Bachelorarbeit

Entwurf und Implementierung einer Objektverfolgung unter Verwendung einer Smart-Camera mit PTZ-Funktionalität

von

cand. inf. Thomas Dedek

Erstprüfer:	Prof. De. rer. nat. Jörg Hähner
Zweitprüfer:	Prof. Dr.-Ing. Christian Heipke
Betreuer:	M.Sc. Dipl.-Ing.(FH) Uwe Jänen

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und unter Verwendung der angegebenen Quellen und Hilfsmittel erstellt habe.

Hannover, den 20.11.2009

Thomas Dedek

Kurzfassung

Die Technik der Videoüberwachung wird ständig weiterentwickelt, die Systeme werden größer und komplexer. Der Entwicklungs- und Administrationsaufwand werden mit steigender Sensoranzahl für zentrale Systemarchitekturen kaum noch beherschbar. Daher ist die Entwicklung verteilter selbstorganisierender Überwachungssysteme ein aktuelles Forschungsziel. Durch den Einsatz von intelligenten Kameras (Smart-Cameras) wird dies möglich. Sie können Objekt selbstständig erkennen und sich daraufhin rekonfigurieren. In verteilten Netzen tauschen sie sich untereinander aus, um z.B. ihre Ausrichtung zu rekonfigurieren oder Objekte über mehrere Kameras hinweg zu verfolgen. Durch diese Fähigkeiten verringern sie den Administrationsaufwand von verteilten Netzen und erhöhen dadurch die Skalierbarkeit.

In dieser Bachelorarbeit sollen Smart-Cameras genutzt werden, um eine Objektverfolgung zu realisieren. Diese setzt sich aus einer passiven und aktiven Komponente zusammen. Der passive Teil behandelt das konsistente Labeling, d.h. die dauerhafte Markierung eines bzw. mehrere Objekte in einem Videostrom. Dies wird in dieser Arbeit mittels des CamShift-Algorithmus realisiert. Der Fokus dieser Arbeit liegt auf dem aktiven Teil Objektverfolgung, d.h. das Nachführen der Kamera zur Trajektorie des Objekts. In dieser Arbeit werden Verfahren vorgestellt, mit denen ein ausgewähltes Objekt unter Berücksichtigung weiterer Objekte in der Szene aktiv verfolgt werden können.

Abstract

Systems for observations are getting larger and more complex. That causes the related technology to improve more and more. The efforts for administration and developing are getting uncontrollable for centralized system architectures due to a growing number of sensor nodes. That's why the development of distributed self-organizing observation systems are a topical goal for science. This is getting possible by utilizing smart cameras. These cameras are able to detect objects independently and to reconfigure themselves. They communicate in distributed networks with other cameras e.g. to reconfigure or to track objects beyond several cameras. These skills contribute to a lower effort for administration and a higher scalability.

In this bachelor thesis smart cameras are used to realize object tracking. This consists of both an the active and the passive part. The passive part handles a solution to consistently label one or more objects in a video stream. This is being realized by the CamShift algorithm. The focus of this thesis is on the active part of tracking. This means an adjustment of the camera to the trajectory of the object.. This thesis introduces active tracking procedures to track one or more selected objects in a scene.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In den vergangenen Jahren und Jahrzehnten zeichnete sich ein gut wahrnehmbarer Trend hin zur Überwachung von sensiblen Orten mit Videokameras ab. Dies hatte Einfluss auf die Weiterentwicklung von Überwachungssystemen. Die ersten Kameras waren noch fest installiert, und damit auf einen stark begrenzten Sichtbereich eingeschränkt. Dies macht eine große Anzahl von Kameras nötig, um ebenso große Flächen zu überwachen. Dadurch erhöhte sich gleichzeitig der Aufwand seitens des Personals und der Datenübertragung. Alle Videodaten mussten zentral gespeichert werden. Bei eventuellen Vorkommnissen wurden diese dann gesichtet und analysiert. Die Einführung schwenkbarer Kameras verbesserte die Situation nur wenig. Die Beherrschbarkeit des Administrationsaufwands ist bei der wachsenden Komplexität in Frage gestellt.

Durch den Einsatz von intelligenten Kameras (Smart-Cameras) wird eine Selbstorganisation der Kameras möglich. Sie sind in der Lage, die Sensordaten direkt auf der Kamera auszuwerten wodurch u.a. die Algorithmen zur Objektverfolgung auf der Kamera ausgeführt werden können. Die Ergebnisse der Objektverfolgung können anschließend von der Kamera genutzt werden um z.B das Sichtfeld zu rekonfigurieren. Diese Eigenständigkeit ermöglicht

unter anderem die Verfolgung von Objekten über die Grenzen des Sichtbereichs einer Kamera hinaus. Diese Eigenständigkeit hat einen starken Rückgang des Administrationsaufwands und damit eine Steigerung der Skalierbarkeit zur Folge.

Diese Bachelorarbeit beschäftigt sich mit der Verfolgung von einem bzw. mehrerer Objekte durch eine einzelne PTZ-Kamera. Der Fokus liegt dabei auf der Nachführung der Kamera. Diese Selbstrekonfiguration ermöglicht eine Objektverfolgung über die Grenzen eines statischen Sichtfeldes hinaus.

1.2 Problemstellung

Der Fokus dieser Arbeit behandelt die Rekonfiguration der Kamera auf Grundlage der Ergebnisse der Objektverfolgung und -klassifikation. Eine triviale Lösung wäre es, den Mittelpunkt des Kamerasichtfeldes auf den Mittelpunkt des Objektes auszurichten. Dies würde allerdings eine hohe Frequenz an Rekonfigurationen zur Folge haben. Das Ziel soll es aber sein die Kamera so zu führen, dass sie sich nur dann bewegt, wenn die Situation dies erfordert. Dafür müssen die Aspekte dieser Forderung in Regeln definiert werden. Gleiches gilt für die Verfolgung mehrerer Objekte.

Eine Beispielsituation für eine Verfolgung wäre die Aufzeichnung eines oder mehrerer Sprecher während eines Vortrags. Die Aufzeichnung des Sprechers soll in der Frontalen durchgeführt werden. Er wird sich in einem festgelegten Bereich frei bewegen. Die naheliegende Lösung bei der das Sichtfeld (Field of View, FoV) der Kamera den gesamten Bewegungsraum abdeckt ist nicht zufriedenstellend. Ein vergrößerte Darstellung, die es erlaubt Mimiken und Gestiken zu erkennen, ist es, welche den Nutzen erhöht. Die Aufgabe der Verfolgung wird zusätzlich durch mögliche Störeinflüsse erschwert. Dabei ist unter anderem an Personen zu denken, welche das Sichtfeld kreuzen, und dabei möglicherweise den Sprecher vollständig verdecken.

Neben der Nachführung der Kamera, dem Hauptmerkmal dieser Arbeit, wird eine Objektdetektion benötigt, welche die Koordinaten eines oder mehrerer Zielobjekte aus einem Bild errechnet. Das gleiche Objekt soll dabei konstant über mehrere Bilder hinweg konsistent markiert werden. Aus den gewonnenen Koordinaten sollen die erforderlichen Steueranweisungen generiert werden. Diese wiederum sollen über eine Schnittstelle zu einem Treiber übergeben werden, welcher die Kamera steuert. Der Treiber stellt die dritte große Komponente im zu erstellenden System dar. Ein weiterer Teil soll das Rahmenprogramm darstellen, welches das Zusammenspiel der drei zuvor genannten Komponenten koordinieren soll. Diesem Teil kommt zudem die Aufgabe zu, den Videodatenstrom über eine Netzwerkverbindung abzufragen, zu dekodieren und für die Objektverfolgung bereitzustellen.

2 Grundlagen

2.1 PTZ-Kamera

Für die Entwicklung des Projekts wurde die PTZ-Kamera „Axis 214 PTZ“ von der Firma Axis verwendet. Das Kürzel „PTZ“ steht dabei für die englischen Wörter pan, tilt und zoom (schwenken, neigen und vergrößern), welche die besonderen Fähigkeiten der Kamera hervorheben. Von genau diesen Fähigkeiten soll in diesem Projekt dahingehend Gebrauch gemacht werden, indem sie für die automatische Ausrichtung der Kamera nach einem definierten Verhalten ohne menschliche Hilfe genutzt werden. Erst sie ermöglichen die Verfolgung eines Objekts über die Möglichkeiten und den Sichtbereich einer herkömmlichen Kamera hinaus.

Illustration 1: Vorderansicht einer Axis 214-PTZ, Quelle: [1]

2.1.1 Bildübertragung

Die PTZ-Kamera ist in der Lage, die Bilddaten über zwei Protokolle zu übertragen: HTTP und RTSP. Dieser Schritt wird bei einem Durchlauf zuerst ausgeführt. Der Zugriffsparameter der Protokolle sind jeweils in einem Dokument des Herstellers[2][3] erläutert. Die Spezifikationen beider Protokolle stehen frei zur Verfügung. Da die Vorteile von RTSP (Anfragen vom Server an den Client, Unterstützung von Sitzungen) gegenüber HTTP in diesem Kontext nicht von Bedeutung sind, wird das HTTP-Protokoll verwendet.

2.1.2 Kompressionsformat

Zur Reduktion des zu übertragenden Volumens der Videodaten werden diese auf Seiten der Kamera komprimiert. Sie unterstützt zwei Formate: Motion JPEG (kurz: M-JPEG) und MPEG-4 Part 2 (ISO/IEC 14496-2). Beide Formate erfordern eine client-seitige Dekomprimierung.

2.2 OpenCV

OpenCV (Open Computer Vision[4] ist der Name einer frei verfügbaren umfangreichen Programmbibliothek, die verschiedene Routinen zur Analyse und Verarbeitung von Bildmaterial zur Verfügung stellt. Aufgrund der vielfältigen Möglichkeiten und der freien Verfügbarkeit wird sie in dieser Bachelorarbeit verwendet. Darüber hinaus ist es von Bedeutung, dass sie eine Implementierung des CamShift-Algorithmus[5] anbietet.

