Aufruf:

Object	- verfolgtes 3D Object
Images	- aktuelle Bilder d. Sequenzen

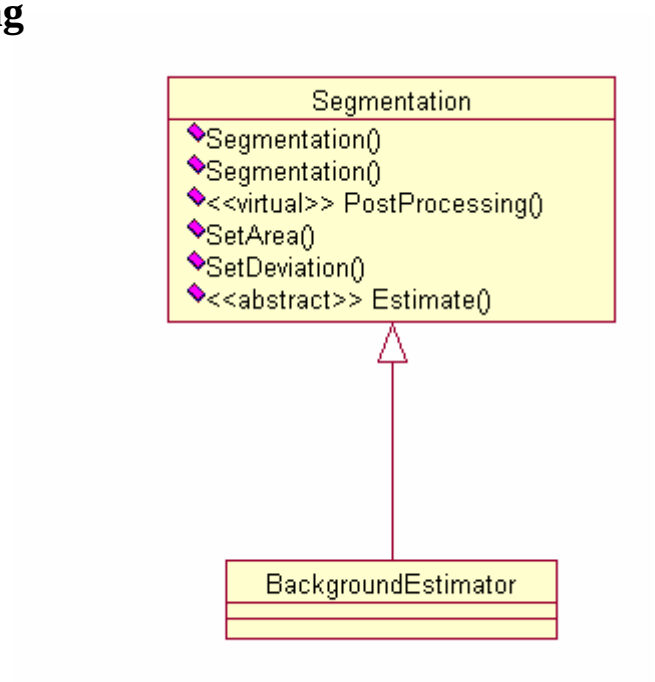
Rückgabe:

3DObject	- rekonstruiertes Objekt (Mesh,Texturen)

Anhang C

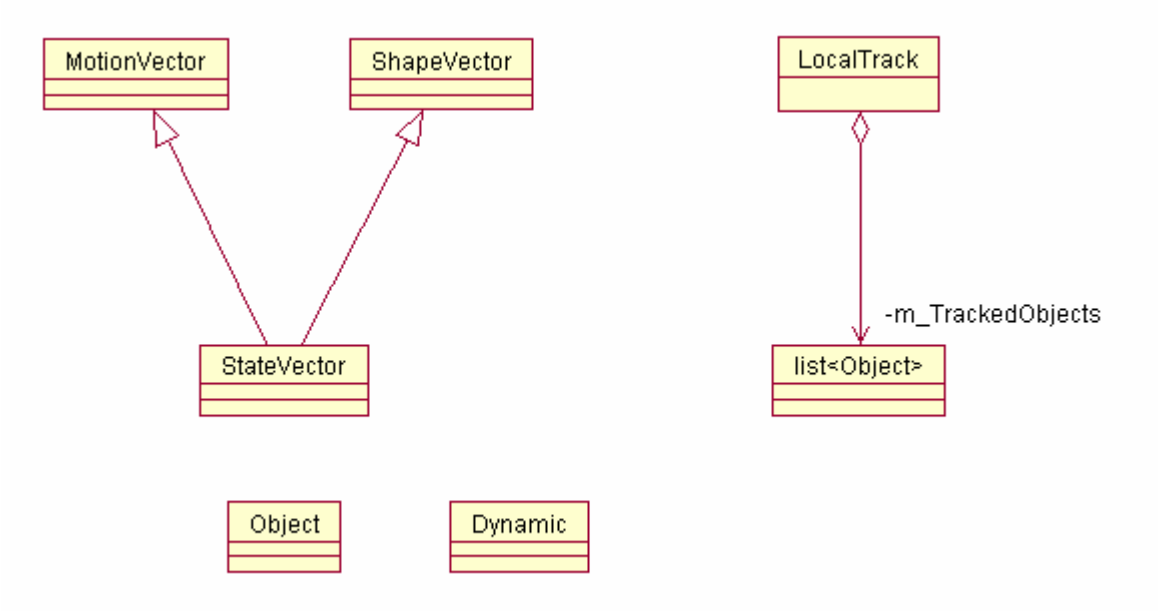
Die Klassendiagramme wurden mit dem Entwicklungswerkzeug Rational Rose erstellt. Es werden keine Hilfsklassen, wie Kalman-Filter oder Matrix dargestellt. Die Beschriftung an den Assoziationen gibt die Multiplizität einer Instanz an.

