Intelligente Kamerasysteme im Anwendungsfeld mobiler Service-Roboter

Dissertation

zur Erlangung des akademischen Grades
Dr. rer. nat.
an der Fakultät für Mathematik, Informatik und
Naturwissenschaften
der Universität Hamburg

eingereicht beim Fach-Promotionsausschuss Informatik von
Hannes Bistry
aus Hamburg

Juni 2013

Gutachter

Prof. Dr. Jianwei Zhang

Dr. Peer Stelldinger

Tag der Disputation: 09.12.2013

Danksagung

Hiermit möchte ich mich herzlich bei Herrn Prof. Dr. Jianwei Zhang dafür bedanken, dass ich an seinem Arbeitsbereich TAMS arbeiten und die vorliegende Dissertation erstellen konnte. Während meiner Beschäftigung hat mir Herr Prof. Dr. Zhang stets ermöglicht, an internationalen Konferenzen und Kooperationen teilzunehmen und eigene Ideen zu verwirklichen. Herrn Dr. Peer Stelldinger danke ich für die Übernahme des Zweitgutachtens sowie für die vielen hilfreichen Ratschläge bei der Erstellung der Dissertation.

Des Weiteren möchte ich allen Mitgliedern des Arbeitsbereichs TAMS danken für die angenehme Arbeitsatmosphäre, die vielen guten Ratschläge und Hilfestellungen. Besonderer Dank gilt auch meiner Familie, die mich während der Zeit der Anfertigung der Dissertation unterstützte.

Hannes Bistry

Abstract

Perceiving the working environment is a central challenge for mobile robots. In the near future, camera systems could contribute significantly to the perception capabilities of artificial intelligent systems. However, the processing of video data is very computationally intensive, especially for high-resolution cameras. On mobile platforms, this issue is critical due to the inherent constraints of size, weight and power consumption. As smart cameras offer lightweight combinations of sensing and processing infrastructure, executing parts of the image processing functions on these systems can yield improvements regarding those challenges.

This thesis presents an integration concept for smart cameras into a robot system and evaluates their application in typical image processing tasks in the field of robotics. Particularly, object detection and face localization are analyzed. Depending on the task and the available processing capabilities, a variable number of image processing subtasks can be carried out on the smart camera. Among other settings, this allows optimizing the pipeline towards either high image processing performance, low utilization of the robot's host PC or low network utilization.

The evaluation shown in this thesis focuses on the performance of the Basler eXcite smart camera. Although this camera's processing power is lower than that of desktop PCs, an appropriate setup of processing functions can result in significantly reduced overall latencies. The employed software's modular architecture also allows offloading image processing functions to further stationary PC-systems in the robot's working environment. For the task of object detection, it is shown that parallel processing can vastly increase the performance, especially when huge databases of objects are used.

Further tests compare the efficiency of the implemented software architecture to the state-of-the-art robotics framework ROS in terms of overhead, that is induced by exchanging data between different software modules. Results confirm a significantly improved behavior in terms of latency and CPU-load.

This thesis also considers future types of smart cameras using multiprocessor or GPU architectures in order to provide improved performance. As part of this investigation, the developed software is extended to support OpenCL-based image processing and then used to validate the advantages of parallel architectures.

Zusammenfassung

Die Erfassung der Arbeitsumgebung stellt für mobile Robotersysteme eine zentrale Herausforderung dar. Kamerasysteme könnten in Zukunft entscheidend zu den Perzeptionsfähigkeiten künstlicher intelligenter Systeme beitragen. Allerdings ist die Verarbeitung von Videodaten sehr rechenintensiv, insbesondere wenn hochauflösende Kameras verwendet werden. Speziell auf mobilen Plattformen stellt dieser Umstand aufgrund der Beschränkung der Recheneinheiten hinsichtlich Größe und Energieverbrauch ein Problem dar. Intelligente Kamerasysteme könnten insoweit zu deutlichen Verbesserungen führen. Durch deren integrierte Recheneinheiten werden Teile der Bildverarbeitung direkt auf den Kamerasystemen ausgeführt.
Bestandteile der Arbeit sind der Entwurf eines Konzepts für die Einbindung von intelligenten Kameras in Robotersysteme und eine Evaluation bei gängigen Bildverarbeitungsanwendungen im Bereich der Robotik; insbesondere wird hierbei auf Objektdetektion und Gesichtslokalisation eingegangen. Je nach Aufgabenstellung und verfügbarer Rechenleistung können mehr oder weniger Teilaufgaben der Bildverarbeitung auf das Kamerasystem ausgelagert werden. So kann die Präferenz beispielsweise auf maximale Performance bei der Bildverarbeitung, minimale Auslastung des Steuerrechners des Robotersystems oder minimale Netzwerkauslastung gelegt werden.
Für die Leistungsmessungen wird die intelligente Kamera Basler eXcite verwendet. Die Tests ergeben, dass diese Kamera in ihrer Leistungsfähigkeit zwar hinter einem Desktop-System zurückbleibt, ihr Einsatz aber durch eine geschickte Verteilung der Teilaufgaben dennoch zu deutlichen Vorteilen im Hinblick auf die Latenz führt. Durch die verwendete modulare Software-Architektur können auch zusätzliche Rechnersysteme im Netzwerk-Umfeld des Robotersystems eingebunden werden. Anhand des Beispiels der Objektdetektion wird dargelegt, wie sich so insbesondere bei einer großen Objektdatenbank eine hohe Steigerung der Verarbeitungsleistung erreichen lässt.
In weiteren Tests wird die Effizienz der implementierten Software mit der Effizienz des weit verbreiteten Frameworks ROS im Hinblick auf den Overhead verglichen, der entsteht, wenn verschiedene Software-Komponenten Daten austauschen. Es stellt sich heraus, dass die hier verwendeten Funktionen eine deutlich geringere Latenz und CPU-Auslastung verursachen.
Im Rahmen der Arbeit wird ferner gezeigt, dass zukünftige intelligente Kameras durch Multi-Core-Prozessoren und durch die Verarbeitung auf den Recheneinheiten von Grafikprozessoren eine Steigerung der Verarbeitungsleistungen erreichen können. Es wird dargelegt, dass die im Rahmen dieser Arbeit entwickelte Softwarearchitektur diese Form der Verarbeitung bereits unterstützt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

In der vorliegenden Arbeit wird untersucht, inwieweit durch intelligente Kamerasysteme die digitale Bildverarbeitung von mobilen Service-Robotern unterstützt werden kann. In diesem Rahmen soll ein autark arbeitendes Softwaresystem für intelligente Kameras entwickelt werden, das die Konfiguration an die aktuelle Aufgabe und verschiedene Randbedingungen anpasst und dabei den Systemen die notwendigen Bildverarbeitungsschritte zuweist. Folgende Eigenschaften und Aspekte sollen bei der Software-Entwicklung berücksichtigt werden:

- Bereitstellung von oft genutzten Funktionen zur Bildverarbeitung
- Vereinfachung der Implementierung von intelligenten Kamerasystemen
- Entlastung des Steuerrechners des Roboters durch Vorverarbeitung
- Erweiterbarkeit, Portierbarkeit und Skalierbarkeit
- Einbindung zusätzlicher Rechnersysteme
- Priorisierte Übertragung von aktuell relevanten Bilddetails

In verschiedenen Szenarien wird das System dann getestet und es werden die Vor- und Nachteile gegenüber einer herkömmlichen Implementierung der Kamerasysteme herausgearbeitet. Zu diesem Zweck werden etablierte Verfahren aus der Bilderkennung ausgeführt und es wird untersucht, wie sich die Auslagerung von Verarbeitungsfunktionen auf die Kamerasysteme im Hinblick auf die Latenz der Bildinformationen, die erreichbare Bildfrequenz, die Netzwerkauslastung sowie auf die Prozessorauslastung auswirkt.

1.1. Motivation

Im Folgenden wird kurz dargelegt, warum diese Themenstellung für die Entwicklung mobiler Robotersysteme relevant ist.

Videodatenverarbeitung zählt zu den rechenintensivsten Anwendungen, die auf Computersystemen ausgeführt werden. Dies liegt hauptsächlich an dem enorm hohen Datenvolumen, das von Kamerasystemen generiert wird. Durch den technischen Fortschritt erreichen digitale Kameras immer höhere Auflösungen und Bildwiederholraten und ermöglichen eine deutliche Steigerung der Qualität der extrahierten Bildinformationen. Allerdings steigt hierdurch die zu verarbeitende Datenmenge und somit auch der Rechenaufwand. Werden auf einem mobilen Robotersystem neben Kameras noch weitere Sensoren und Aktuatoren

von einem zentralen Steuerrechner gesteuert, muss sichergestellt werden, dass die Rechenleistung zur Ansteuerung aller Geräte ausreicht. Gerade bei echtzeitkritischen Geräten wie Aktuatoren kann es zu Problemen führen, wenn das System mit Bildverarbeitungsalgorithmen stark ausgelastet ist. Daher besteht der zentrale Ansatz dieser Arbeit darin, die Bildverarbeitung auf dedizierte Recheneinheiten intelligenter Kameras auszulagern und so dieser Problematik entgegenzuwirken.

1.2. Hintergrund

Eine zentrale Rolle bei der Entstehung des Themas der Arbeit spielt der mobile Service-Roboter TASER des Arbeitsbereichs TAMS. Mit dieser Plattform wird der Einsatz von mobilen Robotern in einem auf Menschen zugeschnittenen Umfeld erforscht. Die Roboterplattform hat etwa die Größe eines Menschen und kann sich über Räder fortbewegen. Auf der Plattform sind mehrere Aktuator- und Sensorsysteme installiert, darunter auch mehrere Kamerasysteme. Beim Betrieb der Plattform ergibt sich das Problem, dass das Verarbeiten der Sensordaten eine hohe Systemauslastung erzeugt und ein gleichzeitiger Betrieb aller Sensoren momentan undenkbar ist. Besonders kritisch verhalten sich die Kamerasysteme aufgrund des großen Datenvolumens, das sie erzeugen, und aufgrund des hohen Rechenaufwands, den gängige Algorithmen der Bildverarbeitung erfordern.

Unter diesen Rahmenbedingungen wurde das Drittmittelprojekt IVUS ins Leben gerufen (2006-2009), in dessen Kontext auch große Teile der Forschungen stattgefunden haben, die dieser Arbeit zugrunde liegen. Bei dem Projekt handelt es sich um eine Kooperation der Universität Hamburg mit der Basler AG aus Ahrensburg, einem Hersteller von Kamerasystemen. Die Projektzielsetzung bestand darin, intelligente Kamerasysteme für mobile Service-Roboter zu entwickeln. Gefördert wurde das Projekt vom BMBF[1] im Rahmen des Förderprogramms „Leitinnovation Servicerobotik“.

1.3. Zielsetzung und Aufgabenstellung

Das Hauptziel dieser Arbeit besteht darin, unter Benutzung der seinerzeit kommerziell erhältlichen Kamera Basler eXcite die Einsatzmöglichkeiten intelligenter Kamerasysteme auf mobilen Service-Robotern zu evaluieren. Eine grundlegende Aufgabe liegt somit darin, Bildverarbeitungsalgorithmen auf der Kamera ausführbar zu machen. Hierdurch soll die Voraussetzung dafür geschaffen werden, dass die Ergebnisse der Bildverarbeitung dem Steuerrechner des Roboters für die Planung der Aktionen oder für weitere Verarbeitungsschritte zur Verfügung stehen. Es sollen zum einen einfachere Operationen auf den Bilddaten implementiert werden, zum anderen sollen die Roboter-typischen Szenarien Gesichtslokalisation und Objekterkennung untersucht werden.

[1] Bundesministerium für Bildung und Forschung (www.bmbf.de)

Hierbei sollen die Vor- und Nachteile des Einsatzes der intelligenten Kamera untersucht werden. Die Kriterien bei der Untersuchung sind die Latenz der Bilddaten, die Bild-Wiederholrate sowie die Auslastung der beteiligten Recheneinheiten und Netzwerkschnittstellen. Es sollen insbesondere Probleme der aktuellen Generation von intelligenten Kameras ausgemacht und mögliche Lösungswege aufgezeigt werden.

Im Verlauf der Arbeit zeigt sich die Notwendigkeit, die Bildverarbeitung in mehrere Teilaufgaben zu unterteilen, die zum Teil auch auf dem Steuerrechner des Service-Roboters ausgeführt werden. Dies eröffnet softwaretechnisch zugleich die Möglichkeit, beliebigen anderen Systemen Teilaufgaben zuzuordnen. Experimente zu diesem so genannten Cloud-Computing sind ebenfalls Bestandteil der Arbeit.

Die Vielzahl der Konfigurationsmöglichkeiten führt zu der Überlegung, inwiefern es möglich ist, automatisch eine geeignete Konfiguration zu finden. Zu dieser Thematik wird ebenfalls ein Lösungsansatz präsentiert.

Es ist ausdrücklich nicht das Ziel dieser Arbeit, die Bildverarbeitungsalgorithmen als solche zu optimieren oder zu evaluieren. Bei der Auswahl der Algorithmen und bei den Experimenten wird jedoch darauf geachtet, dass die Algorithmen unter den gegebenen Bedingungen funktionsfähig sind und verwendbare Ergebnisse liefern.

1.4. Aufbau der Arbeit

Zunächst werden in Kapitel 2 die Grundlagen der Themengebiete dargestellt, die für diese Arbeit von Bedeutung sind. Hierbei wird der Stand der Forschung auf den jeweiligen Themengebieten dargelegt. Zu den relevanten Themengebieten zählen unter anderem digitale Kamerasysteme, ihre Anwendung in der Robotik und die dazugehörigen Algorithmen. In diesem Zusammenhang werden auch intelligente Kamerasysteme vorgestellt und es wird auf ähnliche Forschungsansätze eingegangen. Die Entscheidung, Grundlagen und verwandte Forschung in einem Kapitel darzustellen, liegt darin begründet, dass das Thema Robotik noch ein relativ junges Forschungsthema ist und somit grundlegende Erkenntnisse der Robotik meistens an konkrete Forschungsarbeiten gebunden sind.

Das darauf folgende Kapitel 3 erläutert die Steuerungsarchitektur, die im Rahmen dieser Arbeit für die Kamerasysteme entwickelt wird. Zunächst werden das verwendete intelligente Kamerasystem Basler eXcite sowie der Service-Roboter TASER behandelt. Bei der Vorstellung der Roboterplattform werden die Probleme der aktuellen Implementierung der Kamerasysteme und die möglichen Vorteile des Einsatzes intelligenter Kamerasysteme dargelegt. Hierauf aufbauend wird das im Rahmen dieser Arbeit entwickelte Konzept zur Integration der intelligenten Kameras erklärt. Bei diesem Konzept werden die Algorithmen zur Bildverarbeitung als modulare Funktionen implementiert, die zu Pipelines zusammengefügt werden. Die Software basiert auf dem GStreamer-Framework.
Bei der Beschreibung des Konzepts wird auf Aspekte wie Effizienz, Skalierbarkeit, Erweiterbarkeit und Portierbarkeit eingegangen. Die implementierte Steuerungsarchitektur

erlaubt es, von einer zentralen Instanz aus intelligente Kameras und weitere Systeme zu steuern. Innerhalb der Software ist es möglich, den zeitlichen Ablauf der Bildverarbeitungsschritte zu analysieren und gegebenenfalls zu optimieren. In den weiteren Abschnitten dieses Kapitels werden die implementierten Funktionen vorgestellt, die unter anderem die Verwendung von OpenCV-Funktionen, Gesichtslokalisation, Objektdetektion und Datenaustausch mit dem weit verbreiteten Robotik-Framework ROS ermöglichen.

Nach der Vorstellung der implementierten Software werden in Kapitel 4 Tests beschrieben, anhand derer die implementierte Architektur und die Einsatzmöglichkeiten intelligenter Kameras unter verschiedenen Gesichtspunkten untersucht werden.
Zunächst geht es im Abschnitt 4.1 um grundlegende Tests zur Leistungsmessung der verwendeten intelligenten Kamera Basler eXcite und einen Vergleich mit dem Steuerrechner des Service-Roboters TASER. In weiteren Tests wird dann untersucht, inwieweit Parallelverarbeitung mittels der implementierten Software möglich ist und welche Vorteile sie hervorbringt. Diese Untersuchungen erfolgen sowohl im Hinblick auf die Parallelverarbeitung auf verteilten Systemen als auch im Hinblick auf die Einsatzmöglichkeiten zukünftiger intelligenter Kameras mit mehreren CPU-Kernen.
Unter Zugrundelegung der Testergebnisse werden komplette Szenarien analysiert, bei denen die intelligente Kamera im Verbund mit dem Steuerrechner des Service-Roboters und weiteren Systemen komplexe Bildverarbeitungsaufgaben ausführt. In dem in Abschnitt 4.2 beschriebenen Szenario werden die Regionen aus dem Videodatenstrom extrahiert, in denen Gesichter abgebildet sind. Es wird dargelegt, wie eine Verteilung der Softwarekomponenten auf die Kamera und den Steuerrechner sowohl die Latenz als auch die Netzwerkauslastung deutlich senkt.
Bei dem im darauf folgenden Abschnitt 4.3 beschriebenen Test wird im Kamerabild nach Objekten gesucht. Hierzu erfolgt ein Vergleich mit einer Datenbank von 100 Objekten. Bei diesem Test wird von der Möglichkeit Gebrauch gemacht, Teile der Bildverarbeitungsfunktionen auf Systeme im Umfeld des Roboters auszulagern. Die sich ergebenden erheblichen Leistungssteigerungen werden detailliert erklärt.
Im nächsten Abschnitt wird ausgeführt, wie es möglich ist, automatisch eine Bildverarbeitungsaufgabe auf verschiedene verfügbare Systeme zu verteilen und dabei verschiedene Kriterien wie Latenz und Netzwerkauslastung zu berücksichtigen. Hierzu wird ein Verfahren des maschinellen Lernens vorgestellt, mithilfe dessen in möglichst kurzer Zeit eine sinnvolle Lösung gefunden werden kann.
In Abschnitt 4.5 werden Untersuchungen der implementierten Funktionen zum Datenaustausch mit dem ROS-Framework beschrieben. Dieser Teil ist auch als Vergleich des hier implementierten Konzepts mit dem ROS-Framework aufzufassen. Es wird aufgezeigt, dass das im Rahmen dieser Arbeit gewählte Konzept der Implementierung einer modularen Architektur im Vergleich zu ROS deutlich effizienter ist.
Abschnitt 4.6 beschreibt, wie sich in der implementierten Softwarearchitektur auch die Recheneinheiten von Grafikkarten nutzen lassen. Dies ist insbesondere deswegen von Interesse, weil eine frei programmierbare intelligente Kamera angekündigt ist, die über solche Verarbeitungseinheiten verfügt. Es folgt ein Test von verschiedenen Grafikkarten und eine

erste Abschätzung der Leistungsfähigkeit der angekündigten intelligenten Kamera.
In Abschnitt 4.7 wird dargelegt, wie die im Rahmen dieser Arbeit implementierte Softwarearchitektur genutzt werden kann, um ein weiteres intelligentes Kamerasystem zu konstruieren. Die Besonderheit bei diesem System ist, dass es aus vier Einzelbildern ein Panoramabild erzeugt und hierzu auch die Recheneinheiten von Grafikkarten nutzt.

Kapitel 5 fasst die Ergebnisse dieser Arbeit zusammen und gibt eine Einschätzung zu den Einsatzmöglichkeiten jetziger und zukünftiger intelligenter Kamerasysteme. Darüber hinaus werden mögliche Anknüpfungspunkte für zukünftige Forschungsprojekte aufgezeigt.

2. Grundlagen und verwandte Forschung

Dieses Kapitel befasst sich mit den Grundlagen, die für diese Arbeit relevant sind, und stellt den Stand der Forschung auf den jeweiligen Themengebieten dar. Um an das Forschungsthema dieser Arbeit heranzuführen, wird zunächst ein allgemeiner Überblick über die Themen Robotik und Sensorik gegeben und anschließend zu dem Einsatz von Kameras in der Robotik und den verwendeten Algorithmen übergeleitet. Es folgen Ausführungen zu den Grundlagen von digitalen Kamerasystemen. Danach werden intelligente Kamerasysteme und deren Konzepte ausführlich dargestellt.

In den nächsten Abschnitten 2.4 und 2.5 werden Eingebettete Systeme und verteilte Systeme in der Robotik behandelt. Das zuletzt genannte Themengebiet ist deswegen relevant, weil es sich beim Verbund von Roboterplattform und intelligenter Kamera um ein verteiltes System handelt. Die Aufgabenzuweisung in Bildverarbeitungssystemen ist Gegenstand des folgenden Abschnitts 2.6. Da die Konfiguration bei der späteren Realisierung der Verarbeitungsstrategie datenstromorientiert über ein grafisches Benutzerinterface erfolgt, wird dieses Programmiermodell im vorletzten Abschnitt vorgestellt. Abschließend folgt in Abschnitt 2.8 eine Einordnung des eigenen Forschungsvorhabens im Vergleich zu ähnlichen Arbeiten.

2.1. Robotik und Sensorik

> „I can't define a robot, but I know one when I see one.“ (Joseph Engelberger, vgl.[CBC07, wRo12])

Der Begriff „Roboter“ entstammt einem Drama aus den 20er Jahren des vorigen Jahrhunderts (vgl. [ČN04]). Das Drama handelt von menschenähnlichen Maschinen, die für die Menschen Arbeiten erledigen sollen. Im Verlauf der Handlung kommt es zu einem Aufstand dieser Maschinen gegen die Menschheit.

Heutzutage ist der Begriff Roboter sehr weit gefasst. Eine allgemeingültige Definition ist nicht festgelegt. Die verschiedenen Definitionen des Begriffes [Ric, def12] bilden den Konsens, dass es sich bei einem Roboter um eine Maschine zum Bewegen von Gegenständen handelt. Diese Maschine soll programmierbar sein und über mehrere Freiheitsgrade

verfügen. [1]

Roboter können anhand von verschiedensten Kriterien klassifiziert werden. Ein entscheidendes Kriterium ist, ob es sich um einen stationären oder einen mobilen Roboter handelt. Stationäre Roboter werden oft in der industriellen Produktion eingesetzt. Bei den mobilen Robotern ist die Art der Fortbewegung und damit die Arbeitsumgebung von entscheidender Bedeutung. Beispiele für Roboter mit verschiedenen Fortbewegungsmechanismen:

- autonome Fluggeräte: Hubschrauber [AXZ12], Flächenflugzeuge [FMK+04], Raketen
- Schwimmroboter: autonome Schiffe [Man08], autonome U-Boote [BU01], biologisch inspirierte Roboterfische [H+00]
- Laufroboter: biologisch inspiriert, z.B. Humanoide Roboter [SWA+02], insektenähnliche Fortbewegung [AMK+01]
- Roboter mit Rädern, angepasst z.B. an eine Büroumgebung [Cou10], Fortbewegung im Gelände [SKI05] oder im Straßenverkehr [TMD+06]
- Roboter mit weiteren Antriebsformen, z.B. ähnlich einer Raupe über bewegliche Glieder [WWW+08]

Roboter können auch anhand ihres Einsatzzweckes betrachtet werden. Beispielsweise existieren Erkundungs-Roboter, Militär-Roboter, Edutainment-Roboter und Service-Roboter. Im Rahmen dieser Arbeit sind mobile Service-Roboter von Bedeutung. Service-Roboter im Allgemeinen werden von der International Federation of Robotics folgendermaßen definiert:

> „A service robot is a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations. “ [ifr13]

Somit wird hier eine klare Abgrenzung zwischen Industrie- und Service-Robotern gezogen. Beispiele für Aufgaben, die durch Service-Roboter für Menschen erledigt werden können, sind Staubsaugen [UMN97], Rasenmähen [HCO+87] oder Transport-Aufgaben [HGTH04].

Die Erfassung der Arbeitsumgebung mobiler Roboter erfolgt in der Regel durch verschiedene Sensoren. Ein Überblick über gängige Sensoren verschafft [Fra04]. Dort wird für Sensoren folgende Definition gegeben.

> „A sensor is a device that receives a stimulus and responds with an electrical signal.“ [Fra04]

Der Stimulus, auf den der Sensor mit einem elektrischen Signal reagiert, wird abstrakt beschrieben als eine Quantität, eine Eigenschaft oder ein Zustand. Es können mehrere Unterscheidungskriterien für Sensoren definiert werden. Zunächst muss unterschieden werden, ob der Sensor innerhalb (intrinsisch) oder außerhalb eines Systems (extrinsisch) platziert ist. Des Weiteren ist zu unterscheiden, ob der Sensor einen systeminternen Zustand erfasst oder ob er einen Zustand aus der Umgebung des Robotersystems wahrnimmt. Jeder

[1] Die Anzahl der Freiheitsgrade entspricht prinzipiell der Anzahl der Gelenke eines Roboters

Sensor kann auch als Energiewandler aufgefasst werden, da immer ein Fluss von Energie vom zu messenden Objekt zum Sensor stattfindet. Hierbei muss zwischen aktiven und passiven Sensoren unterschieden werden. Passive Sensoren erzeugen aus dem Stimulus direkt ein elektrisches Signal (z.B. Piezoelektrischer Sensor), während aktive Sensoren eine zusätzliche Energiequelle benötigen. Wenn Sensoren betrachtet werden, die an ein Computersystem angebunden werden, so ist praktisch immer eine zusätzliche Energieversorgung notwendig. Oftmals wird daher im Bereich der Robotik der Unterschied zwischen aktiven und passiven Sensoren so aufgefasst (z.B. in [Win07]), dass aktive Sensoren ein Signal in die Umgebung aussenden (z.B. Sonarimpulse, Lichtimpulse) und die Reflektion detektieren, während passive Sensoren lediglich Stimuli detektieren, die sowieso in der Umgebung vorhanden sind.

Die grundlegende Form des Einsatzes von Sensoren in Robotersystemen kann als „Sense-Plan-Act"-Architektur beschrieben werden (vgl. [OC03]). Hierbei wird über Sensoren ein Zustand detektiert und basierend auf diesem Zustand eine Aktion geplant und ausgeführt. So sollen auch bei dem hier zu implementierenden Konzept von dem intelligenten Kamerasystem verarbeitete Sensordaten bereitgestellt werden, die dann bei der Planung der folgenden Aktionen berücksichtigt werden (beispielsweise Greifen eines erkannten Gegenstandes, Kontaktaufnahme mit einer Person).

Ein weiteres entscheidendes Kriterium für die Klassifikation von Sensoren ist die Art des Stimulus (z.B. elektrisch, akustisch, chemisch, optisch oder mechanisch). Die im Rahmen dieser Arbeit betrachteten Kamerasysteme sind optische Sensoren. Sie messen unter verschiedenen Winkeln die Intensität (Helligkeit) und Wellenlänge (Farbe)[2] der eintreffenden elektromagnetischen Strahlung.

2.1.1. Einsatz von Kameras in der Robotik

Höherentwickelte Lebewesen verdanken einen Großteil ihrer Fähigkeiten ihrem natürlichen Sichtsystem (vgl. [Ing08]). Es versetzt die Lebewesen in die Lage, sich in der Umgebung zu orientieren, Hindernisse und Gefahren zu erkennen, Nahrung zu finden sowie andere Lebewesen und deren Mimik und Gesten zu erkennen. Bei der Entwicklung von Robotersystemen, die den Menschen in seinem Handeln unterstützen sollen, besteht dementsprechend das Bestreben, einige der oben genannten Fähigkeiten auf der Basis von Kamerasystemen zu implementieren.

Bereits in den Anfängen der Robotik wurde die Verwendung von Sichtsystemen in Erwägung gezogen [Hor86]. In erster Linie wurde angestrebt, die Vielseitigkeit von Manipulatoren zu erhöhen, indem durch Kameras die Position und Orientierung von Werkstücken bestimmt werden kann. Auch für mobile Roboter ergeben sich viele Möglichkeiten des Einsatzes von Kamerasystemen. In [Aya91] wird die Gewinnung von dreidimensionalen Bildinformationen durch Stereo-Kamerasysteme beschrieben. Ziel ist es, mobilen Robotern

[2] Die Wellenlänge bzw. physikalisch genauer das Spektrum wird nicht direkt gemessen, sondern in der Form bestimmt, wie ein menschliches Auge es wahrnimmt.

die Erkundung von unbekannten Umgebungen zu ermöglichen. Eine weitere Anwendung von Kameras auf mobilen Robotern wird in [CB92] vorgestellt. Hierbei steht das Ziel im Vordergrund, ein Objekt zu verfolgen (Objekt-Tracking). Die Implementierung soll in Echtzeit ausführbar sein.

Gegenwärtig sind Kamerasysteme für mobile Roboter weiterhin ein Thema, an dem aktiv geforscht wird. Es existieren vielfältige Anwendungsmöglichkeiten:

- Selbstlokalisierung und Kartenbildung [DCK04, DRMS07, JR11],
- Erkennung von Hindernissen [CC11, HFFW06],
- Objekt-Tracking [HHC+12, SEI11],
- Personenerkennung [LJ11, WGR11, VLDS+08],
- Gegenstandserkennung [QBG+09, PPC12],
- Visuell geführtes Greifen [FLZ12].

Einige dieser Bereiche, die für diese Arbeit relevant sind, werden nachfolgend kurz umrissen.

Personenerkennung

Um im Arbeitsumfeld die anwesenden Personen zu erkennen und mit ihnen interagieren zu können, werden auf Robotersystemen oft Algorithmen zur Gesichtsdetektion ausgeführt. Das Wissen über die interagierende Person könnte beispielsweise bei einer Aufgabenstellung des Robotersystems mit berücksichtigt werden.

Gesichtsdetektion ist ein aktiv erforschtes Gebiet in der maschinellen Bildverarbeitung. Ein Überblick über den aktuellen Stand der Forschung liefert [LJ11]. Generell muss zwischen Gesichtsdetektion und Gesichtslokalisation unterschieden werden. Mit Gesichtslokalisation ist gemeint, dass Regionen im Bild bestimmt werden, in denen sich beliebige Gesichter befinden. Die Zuordnung des Gesichts zu (dem System bekannten) Personen soll hier als Gesichtsdetektion bezeichnet werden.

Um Algorithmen zur Gesichtsdetektion einzusetzen, muss im ersten Schritt eine Gesichtslokalisation durchgeführt werden. Hierfür sind Haar-Klassifikatoren eine weit verbreitete Technik [VJ01]. Multimodale Ansätze, wie die Sensor-Daten-Fusion mit Lasermesssystemen, können die Genauigkeit der Daten erhöhen [BH09]. Im Kontext eines mobilen Roboters wird die Gesichtslokalisation in [DHPS07] angewendet. Hier werden etablierte Gesichtslokalisationsalgorithmen durch eine Vorabauswahl von Bildbereichen ergänzt, bei denen die Annahme plausibel ist, dass sich dort Gesichter befinden können. Es werden zum einen Daten von Abstandsmesssystemen und zum anderen Vorwissen über die übliche Größe von Personen mit einbezogen. Der Rechenaufwand und die Fehlerkennungsrate konnten gegenüber dem Anwenden der Algorithmen auf das gesamte Bild stark gesenkt werden.

Bei der Gesichtsdetektion wird der Bildbereich, der das Gesicht enthält, mit einer Datenbasis abgeglichen. Hierfür existieren verschiedene Methoden; weit verbreitet sind Eigenfaces [TP91], Support-Vector-Maschinen [GLC00] und neuronale Netzwerke [LGTB97].

Visuell geführtes Greifen

Bei der Steuerung von Manipulatoren können ebenfalls Videodaten einfließen. Solche Algorithmen werden als „Visual Servoing“ bezeichnet. Die Bilddaten dienen hier als Feedback für die Steuerung der Bewegung eines Aktuators. Ein Überblick über unterschiedliche Ansätze findet sich in [CH06, CH07]. Ein Forschungsprojekt, in dem auf der Basis von Videodaten ein Manipulator gesteuert wurde, ist in [RZH02] beschrieben. Hier werden die Videodaten für die Anfangspositionierung eines Zwei-Backen-Greifers verwendet, um ein Konstrukt bestehend aus Baufix-Elementen zu greifen.

Objekterkennung

Es existieren verschiedenen Ansätze zur Detektion von Objekten. Für die Objekterkennung mit traditionellen Kamerasystemen werden häufig Algorithmen verwendet, die auf sogenannten Feature-Punkten basieren. Diese Feature-Punkte werden in der Szene detektiert und mit den Feature-Punkten bekannter Objekte verglichen. Beispiele hierfür sind SIFT [Low99] und SURF [BTVG06].

Werden genug übereinstimmende Punkte gefunden (mindestens vier), lässt sich die Position und Orientierung des Objektes im Raum berechnen, vorausgesetzt, die Kalibrierungsdaten sind vorhanden und die Position der Punkte bezüglich eines Objektkoordinatensystems sind bekannt (vgl. Abschnitt 2.2.3).

