Technische Fachhochschule Wildau

Fachbereich Betriebswirtschaft-Wirtschaftsinformatik

Diplomarbeit

eingereicht von	:	Stefan Weiße
Mat. Nr.	:	98 277 0585
geb. am	:	13.11.1977
Betreuer	:	Herr Prof. Dr.-Ing. Michael Hendrix Herr Dipl.-Wirtschaftsinformatiker (FH) Peter Bernhardt
Themenstellendes Institut	:	Deutsches Elektronen-Synchrotron (DESY) Zeuthen
Tag der Ausgabe	:	30.01.2003

Kurzthema

Design und Implementierung der Übertragung von Videobildern im PITZ (Photoinjektor-Teststand Zeuthen)

Thema

Diese Diplomarbeit handelt vom Design und von der Implementation des Videoanalysesystems. Dieses besteht aus einem Grabber-Server und einem Standardauswertungsprogramm (Video Client). Besonderes Augenmerk ist in dieser Arbeit auf den evolutionären Werdegang sowie die angewandten Technologien gerichtet.

Danksagung

An dieser Stelle möchte ich Personen danken, ohne die diese Diplomarbeit nicht oder nur unter erschwerten Bedingungen möglich gewesen wäre.

Mein Dank geht an alle Mitarbeiter des DESY Zeuthen, die mich bei meiner Arbeit, insbesondere bei der Erstellung der Diplomarbeit, unterstützt haben. Ganz besonderer Dank geht an Velizar Miltchev vom DESY Zeuthen, der einen Teil der Algorithmen der Programme entwickelt hat und mir auch sonst mit Rat und Tat zur Seite stand. Weiterhin möchte ich meinem Betreuer, Gunter Trowitzsch vom DESY Zeuthen, danken. Die konstante Unterstützung und der Servicegedanke („Ich kümmere mich drum !") halfen mir, mich auf den Kern meiner Arbeit zu konzentrieren.

Besonderer Dank geht auch an die Professoren und Dozenten der Technischen Fachhochschule Wildau, insbesondere Herrn Prof. Dr.-Ing. Michael Hendrix für die sehr praxisbezogene und technisch orientierte Ausrichtung des Wirtschaftsinformatikstudiums. Herrn Dipl.-Wirtschaftsinformatiker Peter Bernhardt danke ich für die Zweitbetreuung meiner Diplomarbeit.

Inhaltsverzeichnis

1 Einleitung

1.1 Abstrakte Beschreibung

PITZ[1] (Photoinjektor-Teststand Zeuthen) ist ein Teststand im DESY Zeuthen für die Erforschung und Entwicklung von lasergetriebenen Elektronenquellen (rf-gun) für Freie-Elektronen-Laser (FEL) und Linearbeschleuniger. Die Optimierung einer Elektronenkanone ist nur durch ein breit gefächertes Diagnosesystem möglich. Ein Kernpunkt in diesem Diagnosesystem ist das Videosystem. Das Ziel ist es, Position und Größe des Elektronenstrahles an verschiedenen Punkten entlang der Strahlenführung durch Diagnosewerkzeuge zu messen.

Das Videosystem dient dazu, vom Elektronenstrahl an den Diagnosewerkzeugen erzeugte Bilder aufzunehmen und über das Netzwerk zu transportieren, so dass an geeigneten PC's innerhalb des DESY Zeuthen remote die unterschiedlichsten Auswertungen vorgenommen werden können. Das System ist auch nötig, da sich während des Experimentes auf Grund von Strahlung niemand im Experimentiertunnel aufhalten und die Experimente direkt beobachten kann.

Diese Diplomarbeit handelt vom Design und von der Implementation des Videoanalysesystems. Besonderes Augenmerk ist dabei auf den evolutionären Werdegang sowie die angewandten Technologien gerichtet.

1.2 Motivation

Die Motivation zur Diplomarbeit entstand aus einem eigenen Projekt während der Semesterferien vor der Diplomphase. Es handelte sich um Echtzeitvideoübertragung über ein lokales Netzwerk. Im DESY Zeuthen gab es eine Einzelplatzlösung, die vielen Anforderungen nicht entsprach, aber kurzfristig Basisfunktionen bereitstellte. Es lag nahe, die selbst gemachten Erfahrungen bei der Erstellung einer Echtzeitvideoübertragung für die Entwicklung und Implementierung des Videosystems im DESY Zeuthen zu nutzen.

1.3 Einführung in den Photoinjektor-Teststand

Der Photoinjektor-Teststand im DESY Zeuthen dient zur Erforschung von lasergetriebenen Elektronenquellen. Strukturell gliedert sich das Experiment in mehrere Aufbauten: Experimentiertunnel, Serverraum, Klystronhalle, Kontrollraum, Laserraum und Zusatzeinrichtungen wie Stromversorgung und Kühlung.

Im Experimentiertunnel, abgeschirmt durch Beton, sind die Aufbauten wie Hohlraumresonator, Kompensationsmagneten, Diagnosewerkzeuge, Kameras etc. entlang der Strahlenführung aufgebaut. An den Experimentiertunnel schließt sich der Serverraum an, in dem Computer und notwendige Anlagen aufgestellt sind. Über Experimentiertunnel und Serveraum befinden sich der Kontrollraum, der Laserraum und die Klystronhalle. Vom Kontroll-

raum aus wird das Experiment durch die Experimentatoren gesteuert und überwacht. Hier befinden sich die Terminals des Kontrollsystems, mit dessen Hilfe Durchläufe angezeigt, nachvollzogen, gesteuert und überwacht werden können. Im Laserraum werden auf einem gedämpften Tisch die für das Experiment benötigten ultrakurzen Laserpulse erzeugt und über Spiegel in den Tunnel geleitet. In der am Kontrollraum anschließenden Klystronhalle wird durch ein 5 MW Klystron die für das Experiment benötigte Hochfrequenzleistung erzeugt und über Hohlleiter in den Tunnel überführt.

Ziel des Experiments ist es, durch Photoemission Elektronenpakete mit möglichst geringer Emittanz, d.h. Produkt aus Strahlgröße und Winkeldivergenz, zu erzeugen. Der Teststand gilt als Vorbereitung für größere Beschleuniger, wie das Zukunftsprojekt TESLA [2].

1.3.1 Aufbau und Funktionsweise des PITZ-Experimentes

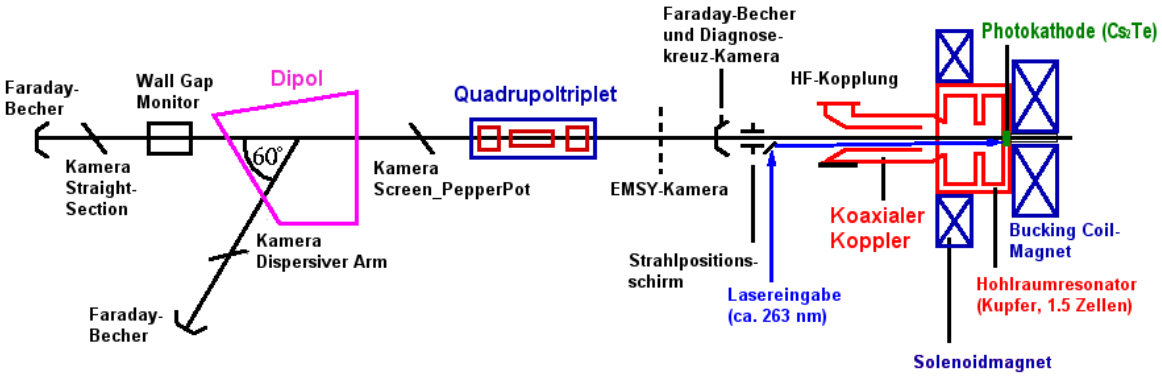

Abb. 1: Schematisches Diagramm des Photoinjektor-Teststandes

Ein Laserstrahl mit einer Wellenlänge von $\lambda = 263\ nm$ wird über den *Lasereingabe-Port* und über Spiegel zur Photokathode geleitet. Durch den Laserstrahl wird auf der *Photokathode* (Cs_2Te) Photoemission hervorgerufen. Die emittierten Elektronen werden in einem 1,5-Zellen Hohlraumresonator aus Kupfer beschleunigt. Die Zahl 1,5 bedeutet das Anderthalbfache der Wellenlänge des Hochfrequenzsystems. Die Elektronenpakete haben beim Austritt aus dem Hohlraumresonator ca. 0,994-fache Lichtgeschwindigkeit. Die Hochfrequenzleistung wird durch ein 5 MW Klystron erzeugt und über Hohlleiter und den *koaxialen Hochfrequenzeinkoppler* in den *Hohlraumresonator* übertragen. Das Elektronenpaket wird dann durch einen *Solenoidmagneten* fokussiert, der mit einem Strom von bis zu 400 Ampere betrieben wird. Ein zweiter Magnet (*Bucking Coil*) kompensiert das magnetische Feld an der Photokathode, um den Anfangsdrehimpuls der Elektronen zu minimieren. Danach misst ein Strahlpositionsschirm die Position des Elektronenpaketes relativ zum Vakuumrohr. Ein *Faraday-Becher* und ein *Beobachtungsschirm* für das Strahlprofil folgen im Strahlverlauf. Etwa einen Meter von der Kathode entfernt schließt sich das Emittanzmesssystem (EMSY) an. Durch ein *Quadru-*

poltriplet kann der Strahl fokussiert und ausgerichtet werden. Das Feld eines *Dipolmagneten* lenkt den Strahl um etwa 60° ab auf den sog. dispersiven Arm, welcher zur Messung der Impulsverteilung der Elektronenpakete benutzt wird. Der *Wall Gap-Monitor* dient zur Messung des Strahlstromes. Im Experiment sind sechs Kameras montiert. Fünf Kameras davon sind auf fluoreszierende Schirme gerichtet, die bei Bedarf in das Vakuumrohr eingefahren werden können.

Kameraname	Beschreibung
Diagnosekreuz	Die Kamera am Diagnosekreuz dient zur Messung von Größe und Position des Elektronenstrahls. Der Schirm ist wie ein Faraday-Becher aufgebaut, so dass gleichzeitig die Ladung des Elektronenpaketes gemessen werden kann.
Screen_PepperPot	Die Kamera an der Position Screen_PepperPot dient nicht nur zur Messung der Größe und Position des Elektronenstrahls, sondern auch zur Beobachtung der Beamlet-Profile, um die transversale Emittanz des Elektronenstrahls zu bestimmen.
Straight Section	Die Kamera ,,Straight Section" dient zur Messung von Größe und Position des Elektronenstrahls.
Dispersiver Arm	Mit der Kamera am dispersiven Arm können Bilder aufgenommen werden, um die Impulsverteilung der Elektronenpakete zu messen. Die Kamera ist nicht für transversale Strahlpositions- und Strahlgrößenbestimmung geeignet.
EMSY	Mit der Kamera am Emittanzmesssystem (EMSY) können Bilder von der Strahlgröße aufgenommen werden, um Emittanzmessungen durchzuführen.
Virtuelle Kathode	Die virtuelle Kathode dient zur Messung der transversalen Größe und Position des Laserstrahls. Die Kamera befindet sich nicht im Elektronenstrahlverlauf. Der Laserstrahl wird vor der Einführung in den Experimentieraufbau (Lasereingabe-Port) durch einen halbdurchlässigen Spiegel aufgeteilt. Ein Teil des Strahles wird auf die Kamera ,,Virtuelle Kathode" geleitet.

1.3.2 Aufbau eines Kontrollsystems

Durch das Kontrollsystem werden die Abläufe im Experiment überwacht, gesteuert, angezeigt und nachvollzogen. Sowohl hardwareseitig als auch softwareseitig besteht ein Kontrollsystem aus mehreren Schichten. In Abb. 2 sind die Hardwareschichten dargestellt:

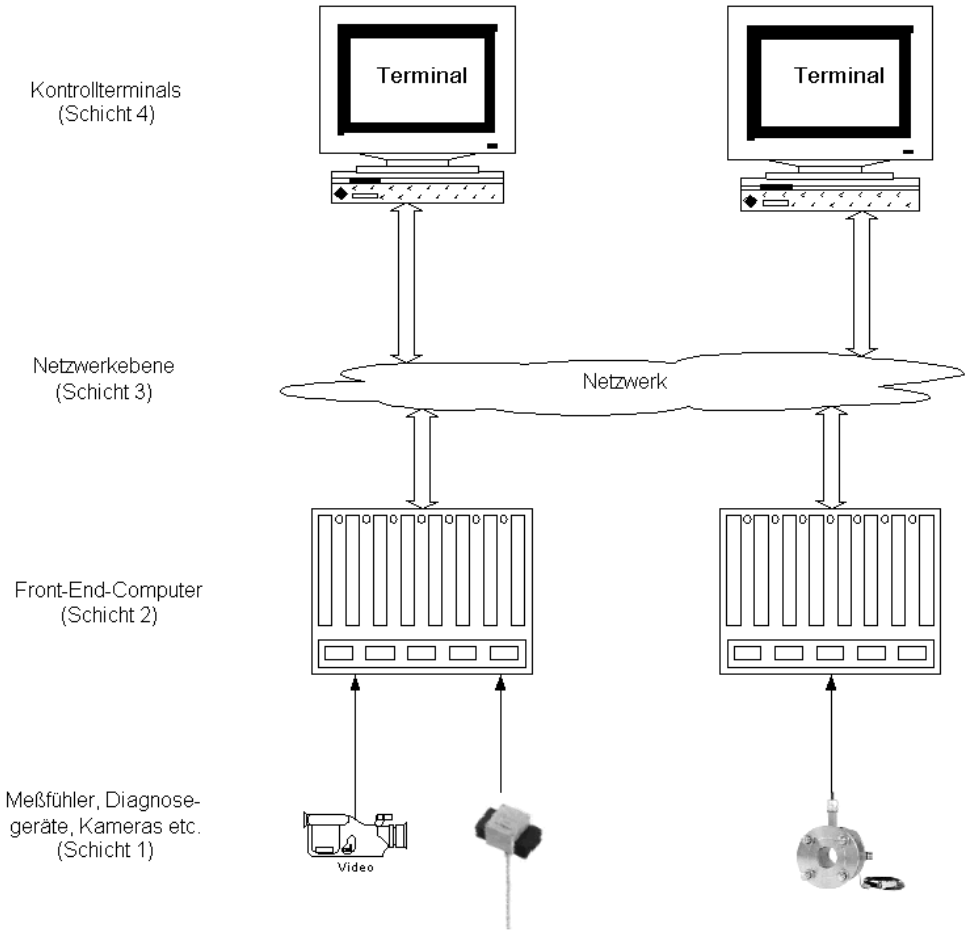

Abb. 2: Hardwareschichten des Kontrollsystems

Es handelt sich um ein verteiltes Kontrollsystem mit Komponenten, die zusammengeschaltet sind. Schicht 1 bildet die Hardware ab, die zur Diagnose benötigt wird. Dazu zählen Diagnoseeinrichtungen, Kameras, Messfühler, Lampen, Spiegel etc. Die Geräte werden von Front-End-Computern gesteuert. Die Front-End-Computer können weiterhin eigene Logik enthalten und ggf. die angeschlossene Hardware autonom steuern. Zwischen den Terminals und den Front-End-Computern befindet sich das Netzwerk zur Kommunikation, d.h. zur Übertragung von Daten und Kommandos. An den Terminals des Kontrollsystems sitzen die Experimentatoren, steuern und überwachen den Experimentablauf.

Neben den Hardwareschichten besteht ein verteiltes Kontrollsystem aus mehreren Softwareschichten, wie Abb. 3 zeigt:

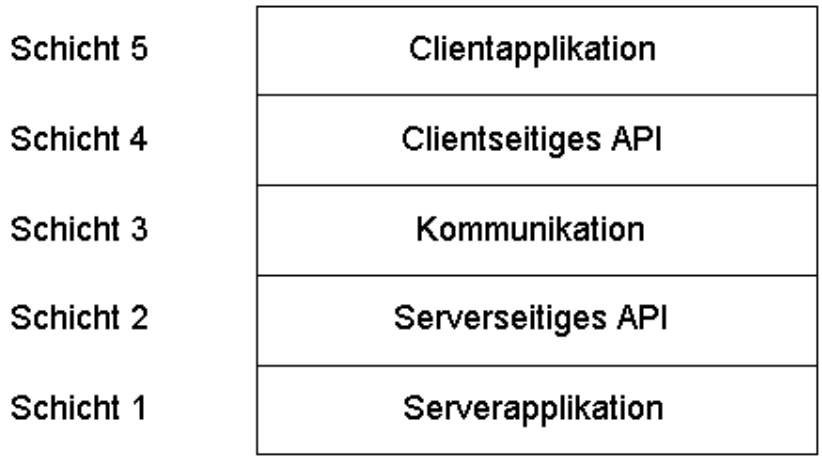

Abb. 3: Softwareschichten des Kontrollsystems

Wie auf der Abb. 3 zu erkennen, gibt es 5 Softwareschichten. Die Clientapplikationen laufen auf den Terminals des Kontrollsystems und werden von den Experimentatoren benötigt, um das Experiment zu steuern und zu verfolgen. Unter den Clientapplikationen befindet sich die Schicht des clientseitigen API's. Es dient zur Abstraktion zwischen Applikation und Kommunikationsschicht und bietet definierte Methoden und Schnittstellen zum Erstellen der Clientapplikationen. Zum Transport von Daten und Kommandos gibt es die Kommunikationsschicht. In dieser Schicht werden Befehle und Daten zwischen Servern und Clients ausgetauscht. Beim PITZ-Experiment findet das DOOCS[1]-Kontrollsystem [3] Anwendung. Dort gibt es in der Kommunikationsschicht drei Protokolle. Das sind Remote Procedure Calls, das TINE-Protokoll [4] und Channel Access. Auf der Serverseite (Front-End-Computer) gibt es auf unterer Schicht das serverseitige API mit definierten Schnittstellen und Methoden zur Programmierung von Serverapplikationen. Darauf bauen die Serverapplikationen auf, welche angeschlossene Geräte wie Lampen, Messfühler, Kameras und Diagnoseeinrichtungen steuern bzw. auslesen.

[1]DOOCS - The Distributed Object Oriented Control System

2 Problemstellung

2.1 Ist-Zustand

Zur Zeit meines Beginns beim DESY Zeuthen war die Hardwareinstallation bereits festgelegt. Erste Videoauswertungen wurden mit einer Grabberapplikation vorgenommen, die als Einzelplatzanwendung nur sehr eingeschränkte Auswertungen erlaubte, störanfällig war und in bestimmten Betriebsarten regelmäßig abstürzte. Die Grabberapplikation läuft auf dem Rechner, in dem die Framegrabberkarte installiert ist. Bedingt durch die Verkabelung der Komponenten steht dieser Rechner im Serverraum, während die Experimentatoren sich normalerweise im Kontrollraum aufhalten. Ein Problem dieser Einzelplatzlösung war behoben worden, in dem ein Physiker vom DESY Zeuthen an die Grabberapplikation vom Berliner Synchrotron (BESSY) [5] eine Socketverbindung anprogrammierte, damit man an einem Rechner im Kontrollraum Videobilder betrachten kann. Mit dieser anprogrammierten Lösung war nur eine einzelne Punkt-Zu-Punkt-Verbindung möglich. Die Übertragung erfolgte unkomprimiert und belastete somit das Netzwerk sehr stark. Weiterhin wurde auf Socketklassen zurückgegriffen, die nicht besonders performant waren.

2.2 Resultierende Probleme

Die oben genannten Nachteile führten zu folgenden Problemen, die durch ein neues System behoben werden sollen:

1. Stabilität

   Unter besonderen Umständen stürzt die alte Grabberapplikation ab. Insbesondere wenn Kabel zu einer der Kameras abgezogen werden, führte das zu einem Crash des Programms, welcher nur durch einen Neustart des Windows-Systems zu beheben war. In einer Testeinrichtung wie PITZ ist es natürlich, dass durch notwendige Umbauarbeiten Kabel häufiger umkonfiguriert werden. Die Kameras befinden sich weiterhin nicht unter direkter Kontrolle des Grabber-PC's, sondern werden durch eine Fernsteuerung im Kontrollsystem des Experiments aus- und eingeschaltet. Auch durch Ausschalten der Kameras kommt es zum Absturz der Grabberapplikation.

2. Fernsteuerbarkeit

   Bedingt durch die Verkabelung steht der Grabber-PC im Serverraum. Möchte jemand einen Parameter an der Grabberapplikation umstellen, so muss er sich in den Serverraum begeben. Fernsteuerung der Parameter und Einstellungen ist nicht möglich.

3. Begrenzung auf einen Clientcomputer

   Durch die anprogrammierte Socketverbindung am alten System war es nur möglich, eine einzige Applikation mit Videobildern zu beliefern. Der gleichzeitige Empfang von Videobildern in mehreren verschiedenen Applikationen, ggf. auf verschiedenen Rechnern, war nicht möglich.

4. Kompression

   Im alten System wurden die Bilddaten unkomprimiert über das Netzwerk übertragen. Das erforderte eine hohe Netzwerkbandbreite.

5. Echtzeitauswertung

   Die alte Grabbingapplikation vom Berliner Synchrotron BESSY erlaubte einfache Analyse des Strahlflecks. Position und Größe konnten bestimmt werden. Weitergehende Analysen und Filterungsfunktionen für die Videodaten waren nicht möglich.

2.3 Anforderungen an das neue System

Durch die Problemstellung im letzten Abschnitt ergeben sich die Anforderungen an das neue System:

1. Strukturelle Anforderungen

   Zu den strukturellen Anforderungen gehören Aufgabenteilung, Integration in bestehende Systeme und Schichtentrennung. Bedingt durch die bauliche Anordnung der Grabber-PC's soll ein Client/Server-Konzept eingeführt werden. Innerhalb dieses Konzeptes erscheint eine Schichtentrennung in First-Level Applikation (Server), Kommunikationsschicht zur Verbindung von Server und Clients und Second-Level-Applikationen (Clients) sinnvoll. Durch diese Modularität soll sichergestellt werden, dass die Lösung durch Austausch von einzelnen Komponenten jederzeit an eine neue Umgebung angepasst werden kann. Weiterhin soll sich die zu entwickelnde Lösung in bestehende Systeme einbinden lassen. Dabei handelt es sich in erster Linie um das Kontrollsystem.