Segmentierung



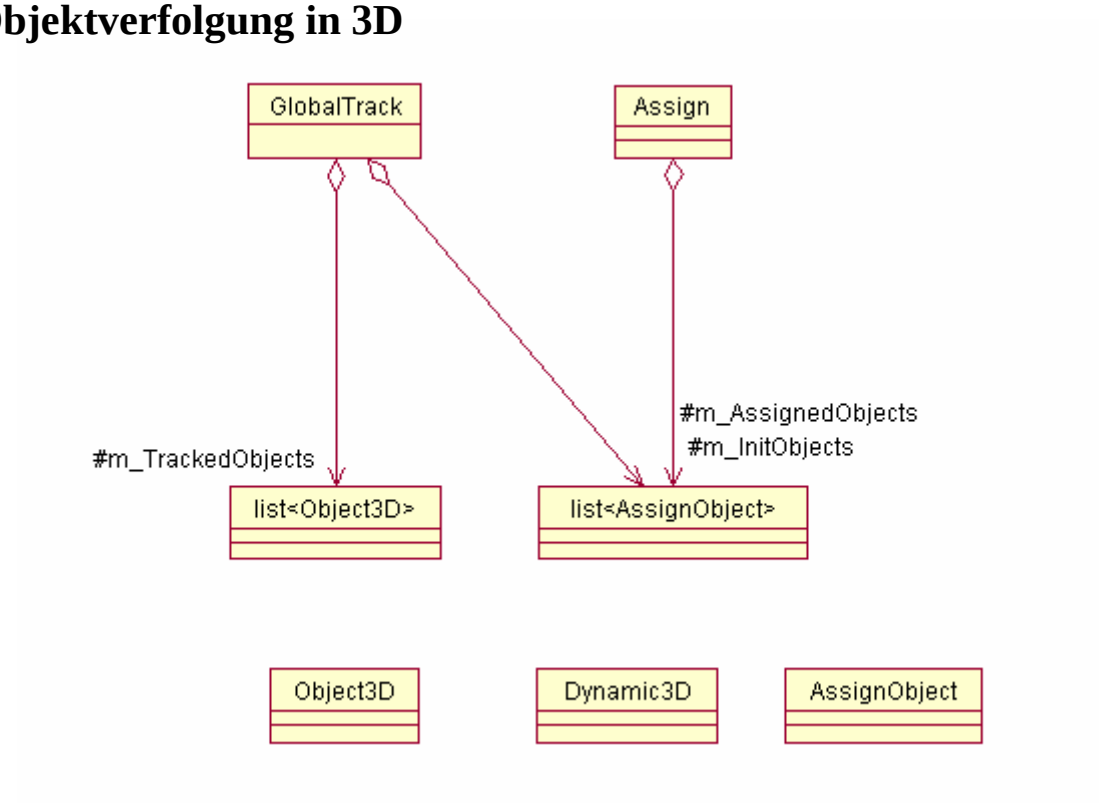
Die Klasse „Segmentation“ bildet die externe Schnittstelle.

Objektverfolgung in 2D



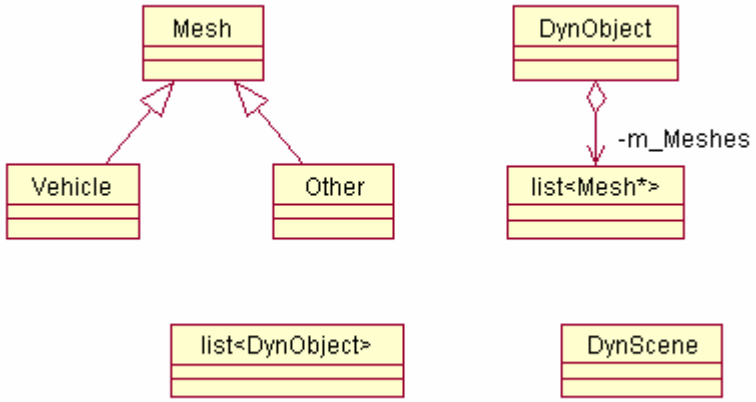
Die externe Schnittstelle wird von der Klasse „LocalTrack“ gebildet. Ein Objekt wird vor allem durch seine Dynamik und dessen Zustand gekennzeichnet.

Objektverfolgung in 3D



Hier wird die externe Schnittstelle von der Klasse „GlobalTrack“ beschrieben. Für die Zuordnung wurde eine weitere Klasse „Assign“ eingeführt. Wie in der Objektverfolgung in 2D ist ein Objekt hauptsächlich durch seine Dynamik gekennzeichnet.