Der Ansatz der Feature-Punkte wird durch den „Line-Mod“-Ansatz in [HBN+08] erweitert. Auch dieser Algorithmus arbeitet auf der Basis von signifikanten Bildpunkten, definiert bei der Beschreibung aber die gesamte Fläche um diese Punkte. Bei der Detektion wird auch die Orientierung berücksichtigt.

Sollen dreidimensionale Bildinformationen mit klassischen Kamerasystemen generiert werden, werden in der Regel zwei oder mehr Kameras verwendet.

Mit dem Begriff „Deep Learning“ bezeichnet man eine neue Klasse von Verfahren zum Trainieren von neuronalen Netzen mit mehreren Schichten [Hin07]. Einsatzgebiet dieser Netze kann auch die Klassifikation von Bilddaten sein. Das Lernverfahren zeichnet sich dadurch aus, dass während des Lernvorgangs von dem modellierten Netz bestimmte Muster erzeugt werden, die dann mit den Eingabedaten verglichen werden. Als weitere Eigenschaft werden bei diesem Verfahren die einzelnen Schichten des Netzes separat trainiert, was einen effizienteren Lernvorgang ermöglicht.

Eine Anwendung dieses Verfahrens wird in [LRM+11] gezeigt. Hier wird in einem Rechencluster mit 1000 Rechnern mit jeweils 16 CPU-Kernen eine Art menschliches Gehirn

simuliert. In diesem neuronalen Netz werden eine Milliarde Verbindungen zwischen den Neuronen trainiert. Als Trainingsdaten für dieses Netz dienen zehn Millionen Bilder mit einer Auflösung von jeweils 200 x 200 Pixeln, die zufällig aus Youtube-Videos ausgewählt sind. Besonderheit bei dieser Form des Lernens ist, dass es sich um sogenannte ungelabelte Trainingsdaten handelt. Dem System werden also keinerlei Informationen übergeben, was sich an welcher Stelle in den Testbildern befindet. Es kann gezeigt werden, dass das Netz selbstständig bestimmte Konzepte lernt und bestimmte Neuronen des Netzes auf bestimmte Objekttypen reagieren. Zur Bestimmung der Erkennungsleistung wird überprüft, inwiefern das System in der Lage ist, 22000 verschiedene Objektkategorien zu erkennen. Es ergibt sich eine korrekte Zuordnung in 15,8 % der Fälle, was zwar im Vergleich zur Leistung des menschlichen Gehirns gering ist, aber im Vergleich zu bisherigen Verfahren eine deutliche Steigerung bedeutet. Bei der großen Anzahl von Objektkategorien läge die Wahrscheinlichkeit einer zufälligen korrekten Zuordnung im Bereich von 0,005 %. Wird nicht die Zuordnung zu einer Klasse, sondern die binäre Klassifikation untersucht, bei der entschieden wird, ob in einem Bild ein bestimmtes Objekt gezeigt ist, ergeben sich deutlich höhere Trefferquoten. Diese liegen beispielsweise bei Gesichtern von Menschen und Katzen im Bereich zwischen 75 % und 80 %. Einerseits zeigt diese Forschung ein hohes Potential von neuartigen Verfahren der künstlichen Intelligenz, andererseits verdeutlicht es den enormen Rechenaufwand bei der Realisierung.

Ein weiterer Ansatz, der unter die Kategorie „Deep Learning“ fällt, wird in [HQZ12] vorgestellt. Hier wird versucht, die Effizienz des Lernverfahrens durch bestimmte Verhaltensregeln der Neuronen zu steigern (K-Means-Algorithmus). Der von den Autoren vorgestellte Algorithmus liefert auf den getesteten Datensätzen verglichen mit etablierten Verfahren ähnliche Ergebnisse, benötigt hierfür jedoch nur einen Bruchteil der Rechenzeit.

2.2. Digitale Kamerasysteme

Dieser Abschnitt befasst sich mit den Grundlagen von digitalen Kamerasystemen. Zunächst soll kurz die Funktionsweise von Kamerasystemen im Allgemeinen behandelt werden. Darauf aufbauend soll betrachtet werden, wie die Aufnahme von Bildern bei digitalen Kamerasystemen funktioniert. Der Schwerpunkt wird auf Faktoren gelegt, die bei der Bildaufnahme insbesondere auf mobilen Robotersystemen eine Rolle spielen.

2.2.1. Grundlagen von Kamerasystemen

Generelle Informationen zur Funktionsweise von Kamerasystemen liefert beispielsweise [AB89]. Die grundlegende Funktion eines Kamerasystems besteht darin, eine Szene auf eine Bildebene zu projizieren. Bei analogen Kameras befindet sich auf der Bildebene ein Film, der belichtet wird, bei digitalen Kameras ein Bildsensor. Die Projektion der Szene kann anhand des Lochkameramodells verdeutlicht werden. Zum Verständnis soll zunächst

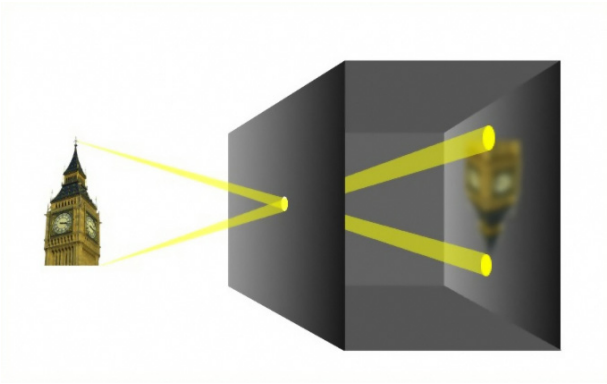


Abbildung 2.1.: Die Funktionsweise einer Lochbildkamera. Ein Objekt (links) wird auf die Bildebene (rechts) der Kamera abgebildet.(Quelle: Anton/de.wikipedia unter cc-by-sa[3])

kurz und stark vereinfacht auf das Strahlenmodell des Lichts eingegangen werden. Die Oberflächen von Objekten reflektieren und absorbieren einfallende Lichtstrahlen abhängig von ihrer Wellenlänge und ihrem Einfallswinkel. Im Allgemeinen reflektieren matte Oberflächen das Licht unabhängig von dem Einfallswinkel in eine beliebige Richtung, während bei reflektierenden Oberflächen der Einfallswinkel zum Lot der Oberfläche gleich dem negativen Ausfallwinkel ist. Es existieren auch Oberflächen, die teilweise reflektieren. Eine entscheidende Rolle spielen des Weiteren Lichtquellen, von denen Lichtstrahlen emittiert werden. Das Spektrum einer Lichtquelle beschreibt, aus welchen Wellenlängen sich das Licht zusammensetzt. Dieses bestimmt letztendlich die wahrgenommene Farbe des Lichtes.

Bei einer Lochkamera treffen nun die von einem Punkt in Richtung des Loches ausgehenden Lichtstrahlen auf einem Punkt der Bildebene auf (s. Abbildung 2.1). Dieses setzt voraus, dass das Loch einen infinitesimal kleinen Durchmesser hat, was in der Praxis nicht zutrifft. Je nach Größe des Loches treffen die von einem Punkt ausgehenden Strahlen vielmehr auf einem Bereich der Bildebene auf. Da nun auf einem Punkt der Bildebene Strahlen von mehreren Punkten der Szene auftreffen, führt dieses, wie auch in oben erwähnter Abbildung angedeutet, zu einem unscharfen Bild. Andererseits führt nur eine ausreichend große Öffnung dazu, dass die eintreffende Lichtmenge groß genug ist.

Aufgrund dieser Einschränkung haben Lochkameras geringe Bedeutung und dienen hauptsächlich zur Verdeutlichung der Funktionsweise. Stattdessen befindet sich an der Stelle des Loches ein Objektiv, das alle von einem Punkt der Szene auf das Objektiv fallende Lichtstrahlen auf einen Bildpunkt projiziert (s. Abbildung 2.2). Dadurch kann eine deutlich größere Lichtmenge eingefangen werden. Hierbei ist die sogenannte Brennweite des Objektivs zu berücksichtigen. Dieser Wert besagt, in welchem Abstand parallel in das Objektiv eintreffende Strahlen auf einen Punkt gebündelt werden. Die Brennweite unterscheidet

[3]Creative Commons Lizenz, Details siehe http://creativecommons.org/licenses/by-sa/3.0/deed.en

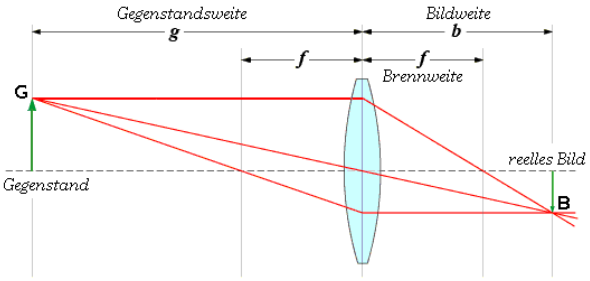


Abbildung 2.2.: Die Funktionsweise einer Kamera mit Sammellinse. (Quelle: Anastasius Zwerg/de.wikipedia unter cc-by-sa)

sich bei den verschiedenen Objektiven und wird hauptsächlich von der Krümmung der Linse beeinflusst. Um ein Objekt mit einer Linse scharf auf die Bildebene abzubilden, muss die folgende Linsengleichung erfüllt sein:

$$\frac{1}{f} = \frac{1}{b} + \frac{1}{g}$$

Hierbei ist f die Brennweite, g die Gegenstandsweite, also der Abstand des Gegenstandes zur Ebene durch die Linse, und b die Bildweite, also der Abstand der Bildebene zur Linse. Das bedeutet in der Praxis, dass nur Objekte mit einem bestimmten Abstand zur Kamera scharf abgebildet werden können und die Bildweite auf den entsprechenden Wert justiert werden muss. Ein Sonderfall ergibt sich, wenn die Gegenstandsweite g gegen unendlich strebt: In diesem Fall strebt die einzustellende Bildweite b gegen die Brennweite f der Kamera. Dies entspricht dem oben erläuterten Fall, da die von dem Gegenstand ausgehenden Lichtstrahlen dann nahezu parallel sind. Das Einstellen der Bildweite wird im Allgemeinen als Fokussieren bezeichnet.

In Abhängigkeit von der Brennweite der Linse ändert sich - bei gleichbleibender Größe des Bildebene - zudem der Bildwinkel und somit der abgebildete Bildausschnitt. Es gilt für den horizontalen Bildwinkel folgende mathematische Beziehung:

$$\alpha = 2 \cdot arctan(\frac{d}{2 \cdot f})$$

Hierbei ist d die Breite der Bildebene (also z.B. des Sensors).

Bei einigen Objektiven ist die Brennweite und somit der Bildwinkel variabel. Diese werden als Zoom-Objektive bezeichnet.

Ein weiterer wichtiger Aspekt bei der Abbildung in einer Kamera ist die Blendenzahl. Diese bestimmt die einfallende Lichtmenge und ist mathematisch als Verhältnis der Brennweite zum Durchmesser der Eintrittspupille (D) definiert:

$$k = \frac{f}{D}$$

Bei Objektiven kann die Eintrittspupille und somit die Blendenzahl oft durch eine mechanische Blende variiert werden, wobei die maximale Blendenöffnung durch die Größe des Objektivs vorgegeben ist. Als Vorteil bei der Verkleinerung der Blende ergibt sich, dass ein größerer Entfernungsbereich scharf auf den Sensor abgebildet wird. Diese sogenannte Schärfentiefe hängt direkt von der Blendenöffnung ab. Als Nachteil ergibt sich jedoch eine geringere Lichtmenge, die auf die Bildebene trifft. Hier ist abhängig von der Szene eine geeignete Abstimmung zu finden. Eine oftmals gewählte Einstellung ist es, Blende und Fokus so einzustellen, dass ab einer bestimmten Entfernung bis ins Unendliche alle Objekte scharf abgebildet werden.

Die Belichtungszeit bestimmt den Zeitraum, in dem Licht auf die Bildebene trifft. Bei analogen Kameras wird die Belichtungszeit über einen mechanischen Verschluss gesteuert, während digitale Kameras[4] auch elektronisch den Zeitraum festlegen können, in dem der Bildsensor einfallendes Licht registriert. Über die Belichtungszeit kann wie auch über die Blende die Lichtmenge gesteuert werden. Als negativer Effekt tritt hier ein, dass bei längerer Belichtungszeit Bewegungen sowohl der Kamera als auch der aufzunehmenden Objekte zu Unschärfe führen.

2.2.2. Funktionsweise digitaler Kameras

Bei analogen Kameras befindet sich auf der Bildebene ein Film, der belichtet wird und sich dadurch chemisch verändert. Durch weitere Prozesse kann von dem belichteten Film ein Foto erzeugt werden.

Bei digitalen Kameras befindet sich an der Stelle des Films ein Sensor, der das einfallende Licht in ein elektrisches Signal verwandelt. Der Sensor besteht aus einzelnen Bildpunkten, den Pixeln, die meist matrixartig als „Array“ angeordnet sind. Die Anzahl an Pixeln wird als Auflösung der Kamera bezeichnet. Die Anzahl der Pixel kann für Bildhöhe und Bildbreite verschieden sein. Die Abmessung der einzelnen Pixel ist oft quadratisch, kann jedoch auch ein anderes Seitenverhältnis aufweisen. Über eine entsprechende Ansteuerungselektronik können die Sensoren für einen Zeitraum für das eintreffende Licht empfindlich geschaltet werden. In den einzelnen Pixeln werden durch das eintreffende Licht Elektronen bewegt, wodurch der Ladezustand der einzelnen Pixel verändert.

Es gibt zwei verbreitete Formen von digitalen Bildsensoren (vgl. [RB10]), die sich hauptsächlich durch die Ansteuerung und das Auslesen der Pixelwerte unterscheiden. Bei CCD-Sensoren werden die Ladungen in den Pixeln jeweils über eine Verschiebelogik zum Nachbarpixel geleitet. Am Rand der Bildmatrix befindet sich die Auswertelogik. Es ergibt sich ein festgelegtes Auslesemuster und es wird stets das gesamte Bild ausgelesen.

[4] Ausnahme digitale Spiegelreflexkameras

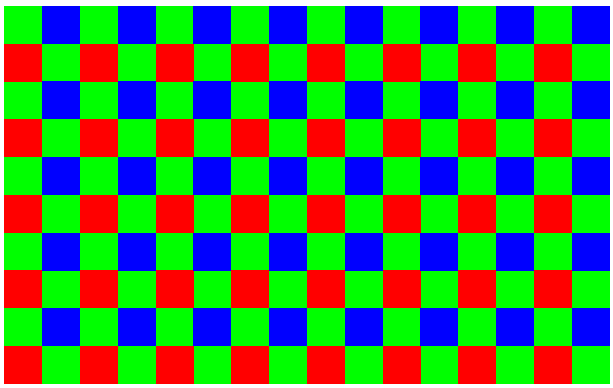


Abbildung 2.3.: Bayer Matrix. Die Farbfilter über Bildsensoren sind oft in Form der Bayer Matrix angeordnet und lassen nur rotes, grünes oder blaues Licht passieren. Es sind doppelt so viele grüne Pixel vorhanden, da das menschliche Auge für grünes Licht besonders empfindlich ist. (Quelle: Amada44/wikipedia unter Public Domain)

Bei CMOS-Sensoren ist es hingegen möglich, die Pixel in beliebiger Reihenfolge auszulesen. Die einzelnen Technologien haben verschiedene Vor- und Nachteile hinsichtlich ihrer möglichen Bildwiederholrate, ihrer Bildqualität und ihres Verhaltens bei Überbelichtung, die in oben genannter Referenz genauer betrachtet werden.

Die Sensoren an sich sind üblicherweise für Licht vom Infrarot- bis zum UV-Bereich sensibel. Ohne weitere Vorkehrungen können mit einem solchen Sensor nur Grauwertbilder aufgenommen werden, da der Sensor gleichermaßen auf die verschiedenen Wellenlängen reagiert. Digitale Farbkameras haben daher über jedem Pixel einen Farbfilter, der nur Licht einer bestimmten Wellenlänge passieren lässt. Ein häufig verwendeter Farbfilter weist eine sogenannte Bayer-Matrix auf (siehe Abbildung 2.3). Die Farbaufnahme wird dabei in der Form durchgeführt, dass sie für das menschliche Auge später eine möglichst originalgetreue Wiedergabe ermöglicht. Das menschliche Auge besitzt ebenfalls Sinneszellen für rotes, grünes und blaues Licht. Die Wahrnehmung der anderen Farben beruht auf einer Wahrnehmung der drei Farbkomponenten mit jeweils unterschiedlicher Intensität. Für das Bild der Kamera werden nun für jeden Bildpunkt ein Farbwert für rot, grün und blau basierend auf den Werten der umliegenden Pixel berechnet.

Digitale Verarbeitung

Unter dem Begriff digitale Verarbeitung werden hier die Schritte zusammengefasst, die bei der Umwandlung der Sensor-Rohdaten in das Ausgabeformat erfolgen. Je nach Kameramodell sind bestimmte Umwandlungsschritte vorgesehen und parametrierbar. Einige sollen hier betrachtet werden.

Umwandlung der Quantisierung Intern arbeiten viele Kamerasysteme mit einer Darstellung der Pixelhelligkeit, die größer ist als das übliche Ausgabeformat von 8 bit pro Farbkanal. Die Verbreitung dieses Ausgabeformats liegt darin begründet, dass gängige Anzeigegeräte (Monitore) mit diesem Bereich angesteuert werden. Bietet eine Kamera eine größere interne Auflösung von beispielsweise 10, 12 oder 14 bit, kann entweder eine Umrechnung auf einen 8 bit-Wert erfolgen oder - wenn dies von den weiteren Verarbeitungsschritten unterstützt wird - mit den hoch aufgelösten Werten gearbeitet werden. Eine Umrechnung erfolgt im einfachsten Fall durch ein Verwerfen der niederwertigen Bits. Hierbei gehen prinzipiell Bildinformationen verloren. Es ist jedoch oftmals der Fall, dass das vorhandene Bildrauschen deutlich größer ist als das hierdurch erhöhte Quantisierungsrauschen.

Denkbar ist auch eine nichtlineare Abbildung der Sensordaten auf die Ausgabedaten mit dem Ziel, den Verlust an Bildinformationen zu minimieren, indem möglichst eine Gleichverteilung der Bildhelligkeiten im Ausgabebild vorliegt. Je nach Kameramodell kann eine parametrierbare Umrechnungsstrategie vorhanden sein. Diese könnte in Form einer Funktion oder Lookup-Tabelle vorliegen. Alternativ bleibt die Ausgabe als Rohdaten und die Umrechnung in Software, was jedoch entsprechenden Rechenaufwand erfordert.

Ebenso möglich sind nichtsurjektive High Dynamic Range (HDR) Abbildungen, die hauptsächlich das Ziel verfolgen, für den Menschen ein Bild mit hohem Dynamikumfang mit möglichst originalgetreuem Bildeindruck auf einem nicht HDR-fähigen Medium (Monitor, Ausdruck) zu bieten. Genauere Beschreibung dieser Algorithmen sind in [RWPD05] zu finden.

Umwandlung des Farbraumes Abhängig vom Kameramodell ist aufgrund der Pixelanordnung in der Kamera eine weitere Umwandlung notwendig. Bei Farbkameras ist meist, wie zuvor erläutert, jeder Bildpunkt für nur eine Farbe zuständig. In diesem Fall muss die Berechnung in das RGB-Format, in das YUV-Format[5] oder in ein Grauwertbild über eine Interpolation erfolgen. Für die Interpolation der Bayer-Matrix gibt es mehrere Möglichkeiten. In [PK07] sind eine Reihe von Interpolationsalgorithmen evaluiert. Dort ist dargelegt, dass abhängig von der Beleuchtungsstärke unterschiedliche Methoden zum optimalen Ergebnis führen. Da die Umwandlung in der Regel in der Kamera hardwarebasiert durchgeführt wird, bleibt alternativ nur die Möglichkeit einer rechenintensiven Software-Umrechnung der Rohdaten.

[5] Ein Bild im YUV-Format unterscheidet sich von einem Bild im RGB-Format durch die Repräsentation. Während bei einem RGB-Bild pro Pixel die rote, grüne und blaue Komponente gespeichert wird, erfolgt die Speicherung bei YUV in einer Komponente für die Helligkeit (Y) und zwei Komponenten für den Farbwert (UV). Oft werden die Komponenten für den Farbwert mit verringerter (halbierter) Auflösung gespeichert, so dass sich mehrere Pixel die Farbwertinformationen teilen. Hieraus resultiert ein verringerter Speicherbedarf.

Weißabgleich

Die Lichtquellen, die eine Szene beleuchten, können deutliche Unterschiede in der Farbtemperatur aufweisen. Somit werden in der Bildaufnahme auch die Gegenstände mit „Farbstichen“ aufgenommen. Bei Bildverarbeitungsalgorithmen, die die Farbinformation eines Bildes benutzen, können sich dadurch Fehler ergeben. Unter Weißabgleich versteht man die nachträgliche digitale Korrektur der Einflüsse des Umgebungslichts. Sie erfolgt in der Regel bei der Farbraumkonvertierung in Hardware. Das Problem beim Weißabgleich ist die Bestimmung der Umgebungslichttemperatur, die aus den Bilddaten nicht sicher ermittelt werden kann. Gebräuchliche Verfahren bestimmen den hellsten Punkt eines Bildes, interpretieren diesen als weiß und berechnen Korrekturfaktoren, die dann auf alle Bildpunkte angewendet werden. In [GG07] ist ein Vergleich von mehreren Verfahren zum nachträglichen Weißabgleich aufgezeigt.

2.2.3. Mathematische Betrachtung einer Kamera

Mathematisch betrachtet handelt es sich bei einer Bildaufnahme um die Projektion eines Punktes im dreidimensionalen Raum auf eine zweidimensionale Bildebene. Diese Projektion soll basierend auf den Beschreibungen in [ope13] wiedergegeben werden. Sie wird durch folgende Gleichung beschrieben:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Hierbei ist $[X, Y, Z]^T$ ein Punkt im einem festgelegten Bezugskoordinatensystem und u und v sind die Pixelkoordinaten der Abbildung dieses Punktes auf der zweidimensionalen Bildebene. Die Parameter $f_x, f_y, c_x,$ und c_y hängen vom Kamerasystem ab und werden als intrinsische Kameraparameter bezeichnet. Durch diese Parameter wird die Brennweite in Pixelbreite (f_x) und Pixelhöhe (f_y) sowie die Lage des Bildmittelpunktes $c_x$und c_y beschrieben. Die übrigen Parameter r_{11} bis r_{33} und t_x bis t_z beschreiben die Position und Orientierung des Kamerasystems im Raum und werden extrinsische Kameraparameter genannt.

Bei der Kamerakalibrierung werden die intrinsischen und extrinsischen Parameter einer Kamera bestimmt. Dies erfolgt über eine ausreichende Anzahl Kalibrierungspunkte mit bekannten Koordinaten und deren (detektierte) Pixelkoordinaten im Kamerabild.

In der Praxis treten bei Abbildungen mit einem Kamerasystem Störeffekte auf, die bewirken, dass die tatsächlichen Pixelkoordinaten von den berechneten Pixelkoordinaten abweichen. Diese Störeffekte lassen sich jedoch mathematisch gut modellieren und korrigieren.

Zunächst wird oben beschriebene Gleichung umgeformt zu:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

wobei R die Rotationsmatrix (r_{11} bis r_{33}) und t der Translationsvektor (t_x bis t_z) ist. Des Weiteren sind:

$$\begin{aligned} x' &= x/z \\ y' &= y/z \\ u &= f_x \cdot x' + c_x \\ v &= f_y \cdot y' + c_y \end{aligned}$$

Unter Einbeziehung verschiedener Verzeichnungsparameter wird das Modell erweitert zu:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

mit

$$\begin{aligned} x' &= x/z \\ y' &= y/z \\ x'' &= x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\ y'' &= y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \\ r &= x'^2 + y'^2 \\ u &= f_x \cdot x'' + c_x \\ v &= f_y \cdot y'' + c_y \end{aligned}$$

Hierbei sind k_1, k_2und k_3 die radialen Verzeichnungskoeffizienten und $p_1$und p_2 die tangentialen Verzeichnungskoeffizienten. Bei einem Kalibrierungsvorgang werden diese Parameter meist direkt mitbestimmt. Es existieren verschiedene Ansätze zur Kamerakalibrierung. Das im Rahmen dieser Arbeit benutzte Verfahren nach Tsai benutzt ein Schachbrettmuster zur Kalibrierung, das in mehreren Positionen aufgenommen werden muss [Tsa87].

Die oben beschriebenen Gleichungen spielen nicht nur bei der Kamerakalibrierung, sondern auch bei der Objektdetektion eine wichtige Rolle. Wenn die intrinsischen Parameter und die Parameter zur Verzeichnung bekannt sind sowie ausreichend (mindestens vier) Punkte eines Objektes im Kamerabild detektiert werden, kann durch Lösung der Abbildungsgleichung die dreidimensionale Position des Objektes zum Kamerasystem berechnet werden. Hierfür ist ebenfalls Voraussetzung, dass die Position der entsprechenden Punkte des Objektes in Bezug auf ein definierbares Koordinatensystem bekannt ist.

2.2.4. Parameter einer Bildaufnahme

Bei der Aufnahme einer Szene spielen viele Parameter eine Rolle. In Abbildung 2.4 ist eine mögliche Einteilung aufgezeigt, die für den Fall des Einsatzes auf einem mobilen Robotersystem sinnvoll erscheint.

Die Steuerungsmöglichkeiten der einzelnen Parameter sollen im Folgenden abgewogen werden. Die Einflüsse der Parameter auf weitere Parameter werden ebenfalls erläutert. Dabei ist die Reihenfolge so gewählt, dass die Parameter, die die meisten Auswirkungen auf weitere Parameter haben, zuerst erwähnt werden.

Kameraposition

Einen entscheidenden Faktor für die Bildaufnahme stellt die Position des Kamerasystems dar. Diese wird maßgeblich durch die Position des Robotersystems sowie durch Vorrichtungen auf dem Robotersystem zum Bewegen der Kameras beeinflusst. Die aktive Steuerung der Roboterplattform kann zum einen darin begründet liegen, einen Bildbereich genauer oder aus einer anderen Perspektive zu betrachten (wie beispielsweise in [KRZ12] beschrieben), zum anderen darin, weitere Gebiete zu erkunden.

Über eine so genannte Pan-Tilt-Unit (PTU)[6] wird eine Kamera in verschiedene Richtungen geschwenkt. So können verschiedene Bereiche des Raumes abgebildet werden, ohne das Robotersystem zu bewegen. Ziel der Benutzung der PTU kann darüber hinaus sein, ein Objekt in den Bildmittelpunkt zu bringen, um anschließend einen Zoomvorgang auf das Objekt durchzuführen. Ebenso kann mit der PTU das Kamerasystem einem sich bewegenden Objekt nachgeführt werden oder die Bewegung des Robotersystems kompensieren.

Es ist manchmal notwendig, dass andere Einstellungen im Zuge der Bewegung des Kamerasystems verändert werden müssen.

Zoom: Bezüglich des Zooms ist zu unterscheiden, ob das System gerade einen Gegenstand genauer betrachtet oder ob durch die Bewegung der PTU ein komplett anderer Bildbereich abgebildet werden soll. Ist Letzteres der Fall, sollte in Weitwinkelstellung zurückgegangen werden, damit ein Überblick der Gesamtszene ermöglicht wird. Beim Heranzoomen kann auch in mehreren Stufen vorgegangen werden, damit das Ziel nicht durch Ungenauigkeit außerhalb des Sichtbereichs gerät.

Verschlusszeit/Blende/Sensorempfindlichkeit[7]: Um Bewegungsunschärfe zu verringern, kann unter Inkaufnahme von erhöhtem Bildrauschen versucht werden, die Verschlusszeit zu reduzieren. Hierzu wird die Empfindlichkeit des Sensors (Gain) heraufgesetzt oder alternativ die Blende vergrößert, was zu geringerer Schärfentiefe führt. Auf diese Funktion kann jedoch verzichtet werden, falls lediglich Interesse an der Aufnahme unter der Zielposition der PTU besteht. Durch die sich ändernde Umgebung können Anpassungen der gesamten Bildhelligkeit notwendig werden.

[6] Schwenk-Neige-Einheit

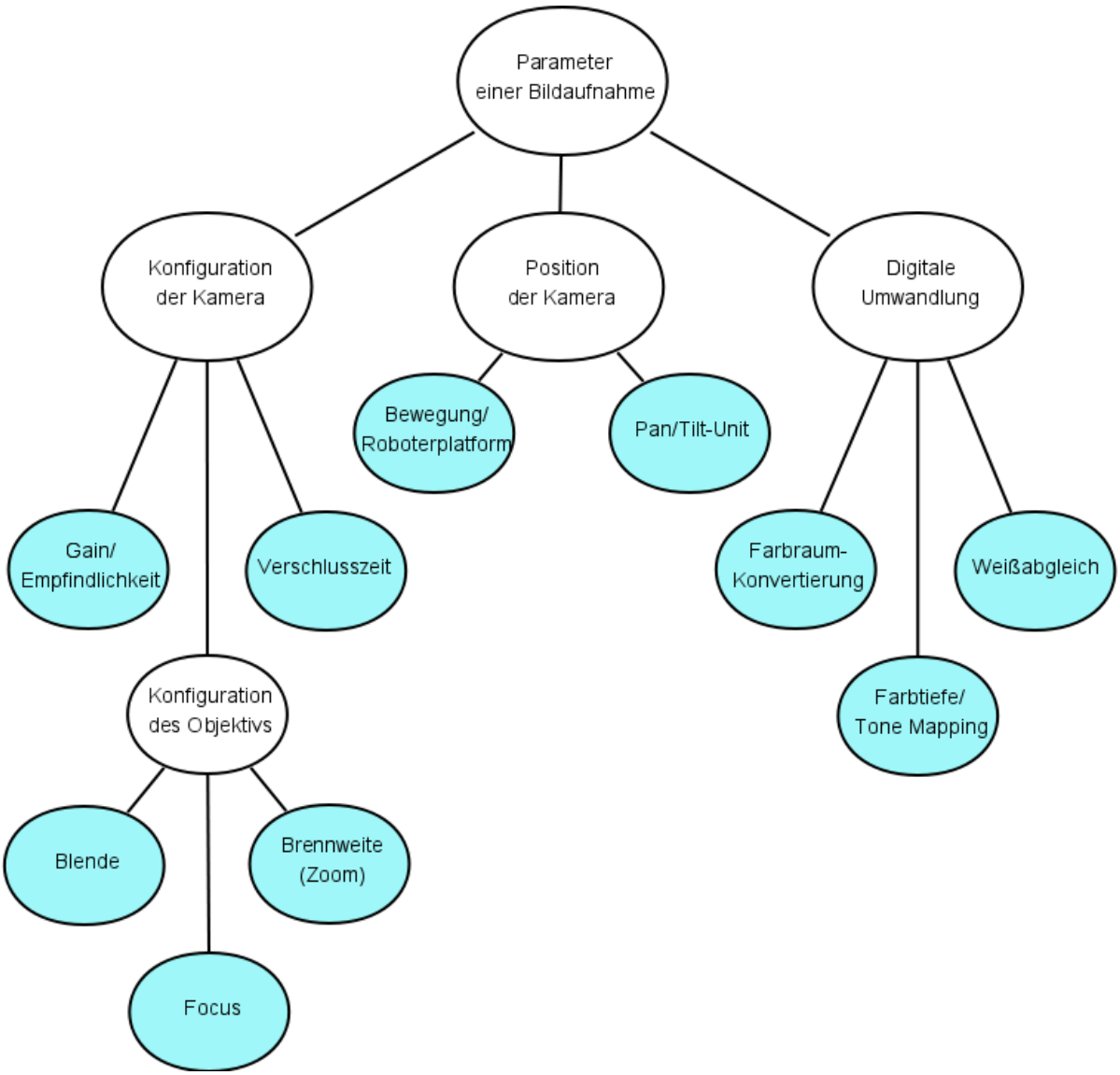


Abbildung 2.4.: Parameter einer Bildaufnahme: Dieses Abbildung zeigt eine mögliche Einteilung der Parameter, die neben der Szene einen Einfluss auf die Bildaufnahme haben

Weißabgleich: In einigen Fällen könnte ein erneuter Weißabgleich sinnvoll sein.

Zoom

Zoom bedeutet die Variation der Brennweite der Linse. Dadurch kann die Größe des Bildwinkels verändert werden. So können mit einem Objektiv sowohl Weitwinkelaufnahmen für einen Überblick über die Szene als auch Detailaufnahmen von entfernten Gegenständen erfolgen. Dadurch, dass diese Gegenstände formatfüllend abgebildet werden, können Bildverarbeitungsalgorithmen das Objekt mit höherer Genauigkeit klassifizieren.