2.2.1 Übertragung und Dekomprimierung von Videos

OpenCV bietet im Auslieferungszustand keine Unterstützung zur Übertragung von Videos

über ein Netzwerk, sondern kann nur das Containerformat AVI öffnen und unkomprimierte Videoströme verarbeiten. Aufgrund des großen Umfangs und der Abweichung von der Hauptthematik des Projekts ist es nachvollziehbar, dass diese Funktionen nicht feste Bestandteile von OpenCV sind. Vielmehr wurden Schnittstellen für andere Projekte entwickelt, welche die benötigten Funktionen innerhalb von OpenCV zur Verfügung stellen. Das wichtigste davon, dass sowohl in der Windows als auch der Unix-Ausführung zum Einsatz kommt ist FFmpeg[6]. Dieses Projekt stellt die Möglichkeit der Übertragung von Videoströmen über verschiedene Protokolle zur Verfügung, u.a. über HTTP. Zudem wird eine Vielzahl von Containerformaten und Kompressionsalgorithmen unterstützt[7]. FFmpeg ist wie OpenCV ebenfalls frei verfügbar. Die Unterstützung und Dokumentation für die Schnittstelle zwischen den beiden ist aber nicht vollständig abgedeckt. Unter Unix, wo die Quelltexte selbst übersetzt werden müssen, wird automatisch nach zu den Schnittstellen passenden Bibliotheken gesucht und diese eingebunden. Die Anwendung wurde transparent gestaltet. Die jeweiligen Methoden werden bei Bedarf automatisch aufgerufen. Unter Windows wird zwar eine dynamische Bibliothek mit den betreffenden Programmteilen mitgeliefert, diese wird aber an keiner Stelle erwähnt oder dokumentiert. Einzig eine in C implementierte Datei mit der Schnittstellendefinition ist vorhanden. Letztendlich steht also die Funktionalität für die Dekomprimierung auch unter Windows zur Verfügung. Die bisherigen betreffenden Funktionsaufrufe müssen fortan um das Suffix „_FFMPEG“ erweitert werden. Da diese Aufrufe unter Unix nicht bekannt sind, müssen sie in dieser Umgebung auf die normalen Aufrufe ohne Suffix umgeleitet werden.

2.2.2 Einbindung der Bibliotheken unter Windows

Unter Windows ist es nötig die folgenden statischen Bibliotheken statisch in das Projekt einzubinden:

cv120.lib
cvaux120.lib
cxcore120.lib
highgui120.lib

Diese statischen Bibliotheken greifen ihrerseits wiederum auf die folgenden dynamischen

Bibliotheken zu:

cv120.dll
cvaux120.dll
cxcore120.dll
highgui120.dll

Es ist sicherzustellen, dass sich die dynamischen Bibliotheken im Suchpfad des Betriebssystems oder im gleichen Verzeichnis wie die ausführbare Datei befinden.

Die Definitionen der Schnittstellen zu den Bibliotheken befinden sich in den folgenden Dateien:

cv.h
cvaux.h
cxcore.h
highgui.h

Zusätzlich ist nun die Einbindung einer Header-Datei für die FFmpeg-Schnittstelle einzubinden:

ffopencv.h

Die dazugehörige dynamische Bibliothek heißt:

opencv_ffmpeg120.dll

2.2.3 Einbindung der Bibliotheken unter Unix

Unter Unix werden die folgenden dynamischen Bibliotheken verwendet:

libcv.so
libcvaux.so
licxcore.so
libhighgui.so

Hinzu kommen die gleichen Header-Dateien wie bei der Windows-Variante:

cv.h

cvaux.h
cxcore.h
highgui.h

Damit die mit dem Suffix versehenen Funktionsaufrufe auf die normalen Funktionen umgelenkt werden, wurden zusätzlich Definitionen für diesen Zweck erstellt. Diese werden nur dann vom Compiler interpretiert, wenn er in einer Unix-Umgebung ausgeführt wird.

2.3 Objektverfolgung

Die Objektverfolgung (Tracking) gliedert sich in zwei Bestandteile. Das ist zum einen die Objektdetektion, das heißt einen bestimmten Objekttyp im Bild zu selektieren. Zum anderen ist dies die Objektklassifikation. Sie beschreibt ein konstantes Labeling des selektierten Objekts über mehrere Frames hinweg. Beide zusammen werden im Folgenden als passive Objektverfolgung (passives Tracking) bezeichnet. Wenn eine Rekonfiguration des Sichtfelds der Kamera hinzu kommt, dann wird von aktivem Tracking gesprochen.

2.4 Passive Objektverfolgung

Die Objektdetektion und die Objektklassifikation werden mittels des CamShift-Algorithmus aus den OpenCV-Bibliotheken durchgeführt. Dieser arbeitet auf Grundlage eines Farbhistogramms, sowie einem Suchbereich. Dieser Suchbereich wird mit Ausnahme der ersten Durchführung durch das Ergebnis des vorherigen Durchgangs definiert. Beim ersten Durchlauf muss dieser manuell festgelegt werden. Durch diese Einschränkung des Suchbereichs wird das konstante Labeling sichergestellt. Wird z.B. ein rotes Objekt in der linken unteren Bildecke verfolgt, so würde der Eintritt eines zweiten roten Objekts in der rechten oberen Bildecke das Ergebnis nicht beeinflussen. Der Suchbereich schränkt den Arbeitsbereich der eigentlich Bilderkennung ein.
Auf Grundlage der Farbstruktur, die in Form eines Farbhistogramms zur Verfügung gestellt

werden muss, wird eine Rückprojektion erzeugt. Sie kann als eine Maske betrachtet werden. Bei ihrer Erzeugung wird die Ähnlichkeit aller Bildpunkte zu dem Farbwert im Farbhistogramm abgebildet. Dies geschieht unabhängig von der Sättigung und der Helligkeit der Bildpunkte. Je näher er sich an der Vorgabe befindet, desto heller wird er dargestellt. Eine Übereinstimmung entspricht einem weißen Punkt in der Rückprojektion. Die Abbildung wird so erzeugt, dass das Resultat einen extrem hohen Kontrast aufweist. Dadurch enthält die Rückprojektion zu einem Großteil nur reine schwarze und weiße Bildpunkte. Durch diesen hohen Kontrast ist es leichter möglich eine zusammenhängende Gruppierung von der Umgebung abzugrenzen. Die drei folgenden Abbildungen sollen den Vergleich verdeutlichen:

Illustration 2: Das Ausgangsbild. Der rote Ball ist das Zielobjekt. Er ist durch eine rote Umrandung markiert worden.

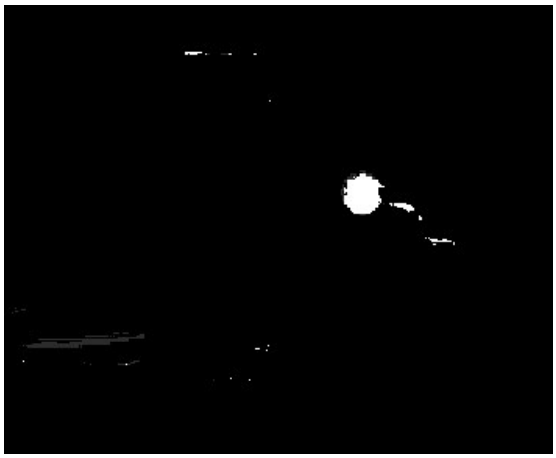

Illustration 3: Die Rückprojektion. Deutlich ist die weiße Erhellung an der gleichen Position zu erkennen, an der sich der Ball befindet.

Illustration 4: Die ist das Histogramm der Farbwerte, die sich innerhalb der Umrandung des Ausgangsbilds befinden. Hier ist die klare Dominanz roter Farbwerte zu sehen.

Diese Rückprojektion bildet zusammen mit dem Suchbereich die Grundlage, auf der CamShift arbeitet. Der Algorithmus sucht nach dem Mittelpunkt, den Ausmaßen und der Ausrichtung des Objekts. Die Ausrichtung ist in dieser Arbeit nicht von Bedeutung.

Die Vorteile von CamShift sind seine robusten Ergebnisse in einem geeigneten Umfeld. Zudem ist es ein leichtgewichtiger Algorithmus. Positiv ist auch, dass er frei verfügbar implementiert zur Verfügung steht. Der Nachteil ist die Tatsache, dass er Bedingungen an das Umfeld stellt, in dem er verwendet wird. Bei der erstmaligen Erfassung eines Objekts kann das Wunschobjekt nicht definiert werden. Der Algorithmus sucht unabhängig innerhalb des Suchbereichs nach der Stelle, die der Suchfarbe am nächsten kommt. Trotz diesem Nachteil erweist er sich in Anbetracht seiner Vorzüge als ausreichend gut geeignet für den Entwicklungsprozess.

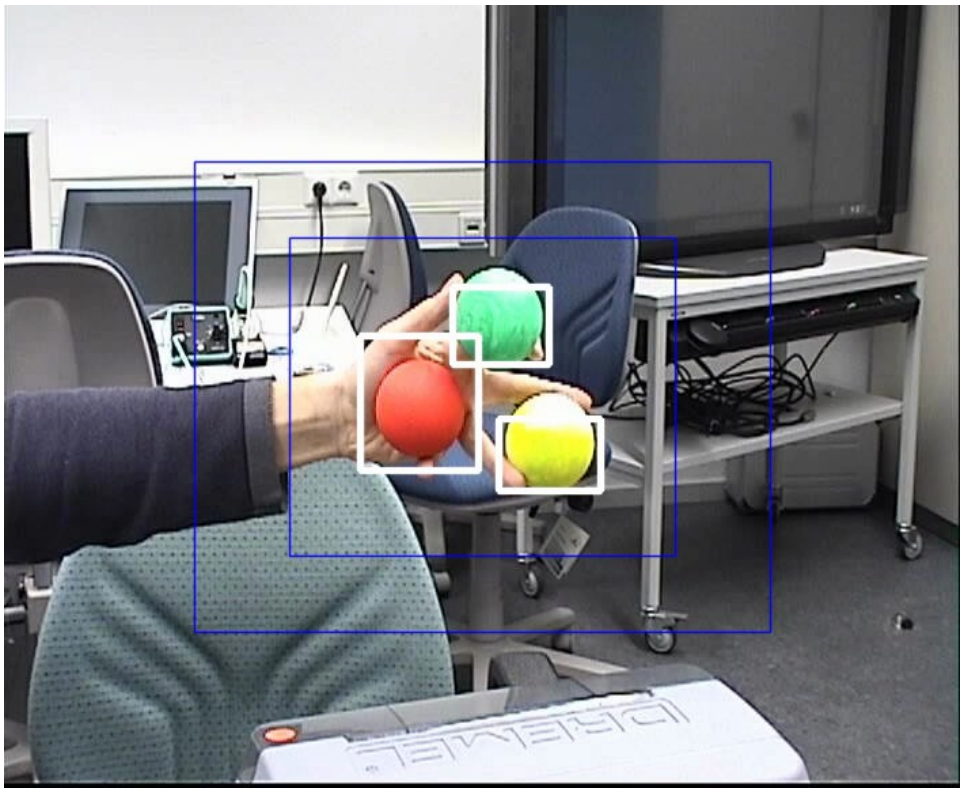

Illustration 5: Drei durch CamShift erkannte Objekte sind markiert.