3D Rekonstruktion



Die 3D Rekonstruktion findet in der Klasse „Mesh“ statt. Die Klasse „DynScene“ verwaltet die Objekte und ist deshalb als externe Schnittstelle zu sehen.

Symbolverzeichnis

B	-	Basisfunktion d. B-Splines
$\mathbf{C}$	-	Kameraposition in Weltkoordinaten
$\mathbf{E}$	-	Einheitsmatrix
$\mathbf{F}$	-	Fehlermatrix
$\mathbf{H}$	-	Homographie
$\mathbf{K}$	-	Kalibrierungsmatrix
$\mathbf{P}$	-	Kameramatrix
Q	-	B-Spline-Kurve
$\mathbf{R}$	-	Rotationsmatrix
S	-	Region
V	-	Kontrollpunkt d. B-Splines
b	-	kubisches Segment d. B-Splines
f	-	Brennweite d. Objektivs
k	-	Zeitindex
m_x, m_y	-	Anzahl d. Pixel pro Bildkoordinate
$\mathbf{p}$	-	Punkt im $\aleph^n$
$\tilde{\mathbf{p}}$	-	Punkt im $\Re^n$
s	-	Intensität eines Pixels
$\mathbf{t}$	-	Translationsvektor
th	-	Schwellwert (Threshold)
$\mathbf{u}$	-	deterministischer Eingangsvektor d. Kalman-Filters
$\mathbf{v}, \mathbf{w}$	-	Zufallsvariablen
$\mathbf{x}$	-	Zustandsvektor
x_0, y_0	-	Schnittpunkt d. z-Achse d. Kamera durch d. Bildebene in Pixel
$\mathbf{z}$	-	Beobachtungsvektor
$\mathbf{\Phi}$	-	Beobachtungsmatrix
$\mathbf{\Gamma}$	-	Stellmatrix
$\mathbf{\Lambda}$	-	Systemmatrix
$\mathbf{\Pi}$	-	Kalman-Gain

$\mathbf{\Theta}$	-	Kovarianzmatrix d. Schätzfehlers
$\mathbf{\Omega}$	-	Kovarianzmatrix d. Systemfehlers
$\mathbf{\Xi}$	-	Kovarianzmatrix d. Meßfehlers
α, β	-	Adaptionsgeschw. d. Hintergrundschätzers f. Vorder- bzw. Hintergrund
χ	-	Konfidenzintervall d. Student's t-Distribution
δ_x, δ_y	-	Schnittpunkt d. z-Achse d. Kamera durch d. Bildebene in Kamerakoordinaten
η_x, η_y	-	Bildseitenverhältnis (aspect ratio)
κ	-	Kalman-Gain d. Hintergrundschätzers
θ	-	Zoomfaktor
ϑ	-	Scherungsparameter d. Kalibrierungsmatrix $\mathbf{K}$
ρ	-	Punkt im $\Re^2$ ($\rho = (x, y)^T$)
τ	-	Zeitkonstante d. e-Funktion
σ	-	Standardabweichung
ξ	-	Skalierungsfaktor
$\mathbf{א}$	-	projektiver Raum

Literaturverzeichnis

[1] R.H. Bartles, J.C. Beatty, B.A. Barsky, “An Introduction to Splines for use in Computer Graphics and Geometric Modelling”, Morgan Kauffmann, 1987

[2] J. Black, Dr. T. Ellis,“Multi Camera Image Tracking”, 2nd IEEE Int. Workshop on PETS, Dec 2001

[3] T.H. Cormen, C.E. Leisserson, R.L. Rivest, C. Stein, “Introduction to Algorithms”, Second Edition, MIT Press, 2001

[4] P. Eisert, E. Steinbach, B. Girod, “3D Shape Reconstruction from Light Fields using Voxel Back-Projection”, Vision, Modeling and Visualization Workshop, Erlangen, pp. 67-74, November 1999

[5] J. P. Ewins, M. D. Waller, M. White, P. F. Lister, “MIP-Map Level Selection for Texture Mapping”, IEEE Transactions on Visualization and Computer Graphics”, Vol. 4, No. 4, October-December 1998

[6] A. Gelb, “Applied Optimal Estimation”, MIT Press, 1974

[7] S. Gupte, Osama Masoud, R.F.K. Martin, N.P. Papanikolopoulos, “Detection and Classification of Vehicles”, IEEE Intelligent Transportation Systems Vol. 3, No 1, March 2002

