Diplomarbeit

Ilia Revout

Design und Realisierung eines Frameworks für Bildverarbeitung

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Ilia Revout

Design und Realisierung eines Frameworks für Bildverarbeitung

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 08.12.2003

Ilia Revout

Thema der Diplomarbeit

Design und Realisierung eines Frameworks für Bildverarbeitung

Stichworte

Bildverarbeitung, Imagegrabbing, Framework, Bildfilter, Robotersehen, Robotersteuerung

Kurzzusammenfassung

Diese Diplomarbeit beschäftigt sich mit dem Design und mit der Realisierung eines Frameworks für die Bildverarbeitung in der Echtzeit für Anwendungen im Bereich autonome mobile Roboter. Mit Hilfe dieses Frameworks soll es möglich sein, für eine bestimmte Problemstellung eine passende Filterkombination zu finden, die zur Problemlösung führen wird. Das Framework unterstützt dabei die Entwicklung von neuen Filtern bzw. bietet die Möglichkeit, schon vorhandene Filter zu integrieren. So entwickelte oder integrierte Filter können dann beliebig kombiniert und einzeln konfiguriert werden. Mit Hilfe vom Framework ist es möglich, spezielle Softwarelösungen im Bereich Bildverarbeitung in kürzester Zeit zu implementieren. Die Testumgebung ist ein Beispiel dafür. Sie übernimmt die Visualisierung von Original- sowie Ergebnisbildern. Gleichzeitig dient diese Testumgebung als ein Editor für das Kombinieren der Filter. Die damit erstellte Kombination kann sofort getestet und im XML-Format abgespeichert werden.

Ilia Revout

Title of the paper

The design and realization of a framework for the image processing

Keywords

Image processing, image grabbing, framework, image filter, robotic expression, robotic control

Abstract

This thesis covers the design and realization of a framework for image processing for real time applications, eps. in the area of autonomous mobile robots. This framework supports the selection and combination of image filters for a specific domain and task. It helps as well in the design and realization of new filters and their interaction with existing ones. The filters can be configured for specific demands in an interactive way. Therefore this framework serves as a basic for rapid prototyping in the area of low level real time image processing. The integrated interface enables fast development test cycles. The filter combination for a specific task can be stored in a XML document. A runtime environment can use this configuration file within an application program.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Ein Bild sagt mehr als tausend Worte. Vielleicht deswegen ist die Bildverarbeitung nicht mehr aus unserem Leben auszudenken. Gesichtserkennung für die Zugangskontrolle oder für Terroristensuche, automatische Videoüberwachung, aber auch die Spezialeffekte bei den Digitalkameras sind nur einige Beispiele aus dem riesigen Bereich der Bildverarbeitung. Die Grundidee bleibt aber immer dieselbe, gleich, ob es sich um Effekte oder Erkennung handelt. Man verändert die vorhandenen Informationen so, dass die neu gewonnenen Daten mehr oder bessere Informationen zur Verfügung stellen.

Seit mehr als vierzig Jahren beschäftigen sich viele grosse Firmen, aber auch einzelne Personen, mit der Bearbeitung von Bildern. Inzwischen gibt es unzählige Programme, die für Endanwender die Arbeit erleichtern. Auch so genannte Frameworks[1] sind überall zu finden. MFC (Microsoft Fundation Class) ist ein bekanntes Beispiel für ein Framework.

Eine gute Übersicht über existierende Programme, Projekte und Dokumente ist auf der Internetseite [Cvh03] zu finden. Dort findet man unter anderem auch weitere Links zu Seiten mit dem für die Bildverarbeitung relevanten Inhalt.

Die grosse Anzahl von Programmen bedeutet leider nicht, dass für jede Problemstellung die passende Lösung zu finden ist. So finden wir im Bereich Roboter nur sehr speziell entwickelte Anwendungen, die schlecht an andere Probleme anzupassen sind.

[1] Ein Framework besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen.[Deu89]

1.2 Worum es hier geht

Diese Diplomarbeit beschäftigt sich mit dem Design und der Implementierung eines Frameworks, welches unter anderem für den Bereich Robotersehen angewendet werden soll.

In der Hochschule für Angewandte Wissenschaften in Hamburg konnte ich unterschiedliche Bildverarbeitungsprojekte kennen lernen. Den größten Zeitaufwand beanspruchte in diesen Projekten die Implementierung von Laufzeit- und Testumgebungen. Genau an dieser Stelle soll das Framework eingesetzt werden. Damit soll erreicht werden, dass die notwendige Plattform schon zur Verfügung steht und man die meiste Zeit für eigene Ideen investieren kann. Diese Arbeit führt den Leser in den gesamten Entwicklungsprozess des Frameworks ein. Aus den Anforderungen entsteht das Design, welches im weiteren dann implementiert und getestet wird. Das entwickelte Framework wird für die Implementierung der Testanwendungen eingesetzt. Damit wird die Tragfähigkeit des gesamten Konzepts überprüft. Viele Abschnitte sind in Form einer Anleitung aufgebaut, so dass diese Arbeit auch bei der Nutzung des Frameworks sehr hilfreich werden kann.

1.3 Übersicht

Im Kapitel 'Anforderungen' werde ich die Ziele des Programms festlegen, sowie etwas über Szenarios berichten, die für die Zielsetzung eine grundlegende Rolle gespielt haben. Im Kapitel 'Design' stelle ich das von mir für die Problemlösung entworfene Design vor. Die Diagramme, die in UML (Unified Modeling Language) definiert sind, helfen dabei, die Designlinien besser zu verstehen. Dabei werden viele Details weggelassen, um den Blick für das Wesentliche nicht zu verlieren.[2] Im nächsten Kapitel untersuche ich die Tragfähigkeit des Designs durch eine Implementierung. Die ausführlichen Anleitungen helfen dabei, die entworfenen Programme besser zu verstehen und sie bedienen zu können. Die entworfenen Module werden im Kapitel 'Evaluation' anhand mehrerer Szenarios getestet. Dabei wird auf die Untersuchung der Ausführungszeit und die Einfachheit der Nutzung ein besonderer Schwerpunkt gelegt. Mehrere Szenarios sollen verschiedene Einsatzmöglichkeiten demonstrieren. Am Ende, im Kapitel 'Resümee', wird die durchgeführte Arbeit zusammengefasst und eine mögliche Richtung der Weiterentwicklung vorgeschlagen.

[2]Das komplette Klassendiagramm finden Sie auf der beigelegten CD-ROM.

1.4 Eingetragene Warenzeichen

Windows®, VisualC++®,Video for Windows® sind eingetragene Warenzeichen von Microsoft.
QuickCam®, Logitech® sind eingetragene Warenzeichen von Logitech.

Kapitel 2

Anforderungen

2.1 Zielsetzung

2.1.1 Ausgangsszenerien

Ich habe mir die Aufgabe gestellt, ein Framework zu entwickeln, welches im Bereich der Objekterkennung anwendbar sein soll. Grundlage für das Framework waren die Szenarios aus den Diplomarbeiten von Rainer Balzerowski [Bal02] und Arne Dieckmann [Die03].

2.1.1.1 Erstes Szenario

Das erste Szenario kommt aus der Diplomarbeit von Rainer Balzerowski [Bal02]. Die Kamera ist unbeweglich befestigt und beobachtet das Feld. Auf dem Feld befinden sich Objekte, die sich farblich voneinander unterscheiden. Die Aufgabe besteht darin, diese Objekte zu erkennen und daraus die Koordinaten der Objekte zu ermitteln. Die Abbildung 2.1 demonstriert die Positionierung der Kamera sowie der Objekte. Die Objekterkennung basiert auf den Farbunterschieden der Objekte. Die Form der Objekte spielt dabei keine Rolle.

2.1.1.2 Zweites Szenario

Die Arbeit von Arne Dieckmann [Die03] liefert die Grundlage für das zweite Szenario. Der Unterschied zum ersten Szenario liegt dabei in der Position der Kamera, sowie in der Art der Objekterkennung. Die Kamera befindet sich nun auf einem Roboter. Die Objekte werden, unter anderem, anhand ihrer Konturen erkannt. Der typische Aufbau aus dem zweiten Szenario ist in der Abbildung 2.2 dargestellt.

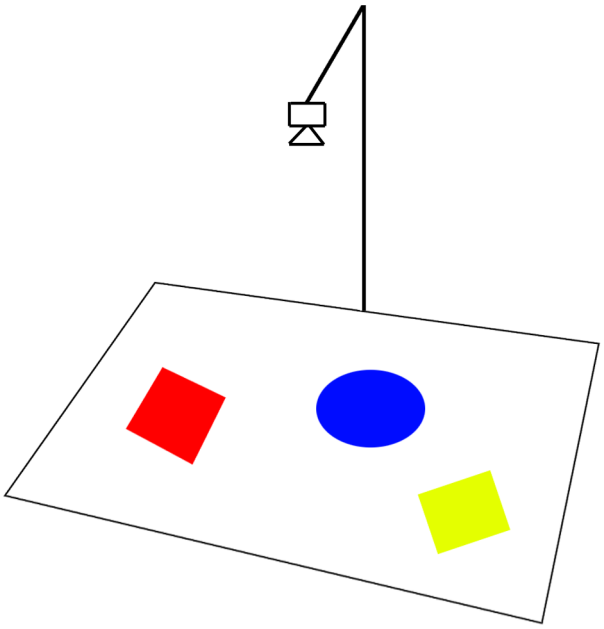

Abbildung 2.1: Erstes Szenario

2.1.2 Erstes versus zweites Szenario

Obwohl beide Szenarios sehr ähnlich aussehen, haben sie gravierende Unterschiede. Die gleichen Aufgaben müssen zum Teil ganz anders gelöst werden. Angenommen die Aufgabe lautet, die Richtung des beweglichen Objektes zu ermitteln. Im ersten Szenario stellt das kein Problem dar. Das Hintergrundbild ändert sich nicht. Es ist relativ einfach zu erkennen, dass sich ein bestimmtes Objekt bewegt hat. Anders sieht es bei dem zweiten Szenario aus. Hier wissen wir nicht, ob sich das Objekt oder die Kamera bewegt haben, oder beide.

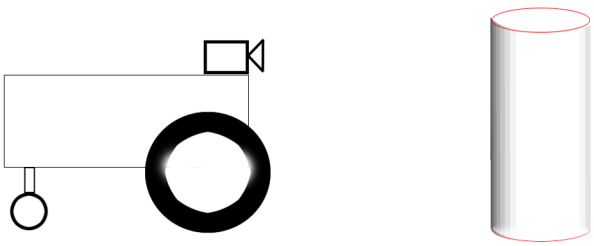

Abbildung 2.2: Zweites Szenario

An diesem Beispiel möchte ich andeuten, dass für die Objekterkennung sehr unterschiedliche Verfahren angewendet werden. Das Framework soll dabei keine konkrete Lösung für diese Szenarios liefern, sondern eine allgemeine Plattform zur Verfügung stellen, mit deren Hilfe solche, aber auch andere Problemstellungen leichter und schneller zu bewerkstelligen sind.

2.1.3 Frameworksszenarios

Die oben beschriebenen Szenarios dienen als Grundlage für den Entwurf. Die typische Anwendung des Frameworks möchte ich anhand von weiteren Szenarios beschreiben.

Es wird eine Aufgabe gestellt, die mit Hilfe des Roboters gelöst werden muss. Die Steuerung des Roboters hängt von der Information ab, die von einer Videokamera zur Verfügung gestellt wird. Dabei spielt es keine Rolle, ob die Kamera stationär oder auf dem Roboter platziert ist. Meistens hat man relativ schnell die ersten Ideen, wie man das Problem lösen kann, bzw. einen Ansatz zur Lösung. Leider ist es nicht trivial, in diesem Bereich einen Test durchzuführen. Es sind ziemlich viele Details, ohne die man nicht weiter kommt:

- Wie kann man eine Kamera anschließen?
- Wie kann man das Bild ins Programm einbinden?
- Wie greift man auf die Bilddaten zu?
- Wie stellt man das Ergebnis auf dem Bildschirm dar?

Sehr schnell wird deutlich, dass aus dem schnellen Ausprobieren leider nichts wird. Genau an der Stelle soll dann das Framework zum Einsatz kommen. Auf alle oben gestellte Fragen kann nun gleich geantwortet werden: das Framework kann es. Das Framework stellt eine fertige Umgebung dar, welche die oben genannten Aufgaben bewerkstelligen kann. Unterschiedliche Komponenten des Frameworks teilen die Aufgaben unter sich auf. So übernimmt die Framework-GUI - Komponente unter anderem das Anzeigen von Original- und Ergebnisbildern, während die andere Komponente (das Kernel) die für die Bearbeitung notwendige Funktionalität zur Verfügung stellt.

Nachdem die Lösungsansätze mit Hilfe der Frameworkanwendung getestet worden sind, kann man mit der konkreten Implementierung anfangen. Damit meine ich die Endanwendung, die man dann auch einsetzen möchte. Auch hierbei soll das Framework die Arbeit erleichtern. Die daraus resultierende Anforderung ist die saubere Trennung allen Frameworkkomponenten, so dass

das Kernel unabhängig z.B. von der GUI-Komponente eingesetzt werden könnte.

2.2 Anforderungen an das Framework

Aus den oben beschriebenen Szenarios resultiert die folgende Liste mit den wichtigsten Anforderungen an das Framework:

1. Das Framework soll das Imagegrabbing durchführen.
2. Es soll möglich sein, verschiedene Filter miteinander zu kombinieren, sowie das Framework um neue Filter erweitern zu können.
3. Das Testen der Filter einzeln, sowie in Kombination, soll erleichtert werden.
4. Es soll möglich sein, das Framework in eine beliebige Anwendung einbinden zu können.
5. Die zusammengestellten Filter sollen gespeichert sowie geladen werden können.

2.3 Randbedingungen

2.3.1 Software

Der Haupteinsatzort soll das Roboterlabor der Hochschule für Angewandte Wissenschaften Hamburg sein.[Rlb03] Zur Zeit werden hier die meisten Programme mit Visual C++ unter Windows entwickelt. Dieser Fakt dient als Voraussetzung zur Implementierung des Frameworks. Das Framework soll unter Windows lauffähig sein, sowie mit Hilfe von Visual C++ erweiterbar sein.

2.3.2 Hardware

Die Hardwareanforderungen beziehen sich auf die Kamera und den Computer. Die Funktionalität des Frameworks soll nicht von den Rechnern bzw. Kameras beeinflusst werden. So soll es kein Problem darstellen, die handelsüblichen Geräte einsetzen zu können.

2.3.2.1 Kamera

Es ist prinzipiell gleich, welche Kamera eingesetzt wird. Zur Zeit sind USB (Universal Serial Bus) Kameras sehr verbreitet. Dafür bieten sich besonders die so genannten 'Webcams' an. In der Abbildung 2.3 ist als Beispiel eine Webcam der Firma 'Logitech' dargestellt.[1] Reicht die Qualität nicht

Abbildung 2.3: Logitech Webcam

mehr aus, dann soll es auch möglich sein, auf etwas teure Technik, wie IEEE1394 ('FireWire'), umzusteigen. Die dafür notwendigen Kameras können beispielsweise wie in der Abbildung 2.4 aussehen. Die Abbildung 2.5 zeigt

Abbildung 2.4: Sony DFW-V500

den maximalen Datentransfer der oben genannten Schnittstellen.

[1] Diese Kamera wurde bei der Entwicklung des Frameworks eingesetzt.

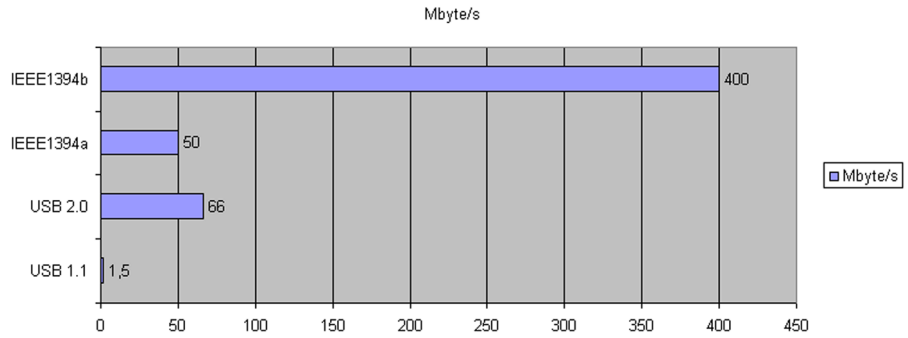

Abbildung 2.5: Geschwindigkeit der Schnittstellen

2.3.2.2 Computer

Minimale Voraussetzungen für den Rechner bestimmen die Softwarepakete, wie das Betriebssystem und die Entwicklungsumgebung, sowie externe Hardware wie die Kamera. Das Windows OS sowie die Entwicklungsumgebung Visual C++ müssen auf dem Rechner lauffähig sein. Dabei soll als wichtige Anforderung gelten, dass das Framework auf dem handelsüblichen Computer (PC oder Laptop) nicht nur ausführbar sein muss, sondern auch die Ergebnisse in akzeptabler[2] Zeit liefern soll. Damit wird es möglich sein, das Framework preisgünstig einsetzen zu können.

[2]Für die meisten Aufgaben im Robotlabor der HAW-Hamburg sind 5-10 f/s ausreichend.

Kapitel 3

Das Design

In diesem Kapitel beschreibe ich die Designrichtlinien des Frameworks. Angefangen von groben Skizzen werde ich jede einzelne Komponente detaliert beschreiben. Die Problemstellung zu jedem Modul soll dabei helfen, die Funktionalität besser zu verstehen.

3.1 Der Aufbau

Die Abbildung 3.1 demonstriert einen groben Aufbau des Frameworks. Die Trennung zwischen dem Kernel und der GUI möchte ich extra betonen und etwas näher erklären. Obwohl das Framework absolut unabhängig von der GUI ist, spielt die GUI (später als 'Testumgebung' bezeichnet) eine bedeutende Rolle. Mit der Testumgebung soll es möglich sein, nicht nur neue Filter zu testen, sondern auch die Kombifilter visuell zusammensetzen und konfigurieren zu können. Damit übernimmt die Testumgebung auch die Rolle des Kombifiltereditors. Die wichtigsten Aufgaben der GUI sind also:

- Grabberinformation (Originalbilder von der Kamera) anzeigen;
- vorhandene Filter auswählen, um sie beliebig zu kombinieren;
- das Ergebnis der Filter anzeigen;
- die Funktionalität des Kernels (wie 'speichern' und 'laden' von Kombifiltern) anbieten.

Mit Hilfe der GUI (der Testumgebung) soll es also möglich sein, eine bestimmte Filterkombination zu erstellen und das Ergebnis sofort visuell zu testen.

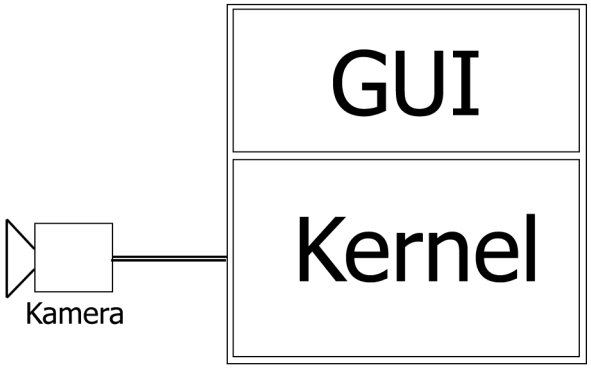

Abbildung 3.1: Das Framework: GUI und Kernel

3.1.1 Die Trennung zwischen dem Kernel und der GUI

Die Trennung zwischen der GUI und dem Kernel wird anhand eines Szenarios erläutert. Gleichzeitig wird eine der Framework-Funktionalitäten beschrieben, da es am besten unter diesem Kontext zu erklären ist. Das Framework ist in der Lage, kombinierte Filter abzuspeichern sowie zu laden. Das mögliche Szenario kann z.B. so aussehen: Mit Hilfe der Framework-GUI wird eine Filterkombination (ein Kombifilter) zusammengestellt und getestet. Dieser Kombifilter wird abgespeichert. Wichtig ist, dass nicht nur die Reihenfolge von Filtern, sondern auch alle Einstellungen persistent gemacht werden, so dass nach dem Neuladen des Kombifilters das Ergebnis das selbe ist. Nun kann dieser Kombifilter wieder geladen werden. Dabei wird er nicht in die Testumgebung (Frameworkkernel mit der GUI) geladen, sondern in eine andere, auf dem Frameworkkernel basierende Umgebung, z.B. die Robotersteuerung. Die Abbildung 3.2 demonstriert dieses Szenario.

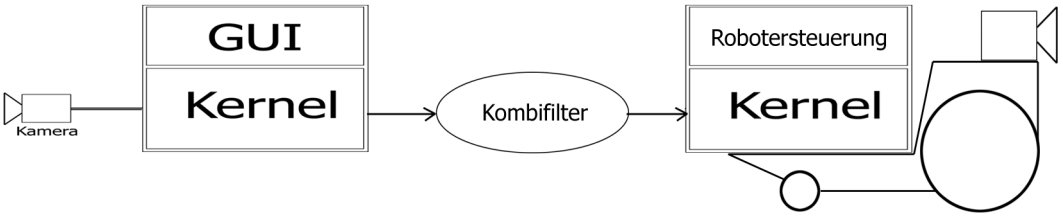

Abbildung 3.2: Das Framework: Speichern/Laden - Szenario

3.2 Das Filterkonzept

Die Hauptaufgabe der Filter besteht darin, die Bilddaten zu verändern, bzw. einige Informationen daraus zu gewinnen. Trotzdem können Filter auch für andere Zwecke eingesetzt werden, aber mehr darüber im Kapitel 'Realisierung'. Allgemein stellt das Framework das Filterkonzept zur Verfügung, so dass schon fertige Filterbibliotheken leicht eingebunden werden können. Aber auch die Neuentwicklung der Filter wird sehr vereinfacht, so dass sie sehr schnell zu implementieren sind.[1] Das Klassendiagramm 3.3 zeigt die Zusammensetzung der Klassen. 'CamFramImageObject' repräsentiert die Bildinformation und stellt die notwendigsten Informationen wie die Breite und die Höhe des Bildes zur Verfügung, sowie Operationen, die den Zugriff auf jeden Punkt des Bildes mit Hilfe von x / y - Koordinaten ermöglichen. Jeder Fil-

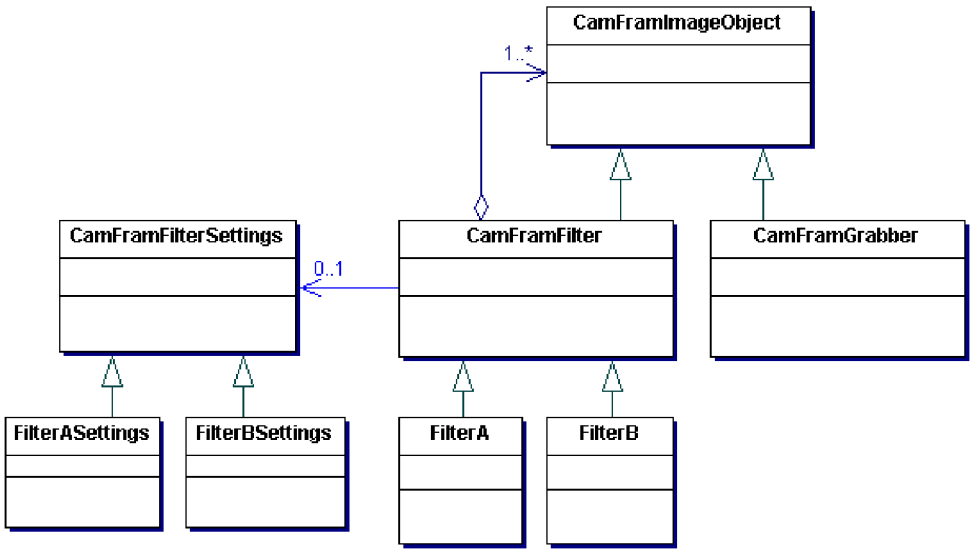

Abbildung 3.3: Filter-Einstellungen

ter kann eine Instanz von der Einstellungsklasse und muss mindestens eine vom Typ 'CamFramImageObject' als Quelle besitzen. Da die Filter selbst von diesem Typ sind, können sie auch als Quelle verwendet werden, so dass ein Filterbaum gebaut werden kann. In der Abbildung 3.4 ist ein Beispiel von einem solchen Filterbaum zu sehen. Jeder Knoten des Baumes kennt seine Väter, aber nicht seine Kinder. Wie es aus der Abbildung 3.4 zu sehen ist, kann als Quelle auch der Grabber verwendet werden. Der Grabber

[1] Ausführlich erfahren Sie über die Implementierung der Filter im Kapitel 'Realisierung'.

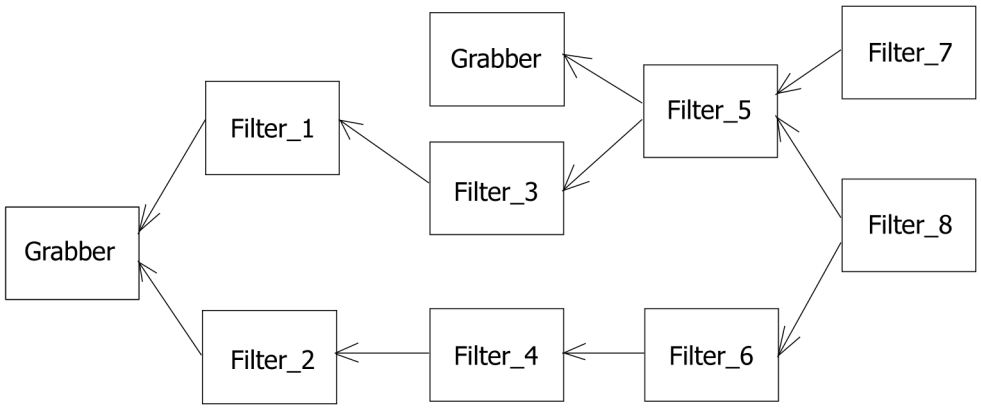

Abbildung 3.4: Filterbaum

ist vom Typ 'CamFramImageObject' (Das Klassendiagramm 3.3) , aber ist kein Filter und braucht deswegen selbst keine Quelle. Der Grabber wird vom Framework zur Verfügung gestellt und kann bei der Zusammensetzung des Filterbaumes an jeder Stelle eingesetzt werden.

Die Tatsache, dass jeder Filter auch ein Bild repräsentiert, erlaubt, das Ergebnis in jedem einzelnen Knoten anzuschauen, so dass jeder Schritt kontrolliert und getestet werden kann, nachdem der Baum schon fertig gestellt ist. Diese Funktionalität wird von der Testumgebung benutzt.

Jeder Filter im Baum kann als Endergebnis ausgewählt werden. Damit wird es möglich, verschiedene Filterkombinationen zusammenzustellen und erst danach zu entscheiden, welche Kombination eingesetzt wird. Dabei wird klar, dass nicht jeder Filter die Informationen aller anderen Filter benutzt, wie z.B. die Filter sieben und acht in der Abbildung 3.4. Dementsprechend sollen nicht alle Filter des Baumes beim Einsatz eines bestimmten Filters ausgeführt werden, um nicht unnötig CPU-Zeit zu verbrauchen. Wenn z.B. der Filter 7 ausgewählt wird, verbrauchen die Filter 2,4,6 und 8 unnötig CPU-Zeit, da diese für den Filter 7 keine Rolle spielen. Dieses Problem wurde mit dem Konzept 'Aktiver Filter' gelöst. Die Filter werden nur dann ausgeführt, wenn sie für das Ergebnis notwendig sind. Wird z.B. der Filter 3 ausgewählt, werden nur 2 Filter (Filter 1 und Filter 3) ausgeführt.