3 Implementierung

Es wurde von vornherein Wert darauf gelegt, dass Projekt modular aufzubauen. Das heißt, dass unabhängige Aufgabenbereiche gekapselt, und durch eine Schnittstelle untereinander zugreifbar gemacht werden. Dadurch verringert sich der Aufwand für die Wartung sowie für einen nachträglichen Austausch eines Moduls.

Zur Unterstützung dieses Verfahrens ist es Hilfreich, nach dem objektorientierten Programmierparadigma zu entwickeln. Das fördert die Wiederverwendbarkeit von Teilen des Quelltextes, sowie dessen Lesbarkeit. Diese Einschränkung wirkt sich auch die Auswahlmöglichkeiten der Programmiersprachen aus. Da nicht alle von ihnen Objektorientierung unterstützen, schränkt sich die Menge ein. Einen weiteren Einfluss auf diese Wahl hat die OpenCV-Bibliothek. Sie wurde hauptsächlich in der Programmiersprache C entwickelt, und stellt seit der Version 1.2 eingeschränkt auch C++ Schnittstellen zur Verfügung. Einen Zugriff auf C-Funktionen oder gar deren Einbindung in eine objektorientierte Struktur ist mit C++ weitestgehend möglich. Daher wird die Programmiersprache C++ in diesem Projekt verwendet. Sie ermöglicht eine hinreichende Kapselung der Module, und erweist sich als sehr geeignet im Zusammenspiel mit den Schnittstellen. Auch die Verfügbarkeit des Compilers, der Entwicklungsumgebung sowie von Literatur haben diese Entscheidung beeinflusst.

3.1.1 Programmstruktur

Das gesamte Projekt besteht aus fünf Klassen, die jeweils eine definierte Aufgabe erfüllen. Beim Programmstart wird zuerst das Rahmenprogramm *Framework* aufgerufen, welches von den beiden Klassen *Detector* und *Tracker* Gebrauch macht. Letztere wiederum benutzt die Klasse *CamDriver*, welche je nach Betriebssystem die Klassen *NetworkWindows* oder *NetworkUnix* benutzt. Hier eine kurze Auflistung der Funktionen:

Framework:

Programmstart, Initialisieren der Objekte, Empfang und Übergabe den Parametern

Detektor:

passives Tracking: Bilderkennung und Klassifikation durch CamShift

Tracker:

aktives Tracking, Rekonfiguration der Kamera

CamDriver:

Erzeugen der Steuerbefehle für die PTZ-Kamera

NetworkWindows / NetworkUnix:

Zur Verfügungstellung der nativen Netzwerkfunktionalität

Die Zusammenhänge sind im folgenden Diagramm dargestellt:

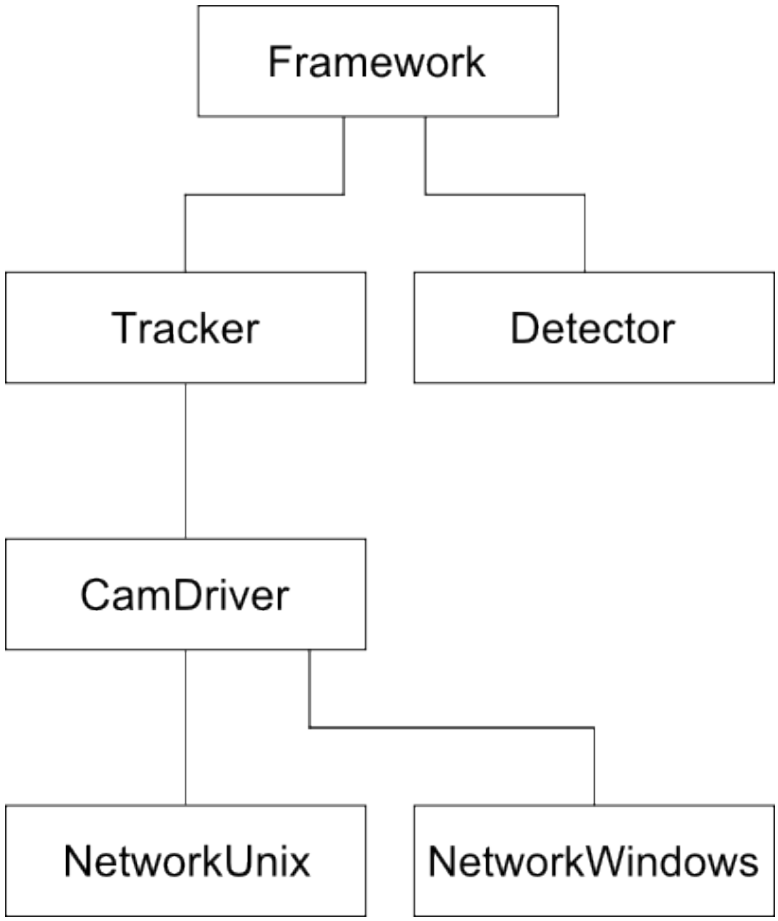

Illustration 6: Struktur der gegenseitigen Verwendung der Klassen.

3.2 Passives Tracking

Wie bereits erwähnt wird für das passive Tracking der CamShift-Algorithmus verwendet. Dieser greift bei seiner Ausführung auf Strukturen zurück, die zuvor vorbereitet werden müssen. Dabei handelt es sich um eine Konvertierung des Farbraums von dem RGB- zum HSV-Modell, eine Einschränkung dessen zur Erhöhung der Erkennungsleistung und der Erzeugung einer Rückprojektion auf Grundlage der jeweiligen Farbstruktur. Diese Vorgehensweise ist aus der Beispieldatei „camshiftdemo.c“ aus den Beispielen des OpenCV-Pakets entnommen.

Nach der Durchführung der Detektion werden zwei Strukturen zurückgegeben, welche die gewünschten Koordinaten enthalten. Diese werden dann auf Anfrage zurückgegeben und

können dann weiterverarbeitet werden.

3.2.1 Tracking mehrerer Objekte

Um den Entwicklungsprozess für das aktive und passive Tracking mehrerer Objekte zu unterstützen, ist es auch nötig mehrere Objekte zu erkennen. CamShift ist allerdings so konzipiert, dass nur ein Objekt pro Durchlauf erkannt werden kann. Daher werden mehrere Durchläufe von CamShift mit dem gleichen Bild durchgeführt. Die Durchläufe unterscheiden sich darin, dass jeweils nach einer anderen Farbe gesucht wird. Dadurch ist eine ausreichende Unterscheidung der Objekte sichergestellt. Die den Farbaufbau tragenden Strukturen müssen zuvor bei der Klasse bekannt gemacht werden.

Eine Zuordnung von Prioritäten kann auf Grundlage der Rückgabewerte der CamShift-Funktion durchgeführt werden. Diese sind die Koordinaten bestehend aus Position und Ausmaß, sowie der Ausrichtung des Objekts. Ein Beispiel ist eine Abbildung der Objektgröße. Große Objekte würden somit eine hohe Priorität erhalten oder umgekehrt. Neben einer Abbildung der Rückgabewerte kann man die Prioritäten manuell festlegen. Dies wurde auch bei der Entwicklung dieses Projekts durchgeführt.

3.2.2 Unterstützung von CamShift

In dieser Klasse werden auch unterstützende Maßnahmen unternommen, um die Robustheit von CamShift zu steigern. So wird zum einen der Bereich, in dem das Objekt gesucht wird automatisch von Bild zu Bild um einen prozentualen Anteil vergrößert. Dies geschieht unabhängig davon ob das Objekt verloren wurde oder nicht. CamShift korrigiert dann wieder die Ausmaße bei einer erfolgreichen Detektion im nächsten Durchlauf. Der Grund für diese Maßnahme ist, dass die Wahrscheinlichkeit dadurch verringert wird, dass das Objekt nicht mehr erkannt wird. Dies zeigt sich besonders bei kleinen Ausmaßen des Objekts.

Wenn sich das Objekt zu schnell aus dem Suchbereich von CamShift heraus bewegt, und damit nicht mehr auffindbar ist, so gibt CamShift den eingegebenen Suchbereich unverändert

bei den folgenden Durchläufen zurück. In diesem Fall wird der Suchbereich durch das eben beschriebene Verfahren immer weiter vergrößert. Befindet sich der Suchbereich ganz nah (weniger als vier Pixel entfernt) an einer Bildkante, so weitet er sich entlang dieser Kante doppelt so schnell aus als an seiner abgeneigten Seite (siehe Illustration 8).

Durch dieses Vorgehen erhöht sich die Robustheit der Detektion gegenüber ruckartigen Bewegungen, die sich innerhalb der Szenen auftreten können. Ermöglicht wird auch das Weiterverfolgen von Objekten, die das Sichtfeld an einem Bildrand verlassen und über denselben kurze Zeit später wieder eintreten. Diese Situationen kommen vor allem bei zu schnellen horizontalen Bewegungen vor, wenn die Kamera nicht schnell genug rekonfiguriert wird. Ein Beispielsituation für diese Erscheinung wäre die bereits genannte Aufzeichnung eines Sprechers.