[8] R. Hartley, A. Zisserman, “Multiple View Geometry in Computer Vision”, Cambrigde Universitity Press 2000

[9] K.-P. Karmann, A. v. Brandt, “Moving Object Recognition Using an Adaptive Background Memory“, V. Cappellini, “Time-varying Image Processing and Moving Object Recognition, 2”, Elsevier Publishers B.V., 1990

[10] R. Koch, M. Pollefeys, L. Van Gool, ”Realistic Surface Reconstruction of 3D Scenes from Uncalibrated Image Sequences”, Journal of Visualization and Computer Animation 11(3): 115-127, 2000

[11] D. Koller, K. Daniilidis, H.-H. Nagel, “Model-Based Object Tracking in Monocular Image Sequences of Road Traffic Scenes”, International Journal of Computer Vision, 10(3): 257-281, 1993

[12] D. Koller, J. Weber, J. Malik, “Robust Multiple Car Tracking with Occlusion Reasoning“, University of California at Berkeley, 1993

[13] M. K. Leug, Y.-H. Yang, “Human Body Motion Segmentation In A Complex Scene”, Pattern Recognition, Volume 20, pp. 55-64, 1987

[14] L. Marcenaro, F. Oberti, C.S. Regazzoni, “Multiple objects color-based tracking using multiple cameras in complex time-varying outdoor scenes”, 2nd IEEE Int. Workshop on PETS, Dec 2001

[15] P. S. Maybeck, “Stochastic Models, Estimation, and Control”, Volume 1, 1979, Academic Press, Inc.

[16] MVTec, Referenzhandbuch HALCON, Version 6.0.1, Juli 2001

[17] H.-H. Nagel, “On the estimation of optical flow: Relations between different approaches and some new results”, Artificial Intelligence 33, pp. 299-324, 1987

[18] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, “Numerical Recipes”, Cambridge University Press, 1986

[19] C. Ridder, O. Munkelt, H. Kirchner, “Adaptive Background Estimation and Foreground Detection using Kalman-Filtering”, Proceedings of the International Conference of Recent Advances in Mechatronics, ICRAM ’95,
pp. 193-199, 1995

[20] M. R. Spiegel, “Theory and Problems of Probability and Statistics”, New York: McGraw-Hill, 1992

[21] A. Watt, F. Policarpo, “3D Games: Volume I: Real-Time Rendering and Software Technology”, Addison Wesley, 2001

[22] G. Welch, G. Bishop, “An Introduction to Kalman Filter“, University of North Carolina, 2002, http://www.cs.unc.edu/~welch

Abbildungsverzeichnis

Erfolgskontrollbericht zum Vorhaben

„Videogestützte Verkehrserfassung (VVE)“

Teilprojekt des Verbundprojekts

„Optische Informationssysteme (OIS) zur Verkehrsszenenanalyse und Verkehrslenkung“

Förderkennzeichen 03WKJ02C

Das Vorhaben „Videogestützte Verkehrserfassung (VVE)“ als Teilprojekt des Verbundprojekts „Optische Informationssysteme (OIS) zur Verkehrsszenenanalyse und Verkehrslenkung“ bezog sich auf die Thematik der Videocodierung und –übertragung in der Verkehrsüberwachung sowie die Erzeugung von Zwischenansichten zwischen den vorhandenen Kamerapositionen, womit das Verkehrsgeschehen z.B. an einer Kreuzung von jedem beliebigen Blickpunkt und –winkel aus betrachtet werden kann. Zum ersten Punkt sollte ein System entwickelt werden, dass es ermöglicht die Signale einer Vielzahl von verteilten Kameras in komprimierter Form zu einer Verkehrsleitzentrale zu übertragen. Dort sollten die Bilder interaktiv dargestellt werden können. Weiterhin sollte eine Fernsteuerung des gesamten Systems von der Zentrale aus möglich sein. Zur Zwischenbildinterpolation sollten geeignete Verfahren der 3D Rekonstruktion entwickelt werden, die es ermöglichen, ein komplettes dynamisches 3D Modell von Verkehrsszenen zu erzeugen. Dies bietet die Funktionalität der freien Navigation, wie sie aus der Computergraphik bekannt ist.