2. Stabilität

   Der Serverteil des Videosystems soll die Stabilitätsprobleme der alten Lösung nicht aufweisen. Auch bei Umbauarbeiten im Experimentiertunnel und der damit verbundenen Abschaltung/Neukonfiguration der Kameras soll der Rest des Videosystems voll funktionsfähig bleiben, so dass nach dem Wiedereinschalten der Kameras ein Weiterbetrieb ohne Administrationsarbeiten möglich ist. Es kann nicht toleriert werden, dass das Serverprogramm abstürzt.

3. Fernsteuerbarkeit

   Alle wesentlichen Parameter in der Grabberapplikation müssen remote einstellbar sein.

Das betrifft speziell die Auswahl des Kameraports, zusätzliche Einstellungen des Grabbers und Softwareparameter der zu erstellenden Lösung. Im Idealfall soll der Server seinen Dienst verrichten, ohne dass jemand zu dem Computer direkten Zugang benötigt.

4. Mehrere gleichzeitig verbundene Clientprogramme
   Durch die neue Softwarelösung soll eine Auswertung der Videodaten im gesamten Zeuthener Institut möglich sein. Es soll möglich sein, dass unterschiedliche Auswertungen gleichzeitig an verschiedenen Orten erfolgen können.

5. Kompression
   Um das Netzwerk bei mehreren Benutzern nicht übermäßig zu belasten, soll auf die zu übertragenen Videobilder eine Kompression angewendet werden. Ein wichtiger Punkt ist die **verlustfreie** Kompression und Dekompression der Videobilder als Grundlage für die zu erfolgenden wissenschaftlichen Auswertungen.

6. Echtzeitauswertung
   Es sollen clientseitig Standardauswertungsmöglichkeiten für die auf den Videobildern enthaltenen Strahlflecke durchgeführt werden können. Wenn möglich soll diese Auswertung in Echtzeit erfolgen. Zu den Auswertungsmöglichkeiten gehören Falschfarbenmodus, Normalisierung, Strahlpositions- und Strahlgrößenberechnung, Hintergrundsubtraktion und Projektionsdarstellung. Die Algorithmen sollen sich auf eine rechteckige Auswahlregion (Area of Interest) innerhalb des Videobildes beschränken lassen.

3 Verwendete Standards

Für die zu planende Entwicklung der Videolösung soll - wo möglich - auf Standards zurückgegriffen werden. Dabei handelt es sich um den Austausch der Daten zwischen First- und Second-Level-Applikationen in der Netzwerkschicht, um die Videokompression und um eine Klassenbibliothek zur Anwendungsentwicklung.

3.1 Sockets

Sockets, entwickelt als Interprozesskommunikation unter Unix, stellen eine einfache Möglichkeit da, zwischen zwei Endpunkten Informationen (Daten) auszutauschen. Sockets bauen im ISO/OSI Schichtenmodell auf der Schicht 4 (Transportschicht) auf. Im TCP/IP- Schichtenmodell bewegt man sich bei der Programmierung auf dem Host-to-Host-Transport-Layer. Netzwerkanwendungen, die Sockets benutzen, bauen oft auf dem Client/Server-Schema auf. Der Server bietet Dienste an, der Client fragt diese nach. Es gibt eine Reihe von Anwendungen und Protokollen, die auf einer Socketschnittstelle aufsetzen, z.B. HTTP, FTP und das Network File System (NFS) [6].

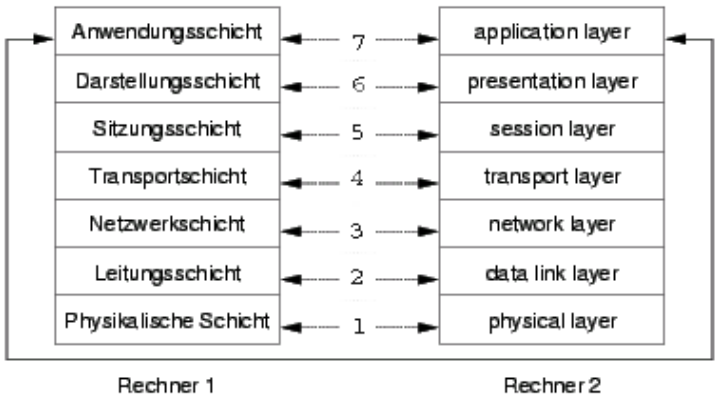

Abb. 4: ISO/OSI Schichtenmodell

Es gibt zwei verschiedene Arten von Sockets: Streaming Sockets und Datagram Sockets. Erstgenannte entsprechen einer Punkt-Zu-Punkt Verbindung zwischen zwei Rechnern und bauen auf TCP (Transmission Control Protocol) auf. Sie garantieren, dass die Daten unverfälscht und in der richtigen Reihenfolge ankommen. Die in dieser Arbeit besprochenen und verwendeten Sockets sind ausschließlich Streaming Sockets. Die andere Socketart sind sogenannte Datagram Sockets. Diese bauen nicht auf TCP, sondern auf UDP (User Datagram Protocol) auf. Hier gibt es keine Garantie, dass die Daten ankommen. Auch die Reihenfolge der ankommenden Pakete kann differieren, d.h. sendet man Paket 1, 2, 3 ins Netzwerk, kann es passieren, dass am anderen Ende Paket 2, 3, 1 herauskommt. Datagram Sockets werden in dieser Arbeit nicht besprochen.

Da die Programme unter Microsoft Windows entwickelt werden, wird die Programmierschnittstelle Winsock (Windows Sockets) benutzt. Darunter versteht man die Windows-spezifische Implementation eines Sockets. Eine genaue Erläuterung findet sich auf dem Microsoft FTP Server[7].

3.2 Das TINE Protokoll

Beim TINE-Protokoll (Three-fold Integrated Network Protocol) handelt es sich um ein vom DESY entwickeltes Kontrollsystemprotokoll, welches auf mehreren Plattformen unter Einsatz von mehreren Netzwerkprotokollen funktioniert. Ein Kontrollsystem ist eine immanent wichtige Komponente in einem physikalischen Experiment wie dem Photoinjektor-Teststand Zeuthen. Mit Hilfe dieses Systems werden die Abläufe im Experiment gesteuert, angezeigt und überwacht. Das TINE Protokoll deckt die Schichten 2-4 eines Kontrollsystemes ab (siehe Abb. 3 in Abschnitt 1.3.2) und wird hier benutzt, um das Serverprogramm, welches die Bilder aufnimmt, zu steuern. Durch dieses Protokoll ist es möglich, das Serverprogramm von verschiedensten Applikationen und Plattformen zu steuern.

3.3 VCM - Video Compression Manager

Der Video Compression Manager[8] ist eine Softwarekomponente innerhalb des Windows Betriebssystems, welche dazu dient, den Aufruf von Bildkompressionskomponenten (sog. CODECs[2]) durch das Anwendungsprogramm zu steuern. Mit Hilfe dieses Managers ist es möglich, dass eine Applikation Bilder bzw. Bildsequenzen unkompliziert komprimieren oder dekomprimieren kann.

Innerhalb dieses Projektes wurde der VCM dazu benutzt, den Videokompressions-CODEC HuffYUV anzusprechen, um die Bilddaten auf dem Server zu komprimieren bzw. auf dem Client wieder zu dekomprimieren.

3.4 MFC - Microsoft Foundation Classes

Bei MFC handelt es sich um eine von Microsoft entwickelte Klassenbibliothek, die das Erstellen von Windows-Programmen entscheidend vereinfacht [9]. Die Funktionalität der Windows-APIs wird in Klassen gekapselt und steht dem Programmierer zur Verfügung. Weiterhin gibt es, wie z.B. im JAVA2 SDK API, viele allgemein verwendbare Klassen, wie Datentypklassen und verkettete Listen. Im Rahmen der Visual-C++ Entwicklungsumgebung können die in MFC eingeführten Klassen einfach verwendet werden und sind durch die beigefügte MSDN Dokumentation auch gut dokumentiert. Im Rahmen dieses Projektes wurde MFC dazu verwendet, die Programmentwicklung zu beschleunigen und zu vereinfachen.

4 Grundlagen

In der Grabberapplikation und dem Standardauswertungsprogramm kommen diverse Algorithmen zum Einsatz. Nach der Darlegung der Videokompression, die beide Applikationen betrifft, werden die Grundlagen der Analyse der Videodaten auf der Clientseite beschrieben.

4.1 Videokompression

Bedingt durch die begrenzte Übertragungsbandbreite wird auf der Netzwerkschicht eine Videokompression auf jedes einzelne Videobild angewendet. Die Kompression soll mehreren Anforderungen genügen.

- Verlustfrei

  Die Kompression muss verlustfrei erfolgen, so dass jedes Bild auf dem Client bitgenau mit dem aufgenommenen Originalbild auf dem Server übereinstimmt. Eine verlustbehaftete Übertragung würde die Berechnungen durch das clientseitige Auswertungsprogramm verfälschen.

[2] Kurzform für COmpression/DECompression

- Echtzeitanforderung

  Der Kompressionsalgorithmus sollte in der Lage sein, bis zu 10 Bilder je Sekunde in Echtzeit zu komprimieren. Der dazu passende Dekompressionsalgorithmus sollte auf dem Client in der Lage sein, diese 10 Bilder in Echtzeit entsprechend zu dekomprimieren. Die Beschränkung auf 10 Bilder pro Sekunde ist durch grundlegende Parameter der Gesamtanlage bestimmt.

- Quellcodeverfügbarkeit

  Der Algorithmus sollte zumindest für die Dekompression im Quellcode frei verfügbar sein, da die Dekompression auf verschiedene Plattformen portiert werden soll.

Es wurden Recherchen durchgeführt, um einen passenden CODEC zu finden. Der ausgewählte CODEC ist HuffYUV[10] und wurde von einem Studenten der Universität Berkeley geschrieben. Der CODEC entspricht den oben genannten Anforderungen. Er kann auf heutigen Rechnern in Echtzeit 10 Bilder pro Sekunde komprimieren und dekomprimieren, arbeitet verlustfrei und der Quellcode ist frei verfügbar, so dass Portierungen auf andere Betriebssysteme und Programmierumgebungen problemlos durchzuführen sind.

Der CODEC basiert auf einem Huffman-Algorithmus mit statischen Tabellen. Die Kompression mit dem Huffman-Algorithmus basiert auf der Annahme, dass verschiedene Bytewerte (Zeichen) verschieden häufig im Datenstrom vorkommen. Häufig auftretende Zeichen werden mit wenigen Bits, selten auftretende Zeichen mit vielen Bits kodiert. Dadurch wird die Datenmenge, die ein Bild beansprucht, in der Praxis um die Hälfte verringert. Inwieweit die Kompression wirklich verlustlos ist, wird in einem späteren Prototyp getestet.

Aus bereits gemachten Erfahrungen beim Einsatz des CODEC wusste ich, dass er in der Lage ist, 25 Bilder pro Sekunde bei einer Vollbildauflösung (768x576 Bildpunke) auf einem Pentium III 700 in Echtzeit zu komprimieren und zu dekomprimieren. Von der Geschwindigkeit her ist er für 10 Hz Framerate geeignet.

4.2 Grundlagen der Videoanalyse

4.2.1 Hintergrundsubtraktion

In den auf dem Server aufgenommenen Bildern ist nicht nur der Elektronenstrahl zu sehen, sondern auch Störsignale, wie z.B. Dunkelstrom oder Reflexionen der Messinstrumente. Es muss eine Möglichkeit geschaffen werden, diese Störsignale aus dem Videobild zu eliminieren. Ein einfaches Verfahren, solche störenden Anteile im Videobild zu entfernen, ist es, ein Hintergrundbild aufzunehmen, welches von jedem vom Server empfangenen Bild subtrahiert wird.

Bei einer normalen Hintergrundsubtraktion subtrahiert man ein Hintergrundbild (einmal aufgenommen) von einem Videobild mit dem Signal. Es gibt viele Algorithmen, die Hinter-

grundsubtraktion weiter zu verbessern, um die Störsignale möglichst vollständig herauszufiltern. In dieser Diplomarbeit werden, neben dem einfachen Verfahren, zu einem bestimmten Zeitpunkt genau ein Hintergrundbild zu nehmen, noch zwei weitere Verfahren erläutert.

Die einfache Subtraktion

Bei der einfachen Subtraktion wird zu einem bestimmten Zeitpunkt ein einzelnes Hintergrundbild aufgenommen, was von allen Bildern, die darauf folgend vom Server geliefert werden, subtrahiert wird. Der Nachteil ist, dass es sich um keine nahezu vollkommene Hintergrundsubtraktion handelt. Ändert der Dunkelstrom nur etwas sein Erscheinungsbild (z.B. pulsiert er), so wird ein Teil des Dunkelstroms auf dem nächsten Bild wieder mit angezeigt. Das ist störend und kann die Ergebnisse verfälschen. Auf der folgenden Abbildung ist die Hintergrundsubtraktion am Beispiel des Dunkelstroms veranschaulicht. Die Bilder wurden im Rahmen des Experimentes am 31. August 2002 an der Stelle des ‚Pepper Pot' aufgenommen.

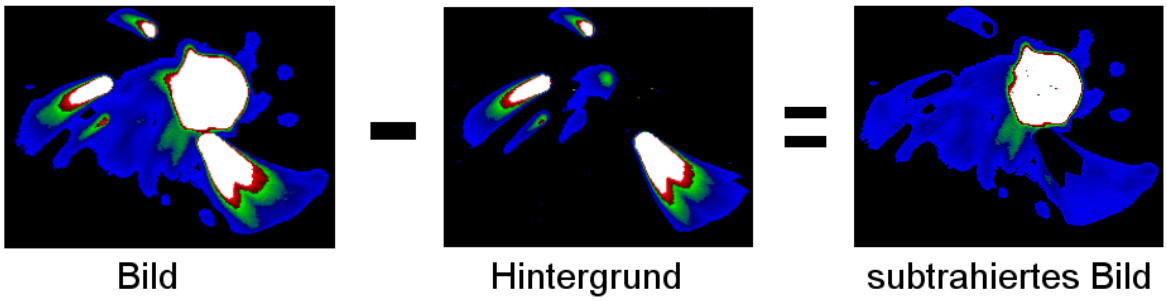

Abb. 5: Hintergrundsubtraktion

Das Originalbild (links) zeigt den Strahlfleck, der durch den Dunkelstrom umrandet wird. Durch das Hintergrundbild (Mitte) wird auf dem rechten Bild ein großer Teil des Dunkelstroms ausgeblendet.

Das Average-Verfahren

Beim Average-Verfahren wird die normale Hintergrundsubtraktion verbessert, in dem nicht nur ein Hintergrundbild aufgenommen wird, sondern mehrere. Im Analyseprogramm wurden gute Erfahrungen mit 5 oder 10 Bildern gemacht. Diese einzelnen Bilder werden dann gemittelt (arithmetisches Mittel) und zu einem Mittelwertbild zusammengefasst. Die folgende Formel veranschaulicht den Ablauf:

$$bkg[i,j] = AVG\{img_k[i,j]\} \left| \begin{array}{l} k \in [1..N] \\ i \in [1..768] \\ j \in [1..574] \end{array} \right. \forall\ (i,j)$$

bkg[i,j] bezeichnet den i-ten Pixel der j-ten Zeile des resultierenden Hintergrundbildes. $img_k[i,j]$ bezeichnet den i-ten Pixel der j-ten Zeile der aufgenommenen Bildes mit der Nummer k. Die Anzahl aller Bilder wird mit N bezeichnet. AVG{} berechnet den Durchschnitt (Mittelwert) aller Pixel. Durch Einsatz dieses Algorithmus ergibt sich eine leichte Verbesserung des Ergebnisses der Hintergrundsubtraktion.

Das Envelope-Verfahren

Beim Envelope-Verfahren wird die normale Hintergrundsubtraktion signifikant verbessert. Wie beim Average-Verfahren wird auch hier nicht nur ein Bild, sondern eine Bildfolge verwendet, um für jeden Bildpunkt den maximalen Pixelwert dieser Folge zu ermitteln. Diese Maximalwerte werden dann zum zu subtrahierenden Hintergrundbild zusammengefasst. Die folgende Formel veranschaulicht den Prozess des Envelope-Verfahrens:

$$bkg[i,j] = MAX\{img_k[i,j]\} \left| \begin{array}{l} k \in [1..N] \\ i \in [1..768] \\ j \in [1..574] \end{array} \right| \forall\ (i,j)$$

bkg[i,j] bezeichnet den i-ten Pixel der j-ten Zeile des resultierenden Hintergrundbildes. $img_k[i,j]$ bezeichnet den i-ten Pixel der j-ten Zeile der aufgenommenen Bildes mit der Nummer k. Die Anzahl aller Bilder wird mit N bezeichnet. MAX{} sucht aus allen Pixeln den größten Wert heraus. Diese Methode zeigt im praktischen Umfeld die besten Resultate. Der Nachteil dieses Verfahrens besteht in der Gefahr, dass kleine Werte, sog. Tails, im Videobild unterdrückt werden.

4.2.2 Statistische Analyse

Bei der statistischen Analyse wird mit Hilfe von gewichteten Mittelwerten versucht, den Mittelpunkt des Strahles auf dem Videobild zu ermitteln. Dabei kommen die folgenden Formeln zum Einsatz:

$$< X >= \frac{\sum\limits_i w_i X_i}{\sum\limits_i w_i} \quad (1) \qquad\qquad < Y >= \frac{\sum\limits_i w_i Y_i}{\sum\limits_i w_i} \quad (2)$$

Die erste Formel wird für die Ermittlung des Mittelpunktes in X-Richtung ($< X >$) verwendet. Bei dem Parameter X_i handelt es sich um die X-Koordinate der entsprechenden Pixelposition i im Videobild, bei dem Parameter w_i handelt es sich um den Graustufenwert an der entsprechenden Stelle i. Analoges gilt für (2).

Nach der Berechnung des Mittelpunktes wird die Standardabweichung vom Strahlmittelpunkt (Größe) berechnet. Das geschieht durch Anwendung der Root-Mean-Square (RMS) Formel:

$$S_x = \sqrt{\frac{\sum_i w_i(X_i - <X>)^2}{\sum_i w_i}} \quad (3) \qquad S_y = \sqrt{\frac{\sum_i w_i(Y_i - <Y>)^2}{\sum_i w_i}} \quad (4)$$

In der Formel (3) handelt es sich bei dem Parameter w_i um den Graustufenwert an der entsprechenden Stelle i. X_i bezeichnet die X-Koordinate der aktuellen Pixelposition i. Der Parameter $< X >$ bezeichnet die weiter oben berechnete X-Koordinate des Mittelpunktes. Analoges gilt für (4).

Diese vier Formeln bilden die Grundlage für die Berechnung des Strahlmittelpunktes und der Strahlgröße. Der Algorithmus, welcher diese Formeln verwendet, ist schnell, d.h. er verbraucht wenig CPU-Zeit, aber er reagiert sehr stark auf Störsignale im Videobild. Diesen Nachteil versucht die Fourieranalyse zu beheben, was im nächsten Abschnitt erläutert wird.

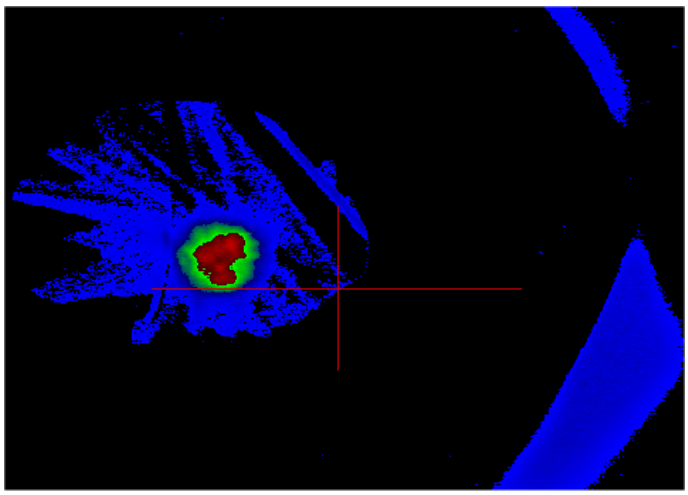

Abb. 6: Statistische Strahlanalyse

In der Abb. 6 wurden die Strahlparameter mit den vorgestellten Formeln berechnet. Der berechnete Strahlmittelpunkt und die berechnete Strahlgröße werden als rotes Kreuz in das Videobild eingezeichnet. Da im Bild Störsignale vorhanden sind, weicht die Berechnung von den tatsächlichen Werten deutlich ab. Dieses Bild, aufgenommen am 08.02.2002 an der Stelle des ‚Pepper-Pot', zeigt einen Strahlfleck mit Teilen des Hintergrundes, die nicht vollständig subtrahiert wurden. Dieses Bild wird bei der Vorstellung der Fourieranalyse erneut verwendet, um das unterschiedliche Verhalten von statistischer Analyse und Fourieranalyse zu verdeutlichen. Dieses Bild zeigt auch wie wichtig eine effektive Hintergrundsubtraktion, insbesondere bei Anwendung der statistischen Strahlanalyse, ist. Für die Anwendung der statistischen Analyse standen fertige Klassen bereit, die integriert wurden.

4.2.3 Fourier-gestützte Analyse

Wie oben erläutert hat die statistische Analyse systembedingte Nachteile. Im Videobild enthaltene Störsignale (Röntgenstrahlen und Reflexionen) fließen in die Berechnung des Strahlmittelpunktes und der Strahlgröße mit ein und können diese signifikant verfälschen. Um diese Nachteile zu umgehen, wird bei der Fourier-gestützten Analyse auf die Projektionen im Videobild eine diskrete Fouriertransformation (DFT) bis zum 5-ten Term angewendet. Diese Transformation verschmiert das Signal soweit, dass Störeinflüsse im Videobild einfach ,,übersehen" werden. Die Fouriertransformation wirkt wie ein Tiefpassfilter. Nachdem die Transformation bis zum 5. Term angewendet wurde, ist die Strahlmitte durch die beiden folgenden Bedingungen definiert:

$$max(I^x{}_{DFT}) = I^x{}_{DFT}(< X >) \quad (5)$$
$$max(I^y{}_{DFT}) = I^y{}_{DFT}(< Y >) \quad (6)$$

In diesen Formeln steht $I^x{}_{DFT}$ und $I^y{}_{DFT}$ für die DFT-expandierten Projektionen. Die Strahlgrößenberechnung basiert auch in der fouriergestützten Analyse auf der Root-Mean-Square (RMS)-Formel aus der statistischen Analyse. Um die Störeinflüsse auch auf die Größenberechnung zu minimieren, wird eine gedachte rechteckige Region um den berechneten Mittelpunkt gebildet, die 2.58 mal so groß ist wie die über das ganze Bild ermittelte RMS-Größe. Der Faktor wurde so gewählt, dass bei einer gedachten Gaußverteilung 99% der Größe des Strahles bei gegebener RMS-Größe abgedeckt werden. In dieser gedachten Region wird dann die RMS-Formel noch einmal angewendet. Die restlichen Teile des Bildes, die eventuell Störsignale enthalten, werden ausgeblendet.