Blende Gemäß der Definition der Blendenzahl ändert eine Betätigung des Zooms auch die effektive Blendenzahl. Dieser Effekt sollte also wenn möglich seitens der Blendeneinstellung kompensiert werden.

Fokus Bei einer Variation des Zooms ist bei einigen Objektiven eine Nachfokussierung erforderlich. Ist der Zusammenhang zwischen Zoom- und Fokuswerten bekannt, können Korrekturen der Einstellung direkt erfolgen, ohne dass eine komplette Neufokussierung durchgeführt werden muss.

Verschlusszeit/Sensorempfindlichkeit: Sollte die Beibehaltung der Blendenstufe nicht möglich sein oder der Bildausschnitt, an den herangezoomt wird, eine andere Helligkeit aufweisen als die bisherige Szene, lässt sich die Belichtung durch Anpassungen der Verschlusszeit und Sensor-Empfindlichkeit korrigieren.

Fokussierung

Bei der Fokussierung kann über eine Verschiebung der Linse(n) das Kamerabild auf Objekte scharfgestellt werden, die einen bestimmten Abstand zur Kamera haben. Objekte mit anderem Abstand werden unscharf abgebildet. Dieser Effekt ist bei offener Blende deutlich stärker ausgeprägt als bei kleiner Blende. Da in der Regel nicht alle Bilddetails scharf abgebildet werden können, muss eine Auswahl getroffen werden, die entweder einen bestimmten Bereich bevorzugt oder einen Mittelwert aus mehreren Bereichen ermittelt.

Blende/Verschlusszeit/Sensorempfindlichkeit: Direkte Auswirkungen auf die weiteren Parameter bestehen nicht. Wenn Objekte mit unterschiedlichen Distanzen scharf abgebildet werden sollen, lässt sich dieses durch eine Verkleinerung der Blende erreichen. Die verminderte Lichtstärke muss durch die Sensorempfindlichkeit und die Verschlusszeit kompensiert werden.

Blende/Verschlusszeit/Sensorempfindlichkeit

Diese drei Parameter beeinflussen allesamt die Bildhelligkeit; aus diesem Grunde werden sie gemeinsam betrachtet. Generell sollte versucht werden, den gesamten Wertebereich

von minimaler bis maximaler Helligkeit auszunutzen, da so die meisten Bildinformationen gewonnen werden können. Eine nachträgliche digitale Veränderung der Bildhelligkeit führt zu einer verschlechterten Bildqualität aufgrund der Quantisierung der Pixelwerte und verstärkt zudem vorhandenes Rauschen.

Während die Blende und die Verschlusszeit die Lichtmenge regeln, die auf den Sensor einfällt, wird über den Gain-Wert die Empfindlichkeit des Bildwandlers eingestellt. Jede Kombination stellt einen Kompromiss aus Schärfentiefe, Bewegungsunschärfe und Bildrauschen dar. Bei kleiner Blende kann in einem großen Entfernungsbereich scharf fokussiert werden, jedoch ist die Lichtmenge gering. Wird die Blende geöffnet, kann zunehmend nur auf einen kleinen Bildbereich fokussiert werden, was für Aufnahmen ungünstig ist, bei denen ein Überblick über die Szene gewünscht ist. Die Verschlusszeit regelt, wie lange der Sensor der Beleuchtung ausgesetzt wird. Mit der Belichtungszeit steigt die Bildhelligkeit, jedoch erhöhen sich die negativen Auswirkungen sowohl von Objektbewegungen als auch von der Eigenbewegung des Kamerasystems. Es tritt zunehmend Bewegungsunschärfe auf. Über den Gain-Faktor lässt sich die Ansteuerung des Sensors so verändern, dass die Empfindlichkeit zunimmt. Ein kleinerer Helligkeitsbereich im Bild wird auf den gesamten digitalen Ausgangswertebereich abgebildet, ohne dass wie bei der digitalen Erhöhung der Helligkeit die prozentual hohen Quantisierungsfehler von dunklen Bildpunkten mitskaliert werden. Physikalisch bedingt führt dieses Verfahren jedoch auch zu einem umso höheren Bildrauschen, je empfindlicher der Sensor geschaltet wird.

2.2.5. Spezielle Kamerasysteme

Viele Kamerasysteme orientieren sich in Bezug auf ihre Abbildungscharakteristik an dem menschlichen Auge. Das Bild wird so aufgenommen, dass es für einen menschlichen Betrachter originalgetreu erscheint. Für viele Anwendungsbereiche werden auch Kameras mit speziellen Fähigkeiten entwickelt, die zusätzliche Informationen bereitstellen. Diese Kameras sollen im Folgenden kurz aufgeführt werden.

Wärmebildkameras

Anstatt der Helligkeit eines Bildpunktes messen Wärmebildkameras die Temperatur des Objektes. Klassische Kameras detektieren Licht in dem für Menschen sichtbaren Bereich, also elektromagnetische Strahlung mit einer Wellenlänge von etwa 400-780 nm. Langwelligere Strahlung fällt in den so genannten Infrarot-Bereich. Abhängig von der Temperatur geben alle Körper gemäß dem Kirchhoffschen Strahlungsgesetz hauptsächlich Infrarotstrahlung ab. Mit steigender Temperatur erhöht sich die Energie der abgegebenen Strahlung und es verringert sich die minimale Wellenlänge. Die abgegebene Strahlung kann auch im sichtbaren Bereich liegen (glühende Gegenstände).

Eine Wärmebildkamera detektiert die von einem Objekt emittierte Infrarotstrahlung und bestimmt so die Temperatur des Objektes. Dieses erfolgt durch einen speziellen Sensortyp.

Über eine Abbildungsfunktion kann aus den Messwerten wieder ein für Menschen sichtbares Bild werden, das beispielsweise die verschiedenen Temperaturen durch verschiedene Farben visualisiert. Auch im Bereich der Robotik ergeben sich vielfältige Einsatzmöglichkeiten, da sich über diesen Kameratyp zusätzliche Informationen über die Umgebung gewinnen lassen. Es können beispielsweise Lebewesen oder andere Objekte von besonderem Interesse detektiert werden.

3D-Kamerasysteme

Das Ziel von 3D-Kamerasystemen ist die Bereitstellung von Tiefeninformationen für jeden Bildpunkt und abhängig vom Typ zusätzlich Helligkeits- und Farbinformationen. Es existieren verschiedene Ansätze, um die Tiefeninformationen zu generieren. Hier ist zwischen passiven Systemen und aktiven Kamerasensoren zu unterscheiden. Bei den aktiven Sensoren wird die Szene mit einer speziellen Lichtquelle beleuchtet.

Bei **Stereo-Kamerasystemen** handelt es sich um passive Systeme, die durch Einsatz von zwei oder mehreren Kamerasystemen dreidimensionale Bildinformationen erzeugen. Über das gleiche Prinzip gewinnen auch Menschen und viele andere Lebewesen einen dreidimensionalen Bildeindruck. Wird ein Bildpunkt von mehreren Kameras erfasst, kann über die Winkel, aus denen dieser Bildpunkt von den jeweiligen Kameras betrachtet wird, die Lage des Punktes im Raum berechnet werden. Hierbei ist das Detektieren von korrespondierenden Punkten in den verschiedenen Kamerabildern die zentrale Herausforderung. Insbesondere bei einfarbigen Flächen ist dies oft nicht möglich. Weitere Voraussetzungen sind, dass die Kamerasysteme synchron belichten und dass die Abbildungsverzeichnungen der Kameras bekannt sind.

Sogenannte **RGB+D** Kameras zählen zu den aktiven Sensoren, die die Szene mit strukturiertem Licht beleuchten. Die Strukturen werden von einem Projektor erzeugt, der oft im Infrarotbereich arbeitet. Über die Detektion der Strukturen im Kamerabild und den bekannten Abstand zwischen Projektor und Kamera kann die Distanz errechnet werden. Eine zusätzliche konventionelle Kamera liefert Helligkeits- und Farbwertinformationen, die dann mit den Tiefeninformationen zu einer sogenannten Punktewolke verrechnet werden. Der Vorteil dieses Prinzips ist die zuverlässige Funktion und die Möglichkeit, auch bei einfarbigen Oberflächen die Tiefeninformation zu bestimmen. Das Verfahren ist hingegen anfällig gegenüber starker Fremdlichteinstrahlung, so dass es zum jetzigen Stand der Technik nicht für den Außenbereich eingesetzt wird. Weitere Probleme ergeben sich bei durchsichtigen Objekten und spiegelnden Oberflächen.

Ein bekanntes Beispiel für diesen Kameratyp ist die Microsoft Kinect [Zha12]. Diese RGB+D Kamera ist als Spielesteuerung für die Konsole XBox360 entwickelt. Aufgrund der USB-Schnittstelle und des im Vergleich zu anderen 3D-Kamerasystemen niedrigen Preises entwickelte sich die Kinect zu einem beliebten Sensor in der Robotik. Basierend auf Daten dieser Kameras kann eine Segmentierung einer Tischszene und eine Vergleich

mit bekannten Modellen von Objekten erfolgen. Viele Algorithmen sind in der Point-Cloud-Library [RC11] implementiert.

Ein weiteres Verfahren zur Erzeugung von Tiefeninformationen findet in den **Time-of-Flight-Kameras** Anwendung [KBK08]. Auch diese Kameras zählen zu den aktiven Sensoren. Hierbei wird die Szene von einer gepulsten Lichtquelle beleuchtet. Es wird dann vom Bildsensor die Laufzeit der Lichtpulse ausgewertet und über die bekannte Lichtgeschwindigkeit die Entfernung des Objektes bestimmt. Die verwendeten Sensoren unterscheiden sich von denen traditioneller Kameras. Die Vor- und Nachteile dieses Systems ähneln denen der RGB+D Kameras.

Hochgeschwindigkeitskameras

Ein weiterer Spezialtyp von Kamerasystemen sind die Hochgeschwindigkeitskameras. Während digitale Videokameras die Szene gewöhnlicherweise mit ca. 60 Bildern pro Sekunde aufzeichnen, erreichen Hochgeschwindigkeitskameras deutlich höhere Wiederholraten. Dieses erfolgt durch eine kürzere Belichtungszeit und Sensoren, die ein schnelles Auslesen unterstützen. Aufgrund der geringeren Belichtungszeit und der damit einhergehenden geringen Lichtmenge ist oft eine aktive Beleuchtung notwendig. Mit diesen Kameras können Vorgänge in extremer Zeitlupe dargestellt werden. Im Bereich der Robotik können bei entsprechender Echtzeitverarbeitung schnelle Reaktionszeiten erzielt werden.

Ein Forschungsprojekt, das dreidimensionale Bildinformationen mit strukturiertem Licht gewinnt, ist in [IYDT07] beschrieben. Hier kommt die Hochgeschwindigkeitskamera FASTCAM-1024PCI von Photron Ltd. zum Einsatz. Diese Kamera zeichnet sich durch eine Auflösung von 1024 x 1024 Pixeln bei einer Bildwiederholrate von 1000 Bildern pro Sekunde aus. Die Bilddaten werden in einem dedizierten Speicher zwischengespeichert, die eigentliche Verarbeitung erfolgt „offline“. Ein Hochgeschwindigkeitsprojektor beleuchtet die Szene mit strukturiertem Licht. Bewegungen einer menschlichen Hand können auf diese Art aufgenommen werden. Dieses Forschungsthema verdeutlicht die hohen Anforderungen der maschinellen Sichtsysteme bezüglich Rechenleistung und Datenvolumina, da hier die Auswertung nicht in Echtzeit erfolgen kann.

2.3. Intelligente Kamerasysteme

Intelligente Kamerasysteme (engl. Smart Cameras) zeichnen sich dadurch aus, dass sie neben der Bildaufnahme auch eine Interpretation des Bildes durchführen können. Einen Überblick über das Thema intelligente Kamerasysteme gibt [Bel09]. Insbesondere wird dort die Entwicklung der intelligenten Kamerasysteme betrachtet. Eine klare Abgrenzung zwischen intelligenten und nicht-intelligenten Kamerasystemen kann nicht allgemeingültig festgelegt werden. Im Jahre 1975 beispielsweise wurde der Begriff „Smart Camera“

für Kameras benutzt, die automatisch die Verschlusszeit steuern und somit an die Szene anpassen konnten [Sch75]. Somit wäre heute praktisch jede Kamera eine intelligente Kamera. Auch eine Definition über das Vorhandensein von Recheneinheiten würde dazu führen, dass nahezu alle digitalen Kameras in diese Kategorie fallen.

Für diese Arbeit soll der Begriff „Intelligente Kamera“ in Anlehnung an [Bel09] so definiert werden, dass die Kamera in der Lage ist, Informationen aus den Bilddaten zu extrahieren und auszugeben. So kann eine Abgrenzung geschaffen werden zu marktüblichen digitalen Kameras, die zwar prinzipiell über Funktionen zum Interpretieren der Szene verfügen (z.B. Gesichtslokalisation, Mimik-Erkennung), diese Informationen jedoch nur zur Bildaufnahme und Bildverbesserung verwenden.

In den folgenden Abschnitten wird auf viele Aspekte von intelligenten Kameras detailliert eingegangen. Ein entscheidender Teil des Forschungsstandes auf diesem Themengebiet zu Beginn des Dissertationsvorhabens wird in [BCJK05] dargelegt. Themen dieser Veröffentlichung sind die Architektur von intelligenten Kameras, die verwendeten Verarbeitungseinheiten sowie ein Modell zur Softwareentwicklung.

Die Autoren sehen ein sich immer weiter verbreitendes Anwendungsfeld für Sichtsysteme mit eingebetteten Verarbeitungseinheiten, beispielsweise in der industriellen Bildverarbeitung, Robotik, Verkehrsüberwachung oder medizinischen Bildverarbeitung. Die Verarbeitungseinheiten der Systeme müssen oft große Datenmengen verarbeiten und es gelten vielfach strikte Echtzeitanforderungen. Wegen ihrer Relevanz für die vorliegende Arbeit werden die dort dargelegten Ergebnisse an späterer Stelle noch ausführlicher betrachtet.

2.3.1. Kategorisierung intelligenter Kamerasysteme

Intelligente Kameras können anhand verschiedener Aspekte kategorisiert werden. Die Recheneinheiten können beispielsweise direkt auf dem Bildsensor, im Kameragehäuse oder auch als externes dediziertes System ausgelegt sein (vgl. [Bel09, MBZ08]). Eine intelligente Kamera kann als eingebettetes System fungieren (z.B. wie in dieser Arbeit als Teil eines Roboters) oder auch Bestandteil eines Netzwerkes aus mehreren Kameras sein [BDM+06]. Ein weiteres Kriterium ist die Architektur der Recheneinheiten (z.B. FPGA, DSP oder CPU).

In Tabelle 2.5 wird die Aufteilung nach [Bel09] wiedergegeben. Zu dieser Aufteilung soll hier angemerkt werden, dass diese in bestimmten Fällen nicht disjunktiv ist, da der Aspekt der Einbettung in ein anderes System noch keine Aussage über die Realisierung der Recheneinheiten trifft. In den später folgenden Beispielen für intelligente Kameras wird jeweils auch die Einteilung in die hier aufgeführten Klassen diskutiert. Auch Kamerasysteme, deren Recheneinheiten nicht in das Kameragehäuse, sondern in kompakte externe Gehäuse integriert sind, werden von [Bel09] als intelligente Kameras betrachtet. Ausschlaggebend ist hierbei, dass die Rechenleistung dediziert für die Bildverarbeitung zur Verfügung steht.

Typ	Beschreibung	Anwendung
Ein-Chip intelligente Kameras	Die Funktionseinheiten befinden sich direkt auf dem Sensorchip	Spielzeug
Eingebettete intelligente Kameras	Die Kamera ist Teil eines anderen Systems	optische Computermäuse, Fingerabdruck-Lesegeräte, Smartphone-Kameras
Eigenständige intelligente Kameras	Die Verarbeitungseinheiten befinden sich im Kameragehäuse	maschinelle Bildverarbeitung
Intelligente Kamerasysteme mit externer Verarbeitung	Die (dedizierten) Verarbeitungseinheiten befinden sich in einem kompakten System in der Nähe der Kamera	Überwachung, Verkehrsüberwachung, maschinelle Bildverarbeitung
Verteilte intelligente Kameras	Die Auswertung der Daten erfolgt im Netzwerk der intelligenten Kameras	Intelligente Videoüberwachung, maschinelle Sichtsysteme

Abbildung 2.5.: Klassifikation intelligenter Kamerasysteme, sinngemäß nach [Bel09]

Leistungsklassen intelligenter Kameras

Intelligente Kamerasysteme können anhand ihrer Leistungsfähigkeit klassifiziert werden. Hierbei kann wie bei Prozessoren üblich die Anzahl der Operationen pro Sekunde betrachtet werden, wobei diese Angabe nicht proportional zur tatsächlichen Performance sein muss. In [BCJK05] wird die Leistungsfähigkeit zudem auf den Wert Instruktionen pro Pixel umgerechnet. Hierbei spielen neben der Rechenleistung des verwendeten Prozessors auch die Bildauflösung und die Bildwiederholrate eine Rolle. Durch diese Angabe kann direkt die Komplexität der ausführbaren Bildverarbeitungsalgorithmen abgeschätzt werden.

Die notwendige Rechenleistung hängt auch stark von der Anwendung ab. Weitere Rahmenbedingungen wie Leistungsaufnahme, Kosten und Gehäusegröße können ebenfalls die realisierbare Rechenleistung beeinflussen. Es werden verschiedene Klassen von Bildverarbeitungssystemen betrachtet, die in Tabelle 2.6 wiedergegeben werden. Bei den Werten für die Rechenleistung ist zu berücksichtigen, dass diese Zahlen aus dem Jahr 2005 stammen.

2.3.2. Parallelisierung auf intelligenten Kamerasystemen

Bei der performanten Verarbeitung der Bilddaten spielt die Parallelisierbarkeit der Bildverarbeitungsaufgaben eine Rolle. In [BCJK05] wird auf verschiedene Formen der Parallelisierbarkeit eingegangen, die nachfolgend sinngemäß wiedergegeben und erläutert werden.

Anwendungsfeld	Ips	IpP	Beschreibung
Mobile Multimedia-Datenverarbeitung	300 MOPs bis 1,5 GOPs	bis 150	Diese Anwendungsklasse beinhaltet einfachere Bildverarbeitungsaufgaben wie Weißabgleich oder Belichtungskontrolle. Aufgrund der Verwendung in mobilen Geräten darf der Stromverbrauch nur gering sein. Es bestehen zudem starke Beschränkungen der Kosten.
Intelligente Heim-Roboter	>3 GOPs	>300	Der Funktionsumfang entsprechender Kamerasysteme umfasst Gesichtslokalisation, Gestenerkennung und intelligente Überwachung. Es bestehen mittlere Beschränkungen hinsichtlich der Kosten der Systeme. Hauptkriterium bei diesen Systemen ist die Zuverlässigkeit, auch unter dynamischen Umgebungsbedingungen. Daher bestehen höhere Anforderungen an die Rechenleistung. Die Auslastung kann zur Laufzeit in Abhängigkeit von der Betriebssituation schwanken.
Industrielle Bildverarbeitung	variabel bis >4 GOPs	variabel	Bei der industriellen Bildverarbeitung bestehen kaum Restriktionen hinsichtlich der Kosten. In bestimmten Fällen können hinsichtlich der Verlustleitung Beschränkungen bestehen. Typische Operationen sind Bildoptimierung und die Detektion von Kanten und Formen. Oft sind hohe Bildwiederholraten vorgegeben, sodass bei ebenfalls vorgegebener Anzahl von Operationen pro Pixel eine hohe Gesamtrechenleistung bis über 4 GOPs notwendig ist.

Abbildung 2.6.: Die verschiedenen Leistungsklassen intelligenter Kameras nach [BCJK05]

Die **Parallelität auf Datenebene** besagt, dass die Verarbeitung der Daten zeitgleich auf mehreren Funktionseinheiten möglich ist. Diese Form der Parallelität wird auch als „Within-Operation Parallelism“ bezeichnet. Dieser Ausdruck besagt, dass eine zusammengehörige Aktion parallelisierbare Arbeitsschritte aufweist. Wird beispielsweise auf jedem Bildpunkt die gleiche Operation angewendet, die nicht von den Ergebnissen anderer in diesem Schritt ausgeführten Operationen abhängt, so können die Operationen parallel ausgeführt werden. Bei der Implementierung kann zudem ausgenutzt werden, dass Operationen auf einem Bildpunkt als Eingangswerte oft nur eine lokal begrenzte Nachbarschaft dieses Bildpunktes benötigen. So ist es möglich, auf einer Funktionseinheit das Bild nicht als Ganzes, sondern zeilen- oder sogar pixelweise zu verarbeiten. Abhängig von der Größe der Nachbarschaft werden üblicherweise auch mehrere Zeilen verarbeitet. Durch die Verarbeitung von Bildabschnitten wird die Nutzung von Verarbeitungseinheiten (z.B. LPA, FPGA) ermöglicht, die eventuell nicht ausreichend Arbeitsspeicher besitzen, um das gesamte Bild in diesem zwischenzuspeichern. Ein weiterer entscheidender Vorteil hierbei ist, dass der Datentransfer zur Verarbeitungseinheit (bzw. von der Verarbeitungseinheit zur nächsten Verarbeitungseinheit) parallel zur eigentlichen Datenverarbeitung erfolgt. Weist der nachfolgende Verarbeitungsschritt ebenfalls den Aspekt einer lokal begrenzten Nachbarschaft auf, kann sogar bei entsprechender Programmierung der nachfolgende Verarbeitungsschritt für den bereits verarbeiteten Teil des Bildes begonnen werden, auch wenn der vorherige Arbeitsschritt noch nicht abgeschlossen ist. Die durch die Lokalität der Daten bedingte Parallelität lässt sich durch Benutzung von SIMD-Systemen ausnutzen. Auf SIMD-Systemen (Single Instruction, Multiple Data) wenden mehrere Funktionseinheiten zeitgleich die gleiche Instruktion auf verschiedenen Teilen der Daten an.

Eine andere Form der Parallelität auf Datenebene stellt die Objekt-basierte Parallelität dar. Diese liegt dann vor, wenn in einem Verarbeitungsschritt bestimmte Elemente eines Bildes ausgewählt werden(z.B. signifikante Bildpunkte, zusammengehörige Bildbereiche), die im folgenden Verarbeitungsschritt unabhängig voneinander näher bestimmt werden. Es könnte nun die Weiterverarbeitung parallel auf verschiedenen Funktionseinheiten erfolgen. Diese Form der Parallelität ist in der Regel komplexer zu handhaben, da nicht alle Bildpunkte nach gleichem Muster verarbeitet werden. Auch stellt diese Form der Parallelisierbarkeit abhängig von der Aufgabe höhere Anforderungen an den Speicherbedarf.

Unter **Parallelität auf Aufgabenebene** ist zu verstehen, dass dem System verschiedene Aufgaben gestellt sind, die zur gleichen Zeit unabhängig voneinander ausgeführt werden können. Diese Form der Parallelisierung wird auch als „Between-Operation Parallelism“ bezeichnet und kann sowohl bei der Hintereinanderausführung von einzelnen Teilaufgaben als auch bei der unabhängigen Ausführung von Algorithmen mit verschiedenen Zielen angewendet werden.

Bei der Hintereinanderausführung von verschiedenen Operationen auf den Bilddaten ergibt sich - ein einzelnes Bild betrachtet - noch keine Verringerung der Latenz. Es werden jedoch mehrere aufeinander folgende Bilder parallel verarbeitet und durchlaufen die einzelnen Verarbeitungseinheiten. Die hieraus resultierenden Vorteile der Parallelisierung liegen in der hohen Auslastung der Verarbeitungseinheiten und in einer höheren möglichen

Bildwiederholrate. Bei der Anwendung der Parallelisierung muss stets darauf geachtet werden, dass ausreichend Funktionseinheiten für die Teilschritte vorhanden und diese für den Zeitraum der geplanten Belegung auch verfügbar sind. Ähnlich dem Pipelining in einem Prozessor müssen Struktur-Konflikte vermieden werden. Dieses erfordert eine genaue Anpassung der Bildverarbeitungsstrategie an die vorhandene Hardware. Bereits bei der Entwicklung eines intelligenten Kamerasystems sollte im Rahmen des Hardware-Software Co-Designs im Hinblick auf die geplante Anwendung die parallelisierte Verarbeitung der Bilddaten berücksichtigt werden.

Die Ausführung von mehreren unabhängigen Algorithmen auf einem Bild kann erforderlich sein, wenn verschiedene Arten von Bildinformationen extrahiert werden sollen. Hierbei ist ebenfalls auf die Verfügbarkeit der einzelnen Verarbeitungseinheiten zu achten, da es ansonsten zu Verzögerungen kommt. Es können auch Mischformen der Hintereinanderausführung und der Ausführung unabhängiger Algorithmen existieren. So ist es beispielsweise möglich, dass Vorverarbeitungsfunktionen für mehrere nachfolgende unabhängige Verarbeitungsalgorithmen genutzt werden können. Zu bestimmten Punkten kann eine Interaktion zwischen verschiedenen Teilaufgaben erforderlich sein.

Eine **Parallelität auf Instruktionsebene** liegt dann vor, wenn sich bestimmte Befehle einer Sequenz von Instruktionen parallel ausführen lassen. Dies ist immer dann der Fall, wenn keine Datenabhängigkeiten zwischen den Befehlen bestehen. Der entsprechende Prozessor muss zudem ausreichend Funktionseinheiten für die parallele Verarbeitung aufweisen. Diese Parallelität lässt sich durch VLIW (Very Long Instruction Word)-Prozessoren ausnutzen. Bei diesen Prozessoren werden zur Kompilierzeit parallelisierbare Instruktionen bestimmt und in einem langen Befehl zusammengefasst (vgl. [HP11]). Zur Laufzeit wird dieser Befehl vom VLIW-Prozessor eingelesen und die enthaltenen Teilbefehle werden ausgeführt.

Aktuelle Prozessoren der RISC/CISC Architektur nutzen die Parallelität auf Instruktionsebene, indem zur Laufzeit die sequentiellen Instruktionen auf die verfügbaren Funktionseinheiten verteilt werden. Die Ausnutzung dieser Form der Parallelität erfolgt also automatisch und transparent für den Benutzer. Zusätzlich können vom Compiler Maßnahmen getroffen werden, um die Parallelisierung zu unterstützen.

Bei der Entwicklung einer intelligenten Kamera können mehrere verschiedene Verarbeitungseinheiten verwendet werden und somit mehrere der genannten Paradigmen der Parallelverarbeitung verwirklicht werden. Neben der allgemeinen Einschränkung, dass sich nicht alle Algorithmen für alle Formen der Parallelisierung eignen, bedarf es einer Anpassung der Anwendung an den jeweiligen Kameratyp.

2.3.3. Programmiermodelle intelligenter Kameras

Bei intelligenten Kamerasystemen, dessen Funktion sich an verschiedene Aufgabenstellungen anpassen lässt, spielt die Art der Programmierung eine entscheidende Rolle. Die

verschiedenen Arten unterscheiden sich in ihrer Komplexität, Flexibilität und in der Möglichkeit, bestimmte Optimierungen vorzunehmen.

Visuelle Programmierung Um intelligente Kameras auf die Aufgabenstellung zu konfigurieren, kommen oft visuelle Programmieroberflächen zum Einsatz. Hier kann der Nutzer in der Regel eine Datenstromorientierte Programmierung der Kamera durchführen. Die Datenstromorientierte Programmierung wird an späterer Stelle noch gesondert behandelt (Abschnitt 2.7). Es wird meist eine Bibliothek mit vordefinierten Funktionen bereitgestellt, die vom Benutzer dann verknüpft werden. Auf diese Art ist eine einfache Programmierung der Systeme auch ohne spezielle Programmierkenntnisse möglich.

Als Beispiel für visuelle Programmierung wird nachfolgend das Programmiermodell der intelligenten Kamera Matrox Iris Gt beschrieben. Die Beschreibung basiert auf den auf der Internetpräsenz[8] zur Verfügung gestellten Informationen. Die Programmierung innerhalb des *Matrox Design Assistant* erfolgt durch Erstellung eines Flussdiagramms (s. Abbildung A.1, Appendix A, Seite 236) auf einem externen System mit Microsoft Windows Betriebssystem. Aus einer Bibliothek, die Funktionen zur Bildanalyse, Bildverarbeitung, Kommunikation, Flusskontrolle sowie zur Ansteuerung der digitalen Schnittstellen der Kamera bereitstellt, werden bestimmte Elemente zu dem Flussdiagramm hinzugefügt und konfiguriert. Benutzerdefinierte Funktionen können durch Programmierung in C# hinzugefügt werden. Während der Entwicklung kann das Programm ohne Kamera anhand von Testdaten ausgeführt und hierbei die Konfiguration der einzelnen Parameter angepasst werden. Die entwickelten Programme werden dann auf die Kamera übertragen, wo sie dann unabhängig ausgeführt werden. Zu den Programmen können auch Web-Interfaces erstellt werden, die dann zur Ausführungszeit Informationen ausgeben oder ein Eingreifen in den Programmablauf erlauben.

Der Hersteller Native Instruments, der für seine grafische Programmierumgebung LabView bekannt ist, bietet für die hauseigenen intelligenten Kameras die Software „NI Vision Development Module“ an (vgl. [nis13]). Die Programmierung kann dann grafisch in LabView oder in einer klassischen Hochsprache unter Benutzung der Bibliothek erfolgen. Die Software enthält viele Funktionen der maschinellen Bildverarbeitung.

Ein weiteres System zur visuellen Programmierung ist die Software „Clicks Vision Application Builder“ für die - mittlerweile nicht mehr vertriebenen - intelligenten Kameras Inca311 und Legend von Philips[9] (vgl. [BCJK05]).

Programmierung in einer Hochsprache Eine Programmierumgebung zur Entwicklung von Anwendungen für intelligente Kameras und weitere Bildverarbeitungssysteme ist HALCON [hal13]. Die intelligente Kamera Basler eXcite, die im Rahmen dieser Arbeit genutzt wird, wird ebenfalls durch eine spezielle Version der Programmierumgebung

[8] www.matrox.com, Stand Februar 2013 [mat13]
[9] Die Geschäftssparte wurde mittlerweile zu NXP übertragen, einer Tochterfirma von Philips.

unterstützt. Neben der Basler eXcite werden auch viele aktuelle intelligente Kameras unterstützt, darunter die MATRIX VISION BlueLYNX-X, die Sony XCI-SX1/V3 und SX100/V100, die Photonfocus SM2-D1024-80 und die Vision Components VC Optimum. Auch kompakte dedizierte Systeme werden unterstützt, z.B. die IMAGO VisionBox oder auch das PandaBoard, wobei letzteres eine OMAP4-basierte Entwicklungsplatine mit unterschiedlichen Einsatzmöglichkeiten darstellt. Die HALCON-Programmierumgebung umfasst eine Entwicklungsumgebung sowie eine Bibliothek mit Bildverarbeitungsfunktionen wie Filterung, Kantendetektion, Segmentierung oder Bildklassifikation. Die eigentliche Programmierung erfolgt jedoch „klassisch“ in C, C++,C# oder Visual Basic. Einige Funktionen können zur Steigerung der Geschwindigkeit auch die Grafikkarte des Systems benutzen.

Spezielle Sprachen zur Parallelverarbeitung Bei der Benutzung von Bibliotheken zur Bildverarbeitung oder Programmierumgebungen sind die dort enthaltenen Algorithmen in der Regel bereits hoch optimiert und unterstützen die Paradigmen der Zielarchitektur. Abhängig von der Architektur sind verschiedene Methoden zur Optimierung der Performance möglich.

- Programmierung auf Assembler-Ebene, um die Gegebenheiten einer Architektur optimal auszunutzen
- Benutzung von Threads bei Programmierung in einer Hochsprache

Die Autoren der Publikation [BCJK05] sehen als Lösung für die automatische Anpassung der Aufgabenstellung an die zur Verfügung stehende Hardware die Programmierung der Algorithmen in einer festgelegten Form, in der explizit die Parallelität auf Datenebene vorgegeben wird. Die Ausnutzung der Parallelität auf Aufgabenebene, die üblicherweise auf einer höheren Ebene der Bildverarbeitung vorkommt, kann ebenfalls auf entsprechende Weise erfolgen. In dem von den Autoren entwickelten Softwaresystem werden dann über ein Remote-Procedure-Call-System[10] die einzelnen Teilaufgaben zur Ausführung gebracht.

2.3.4. Beispiele intelligenter Kameras

Nachfolgend werden einige Beispiele intelligenter Kameras vorgestellt, zunächst solche aus Alltagsgegenständen, danach frei programmierbare intelligente Kameras zur industriellen Bildverarbeitung.