Die folgende Darstellung soll dieses Verfahren verdeutlichen:

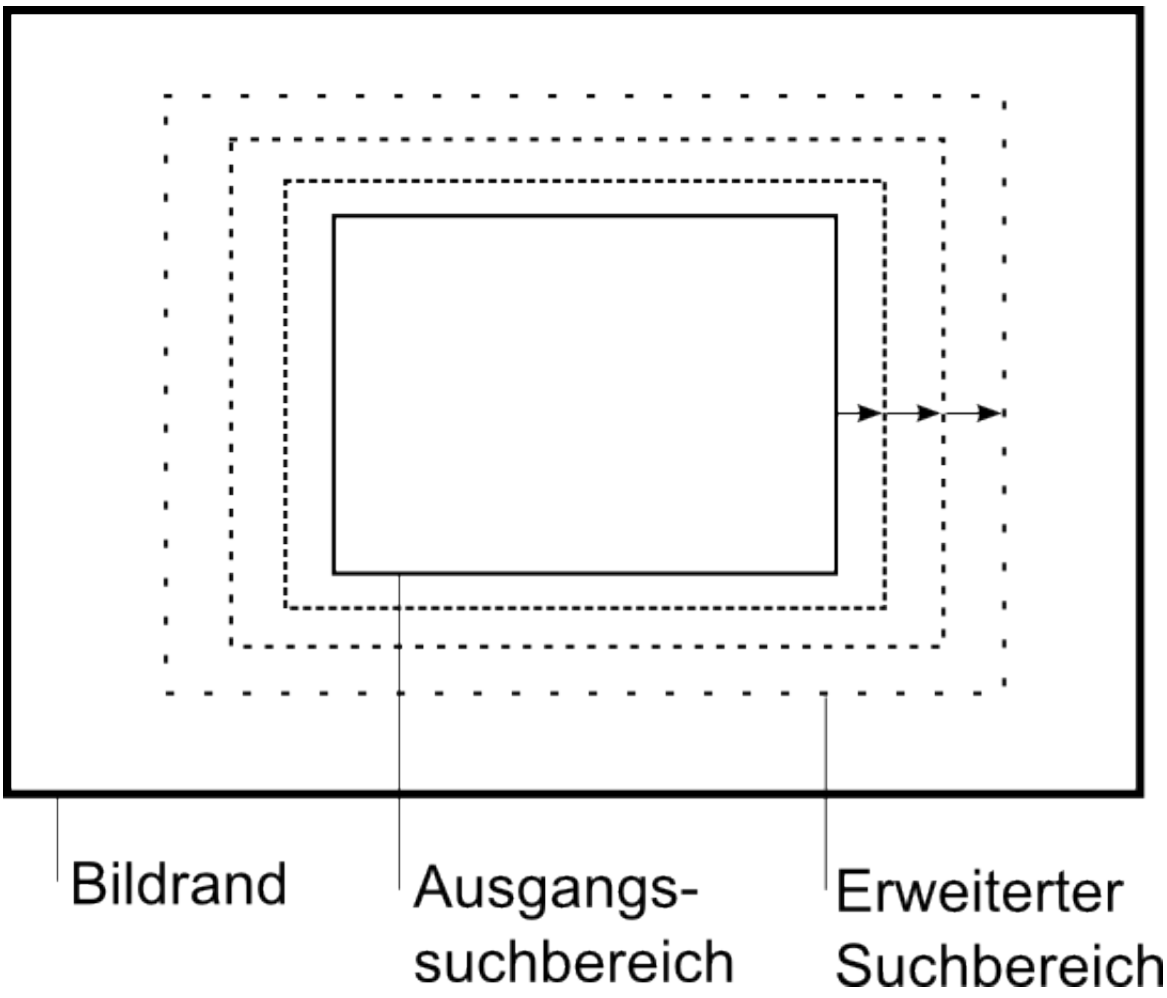

Illustration 7: Suchbereich breitet sich ausgehnd von der Mitte gleichmäßig in alle Richtungen aus.

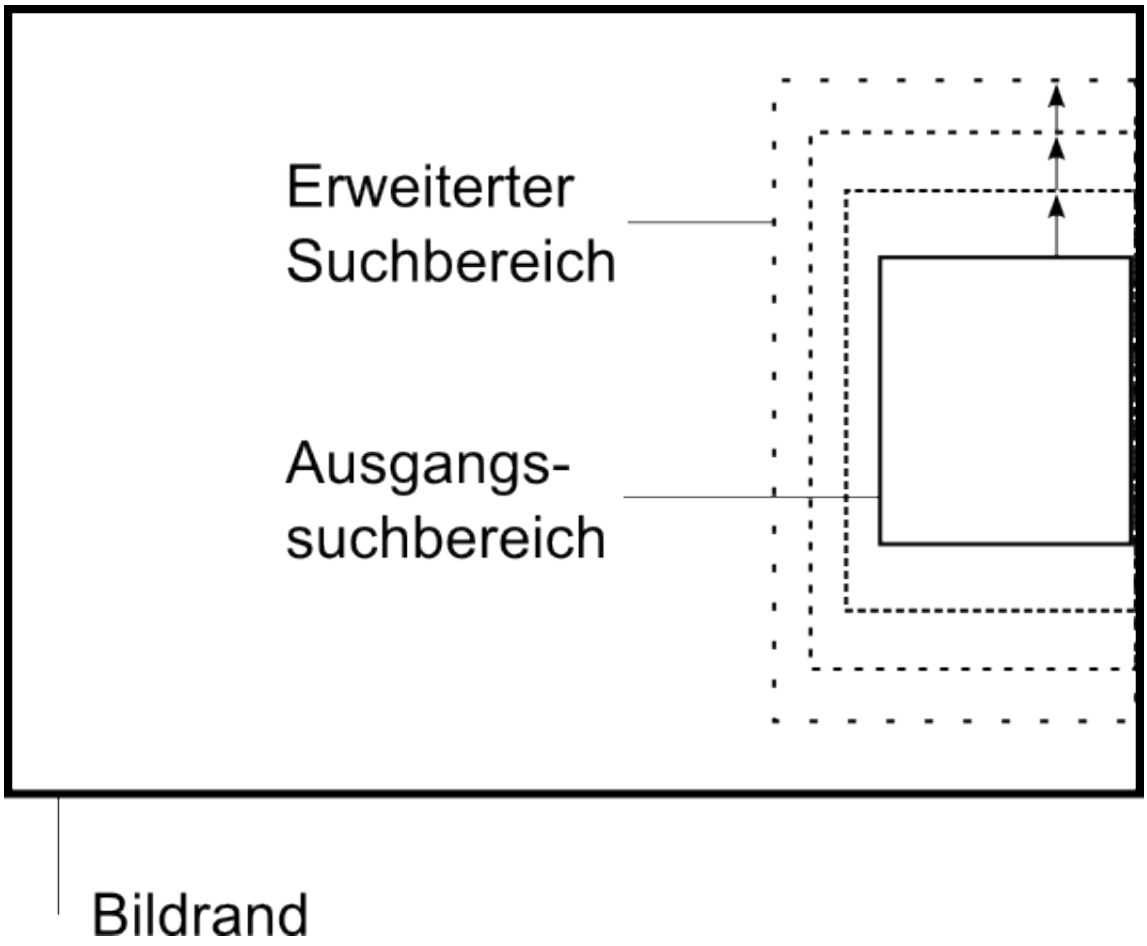

Illustration 8: Der Suchbereich breitet sich ausgehnd vom Bildrand bevorzugt an diesem entlang.

Als zweite Maßnahme wird der Suchbereich nach dem Verlust des Objekts auf die Dimensionen des Bildes zurückgesetzt. Dies hat zur Folge, dass CamShift im gesamten Bild weiterarbeitet, anstatt in im aktuellen Suchbereich.

Da dieser Objektverlust nicht festgestellt werden kann, wird er manuell von der Übergeordneten Klasse ausgelöst. Diese stellt eine Benutzerschnittstelle zur Verfügung, welche die Kontrolle dieses Vorgangs zur Verfügung stellt. Mehr dazu im Kapitel 3.11.2.

3.3 Netzwerk

Um die Steuerkommandos des Kameratreibers übermitteln zu können, wird eine Möglichkeit benötigt, welche den Zugriff auf ein Netzwerk ermöglicht. Übertragen werden sollen die Nachrichten dabei über das TCP/IP-Protokoll. Da diese Netzwerkschnittstelle eine unabhängige Funktionseinheit von dem Kameratreiber darstellt, wurde diese in eine eigenständige Klasse ausgelagert (*NetworkWindows bzw. NetworkLinux*). Die Funktionalität dieser Klasse wurde dabei lediglich auf den projektbezogenen Bedarf angepasst. So ist es in den meisten aller Zugriffe nur nötig Kommandos zu senden, ohne dass deren Antworten benötigt werden.

3.3.1 Aufgaben

Die Anforderungen an die Klassen *NetworkWindows bzw. NetworkLinux* sind Nachrichten zu senden und deren Antworten optional zu empfangen. Da der Empfang der Antwort eine geringe Latenz durch die Wartezeit verursacht, kann dieser durch einen Parameter je nach Bedarf genutzt oder ausgelassen werden. Während der Entwicklung hat es sich gezeigt, dass sich dieser Zeitvorteil positiv auf des Ergebnis auswirkt. Durch diese Verzögerungen konnte der aktive Tracking-Algorithmus auch nur verzögert auf das Geschehen reagieren, was zu einem Sprunghaften Kameraverhalten führt.

Das Senden einer Nachricht setzt das Registrieren eines Sockets und einen Verbindungsaufbau voraus. Nach dem Nachrichtenaustausch werden das Socket und die Verbindung wieder gelöscht.

3.3.2 Betriebssystemspezifische Abhängigkeiten

Ein zusätzliches Argument für die Auslagerung dieses Teils in eine eigene Klasse ist die Abhängigkeit vom darunterliegenden Betriebssystems. Jede Klasse von Betriebssystemen bringt seine eigenen Bibliotheken und deren Aufrufregeln mit, welche untereinander nicht

eindeutig übereinstimmen. Für eine höhere Abdeckung der gängigen Betriebssysteme wurde diese Klasse für dieses Projekt für zwei Betriebssystemfamilien entwickelt: für Microsoft Windows und Linux- bzw. Unix-Derivate. Auf diese wird je nach Plattform vom Kameratreiber zugegriffen. Durch diese Klasse werden die Eigenheiten des jeweiligen darunterliegenden Betriebssystems wie z.B. dem Verbindungsaufbau von den darüberliegenden Klassen abstrahiert, wie in diesem Fall dem Kameratreiber.

Für die Verwendung der jeweiligen Bibliotheken für den Netzwerkzugriff ist es nötig, externe Bibliotheken bzw. Header-Dateien einzubinden. In der Windows-Ausführung stammen diese von Microsoft. Sie werden zur Entwicklung von Programmen unter Windows frei zur Verfügung gestellt. Sie sind eine wichtige Schnittstelle zwischen dem Programm und dem Betriebssystem, um einen Zugriff auf ein Netzwerk zu erhalten. Daher werden sie als Bestandteil vieler Compiler mitgeliefert. Der Name der Bibliothek lautet „libws2_32.a“. Sie wird statisch in das Projekt eingebunden, die ihrerseits wiederum auf die dynamische Bibliothek „winsock.dll„ zugreift. Letzter ist ein Bestandteil jeder Windows-Installation. Die für den Compiler notwendigen Schnittstellendefinitionen dieser Bibliothek werden in der Datei „winsock2.h“ definiert.

Die Unix-Ausführung erfordert lediglich die Einbindung der Header-Datei „arpa/inet.h“.

3.4 Kameratreiber

Alle Anweisungen bezüglich der Kamera werden über diese Klasse ausgeführt.Sie ist dafür Zuständig, die aus allgemeinen Anweisungen (z.B. „gehe vier Grad nach links“) die konkreten Steuerbefehle zu erzeugen. Durch diese Kapselung der kameraspezifischen Anweisungen wird es möglich, diese Klasse nachträglich durch eine andere zu ersetzen. Diese könnte weitere Steuerbefehle beinhalten oder weitere Kameramodelle bzw. Hersteller unterstützen.