Zur Videocodierung und –übertragung konnte auf umfangreiches Vorwissen (inklusive Software, MPEG-4 Codierung und Übertragung über Internet Protokoll) zurückgegriffen werden. Auch zur Zischenbildinterpolation sollte auf vorhandenes Know-how zur disparitätsgestützten Rekonstruktion aufgebaut werden. Es stellte sich jedoch schnell heraus, dass solche Ansätze unter den besonders ungünstigen Randbedingungen in OIS (wenige Kameras mit sehr unterschiedlichen Ausrichtungen) nicht zum Erfolg führen können. Daher musste zur Zwischenbildinterpolation ein neuer Ansatz entwickelt werden.

1. Ergebnisse

Im Folgenden sind die wichtigsten Projektarbeiten und Ergebnisse aufgeführt:

1.1 Videocodierung und -Übertragung

Es wurde ein leistungsfähiges Server/Multiclientsystem zur Übertragung mehrerer Videosignale zu einer Zentrale entwickelt. Dabei wurde nach modernsten Methoden des Softwareengineering vorgegangen (UML, CVS, C++, Multithreading). Das System kennzeichnet sich durch hohe Flexibilität, individuelle Konfigurierbarkeit, Skalierbarkeit und Fernsteuerbarkeit aus.

Jede Kamera im System ist mit einem Videoserver verbunden, der aus einer Reihe von Modulen (Threads) besteht. Die einzelnen Module implementieren die verschiedenen Funktionalitäten wie Framegrabber, Kompression (MPEG-4), Übertragung (RTSP/RTP) und Steuerung. Die Kommunikation zum Client erfolgt über UDP (Bilddaten) und TCP (Steuerung). Über die gesicherte TCP-Verbindung kann jeder Server ferngesteuert werden. Dies beinhaltet Ein-, Ausschalten, Starten, Stoppen, sowie die Einstellung von Qualitätsprofilen. Die Qualitätsstufen (Kombinationen von Bilddimensionen, Wiederholraten, I-Frame-Perioden, Quantisierung) erlauben eine Anpassung an die Netzauslastung, sowie eine Einstellung der erforderlichen Rechenleistung im Server.

Der Multiclient für die Verkehrszentrale übernimmt die interaktive Visualisierung der Videos sowie die zentrale Steuerung und Verwaltung des Systems. Datenpakete werden empfangen, die Videosignale decodiert und nach Benutzereingabe dargestellt. Hierzu existiert im Multiclient zu jedem einzelnen Server im System ein eigener Client. Weiterhin wird eine zentrale Überwachung der Netzwerkbedingungen durchgeführt. Bei Überlastung werden ggf. die Bitraten der einzelnen Server über die Steuerverbindungen reduziert. Schließlich werden Funktionalitäten zur Paketwiederholung bei gestörter Übertragung zur Verfügung gestellt.

Die beschriebenen Module des Videoservers und des Multiclients sind problemlos in den OIS-Demonstrator integriert und bezüglich Funktionalität und Robustheit erfolgreich getestet worden. Weiterhin wurde am FHG-HHI ein Stand-alone-Demonstrator realisiert, der aus 4 Videoservern (Standard PCs mit der entwickelten Software) mit Kameras und einem Multiclient (Standard PC) besteht. Dieses System wurde ausgiebig getestet (z.B. mit LAN und WLAN Verbindungen) und optimiert. Dabei konnte die Funktionalität und Robustheit verifiziert werden. Der HHI-Demonstrator wurde erfolgreich auf zahlreichen internen Vorführungen und externen Ausstellungen einer breiten Öffentlichkeit präsentiert, darunter Lange Nacht der Wissenschaften, International Funkausstellung und CEBIT 2004.

1.2 Zwischenbildinterpolation

Falls eine Kreuzung von mehr als einer Kamera überwacht wird, ist es möglich, ein interaktives, dynamisches 3D Modell der Szene zu erzeugen. Ein solches 3D Modell ermöglicht es, die Szene von jedem beliebigen Standpunkt und Blickwinkel aus zu betrachten, wie es z.B. von Computergrafikmodellen (Spiele, virtuelle Welten) bekannt ist. Dies ermöglicht eine bessere Visualisierung des Geschehens, als es mit den 2D Ansichten allein möglich wäre. So können z.B. Gefahrensituationen und Unfälle sehr viel genauer in ihrer Entstehung analysiert werden. Die 3D Positionen und Trajektorien (auch Geschwindigkeiten) von Fahrzeugen können genau nachvollzogen werden. Zu diesem Zweck wurde das Videostreamingsystem erweitert. Die Videosignale werden von den Videoclients an ein 3D Analysemodul weitergeleitet.