3.3 Die Framework-GUI

Die Framework-GUI stellt nicht nur die grafische Oberfläche (GUI) dar, sondern bietet ein Konzept für die Erweiterung des Frameworks an. Im weite-

ren wird die folgende Frage beantwortet: Welche Komponenten gehören zur Framework-GUI?
Die Testumgebung besteht aus mehreren Dialogen[2]. Ich unterscheide zwischen zwei Dialog-Typen:

- Dialoge für die Testumgebung, mit deren Hilfe die Filter ausgewählt und getestet werden können;
- Dialoge, die zu den Filtern selbst zählen und damit die Funktionalität des Frameworks erweitern. Diese Dialoge werden von den Entwicklern programmiert, die das Framework nutzen. Genau für diesen Typ der Dialoge wurde von mir ein Konzept entwickelt, das die Programmierung und das Einbinden unterstützt.

3.3.1 Die Oberfläche der Frameworktestumgebung

Ein Dialog aus dem ersten Typ stellt die Oberfläche der Frameworktestumgebung dar. Dieser Dialog (der Hauptdialog) übernimmt die Anzeige der Bilddaten, sowie die Steuerung des Kernels. Damit vereint der Hauptdialog das View und den Controller aus dem Model-View-Controller Pattern. Die Abbildung 3.5 zeigt, vereinfacht anhand eines Sequenzdiagramms, die Kommunikation zwischen dem Hauptdialog und dem Kernel.

In der Frameworktestumgebung spielt der Hauptdialog die Anwendungsrolle. Spezifisch für jede Anwendung, die den Kernel nutzt, ist die 'eventHandling()' - Methode. Damit das Vorkommen dieser Methode in jeder Anwendung gewährleistet werden kann, muss die Anwendungsklasse, die mit dem Kernel kommunizieren soll, von der Frameworkklasse 'CamFramReceiver'[3] abgeleitet werden. Der Kernel hält eine Instanz der CamFramReceiver-Klasse. Damit ist es möglich, dem Kernel eine Mitteilung an eingetragene Anwendungen zu versenden, ohne die Anwendungen tatsächlich zu kennen.[4] Das Klassendiagramm 3.6 zeigt die Zusammensetzung der Komponenten.[5]

[2]Ein Dialog ist ein Fenster, das Nachrichten empfängt, verschoben und geschlossen werden kann. Die Anweisungen können zum Zeichnen in seinem Client-Bereich verarbeitet werden. [Kru98]

[3]Die Namensgebung werde ich im weiteren ausführlicher beschreiben.

[4]Es ist auch möglich, mehrere Anwendungen einzutragen. Dieses Konzept wird im Kapitel 'Eventverarbeitung' näher beschrieben.

[5]Die Klassenbezeichnung ist frei ausgewählt und stimmt mit den tatsächlichen Bezeichnungen nicht überein.

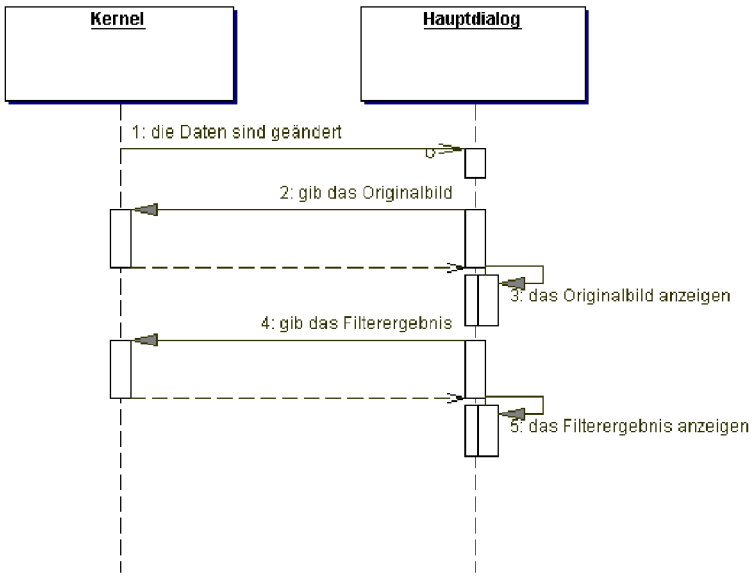

Abbildung 3.5: Das Sequenzdiagramm: Kernel - Hauptdialog

3.3.2 Das Konzept der Frameworkerweiterungsdialoge

Unter Frameworkerweiterungsdialogen sind alle Dialoge gemeint, mit deren Hilfe die Einstellungen für Filter geändert werden können. Jeder Filter, der mit Hilfe des Frameworks entwickelt oder eingebunden wird, kann auch veränderbare Einstellungen haben, die zur Laufzeit verändert werden können. Um die Möglichkeit zu haben, diese Veränderungen in der Testumgebung vornehmen zu können, sind die Dialoge notwendig. Wie diese Dialoge im Framework verwaltet werden, wird hier beschrieben.

Die neu entwickelten Einstellungsdialoge (Instanzen) werden zusammen mit dem Filter, für den die Einstellungen verändert werden müssen, im Kernel registriert. Zur Laufzeit existiert immer die gleiche Anzahl der Dialoginstanzen, unabhängig von der Anzahl der erstellten Filterinstanzen. Sie werden in einem speziellen Dialogpool verwaltet. Dieser Pool ist auch für die Anzeige des passenden Dialoges für die beliebigen Filter zuständig. Wenn die Einstellungen verändert werden müssen, wird zuerst in der Testumgebung der betroffene Filtername ausgewählt. Der Kernel liefert dann eine tatsächliche Instanz des dem Namen entsprechenden Filters. Diese Filterinstanz wird an den Dialogpool übergeben. Der Dialogpool überprüft, ob für die Klasse, von der diese Filterinstanz ist, eine Dialoginstanz existiert. Wenn eine Dialoginstanz gefunden wird, wird dieser Dialog mit den Einstellungen des Filters initialisiert und angezeigt. Das Sequenzdiagramm 3.7 verdeutlicht die-

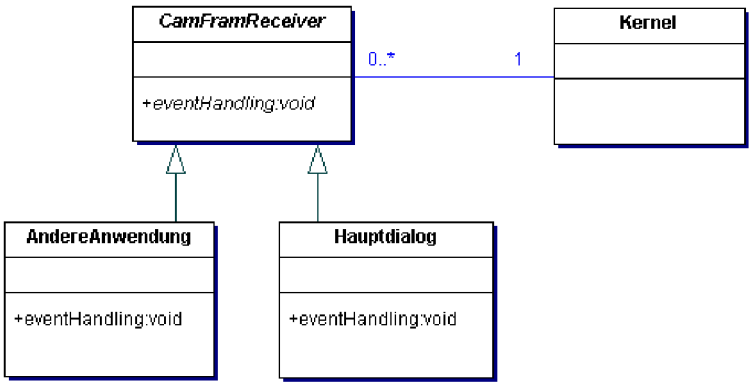

Abbildung 3.6: Kernel - Receiver

ses Verhalten. Um die Entwicklung dieser Dialoge zu erleichtern, wird vom

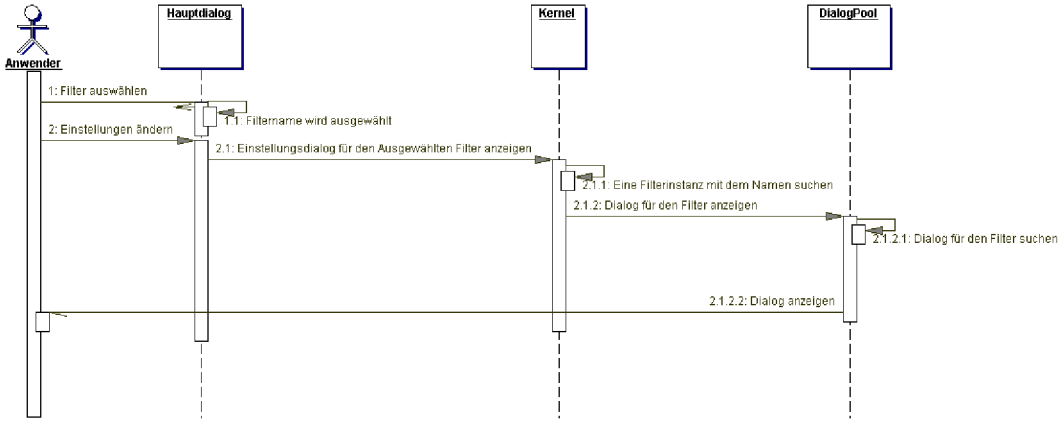

Abbildung 3.7: Dialogpool

Framework eine Klasse angeboten (CamFramFilterSettingsDialog), von der alle Dialoge abgeleitet werden müssen.

3.4 Der Framework-Kernel

Ähnlich wie bei der GUI, wird auch der Frameworkkernel als zwei getrennte Teile betrachtet. Der erste Teil enthält die funktionalen Klassen. Sie decken die gesamte Funktionalität des Frameworks ab. Der zweite Teil besteht aus den für die Framework-Erweiterungen notwendigen Klassen. Im weiteren werden diese beiden Teile ausführlicher beschrieben.

3.4.1 Die Richtlinien des Frameworkkernels

Die Hauptaufgabe des Frameworks ist die Unterstützung beim Kombinieren der Filter, sowie bei deren Test. Die Neuimplementierung oder die Einbindung von fertigen Filtern ins Framework muss nach bestimmten Regeln durchgeführt werden. Für diesen Zweck werden vom Framework drei Klassen zur Verfügung gestellt. Eine davon (CamFramFilterSettingsDialog), die für die Dialoge zuständig ist, wurde bereits erwähnt. Weitere zwei werden hier vorgestellt.

Im Framework werden diverse Funktionalitäten automatisch ausgeführt. Es werden unter anderem neue Filterinstanzen erzeugt, Dialoge werden mit passenden Einstellungen initialisiert, Filter werden automatisch aus- und eingeschaltet. Das alles erfordert, dass alle Komponenten, die für das Framework entwickelt werden, bestimmte Richtlinien einhalten. Genau diesen Zwecken dienen die beiden Frameworkklassen: 'CamFramFilter' und 'CamFramFilterSettings'.[6]

3.4.1.1 Die Klasse 'CamFramFilter'

Die Klasse 'CamFramFilter' beschreibt die Zusammenhänge zwischen den betroffenen Klassen und schreibt die Methoden vor, die in den Unterklassen implementiert werden müssen. So ist z.B. festgelegt, dass jeder Filter eigene Einstellungen besitzen kann. Zusätzlich werden für die Unterklassen zwei Makros zur Verfügung gestellt (DECLARE und IMPLEMENT), die die Deklaration und die Implementierung erheblich erleichtern. Die Abbildung 3.8 zeigt den Aufbau der Klasse 'CamFramFilter'. Diese Klasse stellt eine Funktionalität bereit, die folgendermaßen gegliedert werden kann:[7]

- die Erzeugung
  - 'createFilter'-Methode erlaubt die Erstellung einer neuen Instanz eines Filters.
- die Verwaltung
  - Mit Hilfe dieser Methoden ist es möglich, bestimmte Quellresourcen (Parents) anzusprechen und damit auf deren Daten zuzugreifen. Es sind unter anderem die Methoden 'getImageSource' und 'getImageSourceAt'.

[6] Wie man einen Filter mit den Einstellungen und einem Dialog programmieren kann, wird ausführlich im Kapitel 'Realisierung' beschrieben. Hier geht es nur um das Design des Frameworks.

[7] Die detaillierte Beschreibung der Methoden finden Sie im Kapitel 'Evaluation'. Der gesamte Quellcode befindet sich auf der zur Arbeit gehörenden CD-ROM.

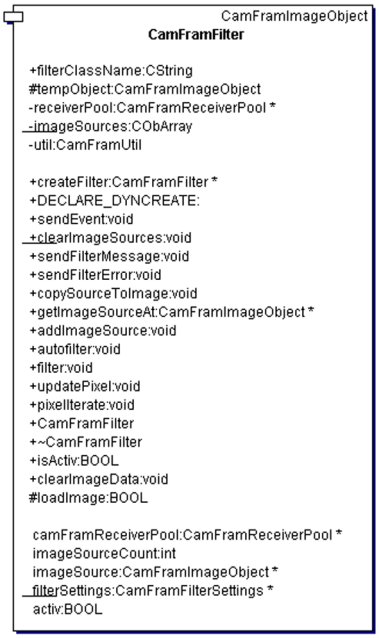

Abbildung 3.8: CamFramFilter

- die Kommunikation
  - Die Methoden 'sendFilterMessage' und 'sendFilterError' sind für die Kommunikation zwischen dem Filter und der Anwendung zuständig. Damit ist jeder Filter in der Lage, bestimmte Mitteilungen oder Fehlermeldungen der Anwendung mitzuteilen.

3.4.1.2 Die Klasse 'CamFramFilterSettings'

Wie schon im Abschnitt 3.4.1.1 erwähnt wurde, kann jeder Filter eigene veränderbare Einstellungen besitzen. Mit den Einstellungen sind in diesem Kontext Werte gemeint, die in den Filtern wieder ausgelesen werden und für den Filterablauf notwendig sind. Als Beispiel kann der Schwarz/Weiß-Filter dienen. Die Aufgabe dieses Filters ist es alle im Bild vorkommenden Farben auf zwei (schwarz und weiß) zu reduzieren. Davon ausgehend, dass es sich um ein Graustufenbild handelt, haben wir 256 unterschiedliche Graustufen, wobei 0 - schwarz und 255 - weiß sind. Der Filter untersucht jeden einzelnen Punkt des Bildes und überprüft, ob der Wert größer oder kleiner/gleich eines bestimmten Grenzwertes ist. Wenn kleiner, dann wird die Farbe durch schwarz, sonst durch weiß ersetzt. Die entscheidene Frage ist nun, wie gross dieser Grenzwert ist. Damit man in der Lage ist, diesen Wert während der

Laufzeit zu verändern, wird der Grenzwert als eine Variable gehalten. Diese Variable wird in dieser Arbeit als Einstellung des Filters genannt und in der Einstellungsklasse gehalten.

Die Klasse 'CamFramFilterSettings' wurde generisch entworfen, so dass die meisten Einstellungen damit abgebildet werden können. Die Abbildung 3.9 zeigt den Aufbau der Klasse. Diese Klasse repräsentiert ein Map, wobei als

Abbildung 3.9: CamFramFilterSettings

Schlüssel ein beliebiges String verwendet werden kann und als Value der Wert der Einstellung. Es werden zur Zeit sieben am der meisten verwendete Werttypen unterstützt: BOOLEAN, BYTE, int, long, double, float und CString. Für jeden Typ existieren get- und set- Methoden. Die Set-Methode erwartet zwei Parameter: key als CString und value abhängig vom Einstellungstyp. Die Get-Methode erwartet nur den Schlüssel und liefert entsprechend den Wert. Die Abbildung 3.10 demonstriert einen typischen Ablauf der Veränderung von Einstellungen. Die Auswahl der Schlüssel ist zwar frei, es ist aber zu empfehlen, folgendes Muster anzuwenden: Filtername '_' Bezeichnung der Einstellung.

Dieses Konzept erlaubt mehrere Szenarios:

- Jeder Filter hält eine Instanz der 'CamFramFilterSettings' - Klasse.

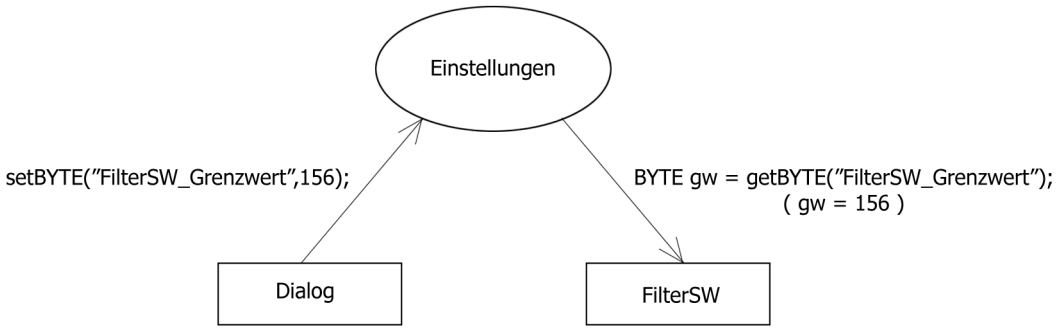

Abbildung 3.10: Dialog - Einstellungen - Filter

- Für jeden Filter wird zusätzlich eine eigene Einstellungsklasse implementiert, die von der Klasse 'CamFramFilterSettings' erbt, aber intern alle Werte zusätzlich hält. Jeder Filter hält entsprechend eine Instanz der eigenen Einstellungsklasse.

Die erste Variante ist ein Standardfall und ist optimal für die Testphase. Der Nachteil liegt darin, dass bei jedem Filterausführungsvorgang auf eine grosse Collection zugegriffen werden muss, was CPU-Zeit kostet. Genau für diesen Zweck wird die zweite Variante angewendet. Die Abbildung 3.11 demonstriert dieses Szenario. Wie man aus der Abbildung 3.11 sehen kann, muss die

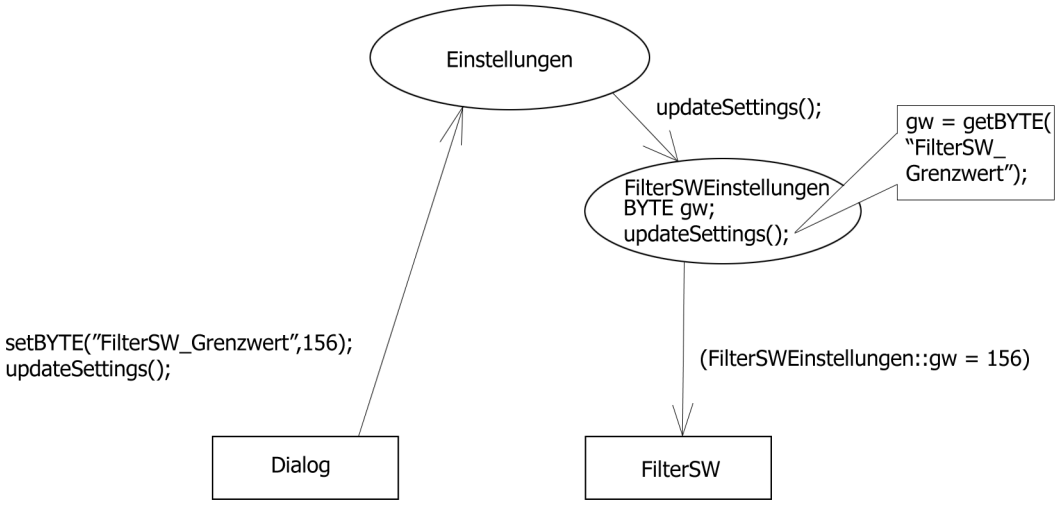

Abbildung 3.11: Dialog - Filtereinstellungen - Filter

neu implementierte Klasse die Methode 'updateSettings()' überschreiben. Bei

dem Dialog muss diese Methode lediglich aufgerufen werden. Diese Variante beschleunigt die Filterausführung und wird auch von mir angewandt.

3.4.2 Die Frameworkfunktionalität

Wenn der, im Abschnitt 3.4.1, beschriebene Teil für die Frameworkerweiterung zuständig war, ist dieser Teil des Frameworks das Herz der Anwendung. Die gesamte Funktionalität des Frameworks wird durch viele Klassen bestimmt. Damit der Nutzen des Frameworks aber übersichtlich bleibt, wurde das Fassaden-Pattern angewendet, so dass das Framework nach außen als eine Klasse ('CamFramework') vertreten ist. Da in dieser Klasse fast alle Auf-

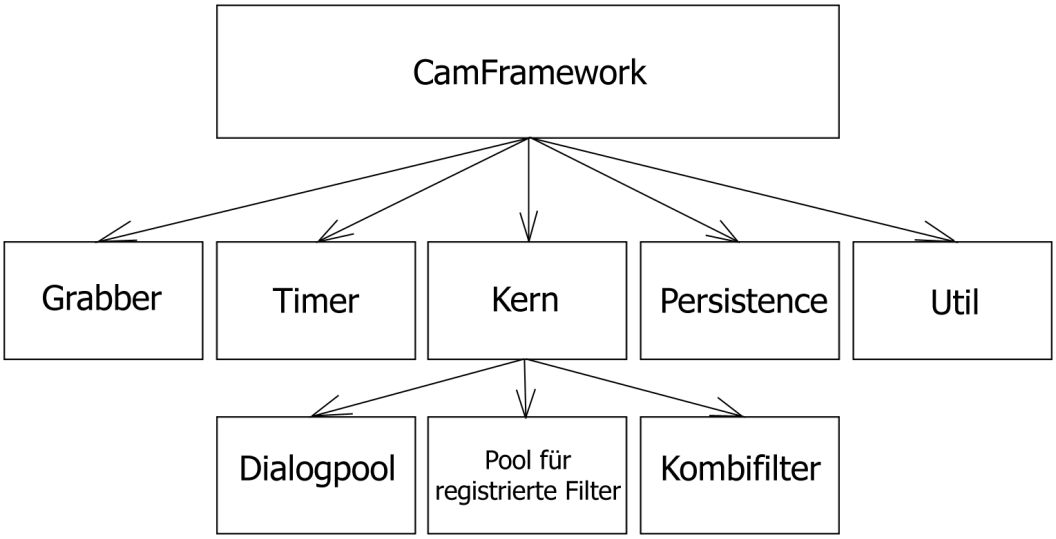

Abbildung 3.12: Komponentenuebersicht

gaben nur weiter delegiert werden, werden sie hier nicht weiter beschreiben, ich verweise an dieser Stelle an den Anhang, in dem das gesamte Klassendiagramm aller Module zu finden ist. Die Abbildung 3.12 zeigt die wichtigsten Komponenten des Frameworks.

3.4.2.1 Der Grabber

Wie in der Abbildung 3.3 schon zu sehen war, ist die Grabber-Klasse ('CamFramGrabber') auch von der Klasse 'CamFramImageObject' abgeleitet und repräsentiert damit auch das Bild. Diese Klasse ist abstract und schreibt für alle Unterklassen eine Methode ('loadImageFromCam') vor, die dafür sorgen soll, dass die Bilddaten gefüllt werden.

Die Testumgebung selbst implementiert eine konkrete Klasse ('Image-Grabber'). Diese Klasse ist für die Video for Windows (VfW)- Videoquellen zuständig. Damit ist es möglich, eine beliebige Videoquelle anzusprechen und die Bilder zu holen, sobald diese Quelle das VfW-Standart unterstützt. Dabei ist eine Voraussetzung zu beachten: das Format der Bilder. Dieses Format hängt vom Videoquelle-Treiber ab. Da das gesamte Framework mit dem Format RGB24 funktioniert, muss die konkrete Grabberklasse dafür sorgen, dass dieses Format für das Framework zur Verfügung gestellt wird. Die Abbildung 3.13 zeigt einen theoretischen Aufbau eines konkreten Grabbers. Die konkre-

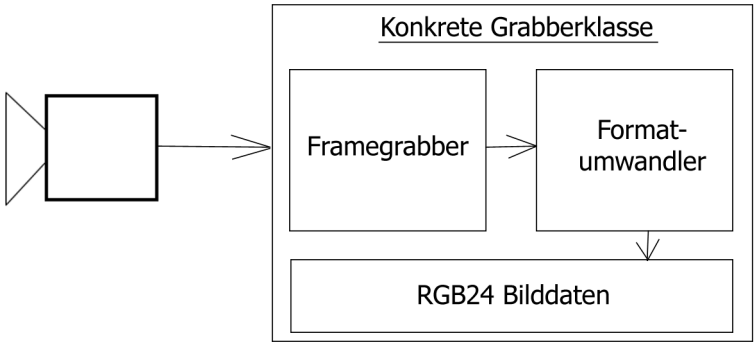

Abbildung 3.13: Aufbau einer konkreten Grabberklasse

te Grabberklasse der Testumgebung implementiert die Formatumwandlung nicht und deswegen ist zu beachten, dass der Treiber der Videoquelle selbst das RGB24 - Format liefern muss.

Allgemein spielt es für das Framework keine Rolle, wo sich die Bildinformation befindet und wie die Daten geholt werden. Theoretisch ist auch folgendes Szenario möglich. Eine Webcam speichert ständig die aktuellen Bilder ab. Via Internet besteht die Möglichkeit, auf diese Datei zuzugreifen. Dann ist es einfach, eine Grabberklasse zu schreiben (z.B. die 'WebCam-Grabber'), die uns die Daten zur Verfügung stellt. Diese Klasse registrieren wir im Framework und schon läuft alles wie gehabt, mit der Ausnahme, dass die Kamera nun vielleicht 20000 km entfernt ist und die Ausführung etwas langsamer verlaufen wird.

3.4.2.2 Der Timer

Mit dem Timer ist ein Thread gemeint, dessen Aufgabe die ständige Ausführung des Kernes ist. Im Framework wird dieses Thread mit der Klasse CamFramThread repräsentiert. Diese Klasse ist sehr einfach aufgebaut. Die einzige Methode 'Run' erledigt die gesamte Threadarbeit.

```
int CamFramThread::Run(){
    while(1){
        if(activ){
            isActiv = TRUE;
            if(kernel!=NULL){
                kernel->exec();
            }
        }
            isActiv = FALSE;
    }
return 1;
}
```

Abbildung 3.14: Run - Methode

In dem Quellcode 3.14 wird deutlich, dass die Methode exec() ständig ausgeführt wird. Mit Hilfe von einer boolischen Variable ist es von außen möglich, die Ausführung zu steuern, bzw. den momentanen Status abzufragen. Aus dem Sequenzdiagramm 3.15 sind die Zusammenhänge zwischen dem

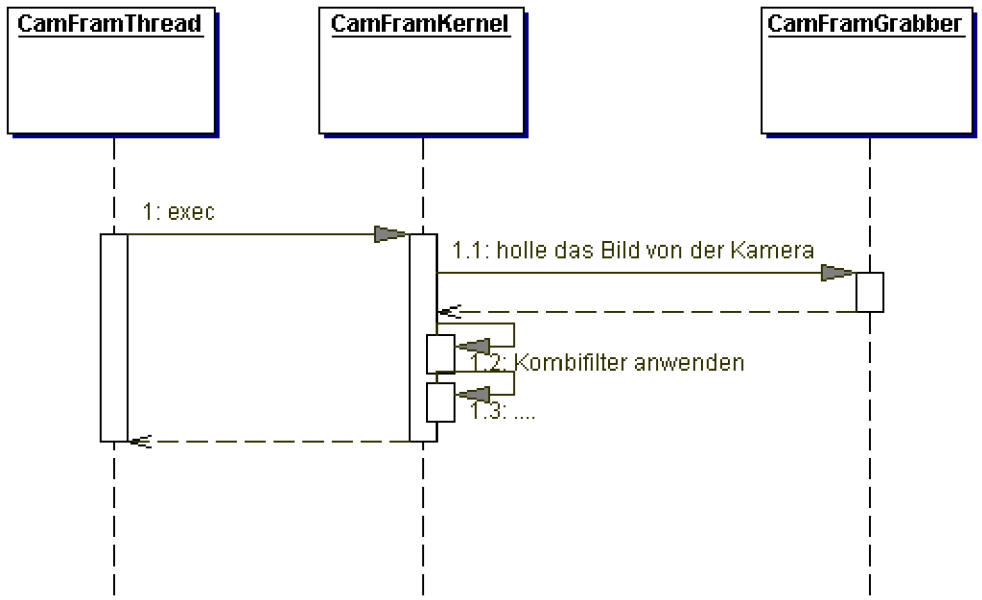

Abbildung 3.15: Thread-Kernel-Grabber

Thread und dem Kernel, sowie zwischen dem Kernel und dem Grabber zu sehen. Diese Art der Kommunikation hat den Vorteil, dass der Grabber nur dann angesprochen wird, wenn es tatsächlich notwendig ist. Es gibt keine Verzögerung und die Ausführungszeit hängt direkt von der Rechnerleistung ab. Alternativ ist es möglich, dass der Grabber unabhängig vom Kernel die Bilder der Kamera permanent ausliest und zur Verfügung stellt. Der Kernel

könnte dann auf das schon ausgelesene Bild zugreifen. Abhängig davon, wie lange es dauert ein Bild zu holen, können unterschiedliche Strategien eingesetzt werden. In diesem Fall enthält diese Methode mehr Nachteile, weil die Zeit, die der Grabber für das Holen des Bildes von der Kamera braucht, sehr kurz ist und CPU-Zeit unnötig verbraucht wird.