3.4.1 PTZ-Funktionen

Ein Großteil der implementierten Funktionen auf Treiberebene stellen Möglichkeiten zur Nutzung der PTZ-Funktionen der Kamera bereit. Diese verwendet HTTP-Requests um die Konfiguration zu ändern. Dazu zählen Bewegungskommandos in horizontaler Richtung, in vertikaler Richtung sowie zum Ändern der Vergrößerung (zoom). Von diesen gibt es drei Typen die sich wie folgt unterscheiden:

- Übergabe der Durchführungsgeschwindigkeit, z.B. „Drehe Kopf mit 20°/s nach rechts“. Im Unterschied zu den beiden folgenden Typen muss diese Bewegung bei Bedarf manuell gestoppt werden.
- Übergabe absoluter Soll-Werte, z.B. „Drehe Kopf zur Position -20°“
- Nicht verwendet wird der dritte Typ: Übergabe von relativen Soll-Werten, unabhängig von den absoluten Ist-Werten, z.B. „Drehe Kopf um 10° nach links“

Des Weiteren können die automatische Fokussierung und die automatische Festlegung der Irisgröße aktiviert und deaktiviert werden. Alle diese Werte können auch aus der Kamera ausgelesen werden.

Aus diesen Befehlen werden auf Anfrage HTTP-konforme Anfragen erzeugt, und diese an die Netzwerkklasse übergeben.

3.4.2 Sendeintervalle

Eine weitere wichtige Aufgabe des Treibers ist es, eine Möglichkeit zur Verfügung zu stellen, die Überflutung der Kamera mit Anfragen zu verhindern. Sie ist zwar dazu in der Lage, eine gewisse Menge innerhalb kurzer Zeit aufzunehmen, und nacheinander abzuarbeiten. Aber diese Möglichkeit findet auf Seiten der Kamera ihre Grenzen. Welche Anfragen dann berücksichtigt oder verworfen werden ist nicht vorhersagbar. Zusätzlich würde sich die Reaktionszeit zum Teil deutlich erhöhen, so dass der letzte Befehl in der Schlange nach seiner

Wartezeit unter Umständen keine Relevanz mehr hat.

Zum Einsatz kommt dieser Schutzmechanismus wenn die Treiberklasse vom Tracker verwendet wird. Aufgrund einer Bildwiederholfrequenz von 25 Bildern pro Sekunde können theoretisch bis zu 25 Anfragen pro Sekunden gestellt werden. Die Folge einer ungefilterten Übertragung wäre, dass die Kamera u.U. noch mit den Anweisungen der ersten drei Sekunden beschäftigt ist, während sich das Zielobjekt aus dem Sichtfeld bewegt. Um dies zu verhindern, kann dem Kameratreiber beim übergeben der Anweisung über einen Parameter mitgeteilt werden, dass dieser Aufruf eine untergeordnete Priorität hat. Dadurch wird beim Aufruf seitens des Treibers geprüft, ob eine zuvor festgelegte Zeitspanne seit der letzten Anfrage (jeglicher Art) vergangen ist. Ist dies der Fall, so wird die Anweisung ausgeführt. Andernfalls verworfen. Die zweite Möglichkeit führt zu einer unverzüglichen Ausführung der Anweisung ohne eine Prüfung. Die Zeitspanne wurde für die Entwicklung auf 500ms festgelegt. Dieser Wert hatte eine gute Reaktionsgeschwindigkeit zur Folge.

Die Algorithmen zur Nachführung der Kamera machen ausschließlich von den niedrig Priorisierten Aufruf Gebrauch. Das Resultat ist eine funktionierende Rückführung, ohne Überlastungen der Kamera.

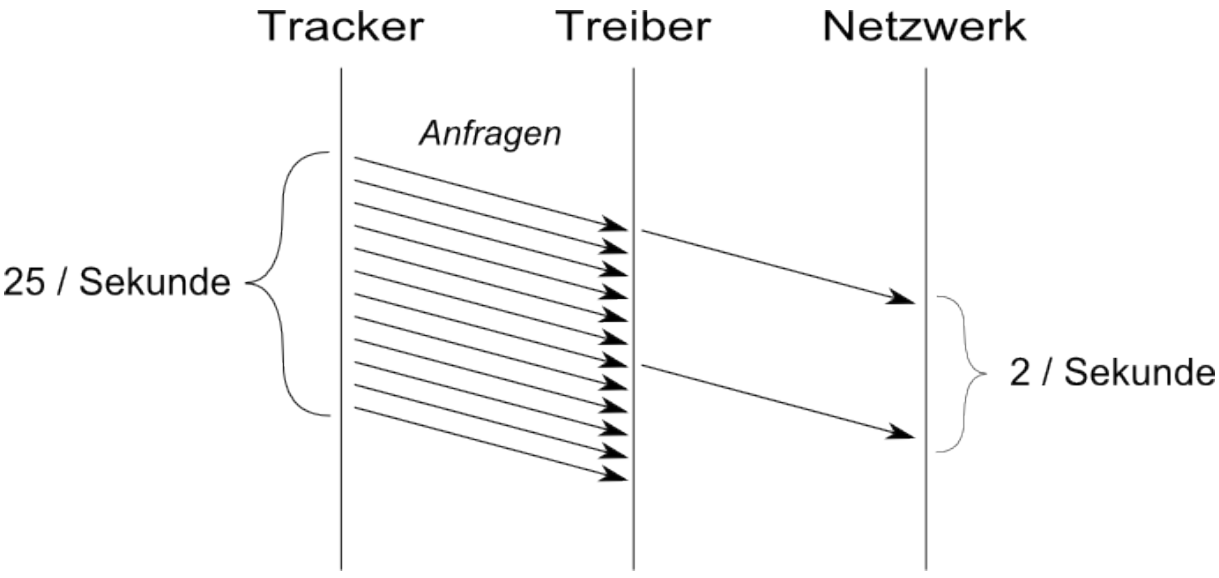

Illustration 9: Anfragefilterung durch den Kameratreiber: Anfragen an den Treiber werden teilweise verworfen

3.5 Aktives Tracking

Alle Programmteile, die für das aktive Tracking zuständig sind, sind in der Klasse *Tracker* zusammengefasst. Diese wird von der Klasse *Framework* erzeugt und verwendet. In dieser Klasse wird Gebrauch gemacht von der Klasse *CamDriver*.

Um einen größtmöglichen Bildausschnitt zur initialen Objektdetektion zu erhalten, wird der Zoom zu Beginn so weit wie möglich herausgefahren. Zusätzlich wird die Kamera auf einen definierbaren Nullpunkt zentriert. Dieser sollte so eingestellt werden, dass sich das Zielobjekt zu Beginn einer Sitzung dort befindet. Sobald ein Objekt dann zum ersten Mal erkannt wurde, wird der Zoom auf einen definierten Standardwert für die weitere Verfolgung vergrößert. Der gleiche Vorgang wird bei einem Objektverlust durchgeführt.

3.6 Verfolgung eines Objekts

3.6.1 Vorbereitung

Das Ziel dieses ersten Verfahrens ist es, ein einzelnes Objekt dauerhaft aktiv zu Verfolgen. Nach den Ergebnissen der passiven Verfolgung soll entschieden werden, ob eine Rekonfiguration der Kamera nötig ist. Ist dies der Fall, so muss entschieden werden wohin, bzw. wie schnell in welche Richtung.
Ein mögliches Szenario für eine solche Verfolgung wäre die bereits erwähnte Verfolgung eines vortragenden Sprechers einhergehend mit der Aufzeichnung des Videostroms. Die Kameraführung soll möglichst selten aktiv werden und sich möglichst weich gestalten, damit das Videomaterial eine hohe Qualität aufweist.
Das passive Tracking des CamShift-Algorithmus liefert die Koordinaten des Objekts. Als zusätzlichen Parameter werden die Ausmaße des Bildes erwartet. Durch sie ist es möglich die relative Position des Objekts im Bild zu bestimmen. Ebenfalls abhängig von den Ausmaßen des Bildes werden zu Beginn zwei Rahmen innerhalb des Bildes definiert. Die folgende Abbildung soll dies veranschaulichen:

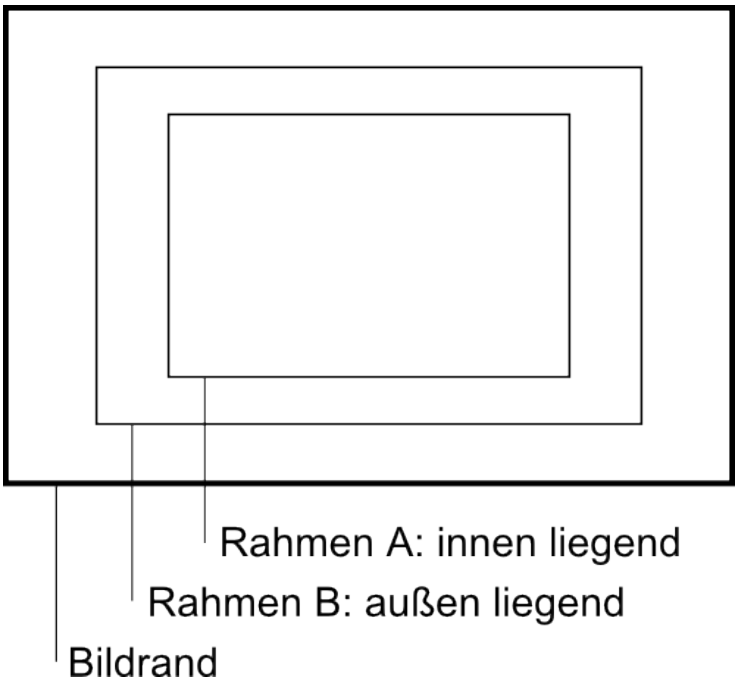

Illustration 10: Aufbau der Rahmen innerhalb des Bildes

Diese Rahmen unterteilen das Bild in drei Bereiche. Diese Bereich bilden die Grundlage für die Entscheidung, ob eine Bewegung nötig ist. Je nach Einsatzzweck und Erfahrungen aus dem praktischen Einsatz können die Ausmaße oder auch die Positionen der Rahmen angepasst werden.

3.6.2 Ablauf

Auf der Grundlage der beiden definierten Rahmen wird zu Beginn ermittelt, ob eine Reaktion nötig ist. Befindet sich das Objekt zum vierten mal in Folge außerhalb von A, so wird eine Rekonfiguration der Kamera ausgelöst. Die Kamera befindet sich fortan im Bewegungszustand. Sie tritt erst aus diesem Zustand aus, wenn sich das Objekt mindestens einmal innerhalb von A befindet.