Nintendo Wiimote

Die Nintendo Wiimote ist ein Spielecontroller für die Spielkonsole Nintendo Wii, die 2006 auf den Markt kam. Die Besonderheit bei der Spielsteuerung ist, dass die Position und

[10] entfernter Funktionsaufruf

Lage des Spielecontrollers sowie die auf ihn wirkende Beschleunigung erfasst und als Eingabe für die Videospiele benutzt wird. Die Wiimote kommuniziert kabellos über eine Bluetooth-Verbindung mit der Konsole und wird über Batterien mit Strom versorgt. Die Funktionsweise der Wiimote wurde per Reverse-Engineering entschlüsselt und in [Lee08] veröffentlicht.

In der Wiimote ist eine Infrarot-Kamera eingebaut, die als intelligente Kamera betrachtet werden kann (siehe Abbildung A.2, Appendix A, Seite 236). Aufgabe dieser Kamera ist es, die Position der Wiimote im Raum zu bestimmen. Hierzu wird über dem Bildschirm ein Gerät mit zwei Infrarotlichtquellen aufgestellt. Diese Lichtquellen werden von der Wiimote erfasst und über die geometrischen Relationen wird die Position der Wiimote berechnet.

Die Kamera verfügt über die Fähigkeit, gleichzeitig bis zu 4 Infrarotlichtquellen zu erkennen und zu verfolgen. Die genauen Daten des Sensors sind nicht veröffentlicht. Laut [Lee08] verfügt die Kamera wahrscheinlich über eine Auflösung von 1024 x 768 Pixeln und eine Bildwiederholrate von 100 Hz. Die Intensität beträgt mindestens 4 Bit pro Pixel. Die Wiimote überträgt keine unverarbeiteten Bilddaten, sondern die Position, Größe und Intensität der erkannten Lichtquellen. Anderenfalls wäre die kabellose Übertragung per Bluetooth nicht möglich, da die Datenrate viel zu hoch wäre. Die Datenrate beträgt für den Fall, dass die kompletten Bilddaten mit 4 bit Intensitätswerten übertragen werden:

$$D = 100Hz \cdot 1024 \cdot 768 \cdot 0,5 = 39,3MB/s$$

Diese Datenrate übersteigt deutlich die mögliche Datenrate von $0,26MB/s$, die unter Bluetooth 2.0 möglich wäre. Somit liegt der Hauptzweck des Einsatzes einer intelligenten Kamera in der Reduktion des Datenvolumens. Vermutlich sind die Tracking-Algorithmen direkt als FPGA realisiert. Dies würde zum einen den Energieverbrauch des mobilen Geräts minimieren, zum anderen wäre ein Full-Custom-Design in Anbetracht der großen Stückzahl sinnvoll.

Aufgrund der Konnektivität über Bluetooth und wegen des Fehlens alternativer Sensorsysteme in dieser Preislage kommt die Wiimote auch in diversen Forschungsprojekten zum Einsatz.

In [SPHB08] wird versucht, verschiedene Gesten zu unterscheiden, die ein Benutzer mit einer Wiimote ausführt. Die Rohdaten werden zunächst gefiltert, um Sensorrauschen und andere Effekte zu unterdrücken. Die Erkennungsrate verschiedener Gesten liegt im Bereich von 90 %.

Die Verfolgung einer Handbewegung mit Hilfe von Infrarot-Reflektoren an der Hand und einer Infrarotbeleuchtung wird in [BN08] vorgestellt. Die Szene wird von zwei fest installierten Wiimotes aufgenommen und die Reflexionen der Reflektoren werden ausgewertet. Über Triangulation lässt sich die dreidimensionale Position der Reflektoren im Raum errechnen.

Optische Computermäuse

Eine optische Computermaus ist ein Beispiel für einen Alltagsgegenstand, in dem ein intelligentes Kamerasystem integriert ist. Mechanische Computermäuse funktionieren nach dem Prinzip, dass in der Maus eine Kugel eingebaut ist, die sich bei Bewegung der Maus dreht. Die Kugeldrehung wird auf zwei Rollen übertragen, deren Drehbewegung über optische Encoder gemessen wird. Insbesondere bei Verschmutzung der Rollen ist die Funktion nicht zuverlässig; regelmäßige Reinigung ist erforderlich.

In der Patentschrift [Kir85] wird eine optisches Sensorsystem für Computermäuse beschrieben. Bei dieser frühen optischen Maus ist eine spezielle Unterlage notwendig, die ein rechteckiges Linienmuster aufweist. Die Maus selbst verfügt über zwei Zeilenkameras, eine für die Detektion der Bewegung in X-Richtung und eine für Detektion der Bewegung in Y-Richtung. Die Oberfläche wird durch Leuchtdioden beleuchtet. Die eigentliche Erkennung erfolgt über den sogenannten Optischen Fluss. Durch den Vergleich zweier aufeinanderfolgender Bilder wird die Bewegung der Maus errechnet.

Heutige optische Computermäuse verwenden stattdessen eine Flächenkamera zur Abtastung der Oberfläche. Die Auflösung des Sensors liegt im Bereich von 16x16 bis 30x30 Pixeln, die Abtastrate liegt bei bis zu 1500 Hz. Eine spezielle Unterlage ist in den meisten Fällen nicht mehr erforderlich, die Funktion ist jedoch meist zuverlässiger, wenn die Oberfläche strukturiert ist. Auf glatten Oberflächen sollen optische Computermäuse mit Infrarot-Laserbeleuchtung Verbesserungen bringen, da hierbei auch auf scheinbar glatten Oberflächen Strukturen sichtbar werden.

Der Einsatz von intelligenten Kameras in optischen Mäusen hat zwei entscheidende Vorteile: Zum einen wird die Kompatibilität zu den Standards der Schnittstellen beibehalten, zum anderen wird die Datenmenge reduziert, so dass die Auslastung der Datenbusse verringert wird.

Mobiltelefon-Kameras als intelligente Kameras

Mit der Anwendung Google Goggles ist es möglich, mit einem geeigneten Mobiltelefon Informationen über Objekte im Sichtfeld der Handy-Kamera abzurufen. Die Anwendung wird auf der Internetseite [gog12] vorgestellt. Als sogenannte App kann diese für Mobiltelefone und Tablet-PCs mit iOS[11] und Android Betriebssystem über das jeweilige Software-Verwaltungs-System installiert werden. Einige der in der aktuellen Version verfügbaren Funktionen sind:

- **Code-Erkennung:** Es können Barcodes auf Produkten sowie sogenannte QR-Codes [Wal09] beispielsweise auf Werbeplakaten ausgewertet werden. Über die QR-Codes wird üblicherweise eine Internetadresse beschrieben, die dann auf dem Mobiltelefon geöffnet werden kann. Die Barcode-Erkennung kann weitere Informationen über erkannte Produkte (Inhaltsstoffe, Preisvergleich) liefern.

[11] Apple iPhone u. iPad

- **Landmarken-Erkennung:** Die Applikation erkennt bekannte Gebäude und Sehenswürdigkeiten im Bildbereich der Kamera. Über die eingebaute Suchfunktion können weitere Informationen hierzu abgerufen werden.
- **Text-Erkennung:** Die Applikation kann verwendet werden, um einen Text zu erfassen. Dieses dient vor allem dazu, den Text anschließend in eine andere Sprache zu übersetzen, beispielsweise auf einer Menükarte.

Das Gesamtsystem aus Mobiltelefon und Software weist in vielerlei Hinsicht Aspekte einer intelligenten Kamera auf. Zunächst einmal zählt hierzu, dass die primäre Aufgabe des Systems nicht die Aufnahme von Bildern ist, sondern die Bereitstellung von extrahierten Bildinformationen. Des Weiteren spricht für diese Einordnung, dass die Bildverarbeitung direkt auf dem Kamerasystem durchgeführt wird und dass die Rechenaufgaben auf das mobile Endgerät und auf stationäre Server aufgeteilt werden.

Verkehrszeichenerkennung

In Kraftfahrzeugen sollen Systeme zur Verkehrszeichenerkennung für eine erhöhte Sicherheit sorgen. Die aktuell gültige Geschwindigkeitsbegrenzung und andere aktuell gültige Verkehrsregelungen werden in das Sichtfeld des Fahrers eingeblendet. Hierdurch soll verhindert werden, dass der Fahrer versehentlich die Geschwindigkeit überschreitet, weil er ein Verkehrsschild übersehen oder das Vorhandensein vergessen hat. Die Erkennung erfolgt in der Regel über eine Kamera an dem Rückspiegel.

Auch eine solche Kamera zur Verkehrszeichenerkennung kann als intelligentes Kamerasystem betrachtet werden. Auch wenn hier die Informationen nicht zwangsläufig direkt im Kamerasystem verarbeitet werden, ist für die Einordnung als intelligentes Kamerasystem ausschlaggebend, dass die Hauptfunktion des Systems darin besteht, abstrakte Bildinformationen zu generieren. In der Forschung ist die Verkehrszeichenerkennung bereits seit längerem ein Thema. Eine FPGA-Implementierung einer Verkehrzeichenerkennung wird in [DLEMSA97] gezeigt.

In Serienfahrzeuge wurde im Jahr 2008 im BMW 7er ein Verkehrszeichenerkennungssystem des Zulieferers Continental eingebaut (vgl. [LS11]). Auch in Fahrzeugen anderer Hersteller sind mittlerweile Verkehrszeichenerkennungssysteme erhältlich.

Frei programmierbare intelligente Kameras

Im Gegensatz zu den oben beschriebenen intelligenten Kameras müssen die im Folgenden beschriebenen Kamerasysteme vom Anwender programmiert werden. Die Kameras sind somit in verschiedenen Anwendungen einsetzbar, auch wenn die Eignung oft auf bestimmte Bereiche eingeschränkt ist. Anhand dreier Beispiele soll diese Kategorie erläutert werden. Die Kameras unterscheiden sich durch ihre Recheneinheiten und dementsprechend auch in ihrer Architektur, Programmierung und Leistungsfähigkeit.

Abbildung 2.7.: Die intelligente Kamera Ximea CURRERA G. Durch Benutzung eines AMD-Prozessors mit integrierter GPU können Bildverarbeitungsalgorithmen beschleunigt auf den Shadern der Grafikeinheit ausgeführt werden. In dem Foto sind vier weitere USB-Kameras abgebildet, die an die CURRERA G angeschlossen werden können. (Quelle: Bildmaterial der Ximea GmbH)

Basler eXcite Die Basler eXcite, die in Rahmen dieser Arbeit genutzt wird, zählt zu den frei programmierbaren intelligenten Kameras. Die Kamera verfügt über einen MIPS Prozessor mit 1 GHz Taktfrequenz, 128 MB Arbeitsspeicher und 128 MB Flash-Speicher. Auf der Kamera ist eine angepasste Linux-Version installiert. Die Kamera war in verschiedenen Ausführungen erhältlich, die sich hinsichtlich des verwendeten Bildsensors unterschieden. Im Jahr 2008 wurde der Vertrieb der Kamera eingestellt. Eine detaillierte Beschreibung der Hardware dieser Kamera findet sich in Abschnitt 3.1.1.

Ximea CURRERA Serie Die Firma Ximea[12] vertreibt ebenfalls intelligente Kamerasysteme. Bei der CURRERA R handelt es sich um eine intelligente Kamera basierend auf einem Intel Atom Prozessor. Sie verfügt über 1 GB Arbeitsspeicher sowie über eine 4 GB SSD[13]. Auf der Kamera kann entweder Windows Embedded oder Linux installiert werden. Es werden mehrere Bibliotheken zur Bildverarbeitung unterstützt, darunter Halcon, LabView und OpenCV. Die Kamera kann mit unterschiedlichen Bildsensoren ausgerüstet werden, die sich in Auflösung (0,4 - 5 MPixel) und Bildwiederholrate (15 - 60 fps) unterscheiden. Wie bei der Basler eXcite steht zur Anbindung eine Gigabit Ethernet Schnittstelle zur Verfügung. Des Weiteren verfügt die Kamera über digitale Ein- und Ausgänge, über serielle und über USB-Schnittstellen.

[12] www.ximea.com
[13] Solid State Drive, Massenspeicher auf Flash-Basis

Die Firma Ximea hat die Markteinführung einer weiteren intelligenten Kamera, der CURRERA G angekündigt (s. Abbildung 2.7)[cur13]. [14] Der Hauptunterschied zur CURRERA R ist, das diese Kamera auf einem AMD-Prozessor basiert (AMD G-T56N), der neben zwei CPU-Kernen mit 1,6 GHz auch eine GPU mit 80 Shadern besitzt. Die GPU kann ebenfalls zur Bildverarbeitung verwendet werden und ermöglicht in Abhängigkeit von der Aufgabenstellung eine deutliche Leistungssteigerung. Auch im Rahmen dieser Arbeit zeigen sich die Grenzen der CPU-basierten Verarbeitung, insbesondere wenn aufgrund des kleinen Gehäuses nur leistungsschwächere CPUs eingesetzt werden. Daher wird hier, wie in Abschnitt 4.6 beschrieben, ebenfalls die Möglichkeit untersucht, GPU basierte Verarbeitung zu unterstützen. Da die Kamera zum Zeitpunkt der Untersuchungen nicht verfügbar ist, wird eine AMD-Grafikkarte mit gleicher Anzahl an Shadern und ähnlicher Taktrate verwendet, um so eine Abschätzung der Leistungsfähigkeit vornehmen zu können.

CMUcam Bei der CMUcam (s. Abbildung A.3, Appendix A, Seite 237) handelt es sich um ein kompaktes Kameramodul mit einer Verarbeitungseinheit basierend auf einem 32-bit Microcontroller [cmu13]. Mittlerweile ist die vierte Version des Systems verfügbar. Das Entwicklungsziel bei der CMUcam besteht darin, eine kostengünstige Möglichkeit zu schaffen, Kameramodule in Eingebettete Systeme zu integrieren. Die Softwareentwicklung ist in der Programmiersprache C mit Open-Source-Werkzeugen zum Kompilieren möglich. Aufgrund der beschränkten Prozessorleistung werden auf dieser Kamera eher einfache Bildverarbeitungsalgorithmen ausgeführt. So wird in [DI09] beispielsweise ein farbwertbasiertes Objekt-Tracking auf einer mobilen Roboterplattform gezeigt. Die Anwendung der CMUcam auf einem Robotersystem zur Unkrautbekämpfung in der Landwirtschaft wird in [KMLR05] vorgestellt. Auf der CMUcam werden dabei die Farbinformationen der Nutzpflanzenreihen ausgewertet und bei der Spurführung des Robotersystems berücksichtigt.

2.3.5. Intelligente Kameras in der Forschung

Im Folgenden werden weitere Forschungsprojekte vorgestellt, die sich mit intelligenten Kamerasystemen beschäftigen. Je nach Projekt verfügen die Kamerasysteme über unterschiedliche Fähigkeiten, durch die sie sich von klassischen digitalen Kameras unterscheiden. Ein entscheidender Aspekt bei der Integration von Funktionseinheiten in die Kamerasysteme ist die Informationsverarbeitung in Echtzeit, die mit Standard-PC-Systemen oft nur unzureichend möglich ist.

Ein Forschungsprojekt zur Erzeugung von Panoramabildern basierend auf einer intelligenten Kamera zeigt [BMMC12]. Die Erzeugung des Panoramabildes erfolgt durch eine rotierende Zeilenkamera, die vertikal ausgerichtet ist und um eine lotrechte Achse rotiert (siehe Abbildung A.4, Appendix A, Seite 237). Zur Verarbeitung der Bilddaten wird ein

[14] Diese Kamera ist zum Zeitpunkt März 2013 noch nicht verfügbar.

digitaler Signalprozessor verwendet. Abhängig von der Auflösung erfolgt die Bildaufnahme an festgelegten Winkelwerten. Die Rotationsgeschwindigkeit beträgt 10 Umdrehungen pro Sekunde, was eine Bildrate von 10 Panoramabildern pro Sekunde ergibt. Aufgrund der stetigen Drehbewegung des Systems ist nur eine kurze Verschlusszeit möglich. Um dennoch Bilddaten in nutzbarer Qualität zu erzielen, kommen ein spezieller Sensor sowie spezielle Algorithmen zur Vorverarbeitung zum Einsatz. Die Autoren heben die Besonderheit hervor, dass die erzeugten Panoramabilder verzeichnungsfrei sind, da die Kamera um ihr optisches Zentrum rotiert.Als Vorverarbeitung wird direkt auf dem Kamerasystem eine Kantendetektion implementiert. Statt Bilddaten werden bei Detektion einer Kante Events ausgegeben. Auf diese Weise kann die Datenrate stark reduziert werden. Dies ist insbesondere deswegen von Bedeutung, weil die Datenübertragung mittels eines Schleifringübertragers erfolgt, der eine begrenzte Bandbreite hat.

Forschungsgegenstand in [SH07] ist die Entwicklung eines Kamerasystems mit FPGA-basierter Vorverarbeitung. Ziel des Projekts ist eine Bildwiederholrate von 1000 Bildern pro Sekunde in Verbindung mit Echtzeitverarbeitung. Aus den Bilddaten soll die planare Bewegung eines Objekts ermittelt werden. Das Gesamtsystem besteht aus einer CMOS-Kamera, die über eine Channel-Link-Schnittstelle an ein PCI-FPGA-Board mit einem Xilinx Virtex Pro 6000 gekoppelt ist. Auf diesem Board stehen 1 GByte Speicher zur Verfügung. Momentan wird bei dem System eine Echtzeitverarbeitung von 64 x 64 Pixeln erreicht, geplant ist eine Erhöhung auf 256 x 256 Pixel.

In [KUI+06] wird ein intelligentes Hochgeschwindigkeitskamerasystem für mobile Roboter gezeigt, das über zwei bewegliche Kameras verfügt. Die hohe Geschwindigkeit bezieht sich zum einen auf die Bildwiederholrate, zum anderen auf die Reaktionszeit des Gesamtsystems auf Bildinformationen. Als Testszenario wird hier die Kameranachführung bei der Verfolgung eines Objekts betrachtet. Die Bildaufnahmesensoren unterscheiden sich von denen herkömmlicher Kameramodelle dadurch, dass schon im Bildsensor eine Verarbeitung vorgenommen wird. Dazu ist der Sensor in mehreren Schichten aufgebaut. Die Bilddatenverarbeitung ist an die Funktion des menschlichen Auges angelehnt. Beim Auslesevorgang werden also von dem Bildsensor schon extrahierte Bildinformationen geliefert. Auf diese Art soll die auftretende Verzögerung, die bei der üblichen Form der digitalen Bildverarbeitung durch das Kopieren der Daten in den Bildspeicher entsteht, umgangen werden. Die Auflösung liegt in der vorgestellten Version bei 64x64 Pixeln. Die Sensoren werden durch einen PC mit einem Pentium-MMX-Prozessor gesteuert.

2.4. Eingebettete Systeme

Zu Grundlagen der Eingebetteten Systeme (*embedded systems*) finden sich Informationen in [Möl03] und [Cat05]. Es wird dort ein Überblick gegeben über die Einsatzgebiete der Eingebetteten Systeme, ihre Strukturen und die Programmierung. Der Bezug dieser Arbeit zu eingebetteten Systemen liegt darin begründet, dass es sich bei den Recheneinheiten der

intelligenten Kameras um Eingebettete Systeme handelt. Im Rahmen dieser Arbeit wird eine Software-Architektur entwickelt, die auch für solche Systeme eingesetzt wird.

Bei Eingebetteten Systemen handelt es sich um informationsverarbeitende Rechnerstrukturen, die in größere Umgebungen integriert sind. Innerhalb dieser Umgebungen ist diesen Rechnerstrukturen eine von vornherein festgelegte Funktionalität zugeordnet. In diesem Merkmal liegt der entscheidende Unterschied zwischen Eingebetteten Systemen und Desktop-PC Systemen, bei denen sich die Funktion ständig entsprechend den aktuellen Wünschen des Anwenders verändert.

Die Umgebungen, in die Eingebettete Systeme integriert sind, zeichnen sich durch eine sehr starke Heterogenität aus. Es können Aufgaben unterschiedlichster Komplexität vorliegen. Sowohl digitale Daten als auch analoge Signale können zu verarbeiten sein. Viele der Geräte, die im Alltagsleben Anwendung finden, sind bereits mit Eingebetteten Systemen ausgestattet. Einige Beispiele aus den verschiedensten Bereichen sind Mobiltelefone, Digitalkameras, Fernsehgeräte, Telefonanlagen, Mikrowellengeräte, Waschmaschinen, Automobile, Flugzeuge, medizinische Geräte und Fahrkartenautomaten. Eingebettete Systeme werden auch zur Robotersteuerung eingesetzt [Brä08]. Ebenso können Rechnerstrukturen zur Steuerung von Industrieanlagen als Eingebettete Systeme betrachtet werden (z.B. Steuerung eines Kernkraftwerks).

2.5. Verteilte Systeme in der Robotik

Die Aufteilung der durchzuführenden Tasks auf mehrere Systeme ist ein weiteres Thema, an dem aktiv geforscht wird. Der Bezug zum eigenen Forschungsvorhaben besteht darin, dass es sich bei intelligenten Kamerasystemen von der Architektur her betrachtet um Computersysteme mit angebundener digitaler Kamera handelt, die in einem Gehäuse untergebracht sind; werden diese über die Netzwerkschnittstelle an den Steuerrechner des Roboters angebunden, so fällt die Gesamtarchitektur in die Kategorie der verteilten Systeme. Bei den Forschungsthemen bezüglich verteilter Systeme in der Robotik reicht die Ausrichtung von grundlegenden Themen, die die Architektur und das Zusammenspiel der verschiedenen Systeme behandeln, bis hin zu Projekten, die verteilte Systeme in erster Linie als Werkzeug einsetzen, um ein bestimmtes Ziel zu erreichen.

Eine Spezialform des Einsatzes verteilter Systeme stellt das sogenannte Cloud-Computing dar [MG11]. Hierunter ist zu verstehen, das ein System Zugriff auf die Rechenkapazität und Dienste einer Vielzahl von Systemen hat. Es können komplexe Aufgaben parallel auf einer Vielzahl von Systemen gelöst werden und somit die Rechenzeiten deutlich reduziert werden. Das oben erwähnte Forschungsprojekt [LRM+11] verwendet zur Berechnung beispielsweise 1000 Systeme. Cloud-Computing erfordert in der Regel besondere Mechanismen zur Verwaltung der Aufgaben und zur Reaktion auf den Ausfall einzelner Systeme.

Als Beispiel für ein Forschungsprojekt, das ein verteiltes System benutzt, um Gegenstände

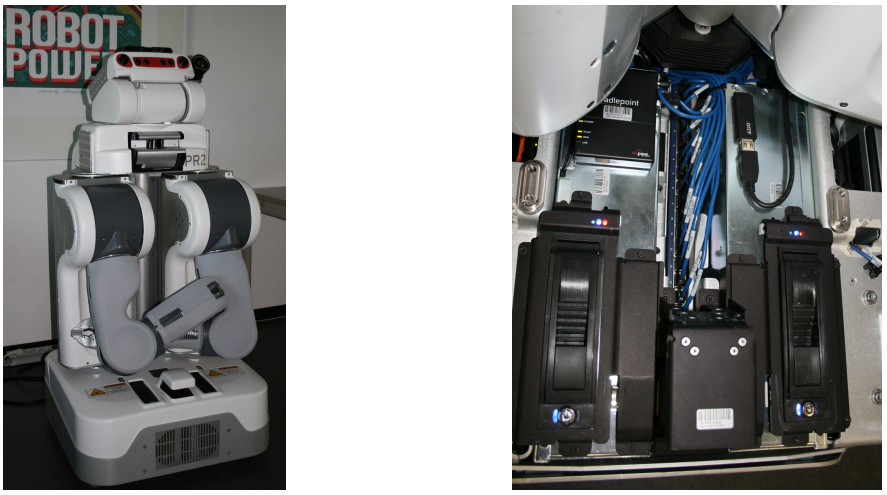


Abbildung 2.8.: Der Service-Roboter PR2 (links) der Firma Willow Garage. Rechts ist (nach Demontage der Gehäuseteile) der zentrale Ethernet-Switch zu sehen, über den die verschiedenen Hardware-Komponenten kommunizieren.

mit einem Roboterkopf zu verfolgen, soll hier [PHZ+06] angeführt werden. Die Bilddaten werden mit einem Stereo-Kamerasystem auf einem beweglichen Roboterkopf aufgenommen. Ziel der Entwicklung ist es, diesen Roboterkopf inklusive Kamerasystem einem Gegenstand in Echtzeit nachzuführen. Als Merkmale werden dreidimensionale Bildinformationen, Farbwert und Form des Zielobjekts ausgewertet. Zusätzlich wird basierend auf der aktuellen Objektbewegung eine Vorhersage der Position integriert, um auch zeitweise Verdeckung zu kompensieren. Im Versuchsaufbau, mit dem das echtzeitfähige Nachführen des Kopfes möglich ist, werden zwei PC-Systeme verwendet. Das eine System basiert auf einem Pentium 4 mit 2,4 GHz und dient zur Bildverarbeitung, das andere basiert auf einem Pentium III mit 700 MHz und wird zur Steuerung des Roboterkopfes eingesetzt.

Das Prinzip der Anwendung verteilter Systeme ist nicht auf Videosensoren beschränkt. In [BWZ07] ist ein System zur Anbindung von SICK-Lasermesssystemen an ein Robotersystem dargestellt. Zwischen den Lasermesssystemen und dem Steuerrechner des Roboters ist ein Microcontroller-Board geschaltet, das die Steuerung der Messsysteme, die zeitkritische Entgegennahme der Messdaten und einfache Vorverarbeitung übernimmt. Es kann eine deutliche Reduzierung der Systemauslastung des Steuerrechners vom Robotersystem erreicht werden. Zusätzlich werden Messdatenverluste vermieden, die zuvor aufgrund nicht eingehaltener Timing-Anforderungen aufgetreten waren.

Als ein weiteres Beispiel für ein Robotersystem, das verteilte Systeme verwendet, soll der Service-Roboter PR2 von Willow Garage erwähnt werden [Cou10]. Das Robotersystem wird von zwei PCs mit einem Intel Core i7 Prozessor gesteuert. Die Benutzung von zwei PCs erfolgt, weil die verwendete Softwarearchitektur sowie die Algorithmen zur Robotersteuerung eine hohe Rechenleistung erfordern. Diese Steuerrechner sowie viele weitere

auf dem Roboter integrierte Geräte (z.B. Kameras) kommunizieren über einen Gigabit-Ethernet-Switch. Auf diese Art ist es möglich, eine Vielzahl von Systemen über eine standardisierte Schnittstelle anzubinden. Beide PC-Systeme sind jeweils über zwei Netzwerkkarten an den Switch angeschlossen, um möglichen Problemen einer zu geringen Bandbreite entgegenzuwirken. Abbildung 2.8 zeigt das Robotersystem sowie den Ethernet-Switch, über den viele der integrierten Geräte kommunizieren.

2.5.1. Software-Frameworks mit Unterstützung verteilter Systeme

Um eine Robotersteuerung auf verteilten Systemen zu implementieren, werden die einzelnen Algorithmen oft in sogenannten Frameworks gekapselt. Mittels dieser Frameworks kann der Implementierungsaufwand durch Bereitstellung von Funktionen zum Verwalten von Diensten und zur netzwerktransparenten Kommunikation verringert werden.

Player

Eine weite Verbreitung hat das Open-Source-Framework Player erlangt, das in [GVS+01] vorgestellt wird. Hierbei handelt es sich um eine Client/Server-Architektur zur Steuerung von Sensoren und Aktuatoren. Der Player Server bietet standardisierten Zugriff auf die Roboterhardware. Viele gängige Sensoren und Roboterplattformen werden unterstützt. Über eine TCP/IP-Verbindung stellt der Client dann eine Verbindung zu dem Server her. Die Kommunikation ist in einem Protokoll festgelegt, so dass der Client in beliebiger Programmiersprache implementiert werden kann. Eine Kopplung an die Simulationsumgebung Stage ist ebenfalls in Player integriert.

Roblet

Ein Softwareframework auf Basis der plattformübergreifenden Programmiersprache Java ist die Roblet-Technologie. Diese wurde in [WSZ06] veröffentlicht. Die Besonderheit hierbei ist, dass zu den einzelnen Instanzen, auf denen die Software läuft, beliebige Java-Programme geschickt werden können, die dann auf bestimmte Geräte des Systems zugreifen können. Hierdurch unterscheidet sich diese Architektur deutlich von allen anderen, da nicht mit festgelegten Aufrufen von Funktionen gearbeitet werden muss und die Funktionalität zur Laufzeit beliebig erweiterbar ist. Im Zusammenhang mit intelligenten Sensoren findet diese Technologie Erwähnung in [WZ08].

ROS

In den vergangenen Jahren hat das Framework für mobile Roboter ROS (Robot Operating System) von Willow Garage zunehmende Verbreitung gefunden [QGC+09]. Da im Rahmen

dieser Arbeit auch gezeigt wird, wie die intelligente Kamera und die hier entwickelten Softwarekomponenten unter ROS eingesetzt werden können, sollen die Konzepte dieses Frameworks genauer beschrieben werden.

Entgegen der Bezeichnung handelt es sich nicht um ein Betriebssystem, sondern um ein Framework, das die netzwerkübergreifende Kommunikation zwischen verschiedenen Software-Komponenten und der Roboter-Hardware regelt. ROS wird hauptsächlich unter Linux (Ubuntu) eingesetzt, wobei auch Windows und MacOS partiell unterstützt werden. In ROS können auch verschiedene Typen von Kameras angesprochen werden.

Das Grundprinzip von ROS besteht darin, dass verschiedene Software-Komponenten, sogenannte „Nodes“, über einen zentralen Server, den sogenannten „Roscore“, kommunizieren. Die Kommunikation erfolgt stets per TCP, somit können sich die verschiedenen „Nodes“ auch auf unterschiedlichen Systemen befinden. Eine Einschränkung hinsichtlich der Interoperation stellt die Tatsache dar, dass sich jeder „Node“ stets nur zu einem „Roscore“ verbinden kann.

Die Interaktion der „Nodes“ erfolgt entweder Push-basiert über „Topics“ oder über entfernte Funktionsaufrufe, die in ROS als „Services“ bezeichnet werden. In beiden Fällen erfolgt die Koordination durch den „Roscore“, und die einzelnen „Topics“ bzw. „Services“ können über konfigurierbare Namen identifiziert werden. Für die Übertragung von Sensordaten wird der Mechanismus der „Topics“ genutzt. Bei Sensordaten gibt es in der Regel pro „Topic“ einen „Publisher“, der die Daten erzeugt, und beliebig viele „Subscriber“, die die Daten empfangen und dann z.B. auswerten und die Ergebnisse auf einem anderen „Topic“ bereitstellen oder über „Services“ Steuerkommandos an die Aktuatoren des Roboters versenden.

Ein Vorteil der Architektur von ROS ist, dass die „Nodes“ beliebig gestartet und beendet werden können. So können beispielsweise beim Absturz eines „Nodes“ die übrigen weiterarbeiten, solange sie nicht von diesem „Node“ abhängen. Aber auch im Falle eines Absturzes kann in der Regel durch ein Neustarten des abgestürzten „Nodes“ weitergearbeitet werden.

2.6. Aufgabenzuweisung in Bildverarbeitungssystemen

In der vorliegenden Arbeit wird ein System entwickelt, mit dem Teilaufgaben der Bildverarbeitung auf verschiedene Systeme verteilt werden. In den ersten Tests erfolgt die Aufteilung auf das intelligente Kamerasystem und den Steuerrechner des Roboters, in weiteren Tests zusätzlich auf externe Systeme. Daher sollen ähnliche Forschungsansätze genauer betrachtet werden.

In [TCP+99] wird ein System vorgestellt, mit dem Bildverarbeitungsaufgaben über einen grafischen Editor gesteuert werden können. Die einzelnen Operationen werden durch IPOs (Image Processing Objects) repräsentiert. Diese IPOs werden dann zu einem Graphen verknüpft. Neben der Reihenfolge der Ausführungsschritte kann auch der exklusive Zugriff auf

Ressourcen modelliert werden. Zur Laufzeit werden die einzelnen Arbeitsschritte auf die verfügbaren Systeme und deren Prozessoren verteilt. Hierbei wird der Rechenaufwand der einzelnen Aufgaben ausgewertet und in die Verteilung mit einbezogen. Das System wird exemplarisch bei der Tiefenrekonstruktion der Bilddaten eines Stereo-Kamerasystems angewendet. Die Funktion beschränkt sich auf die Verarbeitung eines Bilderpaares, die Eignung zur Verarbeitung von kontinuierlichen Datenströmen wird nicht erwähnt. Die Anwendung dieses Systems wird in [TCPO00] im Kontext einer Erkennung von Banknoten gezeigt.