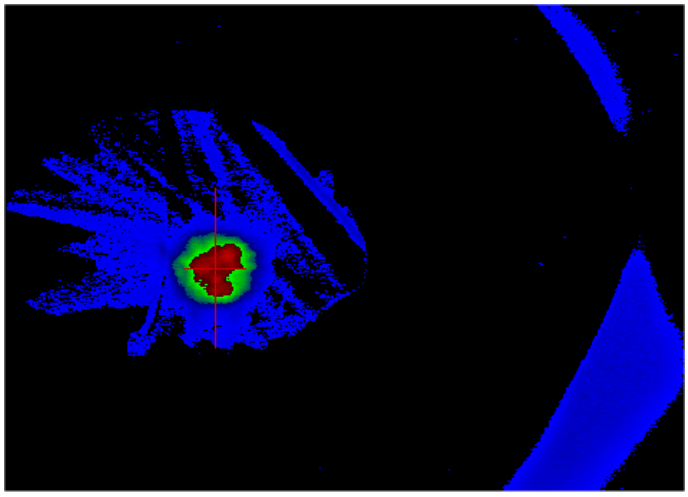

Abb. 7: Fourieranalyse des Elektronenstrahles

In der Abbildung 7 wurden die Strahlparameter mit der Fourieranalyse berechnet. Trotz der im Bild enthaltenen Störsignale liefert die Fourieranalyse auf den gleichen zugrundeliegenden Videodaten wie in Abb. 6 deutlich bessere Werte für die Strahlposition und Strahlgröße.

Für die Anwendung der diskreten Fouriertransformation im Analyseprogramm standen fertige Klassen bereit, die integriert wurden.

4.2.4 Normalisierung

Eine weitere Möglichkeit für die Verstärkung von schwachen Signalen ist, eine Normalisierung auf das Videobild bzw. Teile des Videobildes anzuwenden. Bei schwachen Signalen wird nicht der gesamte Wertebereich (8 Bit) des Videobildes genutzt. Ziel der Normalisierung ist, die Intensität der Pixelwerte zu skalieren, so dass der gesamte Wertebereich genutzt wird. Dies wird durch folgendes Beispiel verdeutlicht. Von folgenden Pixelwerten wird ausgegangen: 8, 16, 25, 28 und 32. Basierend auf dem Pixel mit der größten Intensität (32) kann man einen Faktor berechnen, mit dem alle Pixel multipliziert werden müssen, damit der Wertebereich von 8 Bit ausgeschöpft wird:

$$n = (2^8 - 1)/32 = 7.96875$$

Basierend auf dem ermittelten Faktor kann man folgende Berechnungen auf die Intensitätswerte anwenden:

A	Ausgangswerte	8	16	25	28	32
B	A · 7.96875	63.75	127.5	199.22	223.13	255.0
C	Abrunden(B)	63	127	199	223	255
D	Ergebnisse	63	127	199	223	255

Durch Multiplikation jedes einzelnen Pixels mit dem berechneten Faktor und anschließendem Abrunden wurden die Pixelintensitäten auf den Wertebereich von 8 Bit (0 - 255) skaliert. Um diesen Algorithmus zu verdeutlichen, ist auf dem folgenden Bild ein Elektronenstrahl aus dem laufenden Experiment abgebildet.

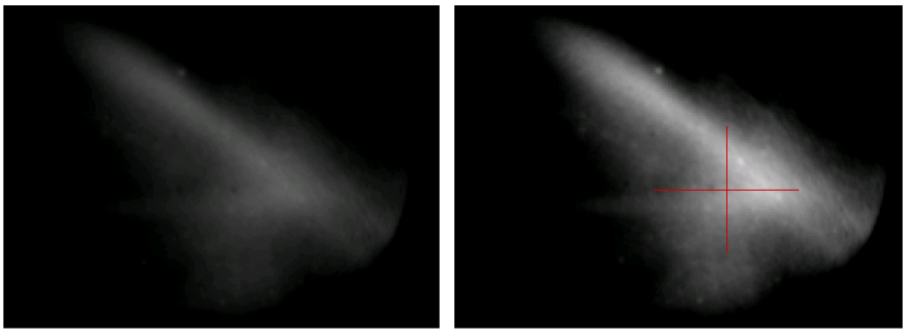

Abb. 8: Nicht-normalisiertes und normalisiertes Bild des Elektronenstrahls

In der Abb. 8 ist auf der linken Seite ein Bild des Flecks ohne Normalisierung abgebildet. Auf der rechten Seite wurde dann der Fleck normalisiert. Man erkennt eine signifikante Verstärkung des Strahlfotos.

Der Vorteil der Normalisierung besteht in der Aufspreizung des Signals. Nachteilig sind die sehr hohe CPU-Belastung durch den Normalisierungsalgorithmus, sowie die Verstärkung von Störsignalen und Rauschanteilen im Videosignal.

4.2.5 Farbkodierung

Im Experiment kann es vorkommen, dass nur ein sehr schwacher Lichtfleck auf dem Videobild sichtbar ist. Bei Darstellung der einzelnen Pixelwerte durch Graustufen kann ein Teil der Information (dunkle Graustufen) im schwarzen Hintergrund des Bildes nicht dargestellt werden. Abbildung 9 und 10 zeigen das gleiche Videobild, einmal in Graustufendarstellung und einmal in Farbkodierung.

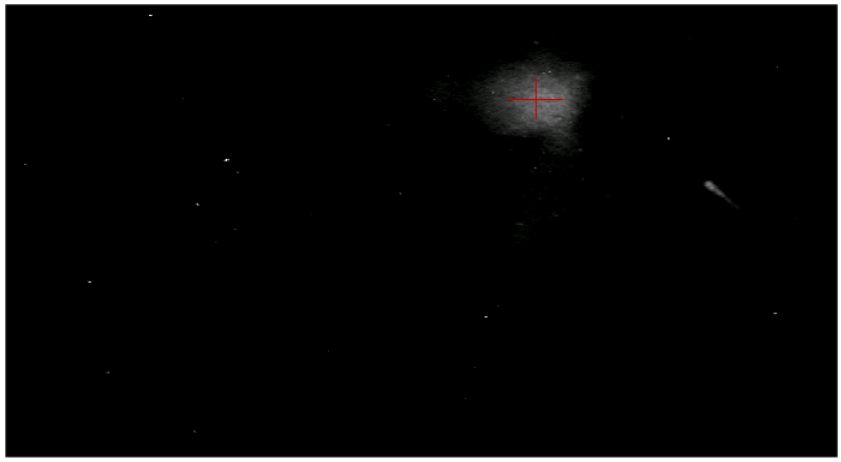

Abb. 9: Graustufenabbildung des Strahls

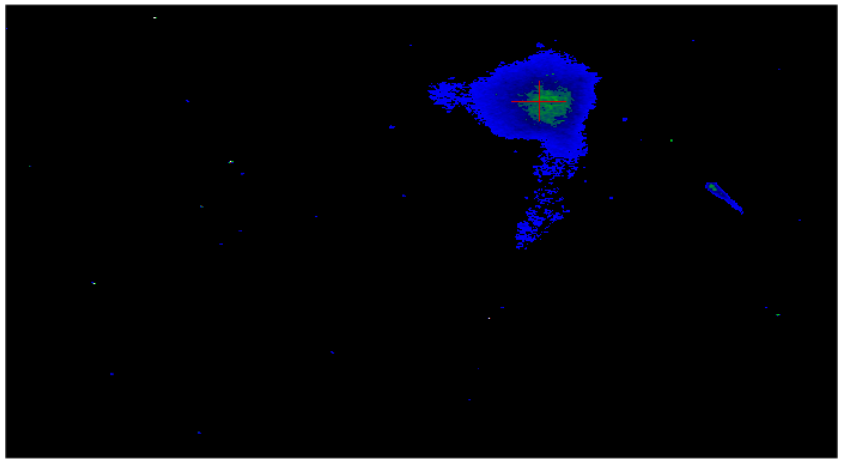

Abb. 10: Farbabbildung des Strahls

Um geringe Signalunterschiede sichtbar zu machen, werden dunkle Stellen durch Blautöne, helle Stellen durch Rottöne und dazwischenliegende Stellen durch Grüntöne visualisiert. Eine exakte Farbtabelle der benutzten Kodierung ist im Anhang aufgeführt. Durch die Falschfarben kann auch bei schwachen Signalen der komplette Elektronenstrahl auf dem Videobild erkannt werden, wie Abbildung 10 zeigt.

Die Vorteile des Falschfarbenmodus liegen in der besseren Erkennbarkeit von schwachen Signalen bzw. Signalunterschieden. Weiterhin hilft die Unterteilung in blaue, grüne und rote Abschnitte, die Intensität des Elektronenstrahls besser abschätzen zu können.

4.2.6 Röntgenstrahlenfilterung

Beim laufenden Experiment entsteht im Experimentiertunnel Röntgenstrahlung. Die Kameras zeichnen diese Röntgenstrahlung mit auf. Die einzelnen Röntgenstrahlen machen sich als weiße Pixel im Videobild bemerkbar und können die Messungen und Berechnungen verfälschen. Deshalb musste ein Weg gefunden werden, diese weißen Pixel durch einen Softwarefilter im Bild zu unterdrücken.

Zum Herausfiltern der Röntgenstrahlen wird der 'discard-single-peak' Algorithmus angewendet. Die Idee dieses Algorithmus ist es, Pixel mit einem Nicht-Null-Wert, welche von Null-Wert-Pixeln (schwarze Pixel) umgeben sind, auszublenden. Man kann diese Logik verallgemeinern, in dem man Pixel, die mehr als p=100% über ihre Nachbarn herausragen, durch einen Pixel ersetzt, der aus dem Durchschnitt aller Nachbarpixel berechnet wird. Der Wert von p ist frei wählbar und hängt von den Arbeitsbedingungen ab.

Der beschriebene Algorithmus filtert Röntgenstrahlen aus den Videobildern heraus, wie auf den beiden folgenden Abbildungen erkennbar ist.

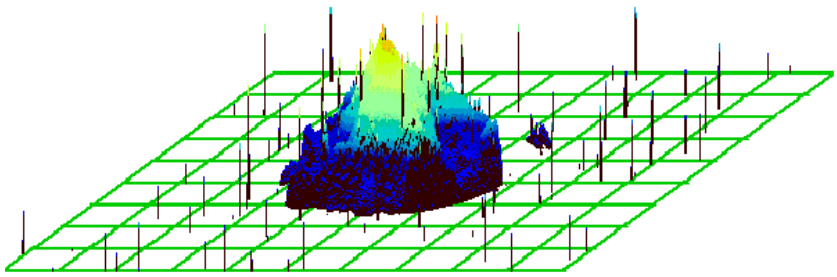

Abb. 11: Nicht-gefiltertes Bild des Strahls

Abb. 11 zeigt eine 3D-Aufnahme vom Emittanzmesssystem (EMSY). Die Röntgenstrahlen erkennt man als einzelne, zufällig verteilte Spitzen im Signal. Abb. 12 zeigt das Resultat der Röntgenstrahlenfilterung mit p=100%. Man kann erkennen, dass die Röntgenstrahlen im Bild unterdrückt sind, aber das Signal durch die Filterung nicht verändert wurde.

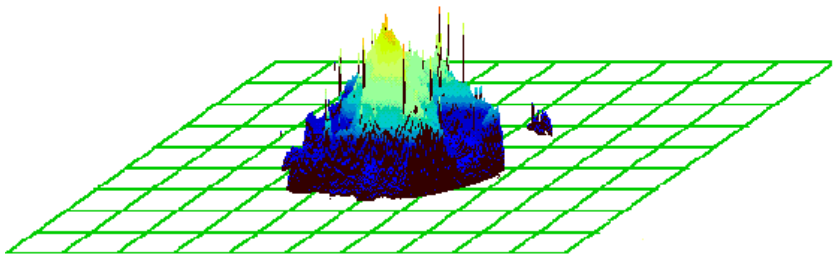

Abb. 12: Gefiltertes Bild des Strahls

5 Fachkonzept und Entwurf

5.1 Festgelegte Hardware

Die Hardware war bereits vor meinem Eintritt ins DESY Zeuthen angeschafft und somit festgelegt. Dabei handelt es sich um die Videokameras und die Framegrabberkarte. Bei den Kameras handelt es sich um das Fabrikat CV-M10 der Firma JAI [11]. Diese Kameras besitzen drei Eigenschaften, die für den Experimentierbetrieb benötigt werden. Bei der ersten Eigenschaft handelt es sich um eine triggergesteuerte Bilderzeugung, d.h. immer wenn ein Triggersignal kommt, nimmt die Kamera ein Bild auf. Das Triggersignal ist nötig, da die Aufnahme der Bilder mit den Makropulsen vom Experiment synchronisiert sein muss. Bei der zweiten Eigenschaft handelt es sich um die Fernsteuerbarkeit. Über eine serielle Schnittstelle kann man die Kameraparameter wie Verstärkung, Shutter-Speed und Triggermodus einstellen. Die dritte Eigenschaft ist der Vollbildmodus, d.h. ein komplettes Einzelbild wird aufgenommen, und nicht, wie normalerweise bei Kameras üblich, zwei Halbbilder mit Zeitversatz.

Bei der Grabberkarte, die die Bilddaten von den Kameras entgegennimmt und digitalisiert, handelt es sich um die PCVision-Karte von Coreco Imaging [12]. Diese Karte nimmt das analoge Bildsignal von der Kamera entgegen, digitalisiert es und legt es im Hauptspeicher des Computers ab. An eine solche Karte kann man über eine Multiplexereinheit 4 Kameras anschließen. Das Grabben ist zu einer bestimmten Zeit immer nur von einer Kamera möglich. Weiterhin gibt es ein API für C/C++, um Anwendungsprogramme für diese Grabberkarte einfacher erstellen zu können. Sowohl die Kameras als auch die Karte sind für einen Graustufenmodus ausgelegt.

5.2 Client/Server-Architektur

Bedingt durch die baulichen Gegebenheiten steht der Server mit der Framegrabberkarte im Serverraum. Das Experiment dagegen wird vom Kontrollraum aus gesteuert und überwacht. Einerseits durch diese bauliche Trennung, andererseits durch die Anforderungen aus

Abschnitt 2.3, ergibt sich eine Client/Server-Architektur mit einer Trennung zwischen Server zum Aufnehmen und Senden der Videobilder und den Clients zum Empfang und zur Anzeige, Speicherung und Analyse der Videobilder. Das Serverprogramm wird auf dem Rechner mit der Framegrabberkarte installiert, während die Clientprogramme im Kontrollraum betrieben werden. Weiterhin soll ein Empfang der Videobilder auch in anderen Räumen, z.B. den Büros der Physiker, möglich sein. Als Clientprogramme sind neben einem Standard-Analyseclient für Windows-Betriebssysteme auch einfache Programme zur Darstellung des Videos oder Anbindungsbibliotheken für andere Betriebssysteme denkbar.

5.3 Merkmale der Architektur

Die Merkmale der Architektur basieren auf den Anforderungen aus Abschnitt 2.3:

1. Integration in das Kontrollsystem

   Um die Integration in das Kontrollsystem zu bewerkstelligen, soll von den zwei im Kontrollsystem verwendeten Netzwerkprotokollen eines unterstützt werden. Dabei handelt es sich um das TINE-Protokoll. Durch dieses Protokoll werden remote alle Komponenten gesteuert und Daten ausgetauscht. Dazu gehört das Umschalten des Kameraports.

2. Mehrere Clients

   An einen Server sollen über das Netzwerk mehrere Clientcomputer angebunden werden können, die alle die gleichen Bilder empfangen. Es sollen so viele Clients, wie für den Betrieb der Anlage und die notwendigen Analysen erforderlich, gleichzeitig an einen Server angebunden werden können. Die Clients sollen in unterschiedlichen Betriebssystemen programmiert werden können.

3. Verlustlosigkeit

   Die Daten (Bilder), die auf dem Client ankommen, sollen eine digitale 1:1-Kopie von den auf dem Server aufgenommenen Bildern darstellen, da die Daten wissenschaftlich ausgewertet werden sollen.

4. Einzelbild und kontinuierlicher Modus

   Vom Client aus soll es möglich sein, eine Einzelbildfolge gegebener Größe aufnehmen zu können (Bildsequenz-Modus, engl. Grab-Mode). Weiterhin soll es einen Modus geben, in dem das Video kontinuierlich empfangen wird (Poll-Modus).

5. Programmiersprache

   Als Programmiersprache wurde C++ aus folgenden Gründen gewählt: Das API der Framegrabberkarte steht für C/C++ zur Verfügung. Weiterhin muss durch die Echtzeitanforderungen eine hardwarenahe Programmierung unterstützt werden. Da bereits

jetzt spätere Erweiterungen und Änderungen abzusehen sind, erscheint eine Nutzung von objektorientierter Programmierung gegeben.

Bedingt durch die Hardware, die baulichen Gegebenheiten und einige Merkmale ergibt sich das folgende schematische Diagramm, auf dem die Client/Server-Architektur erkennbar ist:

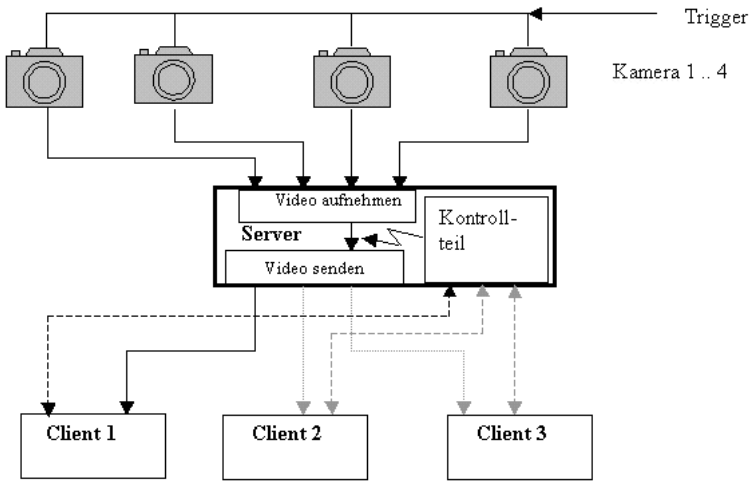

Abb. 13: Schematisches Diagramm der Videoarchitektur

Mit jedem Triggersignal nimmt die Kamera ein Bild auf und sendet es zu der Framegrabberkarte im Server. Das Serverprogramm übernimmt die Aufnahme des Bildes und den Datentransfer an alle verbundenen Clients. Der Kontrollteil steuert den Server und ist remote erreichbar. Im unteren Teil sieht man, dass mehrere Clients angebunden werden können.

Nachdem die Achitektur festgelegt war, galt es in Prototypen wichtige Teile der Funktionalität zu evaluieren.

5.4 Prototyping

5.4.1 Einleitung

Durch die Programmierung von kleinen Prototypprogrammen, die einen Teil der benötigten Funktionalität umfassen, soll sichergestellt werden, dass später bei der Implementierung der kompletten Applikationen keine unvorhergesehenen Probleme auftreten. Abgeleitet von den grundlegenden Anforderungen an das System waren dabei die folgenden Kernpunkte Bestandteil:

- Handhabung des Grabbers
- Einbindung des Netzwerkprotokolls vom Kontrollsystem
- verlustfreie Datenübertragung
- Kompression der Videobilder

5.4.2 Grabbingtest

Eine Grundfunktionalität des Servers in der Videoarchitektur ist das Aufnehmen der einzelnen Videobilder von den Kameras. Diese Funktionalität wird im folgenden als ,,Grabbing" bezeichnet. Bezüglich des Grabbings der Videobilder gab es zwei grundlegende Anforderungen:

- Stabilität
  Der Server sollte unter keinen Umständen abstürzen bzw. sich abnormal verhalten, auch wenn die Kameras während der Aufnahme ausgeschaltet werden oder Kabel abgezogen werden. Im Idealfall soll nach Einschalten der Kameras oder Anklemmen des Kabels das Grabbing weiterlaufen.

- Genauigkeit
  Der Server soll zu jedem Trigger der Kamera, der vom Timingimpuls der Anlage abgeleitet wird, ein Bild aufnehmen. Es soll kein Bild mehrfach aufgenommen oder bei der Auslese übersprungen werden.

Die zu nutzenden Eigenschaften der Hardware und der Software (API) sollen durch die Programmierung eines Grabbingprototypen evaluiert werden. Bei dem alten Grabbingprogramm ,,Pitzcam" des Berliner Synchrotrons BESSY gab es Probleme beim Ausschalten der Kamera bzw. dem Abklemmen des Kamerakabels. Das Programm stürzte ab. Es ließ sich zwar noch beenden, aber bevor nicht ein Neustart durchgeführt wurde, konnten keine Bilder mehr aufgenommen werden (siehe Abschnitt 2.2).

Basierend auf den Quellen des Pitzcam-Programms wurde ein erster Grabbingprototyp geschrieben, welcher das gleiche Absturzverhalten zeigte. Nachdem die Meldung, dass ein Feh-

ler im Grabbingtreiber aufgetreten ist, angezeigt wurde, war kein weiteres Aufnehmen von Bildern mehr möglich. Nach Abschalten der Anzeige der Fehlermeldung des Grabbingtreibers funktionierte das Grabbing-API und der Treiber problemlos. Die Untersuchungen ergaben, dass die Fehlerbehandlung vollständig durch den Grabbingtreiber erfolgen muss, und nicht durch das Anwendungsprogramm erledigt werden darf. Nun konnten im laufenden Betrieb sowohl die Kabel abgezogen, als auch die Kamera ausgeschaltet werden. Das Programm stürzte nicht mehr ab. Man muss nur das Grabbing neu starten und das Programm läuft weiter. Damit ist der erste Punkt, die Stabilität, hinreichend erfüllt, da weitere Unzulänglichkeiten in Bezug auf die Stabilität auch bei längeren Testdurchläufen nicht auftraten.