Während des Bewegungszustands werden in jedem Durchgang (in jedem Frame) die Parameter für eine kontinuierliche Kamerabewegung ermittelt. Diese soll das Objekt auf der kürzesten Strecke zurück zum Bildmittelpunkt führen. Die Anweisung für eine kontinuierliche

Kamerabewegung erfordert die Angabe der Geschwindigkeiten der Kamerabewegung sowohl in horizontaler als auch in vertikaler Richtung. Diese Werte werden aus dem Abstand des Objekts zum Bildmittelpunkt in Pixel errechnet. Damit wird das Objekt aus jeder Position heraus zurück zum Bildmittelpunkt geführt, bzw. die Kamera auf den Objektmittelpunkt ausgerichtet. Dies geschieht so lange bis das Objekt sich innerhalb von A befindet. Der Vorgang beginnt an dieser Stelle von vorne. Die Rückführungsgeschwindigkeit kann durch einen festgelegten Faktor zusätzlich beeinflusst werden.

Der folgende Pseudocode soll dies verkürzt darstellen:

Algorithmus 1 Rekonfiguration bei einem Objekt

```
1:    init:
2:    define Border A
3:    define Border B
4:    if object is outside B
5:          vertical speed ←      verticalDistanceToImageCentre / 6:
                                        speedFactor
7:          horizontal speed ←    horizontalDistanceToImageCentre / 8:
                                        speedFactor
9:    else if object is inside A
10:         vertical speed ← 0
11:         horizontal speed ←    0
```

Bei der Art der Bewegung unterscheidet die Kamera zwischen absoluten, relativen und kontinuierlichen Positionsänderungen. Hierbei, und bei den beiden folgenden Verfahren wird der letzte Typ verwendet. Die Kamera bewegt sich dabei solange nach einer festgelegten Geschwindigkeit, bis sie entweder an ihre physikalischen Grenzen stößt, oder sie zum Stoppen aufgefordert wird.

Da der verwendete Algorithmus immer beide Achsen zusammen auswertet, kann es vorkommen, dass tatsächlich nur die Rekonfiguration einer einzelnen Achse nötig ist. Dies wird vor dem Senden der Anfrage an den Treiber noch separat geprüft und gegebenenfalls korrigiert.

Abschließend wird der Kameratreiber mit der Bewegungsaufforderung aufgerufen. Die beiden ermittelten Werte werden dabei übergeben.

3.7 Verfolgung mehrere Objekte - Verfahren 1

Die Anforderungen an eine Nachführung für mehrere Objekte ist komplexer als bei einem einzelnen Objekt. Hier ist zuerst zu überlegen, worauf genau Wert gelegt werden soll. Mögliche Aspekte, die dabei betrachtet werden, könnten sein: Anzahl der Kamerabewegungen, Anzahl der Objekte, Priorisierung oder Gleichberechtigung innerhalb der Gruppe der Objekte usw.

Bei diesem Verfahren werden neben den Koordinaten auch die Priorität des Objekts übergeben. Alle zur Verfügung stehenden Daten werden in einer Liste gespeichert, wobei das höchstpriorisierte Objekt das erste in der Liste ist.

Einleitend für dieses Verfahren werden die Ausmaße eines Rechtecks definiert, das den Bereich markiert, in dem sich die Objekte befinden müssen, damit sie bei der Berechnung berücksichtigt werden. Dieses Rechtecke wird in jedem Bild so platziert, dass sich das Objekt mit der höchsten Priorität in dessen Mitte befindet. Siehe folgende Darstellung:

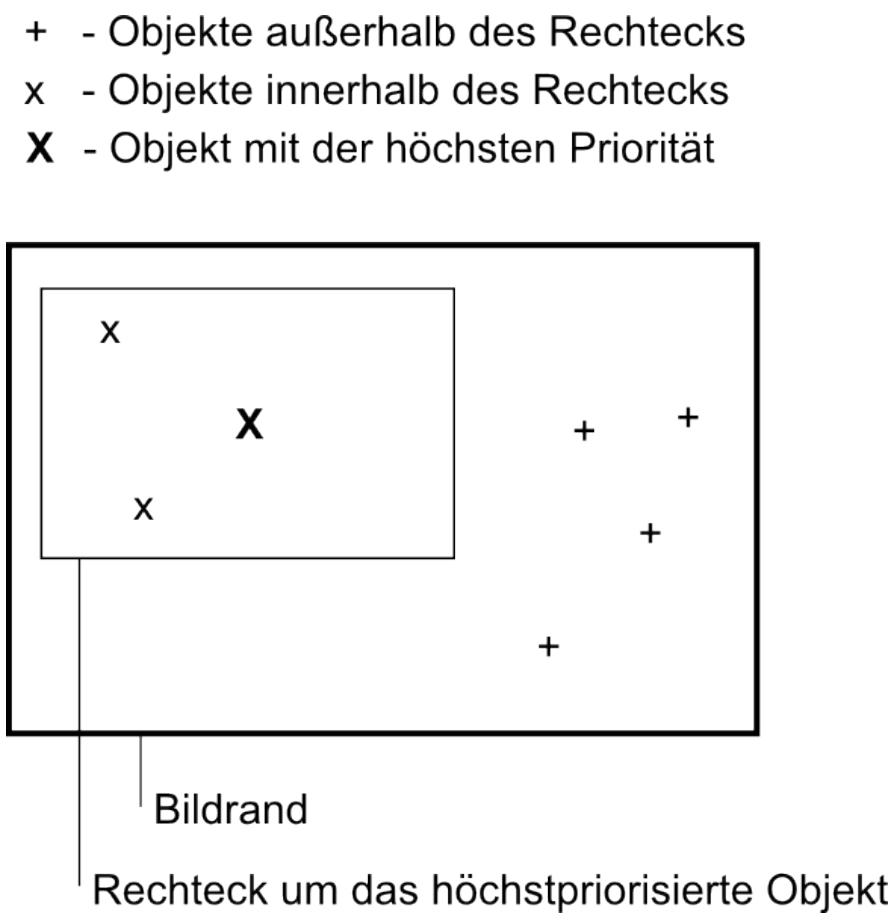

Illustration 11: Darstellung des Auswahlrechtecks. Zu sehen sind zwei Objekte die sich neben dem höchstpriorisierten Objekt innerhalb des Rechtecks befinden. Zusätzlich liegen vier weitere Objekte außerhalb.

Aus den Mittelpunkten aller zuvor ausgewählten Objekte wird anschließend der gemeinsame Mittelpunkt errechnet. Auf diesen soll sich die Kamera ausrichten. Dies tut sie aber erst dann, wenn sich dieser Punkt außerhalb von Rahmen A befindet. Ist dies der Fall, so werden genau wie im Verfahren für die Verfolgung eines Objekts die Geschwindigkeiten für die beiden Achsen X und Y ermittelt. Befindet sich der Mittelpunkt innerhalb von A, so wird die Geschwindigkeit auf Null gesetzt.

Über die vorherigen Prüfungen hinaus werden im Anschluss die Geschwindigkeitswerte auf einen Mindestwert von 2% geprüft. Liegen sie darunter, wo werden sie auf Null gesetzt. Dadurch sollen kleinste zirkulierende Bewegungen herausgefiltert werden. Die eingesetzte Bilderkennung weist oft kleine Abweichungen der Objektgröße aus, obwohl es sich nicht verändert, bzw. bewegt hat. Die dadurch entstehenden Korrekturbewegungen werden durch diesen Filter eliminiert.

3.8 Verfolgung mehrerer Objekte – Verfahren 2

Beim zweiten Verfahren werden die Grenzen der Kamerabewegung festgelegt, in denen sich möglichst viele Objekte zentrieren lassen. Dazu wird von dem im Kapitel 3.6.1 vorgestellten Rahmen B Gebrauch gemacht.

Es gibt für jedes Objekt einen Toleranzbereich von Kamerabewegungen der es erlaubt, das Objekt innerhalb des Rahmens A zu halten oder es dorthin zurückzuführen. Um Unterschied zum vorherigen Verfahren wird nicht mit dem Objektmittelpunkten, sondern mit deren Außenkanten gearbeitet. Die folgende Grafik soll dies veranschaulichen:

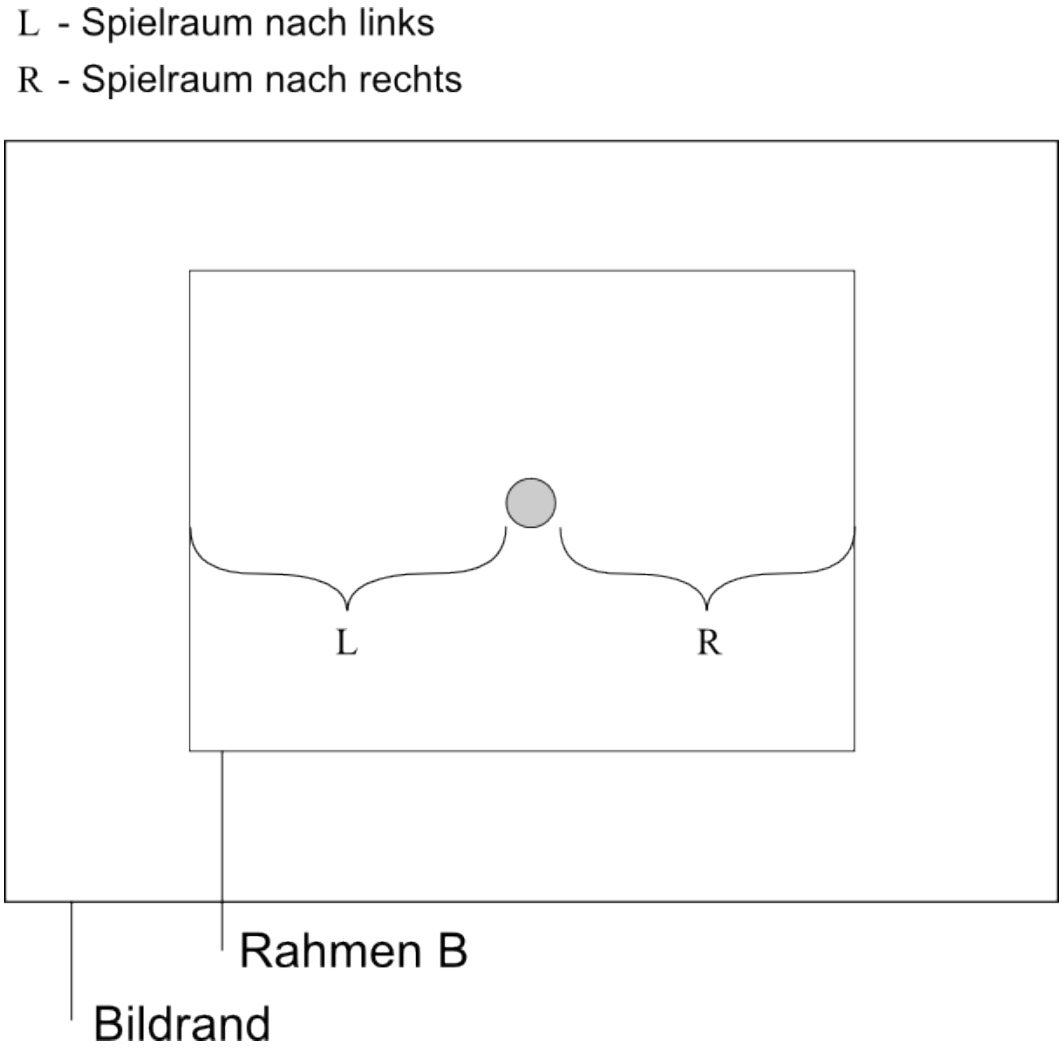

Illustration 12: Zu sehen ist ein Objekt und dessen Spielräume zu den nächsten Kanten des Rahmens in horizontaler Ausrichtung.