Forschungen zum Scheduling von mehreren Bildverarbeitungsaufgaben werden in [XJ05] präsentiert. Ziel ist es, mehrere voneinander unabhängige Algorithmen (Objekterkennung, Gesichtslokalisation, Objekt-Tracking, Background Subtraction, Videokomprimierung) auf einem Videodatenstrom anzuwenden. Die Ausführung erfolgt auf einem Computersystem, das bei klassischem Scheduling nicht in der Lage ist, alle Aufgaben mit akzeptabler Latenz auszuführen. Bei dem gezeigten Ansatz werden die einzelnen Aufgaben zunächst bezüglich der akzeptablen Latenz klassifiziert. Das System wird dann mittels verschiedener Testszenen trainiert. Hierbei werden die Ausführungszeiten der Algorithmen in Abhängigkeit vom Bildinhalt erfasst. Zur Laufzeit wird mit Hilfe der so gewonnenen Informationen das Scheduling optimiert. Es kann gezeigt werden, dass die zeitkritischen Operationen zuverlässig ausgeführt werden. Die unkritischen Operationen mit geringerer Priorität werden in Abhängigkeit von der verfügbaren Rechenleistung so gut wie möglich ausgeführt.

In [WOHK10] wird die Aufgabenzuweisung auf heterogene Multicore-Prozessoren zur Verarbeitung von Multimediadaten untersucht. Hierbei wird insbesondere auf die Verarbeitungszeit und die Datentransferzeit eingegangen. In Bezug auf solche Prozessoren ergibt sich überdies die Herausforderung, dass bestimmte Algorithmen nur auf speziellen Prozessoren ausführbar sind. Es kommt ein Verfahren zum Einsatz, bei dem die Algorithmen zur Laufzeit für bestimmte Prozessortypen übersetzt werden, um so größere Flexibilität bei der Aufgabenzuweisung zu erreichen und unnötigen Datentransfer zu vermeiden. Das Verfahren wird am Beispiel der Berechnung des sogenannten HOG-Deskriptors angewendet, einer Methode zur Beschreibung von Bildpunkten (vgl. [DT05]).

Ein Ansatz, bei dem ein mobiler Roboter Teile der Berechnung zur Objekterkennung und zum Objekttracking auf externe Systeme auslagern kann, wird in [NKLL10] gezeigt. Bei diesem Konzept kann der Roboter die Teile der Bildverarbeitung wahlweise auf seinem Steuerrechner oder auf einem leistungsfähigeren externen Server ausführen. Abhängig von der Bandbreite der drahtlosen Verbindung und der Komplexität der Aufgabe wählt das System ein geeignetes Setup. Es stehen drei vorab festgelegte Konfigurationen zur Auswahl:

- Alle Daten werden lokal auf dem mobilen Roboter verarbeitet.
- Die unkomprimierten Bilddaten werden an den Server übertragen und dort verarbeitet.
- Die Extraktion von Merkmalen erfolgt auf dem Roboter; diese werden zu dem Server übertragen und dort weiterverarbeitet.

Zur Auswahl der passenden Strategie wird ein mathematisches Modell entwickelt, das die gesamte Verarbeitungszeit abschätzt. Es kann gezeigt werden, dass in verschiedenen Situationen unterschiedliche Setups gewählt werden. In praktischen Tests werden die Ergebnisse bestätigt. Insbesondere in Situationen mit beschränkter Bandbreite zeigen sich die Vorteile der verteilten Verarbeitung, da auf diese Art das Datenvolumen im ersten Schritt stark reduziert werden kann.

Ähnliche Ergebnisse wurden auch im Rahmen der Forschungsarbeiten für die hier vorliegende Arbeit erzielt. Diese sind in [BZ09] und in [BZ10] publiziert. Das oben beschriebene Forschungsprojekt [NKLL10] und die vorliegende Arbeit weisen einige ähnliche Aspekte auf. Neben dem Bezug auf mobile Roboter wird die Bildverarbeitung als Hintereinanderausführung von mehreren Schritten betrachtet und auch die Auslagerung von Aufgaben auf Systeme in der Roboterumgebung wird in beiden Arbeiten thematisiert. Unterschiede bestehen jedoch darin, dass der Schwerpunkt der hier vorliegenden Arbeit auf der Auslagerung auf intelligente Kamerasysteme liegt. Des Weiteren wird hier angestrebt, eine größere Anzahl von Algorithmen zu evaluieren und hierfür automatische Konfigurationen zu erzeugen.

2.7. Datenstromorientierte Programmierung

Datenstromorientierte Programmierung wurde zunächst in den 1970er Jahren im Zusammenhang mit speziellen datenstromorientierten Rechnerarchitekturen entwickelt (vgl. [JHM04]). Bei der Weiterentwicklung der Computersysteme mit Von-Neumann-Architektur wurden ebenfalls entsprechende Konzepte integriert und es konnten bestimmte Vorteile der beiden Architekturen kombiniert werden. Seit den 1990er Jahren entstanden mehrere datenstromorientierte Programmiersprachen, beispielsweise LabView [Joh97], Prograph [CGP89] oder Pure Data [Puc96].

Bei Datenstromorientierter Programmierung wird das Programm als gerichteter Graph zur Verarbeitung von kontinuierlichen Datenströmen erstellt [KM66]. Die Knotenpunkte des Graphen entsprechen den Operationen, die Pfeile kennzeichnen die Datenabhängigkeiten. Der Programmablauf kann auch dahingehend interpretiert werden, dass sich Token entlang des Graphen bewegen.

Zur Softwareentwicklung werden bei der Datenstromorientierten Programmierung oft visuelle Programmieroberflächen eingesetzt. Das grafische Programmiermodell besagt zunächst nur, dass der Entwickler sein Programm mit einem grafischen Editor erzeugt und die einzelnen Programmkomponenten in der Regel durch grafische Symbole repräsentiert werden [Sut66]. Durch die Betrachtung des Programms als Graph liegt die Erstellung und Manipulation des Graphen mit Hilfe einer visuellen Programmieroberfläche nahe.

Ein Beispiel für eine Datenstromorientierte Programmierung ist SCIrun[PJ95]. Bei SCIrun handelt es sich um eine Entwicklungsumgebung zur Modellierung, Simulation und zum

Lösen wissenschaftlicher Probleme. Die Entwicklung mit SCIrun erfolgt datenstromorientiert über Module. Verschiedene verfügbare Module nehmen über ihre Eingänge Daten entgegen, verarbeiten diese und leiten die Ausgabe an ihre Ausgänge weiter. Verschiedene Module werden zu einem Netz verbunden, das dann die gewünschte Gesamtfunktionalität bereitstellt. Die Entwicklung erfolgt grafisch: Die Module werden als Boxen und die Verbindungen als Leitungen dargestellt.

Die Relevanz der Datenstromorientierten Programmierung für diese Arbeit liegt darin begründet, dass die Art, wie die intelligente Kamera und weitere Systeme zur Laufzeit konfiguriert werden, auf Konzepten der Datenstromorientierten Programmierung beruht. Wie in Kapitel 3 noch näher erläutert wird, wird die Bildverarbeitungsaufgabe als gerichteter Graph modelliert, die Knotenpunkte entsprechen den Operationen auf den Bilddaten. Ein grafischer Editor wird ebenfalls im Rahmen dieser Arbeit basierend auf einem schon bestehenden Open-Source-Projekt erstellt. Die Implementierung der eigentlichen Operationen erfolgt jedoch in der klassischen Hochsprache C.

2.8. Einordnung des eigenen Forschungsvorhabens

Die eigene Arbeit unterscheidet sich signifikant von den bisher untersuchten Ansätzen. Der Hauptunterschied im Vergleich zu anderen Arbeiten besteht darin, dass in anderen Arbeiten meist eine festgelegte Anwendung auf einem festgelegten Kameratyp implementiert und optimiert wird. Im Gegensatz dazu wird in dieser Arbeit das Ziel verfolgt, möglichst viele relevante Algorithmen zu implementieren und diese vom Prinzip her auf möglichst vielen Typen von intelligenten Kamerasystemen ausführbar zu machen. Aus diesem Grund werden keine Optimierungen für eine bestimmte Architektur vorgenommen (z.B. Assemblerprogrammierung), sondern die Algorithmen in der Programmiersprache C implementiert. Insbesondere besteht so die Möglichkeit, Algorithmen aus frei verfügbaren Bibliotheken mit einzubinden und somit den Funktionsumfang des Systems erheblich zu erweitern. Nur durch diesen Ansatz kann sichergestellt werden, dass das System auch weiterverwendbar ist, wenn eine neuere Kameragenerationen verfügbar ist. Der Fokus liegt zudem auf der Leistungsmessung intelligenter Kamerasysteme sowie auf einer Analyse des Overheads, der durch die Netzwerkverbindung verursacht wird. Für verschiedene Anwendungen kann mit Hilfe der entwickelten Methoden geprüft werden, ob der Einsatz intelligenter Kamerasysteme nutzbringend ist. Gegebenenfalls kann untersucht werden, woran der Einsatz scheitert und welche Voraussetzungen erfüllt sein müssten, damit der Einsatz lohnenswert ist. Hieraus können Designkriterien abgeleitet werden, die für die Entwicklung zukünftiger intelligenter Kamerasysteme speziell für Service-Roboter maßgebend sind. Zum Zeitpunkt der Fertigstellung dieser Arbeit sind keine Forschungsansätze bekannt, die sich genau mit diesem Thema beschäftigen.

Bei Änderung der aktuellen Aufgabenstellungen eines Service-Roboters können der intelligenten Kamera verschiedene Funktionen zugeordnet werden. Eines der Ziele der Implementierung besteht darin, die Bildverarbeitungsleistung eines Robotersystems zu erhöhen

und gleichzeitig die Auslastung des Steuerrechners des Service-Roboters zu verringern. Auf diese Art kann die stabile Ausführung echtzeitkritischer Aufgaben sichergestellt werden.

2.9. Zwischenfazit

In diesem Kapitel sind die Grundlagen sowie der Stand der Forschung insbesondere auf dem Gebiet der intelligenten Kameras dargelegt. Es zeichnet sich ab, dass es kaum Forschungsarbeiten gibt, die generell die verschiedenen Einsatzmöglichkeiten von intelligenten Kameras auf Service-Robotern evaluieren. Die meisten Arbeiten konzentrieren sich auf nur eine Anwendung und verwenden die intelligente Kamera vorrangig als Werkzeug, um ein bestimmtes Ziel zu erreichen. Basierend auf den hier erörterten technischen Rahmenbedingungen soll, wie im nächsten Kapitel dargestellt, ein umfassendes Gesamtkonzept entwickelt werden, um Bildverarbeitungsaufgaben auf intelligenten Kameras auszuführen und zu evaluieren. Die Evaluation erfolgt sowohl für die intelligente Kamera Basler eXcite als auch im Hinblick auf die Einsatzmöglichkeiten zukünftiger leistungsfähigerer Kamerasysteme.

3. Integration des intelligenten Kamerasystems

In diesem Kapitel wird die Entwicklung einer geeigneten Architektur zur Integration des intelligenten Kamerasystems in das Software-System eines mobilen Service-Roboters dargestellt. Nach einer Beschreibung der an den Experimenten beteiligten Systeme wird die implementierte Softwarearchitektur in Abschnitt 3.2 beschrieben. Hierbei wird auf die Anforderungen, die Konzepte, die zentrale Steuerung von verteilten Systemen und die Möglichkeiten von Leistungsmessungen eingegangen. Die darauf folgenden Abschnitte 3.3 und 3.4 stellen die relevanten Funktionen zur Bildverarbeitung und sonstige Funktionen zur Steuerung des Datenflusses vor. Hier wird unter anderem gezeigt, wie Objektdetektion und Gesichtslokalisation implementiert sind und wie die hier entwickelten Funktionen auch unter dem ROS-Framework nutzbar sind.

3.1. Beschreibung der Testhardware

Im Folgenden werden die Hauptkomponenten der Testkonfigurationen vorgestellt. Es handelt sich um die intelligente Kamera Basler eXcite und den Service-Roboter TASER.

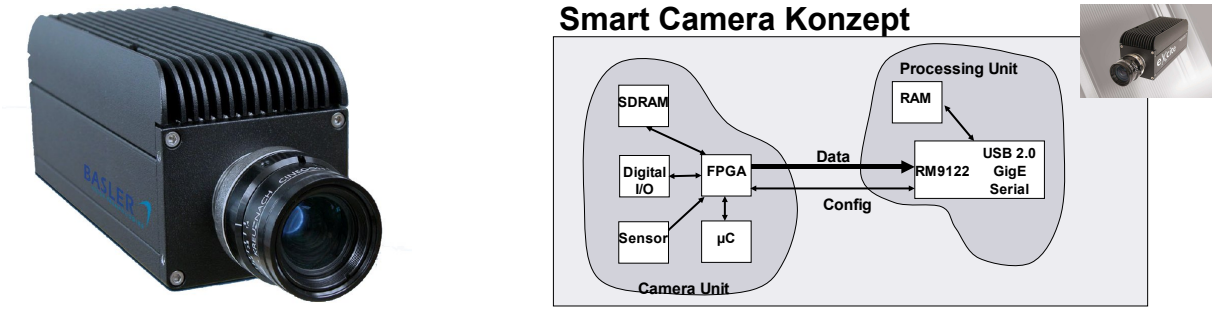


Abbildung 3.1.: Die intelligente Kamera Basler eXcite und die Hardwarearchitektur. Die Bezeichnung RM9122 steht für eine 64-bit MIPS-CPU mit 1 GHz. (Quelle: Basler AG)

3.1.1. Basler eXcite

Die Basler eXcite (s. Abbildung 3.1) ist eine sogenannte intelligente Kamera für den Machine-Vision-Bereich [Bas06]. Von der Kamera gibt es verschiedene Modelle, die mit unterschiedlichen Bildsensoren ausgestattet sind und sich somit in Auflösung und Bildwiederholrate unterscheiden. Die verwendete Kamera Basler eXcite axA1390-19c liefert Farbbilder mit einer Auflösung von 1388 x 1038 Bildpunkten und einer Bildwiederholrate von 18, 7 Bildern pro Sekunde.

Die Recheneinheit der Kamera basiert auf einem 64-bit MIPS Prozessor, der mit 1 GHz getaktet ist. Es sind 128 MB RAM und 128 MB Flash-Speicher vorhanden. Zur Kommunikation verfügt die Kamera über eine Gigabit-Ethernet-Schnittstelle. Auf dem System ist herstellerseits eine angepasste Version des Linux-Betriebssystems mit dem Kernel 2.6 installiert. In der Installation sind Treiber für die Kamera-Module und einige Programmbeispiele enthalten, jedoch keine fertigen Bildverarbeitungsalgorithmen.

Die Kamera verfügt, wie oben in der Abbildung gezeigt, über ein FPGA zur Ansteuerung des Sensors und zur Vorverarbeitung der Bilddaten. Dieses FPGA kann nicht vom Benutzer programmiert werden und stellt eine festgelegte Funktionalität bereit. Er wird hauptsächlich zur Konvertierung der Sensor-Rohdaten in andere Farbmodelle (Grauwertbild, YUV) unter Berücksichtigung von Parametern zum Weißabgleich genutzt. Auf diese Art kann, wie später in Abschnitt 4.1 gezeigt wird, Rechenzeit eingespart und die Auslastung der CPU der eXcite verringert werden.

3.1.2. Service-Roboter TASER

Der Service-Roboter TASER ist eine Forschungsplattform des Arbeitsbereichs TAMS. Der Roboter basiert auf einer Plattform der Firma NEOBOTIX[1] und wurde um einen Aufbau zur Montage zweier Roboterarme erweitert. Die Abbildung 3.2 zeigt ein Foto des Roboters.

Die mobile Plattform steht auf fünf Rädern, von denen zwei unabhängig voneinander über Elektromotoren angetrieben werden, so dass neben Geradeausfahrten auch Kurvenfahrten und Drehbewegungen um die eigene Achse möglich sind.

Für Greifaufgaben sind zwei Roboterarme des Typs Mitsubishi PA-10-6C vorgesehen. Diese sechsgelenkigen Arme haben eine Tragkraft von je 6 kg. An jedem dieser Arme ist eine Roboterhand der Firma Barret Technology angebracht, die über drei einzeln steuerbare Finger verfügt. Über entsprechende Sensoren kann die Kraft gemessen werden, die auf die Finger wirkt.

Zwei Lasermesssysteme vom Typ Sick LMS200 dienen zur Lokalisation in der Umgebung.

[1] www.neobotix.de

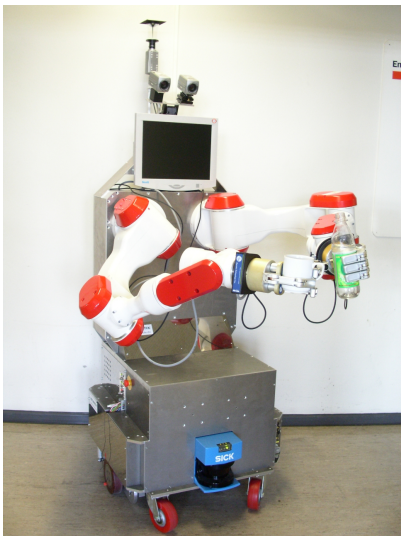


Abbildung 3.2.: Der mobile Service-Roboter TASER des Arbeitsbereichs TAMS (Quelle: Bildmaterial Arbeitsbereich TAMS)

Sie sind über ein am Arbeitsbereich TAMS entwickeltes System an die Netzwerkschnittstelle des Roboters angeschlossen. Als weitere Sensoren sind mehrere Kamerasysteme installiert. Das Stereokamerasystem basiert auf zwei Kameras des Typs Sony DFW-VL500, die über FireWire angebunden sind. Über eine Pan-Tilt-Unit lassen sich diese in nahezu beliebige Richtungen drehen. Das Omnivisionsystem basiert auf einer Kamera ähnlichen Typs. Mittels eines hyperboloiden Spiegels[2] kann ein Bild des gesamten Raumes aufgenommen werden, das dann über entsprechende Operationen in eine andere Perspektive transformiert werden kann.

Der Steuerrechner des Service-Roboters ist ein Industrie-PC mit einem Pentium-4-Prozessor. Aufgabe des Rechners ist es, alle Sensordaten entgegenzunehmen, diese zu verarbeiten und entsprechend der momentanen Aufgabe die Aktuatoren zu steuern. Im späteren Verlauf der Untersuchungen für diese Arbeit wurde das System umgerüstet und es wurde ein leistungsstärkerer PC basierend auf einem Intel Core 2 Quad Prozessor integriert. Dieser Prozessor entspricht zwar nicht dem aktuellen Stand der Technik, wurde aber gewählt, da die Integration unter Beibehaltung sämtlicher anderer Komponenten möglich ist. Da im Zuge der Umrüstung auch das Software-System in weiten Teilen modifiziert werden sollte, war die Roboterplattform in der letzten Phase der Tests nicht einsetzbar. Um die Ergebnisse reproduzieren zu können, wird in den Experimenten zum Teil mit einem PC gearbeitet, der dem früheren Steuerrechner des Service-Roboters entspricht (Intel Pentium 4 mit 2,4 GHz).

Nachfolgend werden noch einmal die Probleme der Roboterplattform dargelegt und es wird diskutiert, inwiefern intelligente Kamerasysteme hierfür einen Lösungsansatz darstellen.

[2]Hyperboloide Spiegel sind für 360° Rundumblicke mit einer Kamera geeignet.

3.1.3. Probleme des Service-Roboters TASER

Auf dem Service-Roboter TASER werden viele verschiedene Sensoren und Aktuatoren an einen Steuerrechner angebunden. Zusätzlich müssen noch weitere für den Betrieb notwendige Softwarekomponenten auf diesem Rechner ausgeführt werden. Im Einzelnen ergeben sich dadurch folgende Schwierigkeiten:

Grundauslastung: Wenn nur die Software-Komponenten gestartet werden, die unbedingt für den Betrieb der mobilen Plattform notwendig sind, ergibt sich bereits eine hohe Auslastung der CPU des Steuerrechners. Wie in [PB06] beschrieben, beträgt die Grundauslastung etwa 75 %. Durch Einsatz eines speziellen Eingebetteten Systems zur Anbindung der Lasermesssysteme und zur Messdatenvorverarbeitung konnte eine Reduzierung der Grundauslastung auf 60 % erreicht werden [BWZ07]. Eine hohe Grundauslastung führt dazu, dass nur noch wenig Rechenleistung für andere Aufgaben wie die Auswertung von Bilddaten zur Verfügung steht. Somit ist hier mit erhöhten Latenzen zu rechnen. Intelligente Kameras könnten durch ihre dedizierten Recheneinheiten zusätzliche Rechenleistung zur Bildverarbeitung bereitstellen und somit eine erhöhte Performance bei Bildverarbeitungsaufgaben gerade bei beschränkten Systemressourcen bewirken.

Steuerung echtzeitkritischer Komponenten: Als besonders problematisch stellt sich der Umstand dar, dass der Steuerrechner des Service-Roboters bei der Ansteuerung der Aktuatoren bestimmte Timing-Anforderungen einhalten muss. Werden diese nicht eingehalten, erfolgt seitens der Aktuatoren eventuell eine ungleichmäßig ausgeführte Bewegung oder eine Notabschaltung. Die Einhaltung der Timing-Anforderungen kann insbesondere dann problematisch sein, wenn das System vollständig mit anderen Aufgaben ausgelastet ist. Durch die Benutzung der Kamerasysteme ergibt sich oft eine vollständige Auslastung der CPU. Diese tritt immer dann auf, wenn die Verarbeitungszeit eines Bildes länger ist als das Intervall zwischen zwei eintreffenden Bildern, was bei bei einem großen Teil der typischen Bildverarbeitungsalgorithmen der Fall ist. Auch wenn prinzipiell ein echtzeitfähiger Kernel im Betriebssystem dafür sorgen sollte, dass die zeitkritischen Anwendungen priorisiert ausgeführt werden, zeigen die Praxiserfahrungen, dass ein stabiler Betrieb nur möglich ist, wenn der Steuerrechner nicht voll ausgelastet ist. Durch die Auslagerung der Bildverarbeitungsaufgaben auf intelligente Kameras kann bewirkt werden, dass der Steuerrechner nicht voll ausgelastet betrieben wird.

Anzahl verschiedener Schnittstellen: Aufgrund der Vielzahl an Komponenten auf einem Robotersystem kann es Probleme geben, wenn sie viele unterschiedliche Schnittstellen aufweisen. Die Ausbaumöglichkeiten des Steuerrechners sind aufgrund von Platzrestriktionen auf eine begrenzte Anzahl an PCI- und USB-Schnittstellen beschränkt. Beispielsweise musste bei TASER die (analoge) Handkamera zeitweise außer Betrieb gesetzt werden, da für die PCI-Videodigitalisierungs-Karte kein Steckplatz verfügbar war. Intelligente Kamerasysteme könnten diesbezüglich Abhilfe schaffen, da sie über eine gewöhnliche Ethernet-Schnittstelle kommunizieren,

die ohnehin bei fast alles PCs vorhanden ist. Über die Installation eines Ethernet-Switches können auch mehrere Komponenten über die Netzwerkschnittstellen kommunizieren. Beispielsweise ist der Service-Roboter PR2 [Cou10] von Willow Garage mit einem Gigabit-Ethernet-Switch ausgestattet, über den mehrere Komponenten verbunden sind (darunter auch Kamerasysteme).

Auslastung der Datenbusse: Insbesondere wenn, wie zuvor erläutert, mehrere verschiedene Komponenten über einen Datenbus (hier die Ethernet-Schnittstelle) kommunizieren, kann es zu Engpässen und infolgedessen zu Verzögerungen und Datenverlust kommen. Auch beim Betrieb der Firewire-Kameras auf dem Service-Roboter kommt es zu den hier beschriebenen Einschränkungen und es lassen sich nicht alle Kameras gleichzeitig bei voller Auflösung nutzen. Intelligente Sensoren bieten hierfür eine Lösung, weil die Sensordaten durch deren integrierte Recheneinheiten vorverarbeitet werden können und nur die Ergebnisse der Verarbeitung gesendet werden. Auf diese Art müssen dann deutlich weniger Daten übertragen werden.

Möglichkeit zur Verwendung anderer Rechnerarchitekturen: Durch die Verwendung von intelligenten Sensoren können innerhalb dieser Sensoren Recheneinheiten verwendet werden, die für den Aufgabentyp besonders geeignet sind. Diese Form der Anpassung ist bei Verarbeitung auf einem Standard-PC nicht gegeben. So ist beispielsweise eine intelligente Kamera angekündigt, die über Grafikprozessoren eine beschleunigte Verarbeitung von Bilddaten ermöglicht (vgl. 2.3.4).

Unter Berücksichtigung der beschriebenen Gegebenheiten und Aspekte wird in den nächsten Abschnitten das entwickelte Konzept zur Integration der intelligenten Kamera in einen Service-Roboter vorgestellt.

3.2. Implementierung der modularen Softwarearchitektur

Beim Einsatz eines intelligenten Kamerasystems auf einem mobilen Robotersystem ergeben sich vielfältige Anforderungen. Bei dem intelligenten Kameramodell, das in dieser Arbeit verwendet wird, handelt es sich um ein Linux-basiertes Computersystem, das außer einem Treiber für die Kamerahardware noch keine Bildverarbeitungsfunktionen implementiert hat. Hauptaufgabe der Steuersoftware ist es, die Bildverarbeitungsfunktionen der intelligenten Kamera für die Software-Komponenten des Service-Roboters bereitzustellen. Die Grundanforderung besteht darin, dass auf dem Kamerasystem Algorithmen ausgeführt werden können, die auf den Bilddaten arbeiten. Ziel dieser Operationen ist entweder eine Manipulation dieser Bilddaten oder die Berechnung von Bildinformationen aus den Bilddaten. Die Manipulation der Bilddaten kann auch erfolgen, um diese in weiteren Verarbeitungsschritten als Quelldaten heranzuziehen. Extrahierte Bildinformationen können in der Robotersteuerung eingesetzt werden (z.B. Greifen eines gefundenen Objekts).

3.2.1. Heterogenität der Bildverarbeitungsaufgaben

Der geplante Einsatz auf dem Service-Roboter TASER bedingt, dass sich die Aufgabenstellungen des gesamten Robotersystems und somit auch der intelligenten Kamera dynamisch ändern. Es muss aus diesem Grunde vorgesehen sein, dass die Bildverarbeitungsfunktionen zur Laufzeit verändert werden können. Ein Eingreifen des Nutzers zur Umkonfiguration des Kamerasystems soll nicht erfolgen.

3.2.2. Bildverarbeitung als Nacheinanderausführung von Operationen

In [SHB07] wird Bildverarbeitung als eine Aufgabe beschrieben, die auf verschiedenen Ebenen angeordnet werden kann. Auf der unteren Ebene liegt die optische Abbildung einer Szene auf ein digitales Bild. Auf den Bilddaten werden dann in der nächsten Ebene einfache Algorithmen zur Vorbereitung ausgeführt (z.B. Rauschunterdrückung, Farbabgleich), in einer folgenden Ebene werden Merkmale wie Kanten und Regionen erkannt. Auf diesen Merkmalen baut dann in einer hohen Ebene die Objekt- und Szenenerkennung auf.

Je nach Anwendungsszenario kann eine andere Kombination aus Verarbeitungsschritten notwendig werden. Die Möglichkeit der Hintereinanderausführung von Algorithmen würde im Hinblick auf die Ziele der Arbeit den Vorteil haben, dass sich der Implementierungsaufwand bei dem Hinzufügen neuer Verarbeitungsstrategien oft darauf beschränken würde, bereits implementierte Funktionen zu verketten, anstatt die Verarbeitung komplett neu zu implementieren.

3.2.3. Skalierbarkeit

Mit Skalierbarkeit ist hier die Fähigkeit der Kamerasoftware gemeint, die Aufgabe der Bildverarbeitung an die Leistungsfähigkeit der vorhandenen Hardware anzupassen. Hier ist erstrebenswert, dass bei Verfügbarkeit von zukünftigen leistungsfähigeren Kameras deren Potential ohne große Änderungen an der Software nutzbar ist. Ebenso wäre die Möglichkeit von Vorteil, einzelne Verarbeitungsschritte auf externe Systeme auszulagern, die bei Bedarf installiert werden können. Bei einer Vielzahl der Bildverarbeitungsalgorithmen kann die Bildauflösung zum Zwecke der Anpassung des Rechenaufwands verringert werden, was jedoch zu einem Verlust von Bildinformationen führt. Hier soll das System eine Möglichkeit bieten, verschiedene Kombinationen im Hinblick auf Laufzeit und Ergebnis der Bilderkennung zu untersuchen, um einen geeigneten Kompromiss zu finden.

3.2.4. Portierbarkeit

Der technologische Fortschritt führt zu immer neueren Kameragenerationen mit anderen Architekturen und Programmierparadigmen. So sind beispielsweise Kameras mit hardwarebasierter Vorverarbeitung der Bilddaten denkbar. Ein Teil der Bildverarbeitungsalgorithmen, die bei einem Kameramodell beispielsweise in Software ausgeführt werden müssen, könnten bei einem anderen Modell als hardwarebasierte Funktion implementiert sein. Für diesen Fall sollen sich die Funktionen durch einfache Umkonfiguration nutzen lassen. Der Aufwand bei der Portierung soll sich neben der Integration und Kapselung zusätzlicher Funktionen möglichst nur auf Adaptionen der Softwareteile zur Kameraansteuerung beschränken; bereits implementierte Algorithmen sollen wenn möglich ohne Änderung funktionsfähig bleiben.

3.2.5. Timing-Kontrolle

Mit Timing-Kontrolle ist hier gemeint, dass bei der Datenverarbeitung stets nachvollziehbar bleiben muss, wann die Bilddaten aufgenommen worden sind. Werden extrahierte Bildinformationen in der Robotersteuerung berücksichtigt, muss sichergestellt sein, dass diese nicht mittlerweile veraltet sind. Dies kann dann der Fall sein, wenn die Bildverarbeitung sehr aufwändig ist oder wenn durch das Betriebssystem oder andere Tasks bedingt die Ausführung der Bildverarbeitung verzögert wird. So muss die Timing-Kontrolle zum einen den permanenten Datenfluss kontrollieren, zum anderen eine Performancemessung der implementierten Algorithmen ermöglichen.

3.2.6. Leistungsfähigkeit

Bildverarbeitungsaufgaben gehören zu den rechenintensivsten Aufgaben im Bereich der Computertechnik. Um das volle Potential der Kamerasysteme auszunutzen, kann nur nativer Code verwendet werden, also solcher, der von einem Compiler für die entsprechende Plattform übersetzt und optimiert wurde. Darüber hinaus muss die Software soweit möglich nach dem „Zero-Copy“-Prinzip arbeiten. Dieses Prinzip besagt, dass die Daten möglichst nicht unnötig im Speicher kopiert werden, sondern stattdessen Zeiger auf den Speicherbereich weitergegeben werden. Insbesondere bei der sequenziellen Ausführung von Operationen stellt sich dieses Problem.

3.2.7. Konsequenzen für die Implementierung

Die aufgezählten Anforderungen zeigen, dass für die gegebene Aufgabenstellung eine Softwarearchitektur von Vorteil ist, bei der die einzelnen Verarbeitungsschritte modular jeweils als eine Einheit ausgelegt sind. Diese werden dann zu einer Kette von Funktionen verknüpft, um die Bildverarbeitungsaufgabe zu erfüllen. Häufig benutzte Funktionen

brauchen so nur einmal implementiert zu werden und können in verschiedenem Kontext wiederverwendet werden. Ebenfalls ergibt sich als Anforderung, dass die Verarbeitungsschritte netzwerktransparent ausführbar sein sollten, d.h., dass es keine Rolle spielt, auf welchem System die Funktion ausgeführt wird.

Anstatt die geforderte Funktionalität von Grund auf neu zu implementieren, soll in dieser Arbeit das GStreamer-Framework[TBW+08] verwendet werden. In diesem Framework sind bereits grundlegende Bestandteile wie die modularisierte Verarbeitung, die Plugin-basierte Erweiterbarkeit, die Unterstützung von Datenübertragung über Netzwerk-Verbindungen und viele Funktionen aus der Bildverarbeitung vorhanden.

3.2.8. GStreamer-Framework

Das GStreamer-Framework ist ein Open-Source Multimedia-Framework, das seit 1999 entwickelt wird. Da seine Entwicklung bzw. Weiterentwicklung kein Bestandteil dieser Arbeit ist, soll hier nur auf die Hauptmerkmale eingegangen werden. Typischer Anwendungsbereich dieser Software ist die Wiedergabe von Multimedia-Inhalten. Gängige Operationen der Verarbeitung von Multimediadaten sind aus diesem Grund schon implementiert.