Beim zweiten Testdurchlauf ging es um die Genauigkeit des Grabbings. Das Grabbing muss mit verschiedenen Triggerfrequenzen laufen. Bisher ist ein Betrieb mit 1, 2, 5 oder 10 Hz durchgeführt worden. Die Trigger werden vom Timingsystem abgeleitet, welches mit einer Genauigkeit von 10^{-8} s arbeitet. Es ist durch diese vorgegebene Genauigkeit zu erwarten, dass die exakten Raten von 1 - 10 Hz auch für das Grabbing zu erwarten sind.

Zur Durchführung der Messung wurde der Grabbingprototyp um die Anzeige der Framerate und eine Log-Datei, in die alle 30 Sekunden die aktuelle Framerate gespeichert wird, erweitert. Bei der Messung der Framerate handelt es sich um eine Durchschnittsmessung seit Beginn des Grabbing. Der Test wurde mit einer Triggerfrequenz von 5 Hz vorgenommen. Man sollte erwarten, dass sich die Framerate mit fortlaufender Zeit immer mehr zu den erwarteten 5 Hz konvergiert. Abb. 14 zeigt einen zu erwartenden Kurvenverlauf. Dieser Verlauf wurde mit einer freilaufenden Kamera, die 25 Bilder pro Sekunde liefert, erstellt.

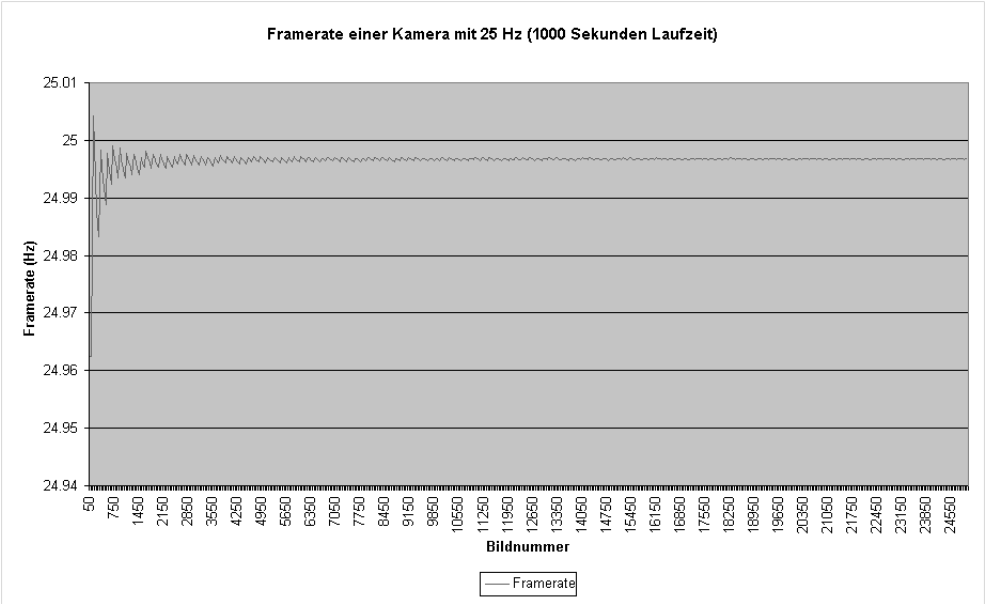

Abb. 14: Bei $\approx$ 25 Hz konvergierende Framerate

Man erkennt, dass sich die Framerate immer mehr einer bestimmten Frequenz annähert. Ein ähnlichen Verlauf sollte man auch bei einem von außen getriggerten Grabbingtest erwarten. Der Test mit dem von außen getriggertem Grabbing lief über 44 Stunden und brachte ein erstaunliches Resultat hervor, welches im Graph in Abb. 15 dargestellt ist.

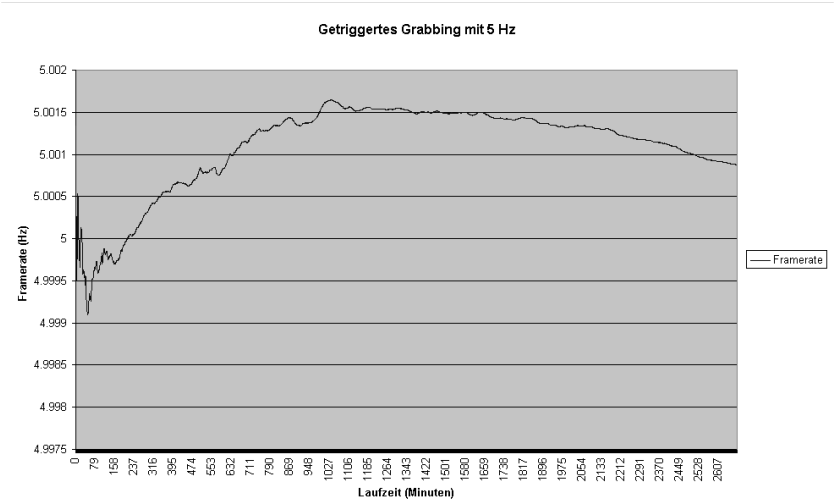

Abb. 15: Verlauf der Framerate bei 5 Hz über 44 Stunden

Der Test lief über 44 Stunden. Man erkennt, dass die Framerate keinesfalls auf eine bestimmte Frequenz konvergiert, sondern über die fast 2 Tage Laufzeit immer wieder stark schwankt. Die Messkurve konvergiert nicht auf 5 Hz. Die Auswertung mit den Fachleuten vom Timingsystem brachte keinen weiteren Hinweis auf das eigenartige Ergebnis. Zu erwähnen ist noch, dass der Verlauf eine Durchschnittsmessung ist. Die aktuelle Framerate war zu bestimmten Zeitpunkten sicherlich noch wesentlich höher oder niedriger, als es die Kurve vermuten lässt.

Ausgehend davon, dass das Timingsystem genau arbeitet, wurde durch Änderungen im Programm (Grabbing-Ansteuerung) unter Zuhilfenahme der ITEX Grabbing API Dokumentation versucht, ein stabiles Grabbingsystem aufzubauen. Leider führten diverse Änderungen nicht zu dem gewünschten Erfolg. Eine auf 5 Hz konvergierende Framerate war nicht möglich.

Der nächste Ansatz versprach weitere Klärung. Mit Hilfe eines Mitarbeiters vom Timingsystem wurde der Triggerimpuls verdoppelt, einmal zu den Kameras geleitet und einmal in ein Modul überführt. Durch ein C-Programm, welches das Modul ansteuert, wurde jeder Triggerimpuls mit einem Zeitstempel versehen und in einer Log-Datei gespeichert. Gleichzeitig wurde der Grabbingprototyp gestartet. Entscheiden war, nachzuweisen, dass der Grabberserver exakt dem Zeitsignal der Anlage folgt, indem die Messkurven vom Modul (Triggerpulsrate) und vom Grabbingprototypen (Framerate) den gleichen Verlauf zeigen. Der Test lief über 17 Stunden und 20 Minuten bei einer Wiederholrate von 1 Hz. Die beiden Messkurven sind in Abb. 16 dargestellt.

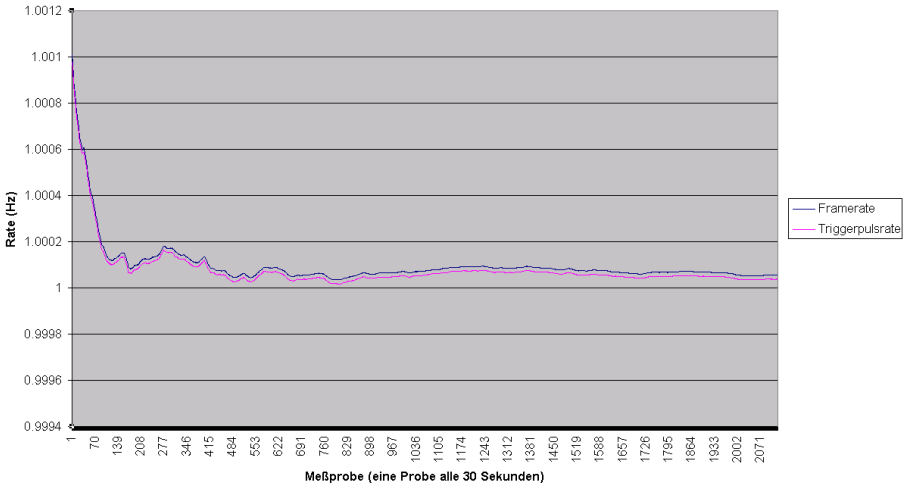

Abb. 16: Messkurven vom Triggermodul und dem Grabbingprototypen

Beide Kurven sind mit der zu erwartenden Genauigkeit deckungsgleich. Die geringen Abweichungen zwischen den beiden Messkurven entstehen durch nicht synchron laufende Echtzeituhren der zur Messung benutzten Computer. Der quadrierte Korrelationskoeffizient (r^2) ergibt für die beiden Kurven eine Übereinstimmung von 99.992 %. Die erste Annahme, dass die abweichende Framerate durch softwaretechnische Probleme verursacht wurde, ist damit hinfällig. Schon der Trigger, der die Kameras veranlasst ein Bild aufzunehmen, weicht von den exakten 1 Hz ab.

Die vorgenommenen Messungen wurden den Mitarbeitern vom Timingsystem vorgelegt. Daraufhin ergab sich, dass das Timingsystem nicht nur von dem sehr genauen Master-Oszillator, sondern auch vom Stromnetz mit unstabiler Netzfrequenz abhängt, da Teile der Elektronik mit dem Stromnetz in Phase sein müssen. Dadurch erklärt sich die Abweichung von der definierten Wiederholrate.

Basierend auf den Resultaten der letzten Messung kann davon ausgegangen werden, dass das Grabbing stabil und genau arbeitet.

5.4.3 TINE-Transfertest

Zur Integration der Softwarelösung in das Kontrollsystem sollte als Netzwerkprotokoll für die Übertragung der Videodaten das TINE-Protokoll verwendet werden. TINE bietet eine an das Kontrollsystem angepasste Abstraktionsschicht mit verschiedenen Methoden des Datenzugriffs (Multicast, Abonnement, Polling, Einzelabfragen).

Zu den Anforderungen an das System gehört der gleichzeitige Empfang des Videos an verschiedenen Computern. Weiterhin wird durch die Bedingung der verlustfreien Videoübertragung eine hohe Netzwerkbandbreite benötigt. Ausgangspunkt war die maximale Bandbreite, die sich aus folgenden Parametern ergibt: Es wird mit einer Auflösung von 768x574 Bild-

punkten gearbeitet. Jeder Pixel hat eine Größe von genau einem Byte. Für den Fall, dass eine Kompression nicht möglich ist, muss das unkomprimierte Vollbild übertragen werden. Bedingt durch die elektronischen Anlagen im Experiment wird das Grabbing maximal mit einer Frequenz von 10 Hz betrieben. Zusätzlich zum Videobild wird noch ein Videoheader übertragen, welcher 88 Byte groß ist. Die Transferrate ist damit:

$$(88\ Byte + 768 \cdot 574 \cdot 1\ Byte) \cdot 10\ \frac{1}{s} = 4409200\ \frac{Byte}{s}$$

Bei einer solch hohen benötigten Bandbreite ist es sinnvoll, den Transfer der Videodaten nicht an jeden Client einzeln über eine Punkt-Zu-Punkt-Verbindung, sondern einen automatischen Verteilmechanismus wie Multicast oder Broadcast vorzunehmen. Das spart Ressourcen auf der Serverseite, da die Daten nur einmal in das Netzwerk gesendet werden müssen und die Verteilung von der Netzwerkhardware übernommen wird. Auch die benötigte Netzwerkbandbreite ist geringer.

Basierend auf der angegebenen Formel müssen im ungünstigsten Fall $4.2\frac{MB}{s}$ verschickt werden. Es wurde ein Prototyp, bestehend aus Server und Client, programmiert, welcher über das TINE-Protokoll per Broadcast[3] Videobilder sendet und empfängt. Ein einzelnes Serverprogramm wurde auf dem Serverrechner gestartet. Im Kontrollraum wurden zwei Clientprogramme auf unterschiedlichen PC's ausgeführt.

Die Testdurchläufe waren enttäuschend. Trotz ständiger Hilfe vom Entwickler des TINE-Protokolles war es nicht möglich, einen stabilen Transfer von $4.2\frac{MB}{s}$ zu etablieren. Die Anzahl der verlorengegangenen Bilder betrug ungefähr 70 %. Wenn die Datentransferrate künstlich auf 1.2 $\frac{MB}{s}$ beschränkt wurde, betrug die Verlustrate nur noch ein Bild pro Minute. Ein weiteres Problem ergab sich auf den Rechnern, die sich im gleichen Subnetz wie der Testaufbau befinden. Auch Computer, die nichts mit der Übertragung der Daten zu tun haben, zeigten eine signifikante Erhöhung der CPU-Belastung an. Das ist problematisch, da man nicht anderen Benutzern die CPU-Zeit wegnehmen sollte.

Da eine stabile Verbindung die Grundvoraussetzung für eine verlustlose Übertragung ist, wurden die Tests abgebrochen. Als nächste Möglichkeit des Transfers der Videodaten boten sich stattdessen verbindungsorientierte Socketverbindungen (TCP/IP) an.

5.4.4 Socketbasierter Transfertest

Für einen Transfertest mit verbindungsorientierten Socketverbindungen gelten die gleichen Anforderungen wie beim Transfertest über das TINE-Protokoll. Es müssen **mindestens** 4409200 Bytes pro Sekunde vom Server zum Client verschickt werden können. Durch die Anforderung, mehrere Clients gleichzeitig an einen Server anbinden zu können, entstehen noch höhere Anforderungen, da die Daten für jedes verbundene Clientprogramm einzeln

[3] Broadcast wurde gewählt, da Multicast von der aktuellen Version nicht zuverlässig unterstützt wurde

gesendet werden müssen. Eine weitere Anforderung, die verlustfreie Übertragung der Videobilder, lässt sich mit Streaming Sockets leicht erreichen, da der Empfang der Daten garantiert wird, solange die Netzwerkbandbreite ausreichend ist. Ausgelassene Bilder (Dropped Frames) sollten nicht vorkommen. Ein systembedingter Nachteil der verbindungsorientierten Sockets ist, dass die Daten für jeden Client einzeln gesendet werden müssen. Das schränkt die Menge der gleichzeitig verbundenen Clients ein. Nur durch die Kompression wird die durch das mehrfache Senden benötigte Netzwerkbandbreite abgemildert.

Der Testaufbau bestand aus einem Serverprogramm auf dem Grabbingrechner im Serverraum und einem Clientprogramm im Kontrollraum zum Empfang der übertragenen Daten. Vor der Durchführung des Tests wurde das 100 MBit Netzwerk mit dem Benchmarkprogramm NETIO [13] geprüft. Die dabei ermittelte Datenrate lag bei $\approx 11 \frac{MB}{s}$. Ziel des socketbasierten Transfertests ist es, diesen $11 \frac{MB}{s}$ sehr nahe zu kommen.

Den Transfertest wurde mit einer selbstgeschriebenen Socketklasse, welche auf das Winsock2-API vom Windows-Betriebssystem aufsetzt, durchgeführt. Auf der Netzwerkstrecke vom Serverraum zum Kontrollraum, welche später zur Übertragung der Bilder genutzt wird, wurden $\approx 10.6 \frac{MB}{s}$ als Transfergeschwindigkeit ermittelt. Der Test wurde mit anderen Netzwerkstrecken wiederholt. Es kam zu ähnlichen Ergebnissen. Niemals sank die Transferrate unter $10 \frac{MB}{s}$.

Das mehrfache Senden der Daten, wenn mehrere Clientprogramme mit gleichen Daten versorgt werden müssen, ist der große Nachteil der verbindungsorientierten Sockets. Da aber durch diese Art des Datentransfers immer noch genügend Clientprogramme an einen Server angebunden werden können und die Übertragung garantiert verlustfrei erfolgt, fiel die Entscheidung auf verbindungorientierte Sockets zur Übertragung der Videobilder.

5.4.5 Kompression

Für eine spätere wissenschaftliche Auswertung der Videobilder ist es unerlässlich, dass die Daten nicht verfälscht werden. Deshalb ist eine Forderung die verlustlose Kompression. Für die Durchführung der Kompression wurde in Abschnitt 4.1 der HuffYUV-CODEC ausgewählt. Nun soll in mehreren Kompressionsprototypen überprüft werden, inwiefern die von HuffYUV durchgeführte Kompression wirklich verlustlos ist.

Implementiert wurden ein Prototypprogramm, welches durch eine Zufallsfunktion Bilder erzeugt, diese komprimiert und dekomprimiert. Das dekomprimierten Bild wird Bit für Bit mit dem Ausgangsbild verglichen. Wenn ein Bitfehler auftritt, wird dieser vom Programm protokolliert. In einem 68 Stunden dauernden Testlauf wurden insgesamt 5.998.365 unterschiedliche Bilder überprüft, ohne dass ein Bitfehler auftrat.

Ein zweites Programm soll die Kompression im realen Umfeld prüfen. Der Grabbingprototyp aus Abschnitt 5.4.2 wurde als Basis genommen und um Kompression, Dekompression

und Validierung erweitert. Der Grabberserver wurde mit einer Kamera im Experimentiertunnel verbunden. Die Triggerrate war auf 5 Hz eingestellt, so dass von der Kamera Bilder im 5 Hz-Rhythmus geliefert wurden. Eventuell auftretende Kompressionsfehler werden in einer Datei protokolliert. In einem Testlauf über 64 Stunden wurden insgesamt 1.152.879 Bilder aufgenommen. Nach Abschluß des Testlaufes enthielt die Protokolldatei keinen Eintrag, d.h. auch hier war nicht ein einziger Bitfehler aufgetreten.

Es wurden insgesamt 7.151.244 einzelne Videobilder überprüft. Ein Videobild besteht aus 768 x 574 Pixeln. Ein einzelner Pixel ist 8 Bit groß. Das ergibt $N_t = 2.5 \cdot 10^{13}$ als Anzahl aller korrekten Bits. Durch Anwendung einer Formel aus der Wahrscheinlichkeitsrechnung ist die Wahrscheinlichkeit eines Bitfehlers bei N_t korrekten Bits:

$$P_f <= \frac{1}{{N_t}^2}$$

Die Wahrscheinlichkeit eines auftretenden Bitfehlers P_f beträgt weniger als $1.57 \cdot 10^{-25}$ Prozent. Es wird also mit an Sicherheit grenzender Wahrscheinlichkeit verlustlos komprimiert. Damit ist die Prototypenphase abgeschlossen, da alle Tests letztendlich erfolgreich verliefen.

5.5 Entwurf des Servers

Beim Entwurf des Servers gelten zwei Maximen: Stabilität (siehe Abschnitt 5.4.2) und Beschränkung auf die wesentlichen Funktionen. Es wird keine aufwändige Bedienoberfläche geben, sondern nur einen einfachen Dialog mit den notwendigen Schaltflächen und Anzeigen. Der Server wird intern in fünf Teile untergliedert:

1. Framegrabber-Ansteuerung

   Dieser Teil wird durch eine Klasse implementiert. Durch Austausch dieser einen Klasse ist es möglich, den Grabberserver auf eine andere Framegrabberkarte umzustellen, da alle Funktionalität zum Ansprechen der Framegrabberkarte über das ITEX Grabbing API in dieser Klasse gekapselt ist. Die nötigen Anpassungen bei Umstellung auf ein anderes API bzw. eine andere Framegrabberkarte müssen nur innerhalb der Klasse vorgenommen werden, der Rest des Quellcodes des Serverprogrammes bleibt unberührt.

2. Senden

   Der Teil des Servers, der sich um das Senden der Videodaten kümmert, wird später von außen gesehen nur wenige Berührungspunkte mit dem Rest des Programms haben. Dieser Teil soll nur initialisiert werden. Nach erfolgreicher Initialisierung übergibt man die einzelnen komprimierten Videobilder. Alle zum Senden nötige Funktionalität befindet sich ausschließlich in diesem Teil. Es ist keine weitere Interaktion mit der Ablaufsteuerung des Hauptprogramms nötig. Das erleichtert es, den Senden-Teil, der in der ersten Programmversion auf verbindungsorientierten Socketverbindungen basiert, durch eine Multicastengine zu ersetzen oder zu erweitern.

3. Kompression

   Der Kompressionsteil wird ebenfalls durch eine Klasse realisiert. Im Idealfall kann man durch Austausch dieser Klasse die Kompression abändern oder durch Erweiterung neue Kompressionsverfahren integrieren. Andere Teile des Programms bleiben erneut unberührt.

4. Kontrollteil

   Der Kontrollteil wird zur Steuerung des Servers benötigt. Als Netzwerkschicht findet das TINE-Protokoll Anwendung. Der Kontrollteil ist eng mit der internen Ablaufsteuerung verknüpft. Hauptsächlich dient der Kontrollteil zur Auswahl des Kameraports sowie zur Bereitstellung der sprechenden Kameranamen für die Client-Anwendung.

5. Programmablauf und Bedienoberfläche

   Beim letzten Teil handelt es sich um die interne Ablaufsteuerung der Applikation sowie um die Steuerungsfunktionen für die Bedienoberfläche. In der MFC-Programmierung in Visual C++ werden Fenster auf Klassen umgemappt. Durch virtuelle Funktionen, die man überschreiben kann, ist es möglich, auf Ereignisse innerhalb des Fensters zu reagieren. Da in dieser Wrapper-Klasse die Verbindung zu dem Benutzerdialog hergestellt wird, liegt es nahe, hier auch die interne Ablaufsteuerung der Applikation vorzunehmen.

Die Bedienoberfläche wurde einfach gehalten. Sie soll möglichst intuitiv benutzbar sein. Die Bedienoberfläche enthält wenige Schaltflächen, die zum Aufruf des Konfigurationsdialoges, zum Starten und Stoppen des Grabbings und zum Verlassen der Applikation dienen. Als Anzeige ist ein Hinweistextfeld vorgesehen, in dem Warnungen und Meldungen der Applikation chronologisch aufgezeichnet werden. Weiterhin gibt es ein Bedienelement zur Auswahl des Kameraports. Dort wird der aktuell selektierte Kameraport mit einem sprechenden Namen gleichzeitig angezeigt. Es sind zusätzlich zwei Anzeigen vorgesehen, eine für die durchschnittliche Bildrate (Framerate) des Grabbers und eine für den sogenannten Heartbeat. Das wird bei jedem aufgenommenen Bild durch das Inkrementieren einer Zahl realisiert. Damit die Zahl nicht zu groß gerät, wird beim Erreichen von fünf wieder auf null zurückgeschaltet. Die Zahlenreihe wird folgendermaßen aussehen: 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 usw.