Bei diesem Verfahren werden neben den Koordinaten auch die Prioritäten der Objekte übergeben, und die zur Verfügung stehenden Daten in einer Liste gespeichert, wobei das höchstpriorisierte Objekt das erste in der Liste ist. Mit diesem Objekt wird der Algorithmus gestartet. Es werden die Toleranzbereiche auf der horizontaler Achse nach links und nach rechts, sowie auf der vertikalen Achse nach oben und nach unten ermittelt. Diese ermittelten Werte werden im Folgenden nur noch so eingeschränkt, dass sichergestellt ist, dass sich das Objekt mit der höchsten Priorität immer innerhalb des Rahmens B befindet.

Anschließend werden diese Werte der Reihe nach von den restlichen Objekten entnommen. Sie werden einzeln überprüft, ob sie sich innerhalb der vorgegebenen Werte des höchstpriorisierten Objekts befinden. Das folgende Beispiel soll dies veranschaulichen:

> *Ein Objekt mit der Priorität 1.0 hat ein Spielraum von 30 Pixeln nach links und 30 Pixeln nach rechts. Ein Objekt mit der Priorität 0.7 hat einen Spielraum von 20 Pixeln nach links und 40 Pixeln nach rechts. Das Resultat ist ein Spielraum von 20 Pixeln nach links (Objekt 1.0 wird nicht gefährdet, Objekt zwei ist gesichert) und von 30 Pixeln nach rechts (die Grenze von 30 darf nicht überschritten werden.*

Befindet sich sowohl mindestens einer der horizontalen als auch mindestens einer der vertikalen Spielräume innerhalb der vorher ermittelten Grenzwerte, so werden die eigenen Werte berücksichtigt. Durch diese Einschränkung ist gewährleistet, dass Objekte die vollständig zurückgeführt werden können bevorzugt behandelt werden. Dafür werden alle anderen Objekte benachteiligt. Sollte das zukünftige Einsatzszenario zeigen, dass alle Objekte durchschnittlich einen größeren Abstand zu dem Objekt mit der höchsten Priorität haben, d.h. keines vollständig zurückgeführt werden kann, so wäre die Aufhebung dieser Restriktion zu empfehlen.

Nach der Überprüfung aller Objekte erhält man einen Spielraum bestehen aus den vier Werten. Aus diesen Werten werden zwei Mittelwerte gebildet, einer für die Horizontale und einer für die Vertikale. Dieser Mittelwert beschreibt einen Punkt, der anschließend durch eine entsprechende Kamerabewegung in die Bildmitte geführt wird.

Nachdem alle Objekte durchlaufen wurden, wird die Geschwindigkeit errechnet. Sie setzt sich zusammen aus dem Mittelwert der jeweiligen Achse und dem bekannten Geschwindigkeitsfaktor. Kleinste Bewegungen (kleiner als 3%) werden bei diesem Verfahren unterdrückt. Der Grund dafür liegt wie bei dem ersten Verfahren auch in der teilweise auftretenden Ungenauigkeit der Bilderkennung.

3.9 Vergleich beider Verfahren für mehrere Objekte

Die Vor- und Nachteile dieser beiden Verfahren lassen sich nicht ohne ein konkretes Einsatzziel erkenne. Ist es gewollt, dass möglichst viele (alle) Objekte zentriert werden, so bietet es sich an das vorherige Verfahren zu verwenden. Soll die Aufmerksamkeit beim zentralsten Objekt liegen, wäre das zweite Verfahren geeigneter, da es weiter Abseits liegende Objekte übergeht.

Beide Verfahren kennen theoretisch keine Obergrenze für die Anzahl der zu berücksichtigenden Objekte. Für die Entwicklung wurde diese jedoch auf den Wert zehn festgelegt. Dieser Grenzwert kann jederzeit verändert werden.

3.10 Verlust aller Objekte

Bisher wurden nur all jene Situation gehandhabt, in denen sich mindestens ein trackingfähiges Objekt im Sichtfeld befindet. Die zweite Möglichkeit ist dass sich alle Objekte aus dem Sichtbereich entfernt haben bzw. aus anderen Gründen nicht mehr erkennbar sind. Dieser Fall kann der Klasse durch eine eigene Methode mitgeteilt werden. Daraufhin wird der Zoom so weit wie möglich herausgefahren. Anschließend dreht sich die Kamera in die zuvor definierte Ausgangsposition zurück. An diesem Punkt kann der gesamte Zyklus von vorne gestartet werden.

3.11 Rahmenprogramm

Das Rahmenprogramm (oder auch Framework) hat eine administrative Bedeutung in dem Projekt. Es erzeugt die notwendigen Objekt und löst deren Initialisierung mit den notwendigen Parametern aus. Auch die Aufrufe der Methoden dieser Objekte wird hier ausgelöst, und deren Rückgabewerte entsprechend weitergeleitet.

3.11.1 Videostrom

Eine Aufgabe des Frameworks ist es, die Verbindung zum übertragen des Videostroms zur Kamera herzustellen. An dieser Stelle kommen die im Kapitel 2.2.1 erwähnten Funktionen zur Dekomprimierung zum Einsatz. Der Videostrom wird über eine URL von der Kamera empfangen und anschließend dekomprimiert. Die Einzelbilder bilden neben den Farbwerten die Eingabe zur Bilderkennung.

Da beim Programmstart die Kamera in eine definierte Ausgangsposition geschwenkt wird, ist nicht sichergestellt, dass das erste empfangene Bild aus dieser Position stammt. Um dies zu erreichen, wird zu Beginn genau 80 Bilder empfangen und verworfen. Dies entspricht bei einer Bildrate von 25 Bildern/Sek. ca. drei Sekunden. Bei der Entwicklung hat sich gezeigt, dass diese Anzahl ausreicht, um diesen Effekt zu vermeiden.

Für die Evaluierung der angewandten Arbeitsweise der Algorithmen zum aktiven Tracking, werden abschließend zusätzlich die dafür verwendeten Rahmen A und B in das angezeigte Videobild gezeichnet.

3.11.2 Benutzerschnittstelle

Um dem Anwender eine Möglichkeit zu geben in den Programmverlauf einzuschreiten, gibt es eine Benutzerschnittstelle. Folgende Kommandos stehen dem Benutzer zur Verfügung:

- Escape-Taste: Beendet das Programm
- r: Setzt den Suchbereich der Bilderkennung (CamShift) zurück auf die Ausmaße des Bildes
- c: Richtet die Kamera auf die Ausgangsposition aus und setzt ebenfalls die Bilderkennung zurück (wie r)
- l: Umschalten zwischen Verlust- und Normalbetrieb. Beim Eintritt in den Verlustbetrieb wird die Detektion gestoppt, das Bild herausgezoomt und die Kamera in die Ausgangsposition zurückgesetzt. Der Eintritt in den Normalbetrieb ist identisch mit dem Programmstart: Detektion starten, bei erstem Objektfund hineinzoomen

Eine Übersicht aller Kommandos wird beim Programmstart zusätzlich auf der Konsole ausgegeben.

Da CamShift das Ereignis eines Objektverlustes nicht signalisieren kann, wird dieser Zustand bei der Entwicklung durch die Benutzerschnittstelle (Taste „l“) manuell ausgelöst und wieder aufgehoben.

4 Evaluation

Zur abschließenden Bewertung der entwickelten Verfahren werden diese durch Tests evaluiert. Dabei wird in unterschiedlichen Testreihen überprüft, ob sich die Anforderungen im Ergebnis widerspiegeln. Für jedes der drei vorgestellten Verfahren gibt es unabhängige Tests. Diese unterscheiden sich teilweise geringfügig Aufgrund ihrer Beschaffenheit.

Das Umfeld simuliert eine Aufzeichnung einer bzw. mehrerer Objekte. Diese werden durch Bälle simuliert, welche entsprechend trackingfähig sind. Bei den Tests für die Verfolgung mehrerer Objekte werden die Objekte derartig bewegt, dass sich die Leistungsfähigkeit der Verfahren aufzeigen lässt.

Zur nachvollziehbaren Überprüfung der Testergebnisse sind die Aufzeichnungen auf der beiliegenden CD-ROM gespeichert.

4.1 Testaufbau

Die Tests werden in einem zur Verfügung gestellten Seminarraum im Institut SRA an der Leibniz Universität Hannover durchgeführt.

4.1.1 Verfolgung eines Objekts

Test 1.1: Bewegung eines Objekts innerhalb des Sichtbereichs sowie Versuche den Sichtbereich an jedem der vier Ränder zu verlassen. Dadurch soll eine Bewegung der Kamera in die entsprechende Richtung ausgelöst werden.

Test 1.2: Manuelle Simulation eines Objektverlustes. Die Kamera soll daraufhin heraus-zoomen und in die Ausgangsstellung zurückkehren.
Anschließend wird die Erkennung und Verfolgung wieder aktiviert, das Objekt damit neu erfasst.

Test 1.3: Nachdem ein falsches Objekt selektiert wurde, soll dies durch betätigen der r-Taste behoben werden.