Hauptbestandteile des Frameworks sind die so genannten **Elemente**. Hierbei handelt es sich um Einheiten, die eine bestimmte Operation ausführen. Sie lassen sich in drei Hauptkategorien einordnen:

- Quell-Elemente (z.B. Kamera-Steuerung, Datei lesen, Netzwerkdaten empfangen)
- Filter-Elemente (Format-Konvertierung, Decodierung, Verarbeitung)
- Ziel-Elemente (Bildausgabe, Datei schreiben, Netzwerkdaten versenden)

Zur Konfiguration hat jedes der Elemente verschiedene Parameter, die gesetzt und ausgelesen werden können, beispielsweise der Betriebsmodus einer Kamera, die Netzwerkadresse oder ein Dateiname. Elemente werden zur Laufzeit zu einer Kette von Elementen verknüpft. Hierbei spielen die so genannten **Pads** die zentrale Rolle. Über Sourcepads (SRC) werden von einem Element Daten innerhalb des GStreamer-Frameworks erzeugt, über Sinkpads (SINK) Daten entgegengenommen. Jedes Element hat abhängig vom Typ eine bestimmte Anzahl von Pads. Über die Pads werden Verbindungen zu den Pads anderer Elemente hergestellt. Dabei werden die Datenstromrichtung und der Datentyp festgelegt. Somit wird sichergestellt, dass nur kompatible Elemente verbunden werden. Quell- und Ziel-Elemente besitzen üblicherweise jeweils ein Pad, Filter-Elemente ein Pad für die eingehenden und eines für die ausgehenden Daten. Eine Kette von Elementen wird innerhalb einer **Pipeline** verwaltet. Vorkehrungen in dieser Pipeline sorgen dafür, dass alle Elemente zeitgleich mit der Datenverarbeitung beginnen und sich synchron zu einem Zeitgeber verhalten. Ein typischer Aufbau einer Pipeline ist in Abbildung 3.3 skizziert.

Neben linearen Pipelines können auch nichtlineare realisiert werden, bei denen sich der Datenstrom verzweigt. Hierzu sind in GStreamer auch Elemente mit einem Eingangs-Pad und beliebig vielen Ausgangs-Pads implementiert, die den Datenstrom duplizieren.

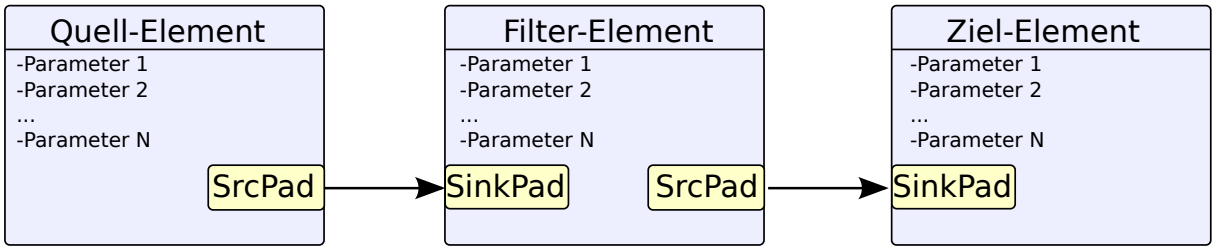


Abbildung 3.3.: Diese Abbildung zeigt eine typische GStreamer-Pipeline. Hier sind drei Elemente über ihre Pads verbunden. Jedes der Elemente kann über verschiedene Parameter konfiguriert werden.

Ebenfalls existieren Elemente mit mehreren Eingängen und einem Ausgang.

Alle gestellten Anforderungen an eine Softwarearchitektur für intelligente Kameras werden von GStreamer erfüllt. Die modulare Struktur kommt der Beschaffenheit der Bildverarbeitungsaufgaben entgegen und lässt sich um beliebige Funktionen erweitern, die dann in verschiedenen Konfigurationen nutzbar sind. Jedes der Elemente kann bei Notwendigkeit ausgetauscht werden. So besteht beispielsweise die Portierung der Anwendung auf eine andere Kamerageneration in einem Austausch des Quell-Plugins für die Kameraansteuerung.

Der Datenfluss zwischen den Elementen erfolgt über das Weiterleiten von Zeigern. Selbst bei der Verzweigung des Datenstromes kann in der Regel ohne zusätzliches Kopieren der Daten ausgekommen werden, da das zugrunde liegende GObject-Framework lediglich die Referenzen verwaltet und nur dann eine Kopie erstellt, wenn auf einen mehrfach referenzierten Datenbereich geschrieben werden soll.

Die Elemente sind darauf ausgelegt, mit variablen Bildauflösungen zu arbeiten und sich selbstständig darauf zu konfigurieren, was sogar zur Laufzeit erfolgen kann. Über schon implementierte Elemente kann die Auflösung bei Bedarf angepasst werden. Dadurch ist es möglich, bestimmte Algorithmen auf Bilddaten mit verringerter Auflösung anzuwenden. Auf diese Art kann eine Abstimmung zwischen Rechenaufwand und Qualität der extrahierten Informationen gewählt werden. Somit ist auch das Merkmal der Skalierbarkeit gegeben, da die Abstimmung an die verfügbare Rechenleistung der intelligenten Kamera angepasst werden kann.

3.2.9. Steuerung der Verarbeitungsketten

Je nach geplantem Einsatzzweck gibt es verschiedene Wege, eine GStreamer-Pipeline zu konstruieren. Die verschiedenen Wege sollen im Hinblick darauf beleuchtet werden, wie sie sich beim Test von Bildverarbeitungsalgorithmen und beim späteren Betrieb auf dem Service-Roboter verhalten. Hierbei soll besonders darauf eingegangen werden, inwiefern es möglich ist, auf verteilten Systemen Verarbeitungsfunktionen auszuführen. Außerdem

stellt sich die Frage, wie die verteilte Ausführung in der Praxis steuerbar ist. Bei Tests ist es möglich, die Verarbeitungspipelines manuell auf den verfügbaren Systemen zu starten. Für den Betrieb auf dem Service-Roboter soll jedoch eine praktikablere Lösung angestrebt werden, bei der die Verarbeitung automatisiert auf verschiedenen Systemen gestartet werden kann.

Pipelines können auf verschiedene Art erzeugt und gestartet werden:

- Benutzung das Kommandozeilentool *gst-launch*
- Benutzung das grafischen Frontends GStreamer Pipeline Editor
- Programmierung in einer Programmiersprache mit GStreamer-Bindings

Die beiden zuerst genannten Programme basieren ihrerseits auch auf den GStreamer-Programmbibliotheken. Die einzelnen Möglichkeiten werden im Folgenden erläutert; danach wird die implementierte Steuersoftware beschrieben.

Das Kommandozeilentool gst-launch

Das Kommandozeilentool *gst-launch* ist hauptsächlich für Testzwecke gedacht, jedoch nicht für die Erstellung von kompletten Anwendungen. Um auf einem entfernten System eine Bildverarbeitungspipeline zu starten, muss zunächst ein Terminal-Login auf dem entfernten System erfolgen, beispielsweise über SSH (verschlüsselt, vgl. [Ylo96]) oder Telnet (unverschlüsselt, vgl. [Sch12]). Auf diesen Systemen erfolgt dann der Aufruf des *gst-launch*-Befehls mit der entsprechenden Pipeline-Beschreibung. Werden mehrere interagierende Pipelines benötigt, muss dieser Vorgang auf allen beteiligten Systemen durchgeführt werden. Hierbei ist gegebenenfalls auf die korrekte Reihenfolge zu achten, da sich Abhängigkeiten zwischen den verschiedenen Pipelines auf verteilten Systemen ergeben können. Beispielsweise muss bei Benutzung der Elemente zur TCP-basierten Datenübertragung vor dem Start der Pipeline mit dem Client-Element die Pipeline mit dem entsprechenden Server-Element gestartet sein, zu dem das Client-Element eine Verbindung aufbauen soll. Somit ergibt sich ein hoher Konfigurationsaufwand, der dem letztlich angestrebten Ziel eines automatisierten Betriebs entgegensteht.

Ein weiteres Problem bei der Benutzung von *gst-launch* ist, dass die Parameter der einzelnen Elemente nur beim Start der Pipeline wählbar sind, jedoch nicht zur Laufzeit verändert werden können. Eine Veränderung wäre beispielsweise notwendig, wenn die Verschlusszeit der Kamera oder die Verarbeitungswege innerhalb der Pipeline geändert werden sollen.

Der GStreamer Pipeline Editor

Das grafische Frontend GStreamer Pipeline Editor erlaubt wie das Kommandozeilentool *gst-launch* die Konfiguration und die Ausführung von GStreamer-Pipelines. Die Konstruktion der Pipeline erfolgt nicht textbasiert, sondern über eine grafische Oberfläche, unter

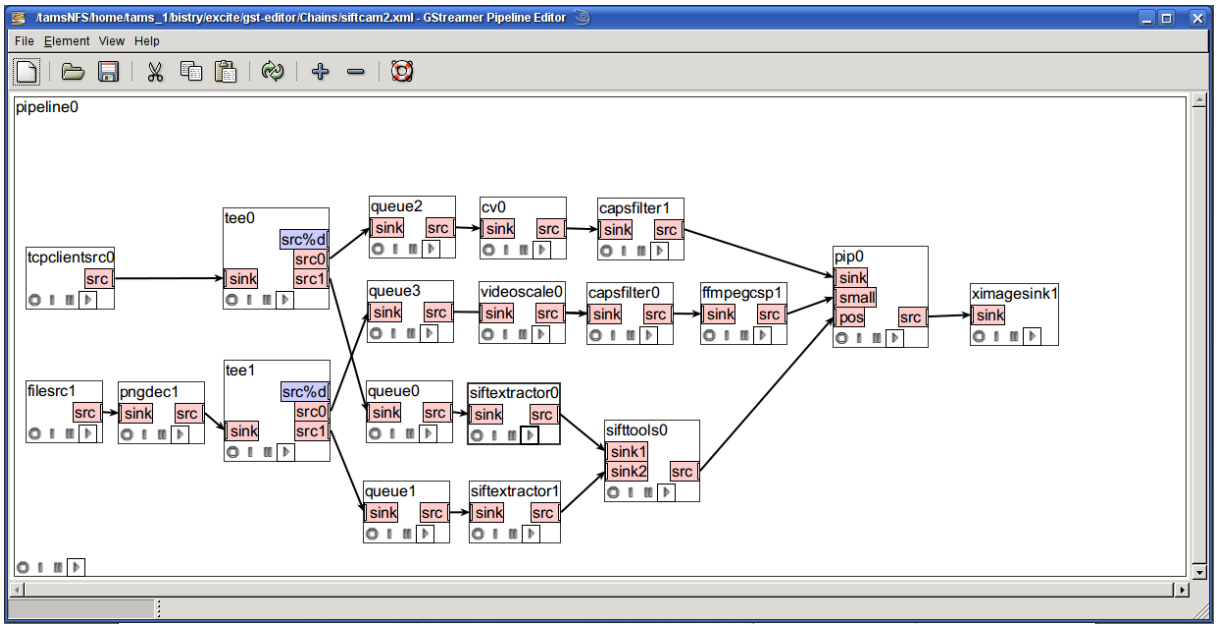


Abbildung 3.4.: Der GStreamer Pipeline Editor und eine konfigurierte Pipeline zur Anzeige eines Videodatenstroms unter Einblendung von übereinstimmenden Merkmalen bezüglich eines Referenzbildes. Mit dieser Pipeline wird beispielsweise das Bild in Abbildung 3.12 (rechts) auf Seite 77 erzeugt.

der die Elemente zu einer Pipeline konfiguriert werden können. Die Konfiguration der Pipeline im Editor kann auch als visuelle Programmierung betrachtet werden (vgl. Abschnitt 2.7).

Das Programm steht wie GStreamer unter einer Open-Source-Lizenz. Zu Beginn des Dissertationsvorhabens war für die verwendete GStreamer-Version 0.10 lediglich eine experimentelle Version des GStreamer Pipeline Editors verfügbar. Diese wurde im Rahmen dieser Arbeit weiterentwickelt, verbessert und veröffentlicht (siehe [gst13]).

Die Konfiguration einer Pipeline bei Benutzung des Editors soll im Folgenden erklärt werden. Ausgangspunkt ist zunächst eine Liste von verfügbaren Elementen. Aus dieser Liste können Elemente ausgewählt und zur Pipeline hinzugefügt werden. Die Elemente werden dann als verschiebbare Rechtecke dargestellt. Auf der linken Seite der Elemente werden die Eingangs-Pads und auf der rechten Seite die Ausgang-Pads visualisiert. Per Maus-Steuerung können nun Verbindungen zwischen Ausgangs-Pad eines Elements und Eingangs-Pad eines anderen Elements hergestellt werden. Zu jedem Element können über ein Kontext-Menü die vorhandenen Parameter angezeigt und verändert werden, teilweise auch zur Laufzeit der Pipeline. Ist die Pipeline konstruiert, kann diese direkt im Editor gestartet werden. Der GStreamer Pipeline Editor mit einer konfigurierten Pipeline ist in Abbildung 3.4 gezeigt.

Das Speichern und Laden von Pipelines ist ebenfalls möglich. Hierbei wird auch die Position der einzelnen Elemente in der Benutzungsoberfläche wiederhergestellt. Neben der Pipeline und ihrer Visualisierung werden auch die Parameter der einzelnen Elemente gespeichert. Die Dateien werden im XML-Format gespeichert. Solche Dateien lassen sich darüber hinaus mit dem Kommandozeilentool *gst-xmllaunch* starten.

Die Benutzung des Pipeline-Editors zur Konfiguration der intelligenten Kamera Basler eXcite bedarf einiger Vorkehrungen. Da die Kamera weder über Bildschirm noch über Eingabegeräte verfügt, muss die Bedienung der Oberfläche über sogenanntes X-Forwarding erfolgen. Hierbei wird das Programm (in diesem Fall der GStreamer Pipeline Editor) lokal auf der Kamera ausgeführt und die Grafikausgabe erfolgt über Netzwerk auf einem anderen Linux-PC. Tastatur- und Maus-Eingaben werden ebenfalls über das Netzwerk übertragen.

Programmierung der Pipeline

Die grundlegende Art, Anwendungen für GStreamer zu schreiben, besteht darin, eine Bildverarbeitungspipeline explizit zu programmieren. Über Funktionsaufrufe werden Elemente erzeugt, konfiguriert, zu einer Pipeline hinzugefügt und miteinander verknüpft. Schließlich kann die Pipeline gestartet werden. Würde die Entwicklung der Anwendungen im Rahmen dieser Arbeit ausschließlich nach diesem Prinzip erfolgen, müsste jede mögliche Systemkonfiguration in der Anwendung vorgesehen sein bzw. es müsste mehrere Anwendungen geben. Da jedoch die Konfiguration zur Laufzeit ein zentrales Ziel der Entwicklung ist, müssen noch zusätzliche Maßnahmen getroffen werden.

Anstatt die Elemente einzeln zu erzeugen, bietet die GStreamer-API auch die Möglichkeit, eine komplette Pipeline aus einem String zu erzeugen. Die Beschreibung der Pipeline mit allen Verbindungen erfolgt in einer festgelegten Syntax. Der große Vorteil hierbei ist, dass dieser String auch dynamisch zur Laufzeit festgelegt werden kann. Auf diese Art können Anwendungen entwickelt werden, die für beliebig viele Pipelines nutzbar sind und die Konfiguration beispielsweise aus einer Datei einlesen. Auf diesem Funktionsaufruf basieren auch die implementierten Anwendungen.

Die verteilte Steuerung

Im Folgenden werden die entwickelten Software-Komponenten behandelt, die es ermöglichen, von einer Instanz aus beliebige Pipelines auf verteilten Systemen zu erzeugen und zu kontrollieren. So ist es beispielsweise möglich, vom Steuer-PC des Roboters aus eine Bildverarbeitungspipeline auf der intelligenten Kamera zu starten. Dazu wird eine Software auf der intelligenten Kamera gestartet, die die Verbindungsanfragen verarbeitet und letztlich die Pipeline startet. Die Implementierung erlaubt, dass die Parameter der einzelnen Elemente direkt über die Netzwerkverbindung gesteuert werden können. Beispielsweise lassen sich die Aufnahmeeinstellungen des Kameratreibers verändern oder es kann der

Pfad verändert werden, auf dem die Bilddaten verarbeitet werden. So kann zur Laufzeit zwischen verschiedenen Bildverarbeitungsstrategien umgeschaltet werden. Bei Bedarf lässt sich die Pipeline auch wieder stoppen oder durch eine andere Pipeline ersetzen.

Für das Gesamtsystem gelten folgende Anforderungen:

- Start von GStreamer-Pipelines auf verschiedenen Systemen
- Pipelinebeschreibung im Syntax von *gst-launch* und GStreamer Pipeline Editor
- Möglichkeit, Pipelinebeschreibungen aus einer Textdatei einzulesen
- GUI[3] zur Steuerung der Parameter einzelner Elemente, auch auf entfernten Systemen
- integrierte Videoanzeige, die von einer lokalen GStreamer-Pipeline genutzt werden kann

Die oben genannten Funktionen werden über drei verschiedene Software-Komponenten realisiert. Es handelt sich um eine GUI, mit der die Pipelines gestartet und kontrolliert werden, einen Daemon, der auf jedem der entfernten Systeme gestartet wird, und das Element „Propexport“, das bei Bedarf innerhalb der GStreamer-Pipelines erzeugt wird. Die Interaktion der drei Komponenten ist in Abbildung 3.5 gezeigt.

Der Datenaustausch für Steuerkommandos erfolgt in der aktuellen Implementierung verbindungslos über UDP-Pakete. Durch Änderungen am Quellcode sind jedoch auch gesicherte Verbindungen möglich. Zunächst werden die verschiedenen Nachrichten-Typen erläutert:

- **Pipeline-Beschreibungen**: Diese werden von den „Propexport“-Elementen zu Daemon-Instanzen gesendet, die dann jeweils die entsprechende Pipeline erzeugen.
- **Steuerbefehle**: Diese werden von „Propexport“-Elementen zu Daemon-Instanzen gesendet und dienen dazu, die entfernten Pipelines von der lokalen Pipeline aus zu starten bzw. zu stoppen.
- **GUI-Beschreibung**: Jedes „Propexport“-Element generiert eine Beschreibung der User-Interface-Elemente der lokalen Pipeline und sendet diese an die vorherige Instanz, also entweder an die GUI-Komponente oder an eine weitere „Propexport“-Instanz. Aus den verschiedenen Beschreibungen wird die Gesamtbeschreibung erzeugt.
- **Parameter-Werte**: Dieser Datentyp wird gesendet, um einen bestimmten Parameter auf einen Wert zu setzen, sowohl von der GUI in Folge einer Benutzereingabe als auch vom „Propexport“-Element, wenn sich ein Wert aus anderen Gründen ändert. Das „Propexport“-Element leitet eine eintreffende Nachricht gegebenenfalls zu einem weiteren „Propexport“-Element weiter, wenn der zu steuernde Parameter zu einem Element der entfernten Pipeline - oder einer weiter entfernten Pipeline - gehört.

Der Programmablauf geht wie folgt vonstatten: Zunächst wird auf allen beteiligten entfernten Systemen - auch auf dem intelligenten Kamerasystem - jeweils ein Daemon-Prozess gestartet. Als Daemon wird die Programmkomponente bezeichnet, die nach ihrem Start

[3]Graphical Use Interface - grafische Programmoberfläche zur Benutzung

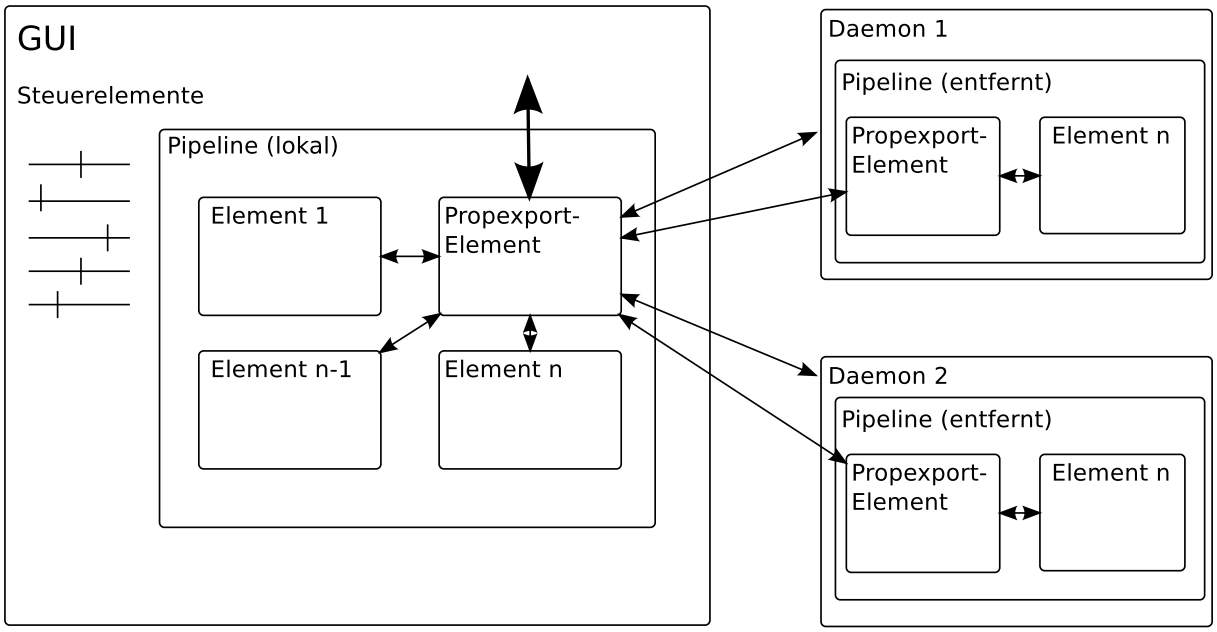


Abbildung 3.5.: Das Gesamtsystem bestehend aus GUI, Daemon und GStreamer-Pipelines mit „Propexport“-Elementen; in der Abbildung sind keine Datenpfade zwischen den GStreamer-Elementen eingezeichnet. Von einem „Propexport“-Element können beliebig viele Daemon-Instanzen angesprochen werden. Über das „Propexport“-Element werden zudem die Parameter der lokalen Elemente kontrolliert sowie Steuerkommandos zu entfernten Instanzen weitergereicht. Es können auch weiter verzweigte Netzwerke erstellt werden, wo z.B. das „Propexport“-Element in der Pipeline des „Daemon 1“ den „Daemon 2“ sowie das dort vorhandene „Propexport“-Element steuert.

auf Kommandos zur Konstruktion, zum Start und zum Stopp einer Bildverarbeitungspipeline wartet. Dies kann beispielsweise auch über Start-Skripte des Linux-Systems erfolgen. Weitere Interaktionen sind auf diesen Systemen nicht notwendig. Auf dem System zur Nutzerinteraktion wird nun das GUI-Programm gestartet. In diesem Programm kann eine Pipeline definiert oder aus einer Textdatei eingelesen werden. Die Pipeline wird in der gleichen Syntax beschrieben, wie dieses auch bei dem Kommandozeilentool *gst-launch* erfolgt. Um Steuerelemente in der GUI kontrollieren zu können und weitere entfernte Systeme zu integrieren, muss in die lokale Pipeline ein „Propexport"-Element eingebunden werden. Innerhalb der jeweiligen Pipeline reicht dieses Element die eingehenden Nutzdaten gegebenenfalls weiter. Als Konfigurationsparameter dieses Elements sind zwei Einträge von besonderer Bedeutung:

- Zum einen wird eine Liste der Parameter von Elementen der lokalen Pipeline festgelegt, die zur Laufzeit vom Benutzer gesteuert werden können. Hierzu werden durch Semikola getrennt der Name des Elements und der Name des Parameters angegeben, optional gefolgt durch einen eingeschränkten Wertebereich, wenn bei der Regelung nur dieser Bereich angefahren werden soll.
- Zum anderen werden in einer Liste Konfigurationsdateien eingegeben, die jeweils eine Pipelinebeschreibung und eine Netzwerkadresse enthalten. Jede dieser Pipelines soll dann auf dem entsprechenden entfernten System zur Ausführung kommen. Innerhalb der exportierten Pipeline kann bei Bedarf ein weiteres „Propexport"-Element vorhanden sein.

Beim Erzeugen des „Propexport"-Elements innerhalb der Pipeline wird an die entsprechenden Daemon-Instanzen die Pipeline-Beschreibung gesendet, bevor die lokale Pipeline gestartet wird. Zunächst werden die Pipelines auf den Zielsystemen gestartet. Somit wird sichergestellt, dass die Reihenfolge beim Start der Pipelines vorhersehbar ist und bestehende Abhängigkeiten zwischen den Pipelines eindeutig definiert werden können. Jedes „Propexport"-Element hat ein Vorgänger-Element und kann beliebig viele Nachfolger-Elemente des gleichen Typs haben. Von dem „Propexport"-Element, dass in der GStreamer-Pipeline innerhalb der GUI erzeugt wird, ist das Vorgänger-Element die GUI selbst. An das Vorgänger-Element werden alle lokal in der Pipeline steuerbaren Parameter sowie alle exportierten Parameter der Nachfolger-Elemente gesendet. Somit sind dann die exportierten Parameter aller Instanzen über die GUI steuerbar, da die Parameter-Beschreibungen über beliebig viele „Propexport"-Elemente letztendlich zu der GUI weitergeleitet werden. Im Falle einer Benutzereingabe werden die veränderten Parameter wieder über die „Propexport"-Elemente zu der Pipeline geleitet, in der sich das entsprechende Element mit dem Parameter befindet.

Beispiel zur verteilten Steuerung

Anhand eines einfachen Beispiels sollen nun die Funktionen der Steuersoftware aufgezeigt werden. Bei diesem Test soll der Videodatenstrom der intelligenten Kamera auf einem

Desktop-PC angezeigt werden. Hierbei sollen bestimmte Kameraparameter zur Laufzeit steuerbar sein. Zunächst wird eine Netzwerkverbindung hergestellt und auf der intelligenten Kamera wird das Daemon-Programm gestartet. Innerhalb des GUI-Programmes auf dem Desktop-PC wird folgende Pipeline gestartet:

```
tcpclientsrc host=tams68 port=1066 protocol=gdp ! propexport
   conffiles=baslercontrol.txt ! timestamper t1=1 t2=1 !
   ffmpegcolorspace ! xvimagesink sync=0
```

Diese Pipeline bewirkt, dass das Bild der intelligenten Kamera lokal angezeigt wird. Hierzu wird über das Element „tcpclientsrc“ eine TCP-Verbindung zum Port 1066 des Rechners „tams68“ aufgebaut, die Bilddaten werden (unverändert) über die Elemente „Propexport“ und „Timestamper“ geleitet, danach erfolgt im Element „ffmpegcolorspace“ eine Anpassung des Farbraumes, damit dieser mit den Vorgaben des Elements „xvimagesink“ übereinstimmt. Das „Propexport“-Element erhält als Parameter den Namen einer Konfigurationsdatei mit folgendem Inhalt:

```
tams68:4444
baslersrc preferred-mode=0 shutter=300 timestamptype=1 name=src
   ! propexport host=BACKADDR exportdesc="src;shutter;src;gain;
   src;brightness" sendport=4445 rcvport=5000 ! tcpserversink
   buffers-max=10 buffers-soft-max=2 recover-policy=1 port=1066
   protocol=gdp sync=0
```

Hiermit wird das „Propexport“-Element angewiesen, zum Port 4444 des Systems „tams68“ (Default-Port des Daemons) die Beschreibung einer Pipeline bestehend aus drei Elementen zu senden: Die Bilddaten werden von dem Element „Baslersrc“ erzeugt, das auf den Kameratreiber zugreift und die Bildaufnahme startet. Das „Propexport“-Element hat hierbei die Funktion, die Parameter „Shutter“[4], „Gain“ und „Brightness“ steuerbar zu machen. Die angegebenen Netzwerkparameter dienen zur Konfiguration der Netzwerkkommunikation des Elements. Das Element „tcpserversink“ erzeugt einen TCP-Server, der die Bilddaten bereitstellt. Zur Flusskontrolle wird in diesem Beispiel das GStreamer-eigene Protokoll GDP verwendet.

Die bei einer ähnlichen Konfiguration erzeugten Bedienelemente sind in Abbildung 3.6 gezeigt.

[4]Verschlusszeit einer Kamera, der Begriff wird verwendet, auch wenn viele digitale Kameras keinen mechanischen Verschluss aufweisen

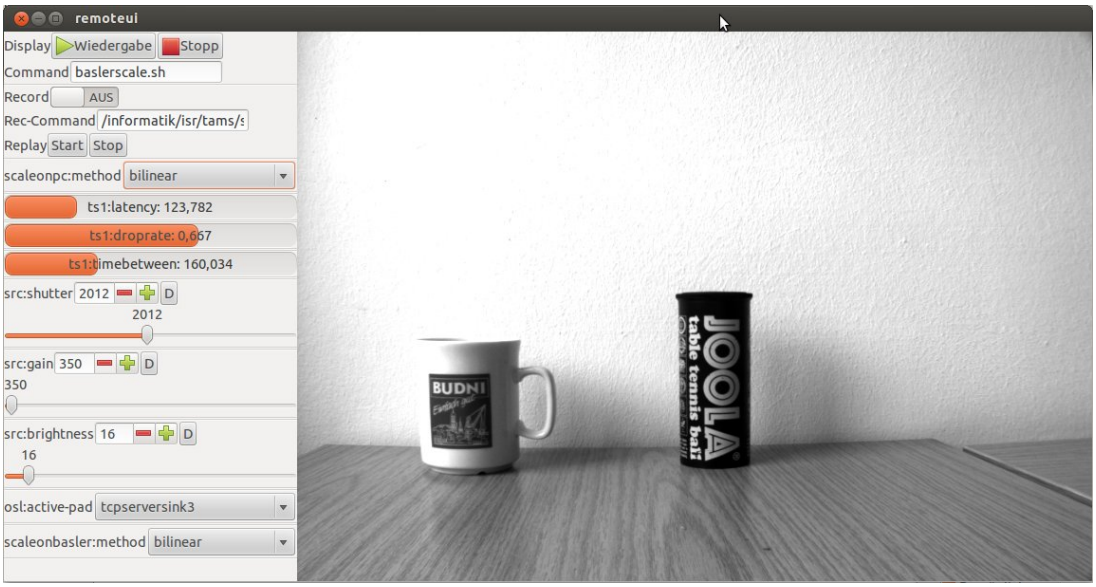


Abbildung 3.6.: Diese Abbildung zeigt die GUI-Komponente der verteilten Steuersoftware. Die intelligente Kamera ist auf eine Tischszene gerichtet. Es werden in der GUI neben dem Live-Bild die dynamisch erzeugten Steuerelemente angezeigt, die teilweise auch die Elemente auf der Kamera steuern, z.B. zur Kontrolle von Shutter, Gain und Brightness. Eine Veränderung der Parameter bewirkt eine Änderung der Einstellung in Echtzeit.

Gründe für die Wahl der Programmstruktur

Die Struktur der Steuersoftware ergibt sich zum einen aus der Zeitfolge, in der die einzelnen Komponenten implementiert wurden, zum anderen aus den Anforderungen, die an die Implementierung gestellt werden. Zunächst wurde die Möglichkeit verfolgt, die Parameter einer Pipeline von einem beliebigen Rechner aus zu steuern. Diese Anforderung führte zur Implementierung der GUI-Software und des „Propexport"-Elements, die miteinander über UDP-Pakete kommunizieren. Als zusätzliche Funktion innerhalb der GUI-Software besteht die Möglichkeit, die Videoausgabe einer auf dem gleichen System ausgeführten Pipeline als in die Oberfläche integrierten Videostrom anzuzeigen. Als weitere Anforderung kam hinzu, dass von einer Pipeline aus weitere Pipelines gestartet werden können. Dies führte zu einer Weiterentwicklung des „Propexport"-Elements und zur Implementierung der Daemon-Komponente.

Im Hinblick auf die Anforderungen und Zielsetzungen erscheint die gewählte Aufteilung der Funktionen vorteilhaft. Ein Ziel besteht darin, die Softwarekomponenten möglichst vielfältig einsetzen zu können. So ist hier beispielsweise die Möglichkeit, auf entfernten Systemen Verarbeitungspipelines aufzusetzen, unabhängig von der GUI-Komponente implementiert. Die GUI-Komponente wird lediglich dazu verwendet, Parameter der Pipeline zur Laufzeit anzuzeigen und zu ändern. Eine Pipeline, die über ein „Propexport"-Element eine Pipelinebeschreibung an ein entferntes System versendet, kann auch mit *gst-launch* oder von einem beliebigen anderen Programm aus gestartet werden, das auf GStreamer basiert. Auf diese Art kann auch die in 3.4.1 gezeigte Einbindung der Kamera Basler eXcite in das Framework ROS über eine einfache Konfigurationszeile erfolgen.