Alle Einstellungen werden in einer Konfigurationsdatei gespeichert. Damit soll die Handhabung vereinfacht werden, da beim Starten des Servers die Auswertung der Datei erfolgt und alle konfigurierbaren Einstellungen vorgenommen werden. Weiterhin lässt sich durch eine Konfigurationsdatei ein einfaches Umkopieren des Serverprogrammes auf eine andere Festplatte oder einen anderen Rechner, unter Beibehaltung aller Einstellungen, realisieren.

Da auf dem Server blockierende (synchrone) Sockets benutzt werden sollen, muss der Transfer der Videodaten in separaten Threads stattfinden. Der Ausfall eines Client kann bei dem verwendeten Protokoll die Übertragung zu allen anderen blockieren. Daraus folgt,

dass jeder Client durch einen eigenen Thread abgearbeitet werden muss. Sollte dann ein Client blockieren, bekommen die anderen trotzdem ihre Bilddaten zugesandt. Die Kopplung zwischen dem Hauptthread und den Sendethreads wird über eine Pointerwarteschlange realisiert. Im Hauptthread wird das einzelne komprimierte Bild nur in die entsprechende, zum Client gehörende Pointerwarteschlange eingereiht. Der Sende-Thread holt sich dann die Daten selbständig aus der Warteschlange.

5.6 Entwurf des Clients

Der Entwurf des Clients gestaltet sich komplexer als der Entwurf des Servers, da der Client wesentlich mehr Funktionalität beinhaltet als das Serverprogramm. Mit dem Analyseprogramm ,,Video Client" soll es möglich sein, sowohl Bilder in Echtzeit zu analysieren, als auch eine Bildfolge aufzunehmen und dann offline zu analysieren. Der Client soll folgende Funktionen realisieren:

- Hintergrundsubtraktion nach verschiedenen Algorithmen
- Normalisierung des Videosignals
- Falschfarbenmodus
- Röntgenstrahlenfilterung
- Verlustlose und schnelle Dekompression
- Strahlposition und Strahlgrößenberechnung nach verschiedenen Algorithmen
- Projektionen des Videosignals
- Selektierbares Auswahlfenster (Area of Interest)
- Skalierung der Berechnungen
- Einstellungen in Konfigurationsdatei speichern
- intuitiver Konfigurationsdialog
- Laden und Speichern von Bildern und Hintergrundinformationen
- Kontrollzugang zum Server
- Schnappschussmodus
- Echtzeitverarbeitung bei einer Bildrate von bis zu 10 Hz

Die notwendige Funktionalität ist Grundlage für die Planung und Bereitstellung der erforderlichen Hardware-Ressourcen. Als Zielsystem wird ein PC mit mindestens Pentium III 866 MHz-Prozessor vorausgesetzt. Auch bei einem derartigen PC ist es bei der späteren Implementierung nötig, neben der Optimierung der Algorithmen auf Assemblersprache und spezielle Befehlssatzerweiterungen (MMX) bei kritischen Sequenzen zurückzugreifen.

Der Code des Clients soll sich in zwei Teile gliedern. Der erste Teil besteht in einer Hauptklasse, in der die interne Ablaufsteuerung und die Anbindung der Bedienoberfläche stattfindet, sowie Hilfsklassen, die spezielle Funktionalität in gekapselter Form zur Verfügung stellen. Hilfsklassen sind z.B. die Socketimplementation, die Anbindung des Kontrollsystems via TINE-Protokoll, Speicherverwaltung, Dekompression, Zusatzdialoge (Konfiguration, Meldungen), Analyse- und Berechnungsklassen und Warteschlange zur Thread-Kopplung.

Im Prinzip besteht der Client aus einer Klasse für die interne Ablaufsteuerung und vielen Komponenten, die richtig zusammengesetzt das komplette Programm ergeben. Die Bedienoberfläche soll intuitiv sein und Tastaturkürzel für häufig benutzte Funktionen bieten. Eine Unterteilung in Menüzeile, einer Leiste zur Anzeige von Werten und zur Steuerung von Teilfunktionen, dem eigentlichen Videofenster und einer Statuszeile erscheint ebenfalls sinnvoll.

Als Basis für die Analysefunktionen im Client gelten die in Abschnitt 3 erläuterten Grundlagen. Klassen dienen im Rahmen dieses Programms zur Abstraktion und Kapselung. Im Idealfall sollte man durch Austauschen einer Klasse bzw. Anpassung einer speziellen Klasse zusätzliche Funktionalität einbinden können. Manche Klassen waren schon vor dem Beginn der Implementation fertiggestellt. Dabei handelt es sich um die Klassen für Strahlmittelpunkt und Strahlgrößenberechnung. Diese werden im Rahmen der Implementierung nur eingebunden und gegebenenfalls angepasst.

Vom Server werden die Bilder zum Clientprogramm über eine synchrone Socketverbindung geliefert. Threads für den Empfang der Videodaten über die Socketverbindung werden nötig, da ein synchroner Socket die Ausführung des Programmcodes blockiert, wenn er auf Daten aus dem Netzwerk wartet. Damit eine Blockierung des Analyseprogramms nicht möglich ist, wird der Empfang der Videobilder in einen separaten Thread ausgelagert. Die Kameraauswahl wird über das TINE-Protokoll abgewickelt werden. Als Schnittstelle zum TINE-Netzwerk bot sich das TINE ActiveX-Control ACOP an. Es lässt sich leicht in Visual C++ integrieren und bietet eine definierte und einfach zu handhabende Schnittstelle zum TINE-Protokoll. Im Idealfall sollte das Control durch eine neue Version ersetzt werden können, ohne dass Änderungen im Code nötig werden.

6 Implementierung

Die Implementierung der beiden Programme, GrabServer und Video Client, erstreckte sich über drei Monate. Hier wird auf die Lösung der wichtigsten Probleme eingegangen. Wer tiefer in die Materie eindringen möchte, dem sei zusätzlich zu den Erläuterungen in diesem Abschnitt der ausführlichst dokumentierte Quellcode nahegelegt. Dieser befindet sich auf der beigelegten CD. Besonderes Augenmerk innerhalb der Phase der Implementierung ist auf die Optimierung des Programms gerichtet. Auf die Methoden und Erfolge der Optimierung wird in einem separaten Abschnitt eingegangen.

6.1 Server

In den folgenden Abschnitten wird auf die Implementierung des Servers eingegangen. Alle Angaben können durch das Klassendiagramm in Abschnitt 6.1.6 nachvollzogen werden.

6.1.1 Übernahme der Prototypenfunktionalität

Beim Server galt es zuerst, die entwickelte Funktionalität aus der Prototypenphase in Klassen zu kapseln, um dann das darauf basierende Programm zu schreiben. Folgende Funktionalität wurde aus der Prototypenphase übernommen:

- Grabberklasse
- Klasse für verlustlose Kompression
- Socketklasse

Zuerst musste die zum Aufnehmen der Videobilder nötige Funktionalität aus dem Grabberprototypen extrahiert und in eine einzelne Klasse überführt werden. Diese Klasse soll dem Anspruch auf Abstraktion und Kapselung genügen. Im Idealfall soll durch Austausch dieser einen Klasse das Serverprogramm auf eine andere Framegrabberkarte umstellbar sein. Die Klasse (CGrab) folgt einem Lebenszyklus. Es gibt Funktionen wie Init(), Done(), StartGrab(), StopGrab() und Funktionen zum Umschalten des Kameraports. Innerhalb der Klasse wird beim Starten des Grabbingvorganges ein Thread gestartet, welcher die Bilder mit Hilfe des API der Framegrabberkarte entgegennimmt. Nach jedem empfangenen Bild wird eine Windows-Botschaft abgesendet. Dieser Mechanismus hat den Vorteil, dass die Verarbeitung an anderer Stelle des Programms fortgesetzt wird. In diesem Fall erfolgt die weitere Bearbeitung durch das Hauptprogramm. Durch die Festlegung auf einen Lebenszyklus kann diese Klasse leicht an andere Umgebungen (API, Framegrabberkarte) angepasst werden.

Nach der Grabberklasse wurde die benötigte Funktionalität des Video Compression Manager (VCM) in eine Klasse übertragen. Diese Klasse (CCompression) soll auch einem Lebenszyklus entsprechen. Es gibt Funktionen wie Init(), Done() und eine Funktion zur Kompression eines Einzelbildes. Die Klasse wurde so implementiert, dass man den zugrundeliegenden Kompressions-CODEC einfach austauschen kann. Dazu muss die Init()-Methode mit einem Code für einen anderen Kompressions-CODEC aufgerufen werden. Durch die Kapselung ist es weiterhin möglich, durch Austausch dieser einen Klasse eine andere Kompressionsarchitektur einzubinden. Der Rest des Serverprogramms wird davon unberührt bleiben. Die Klasse wurde im Rahmen der Implementierung mehrfach getestet und funktionierte fehlerfrei.

Nach dem Erzeugen der Kompressionsklasse musste die aus dem Sockettransfertest entstandene Funktionalität in eine synchrone Socketklasse (SSyncSocket) gekapselt werden. Diese Klasse besteht aus low-level Funktionen und bildet die Übertragungsschicht ab. Die Ablauf-

steuerung des Transfers (Anwendungsschicht) findet in dieser Klasse nicht statt. Die Socketroutinen basieren auf dem Winsock2-API von Windows. Die Klasse soll möglichst einfach einzubinden sein. Es gibt Funktionen zum Erzeugen des Socket, zum Verbinden mit einem anderen Computer, zum Starten des Horchmodus, zum Senden und Empfangen und zum Schließen des Sockets. Weiterhin wurden spezielle Routinen zum Auslesen der IP-Adresse des entfernten Computers und zur Fehlerbehandlung integriert. Die komplette Klasse ist blockierend, d.h. jeder Aufruf, der nicht unmittelbar beendet werden kann, hält die Programmausführung solange an, bis die Funktion abgearbeitet ist. Deshalb werden die Socketroutinen meistens nur von Threads aus aufgerufen.

6.1.2 TINE-Protokoll

Nachdem die aus der Prototypenphase entstandenen Klassen implementiert und getestet waren, galt es, das TINE-Protokoll in das Serverprogramm einzubinden. Die Entscheidung fiel auf die ActiveX-Komponenten zum Ansprechen des TINE-Protokolls, da das die einfachste Möglichkeit war, das TINE-Protokoll in das Serverprogramm einzubinden. Zuvor wurde der Rahmen (Framework) für das Programm durch den MFC-Anwendungsassistenten in Visual C++ erzeugt und alle schon fertiggestellten Klassen hinzugefügt. Das TINE-ActiveX-Control für die Serverseite „Srv" wurde auf dem Hauptdialog der Applikation verankert. Visual C++ erzeugt daraufhin eine Wrapperklasse, um mit definierten C++-Methoden die Funktionalität des Controls ansprechen zu können. Innerhalb der Dialog-, Ablaufsteuerungs- und Verbindungsklasse (CMainDlg) wurde durch den Klassenassistenten eine Klassenvariable für die ActiveX-Control-Klasse erzeugt. Diese ist nötig, um das ActiveX-Control ansprechen zu können. Weiterhin wurde dialoggesteuert eine Serverfunktion, welche vom Control aufgerufen wird, eingebunden. Die Serverfunktion OnEqpFcnSrvctrl1() wird immer aufgerufen, wenn ein Client eine Anfrage über das Protokoll an den Server sendet. Innerhalb dieser Funktion wurde die nötige Funktionalität zum Übertragen der sprechenden Kameranamen und zum Umschalten des Kameraports implementiert. Zusätzlich wurden in die Hauptklasse CMainDlg noch zwei Funktionen eingefügt, die sich um das Initialisieren und Deinitialisieren des TINE-ActiveX-Controls kümmern. In der Testphase wurde gemeinsam mit dem TINE-Entwickler der geschriebene Code evaluiert. Fehler in der Implementation des TINE-Protokolls und der Ansteuerung des ActiveX-Controls wurden erkannt und beseitigt.

6.1.3 Bildübertragungsfunktionen

Die Basis der Bildübertragungsfunktionen ist die aus der Prototypenphase entstandene Klasse SSyncSocket. Diese kapselt aber nur die Low-Level-Socketfunktionen. Es soll ein noch höherer Kapselungsgrad erreicht werden. Vom Hauptprogramm aus soll es möglich sein, nach dem Initialisieren des Socketsubsystems die einzelnen komprimierten Bilder an dieses Subsys-

tem zu übergeben. Die Verteilung und das Senden an alle verbundenen Clientanwendungen soll dann intern durchgeführt werden. Dadurch ist es möglich, die jetzigen Socketroutinen später einmal durch eine andere Netzwerkschnittstelle oder ein anderes Übertragungsprotokoll zu ersetzen. Das könnte zum Beispiel eine Multicastengine sein, die im Gegensatz zum jetzt implementierten System der verbindungsorientierten Sockets die Daten nur einmal ins Netzwerk sendet. Die Verteilung an alle verbundenen Clientanwendungen wird dann von der Netzwerkhardware durchgeführt. Vor der Implementation der eigentlichen Klassen galt es festzulegen, wie die Übertragung der Videobilder abgewickelt werden soll. Auszugsweise wird hier auf die wichtigsten Details eingegangen. Eine genauere Übersicht der Bildübertragungsschnittstelle findet sich in der Programmierdokumentation zum Grabberserver [14].

Nachdem sich eine Clientanwendung zum Server verbunden hat, werden die einzelnen Videobilder sequentiell gesendet. Da in den Videobilddaten wichtige Zusatzinformationen (Breite und Höhe des Videobildes, Kompressionsflag, Kompressionsart, Zeitstempel, Bildnummer etc.) fehlen, wird jedem Bild ein Header vorangestellt, in dem diese Informationen enthalten sind. Dieser Ablauf, dargestellt in der Abb. 17, wiederholt sich solange, bis der die Verbindung abbricht oder vom Client durch Schließen des Socket normal beendet wird.

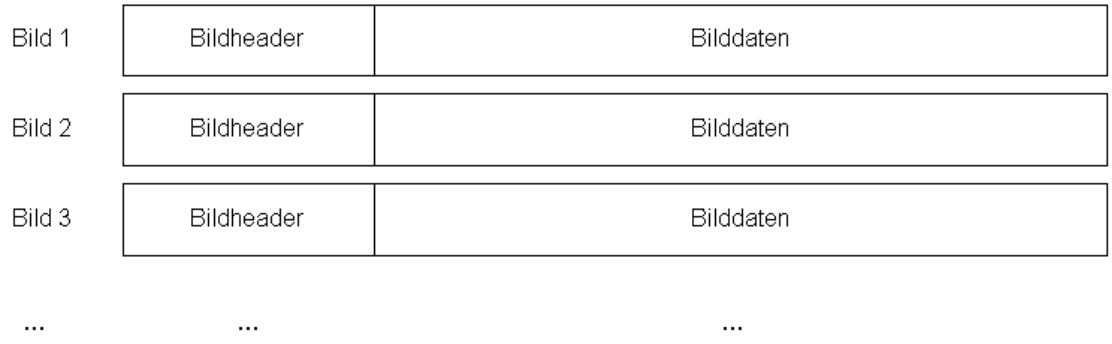

Abb. 17: Datenversand über die Socketschnittstelle

Nach der Festlegung des Protokolls der Anwendungsschicht wurden weitere Klassen für die Bildübertragungsfunktionen entworfen. Dabei handelt es sich zuerst um eine Klasse (CSocketEncapsulation), die an das Hauptprogramm angekoppelt wird und die Bilder entgegennimmt. Intern gibt es dann für jeden verbundenen Client eine weitere Klasse (CSendReceive), die sich um das Senden der Bilder an einen einzelnen Client kümmert. Die beiden Klassen haben eine klare Aufgabentrennung: Die Klasse CSocketEncapsulation kümmert sich um die Verwaltung der einzelnen Clients sowie um das Zustandekommen einer Verbindung, während sich die Klasse CSendReceive um das Senden der Bilder kümmert. Innerhalb der Klasse CSendReceive gibt es zwei Threads, einen zum Senden und einen zum Empfangen von Daten. Die einzelnen Bilder werden durch eine Zeigerwarteschlange von der Klasse CSocketEncapsulation an die einzelnen CSendReceive-Klassen weitergereicht. Diese Arbeitsteilung hat einen

wichtigen Vorteil. Es kann passieren, dass ein Client die Daten nicht in Echtzeit empfangen kann. Da die Socketroutinen blockieren, käme es zu einem Datenstau auf dem Server. Da aber jeder Client durch eine eigene Instanz der Sende/Empfangsklasse CSendReceive abgearbeitet wird, blockiert nur der Sendethread in dieser einen Klasse. Alle anderen Clients erhalten trotzdem ihre Bilder in Echtzeit. Es gibt auf dem Server Möglichkeiten, Bilder ausfallen zu lassen (Dropped Frames), um eine störungsfreie Übertragung zu gewährleisten. Das kann der Fall sein bei einer Überlastung des Servers (zu hohe CPU-Belastung, Netzwerkbandbreite reicht nicht aus) oder dem Blockieren eines einzelnen Clients.

6.1.4 Einstellungen und Konfigurationsdatei

Beim Serverbetrieb gibt es bestimmte Einstellungen, die sich nicht von vornherein festlegen lassen, sondern je nach Bedarf abgeändert werden müssen. Das sind:

1. sprechende Kameraportnamen (Bezeichnungen der Diagnosestationen)
2. Adresse der Framegrabberkarte
3. Netzwerkparameter

Zur Bearbeitung der Einstellungen wurde ein Konfigurationsdialog im Serverprojekt angelegt. Dieser erlaubt die Abänderung der Einstellungen. Es wurde eine Dialogklasse (CConfigDlg) erzeugt und mit Eventfunktionen ausgestattet. Diese werden zur Reaktion auf bestimmte Ereignisse (z.B. Schaltfläche OK gedrückt) benötigt.

Die im Konfigurationsdialog gemachten Änderungen werden in einer Konfigurationdatei abgespeichert. Die Konfigurationsdatei hat den Vorteil gegenüber einer Speicherung in der Windows-Registry, dass diese Datei mit dem Programm mitgegeben werden kann und nicht wie die Registry fest an eine Windows-Installation gebunden ist. Weiterhin ist es möglich, vor dem ersten Start des Programmes eine Vorkonfiguration ohne Nutzerinteraktion durchzuführen. Das lässt sich mit der Registry zwar auch realisieren, doch muss dazu ein Installationsprogramm geschrieben werden, während bei einer Konfigurationsdatei die Installation ein einfacher Kopiervorgang ist. Zur Abstraktion der Konfigurationsdatei wurde die Klasse CConfiguration implementiert. Bei Aufruf der Init()-Methode wird die Konfigurationsdatei eingelesen. Sollte die Datei nicht auffindbar sein, wird sie erstellt und mit Voreinstellungen versehen. Mit get/set-Methoden werden die einzelnen Parameter eingestellt. Die Funktion SaveToFile() speichert die Einstellungen. Bei Aufruf des Destruktors geschieht das automatisch.

6.1.5 Ablaufsteuerung

Nachdem sämtliche Teilfunktionalitäten vorlagen wurde die Verknüpfung der einzelnen Teile implementiert. Dazu gibt es in der Verdrahtungsklasse CMainDlg Funktionen zum Initialisieren und Deinitialisieren der Subsysteme (Grabbing, Kompression, TINE und Netzwerk). Diese Funktionen werden beim Starten bzw. Verlassen der Applikation aufgerufen. Weiterhin wurden die Schaltflächen und Anzeigen auf dem Dialogfenster mit den einzelnen Funktionen in den Teilsystemen verknüpft. Die von der Grabberklasse gesendete Windows-Botschaft wird durch Aufruf der Methode OnNotifyGrabbed() empfangen. In dieser Funktion wird das Videobild von der Grabklasse abgeholt, komprimiert und in die Socketkapselklasse CSocket-Encapsulation hineingepackt. Dabei werden die Anzeigen auf dem Hauptdialog (Framerate und Heartbeat) aktualisiert. Zum Aufruf des Konfigurationsdialogs wurde eine Schaltfläche auf dem Dialog hinzugefügt und mit einer Event-Funktion ausgestattet. In dieser Funktion werden die Einstellungen aus der Konfigurationsdatei an den Dialog übergeben. Danach wird der Dialog aufgerufen. Nach dem Druck auf ,,OK" werden die neuen Einstellungen in die Konfigurationsklasse übertragen und permanent abgespeichert. Danach erfolgt ein Reinitialisieren der Subsysteme, da dies durch geänderte Einstellungen nötig ist.

Nach der Fertigstellung und ersten Funktionstest wurde das Programm in die Betatestphase überführt. Kleinere logische Fehler im Programmablauf sowie Fehler im Zusammenspiel der einzelnen Komponenten wurden erkannt und beseitigt.