Die ursprüngliche Idee bei Antragstellung war, zur 3D Rekonstruktion auf am FHG-HHI vorhandenes Know-How zur Stereorekonstruktion aufzubauen, jedoch musste dieser Ansatz verworfen werden. Die vorhandenen Verfahren gehen von einer sehr engen und gleich gerichteten Positionierung der Kameras aus (Stereosetup, an den menschlichen Augen orientiert), wobei die verschiedenen Ansichten einen sehr großen Überlappungsbereich haben und die korrespondierenden 2D Abbildungen der

3D Punkte in den Ansichten relativ eng beieinander liegen. Bei einer Überwachungsanwendung hat man aber genau gegenteilige Vorgaben. Der zu überwachende Bereich soll möglichst gut mit sowenig Kameras wie möglich abgedeckt werden, um die Kosten gering zu halten. Daher musste ein völlig neuer Ansatz entwickelt werden. Hierzu wird die Szene in statische (Hintergrund) und dynamische (bewegte Verkehrsobjekte) Bestandteile aufgeteilt, die getrennt voneinander modelliert und in einem Gesamtmodell integriert und visualisiert werden.

Der Ansatz geht von einem statischen Kamerasetup aus, das beim Aufbau kalibriert wird. Zur Bestimmung der extrinsischen und intrinsischen Kameraparameter wurde ein geeigneter Algorithmus implementiert. In allen Videosequenzen werden bewegte Objekte vom statischen Hintergrund segmentiert. Auch hierzu wurde ein bekannter Algorithmus implementiert und optimiert.

Das Hintergrundmodell wird interaktiv aus generierten Hintergrundbildern erzeugt, wobei Vorwissen über die 3D Szene ausgenutzt wird. Die vorhandenen segmentierten Kameraansichten werden in einem gemeinsamen 3D Modell durch Rückprojektion überlagert.

Für eine möglichst realistische Approximation von Zwischenansichten bei Navigation in der Szene wurde ein Verfahren zum View-dependent Texture Mapping entwickelt. Hierbei wird je nach virtueller Position in der Szene eine gewichtete Überlagerung der Kamerabilder vorgenommen, wobei die Gewichtung von der geometrischen Relation der virtuellen Ansicht zu den verfügbaren Kamerapositionen abhängt. Je näher die virtuelle Position an einer Kameraposition ist, desto stärker geht die entsprechende Ansicht ein. Dieses Verfahren wurde als Erweiterung in einen MPEG Standard eingebracht (s.u.).

Die Rekonstruktion der bewegten Objekte geht von den segmentierten Konturen aus (s.o.). Diese werden über die Zeit in den Kamerabildern verfolgt (2D Tracking). Alle Ansichten eines bewegten Objekts werden durch Rückprojektion und geometrische Datenfusion zusammengefasst. Schließlich wird die 3D-Trajektorie über die Zeit bezüglich des 3D Modells verfolgt (3D Tracking). Die 3D Rekonstruktion erfolgt durch Auswahl eines vordefinierten 3D Modells aus einer Datenbank, wobei verschiedene Modelle für PKW, Fußgänger, Radfahrer etc. zur Verfügung stehen. Sie werden skaliert und der geschätzten 3D Bewegung gemäß in der Szene ausgerichtet und bewegt. Als Textur werden die segmentierten Originalansichten verwendet.

Schließlich werden statische und dynamische Elemente im Gesamtmodell zusammengefasst. Dieses ermöglicht dann die Funktionalität der freien Navigation

Bis auf eine automatische Auswahl geeigneter 3D Modelle wurden alle Teile des 3D Rekonstruktionssystems implementiert und getestet. Die Module zur Segmentierung und zum 2D Tracking wurden in den Stand-alone-Demonstrator des FHG-HHI integriert und im Livebetrieb getestet. Dieser Teil wurde auf verschiedenen Events (Lange Nacht der Wissenschaften, Internationale Funkausstellung, CEBIT) einem breiten Publikum als Livesegmentierer präsentiert. Die anderen Module wurden bisher noch nicht im Livebetrieb getestet, obwohl die Integration von der Softwareseite aufgrund des durchgeführten Softwareengineering kein Problem darstellen wird.

Die weiteren Module wurden anhand von aufgezeichneten Testsequenzen getestet und optimiert. Dabei konnte die Funktionalität nachgewiesen werden. Bezüglich der Robustheit bleiben jedoch noch einige Probleme. Die Segmentierung ist konzeptionell immer eine Schätzung, die immer mit einer Restfehlerwahrscheinlichkeit behaftet ist. Ziel kann es nur sein, diese durch geeignete Verfahren und deren Optimierung zu minimieren. In den Experimenten sind aus verschiedenen Gründen immer wieder Segmentierungsfehler aufgetreten.