Eine Instanz dieser Klasse wird automatisch in der Klasse CamFramework erstellt und das Thread wird sofort gestartet. Die Ausführung beginnt aber erst nachdem die Variable 'activ' von Außen auf true gesetzt wird.

3.4.2.3 Der Kern

Diese Komponente ist der zentrale Teil des Kernels, repräsentiert im Framework durch die Klasse 'CamFramKernel'. Die Zusammensetzung der Klassen, die zum Kern gehören, wird mit Hilfe des Klassendiagramms 3.16 dargestellt. In der Klasse 'CamFramKernel' werden unter anderem die Methoden zur

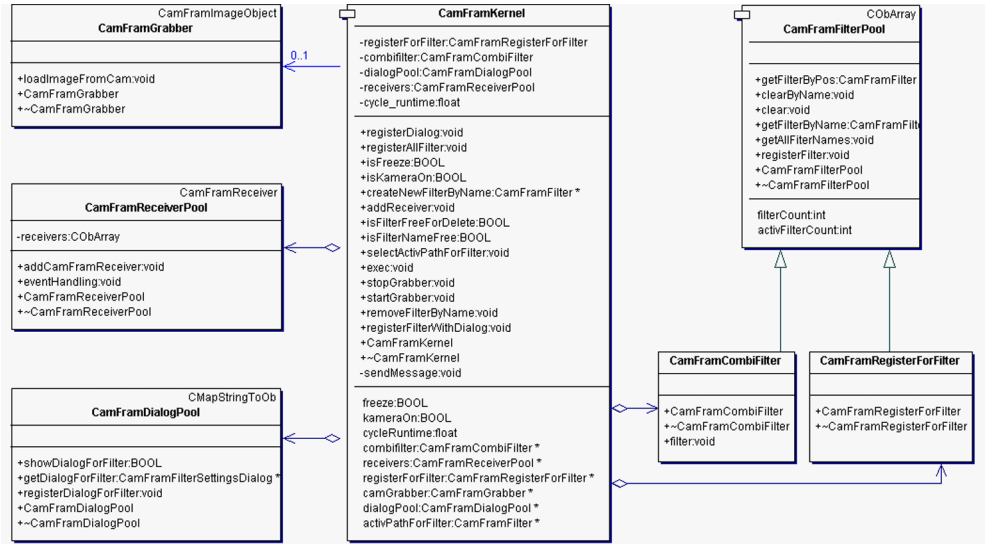

Abbildung 3.16: Kern

Verfügung gestellt, die für die Steuerung der Ausführung zuständig sind, aber auch alle möglichen Register-Methoden, deren Aufgaben darin bestehen, die Komponenten wie Filter und Dialoge zu registrieren und zu verwalten. Die exec()-Methode wurde schon im Zusammenhang mit dem Timer erwähnt. Diese Methode repräsentiert einen Durchlauf vom Grabber bis zum von Filtern gelieferten Ergebnis. Dabei werden die Zusatzinformationen, wie die für die Filterausführung gebrauchte Zeit, ermittelt und mit Hilfe von weiteren,

hier nicht beschriebenen, Methoden zur Verfügung gestellt. Die Verwaltung von in Abbildung 3.16 erwähnten Komponenten geschieht nicht direkt in dieser Klasse, sondern wird an die entsprechenden Klassen delegiert. Für alle Filterdialoge steht die Klasse 'CamFramDialogPool', für alle registrierten Filter die Klasse 'CamFramRegisterForFilter' und für den Kombifilter die Klasse 'CamFramCombiFilter' zur Verfügung. Im weiteren werden diese Klassen ausführlicher erläutert.

3.4.2.3.1 Die Klasse CamFramDialogPool Die Funktionalität dieser Klasse wurde bereits erwähnt. Alle zu den Filtern gehörigen Filterdialoge müssen im Framework registriert und verwaltet werden. Genau diesen Zweck erfüllt die Klasse 'CamFramDialogPool'. Nachdem der Dialog registriert wird, wird er in einem Map verwaltet. Als Schlüssel dient dabei der Name der Filterklasse. Damit wird erreicht, dass für alle Filterinstanzen gleicher Klasse nur eine Dialoginstanz existiert und benutzt wird. Nun ist es möglich eine passende Dialoginstanz zu holen. Dafür wird nur der Name der Filterklasse benötigt. Spezielle Funktionalität bietet die Methode 'showDialogForFilter(CamFramFilter)' an. Diese Methode erwartet als Parameter eine Instanz des Filters und zeigt als Ergebnis den passenden Dialog an. Hier wird deutlich, dass diese Komponente in einer engen Beziehung mit der GUI steht. Diese Komponente ist nur für die Anwendungen interessant, die eine Möglichkeit anbieten, die Filtereinstellungen mit Hilfe der Dialoge zu verändern. Als Beispiel kann die Testumgebung betrachtet werden.

3.4.2.3.2 Die Klasse CamFramRegisterForFilter Sobald die Filter registriert werden, kommt diese Klasse ins Spiel. Diese ist dafür verantwortlich, die Filterinstanzen zu speichern und über den Klassennamen wieder eine Referenz zur Verfügung zu stellen. Das Klassendiagramm 3.16 veranschaulicht, dass diese Klasse von der Klasse 'CamFramFilterPool' abgeleitet ist und keine zusätzliche Funktionalität anbietet. Diese Klasse dient also als ein Container für alle registrierten Filter. Interessant ist der Aspekt, dass alle Instanzen der registrierten Filter im Framework nicht als Filter benutzt werden. Sie dienen lediglich als eine Schablone für die automatische Erstellung neuer Filterinstanzen. Solche neuen Filterinstanzen werden dann in der 'CamFramCombiFilter' verwaltet.

3.4.2.3.3 Die Klasse CamFramCombiFilter Ähnlich wie die 'CamFramRegisterForFilter' - Klasse ist diese Klasse aufgebaut. Sie kommt dann zum Einsatz, wenn ein Kombifilter zusammengestellt wird. Diese ist auch von der 'CamFramFilterPool'-Klasse abgeleitet, aber bietet eine zusätzliche

Funktionalität in Form der Methode 'filter()' an. Die hier verwalteten Filterinstanzen sind automatisch erstellt und stellen die Filterfunktionalität dar.

3.4.2.4 Die Persistenz

Das Persistenz-Modul ist für die Persistenz des Kombifilters verantwortlich. Wie schon beschrieben wurde, ist diese Funktionalität sehr wichtig. Damit wird die Kommunikation zwischen der Testumgebung, in der der Kombifilter erstellt und getestet wird, und der Endanwendung ermöglicht.

Die Klasse stellt zwei Methoden zur Verfügung.

- loadCombiFilter
- saveCombiFilter

Wie die Namen schon sagen, ist es mit diesen Methoden möglich, einen Kombifilter abzuspeichern und wieder zu laden.