6.1.6 Klassendiagramm

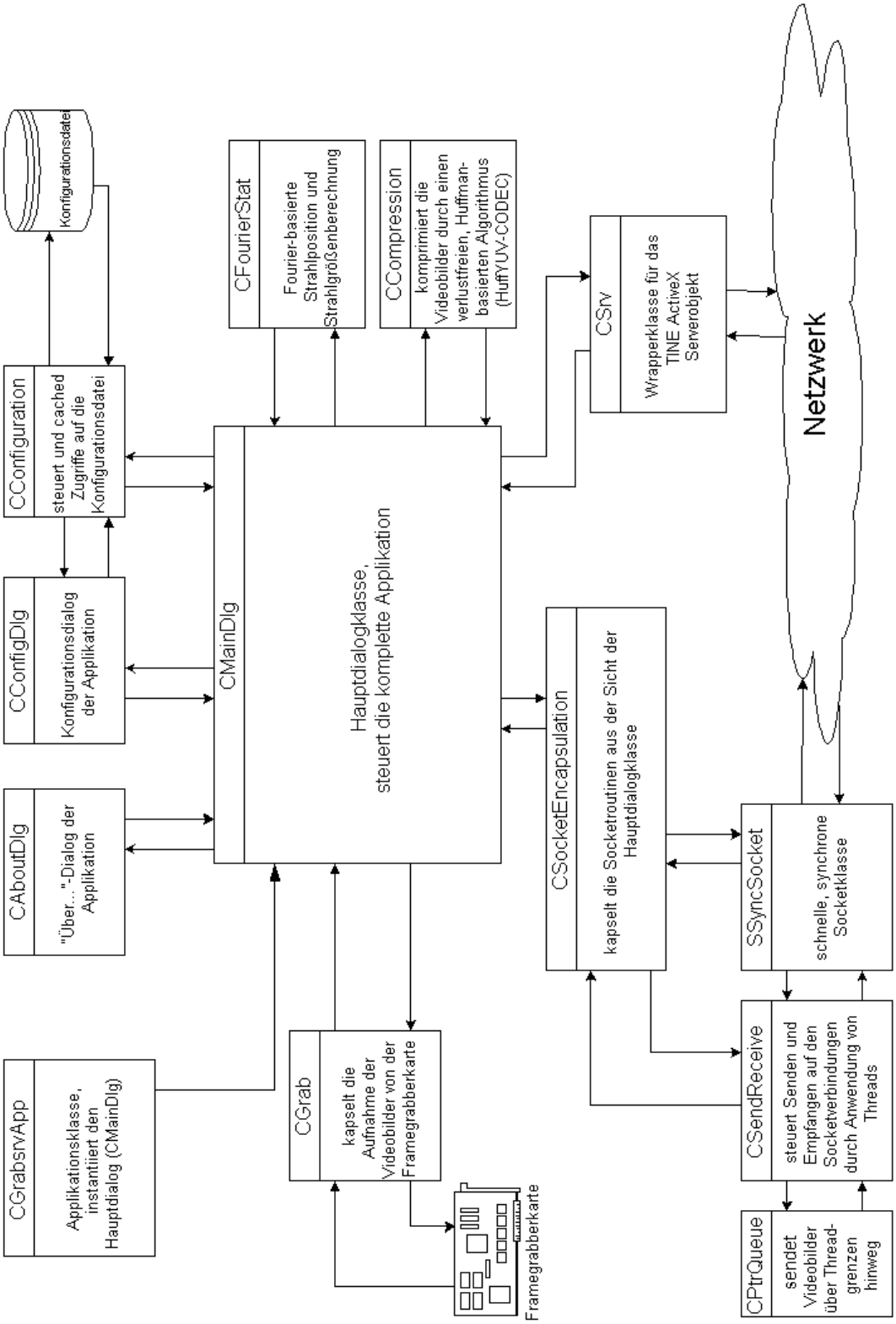

Abb. 18: Klassendiagramm GrabServer

6.2 Client

Die Implementation des Client war aufwändiger als die Implementation des Servers. Nach Fertigstellung der Grundfunktionalität wurden die Algorithmen mit viel Zeitaufwand optimiert. Darauf wird in einem späteren Abschnitt gesondert eingegangen. Zuerst galt es jedoch, Klassen aus dem Serverprogramm im Client wiederzuverwenden, die für die Grundfunktionalität benötigten Klassen zu implementieren und die vorhandenen Analyseklassen in das Clientprogramm zu integrieren. Das Klassendiagramm vom Client ist in Abschnitt 6.2.9 aufgeführt.

6.2.1 Wiederverwendung

Zwei Klassen, ursprünglich für das Serverprogramm geschrieben, konnten im Client wiederverwendet werden. Da die Videodaten über eine Socketschnittstelle acquiriert werden, kann die Socketklasse SSyncSocket erneut verwendet werden. Alle nötige Funktionalität für den Clientsocketbetrieb sind in der Klasse mit enthalten. Da es sich bei der Socketklasse um eine blockierende Klasse handelt, werden die Funktionen der Klasse hauptsächlich von Threads aus aufgerufen. Deshalb wird die Zeigerwarteschlange CPtrQueue auch im Client Verwendung finden.

6.2.2 Dekompression

Für die Dekompression war es notwendig, eine eigene Dekompressionsklasse zu implementieren. Da der Dekompressionsalgorithmus später auch auf andere Plattformen portiert werden soll, wurde der Video-CODEC HuffYUV nicht über den Video Compression Manager (VCM) eingebunden. Stattdessen wurden die wichtigen Quellcodeteile aus dem CODEC in eine ANSI-C++-Klasse übernommen. Um dem Anspruch an Kapselung gerecht zu werden, folgt die Dekompressionsklasse CCompression einem Lebenszyklus. Bevor man die Bilder dekomprimiert, muss die Initialisierungsfunktion Init() erfolgreich aufgerufen worden sein. Die Funktion Init() verlangt einen Vierzeichencode (four character code) zur Identifikation des Dekompressionsalgorithmus. Dadurch ist es später auch möglich, die Klasse mit mehr als einem Algorithmus auszustatten. Nach der erfolgreichen Initialisierung kann man über die Funktion DecompressFrame die Dekompression eines einzelnen Bildes veranlassen. Zur sauberen Beendigung des Programmes oder zur Auswahl eines anderen Dekompressionsalgorithmus muss man über die Funktion Done() die gesamte Klasse deinitialisieren. Die Dekompressionsklasse wurde mit dem Testprogramm zur verlustfreien Kompression evaluiert. Der Test verlief erfolgreich, es traten keine Bitfehler auf.

6.2.3 Implementation der Grundfunktionalität

Nachdem Zeigerwarteschlange, Dekompressionsklasse und Socketklasse bereitstanden, begann die Implementation der Grundfunktionalität. Stufenweise wurde dann das Programm immer weiter ausgebaut, bis die Applikation fertiggestellt war. Zuerst wurde der minimale Rahmen (Framework) für die Applikation durch den in Visual C++ enthaltenen MFC Anwendungsassistenten generiert. Dabei entstanden die Klassen CVideoclntApp und CVideoclntDlg. Die App-Klasse dient nur zur Instantiierung der Hauptdialogklasse CVideoclntDlg. Zuerst wurde die Menüzeile der Applikation erzeugt. In ihr sind schon alle Menüeinträge enthalten. Danach wurde die Rahmenfunktionalität implementiert. Dabei handelt es sich um Logik zum Schalten der Applikation in den Vordergrund (Stay on top) und um die Zeichnung von Häkchen vor aktivierten Menüeinträgen. Weiterhin wurde ein Meldungen-Dialog und die dazugehörige Klasse CMessagesDlg erzeugt. In dieser Klasse werden die Meldungen in der Statuszeile mit Datum und Uhrzeit mitgeloggt. Diese Funktionalität soll die Fehlersuche unterstützen.

Danach ging es um die Funktionalität des Empfangens der Bilder über das Netzwerk und der Anzeige im Videofenster. Dazu war Funktionalität nötig, die bei Größenänderung des Applikationsfensters den Videorahmen automatisch der Größe entsprechend anpasst. Hierzu wurde die Funktion OnSize() der Basisklasse von CVideoclntDlg überschrieben. Dann wurden die Funktionen zum Starten und Stoppen des Poll-Modus implementiert. In der Startfunktion ‚StartPollMode()' geht es um die Verbindungsherstellung zum Server und dem Starten der Empfangsthreads. Dort werden die Bilder vom Server über den Socket empfangen und in die Zeigerwarteschlange gesteckt. Dann wird eine Windows-Botschaft gesendet. An anderer Stelle im Programm wird die Botschaft durch die Funktion OnFrameReceived() empfangen. Dort wird das Bild aus der Zeigerwarteschlange abgeholt, dekomprimiert und über eine Anzeigefunktion (DisplayFunction) auf dem Videorahmen der Bedienoberfläche zur Anzeige gebracht.

Nachdem dieser Teil implementiert war, wurde der Falschfarbenmodus programmiert. Da bei der Anzeige eines Bildes eine Farbpalette mitgegeben werden muss, war es ein leichtes, neben der schon benutzten Graustufenpalette eine weitere Palette in das Programm einzupflegen. Die Umschaltung der Palette wurde in der Funktion UpdatePalette() implementiert. Durch die Architektur des Programms ist es jederzeit möglich, die Palette zu ändern, egal welche Aktion das Programm gerade durchführt. Im Gegensatz dazu werden bei aktiviertem Poll-Modus einige Funktionalitäten des Programms deaktiviert. Im Menü erkennt man das an gesperrten (ausgegrauten) Einträgen. Zum Beispiel kann man bei aktiviertem Poll-Modus keine Bilder laden oder speichern. Auch die Aufnahme eines Hintergrundes ist bei aktiviertem Poll-Modus nicht möglich.

Nach der Implementation der Farbpalette wurde der Bildsequenz-Modus (engl. Grab-

Mode) implementiert. Im Gegensatz zum Poll-Modus, bei dem man Live das Video betrachten kann, nimmt man im Bildsequenz-Modus ein Einzelbild bzw. eine Einzelbildfolge auf, die man danach offline analysieren kann. Es soll auch möglich sein, die Bildsequenz mehrmals hintereinander zu analysieren.

Für den Bildsequenz-Modus gibt es im Image-Menü den Menüpunkt Grab mit vielen Untermenüeinträgen, die die Anzahl der aufzunehmenden Bilder angeben. Für die Aufnahme der Bilder wurde die Funktion GrabFramesFromServer() implementiert. Nach einer erfolgreichen Verbindung zum Server wird die angegebene Anzahl von Bildern vom Server empfangen und die Verbindung beendet. Die Funktionalität zur Analyse der Einzelbildfolge ist Teil des nächsten Schrittes, der Einbindung der vorbereiteten Analyseklassen.

6.2.4 Bildverarbeitungs- und Analyseklassen

Für die Analyse der Bilder standen fertige Klassen bereit, die Bestandteil der bisherigen Lösung waren. Die Klassen CStat und CFourierStat beinhalten Funktionalität zur Strahlmittelpunkts- und Strahlgrößenberechnung, zur Hintergrundsubtraktion, zur Normalisierung und zur Röntgenstrahlenfilterung der Videobilder. Die zwei Klassen implementieren unterschiedliche Konzepte zur Strahlmittelpunktsberechnung. Zur Auswertung der Einzelbildfolgen stand eine Rahmenklasse (CStatFile) um die beiden Analyseklassen zur Verfügung. Die Klassen wurden so eingebunden, dass nach Empfang eines Bildes im Poll-Modus bzw. nach Empfang der Bilder im Bildsequenz-Modus die Analyse mit den gerade eingestellten Parametern durchgeführt wird. Der Rahmenklasse CStatFile teilt man die gerade aktivierten Analysemethoden, Analyseparameter und die Anzahl der Bilder mit. Intern wird dann die entsprechende Analyseklasse ausgewählt und die Berechnung durchgeführt. Bei einer Einzelbildfolge werden durch die Klasse CStatFile auch die Ergebnisse (Strahlmittelpunkt und Strahlgröße) über alle Bilder arithmetisch gemittelt. Außerdem wird aus allen empfangenen Bildern ein Mittelwertbild berechnet[4], welches im Videorahmen angezeigt wird. Die Anzeigefunktion (DisplayFunction) wurde um die Anzeige des Strahlmittelpunktes und der Strahlgröße (rotes Kreuz) erweitert. Weiterhin wurden die berechneten Parameter auch in die Anzeigefelder auf der Toolbar eingetragen.

Nach der Fertigstellung der Klassen kam ein weiteres Problem auf. Die Analyse und Bearbeitung der kompletten Videobilder setzt hohe Anforderungen für die PCs. Auf dem Zielsystem war eine Analyse bei 10 Hz im Poll-Modus nicht möglich, da die CPU-Belastung zu hoch ausfiel. Durch eine von der statistischen Strahlanalyse abgeleitete Klasse (CStatEmpty), welche keine Berechnung und Normalisierung durchführt, ist eine Betrachtung der Videobilder nun auch bei 10 Hz möglich.

[4]Nach dem gleichen Algorithmus wie bei der Average-Methode der Hintergrundsubtraktion, siehe 4.2.1

6.2.5 Anbindung des TINE-Protokolls

Nach der Implementation der Analyse wurde die Kontrollverbindung zum Server hergestellt. Diese benutzt das TINE-ACOP ActiveX-Control. Dieses Control wurde auf dem Hauptdialog der Applikation als verstecktes Element eingebunden. Danach wurde automatisch eine Wrapperklasse (CACOP) für diese Komponente erzeugt. Der nächste Schritt war die Erzeugung einer Variablen für das ActiveX-Control in der Hauptklasse CVideoclntDlg. Danach wurde die nötige Funktionalität durch Methodenaufrufe im ActiveX-Control implementiert. Die zu programmierende Funktionalität umfasst das Herunterladen der Kameranamen beim Start der Applikation und die Umschaltung des Kameraports auf dem verbundenen Server. Dazu wurden zwei Funktionen geschrieben: GetCameraNames() und OnSelchangeCombo1Camera(), die mit der Selektierung eines Kameraports über die Combo-Box in der Toolbar verbunden ist und bei jeder neuen Auswahl eine Verbindung zum Server aufbaut und den Kameraport umschaltet. Dabei ergab sich ein Problem: Es können sich mehrere Clients zum gleichen Server verbinden, aber es kann zu einem bestimmten Zeitpunkt nur eine Kamera aktiv sein. Was passiert, wenn ein Client Bilder pollt und der andere den Kameraport umschaltet? Aus Gründen der Einfachheit stehen die Clients in Konkurrenz. Aber es musste eine Möglichkeit geschaffen werden, damit der andere Client wenigstens eine Nachricht erhält, wenn der Kameraport von einem Client umgeschaltet wird. In der Klasse für die interne Ablaufsteuerung wurde eine Methode (OnTimer) der zugrundeliegenden Basisklasse überschrieben. Durch Initialisierungsroutinen wird diese Methode jede Sekunde einmal aufgerufen. Innerhalb der Methode wird über das TINE-Protokoll eine Verbindung zum ausgewählten Server aufgebaut und der aktuell selektierte Kameraport ausgelesen. Stimmt die Antwort nicht (mehr) mit dem im Client eingestellten Port überein, gibt es eine Warnmeldung. Nach der Implementierung dieser Methode war die Einbindung des TINE-Protokolles abgeschlossen.

6.2.6 Erweiterte Funktionalität

Nach der Basisfunktionalität gab es noch viele aus der Entwurfsphase hervorgegangenen Anforderungen, die noch nicht implementiert waren. Das sind:

- Tastatursteuerung
- Selektierbares Auswahlfenster (Area of Interest)
- Projektionszeichnung
- Laden und Speichern von Bildern und Hintergründen
- Schnappschussmodus

Tastaturhandling

Durch Einsatz von Tastaturkürzeln für wichtige, häufig benutzte Funktionen kann die Bedienung des Programms entscheidend vereinfacht werden. Besonders für erfahrene Nutzer ist diese Funktionalität wichtig, um die Einstellungen ohne Maus vornehmen zu können. In der Visual C++ - Entwicklungsumgebung gibt es ein Hilfsmittel (Accelerator) zur Erzeugung der Tastaturkürzeltabellen und der Methoden, die dann aufgerufen werden sollen. Dort gibt man die Tastenkombination ein und selektiert die Menüpunkt-ID der gewünschten Funktion, welche dann abgearbeitet werden soll. Um die Acceleratortabelle zu aktivieren, muss sie beim Programmstart geladen werden. Das erfolgt in der Klasse zur Instanziierung des Hauptdialoges (CVideoclntApp). Folgender Abschnitt muss in die Methode InitInstance() eingefügt werden:

```
BOOL CVideoclntApp::InitInstance() {
    ...

    // load keyboard shortcuts
    ghAccelTable = LoadAccelerators(AfxGetInstanceHandle(),
        MAKEINTRESOURCE(IDR_ACCELERATOR1));

    // instantiate, initialize and show main dialog
    ...
}
////////////////////////////////////////////////////////////
//
// This function is needed to process the keyboard shortcuts.
// It just uses windows functions to process the keystroke
// (if there is a keystroke) and posts a message with the
// right ID (e.g. of the Menuitem) to the main message
// queue.
//
BOOL CVideoclntApp::ProcessMessageFilter(int code, LPMSG lpMsg) {
 if (code < 0) CWinApp::ProcessMessageFilter(code, lpMsg);

 if (ghDlg && ghAccelTable)
 {
  if (::TranslateAccelerator(ghDlg, ghAccelTable, lpMsg))
      return(TRUE);
 }
 return CWinApp::ProcessMessageFilter(code, lpMsg);
}
```

In der InitInstance-Methode wird vor dem Instantiieren und Initialisieren des Hauptdialoges die Acceleratortabelle geladen. Zur Bearbeitung muss man die Methode ProcessMessage-Filter überschreiben und durch eigenen Quellcode ersetzen, der bei einem Tastaturkürzel die passende Menüpunkt-ID an den Hauptdialog sendet, damit die entsprechende Eventfunktion dort aufgerufen wird.

Selektierbares Auswahlfenster

Während der Analyse soll es laut Entwurf möglich sein, nicht das ganze Videobild, sondern einen Ausschnitt des Bildes zu analysieren. Die entsprechenden Analyseklassen sind auf diese Art der Verarbeitung bereits ausgelegt. Den Konstruktoren wird eine Variable vom Typ CRect (Rechteck) übergeben, welche die Eckpunkte des Auswahlfensters, im folgenden Area-of-Interest genannt, enthält. Zur Vollbildanalyse werden die Eckpunkte des kompletten Bildes übergeben. Ein Vorteil dieser Teilanalyse ist die geringere Rechenzeit, da die Normalisierung und die Strahlmittelpunkts- und Strahlgrößenberechnung nur in diesem Ausschnitt durchgeführt wird.

Die Area of Interest soll vorzugsweise mit Hilfe der Maus, durch „zeichnen" eines entsprechenden Rechteckes, definiert werden. Für die Implementation galt es zuerst, eine Klasse zu schreiben, welche die Mausereignisse auf dem Videorahmen (linke Taste gedrückt, linke Taste losgelassen, Maus bewegt) erfasst und als Events an die Klasse für die Bedienoberfläche und die interne Ablaufsteuerung (CVideoclntDlg) weiterleitet. Diese Klasse (CMyVideoFrame) wurde mit der Dialogressource des Videorahmens verbunden. Damit ist es möglich, die Mausereignisse zeitnah zu erfassen und zu bearbeiten.

Danach mussten die Funktionen zur Auswahl des Area-of-Interest-Rahmens mit Funktionalität gefüllt werden. Wenn man sich im Videofenster befindet, werden durch Druck auf die linke Maustaste die Startkoordinaten für das Auswahlrechteck festgelegt. Solange man die linke Taste gedrückt hält, kann man durch Bewegen der Maus das Rechteck größer oder kleiner machen. Wenn man die linke Taste loslässt, wird die neue Area of Interest festgeschrieben und zur Analyse verwendet. Zurücksetzen kann man das Rechteck durch Drücken und Loslassen der linken Maustaste ohne die Maus zwischenzeitlich zu bewegen. Dann ist wieder das komplette Videobild selektiert. Bei jedem neuen Videobild oder dem Neuzeichnen des aktuellen Bildes wird das Auswahlfenster als rot-schwarze Linie zur Orientierung in das Bild eingeblendet.

Projektionszeichnung

Ein weiterer wichtiger Analysebestandteil sind die X- und Y-Projektionen. Die Berechnung der Projektionen ist schon in den Analyseklassen enthalten. Nur die Anzeige der Projektion musste noch implementiert werden. Bei eingeschalteter Projektionszeichnung wird ein Achtel der Videobildhöhe für den Projektionsrahmen genutzt. Dort wird der Projektionsgraph, zusammen mit Hilfslinien, gezeichnet. Durch eine senkrechte Linie wird die durch den Mauszeiger beschriebene aktuelle Position in der Projektion eingezeichnet. Neben dem senkrechten Strich, also der aktuellen Position, wird der normalisierte Projektionswert an dieser Stelle als Zahl angegeben. Es kann entweder die X- oder Y-Projektion gezeigt werden. Die komplette Zeichnung erfolgt in der Anzeigemethode. Beim Neuzeichnen (Redraw) des

Videobildes wird der Projektionsbereich ebenfalls neu gezeichnet.

Laden und Speichern

Wichtige Funktionen für en Normalbetrieb sind das Laden und Speichern von Rohbildern und Hintergrundbildern. Rohbilder sind die Videobilder, wie sie vom Server empfangen werden, ohne Hintergrundsubtraktion, Normalisierung und Filterung. Es sollen zwei Dateiformate unterstützt werden. Dabei handelt es sich um applikationseigene Formate für Bilder (IMM) und Hintergründe (BKG). Weiterhin soll es möglich sein, die Bilder in einem Format abzuspeichern, welches von anderen Bildverarbeitungsapplikationen gelesen werden kann. Für diesen Zweck wird das BMP-Format verwendet.

Zuerst wurde die Speicherung der applikationseigenen Formate hinzugefügt. Dazu war es nötig, den Standard-Dateiauswahldialog von Windows aufzurufen, um das/die Bild(er) oder den Hintergrund unter dem angegebenen Dateinamen abspeichern zu können. Der Rohbildpuffer wird ausgelesen und zusammen mit Zusatzinformationen wie Breite und Höhe des Bildes und dem Skalierungsfaktor abgespeichert. Bei der Speicherung des Hintergrundes entfällt der Skalierungsfaktor. Beim Laden werden nach der Anzeige des Dateiauswahldialogs die Bilddaten eingelesen und in den passenden Puffer, entweder Rohbildpuffer oder Hintergrundpuffer, übertragen. Nach dem Testen des Ladens und Speicherns der applikationseigenen Formate wurde das BMP-Format hinzugefügt. In der MSDN-Dokumentation gab es Hinweise und Anregungen (DIBLOOK-Beispiel [15]), wie man eine BMP-Datei einlädt. Basierend auf diesen Anregungen wurde das Laden der BMP-Dateien implementiert. Für das Speichern genügte es, nach Erzeugung der entsprechend benötigten Header, die Header, die Palette und die Bilddaten in eine Datei zu schreiben. Das Laden der BMP-Bilder wurde auf Dateien mit einer Auflösung von 768x574 Pixel und einer Farbtiefe von 8 Bit beschränkt. Bei abweichenden Bildparametern wird eine Fehlermeldung ausgegeben. Getestet wurde der erzeugte Programmcode mit eigenen als auch mit fremderzeugten BMP-Dateien.

Schnappschussmodus

Durch den Schnappschussmodus ist es möglich, ein Videobild, so wie es auf dem Bildschirm dargestellt ist, zu speichern. Das geschieht durch das entsprechende Tastaturkürzel oder durch Auswahl des Snap-Button auf der Toolbar. Dadurch wird die Funktion OnTakeSnapshot() der Klasse für Bedienoberfläche und interne Ablaufsteuerung aufgerufen. Nach Ermittlung des Schnappschussverzeichnisses wird der Dateiname aus dem aktuellen Datum und der aktuellen Uhrzeit gebildet und das Bild als BMP-Datei abgespeichert. Für die Speicherung als BMP-Datei stand die Funktion WriteBMP(), die im Rahmen des Ladens und Speicherns entwickelt wurde, zur Verfügung. Ein Schnappschussbild ist bearbeitet, d.h. der Hintergrund wurde subtrahiert, die Bilddaten normalisiert und ggf. ist die Filterung der Röntgenstrahlen erfolgt.