Daher ist zu folgern, dass das entwickelte System zu einer vollautomatischen 3D Rekonstruktion bei Forderung von absoluter Fehlerfreiheit zurzeit nicht geeignet ist. Trotzdem hat es für den Einsatz im OIS-Gesamtsystem einen hohen Wert.

Zum einen kann die Statistik der auftretenden Fehler durch ausgiebige Tests gemessen werden. Es können Kenngrößen wie Fehlerwahrscheinlichkeit, Varianz, etc. angegeben werden, z.B. auch für verschiedene Wetterverhältnisse. Bei einer Anwendung des Systems zur automatischen Verkehrsanalyse (z.B. Abbiegeverhalten) können diese Kenngrößen einbezogen werden. Die ermittelten Daten müssten dann mit Confidencewerten versehen werden, die sich aus der Fehlerstatistik ergeben.

Bei der Visualisierung führen Segmentierungsfehler zu fehlerhafter Darstellung. Dies kann bei manchen Anwendungen tolerierbar sein, wenn es z.B. um die reine Visualisierung geht. Dann ist auch eine vollautomatische Implementierung möglich. In anderen Fällen kann die Fehlerfreiheit durch interaktive, d.h. benutzerassistierte Verarbeitung gewährleistet werden. Das Szenario hierzu wären Offline-Anwendungen, bei denen mit aufgezeichneten Videodaten gearbeitet wird. Bei Fehlern könnte ein Benutzer eingreifen und entsprechende Korrekturen vornehmen. Ein solcher Aufwand ist in manchen Fällen sicher gewährleistet (z.B. Gerichtsverhandlungen nach strittigen Unfällen).

1.3 MPEG 3DAV

Unter dem Namen 3DAV ist im Rahmen von MPEG zeitgleich mit dem Start von OIS eine neue Initiative zur Standardisierung von 3D-Video und -Audio gestartet worden. Es sollen neue AV-Formate entwickelt werden, die über die heute verfügbaren klassischen 2D Formate mit vordefiniertem Standpunkt hinausgehen. Ein wesentliches Merkmal hierbei ist die Interaktivität, d.h. der Zuschauer soll in gewissen Grenzen seinen Standpunkt und/oder Blickwinkel frei wählen können. Die Entwicklung genau solcher Technologie stellte den Kern des Arbeitspakets zur Zwischenbildinterpolation dar. Die Arbeitsgruppe hat sich daher von Beginn an federführend an 3DAV beteiligt. Herr Dr. Smolic ist Chairman der 3DAV-Gruppe in MPEG, Editor des „Applications and Requirements for 3DAV“-Dokuments, das die Anforderungen eines solchen neuen Standards definiert und Editor des „Report on 3DAV Exploration“-Dokuments, das die Ergebnisse der Arbeiten zusammenfasst.

Die Arbeitsgruppe hat maßgeblich an der Definition und Beschreibung des Arbeitsgebietes (Interactive 3D Video and Audio), der Anwendungsszenarien sowie der abgeleiteten Requirements mitgewirkt. Dabei wurden insbesondere auch die Interessen von OIS vertreten und verwirklicht. Dies bedeutet, dass die entwickelte Technologie zu den Anwendungsszenarien und Requirements von 3DAV passt. Weiterhin wurden in 3DAV experimentellen Untersuchungen durchgeführt, die das Ziel hatten, poten-

tielle Technologie im technischen Detail zu evaluieren. Nach Abschluss der Initiierungsphase (Exploration) werden nun im Rahmen von 3DAV konkrete Standardisierungsarbeiten begonnen.

Durch die intensive und regelmäßige Zusammenarbeit mit weltweit führenden Wissenschaftlern in der 3D Szenenrekonstruktion konnte ein umfassendes Know-how zur Realisierung der Zwischenbildinterpolation aufgebaut werden. Dies war insbesondere wichtig, da das ursprüngliche Antragskonzept, eine disparitätsgestützte Zwischenbildinterpolation durchzuführen, sich für das gegebene Szenen- und Kamerasetup an einer Verkehrkreuzung als ungeeignet erwiesen hat. Der stattdessen verfolgte Ansatz einer Unterteilung der Verkehrsszene in statische und dynamische Anteile und deren getrennte Modellierung durch Multitextureobjekte steht in engem Zusammenhang zu Vorschlägen, die in 3DAV untersucht wurden.