Gespeichert wird mit Hilfe von XML. Die Abbildung 3.17 zeigt einen so abgespeicherten Filter.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <combifilter>
    <filter class_name="FilterGraustufen" filter_activ="true" filter_name="FilterGraustufen_1" />
    <filter class_name="FilterLaplace" filter_activ="true" filter_name="FilterLaplace_1" />
  - <filter class_name="FilterSW" filter_activ="true" filter_name="FilterSW_1">
    - <settings>
        <settings_entry name="SW" typ="int" value="22" />
      </settings>
    </filter>
  - <dependency>
    - <dependency_filter name="FilterGraustufen_1">
        <dependency_entry dependency_entry_filter="GRABBER" />
      </dependency_filter>
    - <dependency_filter name="FilterLaplace_1">
        <dependency_entry dependency_entry_filter="FilterGraustufen_1" />
      </dependency_filter>
    - <dependency_filter name="FilterSW_1">
        <dependency_entry dependency_entry_filter="FilterLaplace_1" />
      </dependency_filter>
    </dependency>
  </combifilter>
```

Abbildung 3.17: Persistenz mit XML

Die Attribute haben folgende Bedeutung:

- <**combifilter**><**combifilter**/> Beschreibt den für einen Kombifilter relevanten Bereich.
- <**filter**><**filter**/> Definiert eine Instanz des Filters.

- <filter **class_name="**... Gibt den Namen der Filterklasse an.
- <filter **filter_activ='[true,false]'**... Gibt an, ob die Filterinstanz ausgeführt werden soll.
- <filter **filter_name="**... Gibt den Namen der Instanz an.
- <**settings**><**settings/**> Beschreibt den Bereich für die Einstellung einer Filterinstanz.
- <**settings_entry/**> Ein Wert aus der Einstellung.
- <settings_entry **name="**... Gibt den Namen des Wertes an.
- <settings_entry **typ='[int, float, double, long, BOOLEAN, BYTE, CString]'**... Gibt den Typ des Wertes an.
- <settings_entry **value="**... Gibt den Wert der Variable an.
- <**dependency** ><**dependency/**> Gibt den Bereich an, in dem die Abhängigkeiten zwischen den Filtern beschrieben werden.
- <**dependency_filter** > Die Abhängigkeit einer Filterinstanz.
- <dependency_filter **name="**... Gibt den Namen der Filterinstanz an, für die die Abhängigkeiten beschrieben werden.
- <**dependency_entry**><**dependeny_entry**> Beschreibt die notwendigen Filterinstanzen, die ausgeführt werden müssen, bevor 'dependency_filter' ausgeführt wird.
- <dependency_entry **dependency_entry_filter** ... Gibt den Namen der Filterinstanz an.

Wie aus der XML-Datei eindeutig hervorgeht, werden zuerst alle Instanzen der Filter mit allen Einstellungen beschrieben. Danach werden die Abhängigkeiten festgelegt und damit ein Kombifilterbaum aufgebaut.

Die grundlegenden Funktionalitäten wie Lesen, Parsen und Schreiben werden mit Hilfe von Xerces [Xer03] durchgeführt.

Dieses Modul ist so aufgebaut, dass jeder beliebigen Kombifilter damit abgespeichert und geladen werden kann. Eine zusätzliche Anpassung des Modules ist dabei nicht notwendig. Beim Laden eines Kombifilters muss lediglich darauf geachtet werden, dass alle im Kombifilter verwendeten Filter im Framework bekannt sind.

3.4.2.5 Das Util

Grundsätzlich bietet diese Klasse eine Stelle, an der alle Hilfsmethoden implementiert werden müssen. Unter Hilfsmethoden werden dabei alle Methoden verstanden, die eine nützliche Funktionalität für unterschiedliche Klassen darstellen.

Zur Zeit sind in der Klasse nur zwei Methoden implementiert:

- saveImage
- loadImage

Mit diesen Methoden ist es möglich, die Ergebnisse von beliebigen Filtern oder den Grabber als BMP abzuspeichern, sowie die Daten aus der Datei in das Filterobjekt zu laden. Durch das Speichern ist es möglich, alle Schritte der Ausführung nicht nur in Echtzeit anzuschauen, sondern auch als Bilder abzuspeichern.

Natürlich können auch neue Filter diese Funktionalität nutzen. Es ist möglich, dass es einen Filter gibt, der sein Ergebnis als Bild abspeichert. Wenn die Speicherung z.B. einmal pro Minute durchgeführt wird und dabei alle Bilder als Namen die aktuelle Zeitangaben bekommen, besteht die Möglichkeit, den gesamten Ablauf zu protokollieren. Die 'laden'-Funktionalität wird zu diesem Zeitpunkt schon in einem Filter ('CamFramReferenceImageFile') gebraucht. Dieser Filter repräsentiert ein Referenzbild und holt die dafür notwendigen Daten aus einer Datei.

3.5 Das Eventhandling

Bis jetzt wurde nicht über die Kommunikation zwischen den einzelnen Modulen berichtet. In diesem Kapitel wird auf diesen Aspekt der Frameworkfunktionalität eingegangen. Ich werde folgende Fragen erläutern: was unter dem Eventhandling im Framework verstanden wird, welche Komponenten davon betroffen sind und letztendlich wie es realisiert ist?

3.5.1 Was ist das Eventhandling

Unter Eventhandling verstehe ich eine Art der Kommunikation zwischen den Anwendungskomponenten, die dann gebraucht wird, wenn keine direkte Verbindung zwischen diesen Komponenten möglich ist. Ein Beispiel dafür sind die Testumgebung und das Framework. Die Testumgebung kennt das Fra-

mework[8], das Framework kennt aber die Testumgebung nicht. Es wäre auch fatal, wenn es nicht so wäre, weil sonst das Framework für jede einzelne Anwendung angepasst werden müsste. Es wäre aber oft sehr hilfreich, wenn auch das Framework die Nachrichten versenden könnte. Das ist z.B. der Fall bei dem Exceptionhandling. Sobald irgendwas nicht erwartungsgemäß funktioniert, wird eine Exception geworfen. Meistens ist es dem Framework nicht möglich, selbst das Problem zu beheben, so dass an dieser Stelle die Kommunikation mit der Anwendung unbedingt notwendig ist. Nur so ist es möglich, dem Anwender das problem mitzuteilen. In der Abbildung 3.16 kann man eine Klasse sehen, die noch nicht beschrieben wurde: 'CamFramReceiverPool'. Genau diese Klasse ist für das Eventhandling verantwortlich. Wie diese Klasse aufgebaut ist, werde ich im Kapitel 'Realisierung' ausführlicher erklären, hier reicht es zu wissen, dass der Kern die Hauptkomponente ist, die für das Eventhandling verantwortlich ist.

3.5.2 Betroffene Komponenten

Ein Beispiel mit dem Exceptionhandling habe ich bereits erwähnt. Davon sind eigentlich alle Frameworkklassen betroffen, die mit den Exceptions umgehen müssen. Ganz anders verhält es sich mit den Filtern. Bei der Beschreibung der Filter wurde betont, dass die Filter als Ergebnis nicht nur die Bildinformation haben können. Es ist auch denkbar, dass bestimmte Filter zusätzliche Information aus dem Bild gewinnen können. Es ist oft sinnvoll, diese zusätzlichen Information einerseits so schnell wie möglich der Anwendung zur Verfügung zu stellen, andererseits es nur dann zu machen, wenn es auch sinnvoll ist. Dieses Verhalten möchte ich an einem Beispiel ausführlicher erklären.

Nehmen wir an, dass eine Anwendung geschrieben werden muss, die die Kollision zwischen zwei Objekten kontrollieren soll. Wir nehmen weiterhin an, dass sich die Objekte farblich unterscheiden und die Formen der Objekten ein Quadrat bzw. ein Kreis sind. Für diese Anwendung ist die Bildinformation irrelevant. Die wichtigen Informationen stellen aber die Koordinaten des Schwerpunktes eines jedes Objektes und ihre Radien dar, aus welchen wir ungefähr wissen, wie gross die Objekte sind. Weiterhin wird angenommen, dass der Radius einen bestimmten Mindestwert als Grenze haben soll. Das ist dazu notwendig, dass eventuelle Bild- und Farbstörungen nicht als Objekte erkannt werden. Es handelt sich um eine Echtzeitanwendung, wo jede relevante Information so schnell wie möglich bearbeitet werden muss. An diesem

[8]Mit 'kennt' ist gemeint, dass eine Instanz gehalten wird, so dass es möglich ist direkt eine Nachricht zu versenden. (Eine Methode auszuführen)

Beispiel werden beide oben beschriebenen Verhalten deutlich. Sobald ein Filter die Information gewonnen hat, muss diese Information an die Anwendung gesendet werden. Das ist aber nur dann sinnvoll, wenn diese Information für die Anwendung relevant ist. Wenn z.B. der Radius eines Objektes zu klein ist, dann handelt es sich wahrscheinlich um eine Fehlinformation, z.B. eine Bildstörung. In einem solchen Fall ist es nicht sinnvoll, die Anwendung mit diesen Daten zu beschäftigen.

Dieses Verfahren erlaubt ein zusätzliches Trennen der Anwendung und des Frameworks im Sinne der Filter. Die Anwendung braucht nur die Event-Objekte zu kennen (werden weiter unten beschrieben) und keine Informationen über Filter.

3.5.3 Das Design des Eventhandling

Wie schon erwähnt wurde, muss die Anwendung von der Klasse 'CamFramReceiver' abgeleitet werden. Warum das so ist, wird hier erklärt. Diese Klasse schreibt für alle Unterklassen eine Methode vor: 'eventHandling (CamFramHandlingObject)'. Damit wird gewährleistet, dass diese Klasse die Events empfangen kann. Das Klassendiagramm 3.18 zeigt die Zusammensetzung der Klassen. Es ist leicht zu erkennen, dass es sich dabei um ein Composite-

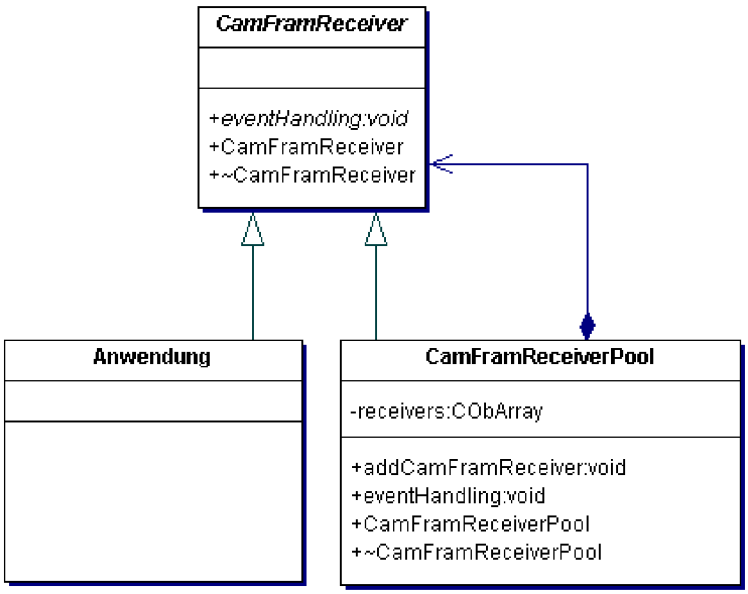

Abbildung 3.18: CamFramReceiverPool

Pattern [Gam01] handelt. Der Kern des Frameworks hält eine Instanz der

Klasse 'CamFramReceiverPool' und ist damit in der Lage, mehrere Receiver zu halten.

3.5.3.1 Das EventhandlingsObjekt

Das Framework stellt die Klasse 'CamFramHandlingObject' bereit, welche das Handlingobjekt darstellt. Diese Klasse besitzt zwei Attribute:

- handlingId als int
- handlingMessage als CString

Damit ist es möglich, einfache Events in Form einer Message zu versenden. Die 'handlingId' definiert die Art und den Typ des Events. Dabei sorgen zwei header-Dateien (CamFramEventID.h und FilterEventID.h) dafür, dass die IDs aus Versehen nicht vertauscht werden. Wie in der Abbildung 3.19

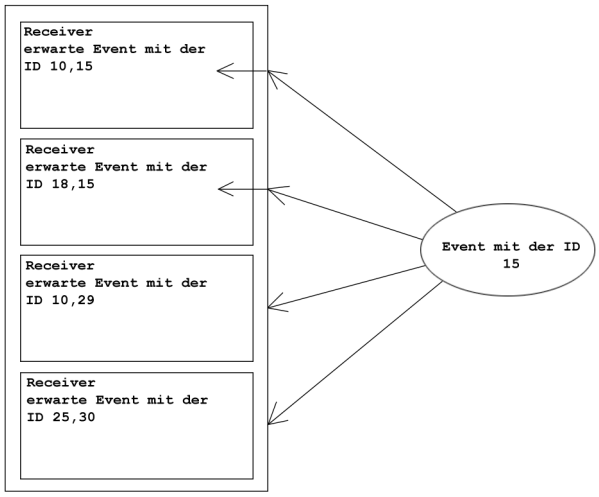

Abbildung 3.19: Eventverarbeitung

zu sehen ist, wird das Event an alle eingetragenen Receiver versendet. Die Entscheidung, ob das Event tatsächlich bearbeitet wird, trifft der Receiver selbst, indem er das 'handlingId' ausliest und mit dem, welches er bearbeiten kann, vergleicht.

Anders verhält es sich bei den Filtern. Hier reicht 'CamFramHandlingObject' nicht aus, da meistens zusätzliche Informationen, wie die Koordinaten des Schwerpunktes, übertragen werden müssen. In diesem Fall muss eine eigene Klasse erstellt werden, die von der oben genannten Klasse abgeleitet

werden muss. Jeder Filter enthält eine spezielle Methode, die für das Versenden des Events zuständig ist. Dabei wird ein Objekt des Types 'EventhandlingsObjekt' versendet, was die 'handlingId'-Abfrage ermöglicht. Sobald die Anwendung (der Receiver) erkannt hat, dass es sich um ein Event handelt, welches sie bearbeiten kann, weiß die Anwendung dann auch den richtigen Typ des Events. Damit ist es möglich, auf den richtigen Typ zu casten um auf alle Daten zugreifen zu können.

Kapitel 4

Die Realisierung

Nachdem das Design im vorherigen Kapitel erläutert wurde, wird in diesem Kapitel das Thema der Framework-Realisierung ausgebaut. Das Programm, welches hier vorgestellt wird, dient nicht nur zur Überprüfung der Tragfähigkeit des Designs, sondern ist auch ein fertiges Produkt, das sofort eingesetzt werden kann.

4.1 Benutzte Plattform

Bevor die Realisierung in Details erklärt wird, muss über die ausgewählte Plattform etwas gesagt werden. Im Kapitel 'Anforderungen' wurden schon die Hauptaspekte für die Implementierung genannt, so dass hier diese Wahl nur bestätigt wird.

Die Realisierung wurde mit Hilfe der Microsoft Visual C++ 6.0 - Entwicklungsumgebung durchgeführt. Das dabei benutzte Betriebssystem war Microsoft XP Pro. Als Videoquelle wurde eine Webcam der Firma Logitech, Model QuickCam Pro 4000 genommen und mit Hilfe der 'Video for Window'-Schnittstelle wurden die Daten dem Programm zur Verfügung gestellt.

4.2 Was wurde gemacht

Es ist schon mehrfach das im Laufe der Diplomarbeit entwickelte Programm angesprochen wurde. Nachfolgend wird konkret über seine Komponenten berichtet. Die folgenden Module sind implementiert worden und stehen zur Zeit zur Verfügung:

- Framework
- Testumgebung (Kombifilter-Editor)

- Testfilter
- Grabber
- Eine Testanwendung (Colisiondetector)

4.2.1 Das Framework

Das Framework stellt dabei die Hauptkomponente dar. Das dem zugrunde liegende Design wurde bereits im vorherigen Kapitel ausführlich beschrieben. Dieses Modul stellt die Laufzeitumgebung zur Verfügung, mit deren Hilfe die Filter ausgeführt und externe Anwendungen angesteuert werden können. Die Abstraktion des Frameworks erlaubt die flexible Nutzung. So ist es möglich, eine beliebige Videoquelle anzubinden, sowie eine beliebige Anwendung anzusteuern.

4.2.2 Die Testumgebung

Die Testumgebung ist eine Anwendung, die vom Framework angesteuert wird. Dieses Modul demonstriert anschaulich die Funktionalität des Frameworks. Damit ist es möglich, neu entworfene Filter schnell einzeln und in der Kombination mit anderen Filter zu testen. Gleichzeitig dient dieses Modul als ein Editor für die Kombifilter. Die Funktionalität dieser Anwendung wird im Abschnitt 4.4 ausführlich beschrieben.

4.2.3 Die Testfilter und der Grabber

Damit die Funktionalität des Frameworks getestet werden kann, wurden die Testfilter entwickelt. Das gleiche gilt auch für die Videoquelle. Dabei wurde eine Webcam mit Hilfe der VfW-Schnittstelle angesprochen.

4.2.4 Eine Testanwendung

Da die Testumgebung sehr umfangreich ist, kann schwer eingeschätzt werden, wie aufwendig die Einbindung einer Anwendung an das Framework ist. Deswegen wurde eine kleine Anwendung entworfen und implementiert, die genau dieses Verhalten demonstrieren soll. Diese Anwendung wird im Abschnitt 5.2 ausführlicher beschrieben.

4.3 Der Projektaufbau

Die beschriebenen Teilbereiche der Realisierung ergeben gleichzeitig den Projektaufbau. Die Testumgebung dient dabei als Anwendung und erzeugt eine ausführbare .exe - Datei. Andere Projekte erzeugen die DLL-Dateien. Wie die Projekte zusammengebunden sind und welche Abhängigkeiten zwischen diesen Projekten existieren, soll hier detaillierter erläutert werden. Die Abbildung 4.1 demonstriert die Sichtbarkeit der Projekte.

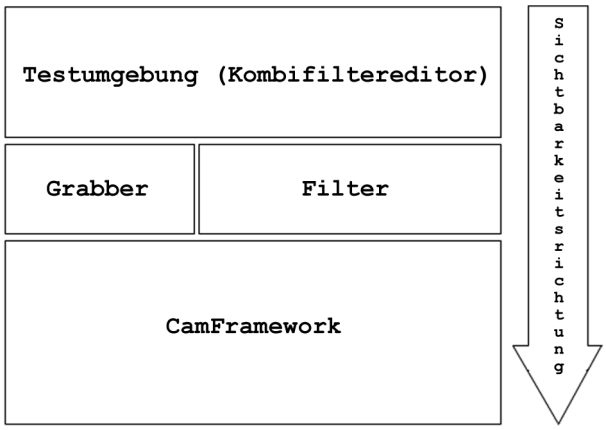

Abbildung 4.1: Sichtbarkeit der Projekte

Ganz oben befindet sich die Testumgebung. Sie sieht alle darunter liegenden Projekte. In der Mitte befinden sich zwei voneinander getrennte Projekte (Grabber und Filter), die auf dem CamFramework basieren und entsprechend auch dieses Projekt sehen können. Ganz unten befindet sich das Herz der Realisierung, das Framework. Dieses Projekt ist von den anderen Projekten absolut unabhängig.

4.3.1 Die Projekte unter Visual C++ 6.0

Es existieren also vier Projekte. Die Abhängigkeiten zwischen diesen Projekten entsprechen der Sichtbarkeit der Projekte aus der Abbildung 4.1. In Abhängigkeit von Projekteinstellungen wird die Weiterentwicklung unterschiedlich schwer sein. Deswegen wird an der Stelle eine mögliche Konfiguration der Projekte vorgeschlagen. Die Abbildung 4.2 demonstriert, wie die Einstellungen vorgenommen werden müssen. Die Idee dabei ist folgende: die Projekte, die Header und lib - Dateien von anderen Projekten brauchen, suchen diese Ressourcen direkt in entsprechenden Projekten. Alle dll-Dateien

werden direkt in das Projekt der Anwendung (Release bzw. Debug) geschrieben.

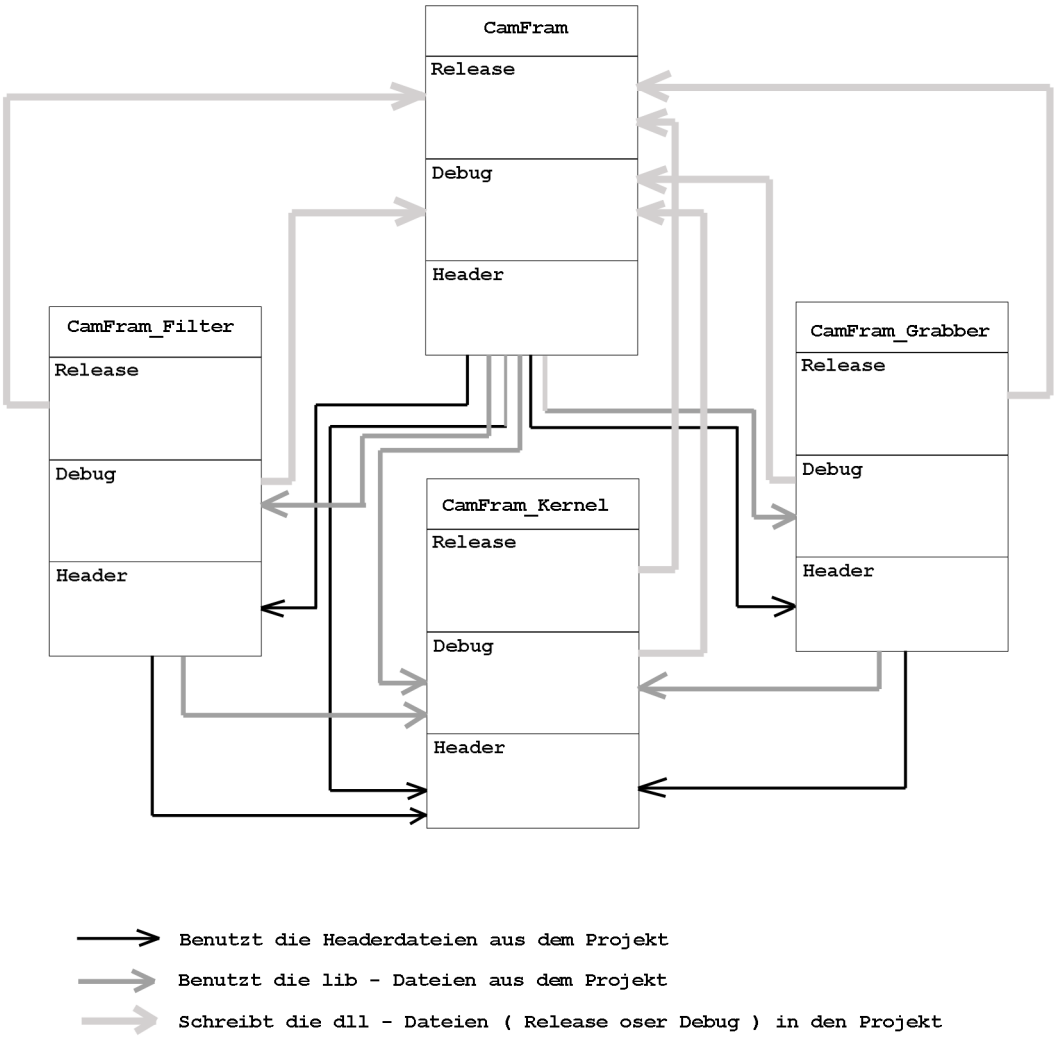

Abbildung 4.2: Die Projekteinstellungen

Grosse Kästchen in der Abbildung symbolisieren die Projekte. Release und Debug sind die Verzeichnisse, Header bezeichnet nur symbolisch den Ort, wo sich die Header-Dateien im Projekt befinden. Meistens sind sie direkt im Projektverzeichnis zu finden.

Der Vorteil dieser Einstellungen liegt darin, dass die Header bzw. lib - Dateien nicht redundant vorkommen und das Projekt, das diese Dateien braucht, immer die aktuellste Version nutzen kann. Da alle dll's gleich in

das Projekt der Anwendung geschrieben werden, kann die Anwendung sofort aus jedem Projekt gestartet werden. Auch das Debuggen von dll's ist damit sofort möglich, ohne zusätzliche Einstellungen vornehmen zu müssen.

4.4 Die Testumgebung

Wie schon mehrfach erwähnt wurde, hat dieses Modul mehrere funktionale Rollen.

- Testumgebung
- Kombifiltereditor

Abhängig davon, in welcher Rolle dieses Modul angewendet wird, unterscheiden sich auch die Vorgehensweisen. Wenn es sich bei der Rolle des Kombifiltereditors hautsächlich um die Reihenfolge der existierenden[1] Filter und ihrer Einstellungen handelt, muss bei der Rolle 'Testumgebung' auch der Quellcode angepasst werden. Mehr dazu wird im Abschnitt 4.6 geschrieben.

4.4.1 Die Oberfläche

Die Abbildung 4.3 zeigt, wie die Oberfläche aufgebaut ist. Die einzelnen Bereiche, sowie die Steuerkomponenten werden nachfolgend beschrieben.

1. Die Programmsteuerleiste. Die Funktionalität wird ausführlicher im Abschnitt 4.4.1.1 beschrieben.
2. In diesem Fenster wird das Bildstream von dem Grabber unverändert angezeigt. Das Fenster zeigt die Bilder immer in gleicher Größe (352 x 288). Es ist zu beachten, dass die tatsächliche Größe anders sein kann. Das Fenster verändert die Originalgröße der Bilder nicht und verkleinert oder vergrößert sie nur für die Darstellung.
3. Sobald mit der Maus im Fenster (2) oder (4) geklickt wird, erscheinen hier einige Informationen, die für die Testzwecke sehr hilfreich sein können.
   (a) 'Position (XY)' zeigt die Koordinaten im Bild, wo mit der Maus geklickt wurde. Diese Koordinaten werden in Abhängigkeit

[1] Unter 'existierende' Filter werden hier alle die Filter verstanden, die im CamFramework registriert sind. D.h. in der Testumgebung auch unter 'Registrierte Filter' angezeigt werden.

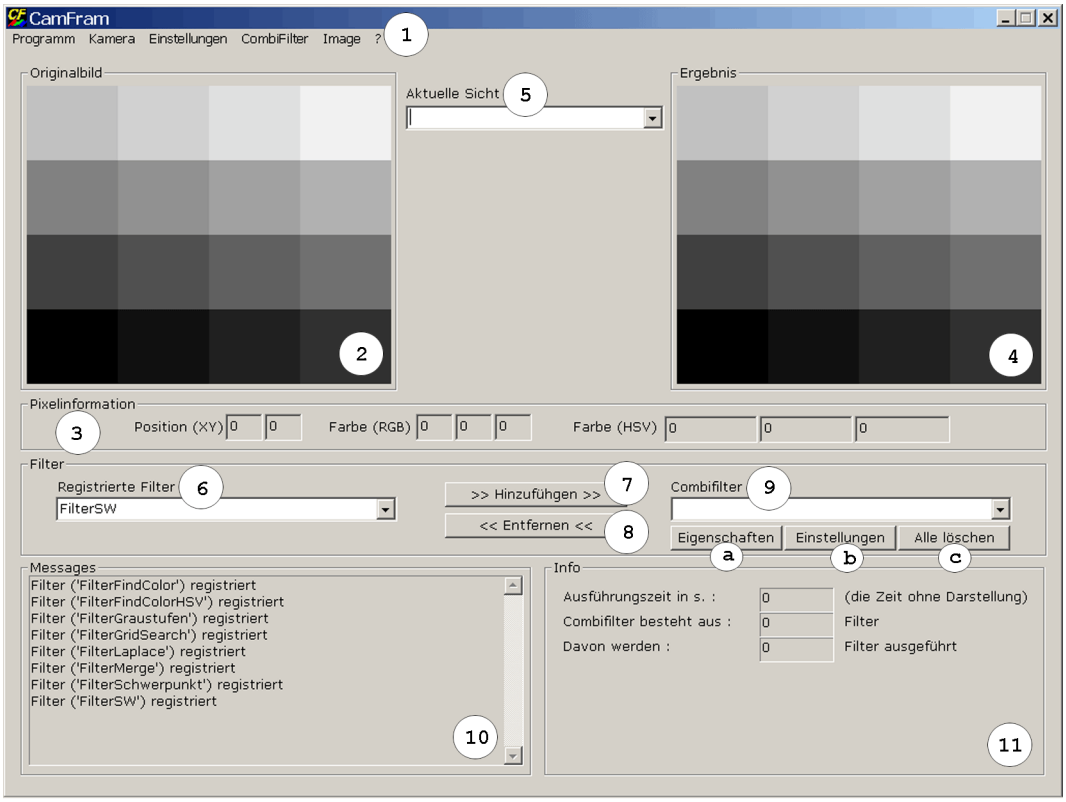

Abbildung 4.3: Die Oberflaeche der Testumgebung

von der Bildauflösung berechnet, obwohl die Fenstergröße nicht verändert wird. So können die Koordinaten z.B. 600 x 400 sein, wenn die Originalgröße der Bilder 640 x 480 ist.

(b) 'Farbe (RGB)' zeigt die RGB-Farbe im Punkt 'Position (XY)'. Dabei ist zu beachten, dass diese Information zum Zeitpunkt der Mausklicks ermittelt wird und weiter nicht automatisch aktualisiert wird.

(c) 'Farbe (HSV)' zeigt ähnlich wie RGB-Farbe die HSV-Farbe an. Auch dabei handelt es sich um den Zeitpunkt der Mausklicks.

4. In diesem Fenster wird das Ergebnis der Filter angezeigt. Am Anfang, wenn die Kamera schon gestartet ist, aber noch kein Filter ausgewählt wird, wird das Originalbild angezeigt. Dann sind beide Fenster (2 und 4) identisch.

5. In dieser Combobox werden alle Filter angezeigt, die im Kombifilter

vorkommen. Automatisch wird immer der letzte Filter angezeigt, damit zeigt das Fenster 4 das Ergebnis des Kombifilters. Es ist aber möglich, beliebige Filter aus dieser Combobox auszuwählen und damit das Zwischenergebnis anzeigen zu lassen.

6. Diese Combobox zeigt alle im Kernel registrierten Filter. Diese Filter können beliebig kombiniert werden. Diese Kombination wird auch als Kombifilter bezeichnet.

7. Um einen registrierten Filter in den Kombifilter zu übernehmen, muss dieser Button geklickt werden. Es erscheint ein neues Fenster, in dem alle dafür notwendigen Einstellungen vorgenommen werden können. Die Abbildung 4.4 zeigt, wie das Fenster aufgebaut ist.

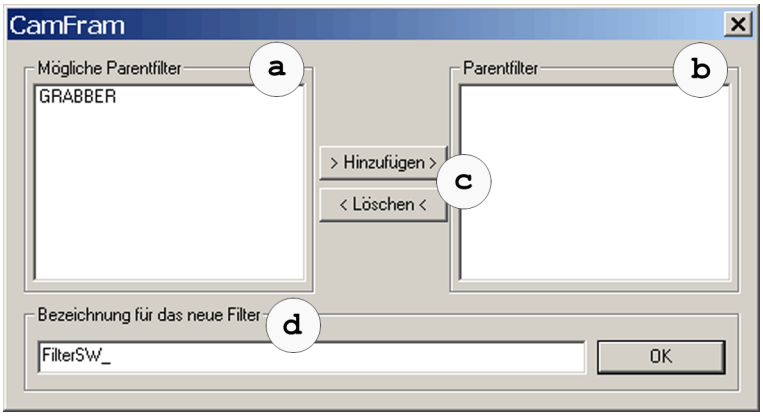

Abbildung 4.4: Fenster - Ein neuer Filter

(a) In diesem Fenster werden alle schon im Kombifilter vorhandenen Filter angezeigt, die als 'Parent'[2] für neuen Filter in Frage kommen.

(b) Hier werden alle übernommenen Parentfilter angezeigt. Es ist wichtig zu wissen, dass sich in diesem Fenster mehr Filter befinden können, als der gerade erstellte Filter bearbeiten kann. Dieses Verhalten ist dann notwendig, wenn ein bestimmter Filter nicht für das Gesammtergebnis gebraucht wird, aber trotzdem ausgeführt werden muss. Die Abbildung 4.5 demonstriert die mögliche Situation. Der Filter_3 braucht nur einen Filter als Quelle (Fil-

[2]Hier wird ein Filterbaum aufgebaut. Wie es schon im Kapitel 'Design' erwähnt wurde, kennt jeder Filter seine Vorgänger, hier als Parents genannt.

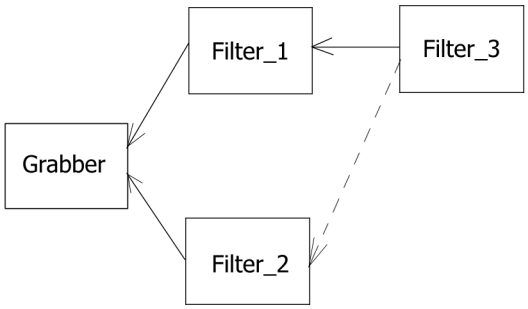

Abbildung 4.5: Aktivieren von bestimmten Filtern

ter_1). Wenn als Ergebnis der Filter_3 ausgewählt wird, werden alle unnötigen Filter als 'inaktiv' markiert und damit nicht ausgeführt. Dieses Konzept wurde bereits im Abschnitt 3.2 beschrieben. Das geschieht auch mit dem Filter_2 in diesem Beispiel. Es ist aber denkbar, dass bestimmte Filter trotzdem ausgeführt werden müssen, obwohl sie für das Endergebnis nicht nötig sind. Dieses Problem wird gelöst, indem der Filter, der ausgeführt werden soll, als eine zusätzliche Quelle eingetragen wird. Hier bekommt der Filter_3 den Filter_2 als zweite Quelle. Für den Filter_3 hat es absolut keine Bedeutung, da nur die erste Quelle abgefragt wird, der Filter_2 wird nun aber ausgeführt.

   (c) Mit diesen Buttons ist es möglich, einen oder mehrere Parentfilter aus dem Fenster a) ins Fenster b) oder umgekehrt zu verschieben.

   (d) Am Ende muss noch ein Name für den neuen Filter bestimmt werden. Der Name muss eindeutig sein und darf nicht genau so lauten, wie der Klassenname des Filters.

8. Mit diesem Button ist es möglich, einen im Kombifilter vorhandenen Filter aus diesem zu entfernen. Dabei wird überprüft, ob die vom zu entfernenden Filter abhängigen Filter existieren oder der Filter gerade ausgeführt wird. In beiden Fällen wird eine entsprechende Fehlermeldung angezeigt.

9. In dieser Combobox werden alle Filter angezeigt, die schon zu einem Kombifilter gehören. Die Buttons hinter der Combobox beziehen sich entweder auf einen ausgewählten Filter (a und b), oder auf den ganzen Kombifilter (c).

   (a) Dieses Button bezieht sich auf den in der Combobox (9) markier-

ten Filter. Damit ist es möglich, die Baumstruktur im Nachhinein zu verändern. Auch der Filtername kann geändert werden. Bei klicken auf diesen Button erscheint schon beschriebenes Fenster 4.4.

(b) Auch dieser Button bezieht sich auf den markierten Filter. Abhängig davon, ob der ausgewählte Filter die Einstellungsmöglichkeiten anbietet, wird ein für den Filter passender Dialog angezeigt. Für die Filter ohne Einstellungsmöglichkeiten wird eine entsprechende Meldung ausgegeben.

(c) Dieser Button bezieht sich auf den gesamten Kombifilter. Alle Filter in der Combobox werden gelöscht. Beide Fenster (2 und 4) zeigen dabei die gleichen Bilder von der Videoquelle.

10. Wie schon erwähnt wurde, nutzt diese Testumgebung den Frameworkkernel. Alle Kernel-Mitteilungen werden hier angezeigt.

11. Der Kernel stellt ein paar zusätzlichen Informationen zur Verfügung, die direkt abgefragt werden können. Diese Informationen werden in diesem Fenster angezeigt.

4.4.1.1 Die Steuerleiste der Oberfläche

In diesem Abschnitt wird die Funktionalität der Steuerleiste detaillierter beschrieben.

- 'Programm' (Die Abbildung 4.6) enthält nur ein Eintrag: 'Schliessen'. Die Funktionsweise entspricht dem Kreuzchen in der Testoberfläche und bewirkt die Schließung der Anwendung.

Abbildung 4.6: Steuerleiste - Programm

- 'Kamera' (Die Abbildung 4.7) enthält die Befehle, die die Ausführung des Arbeitszyklus[3] des Kernels ansteuern.

[3]Die Kamera wird beim Programmstart initialisiert und gestartet. Die Daten von der Kamera werden erst dann ausgelesen, wenn sie gebraucht werden. Der Arbeitszyklus ist die Stelle, in der die Kamera angesprochen wird.

Abbildung 4.7: Steuerleiste - Kamera

  - 'Start' startet die Ausführung des Kombifilters. Es werden folgende Schritte nacheinander im Zyklus durchgeführt:
    * neues Bild von der Kamera holen
    * Kombifilter ausführen.[4]
  - 'Stop' beendet die Ausführung des oben genannten Arbeitszyklus.
  - 'Einfrieren' verändert den Arbeitszyklus so, dass die Kamera nicht mehr angesprochen wird. Damit wird erreicht, dass die Ausführung des Kombifilters immer auf dasselbe Bild durchgeführt wird. Ein Beispielszenario für diese Option könnte z.B. so aussehen: es muss eine Farbe ausgefiltert werden. Die Daten der Farbe müssen zuerst ermittelt werden, indem mit der Maus im Fenster des Originalbildes auf die Farbe geklickt werden muss. Das Problem liegt darin, dass das Objekt, dessen Farbe ermittelt werden muss, sich sehr schnell bewegt, so dass nicht festgestellt werden kann, ob die richtige Farbe getroffen wurde. Mit der Option 'Einfrieren' ist es nun möglich das Bildstream anzuhalten, die richtige Farbe anzuklicken und damit die Einstellungen richtig vorzunehmen. Wenn die Ausführung fortgesetzt werden soll, muss wieder der Befehl 'Start' ausgeführt werden.

- 'Einstellungen' (Die Abbildung 4.8) enthält die Einstellungsmöglichkeiten der Kamera. Diese Funktionalität wird von der 'Video for Windows' - Schnittstelle angeboten und wird somit als einfache Delegation in der Testumgebung angesehen. Zu beachten ist der Aspekt, dass auch die 'VfW'-Schnittstelle weiter an einen konkreten Treiber der Videoquelle delegiert. Damit können die Einstellungsfenster, die nach dem Klicken der beiden möglichen Kommandos erscheinen, abhängig von dem Kameratreiber unterschiedlich aussehen. Die Grundfunktionalität muss aber überall vorhanden sein.
  - 'Kamera' öffnet ein Fenster mit den für die konkrete Kamera spezifischen Einstellungsmöglichkeiten. So ist es z.B. möglich, die Hel-

[4]Der Ablauf wurde im Detail schon im Kapitel 'Design' vorgestellt.

Abbildung 4.8: Steuerleiste - Einstellungen

ligkeit oder den Kontrast einzustellen. Aber auch kompliziertere Funktionen, wie 'Face Tracking'[5] sind möglich. In jedem Fall muss die Möglichkeit vorhanden sein, die Videoquelle auswählen zu können. Das ist dann notwendig, wenn mehrere Videoquellen angeschlossen sind.

  - 'Format' öffnet ein Fenster, in dem die Auflösung und das Format der Bilder eingestellt werden können. Diese Einstellung ist auch eine Grundfunktionalität und muss überall vorhanden sein.[6]

- 'Combifilter' (Die Abbildung 4.9) repräsentiert die Persistenzbefehle. Damit ist es möglich, neu kreierte Kombifilter abzuspeichern, sowie ein schon vorhandenen zu laden.

Abbildung 4.9: Steuerleiste - Kombifilter

  - 'Filter laden' öffnet einen Dialog, in dem die Auswahl der Dateien möglich ist. Nachdem der Filter ausgewählt wurde, wird er geladen. Der schon im Programm vorhandene Filter geht dabei verloren!
  - 'Filter speichern' öffnet einen Dialog, mit dessen Hilfe der Filter an einer beliebigen Stelle unter einem beliebigen Namen[7] abgespeichert werden kann.

[5]Diese Funktion ist im Treiber für die Logitech QuickCam Pro 4000 integriert und ermöglicht automatische Gesichtserkennung, so dass die Kamera mit Hilfe von Zoom-Funktion versucht, das Gesicht immer in der Mitte zu platzieren.

[6]In diesem Fenster muss das Format RGB24 eingestellt werden. Der Grund dafür wurde im Kapitel 'Design' ausführlich beschrieben.

[7]Unter 'beliebigen' sind in diesem Kontext das vom Betriebsystem erlaubte Format und Ort gemeint.

- 'Image' (Die Abbildung 4.10) enthält die Befehle für die Hilfsfunktionalität. So ist es möglich, die Bilder aus den beiden Videofenstern der Testumgebung im *.BMP - Format abzuspeichern.

Abbildung 4.10: Steuerleiste - Image

  - 'Originalimage speichern unter' speichert, wie die Bezeichnung schon sagt, das Originalimage ab. Auch hier erscheint der dafür notwendige Dialog, welches schon erläutert wurde.
  - 'Aktuelle Sicht speichern unter' speichert die aktuelle Sicht des Filterfensters ab. Ansonsten ist die Funktionalität identisch mit der von 'Originalimage speichern unter'.

- '?' (Die Abbildung 4.11) enthält die Befehle, die Information über das Programm liefern.

Abbildung 4.11: Steuerleiste - Info

  - 'Info' hier wird die Versionsnummer des Programms angezeigt.

4.4.2 Das Kombifiltereditor

In diesem Abschnitt geht es um die Vorgehensweise der Entwicklung eines Kombifilters. Noch ein Mal zur Erinnerung: ein Kombifilter wird mit Hilfe der Testumgebung aus den im Framework registrierten Filtern zusammengestellt. Der Code der Anwendung muss dabei nicht verändert werden. Das Ziel besteht darin, eine passende Filterkombination mit bestimmten Einstellungen rauszufinden, die für eine konkrete Problemstellung die beste Lösung anbietet. Das Ergebnis kann dabei als reine Information betrachtet werden,

damit ein neuer Filter entwickelt werden kann, oder als ein selbständiger Filter, der aus einer anderen Anwendung ausgelesen und ausgeführt wird.[8]

4.4.2.1 Step by Step - ein Kombifilter

Um besser die Funktionalität der Testumgebung als Kombifiltereditor zu demonstrieren, soll ein Kombifilter entwickelt werden. Die Beispielaufgabe, die der Kombifilter lösen soll, besteht darin, zwei bestimmte, unterschiedlich gefärbte Objekte zu finden und deren Schwerpunkte zu ermitteln.

Diese Aufgabe wurde so gewählt, dass die Lösung mit allen schon zur Verfügung stehenden Mitteln, in unserem Fall mit vorhandenen Filtern, ermittelt werden kann. Das Bauen eines Kombifilters kann folgendermaßen gegliedert werden:

- Überlegen, welche Schritte gemacht werden müssen;
- Prüfen, ob dafür notwendige Filter schon vorhanden sind;
- Filter zusammensetzen;
- Den Kombifilter abspeichern;

Bei der Lösung der oben gestellten Aufgabe wird der zweite Punkt übersprungen, da die dafür notwendigen Filter schon zu Verfügung stehen.

4.4.2.1.1 Überlegen, welche Schritte gemacht werden müssen. Die Aufgabenstellung verrät schon die Reihenfolge der Filter. Zuerst muss eine bestimmte Farbe gefunden werden. Das heißt, dass alle anderen Farben ausgeblendet werden müssen. Danach kann der Schwerpunkt dieser Farbe ermittelt werden. Das gleiche wird für die zweite Farbe wiederholt. Am Ende können beide Ergebnisse zusammengefügt werden, damit beide Schwerpunkte auf einem Bild zu sehen sind. Die Abbildung 4.12 zeigt, wie die Filter zusammengesetzt werden müssen.

Die Methode, wie die gesuchten Farben ermittelt werden, ist frei zu wählen. Zur Zeit sind zwei Filter implementiert, die diese Aufgabe bewerkstelligen können. 'FindeColor' sucht anhand des RGB-Farbraumes und 'FindColorHSV' mit Hilfe des HSV-Farbraumes. Abhängig davon, in welcher Umgebung die Farben gesucht werden, können unterschiedliche Filter angewendet werden. Der 'FindeColorHSV'-Filter ist weniger beleuchtungsabhängig, so dass damit im Allgemeinen ein besseres Ergebnis erzielt werden kann.

[8] Solche Anwendung muss auf dem CamFramework basieren.

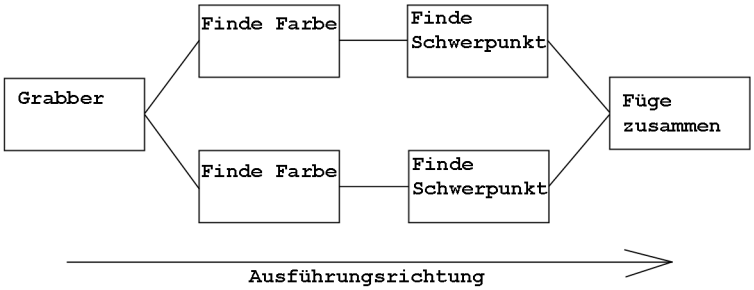

Abbildung 4.12: Kombifilter - Suche Schwerpunkte

4.4.2.1.2 Prüfen, ob dafür notwendige Filter schon vorhanden sind.

Wie es schon erwähnt wurde, kann dieser Punkt für unsere Aufgabenstellung ignoriert werden.

Im Allgemeinen müssen alle die Filter, die gebraucht werden, aber noch nicht vorhanden sind, implementiert werden. Dabei muss die Entscheidung getroffen werden, ob die fehlenden Filter nicht mit einer Kombination aus schon existierenden Filtern ersetzt werden können. Als Beispiel dafür kann unsere Aufgabenstellung dienen. Natürlich könnte man einen Filter implementieren, der alles auf ein Mal durchführt. Das ist sicherlich auch sinnvoll, wenn die Ausführungszeit des Kombifilters nicht zufrieden stellend ist, aber für das Testen und für die Anwendungen, für die das Kombifilter-Temp ausreichend ist, reicht es auf jeden Fall aus. Dabei wird auch die Zeit gespart, die für die Implementierung eines neuen Filters notwendig ist.

4.4.2.1.3 Filter zusammensetzen. Nachdem die Testumgebung gestartet wurde, muss die Ausführung des Kernels gestartet werden. Das geschieht indem in der Menü-Leiste 'Kamera' der Befehl 'Start' ausgewählt wird. Danach zeigen beide Fenster (das Originalbild und das Ergebnis) das selbe Bild, wie in der Abbildung 4.13 zu sehen ist. Die Testumgebung signalisiert damit, dass sie für die Arbeit bereit ist. Dabei ist auch das Fens-

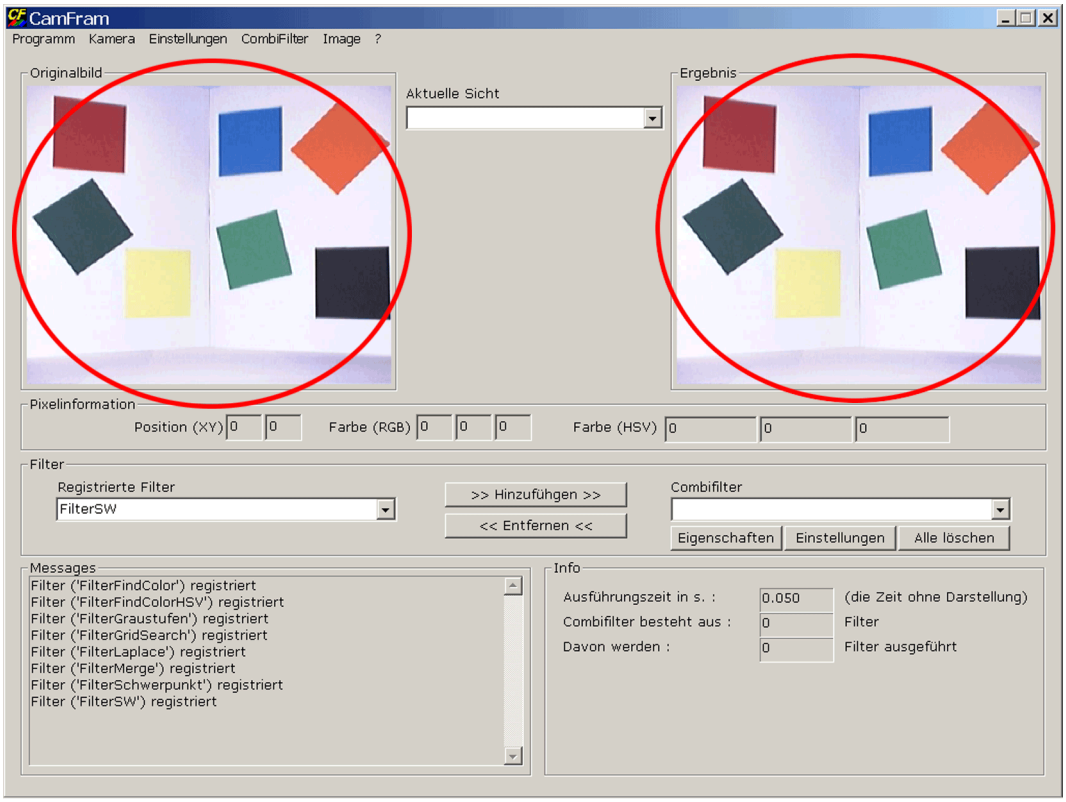

Abbildung 4.13: Einen Kombifilter bauen: der Anfang

ter 'Messages' zu beachten. Jeder Filter, der im Framework registriert wird, wird in diesem Fenster protokolliert. Damit ist es leicht erkennbar, ob alles problemlos gestartet wurde. Das Originalbild enthält sieben Quadrate mit unterschiedlichen Farben. Der zu erstellende Filter soll aus diesen Objekten nur das gelbe und das blaue bearbeiten.

1. Zuerst muss eine bestimmte Farbe gefiltert werden. Der dafür notwendige Filter heißt 'FilterFindeColorHSV'. Dieser Filter wird im Combobox 'Registrierte Filter' ausgewählt.

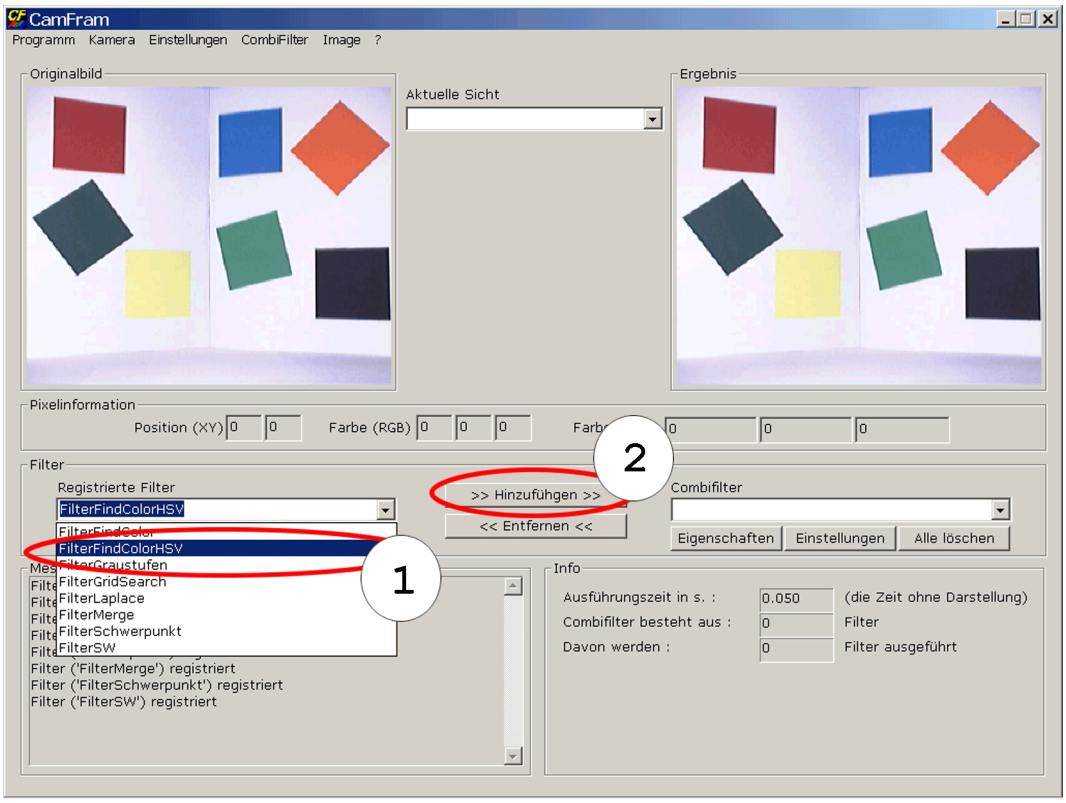

Abbildung 4.14: Einen Kombifilter bauen: Schritte 1-2

2. Mit dem Klicken auf den Button 'Übernehmen' wird eine neue Instanz von dem ausgewählten Filter erstellt und zu unserem noch leeren Kombifilter hinzugefügt. Dabei braucht die Testumgebung noch zusätzliche Informationen, wie:

   - Wer ist für diese Filterinstanz der Parent?
   - Wie heißt die neue Filterinstanz?

   Gleich nach dem Klicken des 'Übernehmen' - Buttons erscheint das Fenster, in dem diese Eingaben vorgenommen werden müssen.

3. Weil wir mit dem Bauen des Kombifilters erst jetzt angefangen haben, steht rechts im Fenster 'Mögliche Parentfilter' nur ein Eintrag: 'GRABBER'. Es wurde schon erwähnt, dass der Grabber immer vorhanden ist und an eine beliebige Stelle im Kombifilter angesetzt werden kann. Dieser 'GRABBER' wird nun ausgewählt.

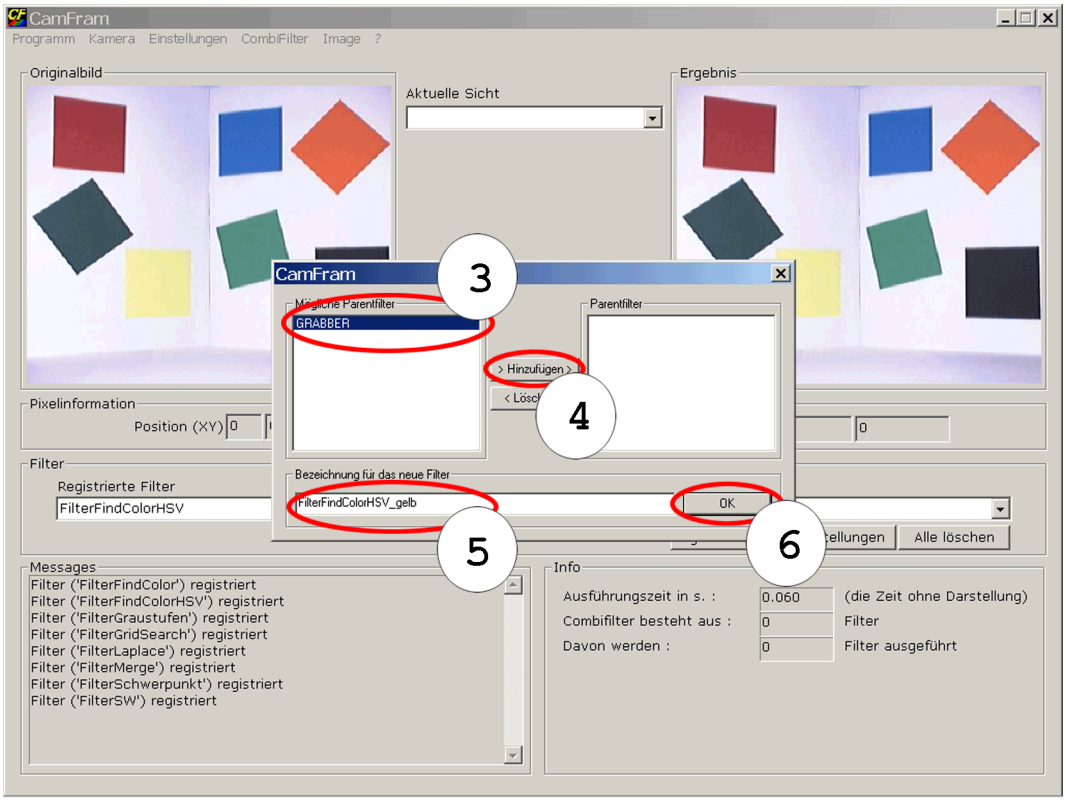

Abbildung 4.15: Einen Kombifilter bauen: Schritte 3-6

4. Der Button 'Hinzufügen' bewirkt, dass der Grabber als Parent für die neue Filterinstanz genommen wird. Der Eintrag erscheint dabei im Fenster 'Parentfilter'.

5. Jetzt muss ein Name für die Filterinstanz angegeben werden. Dabei ist zu beachten, dass dieser Name nicht identisch mit dem Klassennamen des Filters sein darf. Die Testumgebung hilft dabei, indem der neue Name vorgeschlagen wird. Der Aufbau sieht folgendermaßen aus: 'Klassenname'+'_'. Es ist sehr zu empfehlen, den Namen aussagekräftig zu machen, da sonst schwierig wird, aus vielen Filterinstanzen die für die

Einstellungen richtige Instanz rauszufinden. Da als erste Farbe 'gelb' ausgewählt wurde, bekommt diese Filterinstanz den Namen: 'FilterFindeColorHSV_gelb'.

6. Sobald der Button 'OK' geklickt wird, ist ein Kombifilter mit einem Filter erstellt.

7. In der Abbildung 4.16 ist zu sehen, dass das 'Ergebnis'-Fenster nun schwarz ist. Die Erklärung dafür ist folgende: die Filterinstanz weiß noch nicht, welche Farbe gesucht wird. Um die notwendigen Einstellungen vorzunehmen, muss mit dem Mauszeiger im Fenster 'Originalbild' auf die zu filternde Farbe geklickt werden. In unserem Beispiel soll auf das gelben Viereck geklickt werden.

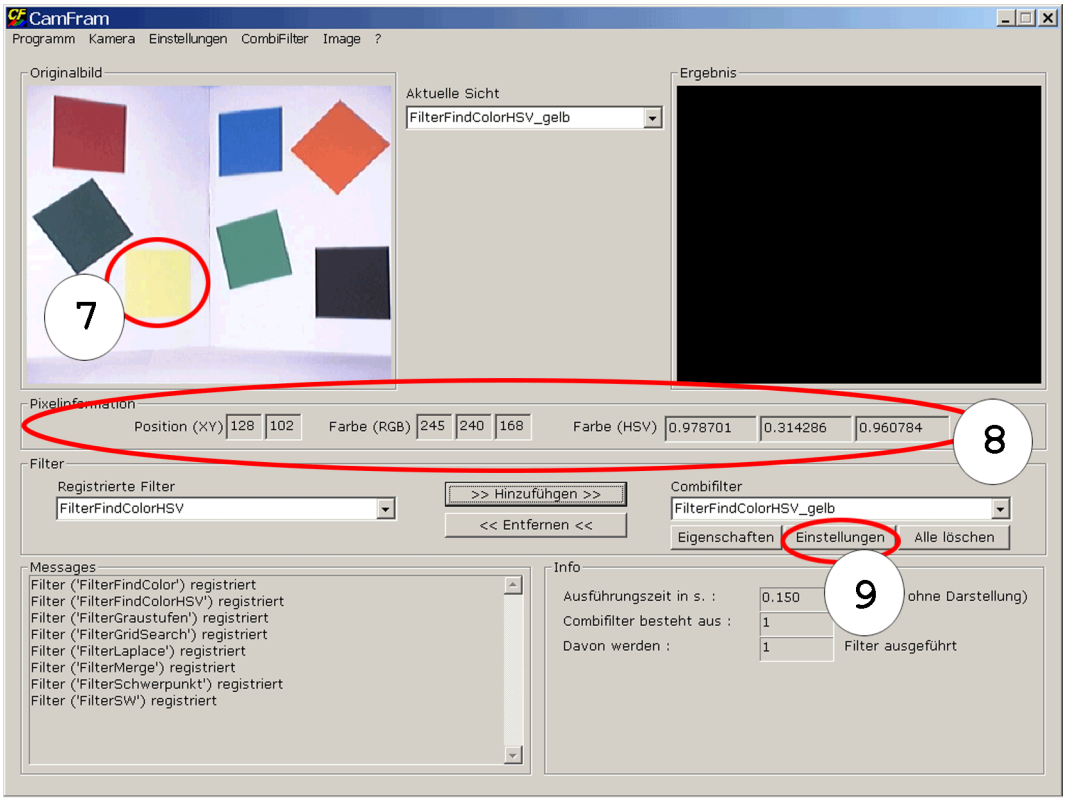

Abbildung 4.16: Einen Kombifilter bauen: Schritte 7-9

8. Sobald es gemacht wird, erscheinen unter dem Fenster die Informationsdaten. Was diese Informationen bedeuten, wurde schon im Abschnitt 'Die Oberfläche' beschrieben.

9. Das Fenster, in dem die Einstellungen für die Filterinstanz vorgenommen werden können, erscheint sofort nach dem Klicken des Buttons 'Einstellungen'. Dabei muss beachtet werden, dass im Combobox 'Combifilter' der Name der Filterinstanz steht, für die die Einstellungen vorgenommen werden sollen.

10. Um die Eintipparbeit zu minimieren, besteht die Möglichkeit, alle Werte, die zu dem aktuellen Filter gehören, aus der Testumgebung zu übernehmen. Jeder Dialog für die Filtereinstellungen hat die Möglichkeit, die Informationen, die für ihn relevant sind aus der Testumgebung zu holen, vorausgesetzt, dass die Testumgebung diese Informationen bereitstellt. Mit dem Klicken des Buttons 'Übernehmen' werden die Werte geholt.

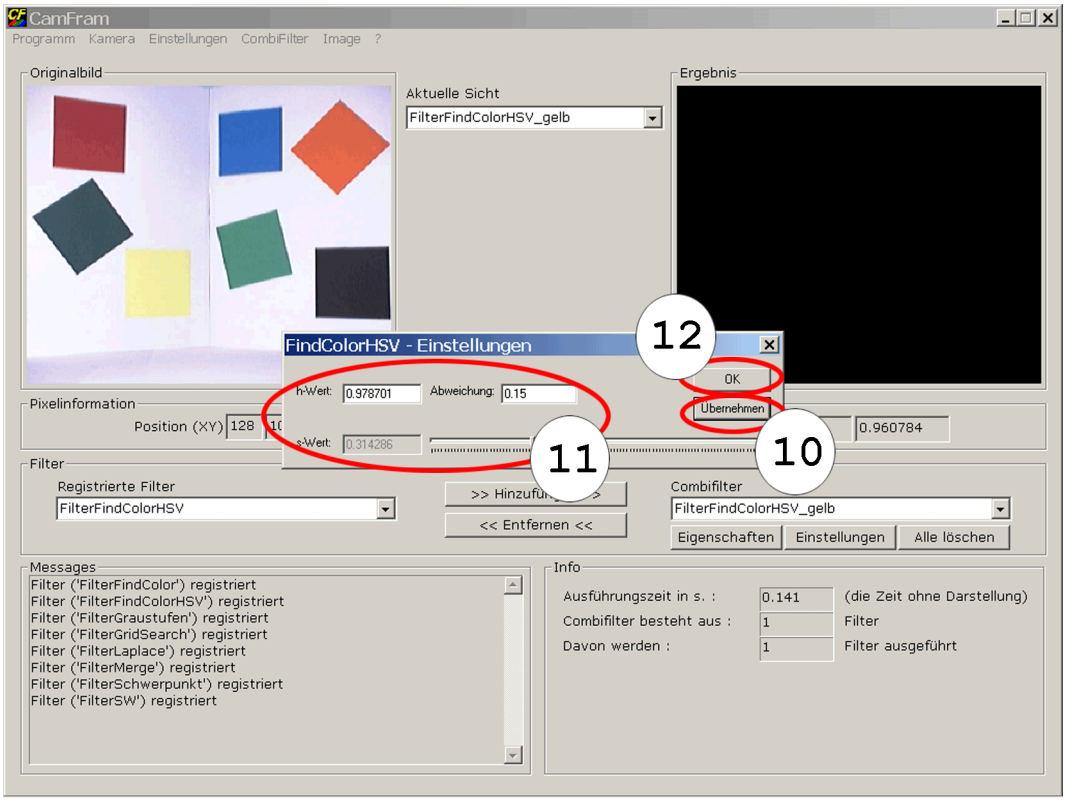

Abbildung 4.17: Einen Kombifilter bauen: Schritte 10-12

11. Die angezeigten Werte können selbstverständlich von Hand korrigiert werden. Das ist meistens der Fall, wenn die Feineinstellungen vorgenommen werden müssen.

12. Mit dem 'OK' -Button werden die Einstellungen bestätigt.

13. Jetzt weiß unsere Filterinstanz, welche Farbe gesucht werden soll. Falls das Ergebnis nicht zufriedenstellend sein sollte, können die Schritte sieben bis zwölf noch ein Mal wiederholt werden.

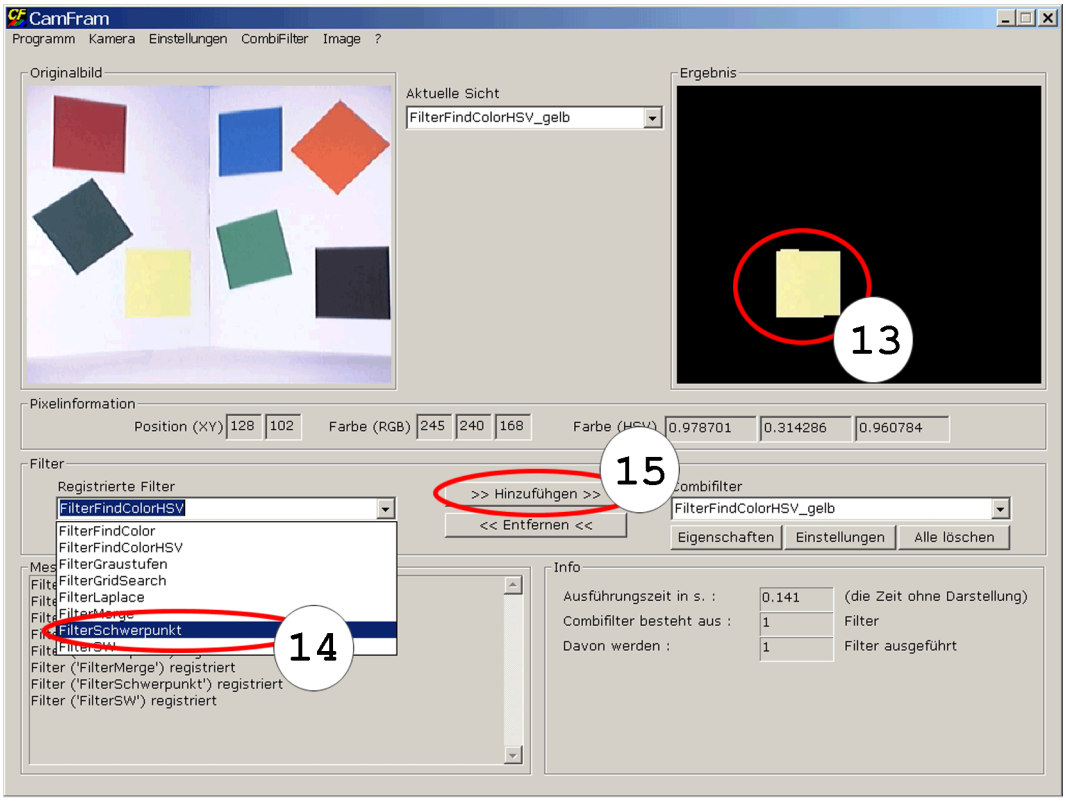

Abbildung 4.18: Einen Kombifilter bauen: Schritte 13-15

14. Als nächstes soll der Schwerpunkt dieser Farbe ermittelt werden. Dafür muss aus der Combobox 'Registrierte Filter' der Eintrag 'FilterSchwerpunkt' ausgewählt werden.

15. Mit dem Button 'Hinzufügen' erscheint das schon bekannte Fenster, in dem die Entscheidungen über den Namen und den Parent getroffen werden müssen.

16. Rechts in diesem Fenster stehen zu diesem Zeitpunkt schon zwei Einträge: 'GRABBER' und 'FilterFindColorHSV_gelb'. Für den

Schwerpunkt-Filter soll als Parent die Filterinstanz 'FilterFindColor-HSV_gelb' dienen (Abbildung4.19).

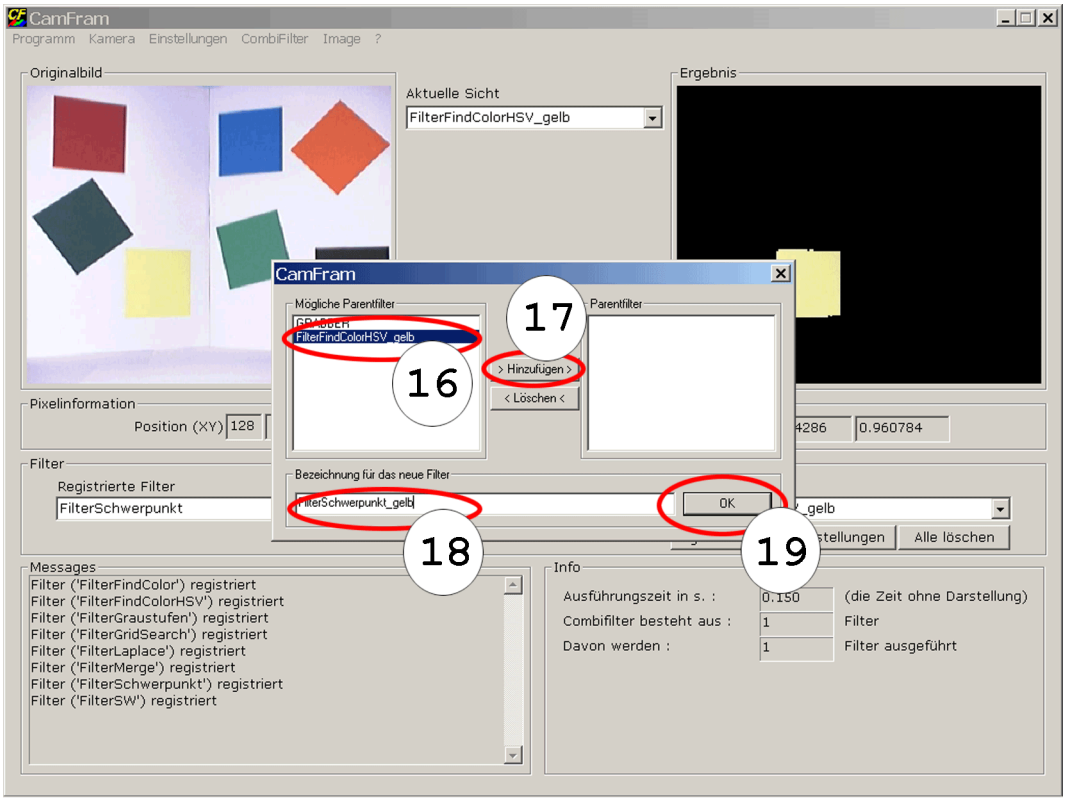

Abbildung 4.19: Einen Kombifilter bauen: Schritte 16-19

17. Mit dem Klicken auf den Button 'Hinzufügen', wird die markierte Filterinstanz für unseren Filter als Parent übernommen.

18. Auch für diese Filterinstanz gilt, den Namen möglichst aussagekräftig zu machen. Der von der Testumgebung vorgeschlagene Name 'Filter-Schwerpunkt_' wird deswegen um 'gelb' erweitert. Diese Erweiterung soll darauf hinweisen, dass diese Filterinstanz auf gelbe Farbe angesetzt ist.

19. Mit dem Klicken auf den Button 'OK' wird unser Kombifilter um einen neuen Filter ('FilterSchwerpunkt_gelb') erweitert.

20. Danach wird sofort das Ergebnis des Filtereinsatzes sichtbar: im rechten Fenster erscheint ein weißer Kreis und sein Mittelpunkt(Abbildung

4.20). Dieser Filter bietet keine Einstellungsmöglichkeiten an, deswegen wird sein Ergebnis gleich angezeigt.

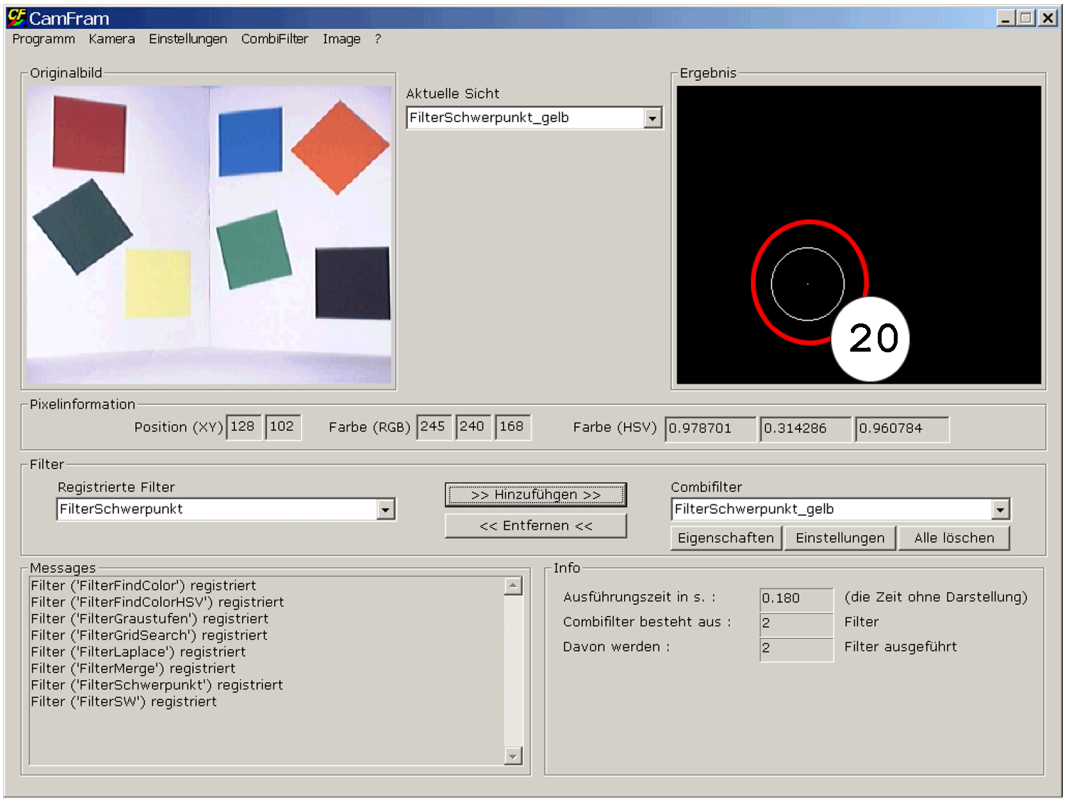

Abbildung 4.20: Einen Kombifilter bauen: Schritt 20

Nun haben wir ein Objekt mit der gelben Farbe gefunden und deren Schwerpunkt ermittelt. Jetzt muss auch das zweite Objekt bearbeitet werden. Alle bis jetzt gemachte Schritte müssen also für die Farbe 'blau' wiederholt werden (von eins bis zwanzig). Da es sich dabei um die gleiche Vorgehensweise handelt, werden diese Schritte nicht weiter erläutert. Auch hier gilt die Regel über die aussagekräftigen Namen.

21. Nachdem alle Schritte für die zweite Farbe wiederholt worden sind, muss das Ergebnisfenster ein ähnliches Bild zeigen, wie bei der ersten Farbe auch. Der Unterschied liegt nur darin, dass der Schwerpunkt jetzt für die blaue Farbe angezeigt wird (Abbildung 4.21).

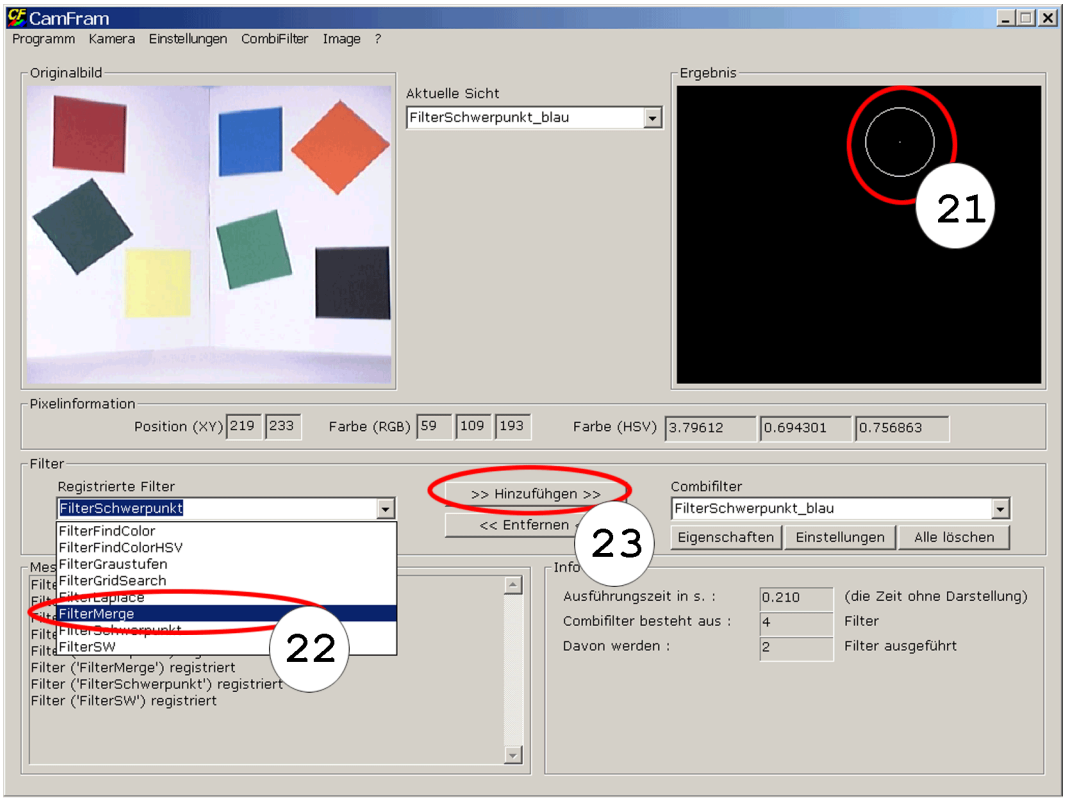

Abbildung 4.21: Einen Kombifilter bauen: Schritte 21-23

22. Um jetzt beide Ergebnisse zusammen darstellen zu können, müssen sie zusammengefasst werden. Der dafür zuständige Filter ist 'FilterMerge'. Dieser Filter wird nun aus der schon bekannten Combobox 'Registrierte Filter' ausgewählt.

23. Auch diese Filterinstanz muss vorkonfiguriert werden. Der Button 'Hinzufügen' öffnet das schon bekannte Fenster.

24. Wenn alles richtig durchgeführt wurde, müssen im Fenster 'Mögliche Parents' fünf Einträge stehen. Anders als es bei den anderen Filterinstanzen der Fall war, kann dieser Filter gleichzeitig mit mehreren

Parents umgehen. Es werden zwei Filterinstanzen ausgewählt, die das Ergebnis (Die Suche des Schwerpunktes) darstellen. Im Beispiel sind es 'FilterSchwerpunkt_gelb' und 'FilterSchwerpunkt_blau'.

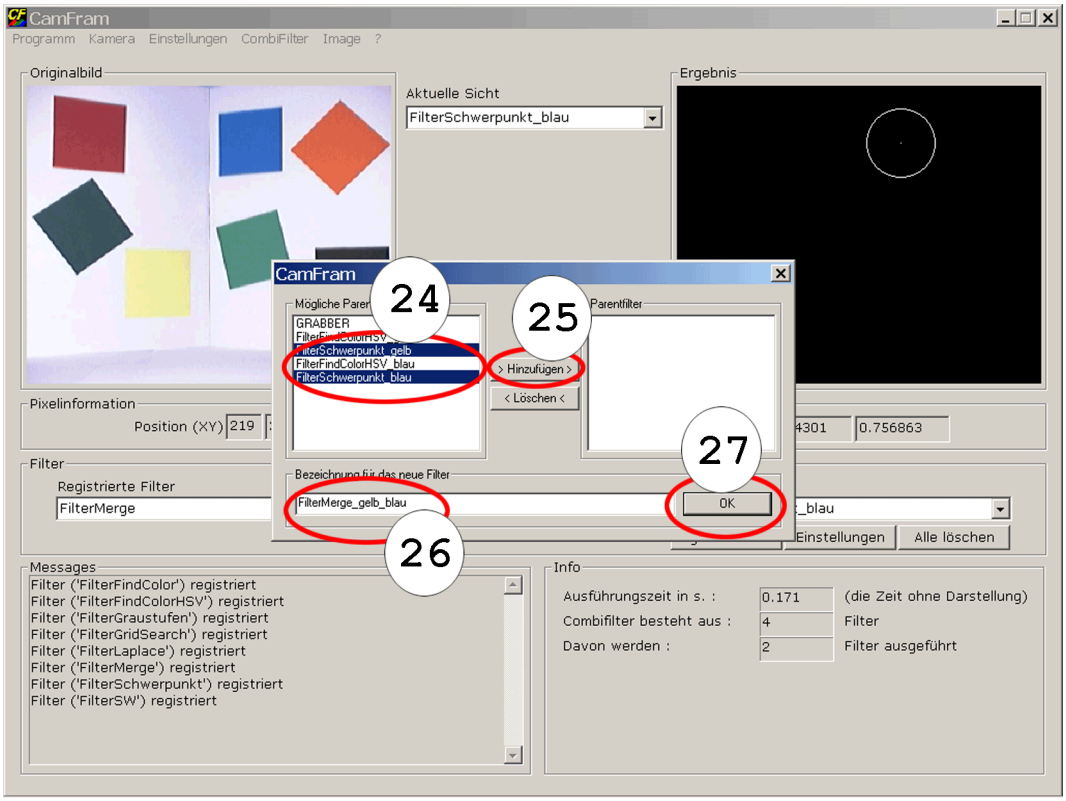

Abbildung 4.22: Einen Kombifilter bauen: Schritte 24-27

25. Die markierten Filterinstanzen werden als Parents für den aktuellen Filter übernommen. Im Allgemeinen spielt es eine Rolle, in welcher Reihenfolge die Filterinstanzen als Parents eingefügt werden, aber für unseren Merge-Filter ist es ohne Bedeutung.

26. Dem Filter wird ein Name gegeben: 'FilterMerge_gelb_blau'.

27. Mit dem Klicken auf den Button 'OK' werden alle Einstellungen für den Merge-Filter übernommen und er selbst in den Kombifilter eingefügt.

28. Wenn das 'Ergebnis'-Fenster zwei Kreise, jeweils mit dem Punkt in der Mitte, anzeigt, heißt es, dass alles erfolgreich durchgeführt wurde. Nun kann dieser Filter abgespeichert werden und immer wieder, wenn er

gebraucht wird, geladen werden. Abhängig von der Beleuchtung müssen die Einstellungen dann angepasst werden.

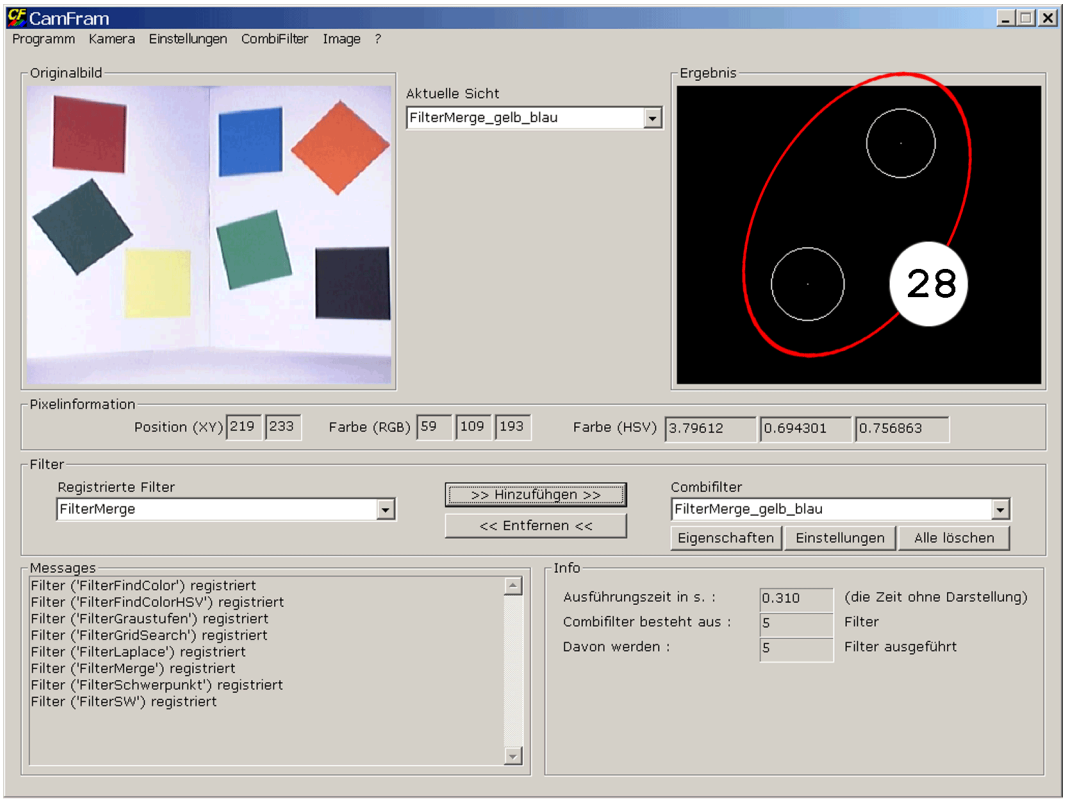

Abbildung 4.23: Einen Kombifilter bauen: Schritt 28

Das oben beschriebene Beispiel demonstriert, wie Kombifilter zusammengesetzt werden können. Mit unterschiedlichen Einstellungen entstehen sehr viele Kombinationmöglichkeiten, so dass vieles ausprobiert werden kann.

4.4.3 Das Testen eines Filters

Das Testen eines neu entwickelten Filters kann viel besser durchgeführt werden, wenn das Ergebnis visualisiert werden kann. Genau diese Aufgabe erfüllt die Testumgebung. Im Grunde genommen unterscheidet sich das Testen nicht von dem oben schon beschriebenen Bauen von Kombifiltern. Viel interessanter ist die Frage, wie ein neuer Filter entwickelt werden kann. Diese Frage wird im Abschnitt 4.6 ausführlich beantwortet.

4.5 Die Frameworkerweiterungen

Die Projekte des Frameworks können um weitere neue Filter und Grabber erweitert werden. Wie und wo diese Komponenten entwickelt werden können, beantworten die nächsten Abschnitte dieser Arbeit.

4.5.1 Neue Filter

Die Implementierung aller Filter[9] geschieht im Projekt 'CamFram_Filter'. Das Klassendiagramm 4.24 demonstriert die Sichtbarkeit dieses Projektes nach Außen. Die graumarkierten Klassen sind abstrakt und befinden sich im Projekt 'CamFram_Kernel'. Die konkreten Filter- sowie Einstellungsklassen sind dabei nicht dargestellt. Obwohl im Projekt alle Filter implementiert

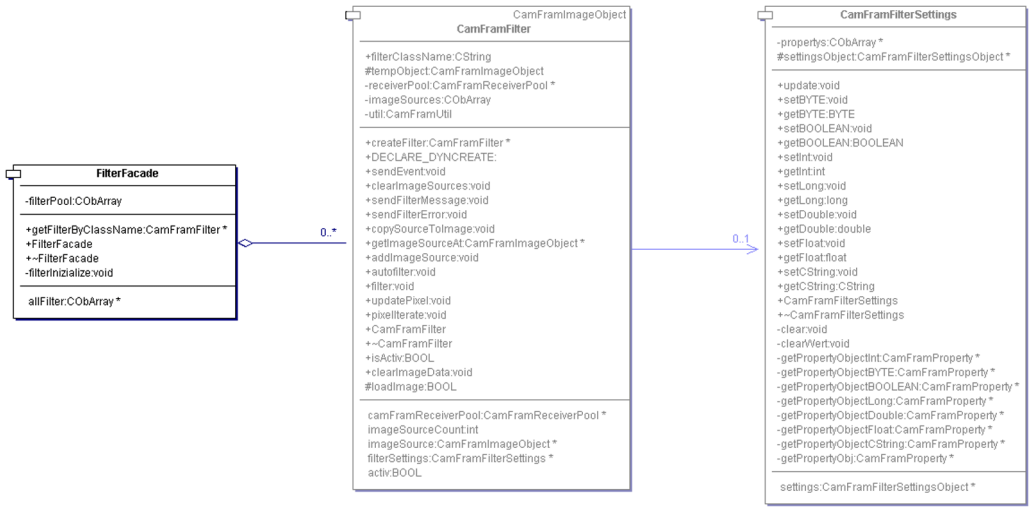

Abbildung 4.24: Filterfacade

werden, sind sie nach Außen nicht sichtbar. Die Klasse 'FilterFacade' ist ein Stellvertreter für alle Filter im 'CamFram_Filter'-Projekt. Für die Implementierung bedeutet es, dass ein neu entwickelter Filter mit einer neuen Version der dll-Datei sofort für alle Projekte verfügbar wird.

Alle Filter, die irgendwelche Einstellungsmöglichkeiten anbieten, müssen mit Hilfe der Testumgebung einstellbar sein. Dafür sind spezielle Einstellungsdialoge notwendig. Das bedeutet, dass das 'CamFram'-Projekt auch angepasst werden muss.

[9]Wie ein neuer Filter implementiert werden kann, wird im weiteren detaillierter besprochen.

Eine Filtererweiterung bewirkt dabei, dass die gesamte Funktionalität der Testumgebung erweitert wird.

4.5.2 Neue Grabber

Ganz anders ist es bei dem Grabber. Falls es notwendig wird, eine Kamera anzuschließen, die keine 'VfW' - Schnittstelle zur Verfügung hat, muss ein neuer Grabber für diese Kamera implementiert werden. Dabei ist zu beachten, dass eine neu erstellte Klasse die 'alte' Grabberklasse komplett ersetzen muss. Das bedeutet etwas mehr Verwaltungsaufwand, ermöglicht aber eine reibungslose dll-Ersetzung. Da der Kamerawechsel in der Regel nicht oft durchgeführt wird, hat die dll-Ersetzung eine größere Gewichtung als der Verwaltungsaufwand.

4.6 Einen neuen Filter erstellen

Dieser Abschnitt ist in einer HowTo-Form aufgebaut und dient zum Verständnis des Aufbaus von neuen Filtern[10].

4.6.1 Das Ziel

Das Ziel ist es, einen neuen Filter zu implementieren und mit der Testumgebung ihn zu testen. Dabei werden folgende Bereiche praktisch bearbeitet:

- Die Implementierung einer neuen Filterklasse
- Die Implementierung einer neuen Einstellungsklasse
- Die Implementierung eines neuen Dialoges für die Einstellungen
- Die Einbindung allen neuen Klassen ins Framework

4.6.2 Die Aufgabenstellung

Es soll ein Filter implementiert werden, der alle Farbwerte invertieren kann (der Inverter). Dabei soll es möglich sein, jeden einzelnen Farbkanal (R,G und B) getrennt voneinander zu invertieren. In einem Dialogfenster soll es möglich sein, diese Farbkanäle für die Invertierung aktivieren bzw. deaktivieren zu lassen.

[10]Der vollständiger Code für diesen Filter befindet sich auf der beigelegten CD-ROM

4.6.3 Die Implementierung

4.6.3.1 Die erste Überlegung

Bevor mit der Implementierung angefangen werden kann, muss zuerst geklärt werden, was genau unter der Invertierung der Farbe technisch zu verstehen ist. Wie schon erwähnt wurde, arbeitet das Framework mit RGB-Farbraum. Das heißt, dass jeder Farbpunkt des Bildes aus drei Farbkanälen besteht: R steht für Rot, G steht für Grün und B steht für Blau. Jeder Kanal kann einen beliebigen Wert im Bereich von 0 bis 255 annehmen.

Das Invertieren bedeutet nichts anderes als Subtraktion jedes Farbkanals von seinem maximal möglichen Wert. Wenn also ein Farbpunkt invertiert werden soll, müssen die neuen Werte der Farbkanäle folgendermaßen berechnet werden:

- R(neu) = 255 - R(alt)
- G(neu) = 255 - G(alt)
- B(neu) = 255 - B(alt)

Die Anforderung, jeden Farbkanal einzeln invertieren zu können, ist dann auch leicht zu verstehen. Es wird nur der Farbkanal invertiert, der dafür aktiviert wurde. Mit dem Aktivieren ist z.B. eine boolesche Variable gemeint, die im Fall der Aktivierung den Wert 'TRUE' bekommt.

Jetzt, da das Prinzip der Filterfunktion verständlich ist, kann zu der Implementierung übergegangen werden.

4.6.3.2 Die Reihenfolge der Implementierung

Was also muss implementiert werden?

- eine Filterklasse;
- eine Einstellungsklasse;
- ein Dialog für die Testumgebung, damit die Einstellungen vorgenommen werden können.

Da die zu implementierenden Klassen andere schon voraussetzen, ist es sinnvoll eine bestimmte Reihenfolge zu berücksichtigen. Die Filterklasse nutzt die Einstellungen, um die Entscheidung treffen zu können, ob ein bestimmter Farbkanal invertiert sein sollte. Der Dialog verändert die Einstellungen, braucht also auch die Einstellungsklasse. Um den Dialog testen zu können, muss er in der Testumgebung aufgerufen werden. Es ist aber nur

dann möglich, wenn dieser Dialog gebraucht wird. Das ist der Fall, wenn die Einstellungen für einen konkreten Filter verändert werden müssen. Also braucht der Dialog indirekt auch die Filterklasse. Aus diesen Überlegungen ist die Reihenfolge der Implementierung erkennbar:

1. Die Einstellungsklasse implementieren
2. Die Filterklasse implementieren
3. Der Dialog für die Einstellungen implementieren

4.6.3.3 Die Einstellungsklasse implementieren

Die Einstellungsklassen befinden sich im 'CamFram_Filter'-Projekt. In diesem Projekt wird eine neue Klasse erstellt, die von der 'CamFramFilterSettings'-Klasse abgeleitet werden muss. Dabei ist zu beachten, dass die neue Klasse eine Klasse vom Typ 'Allgemeine Klasse' ist. Der Name der neuen Einstellungsklasse soll folgende Struktur aufweisen: 'FilterSettings' + Funktion der Filter. In unserem Fall soll die Klasse also 'FilterSettingsInverter' heißen. Der Assistent erstellt zwei Dateien: eine Header-(.h) und eine Implementierungsdatei(.cpp). Die beiden Dateien müssen angepasst werden. Die Abbildung 4.25 demonstriert die notwendigen Anpassungen der Header - Datei. Die grau-markierten Elemente müssen eingefügt werden.