Für das Speichern von Rohbildern steht die normale Bildspeicherfunktion zur Verfügung.

6.2.7 Konfigurationsdateihandling

Beim Betrieb des Client gibt es bestimmte Einstellungen, die sich nicht von vornherein festlegen lassen, sondern je nach Bedarf abgeändert werden müssen. Das sind:

1. Serverparameter (Adresse, Portnummer)
2. Vier Kameraportskalen für jeden Server
3. Schnappschussverzeichnis
4. Aktuell selektierter Server

Zur Bearbeitung der Einstellungen wurde ein Konfigurationsdialog im Projekt angelegt. Dieser erlaubt die Abänderung der Einstellungen. Es wurde eine Dialogklasse (CConfigDlg) erzeugt und mit Eventfunktionen ausgestattet. Diese werden zur Reaktion auf bestimmte Ereignisse (z.B. OK gedrückt) benötigt. Innerhalb des Dialoges sind die einzelnen Server auswählbar. Außerdem ist es möglich, einen neuen Server hinzuzufügen. Für das Schnappschussverzeichnis wurde ein Auswahldialog in einer Windows-DLL, wie er auch im Windows-Explorer Verwendung findet, übernommen. Damit ist es möglich, ein bestimmtes Verzeichnis oder Netzwerklaufwerk zur Speicherung der Bilder anzugeben. Beim Betätigen der OK-Schaltfläche werden die Einstellungen geprüft und bei erfolgreicher Prüfung übernommen.

Die im Konfigurationsdialog gemachten Änderungen werden in einer Konfigurationdatei abgespeichert. Die Konfigurationsdatei hat den Vorteil gegenüber einer Speicherung in der Windows-Registry, dass diese Datei mit dem Programm mitgegeben werden kann und nicht wie die Registry fest an eine Windows-Installation gebunden ist. Weiterhin ist es möglich, vor dem ersten Start des Programmes eine Vorkonfiguration ohne Nutzerinteraktion durchzuführen. Das ist zwar mit der Registry auch möglich, nur muss dazu ein Installationsprogramm geschrieben werden, während bei einer Datei die Installation durch ein einfaches Kopieren möglich ist.

Für die Speicherung der Daten in einer Konfigurationsdatei ist das Section/Key/Value-Schema ausreichend. Für jeden Server gibt es einen eigenen Abschnitt (Section) für die einem einzelnen Server zuordenbaren Werte. Weiterhin gibt es einen Defaultabschnitt für die anderen Werte wie den aktuell selektierten Server und das Schnappschussverzeichnis. Bei einer Webrecherche wurde eine Klasse gefunden, die die geforderte Funktionalität abdeckt. Dabei handelt es sich um die Klasse CIniEx [16]. Diese Klasse verfügt über einfache Methoden zum Lesen (getValue) und Schreiben (setValue) der einzelnen Werte. Die Methoden sind mit den in MFC angebotenen Funktionen zum Zugriff auf die Registry vergleichbar.

6.2.8 Spezielle Hintergrundsubtraktion

Parallel zur Implementierung wurden arbeitsfähige Zwischenversionen im Experiment zur Videoanalyse eingesetzt. Eine Erweiterung der Funktionalität für die Subtraktion des Hintergrundes ergab sich aus den dabei gemachten Erfahrungen. Die Funktion zum Aufnehmen eines Hintergrundbildes wurde durch eine zweite Version erweitert, welche mehrere Hintergrundbilder aufnimmt und diese entweder nach dem Envelope-Verfahren oder nach dem Average-Verfahren zu einem einzelnen Hintergrundbild zusammenfasst (siehe Abschnitt 4.2.1). Für die Erzeugung dieser Funktion wurde die Funktion zum Aufnehmen mehrerer Bilder vom Server (siehe Abschnitt 6.2.3) als Basis genommen. Resultat dieser neuen Funktion ist wieder nur ein einzelnes Hintergrundbild, welches im normalen Hintergrundbildpuffer Platz findet.

6.2.9 Klassendiagramm

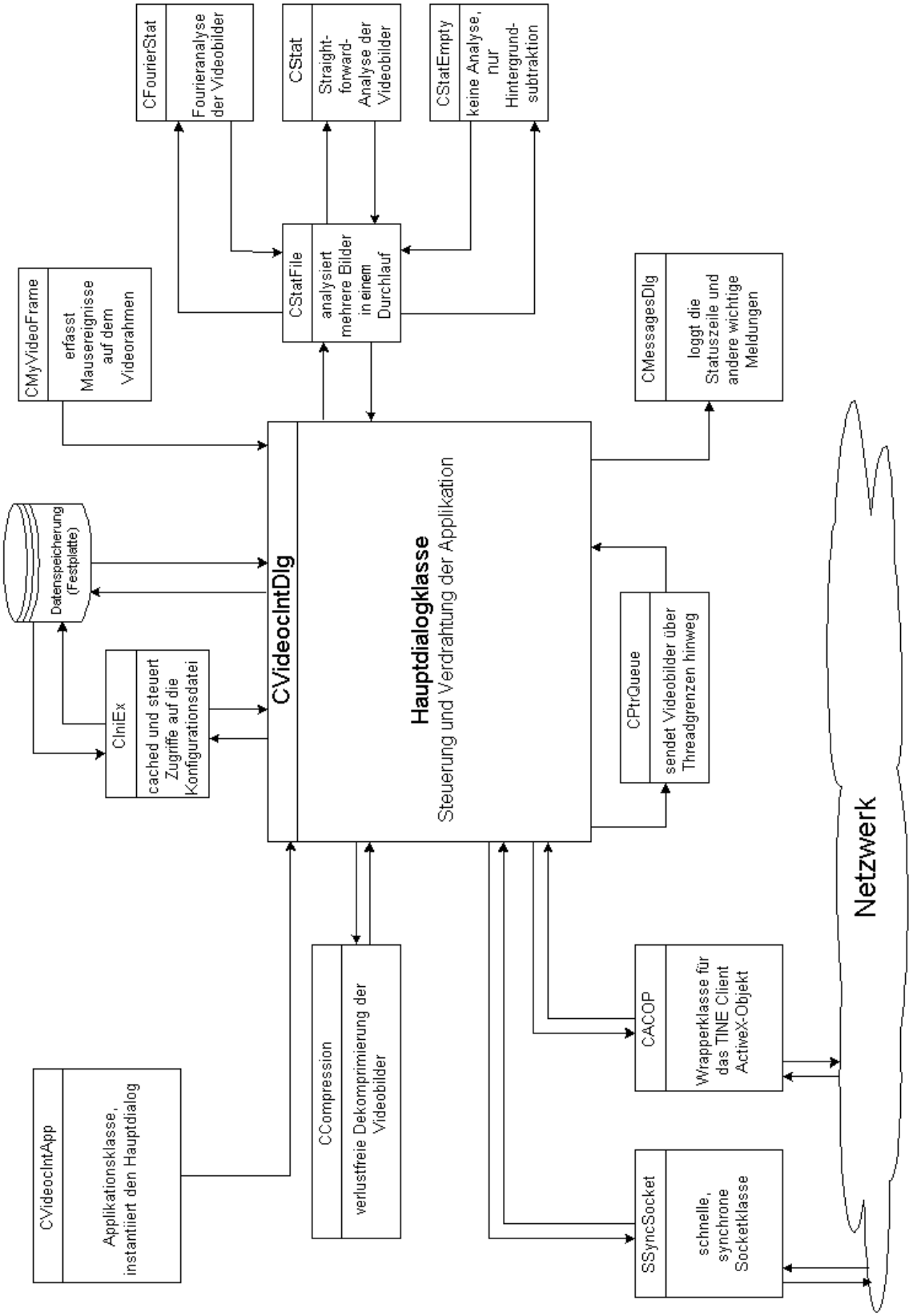

Abb. 19: Klassendiagramm Video Client

6.3 Optimierung

Ziel der Optimierung war, die Ressourcenanforderungen der Applikation zu verringern. Durch mehrere Ansätze wird versucht, die CPU-Belastung signifikant zu verringern. Drei Stufen werden dabei abgearbeitet. Zuerst wird der Programmcode nicht mehr durch den in Visual-C++ integrierten Microsoft-Compiler, sondern durch den Intel-C++ Compiler kompiliert werden. Weiterhin werden die Ansätze der Speicherverwaltung (Memory-Alignment) und der Verwendung von MMX-Instruktionen für kritische Programmabschnitte verfolgt. Hauptsächlich musste der Client optimiert werden, da die CPU-Belastung auf dem Analysecomputer, einem Pentium III 866 MHz-PC, zu hoch war. Die Erfahrungen flossen auch in die Optimierung des Serverprogramms ein.

6.3.1 Intel-C++ Compiler

Für die erste Stufe der Optimierung fiel die Entscheidung auf den Intel C++ Compiler 6.0 [17], da für den Einsatz im Videosystem nur Intel-basierte Systeme verwendet werden. Der Einsatz verspricht Vorteile gegenüber dem in Visual C++ eingebauten Microsoft-Compiler durch weitergehende Optimierungsstrategien. Ein immanent wichtiger Vorteil besteht in der nahtlosen Einbindung in die Visual C++ Entwicklungsumgebung von Microsoft. Man braucht den Intel-Compiler nur zuschalten und keine Änderungen am Quellcode vorzunehmen. Eine weitergehende Optimierung findet aber erst nach Anpassung der Parameter für die Kompilierung statt. Ein weiterer Vorteil des Einsatzes gegenüber anderen Methoden ist die Optimierung des **gesamten** Programmcodes und nicht nur weniger kritischer Stellen.

Ein Testlauf soll zeigen, wie viel CPU-Zeit sich durch den Intel Compiler einsparen lässt. Dazu wurde der Quellcode zweimal kompiliert, einmal mit dem in Visual C++ integrierten Microsoft Compiler und einmal mit dem Intel C++ Compiler 6.0 . Beim Microsoft Compiler wurde auf Geschwindigkeit optimiert. Die benutzten Optimierungsoptionen für den Intel C++ Compiler waren:

Option	**Beschreibung**
/O3	Optimierung auf Geschwindigkeit
/QaxM	Einsatz von MMX-Instruktionen
/G6	Optimierung für Pentium II und Pentium III Prozessoren
/Og	Globale Optimierung
/Ob2	Inline-Funktionen erzeugen wenn möglich

Während der Durchführung des Tests war die Grabbingfrequenz auf 10 Hz eingestellt. Das Analyseprogramm wurde auf einem Pentium IV 1700 MHz gestartet. Alle für den Live-Betrieb möglichen Analyseoptionen wurden eingeschaltet, d.h. Hintergrundsubtraktion, Fourierana-

lyse, Normalisierung und X-Projektion. Während das Analyseprogramm lief wurde der Windows NT Task Manager im Hintergrund ausgeführt. Vom Diagramm der CPU-Zeit wurde ein Bildschirmfoto angefertigt. Zwei Testdurchläufe fanden statt: Beim ersten Test wurde das Analyseprogramm, kompiliert durch den Microsoft Compiler, gestartet. Für den zweiten Testlauf wurde der vom Intel C++ Compiler erzeugte Maschinencode verwendet.

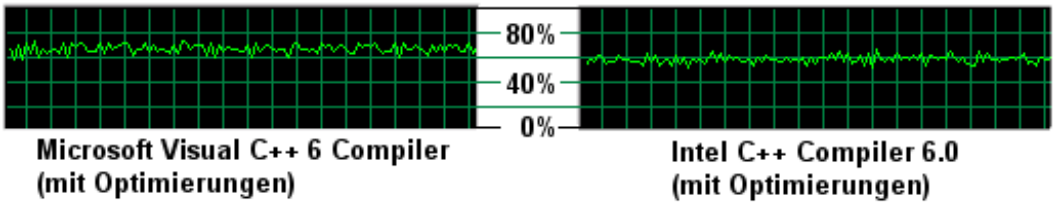

Abb. 20: Vergleich Microsoft Visual C++ und Intel C++ Compiler

Die Ergebnisse sind in Abb. 20 dargestellt. Grob abgeschätzt beträgt der Unterschied zwischen den Kurvenverläufen knapp 10 %. Nur durch eine Neukompilierung und das setzen von speziellen Compileroptionen lassen sich im Maximalbetrieb ca. 10 % CPU-Belastung auf dem Testcomputer einsparen.

Nach dieser ,,Grundoptimierung" wurde festgestellt, dass einige Kernroutinen noch weitere Optimierung nötig hatten. Insbesondere geht es dabei um die Bildverarbeitung in den Analyseklassen. Die Hintergrundsubtraktion von 10 Videobildern mit einer Auflösung von 768x574 Pixel stellt hohe Anforderungen an den Clientcomputer.

6.3.2 Optimierung durch MMX-Instruktionen

Nachdem die Grundoptimierung durch eine Übersetzung mit Hilfe des Intel-Compiler durchgeführt war, wurde eine Routine, die oft durchlaufen wird und eine sehr hohe CPU-Belastung zur Folge hat, optimiert. Der Algorithmus, auf den das zutrifft, ist die Hintergrundsubtraktion. Eine Hintergrundsubtraktion wird in C++ durch folgenden Quellcode implementiert:

```
for (unsigned int i=0;i<ImageHeight*ImageWidth;i++)
{
  int pix = (int) dataptr[i];
  int back = (int) backgnd[i];
  int out;

  if ((out = pix-back) < 0) out = 0;

  outptr[i] = (unsigned char) out;
}
```

Wenn der Compiler nicht optimiert, müssen für jeden Pixel mindestens zwei Operationen durchgeführt werden. Zuerst erfolgt eine Subtraktion des Hintergrundpixels vom Bildpixel. Danach wird ein Vergleich durchgeführt, ob das Ergebnis kleiner Null ist. Wenn das der Fall ist, wird als dritte Operation der resultierende Pixel gleich Null gesetzt. Im Normalfall benötigt man pro Pixel zwei Operationen, die Subtraktion und den Vergleich. Bei 10 Hz Bildwiederholrate und einer Bildauflösung von 768x574 Pixel ergibt sich als Anzahl der durchzuführenden Operationen pro Videobild:

$$768 \cdot 574 \cdot 2 \; Operationen \cdot 10 \; Hz = 8816640 \; Operationen$$

Bei den benötigten ≈ 8.8 Millionen Operationen ist das Laden der Daten in Register und die Rückspeicherung vom Register in den Speicher vernachlässigt worden.

Einen guten Ansatz zur Verringerung der Anzahl der Operationen bietet der MMX-Instruktionssatz, der in modernen CPUs eingebaut ist. MMX bietet SIMD-Instruktionen[5]. Mit einer einzelnen Instruktion können mehrere Datenwörter auf einmal bearbeitet werden. Die Register für MMX sind 64 Bit breit, es finden acht 8-Bit breite Pixel Platz. Die Instruktionen von MMX sind so ausgelegt, dass man angeben kann, wie breit die Daten in den Registern sind. Des weiteren gibt es bei MMX eine Subtraktionsoperation, mit der der Vergleich eingespart werden kann. Diese Operation nennt sich ,,subtraction with unsigned saturation per byte". Durch diese Operation ist es möglich, acht Pixel auf einmal so zu subtrahieren, dass der Zielwert, wenn er kleiner als Null ist, auf Null gesetzt wird. Das folgende Beispiel verdeutlicht dies. Im Beispiel enthält das Register reg1 acht Pixel aus dem Rohbild und das Register reg2 acht Pixel des Hintergrundbildes.

	reg1	255	127	64	32	15	8	2
-	reg2	128	255	64	33	8	1	0
=		127	0	0	0	7	7	2

Das Beispiel aus der Tabelle wird in einer einzigen Operation abgearbeitet. Berechnet man nun wieder die Anzahl der Operationen, so ergibt sich:

$$768 * 574/8 * 1 \; Op. * 10Hz = 551040 \;\; Operationen$$

Bei der MMX-Methode werden nur 6.25 % der für die alte Methode benötigten Operationen ausgeführt. Wieviel Leistungssteigerung davon in der Praxis übrigbleibt, da der Intel-Compiler auch den C++ Quellcode gut optimieren kann, soll durch ein Testprogramm evaluiert werden. Dieses Programm führt Hintergrundsubtraktionen durch. Der erste Testlauf wurde mit C++-Code, welcher durch den Intel-Compiler optimiert wurde, durchgeführt. Dabei fanden die gleichen Optimierungsparameter wie in Abschnitt 6.3.1 Anwendung. Beim

[5] Single Instruction Multiple Data

zweiten Testlauf wurde die Hintergrundsubtraktion mit MMX-Instruktionen durchgeführt. Das Programm läuft genau 30 Sekunden und führt in einer Schleife so viele Hintergrundsubtraktionen wie möglich durch. Nach 30 Sekunden beendet sich das Programm automatisch und protokolliert die Anzahl der durchgeführten Hintergrundsubtraktionen pro Sekunde. Die Programme wurden jeweils 4 mal gestartet. Die Anzahl der Hintergrundsubtraktionen pro Sekunde, die durchgeführt wurden, sind in der folgenden Tabelle aufgetragen:

Testlauf Nr.	**C++**	**MMX**
1	582.41	807.39
2	581.58	817.76
3	581.24	823.49
4	580.24	804.05
Durchschnitt	581.37	813.17
Abweichung	0 %	+39.87 %

Das Resultat der Testläufe ist ein Geschwindigkeitszuwachs bei der Hintergrundsubtraktion um $\approx 40\%$ durch Verwendung der MMX-Instruktionen. Die Einsparung im kompletten Analyseprogramm ist aber geringer als 40 %, da die Hintergrundsubtraktion nur einen Teil der durchlaufenen Algorithmen ausmacht.

6.3.3 Optimierung durch verbesserte Speicherverwaltung

Ein wichtiger Punkt bei der Optimierung auf modernen Prozessoren ist die korrekte Speicherausrichtung, das sogenannte Memory-Alignment. Nicht ausgerichtete Speicherzugriffe kosten Strafzyklen und verlangsamen somit den Speicherzugriff. Da viele Speicherzugriffe durchgeführt werden, kann ein richtig eingesetztes Memory Alignment Vorteile bringen. Besonders viele Speicherzugriffe gibt es bei der Hintergrundsubtraktion, bei der Normalisierung und der Analyse der Videodaten.

Die normalen Speicheroperatoren wie new und delete in Visual C++ nehmen keine Speicherausrichtung vor. Beim Intel-Compiler gibt es die speziellen Speicherroutinen _mm_malloc() und _mm_free(). Beim Microsoft Visual C++ - Compiler benötigt man das Visual C++ Processor Performance Pack [18]. Dort gibt es die Funktionen _aligned_malloc() und _aligned_free(). Den Allokationsroutinen _mm_malloc() und _aligned_malloc() wird die Granularität der benötigten Speicherbereiche als Parameter übergeben. In der vorliegenden Applikation werden die Speicherbereiche maximal als 64 Bit-Wörter angesehen. Dafür empfiehlt sich eine Granularität von 8, d.h. alle Speicherbereiche beginnen an durch 8 teilbaren Adressen [19]. Für die Speicherallokation wurde die Klasse CMemoryManagement erstellt. Mit Hilfe von Präprozessordirektiven werden die für den aktuell selektierten Compiler benötigten Allokations-

und Deallokationsroutinen bestimmt. Die Routinen kommen gekapselt zum Einsatz. In der Applikation gibt es dann die Funktionen sw_malloc() und sw_free().

Inwieweit die Speicherausrichtung im Analyseprogramm Vorteile bringt, soll durch ein Testprogramm evaluiert werden. Im Testprogramm läuft eine Schleife für 30 Sekunden und führt Hintergrundsubtraktion und Normalisierung der Videodaten durch. Nach 30 Sekunden beendet sich das Programm automatisch und erzeugt eine Ausgabe, wie viele Durchläufe der Schleife (Hintergrundsubtraktion + Normalisierung) durchgeführt wurden. Das Programm wurde einmal mit und einmal ohne Speicherausrichtungsroutinen gestartet. Die Ergebnisse, d.h. die Schleifendurchläufe pro Sekunde, sind in der folgenden Tabelle eingetragen:

Testlauf Nr.	**ohne** Speicherausrichtung	**mit** Speicherausrichtung
1	137.45	145.79
2	138.19	145.95
3	138.25	145.85
Durchschnitt	137.96	145.86
Abweichung	0 %	+5.73 %

Das Resultat der Testläufe ist ein Geschwindigkeitszuwachs bei Hintergrundsubtraktion und Normalisierung um $\approx 5\%$ durch Verwendung der Speicherausrichtung.

6.3.4 Zusammenfassung

Zum Abschluß der Optimierung wurde das Analyseprogramm auf dem Zielsystem, einem Pentium III 866 MHz-PC, gestartet. Die Framerate war auf 10 Hz eingestellt. Alle Analyseoptionen wurden eingeschaltet. Beim Testlauf **vor** der Optimierung lag die CPU-Belastung bei 100 %, der PC wurde überfordert. Nach Abschluss der Optimierung liegt die CPU-Belastung auf dem Zielsystem zwischen 90 und 95%. Die Optimierung hat ihr Ziel erreicht, die Applikation ist auf dem Zielsystem ohne Einschränkungen lauffähig.

7 Ausblick

Die in dieser Arbeit besprochenen Programme werden seit einigen Monaten im Produktiveinsatz beim PITZ-Experiment im DESY-Zeuthen eingesetzt. Bis auf kleinere Probleme verlief der Produktiveinsatz erfolgreich. Es ist geplant, das System noch weiter auszubauen. Es wird durch weitere Auswertungsprogramme, die ihre Bilder vom Server holen, ergänzt werden. Auch ist geplant, das Standardauswertungsprogramm (Video Client) algorithmisch auszubauen und zu verbessern. Beispiele für geplante Verbesserung sind die weitere Optimierung der Hintergrundsubtraktion und die Geschwindigkeitsanpassung der Röntgenstrahlenfilterung (für den Live-Betrieb).

Beim Server wird darüber nachgedacht, auch den Transfer der Videobilder über das TINE-Protokoll in der Multicast-Betriebsart abzuwickeln. Neuere Versionen des TINE-Protokolls und eine Neukonfiguration der Netzwerkhardware zeigen eine deutliche Verringerung der verlorengegangenen Bilder. Damit wäre es möglich, nahezu unbegrenzt viele Clients gleichzeitig an den Server anbinden zu können, und das bei geringerer Belastung des Servers und des Netzwerkes. Weiterhin soll der Server ausgebaut werden, um spezielle Analysedaten permanent verfügbar zu haben. Angedacht ist in diesem Zusammenhang ein Spiegelsteuersystem, welches den Laserstrahl automatisch immer auf den gleichen Punkt der Photokathode lenkt.