Umgekehrt konnten die Ergebnisse aus VVE in MPEG eingebracht werden. Die entwickelten Methoden zum View-dependent Texture Mapping wurden für eine aktuelle Erweiterung des MPEG-4 Computergrafik Standards Animation Framework eXtension (AFX) vorgeschlagen und akzeptiert. Dieses Ergebnis aus VVE wird damit im neuen Standard weltweite Verbreitung finden.

2. Verwertung

Das entwickelte System zur Videocodierung und –Übertragung und die enthaltenen Module sind nicht nur für das OIS Szenario der Verkehrüberwachung geeignet, sondern können auch für eine Vielzahl anderer Anwendungen der Videoübertragung eingesetzt werden, insbesondere bei Verwendung in Systemen mit mehreren Kameras. In Frage kommen z.B. allgemeine Überwachungssysteme (insb. große Gelände und Gebäude, Flughäfen, Bahnhöfe, etc.), Fernwartung oder Videokonferenzen mit mehreren Teilnehmern. Zurzeit laufen verschiedene Verhandlungen mit öffentlichen Geldgebern und Industriepartnern über aufbauende Projekte sowohl im Bereich Straßenverkehr als auch in anderen Anwendungsgebieten. Auch eine Vermarktung im Rahmen eines Start-up-Unternehmens oder in Lizenz ist möglich zurzeit jedoch nicht geplant.

Die 3D Rekonstruktion stellt eine Software dar, die für eine Verkehrszentrale vorgesehen ist. Sie kann in verschiedener Weise konfiguriert und somit in verschiedenen Gesamtpaketen vermarktet werden. Die Funktionalität steht zwar nicht im Mittelpunkt des OIS-Projekts, stellt aber einen bedeutenden Zusatz dar, der eine Abgrenzung (Added Value) von existierenden Systemen zulässt. Auch das hier aufgebaute Know-how und die Implementierungen lassen sich in vielfältiger Art und Weise weiter verwerten. Wie zum Beispiel die Arbeit der MPEG 3DAV Gruppe und die intensive weltweite Forschung auf diesem Gebiet beweist, wird Verfahren zur interaktiven 3D Navigation in realen Umgebungen in Zukunft ein hohes Potential eingeräumt (interaktive Medien, Entertainment, Kommunikation, Überwachung, Visualisierung, etc.). Durch die Arbeiten in OIS ist das FHG-HHI hierzu hervorragend aufgestellt. Es konnten intensive Kontakte zu Firmen und Instituten aus aller Welt geknüpft werden. Zurzeit laufen Verhandlungen mit öffentlichen Geldgebern und Industriepartnern über aufbauende Projekte in verschiedenen Anwendungsfeldern.

Der Projekterfolg und die internationale Sichtbarkeit des Projekts lassen sich auch anhand der Veröffentlichungen belegen:

- 4 Artikel in namhaften internationalen Fachzeitschriften
- 13 Artikel für namhafte internationale Fachkonferenzen
- 34 MPEG Beiträge
- 3 Beiträge zu Ausstellungen
- 9 eingeladene Vorträge bei namhaften internationalen Instituten, Firmen und Konferenzen
- 4 Diplom- und Studienarbeiten

3. Beitrag zu den förderpolitischen Zielen

Die erfolgreiche Realisierung der Systeme zum Videostreaming und zur Zwischenbildinterpolation stellen einen wichtigen Beitrag zum Erfolg des Gesamtprojekts OIS dar. Dieses wiederum bildet eine Keimzelle des Wachstumskerns „Anwendungszentrum intermodale Verkehrstelematik (AZVT)“ in Berlin/Brandenburg. Der erfolgreiche Verlauf und Abschluss des OIS Projekts (mit seinem Teilprojekt VVE) trug daher maßgeblich zur Etablierung des Wachstumskerns bei. Dies beinhaltet sowohl die Erfüllung der technisch/wissenschaftlichen Ziele als auch die Bildung eines Netzwerks zur Forschung, Entwicklung und Vermarktung. Das FHG-HHI ist Bestandteil dieses Netzwerks und trägt seine spezifische Expertise bei. Die Kontakte zu KMU und anderen Foschungseinrichtungen werden auch in Zukunft für gemeinsame Vorhaben genutzt werden.

4. Zeit- und Kostenplanung

Die Zeit- und Kostenplanung des Projekts wurde eingehalten.