Eine weitere Designentscheidung besteht darin, die Steuerung der Parameter beliebiger Elemente über das „Propexport"-Element auszuführen. Die Alternative bestünde darin, dass die Applikation, in der die Pipeline erstellt wird - in diesem Fall also die GUI und der Daemon - hierfür zuständig wäre. Dass hiervon Abstand genommen wurde, liegt ebenfalls darin begründet, dass die Softwarekomponenten möglichst vielfältig einsetzbar sein sollen. Durch die gewählte Implementierung besteht die Möglichkeit, auch auf anderen Systemen die Benutzungsschnittstelle anzuzeigen als auf dem System, auf dem die Pipeline zur Ausführung kommt. Denkbar wäre auch eine Steuerung über Tablet-PCs, die jedoch im Rahmen dieser Arbeit nicht implementiert wird.

3.2.10. Timing-Analyse

Innerhalb von GStreamer beschränken sich die vorgegebenen Timing-Funktionen darauf, Bild und Ton relativ zueinander zu synchronisieren, jedoch sind keine Informationen über die absoluten Zeitpunkte der Aufnahme der Daten verfügbar. Aus diesem Grund müssen an dieser Stelle Erweiterungen implementiert werden.

Voraussetzung für systemübergreifende Timing-Kontrolle ist stets, dass sämtliche verschiedenen Systemuhren synchron laufen. Dieses lässt sich durch die NTP-Zeitsynchronisation

erreichen, die einen kontinuierlichen Abgleich der Systemuhren vornimmt und versucht, die Gangabweichungen von vornherein zu kompensieren. Der Algorithmus, der der Zeitsynchronisation zugrunde liegt, ist in [Mar84] beschrieben. Die erreichbare relative Genauigkeit bezüglich zweier Systeme in einem lokalen Netzwerk wird in [OP96] mit 0,2 ms gemessen. Die NTP-Software, die auf die intelligente Kamera Basler eXcite portiert wurde, gibt für die erreichte Genauigkeit zur Laufzeit ebenfalls eine Gangabweichung in dieser Größenordnung aus. Im Vergleich zu einer typischen Belichtungszeit im Bereich von 20 ms kann dieser Wert als hinreichend genau bezeichnet werden. Auch die Ausführungszeiten der Bildverarbeitungsalgorithmen sind deutlich größer als diese Abweichung der Systemuhren.

Es sind zwei verschiedene Methoden implementiert, um das dynamische Verhalten von Pipelines zu analysieren und Performancemessungen durchzuführen. Die Methoden unterscheiden sich in ihrem Aufwand und Einsatzzweck. Durch Verwendung des GStreamer-Elements „Timestamper“ können die Latenz, die Bildwiederholrate und der Anteil verworfener Bilddaten an einer Stelle der Pipeline gemessen werden. Es besteht jedoch auch die Möglichkeit, mit der zuvor erklärten Steuersoftware den kompletten Ablauf der Verarbeitung zu protokollieren.

3.2.11. Analyse mit dem Element „Timestamper“

Das Element „Timestamper“ kann den Zeitstempel einer Dateneinheit setzen und auslesen. Diese Variable ist eine vorzeichenlose 64-bit Integer-Zahl und wird von GStreamer in der Einheit „ns“ interpretiert. Beim typischen Anwendungsfall von GStreamer, dem Abspielen einer Mediendatei, entspricht dieser Zeitstempel der aktuellen Abspielposition. Über diese Zeitstempel wird gesteuert, dass Videodaten genau synchron zum Ton angezeigt werden. Auch bei der Anzeige von Live-Quellen, wie beispielsweise einer Webcam, beginnen die Zeitstempel bei null und geben die Zeit an, die seit Pipeline-Start vergangen ist.

Für die Auswertung der Latenz der Bilddaten, wie sie im Rahmen dieser Arbeit notwendig ist, ist hingegen der NTP-Zeitstempel von Bedeutung. Insbesondere auf verteilten Systemen, auf denen die Pipelines zu verschiedenen Zeitpunkten gestartet werden, ist ein gemeinsamer zeitlicher Referenzwert notwendig. Aus diesem Grund wird bei vielen der im Rahmen dieser Arbeit durchgeführten Experimente der Wert mit dem aktuellen NTP-Zeitstempel überschrieben. Bei der Abfrage der NTP-Systemzeit vom Betriebssystem wird die vergangene Zeit seit dem 1.1.1900 wiedergegeben. Die Darstellung wird in die Anzahl der Nanosekunden seit diesem Zeitpunkt umgerechnet. Die 64 bit sind ausreichend, um diesen Wert zu speichern. In den Quell-Elementen, die im Rahmen dieser Arbeit implementiert werden, erfolgt das Setzen des NTP-Zeitstempels als Grundeinstellung. Um auch andere Quellelemente in Verbindung mit einer Zeitmessung verwenden zu können, besteht die Möglichkeit, über ein „Timestamper“-Element, welches direkt hinter das Quellelement platziert wird, den Zeitstempel auf NTP-Zeit zu setzen. Über ein zweites „Timestamper“-Element kann dann die Auswertung erfolgen. Bei der Auswertung wird der Zeitstempel der Dateneinheit mit der aktuellen Systemzeit verglichen.

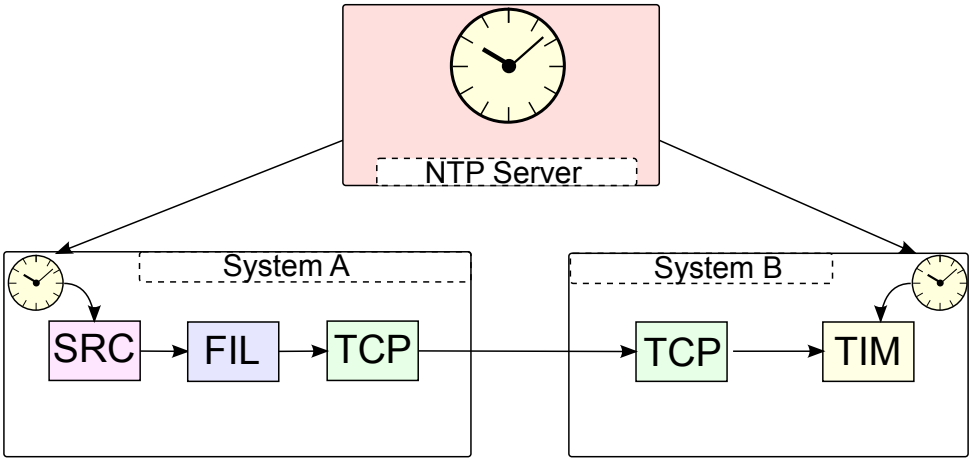


Abbildung 3.7.: An einem Beispiel soll die Timing-Analyse verdeutlicht werden. Die Systeme synchronisieren ihre Zeit mit einem NTP Zeitserver. Der Zeitstempel wird im Kamera-Quellelement (SRC) gesetzt und am Ende der Verarbeitung von einem Auswertungselement (TIM) mit der aktuellen Zeit verglichen. Die Daten werden hier von einem fiktiven Filter-Element(FIL) verarbeitet und über eine Netzwerkverbindung(TCP) übertragen.

Um die Ausführungszeit einer einzelnen Operation oder einer Folge von Operationen zu messen, können zwei „Timestamper“-Elemente auch an beliebige Stellen in der Pipeline platziert werden. Eines davon setzt den Zeitstempel, wenn die Daten weitergeleitet werden und die zu messende Zeitspanne beginnt. Das zweite Element führt die Auswertung durch, indem der Zeitstempel mit der aktuellen Systemzeit verglichen wird. Eine Visualisierung der Funktionsweise der Analyse mit dem „Timestamper“-Element ist in Abbildung 3.7 gezeigt.

3.2.12. Analyse über die Auswertung von Logdateien

Die Möglichkeit, das dynamische Verhalten der Pipeline analysieren zu können, ist ebenfalls in sämtliche im Rahmen dieser Arbeit implementierten Komponenten der Steuersoftware integriert. Dadurch werden die genauen zeitlichen Abläufe der Datenverarbeitung sichtbar. Es soll stets nachvollziehbar sein, wann welche Daten in welchem Element verarbeitet werden. Hierzu werden zur Laufzeit Informationen zum Datenfluss zwischen allen Elementen aufgezeichnet, im Einzelnen bestehen diese aus:

- Name von Quellelement und Zielelement
- Zeitstempel des Buffers[5] (Zeitpunkt der Aufnahme)
- Fortlaufende Nummer des Buffers (Offset)

[5]Erläuterung des Begriffs Buffer: Speicherbereich, meist ist gemeint, dass dieser eine bestimmte Größe hat und zusammenhängende Informationen enthält, also etwa ein Bild

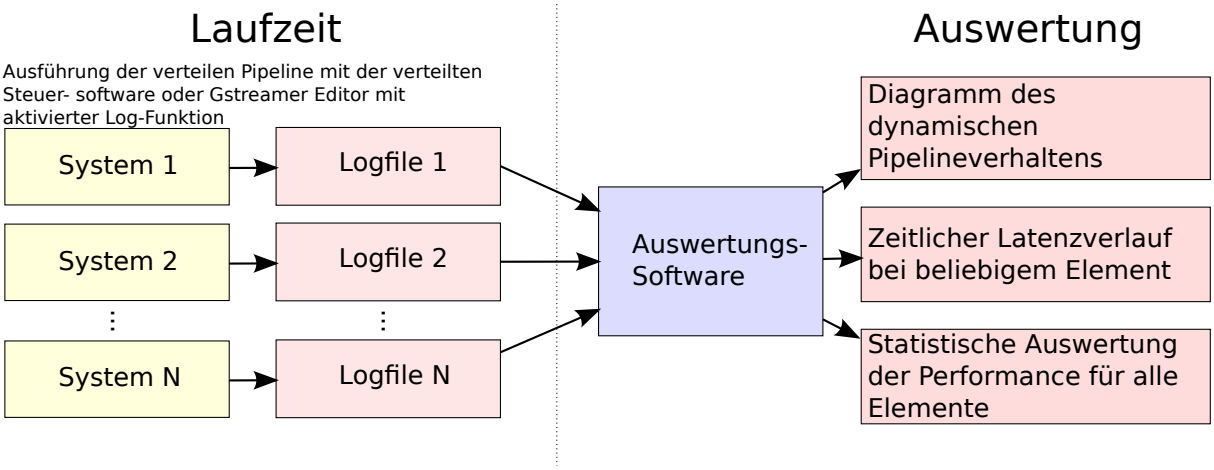


Abbildung 3.8.: Diese Abbildung zeigt die Erzeugung und Auswertung der Logfiles. Die Ausführung wird für eine gewisse Zeitspanne auf allen beteiligten Systemen unter Aktivierung der Log-Funktion gestartet. Sämtliche Log-Dateien werden dann von der Auswertungssoftware eingelesen und korreliert. Die Erzeugung der Graphen erfolgt mit Tools wie Latex oder GnuPlot.

- Zeitstempel der gegenwärtigen Systemzeit
- Größe der Nutzdaten
- Prozess-ID (für evtl. zukünftige Erweiterungen)

Der gesamte Vorgang der Aufzeichnung und Auswertung der Logfiles erfolgt wie in Abbildung 3.8 beschrieben und wird im Folgenden näher ausgeführt.

Auswertungsvorgang

Im ersten Schritt werden alle für die Auswertung relevanten Logfiles geöffnet. Die Einträge der Dateien werden in das programminterne Format eingelesen und die verschiedenen Listen von Log-Einträgen werden zu einer chronologisch geordneten Liste zusammengefügt. Anschließend wird diese erzeugte Liste durchgearbeitet und dabei eine Liste von Elementen und deren Relation zueinander gebildet.

Bei der Erzeugung dieser Liste der Elemente werden die TCP-Elemente gesondert betrachtet. Über diese Elemente werden die Verarbeitungsketten auf verschiedenen Systemen miteinander verknüpft. Angestrebt wird hier, die Verzögerung durch die Datenübertragung zwischen den Systemen ebenfalls messen zu können. Im Rahmen der Implementierungen sollen jeweils mehrere miteinander verbundene TCP-Elemente als ein einziges Element betrachtet werden. Dieses ist anhand von Grafik 3.9 gezeigt. Die Verarbeitungszeit für dieses fiktive Element ist die Zeit, die für die Datenübertragung benötigt wird.

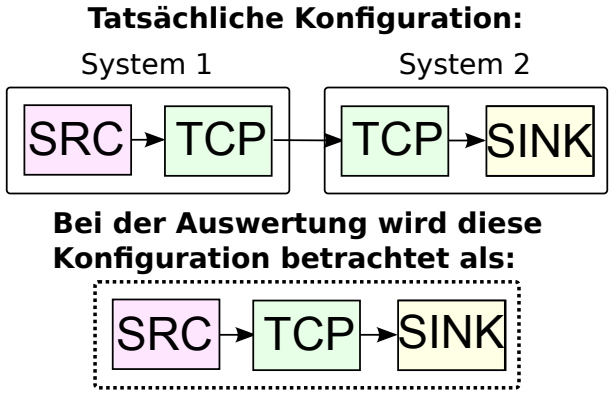


Abbildung 3.9.: Auswertung von Netzwerkverbindungen: Die beiden Elemente zur TCP-Datenübertragung werden in der Auswertungssoftware als ein Element betrachtet. Auf diese Art kann die Verzögerung durch die Netzwerkübertragung besser verdeutlicht werden.

Im Evaluationsablauf wird die Liste der Einträge chronologisch nach dem Zeitstempel der Einträge durchgegangen. Für jeden Eintrag wird folgendermaßen verfahren:
Das Quellelement des Eintrages ist das Element, das in diesem Schritt evaluiert wird. Es wird rückwärts nach dem Ereignis gesucht, das die Verarbeitung dieses zu evaluierenden Elements ausgelöst hat. Der zu suchende Eintrag ist dadurch bestimmt, dass der Buffer mit der gleichen Offset-Nummer und dem gleichen Zeitstempel an das hier evaluierte Element übergeben wird. Es wird nun die Differenz der beiden Zeitpunkte berechnet und dieser Wert abgespeichert.
Für den Fall, dass entsprechendes Element mehrere Sink-Pads hat, wird in dieser Implementierung als Startzeitpunkt für die Evaluation das Datenpaket berücksichtigt, das als letztes eingetroffen ist. Damit wird o.B.d.A. angenommen, dass ein Element mit mehreren Sink-Pads stets zunächst auf alle notwendigen Daten wartet, bis mit Berechnungen begonnen wird. Aus diesem Grund kann die Evaluation auch nicht sinnvoll durch Vorwärtssuche implementiert werden: Hier würde die Wartezeit einzelner Datenpakete als Verarbeitungszeit mit angerechnet werden.
Ebenfalls muss der Konstellation Rechnung getragen werden, dass ein Element mehrere Source-Pads besitzt oder pro Eingangs-Buffer unterschiedlich viele Ausgangs-Buffer erzeugt. Wichtig ist hierbei, dass direkt hinter diesem Element „Queue“-Elemente platziert werden, um das Element von dem Rest der Pipeline zu entkoppeln, indem die Weiterverarbeitung jeweils in einem anderen Thread abläuft. Angenommen, ein Element hat zwei Ausgänge und würde pro Eingangs-Buffer auf jedem seiner Source-Pads jeweils einen Ausgangsbuffer erzeugen, würde sich ohne eingefügte „Queue“-Elemente folgender Ablauf ergeben: Zunächst würde an das betrachtete Element der Buffer übergeben. Dieses startet die Berechnungen und gibt nach Fertigstellung die Ergebnisse auf einem der beiden Source-Pads aus. Dann würde zunächst der dahinterliegende Teil der Verarbeitungsfunktionen ausgeführt werden und erst nach Abschluss dieser Teilkette könnte von dem Element der zweite Ausgang mit Daten versorgt werden. Im Fall der eingefügten „Queues“

dauert das Übergeben des Datenpakets nur kurze Zeit und es kann gleich darauf der zweite Ausgang mit Daten versorgt werden. Die Weiterverarbeitung der Daten läuft dann jeweils in einem eigenen Thread ab. Somit ergeben sich Messergebnisse, die der tatsächlichen Verarbeitungszeit entsprechen. Hierbei muss jedoch berücksichtigt werden, dass zwei Messungen pro Eingangsdatensatz erzeugt werden.

Grafische Darstellung

Oft ist es notwendig, den genauen zeitlichen Ablauf der Verarbeitung einer Folge von Bildern zu betrachten. Aus diesem Grund ist die Möglichkeit vorgesehen, für einen frei wählbaren Zeitbereich automatisiert ein Ablaufdiagramm erstellen zu lassen. Hierbei wird als Referenzzeit der Zeitstempel der ersten Dateneinheit in diesem Zeitfenster gewählt. Die erste Dateneinheit wird mit eins nummeriert, alle folgenden Dateneinheiten erhalten aufsteigende Nummern. Der Grund, warum der Datenfluss üblicherweise nicht zum Startzeitpunkt der Pipeline betrachtet wird, liegt darin, dass in dieser Phase viele Elemente noch zusätzliche Funktionen aufrufen. Dies würde das Ergebnis beeinflussen. Zur besseren Übersicht werden keine vor dem gewählten Zeitpunkt erzeugten Dateneinheiten dargestellt, auch wenn diese möglicherweise noch Elemente belegen und für Verzögerungen im gewählten Zeitfenster verantwortlich sein können.

Eine weitere Funktion zur grafischen Auswertung sieht vor, dass die Latenz der Bilddaten im zeitlichen Verlauf dargestellt wird. Dieses erfolgt auf die Art, dass an einem definierbaren Punkt der Pipeline die Zeitpunkte des Eintreffens der Bilddaten und das jeweilige Alter der Bilddaten ausgewertet werden. Es werden jedoch nicht nur die Eintreffzeitpunkte betrachtet, sondern auch die Zeitspanne zwischen dem Eintreffen zweier Dateneinheiten. Während dieses Zeitraums erhöht sich das Alter der aktuell verfügbaren Bilddaten linear. Daher ergeben sich in der grafischen Auswertung regelmäßig Sägezahnmuster.

3.2.13. Liste der implementierten Elemente

Im Folgenden werden die GStreamer-Elemente vorgestellt, die im Rahmen dieser Arbeit eine bedeutende Rolle spielen. Hierbei handelt es sich um bereits vorgegebene Elemente und solche, die im Rahmen dieser Arbeit neu implementiert oder signifikant modifiziert werden. Die bereits im GStreamer-Framework vorgegebenen Elemente dienen tendenziell eher der klassischen Videodatenverarbeitung, wie sie in Medienwiedergabeprogrammen und Audio- und Video-Kommunikationssoftware benötigt wird. Robotikspezifische Bildverarbeitungsalgorithmen können mit diesen Elementen allein nicht ausgeführt werden.

Die neu implementierten Elemente sind in den Tabellen 3.1 und 3.2 aufgeführt. Tabelle 3.3 gibt einen Überblick über bereits vorgegebene Elemente, die im Rahmen der Arbeit modifiziert und erweitert werden.

Element	Funktion
Adjuster	automatische Steuerung von Gain und Brightness der intelligenten Kamera
Baslersrc	Zugriff auf die Videodaten der intelligenten Kamera
Bridge	Einbindung von beliebigen Kommandozeilentools in die Verarbeitungskette
Clcolorspace	OpenCL basierte Farbraumkonvertierung YUV422 auf RGBA (ohne Beschreibung)
Clremap4	OpenCL basierte Erzeugung eines Panoramas aus 4 Einzelbildern (Erzeugen der Abbildungstabellen durch eine Software von G. Cheng, s. 4.7)
Clrgbafilter	OpenCL basierte Filterung von Bildern im RGBA-Format (z.B. Sobel), Beschreibung in 4.6
Clundistort	OpenCL basierte Korrektur der Linsenverzeichnung, Beschreibung in 4.6
Gstadder	summiert die Pixelwerte auf; - auf gefilterte Bilddaten angewendet - kann der Summand als Schärfekriterium angewendet werden; Regions-of-Interest können bevorzugt behandelt werden (teilweise erweitert durch J. Liebrecht, keine Beschreibung)
Gstcl	wendet beliebige OpenCL-Kernel auf monochrome Bilddaten an, z.B. Laplace, Beschreibung in 4.6
Gstcv	exportiert ausgewählte Funktionen des OpenCV-Frameworks: Canny-Filter [Can86]; Sobel-Filter; Laplace-Filter ; Farbraum-Konvertierung; Optischer Fluss über Differenzbildung, Horn-Schunck-Verfahren [HS94],Lucas-Kanade [LK81b]; Bildglättung über Gauß und Median; Haar-basierte Merkmalsdetektion, z.B. Gesichtslokalisation [VJ01]; Hough-basierte Kreisdetektion [Hou62]
Inotifysrc	ermittelt über das Inotify-System (Bestandteil des Linux-Kernels) neu erzeugte Dateien und liest diese ein, führt wahlweise Kommandos aus, nutzbar in Kombination mit Kommandozeilen-Elementen (Anwendungsbeispiel: Nutzung von Spiegelreflexkameras auf Robotern)
Multivis	Element zur Einblendung verschiedener extrahierter Bildinformationen in das Video

Tabelle 3.1.: Liste der im Rahmen dieser Arbeit entwickelten GStreamer-Elemente Teil 1

Element	Funktion
Pip	Element zur Anzeige des Videos unter Einblendung beliebig vieler Bildausschnitte in das Hauptvideo
Pipgen	Element zum Ausschneiden mehrerer Bildausschnitte aus dem Eingangsbild, versendet einzelne Ausschnitte nacheinander
Propexport	Element, mit dem sich mehrere Parameter einer Pipeline über Netzwerk-Sockets steuern lassen (bereits in 3.2.9 beschrieben)
Ptucontrol	Element, mit dem sich verschiedene Schwenk-Neige-Einheiten steuern lassen; Steuerung von Fokus und Zoom über ein Arduino-basiertes Interface (Interface und Steuerbefehle von J. Liebrecht, Unterstützung für die einzelnen PTU-Typen wurde von J. Liebrecht und Gang Cheng implementiert)
RGB2Gray	Berechnung eines Grauwertbildes, variable Gewichtung der Farbkanäle
ROSParam	ähnlich dem „Propexport“-Element, ermöglicht die Steuerung von Parametern einzelner Elemente über den ROS-Parameter-Server
ROSSiftfolder	ähnlich dem „Siftfolder“-Element, publiziert Ergebnisse im ROS-Framework
ROSSink	publiziert Bilddaten aus GStreamer im ROS-Framework
ROSSrc	Zugriff auf Bilddatenströme aus dem ROS-Framework
SHMSink	Interprozesskommunikation über Shared Memory, Sender
SHMSrc	Interprozesskommunikation über Shared Memory, Empfänger
Siftextractor	Berechnung des SIFT-Vektors aus den Eingangsdaten; verwendet eine modifizierte Version der OpenCV-basierten SIFT-Implementierung von [HF07]
Siftfolder	vergleicht den eingehenden SIFT-Vektor mit allen Bilddateien in einem Ordner
Sifttool	vergleicht zwei eingehende SIFT-Vektoren, errechnet wenn möglich die Transformation
Timestamper	Element zur Auswertung von Zeitstempeln der Dateneinheiten (bereits in 3.2.9 beschrieben)
Trigger	ermöglicht netzwerkübergreifende Flusskontrolle, verwirft Daten, wenn Verarbeitung vorheriger Daten noch nicht abgeschlossen ist
Videosplitter	teilt Eingabebild in mehrere Abschnitte, horizontal oder vertikal, variable Überlappung
Videojoin	fügt die einzelnen überlappenden Bildabschnitte wieder zu einem Gesamtbild zusammen

Tabelle 3.2.: Liste der im Rahmen dieser Arbeit entwickelten GStreamer-Elemente Teil 2

Element	Funktion
Multisync	Element zum Synchronisieren der Daten, die auf verschiedenen Wegen verarbeitet wurden. Dieses Element basiert auf dem „Multiqueue“-Element und wurde um verschiedene Synchronisationsmöglichkeiten erweitert.
gsttcp	Element zum Datentransfer über TCP-Verbindungen; das Element wurde erweitert, um zur Laufzeit Formatänderungen zuzulassen
multifile	schreibt eingehende Daten ins Dateisystem; wurde erweitert um die Möglichkeit, den Nutzer per Dialog-Fenster zur Namenseingabe aufzufordern
outputselector	Erweiterung des bestehenden Elements „outputselector“; verteilt die Daten abwechselnd auf die verschiedenen Verarbeitungspfade
ximagesink	Anzeige der Videodaten; trivialer Bugfix, bereits eingepflegt

Tabelle 3.3.: Liste der modifizierten GStreamer-Elemente

3.3. Elemente zur Bildverarbeitung

RGB2Gray (implementiert)

Das Element „RGB2Gray“ dient zur Erzeugung eines Grauwertbildes aus einem RGB-Farbbild. Diese Funktionalität wird ebenfalls durch das GStreamer-Element „Ffmpeg-colorspace“ bereitgestellt. Mit dem Element „RGB2Gray“ ist es zusätzlich möglich, die Gewichte zu modifizieren, mit denen die einzelnen Farbkanäle in den Grauwert einfließen. Es können auch negative Gewichte eingestellt werden. Hierdurch können bestimmte Farben herausgefiltert werden. Die Grundeinstellung ist analog zu der Festlegung im YUV-Standart (siehe [Poy12]):

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Das Element wird im Rahmen dieser Arbeit als Vorverarbeitung zur Hough-basierten Erkennung von runden einfarbigen Objekten ohne Textur eingesetzt. Die Einstellung der Gewichte ermöglicht hierbei, das Objekt optimal vom Hintergrund abzuheben. In der folgenden Tabelle sind exemplarisch einige Kombinationen von Hintergrund- und Objekt-Farben eingetragen und mögliche Gewichte eingetragen, die einen hohen Kontrast im Grauwertbild erzeugen.

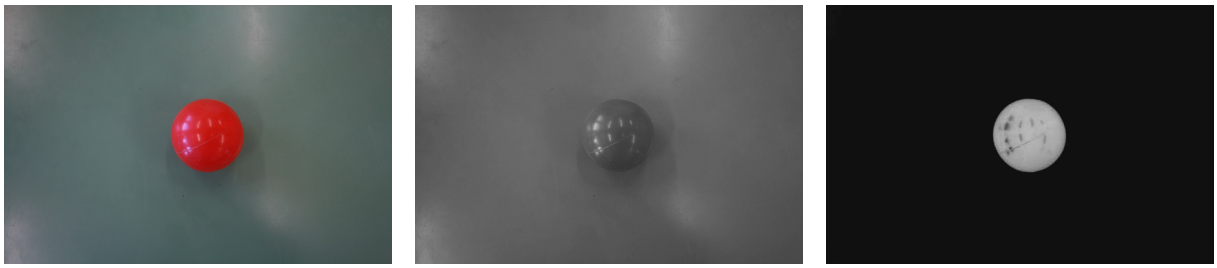


Abbildung 3.10.: Diese Bilder zeigen eine Tischszene, die mit einer Farbkamera aufgenommen ist (links). Zur Vorverarbeitung soll aus diesem Bild ein Grauwertbild erzeugt werden, z.B. zur anschließenden Kreisdetektion. Mit der Standardumrechnung ergeben sich nur geringe Kontraste (Mitte), mit angepassten Faktoren für die einzelnen Farbkanäle (grün -1, rot +1) hebt sich der Ball klar vom Hintergrund ab (rechts).

Hintergrund	Objekt	R	G	B	Erläuterung
weiß	rot	1	-1	-1	Es werden Bereiche hervorgehoben, die einen hohen Rot-Anteil, aber geringere Grün- und Blau-Anteile haben.
weiß	gelb	1	1	-2	Gelb unterscheidet sich in der RGB-Darstellung von weiß durch den fehlenden Blau-Anteil.
grün	gelb	1	-1	0	Gelb unterscheidet sich in der RGB-Darstellung von grün durch den zusätzlichen Rot-Anteil.
grau	gelb	1	1	-2	Es liegt die gleiche Situation vor wie bei weißem Hintergrund.

Wird bei der Berechnung der 8-bit Wertebereich überschnitten, wird der Wert auf 0 bzw. auf 255 gesetzt.

Abbildung 3.10 zeigt ein Beispiel einer Szene, bei der ein roter Ball auf einer grünlichen Tischplatte liegt. Es ist deutlich zu erkennen, dass dieser im Grauwertbild einen höheren Kontrast aufweist, wenn die Umwandlung mit dem hier vorgestellten Element RGB2Gray und angepassten Gewichten erfolgt.

OpenCV-basierte Funktionen (implementiert)

Das Element „Gstcv“ integriert mehrere Algorithmen aus der OpenCV-Bibliothek [Bra00]. Hierbei handelt es sich um eine frei verfügbare Software-Bibliothek, die von Intel entwickelt wurde. Mittlerweile ist die Version 2.4.3 der Bibliothek erschienen[6] und die Weiterentwicklung wird von Willow Garage vorangetrieben, einer US-amerikanischen Firma für Robotiktechnologie. OpenCV enthält eine große Anzahl von leistungsfähigen und hoch

[6]Stand: Februar 2013

optimierten Operationen zur Bildverarbeitung, die auf vielen aktuellen Forschungsergebnissen basieren. Daher werden im Rahmen dieser Arbeit die Algorithmen aus OpenCV innerhalb der GStreamer-Umgebung und somit auch auf der intelligenten Kamera nutzbar gemacht. Bei der Kapselung liegt der Schwerpunkt bei der Umwandlung der Datentypen. Es kann erreicht werden, dass sowohl das GStreamer-Framework als auch die OpenCV-Funktionen auf den gleichen Speicherbereichen arbeiten, so dass zusätzlicher Overhead durch Umkopieren vermieden wird.

Im Folgenden werden die basierend auf OpenCV implementierten Algorithmen vorgestellt:

- Farbraum-Konvertierung: Viele der verschiedenen Farbmodelle lassen sich mit Hilfe von OpenCV umwandeln. Eine solche Umwandlung ist dann notwendig, wenn die nachfolgende Operation Bilddaten in einem anderen Format benötigt. Da jedes GStreamer-Element Informationen über die unterstützten Datentypen bereitstellt, kann die Konfiguration des Konvertierungselements selbstständig erfolgen. Ebenfalls ist die Umwandlung eines Farbbildes in ein Grauwertbild möglich. Auch sensorspezifische Modelle wie das Bayer-Pattern werden unterstützt.
- Gauß-Glättung: Über einen Gauß-Kernel mit konfigurierbarer Größe kann eine Faltungsoperation ausgeführt werden, die eine Glättung des Bildes bewirkt. Hierdurch wird das Bildrauschen verringert, jedoch gehen auch Bilddetails verloren. Nachfolgende Algorithmen können unter Umständen bessere Ergebnisse erzielen, wenn Bildstörungen zuvor herausgefiltert werden.
- Optischer Fluss: Die Detektion des Optischen Flusses dient der Bewegungserkennung. Zwei aufeinanderfolgende Bilder werden verglichen und die Bewegung jedes einzelnen Pixels wird abgeschätzt. Hierzu sind in OpenCV das Verfahren von Horn & Schunck [HS81] und das von Lucas & Kanade [LK81a] implementiert. Die Abschätzung der Bewegung kann für ein Robotersystem zum Tracking von Gegenständen, zur Abschätzung der Eigenbewegung oder auch zur Aufmerksamkeitssteuerung benutzt werden. Im Surveillance-Bereich[7] ergeben sich analog viele Anwendungsgebiete, beispielsweise die Kontrolle der Bewegungsrichtung. Von dem „Gstcv“-Element werden zusätzlich die Koordinaten von Regionen ausgegeben, in denen Objektbewegungen auftreten.
- Hough-basierte Kreiserkennung: Über die Hough-Transformation [Hou62] können Objekte und Bildelemente anhand ihrer Form detektiert werden. Die Form der Elemente muss über bestimmte Parameter definierbar sein, beispielsweise bei einer Geraden über den Winkel und den minimalen Abstand zum Ursprung, bei einem Kreis über den Mittelpunkt sowie den Radius. Zunächst muss eine Kantendetektion erfolgen, beispielsweise über den Canny-Algorithmus. Jeder Bildpunkt, der als Kante klassifiziert wird, wird in einen mehrdimensionalen Modellraum transformiert. Dabei werden für den aktuellen Bildpunkt alle Kombinationen von Parametern der Form betrachtet, von der dieser Bildpunkt ein Teil sein könnte. Für diese Kombinationen von Parametern werden dann im Modellraum die Funktionswerte inkrementiert. Der Modellraum ist

[7] Der Begriff Surveillance bezeichnet die Überwachung durch Kameras.

somit als eine Art Histogramm aufzufassen. Im folgenden Schritt wird im Parameterraum nach Häufungen gesucht. Hierzu können gängige Methoden des Clusterings verwendet werden. Die Schwerpunkte dieser Häufungen entsprechen dann den detektierten Objekten.