Weiterhin soll eine Streak-Kamera, welche zur Optimierung des Laserstrahls dient, mit in das Videosystem eingebunden werden. Ein neues Auswertungsprogramm würde dann die Bilder von der Streakkamera empfangen und die Laserparameter auf Basis der empfangenen Bilder autonom optimieren.

A Inhalt der CD

Auf der dieser Diplomarbeit beiliegenden CD sind alle WWW-Quellen, der Sourcecode, die Dokumentationen, die Zusatztools und diese schriftliche Arbeit selber abgelegt. Die Struktur ist auf der folgenden Abbildung erkennbar.

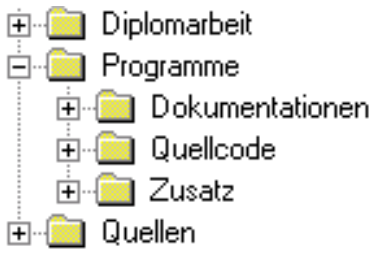

Abb. 21: Ausgewählte Verzeichnisse auf der beigelegten CD

Im Verzeichnis ,,Diplomarbeit" befindet sich das Schriftstück im PDF-Format. Im Verzeichnis ,,Programme" befindet sich der Quellcode, die Dokumentationen und zusätzliche Dateien zu den Programmen. Im Verzeichnis ,,Quellen" sind die in der Diplomarbeit zitierten Quellen abgelegt.

B Farbmapping-Tabelle

In der folgenden Tabelle ist die Zuordnung zwischen Graustufenwerten und den Farbwerten beim Falschfarbenmodus aufgetragen. Die Zahlen geben Intensitätswerte der Graustufe bzw. Farbe wieder. Null bedeutet keine Intensität, während 255 die höchste Intensität bedeutet. Die anderen Intensitäten liegen linear dazwischen. Die Farbkomponenten Rot, Grün und Blau werden additiv zur passenden Farbe gemischt.

Graustufe	Rot	Grün	Blau	Farbe	Graustufe	Rot	Grün	Blau	Farbe
0	0	0	0		32	0	0	202	
1	0	0	255		33	0	0	200	
2	0	0	253		34	0	0	198	
3	0	0	251		35	0	0	197	
4	0	0	249		36	0	0	195	
5	0	0	248		37	0	0	193	
6	0	0	246		38	0	0	192	
7	0	0	244		39	0	0	190	
8	0	0	243		40	0	0	188	
9	0	0	241		41	0	0	187	
10	0	0	239		42	0	0	185	
11	0	0	238		43	0	0	183	
12	0	0	236		44	0	0	181	
13	0	0	234		45	0	0	180	
14	0	0	232		46	0	0	178	
15	0	0	231		47	0	0	176	
16	0	0	229		48	0	0	175	
17	0	0	227		49	0	0	173	
18	0	0	226		50	0	0	171	
19	0	0	224		51	0	0	170	
20	0	0	222		52	0	0	168	
21	0	0	221		53	0	0	166	
22	0	0	219		54	0	0	164	
23	0	0	217		55	0	0	163	
24	0	0	215		56	0	0	161	
25	0	0	214		57	0	0	159	
26	0	0	212		58	0	0	158	
27	0	0	210		59	0	0	156	
28	0	0	209		60	0	0	154	
29	0	0	207		61	0	0	153	
30	0	0	205		62	0	0	151	
31	0	0	204		63	0	0	149	

Graustufe	Rot	Grün	Blau	Farbe	Graustufe	Rot	Grün	Blau	Farbe
64	0	0	147		96	0	89	93	
65	0	0	146		97	0	91	91	
66	0	0	144		98	0	94	90	
67	0	0	142		99	0	96	88	
68	0	0	141		100	0	98	86	
69	0	0	139		101	0	100	85	
70	0	0	137		102	0	103	83	
71	0	0	136		103	0	105	81	
72	0	0	134		104	0	107	79	
73	0	0	132		105	0	109	78	
74	0	0	130		106	0	111	76	
75	0	0	129		107	0	114	74	
76	0	0	127		108	0	116	73	
77	0	0	125		109	0	118	71	
78	0	0	124		110	0	120	69	
79	0	0	122		111	0	123	68	
80	0	0	120		112	0	125	66	
81	0	0	119		113	0	127	64	
82	0	0	117		114	0	129	62	
83	0	0	115		115	0	132	61	
84	0	0	113		116	0	134	59	
85	0	0	112		117	0	136	57	
86	0	67	110		118	0	138	56	
87	0	69	108		119	0	141	54	
88	0	71	107		120	0	143	52	
89	0	73	105		121	0	145	51	
90	0	76	103		122	0	147	49	
91	0	78	102		123	0	149	47	
92	0	80	100		124	0	152	45	
93	0	82	98		125	0	154	44	
94	0	85	96		126	0	156	42	
95	0	87	95		127	0	158	40	

Graustufe	Rot	Grün	Blau	Farbe	Graustufe	Rot	Grün	Blau	Farbe
128	0	161	39		160	0	232	0	
129	0	163	37		161	0	234	0	
130	0	165	35		162	0	237	0	
131	0	167	34		163	0	239	0	
132	0	170	32		164	0	241	0	
133	0	172	30		165	0	243	0	
134	0	174	28		166	0	246	0	
135	0	176	27		167	0	248	0	
136	0	179	25		168	0	250	0	
137	0	181	23		169	0	252	0	
138	0	183	22		170	0	255	0	
139	0	185	20		171	67	0	0	
140	0	187	18		172	69	0	0	
141	0	190	17		173	71	0	0	
142	0	192	15		174	73	0	0	
143	0	194	13		175	76	0	0	
144	0	196	11		176	78	0	0	
145	0	199	10		177	80	0	0	
146	0	201	8		178	82	0	0	
147	0	203	6		179	85	0	0	
148	0	205	5		180	87	0	0	
149	0	208	3		181	89	0	0	
150	0	210	0		182	91	0	0	
151	0	212	0		183	94	0	0	
152	0	214	0		184	96	0	0	
153	0	217	0		185	98	0	0	
154	0	219	0		186	100	0	0	
155	0	221	0		187	103	0	0	
156	0	223	0		188	105	0	0	
157	0	225	0		189	107	0	0	
158	0	228	0		190	109	0	0	
159	0	230	0		191	111	0	0	

Graustufe	Rot	Grün	Blau	Farbe	Graustufe	Rot	Grün	Blau	Farbe
192	114	0	0		224	185	0	0	
193	116	0	0		225	187	0	0	
194	118	0	0		226	190	0	0	
195	120	0	0		227	192	0	0	
196	123	0	0		228	194	0	0	
197	125	0	0		229	196	0	0	
198	127	0	0		230	199	0	0	
199	129	0	0		231	201	0	0	
200	132	0	0		232	203	0	0	
201	134	0	0		233	205	0	0	
202	136	0	0		234	208	0	0	
203	138	0	0		235	210	0	0	
204	141	0	0		236	212	0	0	
205	143	0	0		237	214	0	0	
206	145	0	0		238	217	0	0	
207	147	0	0		239	219	0	0	
208	149	0	0		240	221	0	0	
209	152	0	0		241	223	0	0	
210	154	0	0		242	225	0	0	
211	156	0	0		243	228	0	0	
212	158	0	0		244	230	0	0	
213	161	0	0		245	232	0	0	
214	163	0	0		246	234	0	0	
215	165	0	0		247	237	0	0	
216	167	0	0		248	239	0	0	
217	170	0	0		249	241	0	0	
218	172	0	0		250	243	0	0	
219	174	0	0		251	246	0	0	
220	176	0	0		252	248	0	0	
221	179	0	0		253	250	0	0	
222	181	0	0		254	252	0	0	
223	183	0	0		255	255	255	255	

C Begriffserläuterungen

Begriff	Beschreibung
ActiveX-Control	Unter ActiveX-Control versteht man eine Softwarekomponente innerhalb des Windows-Betriebssystems. Der ActiveX-Standard wurde von Microsoft entwickelt.
API	Unter API (Application Programming Interface) versteht man eine Schnittstelle (Sammlung von Routinen und Datenstrukturen), die als Grundlage für Applikationsprogrammierung benutzt wird. Ein gutes API vereinfacht die Programmierung einer Applikation.
Area of Interest	Area of Interest heißt sinngemäß selektierbares Auswahlfenster. Es ist eine rechteckige Region des Videofensters, die für die Analyse wichtig ist. Auswertungen finden nur in diesem ausgewählten Bereich statt.
Assembler	Unter Assembler versteht man ein Programm, welches die symbolische Assemblersprache in Maschinencode übersetzt. Assemblersprache ist schwer zu erlernen und besteht aus den atomaren Instruktionen eines Prozessors.
Auswahlfenster	siehe Area of Interest
Broadcast	Broadcast bezeichnet das Senden von Informationen an **alle** verbundenen Teilnehmer. Zum Beispiel ist eine Fernseh- oder Radioausstrahlung an alle Teilnehmer ein Broadcast. Im Datennetz wird Broadcast verwendet, um Informationen und Anfragen an alle Teilnehmer zu verteilen.
Client	Ein Client ist Teil einer Client/Server-Architektur. Ein Client verbindet sich typischerweise zu einem Server, um dessen Dienste in Anspruch zu nehmen.
CODEC	Kurzform für COmpression/DECcompression. Softwarekomponente, die sich um das Komprimieren und Dekomprimieren von Datenströmen kümmert. CODECs sind weit verbreitet im Audio/Video-Bereich.
DFT	Abkürzung für Diskrete Fouriertransformation. Methode in der Mathematik basierend auf der Annahme, dass sich jede Wellenform als Überlagerung simpler Sinusschwingungen darstellen lässt. Die DFT findet in dieser Arbeit Anwendung als Strahlmittelpunktsberechnung.

Begriff	**Beschreibung**
Dropped Frames	Unter Dropped Frames versteht man das Auslassen von Bildern in einer Einzelbildfolge oder eines kontinuierlichen Bilderstromes. Dropped Frames entstehen auf Grund von Rechenleistungs- oder Bandbreitenengpässen.
Dunkelstrom	Normalerweise werden durch Laserimpulse an der Kathode die Photoelektronen zum Austreten aus der Kathode bewegt. Danach werden die Elektronenpakete durch ein sehr starkes elektrisches Feld geführt (35 $\frac{MV}{m}$) und beschleunigt. Es kann aber vorkommen, dass auch ohne einen Laserstrahl durch das starke Feld Elektronen aus den umgebenden Materialen (z.B. Kupfer) herausgelöst und beschleunigt werden. Diese werden als Ladungsträger (Dunkelstrom) auf den Bildschirmen und Diagnosewerkzeugen sichtbar.
Emittanz	Unter Emittanz versteht man das Produkt aus Divergenz und RMS-Größe eines Elektronenstrahls. Siehe auch EMSY
EMSY	Kurzform für Emittance Measurement System. EMSY ist ein Messsystem im PITZ-Experiment, um die Emittanz eines Elektronenpaketes zu ermitteln. Unter Emittanz versteht man das Produkt aus Divergenz und RMS-Größe des Elektronenstrahls. Ziel des Experimentes ist es, Elektronenpakete mit möglichst geringer Emittanz zu erzeugen.
Falschfarbenmodus	Unter Falschfarbenmodus versteht man die Benutzung von (falschen) Farben um Intensitäten der Pixel wiederzugeben. Der Falschfarbenmodus dient zur Sichtbarmachung geringer Intensitäten und Intensitätsunterschieden. Der Falschfarbenmodus erleichtert auch die Abschätzung der Stärke eines Videosignales.
FEL	Abkürzung für Freie Elektronen Laser. Bei einem Freie Elektronen Laser werden, im Gegensatz zum Festkörperlaser, die Photonen von ungebundenen Elektronen erzeugt, die sich in einem wechselnden Magnetfeld bewegen. Bei einem Festkörperlaser dagegen werden die Photonen durch gebundene Zustände der Elektronen in Atomen erzeugt [20].

Begriff	**Beschreibung**
Framegrabberkarte	Eine Framegrabberkarte ist eine Steckkarte für einen Computer, mit deren Hilfe es möglich ist, Videobilder zu digitalisieren und in den Computer einzulesen.
Framerate	Unter dem Begriff Framerate versteht man die Bildwiederholrate einer Videoübertragung. Die Einheit ist Bilder pro Sekunde ($1\ s^{-1} = 1Hz$).
Halbbild	Fernsehen arbeitet nicht wie ein moderner Computerbildschirm mit Vollbildern, sondern mit Halbbildern. Jedes Fernsehbild ist aus zwei ineinander verwobenen Halbbildern zusammengesetzt. Diese beiden Halbbilder werden zu unterschiedlicher Zeit aufgenommen. Halbbildaufzeichnung ist für die Analyse der physikalischen Experimente im PITZ ungeeignet.
HuffYUV	HuffYUV ist ein Videokompressionscodec, welcher basierend auf statischen Tabellen die Huffman-Komprimierung auf ein einzelnes Videobild anwendet. Die Kompression ist schnell und verlustlos.
ITEX	ITEX ist das API zum Ansprechen der Framegrabberkarte PCVision von Coreco Imaging.
Kameraport	Ein Kameraport ist ein Verbindungspunkt an einer Framegrabberkarte, an den eine Videokamera angeschlossen werden kann.
Makropuls	Unter Makropuls versteht man die Zusammenfassung von gleichartigen Mikropulsen. Ein Mikropuls ist ein kurzer Laserblitz. Mehrere Mikropulse (Pikosekundenzeitdauer) werden zu einem Makropuls mit einer Zeitdauer von Mikrosekunden zusammengefasst. Jeder Mikropuls erzeugt durch Illumination an der Photokathode ein Photoelektronenpaket. (siehe Abschnitt 1.3.1).
MFC	MFC ist die Abkürzung für Microsoft Foundation Classes, eine Klassenbibliothek zur Vereinfachung der Entwicklung von Windows-Programmen.
MMX	Kurzform für Multimedia-Extensions. MMX ist eine Befehlssatzerweiterung für x86-PCs, die es erlaubt, mit einer Instruktion eine Operation auf eine Vielzahl von gleichartigen Datenelementen anzuwenden. Das Ziel ist eine Geschwindigkeitsteigerung.

Begriff	Beschreibung
MSDN	MSDN ist die Abkürzung für Microsoft Developer Network, einem Netzwerk, in dem für Entwickler alle Informationen zu Microsoft Produkten und zur Programmentwicklung unter Microsoft Betriebssystemen verfügbar sind.
Multicast	Unter Multicast versteht man das Versenden von Daten an eine Gruppe von Empfängern. Beispielsweise ist eine Videokonferenz mit drei oder mehr Teilnehmern ein Multicast-Vorgang.
Netio	Benchmarkprogramm zur Einmessung der Netzwerktransfergeschwindigkeit via TCP/IP.
Pepper Pot	Pepper-Pot ist eine Anordnung im PITZ-Experiment, um Beamlet-Muster zu erzeugen und zu analysieren. Der Elektronenstrahl wird durch Spaltmasken geführt. Durch die entstehenden Beugungsbilder kann die Divergenz des Strahls ermittelt werden.
PITZ	Abkürzung für Photoinjektor-Teststand Zeuthen, Experimentiereinrichtung zur Erforschung von lasergetriebenen Elektronenquellen für Freie-Elektronen-Laser und Linearbeschleuniger.
Pixel	Ein Pixel (Bildpunkt) ist ein elementarer Bestandteil eines Bildes. Ein Bild besteht aus N Pixeln.
Präprozessor	Der Präprozessor ist eine Einrichtung in einem C-Compiler, welcher vor der Kompilierung des Quellcodes vom Programmierer gewünschte Änderungen in diesem vornehmen kann.
Prototyping	Unter Prototyping versteht man die Entwicklung von Prototypenprogrammen vor der eigentlichen Programmimplementation, um nicht verzichtbare Funktionalität zu testen und sicherzustellen, dass später alles erwartungsgemäß funktioniert.
RMS	Kurzform für Root-Mean-Square (quadratischer Mittelwert).
Server	Als Server bezeichnet man eine Hardware/Software-Kombination, d.h. einen Computer, der bestimmte Dienste anbietet, die von anderen Computern, den Clients, benutzt werden können.
Shutter-Speed	Der Shutter-Speed ist die Blendenöffnungszeit einer Kamera.
Streakkamera	Eine Streakkamera ist keine Kamera in eigentlichem Sinn, sondern eines der schnellsten optischen Detektorsysteme der Welt. Mit einer Streakkamera kann man pikosekundenkurze Lichtmikropulse aufnehmen.

Begriff	**Beschreibung**
TCP	Abkürzung für Transmission Control Protocol. Unter TCP versteht man ein Protokoll oberhalb des Internetprotokolles (IP) zur Steuerung des Transfers von Daten. TCP ist ein Protokoll der Transportschicht und beinhaltet Fehlererkennung und Fehlerkorrektur auf dem Übertragungsweg.
Thread	Ein Thread ist ein separater Ausführungspfad in einem Programm, der parallel zum Hauptprogramm abgearbeitet wird.
Tiefpassfilter	Unter einem Tiefpassfilter versteht man eine Filterung, die niedrige Frequenzen durchlässt aber hohe Frequenzen blockiert.
TINE	Abkürzung für Three-fold Integrated Network Environment. TINE ist ein Netzwerkprotokoll innerhalb des DESY zur Verbindung von Computern innerhalb von Kontrollsystemen. Es enthält server- und clientseitige APIs und deckt die Schichten 2-4 eines Kontrollsystems ab (siehe Abb. 3).
Trigger	Unter Trigger versteht man ein elektronisches Steuer- oder Schaltsignal.
UDP	Abkürzung für User Datagram Protocol. Unter UDP versteht man ein Protokoll oberhalb vom Internet Protokoll (IP). UDP ist wie TCP ein Protokoll der Transportschicht. Es ist verbindungslos und bietet nur sehr eingeschränkte Fehlerkorrekturmechanismen.
Unicast	Unter Unicast versteht man eine Verbindung zwischen **einem** Sender und **einem** Empfänger.
VCM	Kurzform für Video Compression Manager. Der VCM ist eine Softwarekomponente innerhalb von Windows, die es ermöglicht, unkompliziert (De-)Kompressionsalgorithmen auf Videobilder anzuwenden.

Abbildungsverzeichnis

Quellenverzeichnis

[1] Homepage des Photoinjektor-Teststandes (PITZ) im DESY-Zeuthen,
Zugriffsdatum 02.11.2002,
URL: http://desyntwww.desy.de/pitz/

[2] Homepage des TESLA-Projektes,
Zugriffsdatum 02.11.2002,
URL: http://tesla.desy.de/

[3] DOOCS - The Distributed Object Oriented Control System,
Homepage von DOOCS,
Zugriffsdatum 15.12.2002,
URL: http://doocs.desy.de

[4] Homepage des TINE-Protokolles auf dem WWW-Server vom DESY,
Zugriffsdatum 02.11.2002,
URL: http://desyntwww.desy.de/tine/

[5] BESSY - Berliner Elektronenspeicherring - Gesellschaft für Synchrotronstrahlung m. b. H.
Zugriffsdatum: 12.11.2002,
URL: http://www.bessy.de/home.php

[6] Einführung in Sockets bei der Fachhochschule Worms,
Zugriffsdatum: 28.11.2002
Einstieg: http://www.ztt.fh-worms.de/de/sem/ss95/sockets/node1.html,
Direkter Bezug: http://www.ztt.fh-worms.de/de/sem/ss95/sockets/node2.html#SECTION00020000000000000000

[7] Winsock-Spezifikation auf dem FTP-Server von Microsoft,
Zugriffsdatum 02.11.2002,
URL: ftp://ftp.microsoft.com/bussys/winsock/spec11/winsock.txt

[8] Technische Spezifikation des Video-Compression-Manager (VCM) auf dem WWW-Server von Microsoft,
Zugriffsdatum 02.11.2002,
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/avicomp_550y.asp

[9] MFC - Microsoft Foundation Classes,
Zugriffsdatum: 10.11.2002,
Einstiegspunkt: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/_mfc_About_the_Microsoft_Foundation_Classes.asp
Bezug: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/_mfc_general_class_design_philosophy.asp

[10] Homepage des HuffYUV-Videocodecs von Ben Rudiak Gould,
Zugriffsdatum 02.11.2002,
URL: http://math.berkeley.edu/~benrg/huffyuv.html

[11] Homepage der Firma JAI (Kamerahersteller),
Zugriffsdatum 02.11.2002,
URL: http://www.jai.com

[12] Homepage der Firma Coreco Imaging (Framegrabberhersteller),
Zugriffsdatum 02.11.2002,
URL: http://www.imaging.com

[13] Downloadadresse des NETIO-Benchmarks v1.14 auf dem FTP-Server von LEO,
Zugriffsdatum 02.11.2002,
URL: ftp://ftp.leo.org/pub/comp/os/os2/leo/systools/netio114.zip

[14] Stefan Weisse,
GrabServer Programmer's Documentation,
Revision 6 vom 10. Oktober 2002,
S. 10f

[15] DIBLOOK-Beispiel - Laden von BMP-Dateien,
Zugriffsdatum 17.12.2002,
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcsample/html/_sample_mfc_diblook.asp

[16] CIniEx - Klasse mit Routinen zur Konfigurationsdateibehandlung,
Zugriffsdatum: 02.11.2002,
URL: http://codeguru.earthweb.com/data-misc/CIniEx.html

[17] Intel-Compiler 6.0 für Windows Homepage,
Zugriffsdatum 02.11.2002,
URL: http://www.intel.com/software/products/compilers/c60/

[18] Microsoft Processor Performance Pack for Visual C++ 6 SP5,
Zugriffsdatum: 02.11.2002
Readme-Datei: http://download.microsoft.com/download/vb60ent/Update/6/W9X2KXP/EN-US/ppreadme.exe,
Download: http://msdn.microsoft.com/vstudio/downloads/tools/ppack/default.asp

[19] Intel Architecture Optimization - Reference Manual,
Intel Corporation 1998/1999,
Order Number: 245127-001,
S. 2-12

[20] TESLA Technical Design Report,
Part V - The X-Ray Free Electron Laser,
März 2001,
S. 23ff,
Zugriffsdatum: 07.03.2003
http://tesla.desy.de/new_pages/TDR_CD/PartV/xfel.pdf

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Diplomarbeit selbst angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

.....