```
// FilterSettingsInverter.h: Schnittstelle für die Klasse FilterSettingsInverter.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTERSETTINGSINVERTER_H__A3A6434E_4D41_4A0A_B685_56AEB1002A93__INCLUDED_)
#define AFX_FILTERSETTINGSINVERTER_H__A3A6434E_4D41_4A0A_B685_56AEB1002A93__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CamFramFilterSettings.h"

class FilterSettingsInverter : public CamFramFilterSettings
{
public:
    FilterSettingsInverter();
    virtual ~FilterSettingsInverter();

    void update();
    BOOL invertR;
    BOOL invertG;
    BOOL invertB;

};

#endif // !defined(AFX_FILTERSETTINGSINVERTER_H__A3A6434E_4D41_4A0A_B685_56AEB1002A93__INCLUDED_)
```

Abbildung 4.25: FilterSettings: Die Anpassung der Header-Datei

- #include CamFramFilterSettings.h Da unsere Klasse von der Klasse 'CamFramFilterSettings' erbt, muss diese Klasse natürlich sichtbar sein. Das wird mit der #include-Anweisung erreicht.

- Die Oberklasse schreibt vor, dass die Methode 'update()' überschrieben werden muss. Die Funktionalität dieser Methode wurde bereits im Kapitel 'Design' beschrieben. Zuletzt müssen die drei booleschen Variablen deklariert werden, die für die Entscheidung, ob ein Farbkanal invertiert werden soll, zuständig sind.

Jetzt muss die Implementierungsklasse(.cpp) angepasst werden. Die Abbildung 4.26 zeigt die Bereiche der cpp-Datei (grau markiert), die eingefügt werden müssen.

```
// FilterSettingsInverter.cpp: Implementierung der Klasse FilterSettingsInverter.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "FilterSettingsInverter.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Konstruktion/Destruktion
//////////////////////////////////////////////////////////////////////

FilterSettingsInverter::FilterSettingsInverter(){
    invertR = TRUE;
    invertG = TRUE;
    invertB = TRUE;
    setBOOLEAN("FilterSettingsInverter_invertR",invertR);
    setBOOLEAN("FilterSettingsInverter_invertG",invertG);
    setBOOLEAN("FilterSettingsInverter_invertB",invertB);
}

FilterSettingsInverter::~FilterSettingsInverter(){
}

void FilterSettingsInverter::update(){
    invertR = getBOOLEAN("FilterSettingsInverter_invertR");
    invertG = getBOOLEAN("FilterSettingsInverter_invertG");
    invertB = getBOOLEAN("FilterSettingsInverter_invertB");
}
```

Abbildung 4.26: FilterSettings: Die Anpassung der cpp-Datei

- Im Konstruktor müssen den booleschen Variablen die Anfangswerte zugewiesen werden. Da alle Variablen den Wert 'TRUE' bekommen, bedeutet es für den Filter, dass am Anfang alle Farbkanäle invertiert werden. Zu beachten sind auch die unteren drei Zeilen im ersten markierten Bereich. Diese sind wichtig für die Kommunikation zwischen dieser

Klasse und dem Dialog, mit dessen Hilfe die Einstellungen geändert werden können.

- Die in der Header-Datei deklarierte Methode 'update()' muss implementiert werden. Auch hier wird auf das Kapitel 'Design' verwiesen. Dort wurde beschrieben, wie diese Methode auszusehen hat.

Damit ist die Implementierung der Einstellungsklasse beendet und es kann zur Implementierung der Filterklasse übergegangen werden.

4.6.3.4 Die Filterklasse implementieren

Auch die Filterklassen befinden sich im 'CamFram_Filter'-Projekt. Der Name der neu zu entwickelnden Klasse soll folgende Form aufweisen: 'Filter' + Funktionalität des Filters. Für unsere Klasse lautet der Name: 'FilterInverter'. Auch diese Klasse ist vom Typ 'Allgemeine Klasse' und wird mit Hilfe des Assistenten erstellt. Als Oberklasse soll die Klasse 'CamFramFilter' eingegeben werden. Bevor mit der Implementierung des Filters angefangen werden kann, müssen die beiden Dateien angepasst werden. Die Abbildung 4.27 zeigt die Zeilen in der Header-Datei, die eingefügt werden müssen.