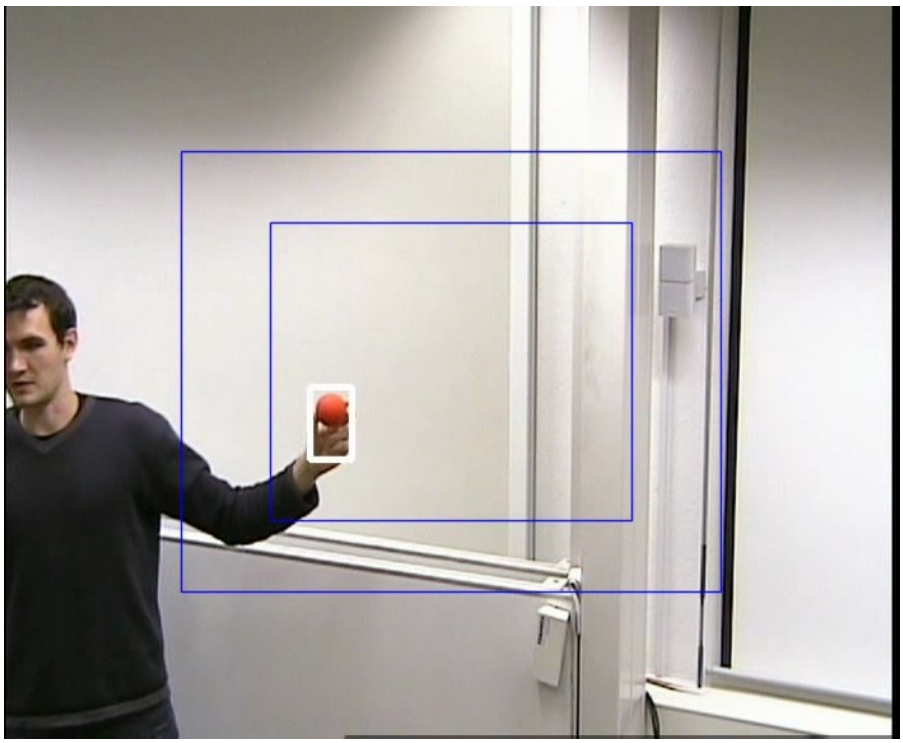

Illustration 13: Test eines Objekts: roter Ball. Der weiße Rahmen um den Ball herum, stellt den Suchbereich dar. Ähnlich bei Illustration 14.

4.1.2 Verfolgung mehrerer Objekte – beiden Verfahren

Test 2.1 V 1/V2: Es werden drei Objekte in die gleich Richtung bewegt.

Test 2.2 V1/V2: Es werden drei Objekte abwechselnd in unterschiedliche Richtungen bewegt.

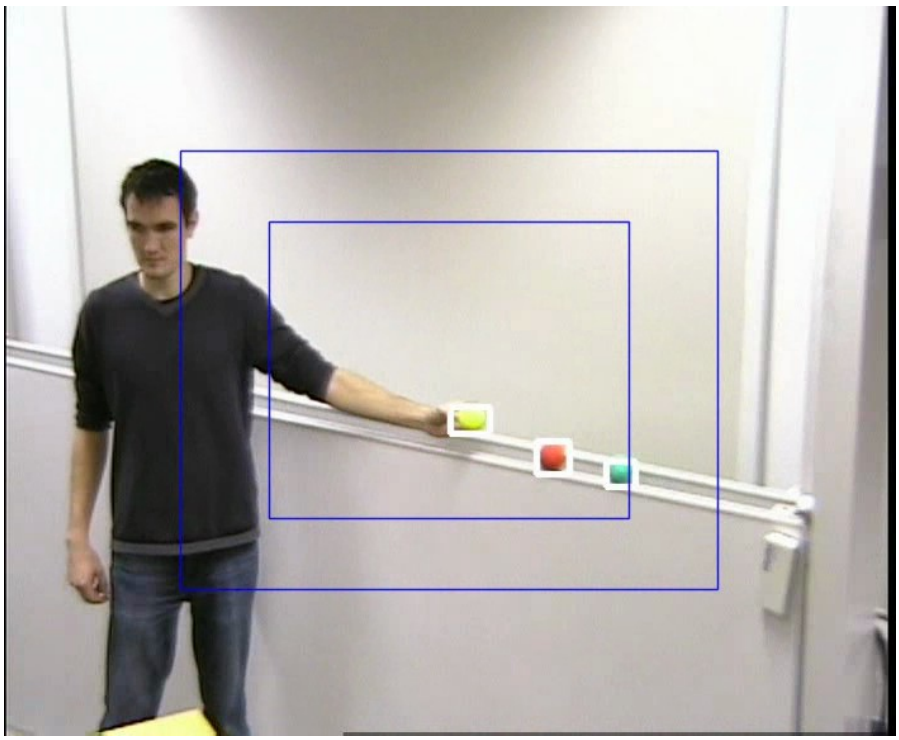

Illustration 14: Test dreier Objekte: roter, grüner und gelber Ball. Um die Bälle herum ist der weiße Rahmen um deren jeweiligen Suchbereiche zu sehen.

4.2 Ergebnisse

Alle Testdurchführungen konnten mit Erfolg abgeschlossen werden. Bei allen Tests der drei Verfahren konnte deren Leistungsfähigkeit gezeigt werden. Auch ungenaue Ergebnisse beeinträchtigen die Nachführung der Kamera nicht. Es hat sich gezeigt, dass die Tracking-Algorithmen die Kamera hinreichend rekonfigurieren.

Bei dem Testaufbau musste allerdings auf die besonderen Anforderung der Bilderkennung an das Umfeld Rücksicht genommen werden. Die für die Testobjekte gewählten Farben durften nicht im Sichtfeld der Ausgangsposition auftreten. Daher müssen sie vorher genau mit Rücksicht auf das Umfeld ausgewählt werden. Trotz der bedachten Wahl kam es vereinzelt zu Fehldetektion. In einigen Fällen wurde das Gesicht des Testers selektiert anstatt der eigentlichen Testobjekte. Da sich diese Fehlerkennung während der Tests als robust erwies, wurde versucht damit zu arbeiten. Dies kommt auch einem möglichen Einsatzzweck der Verfolgung eines Sprechers sehr nahe. Die Verfolgungsalgorithmen haben auch mit

selektierten Gesichtern erfolgreich gearbeitet. Die Kamera wurde in allen Fällen korrekt rekonfiguriert.

5 Zusammenfassung

Die Ziele, die für diese Arbeit gesetzt wurden, konnten erfüllt werden. Es wurde eine aktive Objektverfolgung bestehend aus Objektdetektion, kontinuierlichem Labeling und Kameranachführung entwickelt und implementiert. Zur Umsetzung der Objektdetektion und dem konstanten Labeling wurde eine externe Lösung in das Projekt integriert. Die Kameranachführung arbeitet auf Grundlage der Ergebnisse dieser Analysen. Sie wurde in drei Varianten entwickelt, welche sich durch unterschiedliche Funktionsweisen auszeichnen. Zwei davon können mehrere Objekte kontinuierlich verfolgen. Dabei wird neben deren Positionen zusätzlich auf deren Prioritäten Rücksicht genommen.
Neben dem Modul für die eigentliche Aufgabe, der Kameranachführung, sind weitere Module entstanden. Dazu gehören neben der bereits erwähnten Bilderkennung ein Kameratreiber, der die nativen Steuerkommandos für die Kamera erzeugt, sowie Klassen die einem unabhängigen Zugriff auf ein Netzwerk ermöglichen.
Die Erfüllung der Anforderungen wurden in einem abschließenden Gesamttest nachvollzogen. Alle drei Verfahren mussten dabei in geeigneten Beispielsituationen zeigen, dass sie ihre Aufgaben korrekt durchführen. Trotz teilweise verfälschter Eingaben haben sie dabei alle ihre Erwartungen erfüllt.

6 Ausblick

Weiterführend ist daran zu denken, mehrere Kameras miteinander zu vernetzen, um ein Objekt kontinuierlich über mehrere Kameras hinweg zu verfolgen, und damit eine Lückenlose Überwachung zu gewährleisten. Dieses Projekt wäre eine geeignete Grundlage für diese Weiterentwicklung. Das Spektrum der Einsatzziele lässt sich durch die Variation der

Parameter abdecken.

Der Rahmen der Entwicklung, der durch die Anforderungen dieser Bachelorarbeit vorgegeben war erlaubte es nicht in jedem Fall alle Möglichkeiten umzusetzen. Dies betrifft sowohl die Wahl der Parameterwerte, als auch die Entwicklung ganz anderer Ansätze. Einige dieser weiterführenden Ideen, die nicht nur das aktive Tracking betreffen, sind die Folgenden:

- die Netzwerk- sowie die Treiberfunktionen könnte mittels Multithreading zeitlich unabhängig gemacht werden
- abwägen, ob der Tracker auf den Mittelpunkt oder die äußeren Kanten der Objekte reagieren soll
- wenn das Objekt an einer Kante das Sichtfeld verlässt, dann dorthin für eine bestimmte Distanz weiterschwenken
- den Zoom von der Objektgröße abhängig machen
- die Gewichtung der Objekte bei Tracking von der Qualität der Prioritäten abhängig machen
- die Detektion durch eine andere ersetzen (z.B. face detection, Bewegungserkennung)

7 Quellennachweis

[1] AXIS Communications, AXIS 214 PTZ User's Manual, http://www.axis.com/products/cam_214/

[2] AXIS Communications, Axis VAPIX API, HTTP API, http://www.axis.com/techsup/cam_servers/dev/cam_http_api_index.php

[3] AXIS Communications, Axis VAPIX API, RTSP API, http://www.axis.com/techsup/cam_servers/dev/cam_http_api_index.php

[4] OpenCV (Open Source Computer Vision),

http://sourceforge.net/projects/opencvlibrary

[5] G.R. Bradski, Computer video face tracking for use in a perceptual user interface, Intel Technology Journal, Q2 1998, ftp://download.intel.com/technology/itj/q21998/pdf/camshift.pdf

[6] FFmpeg, http://ffmpeg.org/

[7] Unterstützte Formate, Codecs und Protokolle, http://ffmpeg.org/general.html#SEC4

8 Abkürzungsverzeichnis

HTTP – Hypertext Transfer Protocol
RTSP – Real Time Streaming Protocol
JPEG – Joint Photographic Expert Group
MPEG – Motion Picture Expert Group
AVI – Audio Video Interleave
TCP/IP – Transmission Control Protocol / Internet Protocol

9 Anhänge

Zu dieser Ausarbeitung liegt eine CD-ROM / DVD bei, welche die Quelltexte und die Testvideos beinhaltet. Die Quelltexte sind in eine Unix- und eine Windowsversion aufgeteilt.