```
// FilterInverter.h: Schnittstelle für die Klasse FilterInverter.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTERINVERTER_H__8DE8F615_45B1_455C_B857_8D9E26743EEA__INCLUDED_)
#define AFX_FILTERINVERTER_H__8DE8F615_45B1_455C_B857_8D9E26743EEA__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CamFramFilter.h"
#include "FilterSettingsInverter.h"

class FilterInverter : public CamFramFilter {

DECLARE(FilterInverter);

public:
    FilterInverter();
    virtual ~FilterInverter();

    void filter();
    void updatePixel(int paramX,int paramY);

private:
    FilterSettingsInverter *settings;

};

#endif // !defined(AFX_FILTERINVERTER_H__8DE8F615_45B1_455C_B857_8D9E26743EEA__INCLUDED_)
```

Abbildung 4.27: Filter: Die Anpassung der h-Datei

- #includes - Anweisungen machen die Oberklasse sowie die Einstellungsklasse für den Filter sichtbar.
- DECLARE(FilterInverter); - Dieses Macro übernimmt alle notwendigen Deklarationen für den Filter. Dabei bekommt der Filter unter anderem die Fähigkeit, neue Instanzen von sich selbst zu erzeugen.
- Die Methoden : 'filter()' und 'updatePixel(int paramX,int paramY)' sind von der Oberklasse vorgeschrieben und müssen überschrieben werden.
- Zuletzt muss ein Zeiger auf die Einstellungsklasse deklariert werden.

Bevor zur Anpassung der Implementierungsklasse übergegangen wird, muss hier etwas über die Art der Filterausführung berichtet werden. Grundsätzlich gilt, dass die Methode 'filter()' den gesamten Filter kapseln soll. Diese Methode wird vom Framework aufgerufen. Da viele Filter über alle Pixel iterieren müssen, wird diese Iteration vom Framework übernommen. Das Sequnzdiagramm 4.28 demonstriert die Kommunikation zwischen den beteiligten Objekten.

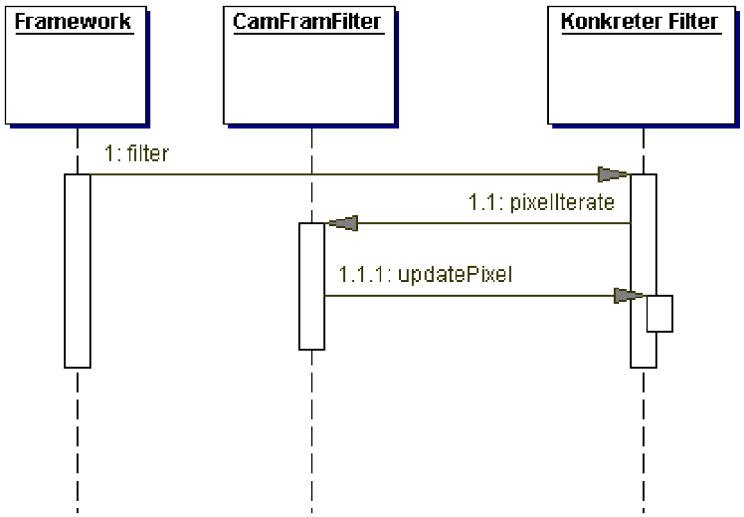

Abbildung 4.28: Der Aufruf der filter-Methode

Zu beachten ist, dass die Klasse 'CamFramFilter', die Oberklasse des konkreten Filters ist. Wenn also die Iteration über alle Pixel vom Framework übernommen werden soll, dann muss die filter()-Methode eine einzige Zeile enthalten: pixelIterate();. Diese Methode wird in der Oberklasse aufgerufen

und implementiert die Iteration über alle Koordinaten des Bildes. Für jede gültige Pixelkoordinate wird die Methode 'updatePixel(int x, int y)' aufgerufen, die in der konkreten Filterklasse implementiert werden muss. Diese Methode übernimmt dann die eigentliche Filterarbeit für einen Pixel, dessen Koordinaten als Parameter übergeben werden.

Falls die Iteration nicht erwünscht ist, oder es eine andere Art der Iteration sein soll, dann muss alles in der Methode 'filter()' implementiert werden und die Methode 'updatePixel(int x, int y)' muss leer implementiert bleiben.

Die Abbildung 4.29 zeigt die Zeilen, die eingefügt werden müssen.

```
#include "stdafx.h"
#include "FilterInverter.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif
//////////////////////////////////////////////////////////////////////
// Konstruktion/Destruktion
//////////////////////////////////////////////////////////////////////
IMPLEMENT(FilterInverter)

FilterInverter::FilterInverter(){
    filterClassName = "FilterInverter";
    setImageObjectName(filterClassName);
    settings = new FilterSettingsInverter;
    setFilterSettings(settings);
}

FilterInverter::~FilterInverter(){
    delete getFilterSettings();
}

void FilterInverter::filter(){
    pixelIterate();
}

void FilterInverter::updatePixel(int paramX,int paramY){
    BYTE newR=0;
    BYTE newG=0;
    BYTE newB=0;

    newR = getImageSource()->getPixelR(paramX,paramY);
    newG = getImageSource()->getPixelG(paramX,paramY);
    newB = getImageSource()->getPixelB(paramX,paramY);

    if(settings->invertR){ newR = 255-newR; }
    if(settings->invertG){ newG = 255-newG; }
    if(settings->invertB){ newB = 255-newB; }

    setPixel(paramX,paramY,newR,newG,newB);
}
```

Abbildung 4.29: Filter: Die Anpassung der cpp-Datei

- Als erstes muss ein IMPLEMENT - Macro eingefügt werden. Dieses

Macro implementiert die notwendige Funktionalität, die mit Hilfe des DECLARE-Macro in der header-Datei definiert wurde.

- Der Konstruktor muss erweitert werden. Es wird ein Name für den Filter gesetzt und eine Instanz der Einstellungsklasse erzeugt.
- Der Dekonstruktor muss dafür sorgen, dass die Einstellungsklasse - Instanz wieder zerstört wird.
- Da dieser Filter über alle Pixel iterieren muss, wird die Funktionalität des Frameworks genutzt. Die filter()-Methode ruft die einzige Methode 'pixelIterate()' auf.
- Die Methode 'updatePixel(..)' implementiert das Invertieren für einen Pixel. Dabei wird jeder Farbkanal des Pixels einzeln ausgelesen und die Invertierung gemacht, wenn diese erwünscht ist. Am Ende werden neue Farbkanalwerte in ein neues Pixel geschrieben.

Damit ist die Implementierung des Filters abgeschlossen. Hier wird deutlich, wie das Framework die Implementierungsarbeit abnimmt. Auch kompliziertere Filter lassen sich damit viel leichter implementieren. Der letzte Schritt, welcher noch durchgeführt werden muss, ist das Registrieren des neu zu entwickelnden Filters in der 'FilterFacade'-Klasse. Zur Erinnerung muss gesagt werden, dass die Testanwendung, sowie alle anderen Anwendungen, die auf dem Framework basieren, die Filterklassen bzw. deren Einstellungsklassen nicht kennen. Die einzigen Informationen, die zur Verfügung stehen, sind die Filternamen und die Tatsache, dass jede Instanz die 'filter()'-Methode bearbeiten kann. Alle existierenden Filter werden mit Hilfe der 'FilterFacade'-Klasse nach Außen zur Verfügung gestellt.

Das Registrieren muss an zwei Stellen durchgeführt werden.

- In der Header-Datei der 'FilterFacade'-Klasse
- In der Implementierungsklasse

In der Header-Datei muss die neue Filter-Klasse importiert werden. Die Abbildung 4.30 zeigt die Stelle, an der die #include-Anweisung eingefügt werden soll. Die Anpassung der cpp-Datei besteht ausschließlich aus der Erweiterung der 'filterInizialize()'-Methode. Die Abbildung 4.31 zeigt die notwendige Änderung.

Sobald das 'CamFram_Filter'-Projekt kompiliert ist, kann der neue Filter getestet werden. Dank der Projekteinstellungen, welche im Abschnitt 4.3 beschrieben wurden, kann gleich aus diesem Projekt gestartet werden.

```
// FilterFacade.h: Schnittstelle für die Klasse FilterFacade.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTERFACADE_H__62187AE7_871A_443B_B645_6B7AEDF78ADD__INCLUDED_)
#define AFX_FILTERFACADE_H__62187AE7_871A_443B_B645_6B7AEDF78ADD__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#include "CamFramFilter.h"
#include "FilterFindColor.h"
#include "FilterFindColorHSV.h"
#include "FilterGraustufen.h"
#include "FilterGridSearch.h"
#include "FilterLaplace.h"
#include "FilterMerge.h"
#include "FilterSchwerpunkt.h"
#include "FilterSW.h"

#include "FilterInverter.h"

#endif // _MSC_VER > 1000

class AFX_EXT_CLASS FilterFacade
{
public:
    CObArray* getAllFilter();
    CamFramFilter* getFilterByClassName(CString paramClassName);
    FilterFacade();
    virtual ~FilterFacade();


private:
    void filterInizialize();
    CObArray filterPool;

};

#endif // !defined(AFX_FILTERFACADE_H__62187AE7_871A_443B_B645_6B7AEDF78ADD__INCLUDED_)
```

Abbildung 4.30: FilterFacade: Die Anpassung der h-Datei

Die Testumgebung entnimmt sofort die veränderten Dateien aus dem 'Cam-Fram_Filter'-Projekt, so dass alle Veränderungen sofort sichtbar werden. Wenn alles richtig gemacht wurde, muss das Bild ähnlich der Abbildung 4.32 sein.

Dabei ist folgendes zu beachten: im Fenster 'Registrierte Filter' muss nun der Eintrag 'Filter ('FilterInverter') registriert' erscheinen. Nun muss ein Kombifilter mit diesem einen Filter gebaut werden (Im Kapitel 'Step by step - ein Kombifilter' wurden alle dafür notwendigen Schritte detailliert beschrieben).

Sobald der Kombifilter erstellt wurde, muss das Originalbild im 'Ergebnis'-Fenster invertiert angezeigt werden. Der Grund, warum das Ergebnis sofort sichtbar ist, liegt darin, dass in der Einstellungsklasse die 'default'-Einstellungen für die Entscheidung, ob die Farbkanälle invertiert sein müssen, alle auf 'TRUE' gesetzt wurden.Wenn jetzt versucht wird, auf den Button 'Einstellungen' zu klicken, wird eine Meldung ausgegeben, dass dieser Filter keine Einstellungsmöglichkeiten hat (Abbildung 4.33). Die Erklärung dafür

ist folgende: die Testumgebung überprüft nicht die Einstellungsmöglichkeiten der Filter, sondern die Möglichkeit, diese Einstellungen zu verändern. Da noch kein Dialog dafür implementiert wurde, wird diese Meldung angezeigt. Damit wird nun zum weiteren Abschnitt übergegangen, nämlich zur Implementierung des Dialoges für den 'Inverter'-Filter.

4.6.3.5 Den Dialog für die Filtereinstellungen implementieren

Hauptsächlich werden alle Dialoge für die Änderung der Filtereinstellungen in der Testumgebung gebraucht. Deswegen werden sie im 'CamFram'-Projekt implementiert. Auch hier ist es sinnvoll, zuerst zu überlegen, wie der Dialog aussehen soll und welche Funktionalität dabei nützlich wäre. In unserem Beispiel handelt es sich um drei boolesche Variablen. Deswegen eignet sich die 'Checkbox' besonders gut für diese Aufgabe. Es muss möglich sein, die Eingaben zu bestätigen (der 'OK'-Button) bzw. alles zu verwerfen (der 'Verwerfen'-Button). Es wäre auch sehr hilfreich, wenn die Änderungen sofort sichtbar wären, ohne dabei den Dialog schließen zu müssen. Das Sequenzdiagramm 4.34 zeigt in einer vereinfachten Form die Kommunikation zwischen den beteiligten Objekten. Folgende Punkte müssen dabei beachtet werden:

- Sobald der Dialog angezeigt wird, müssen die aktuellen Einstellungen geholt werden. Dabei werden sie noch extra gehalten, auch nachdem sie geändert wurden, für den Fall, dass der Anwender die Änderungen verwerfen möchte.
- Der Befehl 'Anwenden' übernimmt die vorgenommenen Änderungen in den Einstellungen, schließt den Dialog aber nicht, so dass das Ergebnis sofort angeschaut werden kann.
- Die Befehle 'OK' und 'Verwerfen' sind nicht in dem Sequenzdiagramm dargestellt. Beide Befehle schließen den Dialog. Der Unterschied liegt

```
void FilterFacade::filterInizialize(){
    filterPool.Add(new FilterFindColor);
    filterPool.Add(new FilterFindColorHSV);
    filterPool.Add(new FilterGraustufen);
    filterPool.Add(new FilterGridSearch);
    filterPool.Add(new FilterLaplace);
    filterPool.Add(new FilterMerge);
    filterPool.Add(new FilterSchwerpunkt);
    filterPool.Add(new FilterSW);

    filterPool.Add(new FilterInverter);
}
```

Abbildung 4.31: FilterFacade: Die Anpassung der cpp-Datei

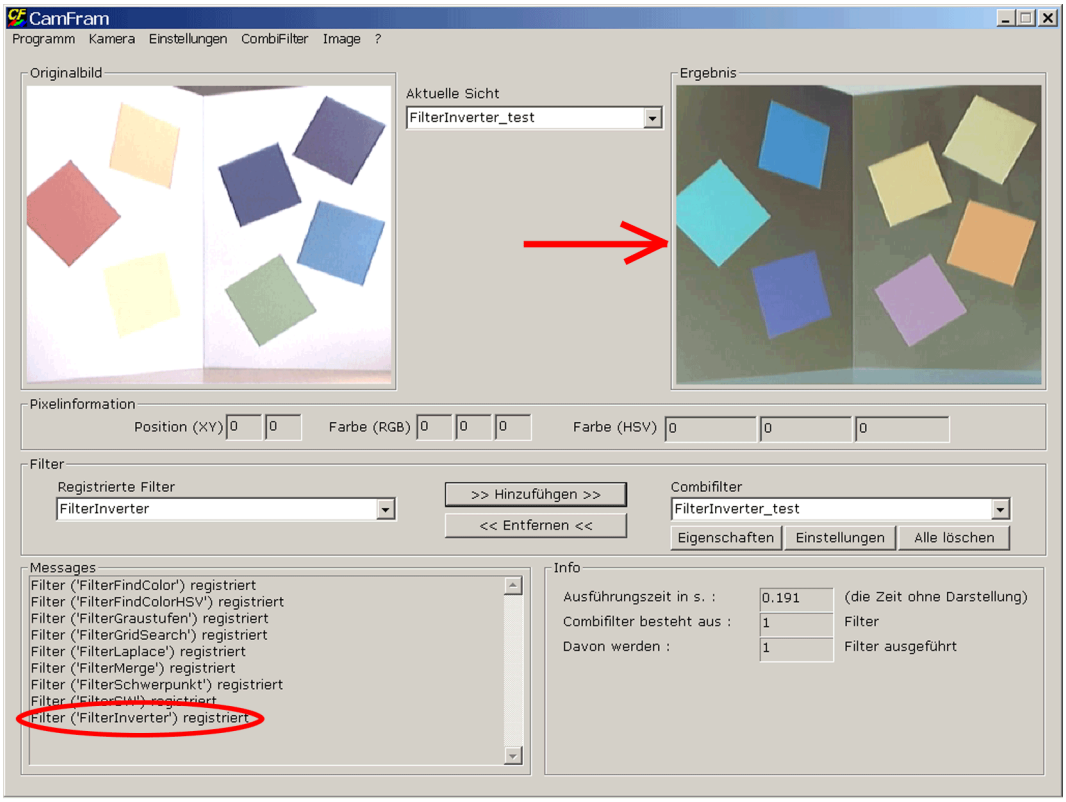

Abbildung 4.32: Test des 'Inverter'-Filters

nur darin, dass bei 'OK' alle Änderungen übernommen werden, bei 'Verwerfen' alle Einstellungen, die beim Öffnen des Dialogs vorhanden waren, wiederhergestellt werden.

Der Dialog selbst wird mit Hilfe des Formularassistenten der Visual C++ -Entwicklungsumgebung aufgebaut. Hier wird davon ausgegangen, dass der Leser mit diesem Assistent umgehen kann. Das Design des Dialoges kann beispielsweise wie in der Abbildung 4.35 aussehen.

Der Name des Dialoges muss folgendermaßen aufgebaut werden: 'Filter-Dialog' + die Filterfunktionalität. In unserem Beispiel mit dem Inverter muss der Dialog also 'FilterDialogInverter' heißen. Sobald die Dateien (die h- und cpp-Datei) erstellt werden, müssen sie angepasst werden.

- Alle Dialoge, die für die Filtereinstellungen implementiert werden, müssen von einer Klasse des Frameworks abgeleitet werden. Da es im Assistent nicht möglich ist, für den Dialog eine andere Klasse als 'CDialog' auszuwählen, muss in diesen beiden Dateien die Änderung von

Abbildung 4.33: Keine Einstellungsmoeglichkeiten

Hand vorgenommen werden. Überall muss die Klasse 'CDialog' durch 'CamFramFilterSettingsDialog' ersetzt werden.

- Die Klasse 'CamFramFilterSettingsDialog' schreibt vor, bestimmte Methoden zu überschreiben:
  - CString getFilterClassname();
  - void updateSettings();

  Die 'getFilterClassname()'-Methode muss einen Namen der Filterklasse zurückliefern, deren Einstellungen verändert werden müssen. Die 'updateSettings()'-Methode implementiert die Änderungsübernahme aus dem Dialog in die Einstellung.

- Damit der Dialog beim Erscheinen die aktuellen Werte anzeigt, muss die Methode 'OnInitDialog' angepasst werden. Wie die Methoden implementiert sind, wird in dem Quellcodeausschnitt 4.36 gezeigt.

4.6.4 Zusammenfassung der Filtererstellung

Das oben beschriebene Beispiel soll die Fähigkeiten des Frameworks im Sinne der Erweiterbarkeit demonstrieren. Das Konzept der Frameworkerweiterung bietet eine abstrakte Kapselung der Filter. So ist es im oben beschriebenen Beispiel möglich gewesen, einen neuen Filter selbst zu implementieren. Aber auch das Einbinden von fertigen Filtern stellt kein Problem dar. Das soll auch meistens der Fall sein, da der größte Teil der Filter schon längst implementiert ist und frei zur Verfügung steht. So eingebundene Filter können dann sofort in der Testumgebung mit anderen Filtern kombiniert und getestet werden.

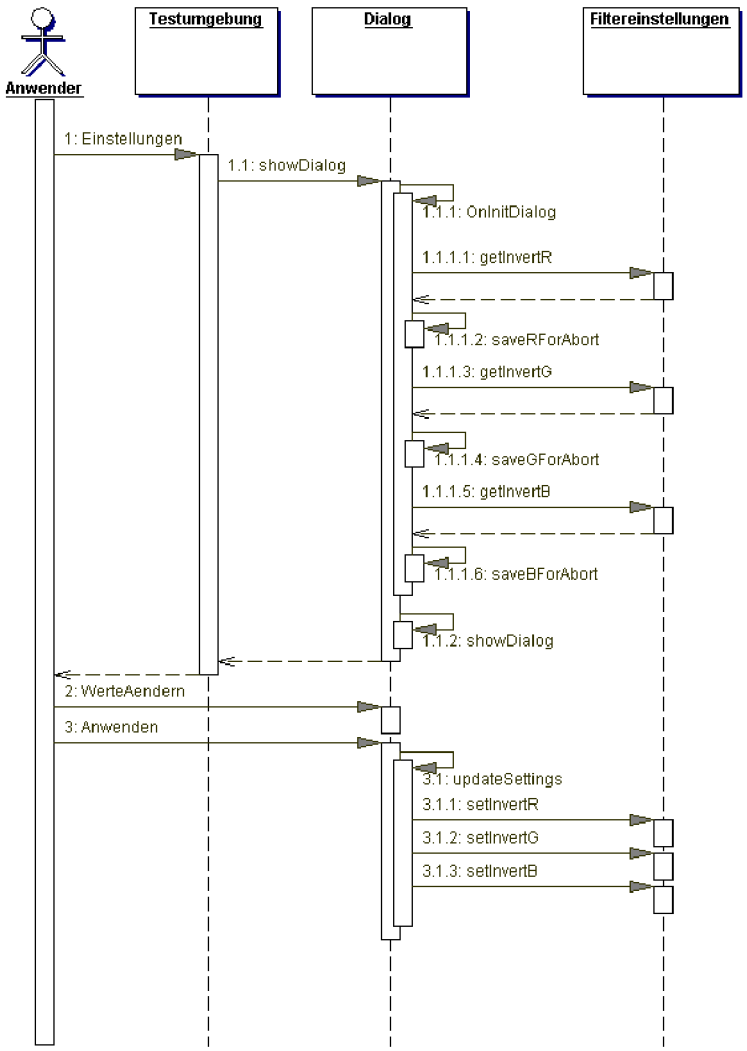

Abbildung 4.34: Die Aenderung der Einstellungen mit Hilfe des Dialoges

Abbildung 4.35: Der Dialog 'FilterInverter - Einstellungen'

```
BOOL FilterDialogInverter::OnInitDialog(){
    CamFramFilterSettingsDialog::OnInitDialog();
    if(filterSettings!=NULL){
        m_invertR = filterSettings->getBOOLEAN("FilterSettingsInverter_invertR");
        m_invertG = filterSettings->getBOOLEAN("FilterSettingsInverter_invertG");
        m_invertB = filterSettings->getBOOLEAN("FilterSettingsInverter_invertB");
        rForAbort = m_invertR;
        gForAbort = m_invertG;
        bForAbort = m_invertB;
        UpdateData(FALSE);
    }
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX-Eigenschaftenseiten sollten FALSE zurückgeben
}

void FilterDialogInverter::updateSettings(){
    UpdateData(TRUE);
    if(filterSettings!=NULL){
        filterSettings->setBOOLEAN("FilterSettingsInverter_invertR",m_invertR);
        filterSettings->setBOOLEAN("FilterSettingsInverter_invertG",m_invertG);
        filterSettings->setBOOLEAN("FilterSettingsInverter_invertB",m_invertB);
        filterSettings->update();
    }
}

CString FilterDialogInverter::getFilterClassname(){
    return "FilterInverter";
}

void FilterDialogInverter::OnCancel(){
    if(filterSettings!=NULL){
        filterSettings->setBOOLEAN("FilterSettingsInverter_invertR",rForAbort);
        filterSettings->setBOOLEAN("FilterSettingsInverter_invertG",gForAbort);
        filterSettings->setBOOLEAN("FilterSettingsInverter_invertB",bForAbort);
        filterSettings->update();
    }
    CDialog::OnCancel();
}

void FilterDialogInverter::OnOK(){
    updateSettings();
    CDialog::OnOK();
}

void FilterDialogInverter::OnButtonAnwenden(){
    updateSettings();
}
```

Abbildung 4.36: Der Dialog: der cpp-Dateiausschnitt

Kapitel 5

Evaluation

Die vorherigen Kapitel haben sich hauptsächlich mit dem Design sowie mit der Realisierung beschäftigt. Dort wurden die einzelnen Komponenten detailliert beschrieben. In diesem Kapitel geht es um das gesamte Produkt. Es stellt sich die Frage, ob alle am Anfang gestellten Anforderungen erfüllt wurden. Mit Hilfe von zwei Beispielen wird versucht, diese Frage zu beantworten. Ein Beispiel kommt aus dem ersten Szenario des Kapitels 'Anforderungen'. Es handelt sich dabei um die Diplomarbeit von Rainer Balzerowski [Bal02].Dieses Beispiel soll dann die Möglichkeit des Frameworks demonstrieren, mit Hilfe von Filterkombinationen unterschiedliche Aufgaben erledigen zu können.

Das zweite Beispiel soll eine neue Anwendung sein. Damit soll gezeigt werden, wie das Framework als Laufzeitumgebung eingesetzt werden kann.

5.1 Erstes Beispiel

Anhand von diesem Beispiel soll demonstriert werden, dass mit Hilfe der Testumgebung, basierend auf dem Framework, die Aufgaben solcher Art realisierbar sind. Als Aufgabe wurde folgendes Szenario ausgewählt: Auf dem Robocup-Spielfeld wurde ein Mindstorm-Roboter positioniert. Auf dem Roboter wurde eine zweifarbige Platte befestigt, so dass eine Farbe vorne und eine hinten war. Auch ein Puck mit einer anderer Farbe wurde auf dem Spielfeld platziert. Ziel war es, so schnell wie möglich mit Hilfe der Testumgebung die Schwerpunkte dieser drei Farben zu ermitteln. Die Abbildung 5.1 zeigt dieses Szenario.

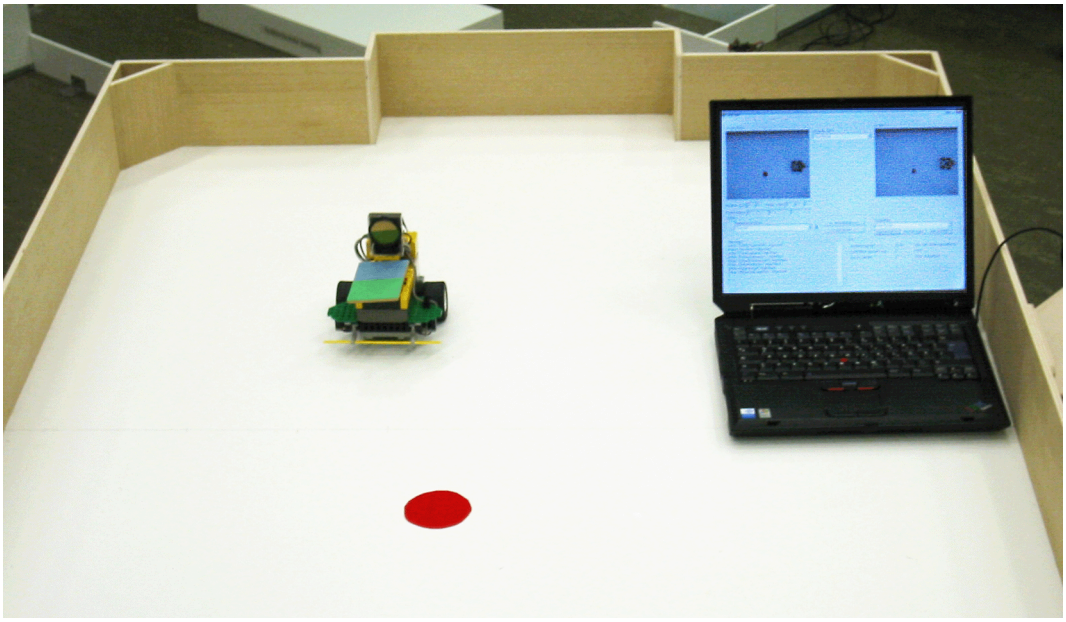

Abbildung 5.1: Die Umgebung des ersten Beispieles

5.1.1 Die Durchführung

Im Kapitel 'Realisierung' wurde bereits ein detailliertes Beispiel gegeben, wie die Kombifilter zusammengestellt werden. Da diese Aufgabe nur dahingehend anders ist, dass statt zwei Farben, wie im Kombifilter-Beispiel, drei Farben gesucht werden, wird hier die ausführliche Beschreibung alle dafür notwendige Schritte ausgelassen.

5.1.2 Das Ergebnis

Nachdem alles vorbereitet war, dauerte es ca. fünf Minuten bis alle Schwerpunkte ermittelt und angezeigt wurden. Dazu kommen auch weitere Vorteile, die das Framework mit sich bringt.

- Die Anzahl der Farben kann beliebig variiert werden.
- Der fertig gestellte Kombifilter kann abgespeichert und für unterschiedliche Lichtverhältnisse angepasst werden. Damit können für unterschiedliche Beleuchtungen verschiedene Einstellungen geladen werden.

Die Abbildung 5.2 zeigt das Ergebnis des Versuches. Im Fenster 'Ergebnis' sind die Schwerpunkte alle drei Farben zu sehen. Die Information über die

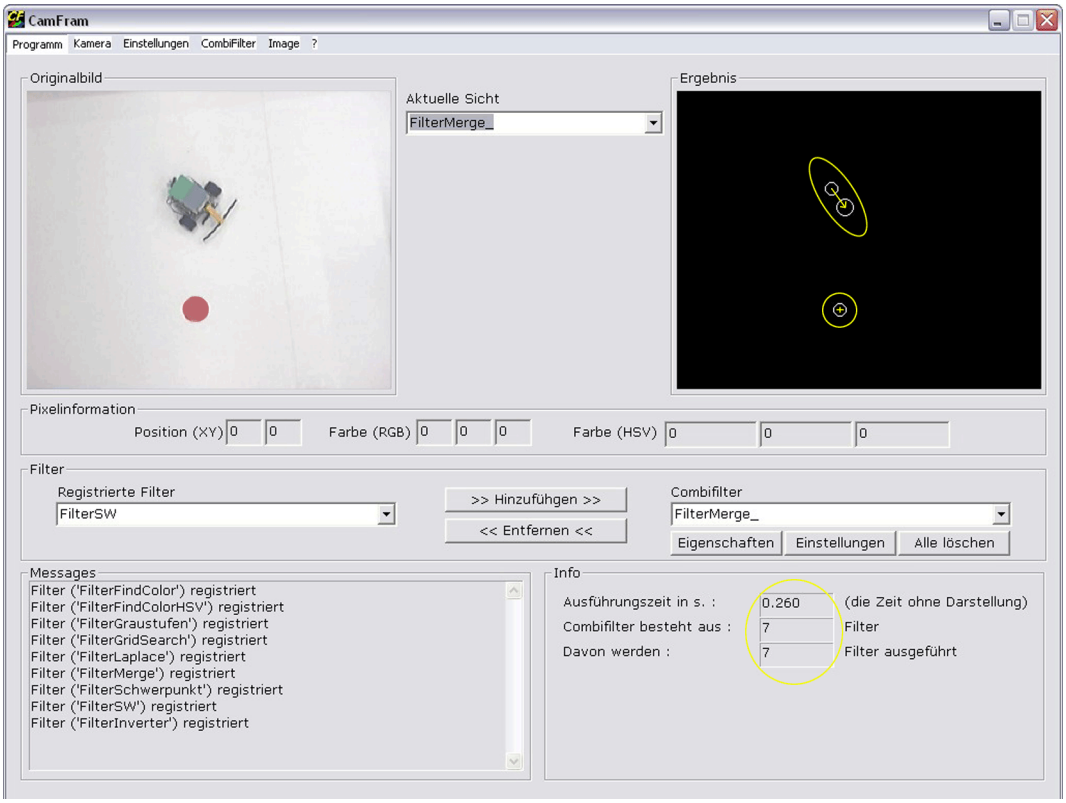

Abbildung 5.2: Das Ergebnis des Testversuches

Farbangehörigkeit zum Objekt, erlaubt auch weiteres Wissen zu gewinnen. So können nicht nur die Koordinaten des Roboters, sondern auch seine Ausrichtung ermittelt werden (In der Abbildung durch einen Pfeil dargestellt.[1]).

Interessant sind auch die quantitativen Daten, die unten im Fenster 'Info' in der oben genannten Abbildung dargestellt werden. Um diese Aufgabe bewerkstelligen zu können, müssten sieben Filter (drei FindeColorHSV-Filter, drei Schwerpunkt-Filter und ein Merge-Filter) eingesetzt werden. Die Ausführungszeit dieses Kombifilters beträgt nur 0.26 Sekunden.[2] Dabei ist auch die Auflösung der Kamera zu beachten. Die Ergebnisse wurden mit der Auflösung 352 x 288 erzielt. Für diese Aufgabenstellung hätte eine viel kleinere Auflösung (160 x 120) gereicht, aber für eine bessere Darstellung wurde die oben angegebene Auflösung eingestellt. Diese Daten beweisen, dass die

[1]Dieser Pfeil wurde nicht von der Testumgebung angezeigt, sondern für die bessere Verständigung von mir gezeichnet.

[2]Das Programm wurde auf einem Laptop mit Intel Pentium III-M 1.4 GHz und 768Mb RAM ausgeführt.

Anforderung der Echtzeit erfüllt ist und das Framework auch auf handelsüblichen Rechnern gute Zeit-Ergebnisse liefern kann.

5.2 Zweites Beispiel

Das Ziel dieses Beispiels ist die Demonstration des Frameworks als eine Laufzeitumgebung für andere Anwendungen. Gleichzeitig wird damit ein Beweis dafür durchgeführt, dass diese Anforderung erfüllt ist. Anhand des schon bekannten Schwerpunktfilters soll eine Anwendung entwickelt werden, die eine Kollision zwischen zwei unterschiedlich gefärbten Objekten erkennen.

5.2.1 Die Vorgehensweise

Wie bereits im Kapitel 'Design' berichtet wurde, besteht die Möglichkeit, einen Kombifilter der Anwendung, die auf dem Framework basiert, zu übergeben. Diese Funktionalität wird vom Framework angeboten und muss durch die Anwendung zur Verfügung gestellt werden. Um die Anwendung unabhängig von den Farben der Objekte zu machen, wird die 'Filter-Laden'-Funktionalität benutzt, und damit die Konfiguration durchgeführt. Das bedeutet, dass die Anwendung selbst die gesuchten Farben nicht kennt. Diese Information bekommt sie beim Laden des Kombifilters mit. Das Sequenzdiagramm 5.3 soll das Verhalten der Anwendung beim Starten verdeutlichen.

5.2.1.1 Die Initialisierung der Anwendung

Eine denkbare Initialisierung kann folgende Schritte aufweisen:

- Eine Instanz des Grabbers erstellen.
- Eine Instanz der FilterFacade erstellen.
- Eine Instanz des Frameworks erstellen.
- Framework initialisieren:
  - Den Grabber dem Framework übergeben.
  - Mit Hilfe von FilterFacade alle Filterinstanzen dem Framework übergeben.
  - Sich selbst(die Anwendung) im Framework registrieren.

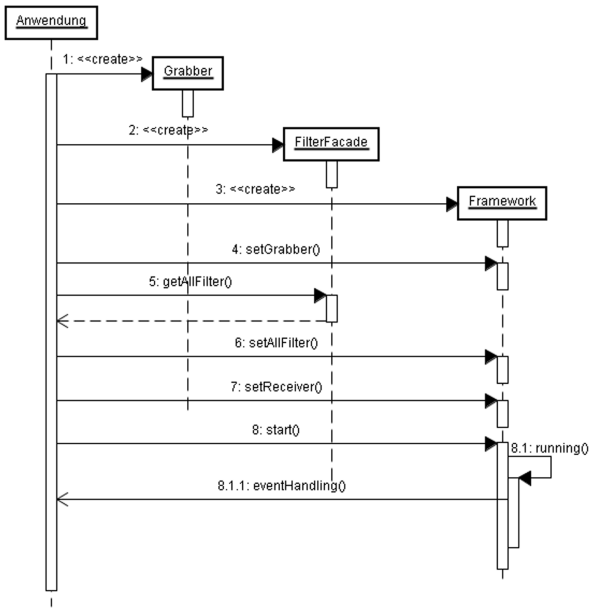

Abbildung 5.3: Startvorgang der Anwendung

- Einen Kombifilter laden. Dafür wird die Information gebraucht, wie die Datei heißt und wo sie sich befindet. In dieser Anwendung wird davon ausgegangen, dass die Datei 'filter.xml' heißt und sich in einem mit der Anwendung gleichen Verzeichnis befindet.
- Ausführung wird gestartet.

Das Komponentendiagramm 5.4 soll noch ein Mal die Zusammenhänge aller beteiligten Komponenten demonstrieren. Alle Komponenten hängen vom Framework ab. Die Erklärung liegt darin, dass das Framework die Basisklassen für die Entwicklung der Komponenten anbietet. Die Entscheidung, wie das Framework initialisiert werden soll (welcher Grabber und welche Filter), wird in der Anwendung getroffen. Diese Entscheidung ist der Grund dafür, dass die Anwendung die 'CamGrabber'- und die 'CamFilter'- Komponente referenziert und entsprechend davon abhängig ist.

5.2.1.2 Der Programmablauf

Der Programmablauf ist identisch mit dem aus der Testumgebung. Da das Framework mit Hilfe des Kombifilters konfiguriert ist und alle für die Ausführung notwendigen Komponenten hat (den Grabber und die Filter), wird die ganze Arbeit im Framework absolut selbständig durchgeführt. Die

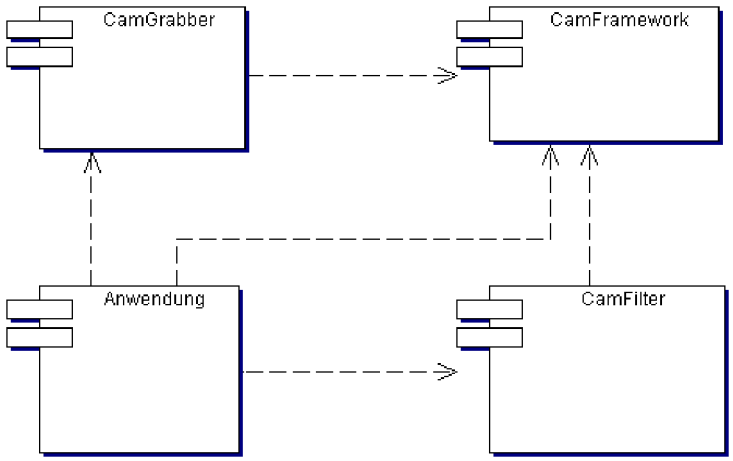

Abbildung 5.4: Die Anwendungskomponenten

Anwendung wartet lediglich auf Events (Das Event-Konzept wurde im Kapitel 'Design' beschrieben). Abhängig davon, ob das Event für die Anwendung bestimmt ist, muss sie entsprechend darauf reagieren.

In dieser Anwendung geschieht die Kommunikation vollständig über das Eventhandling. Der Schwerpunktfilter erzeugt ein spezieles Event, welches alle notwendigen Informationen enthält (den Radius, die Koordinaten des Schwerpunktes und den Namen der Filterinstanz). Damit verfügt die Anwendung über ausreichende Informationen, um die Entscheidung über die Kollision treffen zu können.

5.2.2 Die Realisierung des Kollisionsdetektors

Diese Anwendung besteht aus einem Dialog und zeigt textual die Koordinaten der gesuchten Objekte sowie deren Radius. Zusätzlich wird der Abstand zwischen beiden Objekten grafisch dargestellt. Zwei Balken, wie in der Abbildung 5.5 zu sehen ist, verändern zusätzlich ihre eigene Farbe, abhängig von dem Abstand (Grün - Gelb - Rot) zwischen den Objekten. Kurz vor der Kollision wird ein akustisches Signal ausgegeben.[3]

5.2.3 Das Ergebnis

Es war interessant, die Zeit, die für die Implementierung gebraucht wurde, zu messen. Eine Anwendung dieser Größe konnte man innerhalb von einer

[3]Der vollständige Code dieser Anwendung befindet sich auf der CD-ROM.

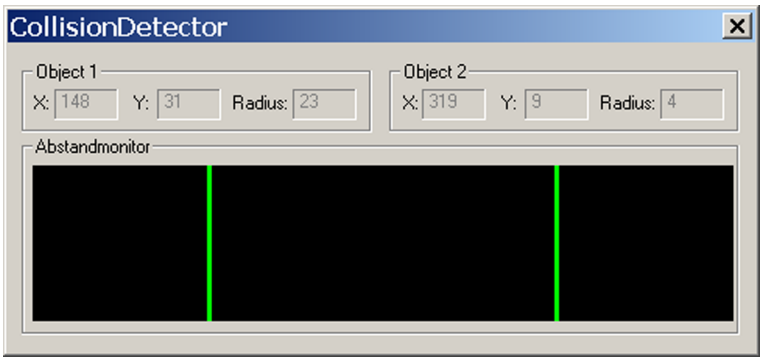

Abbildung 5.5: Das Collisiondetector

Stunde komplett implementieren und testen. Das Framework hat die Aufgabe der Laufzeitumgebung für diese Anwendung bewerkstelligt und damit seine Tragfähigkeit bewiesen.

5.3 Weitere Techniken

In oberen Abschnitten wurde detailliert beschrieben, wie eine Anwendung mit Hilfe des Frameworks entwickelt werden könnte. Dabei wurde davon ausgegangen, dass die Entwicklung mit Hilfe von Visual C++ 6.0 stattfindet und das daraus entstandene Programm eine C++ - Windows - Anwendung ist. Nun stellt sich sicherlich die Frage, ob es möglich ist, dass auch beliebige Anwendungen vom Framework profitieren können. Ob es funktioniert und wie dabei vorgegangen werden kann, wird in weiteren Abschnitten erklärt.

5.3.1 Das Problemstellungsszenario

Um die Problematik deutlicher zu machen, wird hier ein kleines Beispielszenario vorgestellt. Mit Hilfe einer Kamera sollen die Schwerpunkte einer undefinierten Anzahl von Objekten ermittelt werden. Diese Information soll für alle 'Interessenten' zur Verfügung gestellt werden. Als solche Interessenten können z.B. Roboter auftreten, die eigene Koordinaten für die unterschiedlichen Berechnungen brauchen. Obwohl das Szenario sehr stark dem GPS (Global Position System) ähnelt, gibt es zwischen diesem Szenario und GPS gravierende Unterschiede. In unserem Fall kennt die Kamera alle Objekte und stellt die gewonnene Information für die Anfragen bereit. Die Roboter müssen, falls diese Information erforderlich ist, das System selbst abfragen. Bis jetzt sieht alles unproblematisch aus. Das Problem tritt erst bei der

Aufgabeverteilung ein. Es existieren schon fertige Programme, die bestimmte Teile dieser Aufgabe erledigen sollen. Die Kommunikation zwischen den Robotern und dem Server ist mit Java realisiert. Die Erkennung der Koordinaten soll vom Framework bewerkstelligt werden. Da das Framework als eine Laufzeitumgebung ausgeführt wird, gibt es keine Möglichkeit, es in die Java-Umgebung zu integrieren. Also besteht das Problem in der Kommunikation zwischen den unterschiedlichen Komponenten, dem Framework und einer Anwendung, die die gesamte Kommunikation übernehmen soll.

5.3.2 Die mögliche Problemlösung

Obwohl diese Situation auf den ersten Blick etwas schwierig erscheinen mag, kann sie schnell unter Kontrolle gebracht werden. Der Lösungsansatz, welcher hier vorgeschlagen wird, ist keine einzelne Lösung. Damit soll nur demonstriert werden, wie Probleme solcher Art allgemein gelöst werden können. Die Abbildung 5.6 zeigt die dabei beteiligten Schichten.

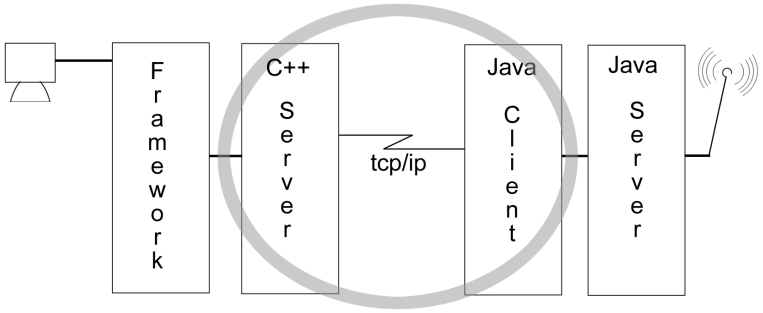

Abbildung 5.6: Anbindung einer beliebigen Anwendung

Wie schon oben erwähnt wurde, übernimmt das Framework die Koordinatenbestimmung der Objekte. Selbstverständlich muss eine neue Anwendung entworfen und implementiert werden, die das Framework initialisieren und die Ausführung steuern wird. Wie solche Anwendungen implementiert werden können, wurde bereit ausführlich beschrieben. Der entscheidende Unterschied liegt aber darin, dass diese Anwendung als ein Server implementiert werden muss.

Der Client wird im Fall des oben genannten Szenarios in Java implementiert und spielt dabei auf der anderen Seite den Kommunikationspartner. Der eigentliche Server, dessen Aufgabe daran besteht, die Daten zur Verfügung zu stellen, kennt die Seite mit dem Framework nicht. Damit ist die Technologie verborgen, was für einen leichten Austausch der Schichten sprechen kann.

Eigentliche Schichten, die implementiert werden müssen, befinden sich in der Mitte (in der Abbildung 5.6 mit dem Ellipse markiert). Die Kommunikation zwischen dem C++ - Server und dem Client wird abstrakt gehalten. Es können unterschiedliche Technologien eingesetzt werden. Hier wird nicht weiter darauf eingegangen. Ein weiterer Vorteil bei dieser Lösung liegt in der Tatsache, dass der Client sich nicht unbedingt auf dem gleichen Rechner befinden muss. Abhängig davon, wie schwer die Aufgabe für das Framework ist, wird der Rechner unterschiedlich stark ausgelastet, so dass es in vielen Fällen sinnvoll wird, den Framework-Server auf dem eigenen Rechner ausführen zu lassen.

Kapitel 6

Resümee

In diesem Kapitel soll die durchgeführte Arbeit zusammengefasst werden. Die Erkenntnisse, die während der Arbeit gewonnen wurden, sowie die Vorschläge zur Weiterentwicklung, schließen diese Arbeit ab.

6.1 Die Erkenntnisse

Das Ziel dieser Diplomarbeit war es, ein Framework mit der folgenden Funktionalität zu entwickeln:

- das Auslesen der Bildinformation aus einer Videoquelle (Grabbing) kapseln;
- für den Anwender eine einfache Art anbieten, auf die Bildinformation zuzugreifen, sowie die Daten zu verändern;
- das Kombinieren von mehreren Filtern miteinander (Kombifilter) erlauben;
- Kombifilter speichern und laden;
- alle Filter automatisch ausführen;
- den Zugriff auf jeden Bearbeitungsschritt erlauben;
- einfache Erweiterung des Frameworks um weitere Filter oder Grabber;
- die Implementierung von neuen Videokamera-basierenden Anwendungen (unter anderem die Robotersteuerung) oder Bildverarbeitung unterstützen.

Außerdem sollte eine Testumgebung realisiert werden, mit deren Hilfe das visuelle Testen der Filter möglich sein sollte. Gleichzeitig sollte sie eine Art des Kombifiltereditors darstellen, damit die Kombifilter visuell zusammengesetzt und eingestellt werden könnten.

Im Verlaufe der Frameworkentwicklung wurden als Beispiele einige Filter und ein Grabber implementiert, die für die weitere Nutzung eingesetzt werden können.

Die Collisiondetector-Anwendung diente als Beweis der Tragfähigkeit des Frameworks als eine Laufzeitumgebung.

Die mehrfach durchgeführten Tests und die Implementierung der Testanwendung haben gezeigt, dass das Framework allen gestellten Anforderungen gewachsen ist.[1] Die Entwicklungszeit solcher Art der Anwendungen wie die 'Collisiondetector' wurde mit Hilfe des Frameworks erheblich reduziert, so dass die komplexeren Systeme in kürzerer Zeit implementiert werden können.

Die durchgeführte Arbeit hat gezeigt, wie aufwendig die Entwicklung des Frameworks im Bereich der Bildverarbeitung sein kann. Je mehr Abstraktion eingebaut wurde, desto mehr stieg die Komplexität der Entwurfskonzepte an. Die Leistung der Rechner spielt immer noch eine entscheidende Rolle bei der Ausführung. Die neusten Schnittstellenformate wie USB2.0 oder IEEE1394b bringen zur Zeit keine wirkliche Performanzsteigerung, da die Filterberechnung immer noch entscheidend mehr Zeit in Anspruch nimmt, als das Hohlen der Bilder. Trotzdem haben alle Tests gezeigt, dass auch preisgünstige handelsübliche Computer sehr vernünftige Ergebnisse liefern und damit eingesetzt werden können.

6.2 Die Weiterentwicklung

Obwohl alle am Anfang gestellten Anforderungen in der Arbeit erfüllt wurden, kann an mehreren Stellen noch viel getan werden. Die Komplexität der Arbeit und die Zeit waren ständige Gegner dieses Projekt. Nicht alle Ideen konnten deswegen realisiert werden. Viele davon können als 'nice to have' bezeichnet werden, manche aber verdienen die Aufmerksamkeit. In diesem Abschnitt sollen genau solche Ideen aufgelistet werden, und damit die Richtung der Weiterentwicklung empfohlen werden.

6.2.1 Das Framework

- Das Exceptionhandling

[1] Die entwickelten Komponenten werden bereits für weitere Projekte in dem Roboterlabor der HAW eingesetzt.

  - Obwohl das Framework mit den Exceptions gut umgehen kann, wurde das Konzept des Exceptionshandling wegen des Zeitmangels nicht ausgearbeitet. Da dieses Thema aber sehr wichtig ist, steht es an der ersten Stelle der Weiterentwicklung. Besonders ist dabei die Bearbeitung der externen Fehler zu beachten. Das Framework wird mit Komponenten wie Grabber und Filter von Außen initialisiert. Alle Exceptions, die in diesen Modulen auftreten, müssen vom Framework abgefangen werden. Das Framework allein ist aber nicht in der Lage, über die Exceptionverarbeitung eine Entscheidung zu treffen. Das bedeutet, dass die Kommunikation mit der initialisierenden Anwendung notwendig ist.

- Persistenz (schema)

  - Zur Zeit wird der Kombifilter im XML-Format abgespeichert und geladen. Das Problem liegt im Laden des Kombifilters. Es wird davon ausgegangen, dass die eingegebene xml-Datei tatsächlich ein Kombifilter ist. Hier muss mit Hilfe von 'schema' oder 'dtd' die Validierung durchgeführt werden, damit die Richtigkeit der Datei festgestellt werden kann.

- Gleichzeitige Unterstützung mehreren Grabber

  - Das Konzept des Kombifilters erlaubt die gleichzeitige Nutzung mehrerer Grabber. Bis jetzt wurde das nicht realisiert. Zur Zeit ist es möglich, zur Laufzeit eine andere Kamera mit Hilfe eines einzigen Grabbers anzusprechen.

6.2.2 Der Grabber

- Umwandlungsschichten (Farbformat - RGB24)

  - Wie im Kapitel 'Design' schon erwähnt wurde, kann das Framework nur mit dem RGB24 - Format umgehen. Diese Tatsache hat die Konsequenz, dass die Bilder vom Grabber in diesem Format geliefert werden müssen. Diese Aufgabe muss jeder Grabber, der für die Kameras implementiert wird, bewerkstelligen können. Es ist empfehlenswert, wenn das Framework die unterschiedlichen Formatumwandlungen anbieten würde.

6.2.3 Die Testumgebung

- Graphische Darstellung des Kombifilters

– Oft ist es sehr hilfreich, eine Möglichkeit zu haben, den zusammengestellten Kombifilter als ein Baum anschauen zu können. Es ist zur Zeit nicht möglich, alle Zusammenhänge der einzelnen Filter gleichzeitig zu sehen. Ein möglicher Lösungsansatz wäre, diese Information aus der dem Filter entsprechenden xml-Datei auszulesen.

Anhang A

Inhalt der CD-ROM

Die zu dieser Arbeit zugehörige CD-ROM enthält folgende Verzeichnisstruktur:

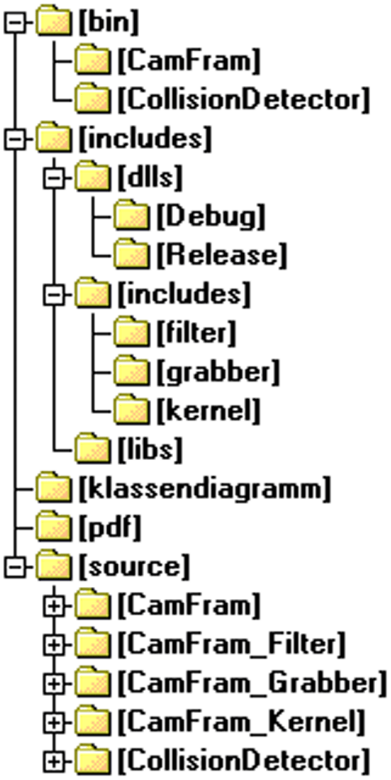

Abbildung A.1: Inhalt der CD-ROM

- 'bin' - enthält die ausführbaren Dateien. Im Verzeichnis 'CamFram'

befindet sich die Testumgebung. Die Collision-Testanwendung befindet sich im 'CollisionDetector'-Verzeichnis.

- 'includes' - enthält alle notwendigen .dll, .lib und .h Dateien für die Implementierung einer auf dem Framework basierenden Anwendung.
- 'klassendiagramm' - enthält ein gesamtes Klassendiagramm des Frameworks, der Testumgebung, aller Filter und der Grabber.
- 'pdf' - enthält diese Arbeit in pdf-Format
- 'source' - enthält den gesamten Quellcode aller Projekte. Die Unterverzeichnisse stellen dabei die Visual C++ -Projekte dar.

Literaturverzeichnis

[Kru98] Kruglinski, David : Inside Visual C++ 6.0. ISBN 3-86063-461-5. 1998

[Deu89] L. Peter Deutsch. Design reuse and frameworks in the Smalltalk-80 system. In Ted J. Biggerstaff und Alan J. Perlis, Hrsg., Software Reusability, Volume II: Applications and Experience, Seite 57-71. Addison-Wesley, Reading, MA, 1989

[Gam01] Entwurfsmuster . Elemente wiederverwendbarer objektorientierter Software. ISBN 3827318629, Addison-Wesley, März 2001

[Har00] Eliotte Rusty Harold: Die XML-Bibel. Aus dem Amerikanischen von Reinhard Engel- - Bonn: MITP-Verlag, 2000 ISBN 3-8266-0627-2

[Sch00] Hans-Jürgen Scheibl: Visual C++ 6.0 für Einsteiger und Fortgeschrittene. Wien:Hanser ISBN 3-446-19548-3

[Str00] Bjarne Stroustrup: Die C++ Programmiersprache.Addison-Wesley ISBN 3-8273-1756-8

[Bal02] Balzerowski, Rainer: Realisierung eines Webcam basierten Kamera-Systems für mobile Roboter. online. 2002. URL http://www.informatik.haw-hamburg.de/ lego/Projekte/Balzerowski/diplomarbeit-www.pdf Zugriffsdatum: 10.11.2003

[Die03] Dieckmann, Arne: Verbesserung visueller Objekterkennung für mobile Roboter. URL http://www.informatik.haw-hamburg.de/ robots/diekmann/diplom.pdf Zugriffsdatum: 10.11.2003

[Cvh03] The computer vision homepage. URL http://www-2.cs.cmu.edu/afs/cs/project/cil/ftp/html/vision.html Zugriffsdatum: 10.11.2003

[Rlb03] Robot-Labor der Hochschule für Angewandte Wissenschaften Hamburg. URL http://www.informatik.haw-hamburg.de/ robots/ Zugriffsdatum: 10.11.2003

[Xer03] XML - Projekt von Apache. URL http://xml.apache.org/xerces-c/index.html Zugriffsdatum: 10.11.2003

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw. §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe

---

Ort, Datum Unterschrift