- Objekt-Erkennung nach Viola-Jones [VJ01] mit Haar-Klassifikatoren: Die Verwendung von Haar-Klassifikatoren ist immer dann sinnvoll, wenn die Zugehörigkeit eines Bildelementes zu einer Kategorie bestimmt werden soll. Im Gegensatz zu dem später vorgestellten SIFT-Verfahren können auch Strukturen erkannt werden, die weniger signifikante Bildpunkte aufweisen. Die zu detektierenden Objekte können vielmehr anhand komplexer Merkmale erkannt werden. Somit unterscheidet sich dieses Verfahren von der oben erwähnten Hough-Transformation, mit der effizient nur einfache Formen erkannt werden können.
  Über Haar-Klassifikatoren es möglich, mehrere Instanzen eines Objektes zu detektieren. Auch verschiedene Skalierungen können detektiert werden. Im Rahmen dieser Arbeit wird diese Funktion zur Gesichtslokalisation eingesetzt. Es können in Abhängigkeit von dem verwendeten Klassifikator Gesichter in Frontalansicht, im Profil oder ein menschlicher Oberkörper detektiert werden. Um weitere Objekte detektieren zu können, müssen über einen manuell zu klassifizierenden Satz von Beispielbildern Klassifikatoren trainiert werden[8].
  Über ein zusätzliches Ausgangs-Pad werden Koordinaten-Informationen über die Regionen ausgegeben, in denen sich detektierte Gesichter befinden.

Mittlerweile wurden von anderer Seite ebenfalls frei verfügbare OpenCV-basierte GStreamer-Elemente veröffentlicht, die einen anderen Schwerpunkt und Ansatz verfolgen und in dieser Arbeit nicht benutzt werden.

Entzerrung des Bildes (implementiert)

Wichtige Schritte bei der Bildverarbeitung stellen die Kamerakalibrierung und die Korrektur der Abbildungsfehler dar. Viele Bildverarbeitungsalgorithmen in der Robotik setzen entzerrte Bilddaten voraus. Diese Funktionen werden von dem Element „Undistort“ bereitgestellt. Das Element benutzt die OpenCV-Bibliothek [Bra00] und teilweise Quellcode aus [BK08].

Neben der Korrektur der Verzeichnungen wird auch eine Funktion zur Kalibrierung der Kamera bereitgestellt. Beide Funktionen werden zur Laufzeit auf einem Bilddatenstrom angewendet. In Abhängigkeit von der Konfiguration ändert sich der Funktionsablauf des Elements. Werden bei der Konfiguration bereits in einer Datei gespeicherte Kameraparameter angegeben, erfolgt keine erneute Bestimmung, sondern nur die Laufzeit-Umrechnung der eingehenden Bilddaten.

[8] Die entsprechende Funktion ist nicht im Rahmen dieser Arbeit implementiert/getestet. Die in dieser Arbeit verwendeten Klassifikatoren sind Bestandteil der OpenCV-Bibliothek.

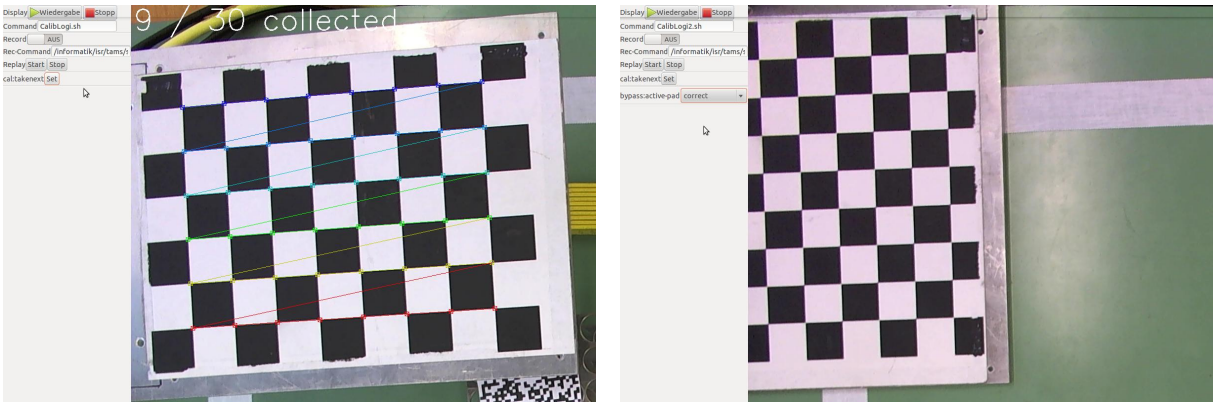


Abbildung 3.11.: Ablauf der Kamerakalibrierung: Links ist der Kalibrierungsvorgang mit dem hier entwickelten Element gezeigt, bei dem das Bild noch nicht rektifiziert ist. Rechts ist das rektifizierte Bild zu sehen. Anhand des Schachbrettmusters ist zu erkennen, dass die Linsenverzeichnung minimiert wird.

Die Kalibrierung erfolgt mit einem Schachbrettmuster, dessen Eckpunkte von dem Kalibrierungsalgorithmus erkannt werden. Basierend auf der Annahme, dass es sich bei dem Kalibrierungsmuster um ein unverzerrtes Rechteck handelt, werden die Koeffizienten der Verzeichnung über die Lage der Eckpunkte, die vom Algorithmus mit Sub-Pixel-Genauigkeit bestimmt werden, berechnet. Die Anzahl der Felder des Schachbrettmusters ist variabel und kann vor dem Start des Algorithmus festgelegt werden. Der Kalibrierungsvorgang mit dem hier entwickelten Element und der ebenfalls hier entwickelten GUI ist in Abbildung 3.11 gezeigt.

Benutzung anderer Kalibrierungsalgorithmen Der in diesem Element integrierte Algorithmus zur Bestimmung der Kameraparameter ist im Vergleich zu anderen Implementierungen recht einfach. Andere Implementierungen können noch zusätzliche Optimierungen enthalten und möglicherweise bessere Ergebnisse liefern. Durch die Möglichkeit, die Kalibrierungsdaten aus einer Datei einzulesen, können auch andere Verfahren zur Kalibrierung genutzt werden. Im Folgenden sollen zwei Beispiele genannt werden.

Eine Möglichkeit zur Kamerakalibrierung bietet das ROS-Framework. Hier lässt sich der Kalibrierungsprozess über eine grafische Oberfläche initiieren und überwachen. Intern verwendet die Kamerakalibrierung unter ROS ebenfalls die OpenCV-Bibliothek. Die Integration der intelligenten Kamera in das ROS-Framework wird in Abschnitt 3.4.1 gezeigt. Somit stehen auch die in ROS implementierten Kalibrierungsalgorithmen zur Verfügung. Die erzeugte Datei im YAML-Format enthält die notwendigen Parameter, die in die entsprechenden Konfigurationsdateien des hier vorgestellten Elements übertragen werden können.

Eine weitere Option zur Kalibrierung bietet die Matlab Camera Calibration Toolbox

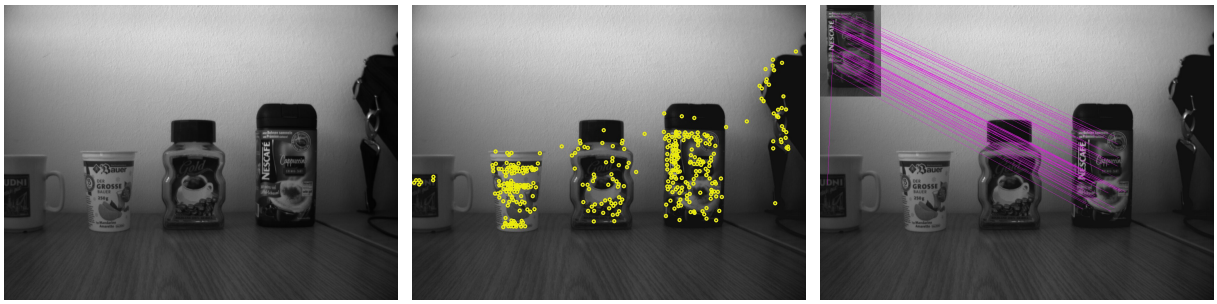


Abbildung 3.12.: Ablauf der SIFT-basierten Objekt-Detektion. Von einer Tischszene (links) werden vom Element „Siftextractor“ die Feature-Punkte bestimmt und beschrieben (Mitte). Durch einen Vergleich mit den Feature-Punkten eines bekannten Objekts werden übereinstimmende Punkte (rechts) detektiert und es kann eine Transformation berechnet werden.

[Bou04]. Diese ermöglicht die interaktive Kalibrierung der Kamera über eine GUI. Hier kann ein Satz Bilder geladen werden; die Detektion der Kalibrierungspunkte kann manuell überprüft und gegebenenfalls korrigiert werden.

Sift-basierte Elemente (implementiert)

Zur Objekterkennung sind mehrere Elemente basierend auf dem Scale-Invariant-Feature-Transform (SIFT) Algorithmus implementiert. Dem SIFT-Algorithmus kommt bei der Erkennung von lokalen Bildmerkmalen zunehmend Bedeutung zu. Vorgestellt wurde der Algorithmus 1999 von D. Lowe an der University of British Columbia [Low99]. Ziel des Verfahrens ist die Beschreibung von signifikanten Bildpunkten in einer Form, die invariant gegenüber Translation, Rotation und Skalierungsmaßstab ist. Ebenso können Beleuchtungsschwankungen und Bildrauschen teilweise kompensiert werden. Durch diese Eigenschaften eignet sich der Algorithmus insbesondere für den Einsatz auf mobilen Robotersystemen, da hier die Position zwischen Kamera und zu erkennendem Bildpunkt stets variiert. Die Implementierung bietet größtmögliche Flexibilität hinsichtlich der Anwendungsgebiete. Es ist zum einen möglich, in Live-Bildern einer Kamera nach bestimmten Objekten zu suchen. Zum anderen besteht die Möglichkeit, korrespondierende Bildpunkte zweier Live-Bilder zu finden. Auf diese Art können dreidimensionale Bildinformationen extrahiert werden. Zudem ist Flexibilität in der Hinsicht gegeben, dass die verschiedenen Arbeitsschritte auf verschiedenen Systemen ausgeführt werden können. Aus diesem Grund sind hier die Berechnung des SIFT-Feature-Vektors, der Vergleich zweier Vektoren und die Objektsuche in einer Datenbank in jeweils einem Element gekapselt. Der Vorgang der SIFT-basierten Objektdetektion ist in Abbildung 3.12 gezeigt.

Siftextractor Das Feature-Extraktions-Element nimmt ein Eingabebild entgegen, wählt mehrere signifikante, skalierungsinvariante Punkte und beschreibt diese entsprechend dem

SIFT-Algorithmus. Die Implementierung basiert auf einer modifizierten Version der OpenCV-basierten Implementierung von R. Hess [HF07]. Einige im Rahmen dieser Arbeit hinzugefügte Optimierungen, beispielsweise die Wiederverwendung von allozierten Speicherbereichen und der Verzicht auf unnötige Konvertierungen, verbessern die Performance und verringern den Speicherbedarf, insbesondere bei Live-Bildern. Da Feature-Extraktion und der Vergleich zweier Vektoren in unterschiedlichen Elementen stattfinden, spielt das Datenformat zur Repräsentation der Feature-Vektoren eine entscheidende Rolle. Hier sind verschiedene Methoden implementiert:

- Abbildung der internen Speicherrepräsentation (1140 Bytes pro Feature)
- Lowe-Repräsentation (vorgegeben, 361 Bytes pro Feature, Plaintext)
- modifizierte Lowe-Repräsentation (hier implementiert, 160 Bytes pro Feature)

Eine Methode zur Repräsentation der SIFT-Vektoren ist die direkte Verwendung der Speicherstruktur, wie sie innerhalb der SIFT-Bibliothek vorliegt. Da jedoch in dieser Struktur bereits eine Menge von zusätzlichen Informationen gespeichert ist, benötigt diese Variante viel Speicherplatz. Wenn jedoch sowohl Feature-Berechnung als auch Vergleich auf einem System erfolgen, hat diese Methode dennoch Vorteile, da keine Konvertierung der Formate erfolgen muss.

Die Implementierung von Lowe stellt die Werte als ASCII-Text dar, wobei die einzelnen Werte durch das „Space“-Zeichen voneinander getrennt sind. Die Fließkommazahlen werden normalisiert und auf 8 Bit gerundet. Dieser Schritt würde beim Feature-Vergleich sowieso erfolgen. Vorteile dieser Repräsentation sind, dass sie für Menschen einfach lesbar ist und unabhängig von der Byte-Reihenfolge des Computersystems ist. Durch Weglassen nicht benötigter Informationen ist die Darstellung recht kompakt, jedoch lässt sich durch direktes Speichern der normalisierten 8-bit Werte noch weiterer Speicherplatz sparen. Dieses wird in der modifizierten Form der Lowe-Repräsentation durchgeführt.

Die Bedeutung des Speicherplatzbedarfs soll an folgendem Beispiel erläutert werden: Es werden mehrere typische Kamera-Bilder während des Betriebs des Service-Roboters aufgenommen und zwischengespeichert. Die Feature-Berechnung eines Tischszenen-Bildes, aufgenommen mit der Basler eXcite mit 1,45 Megapixeln Auflösung, bringt im Mittel ungefähr 300 Features-Punkte hervor. Dieser Wert variiert mit dem Bildinhalt. Somit wird es möglich, die signifikanten Informationen eines Bildes mit 1,41 MB in etwa 50 kB zu speichern. Bei Verarbeitung auf der intelligenten Kamera kann durch Vorverarbeitung die Datenmenge stark reduziert werden, in diesem Beispiel um über 95 %. So kann also das Konzept der intelligenten Kamera voll ausgenutzt werden. Eine geringere Auslastung der Netzwerkverbindung und der weiteren Systeme ist die unmittelbare Folge.

Sifttool Das Element „Sifttool“ hat mehrere verschiedene Funktionen. Es dient zum einen zum Vergleich zweier Feature-Vektoren. Zum anderen kann das Element zwei Feature-Vektoren, die von benachbarten überlappenden Bildabschnitten stammen, zu einem Feature-Vektor zusammenfügen.

Bei dem Feature-Vergleich nimmt das Element an seinen zwei Eingangs-Pads Feature-Vektoren entgegen und berechnet korrespondierende Punkte. Damit dieses Element sowohl mit Feature-Vektoren eines Videodatenstroms als auch mit solchen eines Standbildes arbeiten kann, werden pro Eingangs-Pad die verfügbaren Daten zwischengespeichert. Pro Eingang kann konfiguriert werden, ob eintreffende Daten den Berechnungsprozess auslösen. Aus Performance-Gründen wird für den Vergleich ein multidimensionaler Baum der Eingangsdaten berechnet [Ben75]. Stammt einer der Feature-Vektoren von einem Standbild, so wird von diesem der Baum gebildet und muss somit nur einmal berechnet werden. Abhängig von dem Format der Eingangs-Daten muss als erster Schritt eine Umrechnung in das interne Datenformat erfolgen. Es sind mehrere Optionen implementiert, wie mit dem Ergebnis des Matching-Prozesses weiter verfahren werden kann. Zum einen kann direkt eine Liste von korrespondierenden Bildpunkten ausgegeben werden, die dann zur Visualisierung oder Weiterverwendung in einem der folgenden Elemente dient. Zum anderen kann auch direkt eine Transformation zwischen den beiden Bildern berechnet werden. Hierfür wird der RANSAC-Algorithmus genutzt [FB81]. Wenn also eines der Bilder ein bestimmtes Objekt darstellt, so kann die Position dieses Objektes in dem zweiten Bild ausgegeben werden.

Siftfolder Bei dem Element „Siftfolder“ handelt es sich um eine Modifikation des Elementes „Sifttool“, die es erlaubt, nacheinander und unabhängig voneinander nach mehreren Objekten zu suchen. Als Einsatzzweck ist hier die Szenenanalyse zu nennen; dabei können alle dem System bekannten Objekte überprüft und gefunden werden. Dieses Element besitzt nur ein Eingangs-Pad für die Feature-Vektoren des zu analysierenden Videostromes. Auch wenn prinzipiell die Möglichkeit bestehen würde, hier das Konzept der modularen Verarbeitung zu integrieren, wird im Hinblick auf eine einfache Nutzbarkeit dem Element direkt ein Pfad mit Bilddateien übergeben. Das Element führt dann beim Start der Verarbeitung folgende Schritte aus: Einlesen der Bilddaten, Decodierung und Berechnung der jeweiligen Feature-Vektoren. Es stehen zur Laufzeit dann in Abhängigkeit von der Anzahl der Bilddateien mehrere Feature-Vektoren zum Abgleich bereit. Jeder eingehende Feature-Vektor wird sukzessive mit der Liste von gespeicherten Vektoren abgeglichen. Erkannte Objekte werden dann einzeln ausgegeben. Die verwendete Dateistruktur gibt neben der zuvor beschrieben Transformationsmatrix noch den Dateinamen, die Auflösung dieser Datei sowie die Auflösung des Videodatenstroms zur Bilderkennung aus. Diese Informationen sind notwendig, um in einem späteren Schritt die korrekte Position des gefundenen Objektes im Video zu markieren. Hier ist eine zusätzliche Umrechnung erforderlich, wenn die Ausgabebildgröße nicht mit der Bildgröße übereinstimmt, die zur Erkennung verwendet wird. Dieses Vorgehen kann jedoch aus folgenden Gründen erforderlich sein:

- Herunterskalierung der Bildgröße zur Verringerung des Rechenaufwandes bei der Merkmalsextraktion
- Herunterskalierung der Ausgabe-Bildgröße zur Anpassung an das Ausgabegerät (z.B. wenn der Monitor nur eine geringere Auflösung unterstützt)

Auch in diesen Fällen können dann die detektierten Objekte an der korrekten Position angezeigt werden.

Das Element „Siftfolder“ kann zur Laufzeit die Liste von Feature-Vektoren aktualisieren. Bilddaten, die in dem angegebenen Ordner neu erzeugt werden, nimmt das Element automatisch in die Liste mit auf. Hierzu wird das Inotify-System des Linux-Kernels verwendet [Lov05]. Über dieses System kann überwacht werden, ob in einem bestimmten Verzeichnis Dateien verändert werden. Wird eine Bilddatei dem Verzeichnis hinzugefügt, erhält das Programm hiervon Kenntnis und startet die Aktualisierung der Liste. Diese Funktion ist insbesondere dann notwendig, wenn ein sogenanntes Online-Learning-System implementiert wird, bei dem der Benutzer Bildregionen markiert und den darin enthaltenen Gegenstand benennt.

Eine weitere Funktion des Elements sieht vor, dass nur ein Teil der im Verzeichnis gespeicherten Bilddaten berücksichtigt wird. Dadurch wird die Möglichkeit geschaffen, den Abgleich mit großen Objektdatenbanken auf mehrere Systeme auszulagern. Jedes dieser Systeme soll nun einen Teil der Vergleiche durchführen. Die gegenwärtige Implementierung sieht vor, dass in jedem „Siftfolder“-Element die Gesamtzahl der Elemente N und eine individuelle Nummer zwischen 0 und $N-1$ angegeben wird. Die Bilddaten werden fortlaufend nummeriert. Über den Restwert einer Division wird bestimmt, welches der Elemente für welches Objekt zuständig ist. Es muss sichergestellt sein, dass alle „Siftfolder“-Elemente auf die gleiche Datenbasis zugreifen können (netzwerkbasiertes Dateisystem). Die extrahierten Bildinformationen können dann auf verschiedene Art zusammengeführt werden. Zum einen können die Ergebnisse per UDP an eine Instanz des Elements „updsrc“ gesendet werden, zum anderen kann die Verbindung auch gesichert über TCP erfolgen und das Zusammenführen von dem Element „Input-Selector“ bewirkt werden, wenn dort alle Eingänge aktiviert werden. Untersuchungen zur Objekterkennung auf verteilten Systemen werden in Abschnitt 4.3 behandelt.

Kamera-Steuerung Basler eXcite (implementiert)

Um die Daten der intelligenten Kamera zur weiteren Verarbeitung bereitzustellen, werden in dem Element „Baslersrc“ die Treiber der Kamerahardware gekapselt. Die automatische Regelung der Kameraparameter ist im Element „Adjuster“ implementiert. Diese Elemente werden hier unter den Elementen zur Bildverarbeitung aufgeführt, da im Element „Adjuster“ basierend auf einer Analyse des Bildes eine Regelung vorgenommen wird.

Durch das Element „Baslersrc“ werden die Parameter der Aufnahmesteuerung über die gängigen Mechanismen des GStreamer-Frameworks nach außen hin zugänglich gemacht. So ist es möglich, den vollen Funktionsumfang der eXcite zu nutzen. Hierzu zählen auch die hardwarebasierte Farbraumkonvertierung, die Kontrolle über Belichtungszeit und die Einstellung der Sensorempfindlichkeit. Es können also je nach Anwendungsszenario Bilddaten in verschiedenen Farbmodellen ausgegeben werden, ohne dass eine Verwendung von zusätzlichen Elementen zur softwarebasierten Umrechnung zu erfolgen hat.

Die Implementierung dieses Elements sieht es vor, dass die gleichen Speicherbereiche vom Treiber und von den weiteren Elementen in der Verarbeitungskette genutzt werden. Hierzu wurde ein Mechanismus implementiert, der den gegenseitigen Ausschluss der Schreibzugriffe sicherstellt. Durch das Vermeiden zusätzlicher Kopiervorgänge ist die Effizienz der komponentenbasierten Implementierung nahezu identisch mit einer direkten Implementierung. Als Konfigurationsoption kann angegeben werden, wie viele Speicherbereiche alloziert werden und wie viele davon gleichzeitig in die Warteschlange zur Bildaufnahme gestellt werden. Die Zahl der zu allozierenden Speicherbereiche bestimmt maßgeblich den Speicherbedarf des Gesamtprogrammes, was insbesondere aufgrund des beschränkten Speicherausbaus der eXcite von Bedeutung ist. Die Anzahl der gleichzeitig in die Warteschlange gestellten Speicherbereiche beeinflusst unter bestimmten Betriebsbedingungen die Latenz und den Durchsatz. Hierzu muss das Gesamt-Scheduling einer GStreamer-Pipeline betrachtet werden. Unabhängig davon, ob alle nachfolgenden Verarbeitungsschritte abgeschlossen sind, werden vom Treiber der eXcite-Kamera Bilder aufgenommen, solange noch Speicherbereiche in der Warteschlange vorhanden sind. Dauert die Verarbeitung nun länger als das Intervall zwischen zwei Bildern, so werden veraltete Bilddaten ausgegeben. Auf diese Art kann jedoch der Gesamtdurchsatz bei temporären Verzögerungen aufrechterhalten werden. Ist dieses Verhalten nicht erwünscht, muss mit einem „Queue“-Element, das hinter dem hier betrachteten „Baslersrc“ Element platziert wird, das Verwerfen von Bilddaten bewirkt werden, wenn diese nicht rechtzeitig verarbeitet werden können.

Steuerung der Aufnahmeparameter Das Element „Adjuster“ dient zur Steuerung verschiedener Parameter der Kamera, die die Bildhelligkeit beeinflussen. Wie in Abschnitt 2.2.4 erläutert, haben mehrere Parameter einen Einfluss auf die Bildhelligkeit. Neben der Blendenöffnung des Objektivs spielen hier die Belichtungszeit (Shutter) und die Empfindlichkeit (Gain) des Bildsensors eine Rolle. Die beiden zuletzt genannten Parameter sind Kamera-intern regelbar. Das Element “Baslersrc“ exportiert diese als Parameter. Als Kamera, die in erster Linie für industrielle Bildverarbeitung konzipiert ist, bietet die Basler eXcite keine automatische Steuerung. Da sich bei Einsatz auf einem mobilen Roboter die Lichtverhältnisse der Szene dynamisch ändern, wird eine automatische Regelung dieser Parameter implementiert.

Als zu regelnde Größe soll anstatt konkreter Werte für Gain und Shutter ein virtueller Parameter der Bildhelligkeit eingeführt werden. Durch Kombination der Werte für Gain und Shutter lässt sich ein Wertebereich von 1 - 80921 für die Helligkeit definieren, wobei der Maximalwert das Produkt beider Verstärkungsfaktoren darstellt. Von der implementierten Regelung kann nun direkt dieser Parameter angesprochen werden, dessen Wert sich linear zu den Helligkeitswerten der einzelnen Pixel verhalten soll.

Die Helligkeitsregelung basiert auf einer Auswertung des Histogramms des Bildes oder eines Bildausschnittes (Region-of-Interest). Für den Fall eines 8-bit Grauwertbildes ergeben sich 255 Klassen, in die alle Pixel entsprechend ihrem Wert eingeordnet werden. Pixel mit

einem Wert von 0 werden als unterbelichtet eingestuft, Pixel mit einem Wert von 255 als überbelichtet.

In einer zweiten Funktion werden dann ausgehend von der gewünschten Gesamthelligkeit konkrete Werte für Shutter und Gain berechnet. Der Shutter wirkt sich linear auf die Bildhelligkeit aus und kann in einem Wertebereich von 1-4095 gewählt werden. Die Auswirkung des Gain-Parameters kann den Spezifikationen des Kamerasystems entnommen werden. Der zulässige Wertebereich geht von 350 bis 1023, das Verhalten ändert sich ab dem Wert 512. Für die Verstärkung in Dezibel gilt:

- für 350 bis 511:
  $dB(G) = 20 \cdot \log_{10}(\frac{658+G}{658-G} - 10298)$
- für 512 bis 1023:
  $dB(G) = 0,0354 \cdot G - 10298$

Somit gilt für die Verstärkungsfaktoren:

- für 350 bis 511:
  $a_g(G) = \frac{658+G}{658-G} \cdot 10^{-0,5149}$
- für 512 bis 1023:
  $a_g(G) = 10^{\frac{0,0354 \cdot G - 10,298}{20}}$

Hieraus ergibt sich ein maximaler Verstärkungsfaktor von 19,8.

Es kann die Präferenz gesetzt werden, welche dieser Regelgrößen zur Erhöhung der Bildhelligkeit stärker genutzt werden soll. Somit wird festgelegt, ob eher Bewegungsunschärfe (lange Belichtungszeit) oder Bildrauschen (hoher Gain-Wert) in Kauf genommen werden soll. Theoretisch bestehen mehrere Möglichkeiten, ausgehend von einem Helligkeitswert und einem Präferenz-Parameter eine Kombination von Shutter und Gain zu ermitteln. Die implementierte Methode sieht vor, dass die maximal erwünschte Verschlusszeit vorgegeben wird. Immer wenn mit dieser Verschlusszeit nicht die gewünschte Helligkeit erzielt werden kann, wird zusätzlich der Gain-Parameter erhöht. Mathematisch wird bei der Ermittlung der Parameter so vorgegangen, dass zunächst für die gegebene Bildhelligkeit und die Verschlusszeit der notwendige Verstärkungsfaktor a_G berechnet wird; dann wird unter Berücksichtigung der zwei Regelbereiche wie folgt verfahren:

- Ist $a_G < 1$, wird Gain auf 350 (Verstärkungsfaktor 1) gesetzt und die Verschlusszeit gleich der Bildhelligkeit gesetzt.
- Für $1 < a_G < 2,46$ gilt für den Gain-Wert: $G = 658,0 \cdot \frac{a_G - 0,31}{a_G + 0,31}$ und für die Verschlusszeit das vorgegebene Maximum.
- Für $2,46 < a_G < 19,8$ gilt für den Gain-Wert: $565,0 \cdot log10(a_G) + 291,4$ und für die Verschlusszeit das vorgegebene Maximum.
- Für $a_G > 19,8$ wird der Gain-Wert auf das Maximum von 1023 gesetzt und die Verschlusszeit über die Vorgabe erhöht, um die Bildhelligkeit zu erzielen.

Die maximal erwünschte Verschlusszeit kann zur Laufzeit verändert werden. Wenn starke Objektbewegungen oder auch Bewegungen des Robotersystems stattfinden, macht es Sinn,

die Verschlusszeit zu reduzieren, um Bewegungsunschärfe zu vermeiden. Die Information hierzu kann entweder direkt durch Interaktion mit der Robotersteuerung oder durch eine Bestimmung des Optischen Flusses aus den Bilddaten gewonnen werden.

3.4. Elemente zur Steuerung des Datenflusses

Capsfilter (vorgegeben)

Dieses vorgegebene Element nimmt selbst keine Manipulation am Datenstrom vor, es kontrolliert lediglich den Typ der Daten, der über dieses Element fließen darf. Das Element ist also hauptsächlich während der Phase der Format-Aushandlung von Bedeutung. Die Datentypen, die erlaubt sein sollen, und verschiedene Attribute werden vom Nutzer festgelegt. Auch wenn die Aushandlung des Formats zwischen den Elementen automatisch möglich ist, kann die manuelle Konfiguration in folgenden Fällen sinnvoll und notwendig sein:

- Es gibt mehrere Möglichkeiten, von denen aus Benutzer-Sicht eine zu bevorzugen ist.
- Ein zwischen zwei Elementen ausgehandeltes Format führt an späterer Stelle in der Pipeline zu Komplikationen. Dies ist insbesondere problematisch bei Verteilung der Elemente über das Netzwerk.
- Die Formataushandlung einiger Elemente kann fehlerbehaftet sein.

Das gewünschte Format wird in Textform bei den Eigenschaften des entsprechenden „Capsfilter“-Elements eingegeben. Eine gängige Konfiguration im Rahmen dieser Arbeit könnte beispielsweise *„video/x-raw-rgb, width=640, height=480“* lauten. Hiermit wird das Format auf „RGB“ festgesetzt und die Auflösung bestimmt. Alle weiteren, hiermit nicht festgelegten Attribute werden weiterhin automatisch abgestimmt. Wie bei dem normalen Verbinden von Elementen kann die Verbindung scheitern, wenn das geforderte Datenformat nicht bereitgestellt werden kann.

Queue (vorgegeben)

Das vorgegebene „Queue“-Element dient dazu, Datenpakete zwischenzuspeichern und weiterzuleiten. Die „Queue“ funktioniert nach dem FIFO-Prinzip, das besagt, dass das älteste Datenpaket zuerst an das nächste Element weitergegeben wird. Haupteinsatzzweck einer „Queue“ ist das Erzeugen eines Puffer-Speichers, um genügend Daten bereitzuhalten, damit es bei der Wiedergabe von Medien nicht zu Aussetzern kommt. Ebenso findet die „Queue“ dort Verwendung, wo von einer Live-Datenquelle, also z.B. einer Videokamera, mit einer festen Wiederholrate Daten erzeugt werden. Durch den Scheduler eines Betriebssystems und andere Tasks kann es dazu kommen, dass einzelne Datenpakete nicht in der vorgesehenen Zeit verarbeitet werden können, auch wenn die Rechenleistung prinzipiell

ausreichen würde. Für diesen Fall bietet sich die „Queue“ als Zwischenspeicher an, damit die Daten nicht verloren gehen.

Eine für die Anwendung im Rahmen dieser Arbeit entscheidende Fähigkeit des „Queue“-Elements ist die Fähigkeit zum Verwerfen von Datenpaketen. Dies spielt insbesondere bei solchen Szenarios eine Rolle, bei denen die Verarbeitung der Bilddaten länger dauert als das Intervall zwischen zwei aufgenommenen Bildern. In diesem Fall würde es zu einem stetigen Anwachsen des Füllstandes der „Queue“ kommen, was zum einen dazu führt, dass der Arbeitsspeicherbedarf des Programms kontinuierlich ansteigt und es zu Instabilitäten kommt, zum anderen dazu, dass die Latenz der Daten immer weiter erhöht wird. Da im Rahmen dieser Arbeit möglichst aktuelle Bildinformationen gewünscht sind, macht es Sinn, eine noch nicht weiter verarbeitete Information zu verwerfen, sobald eine neuere vorliegt. Um eine möglichst geringe Latenz zu erzielen, wird im Rahmen dieser Arbeit der maximale Füllstand der „Queue“ meist auf ein, maximal auf zwei Bilder begrenzt.

Neben der Zwischenspeicherung von Daten hat die Verwendung des „Queue“-Elements noch einen weiteren Effekt. Das Element sorgt auch dafür, dass die Weiterverarbeitung der Daten ab dem Folgeelement in einem anderen Thread erfolgt. So kann direkt nach Übergabe der Daten an die „Queue“ in dem vorherigen Teil der Kette mit der Verarbeitung oder Bereitstellung des nächsten Datenpakets begonnen werden. Durch die Aufteilung auf verschiedene Threads werden also mehrere Schritte der Pipeline parallelisiert. Dies führt zu großen Vorteilen, wenn ein Element auf eine Ressource wartet, beispielsweise beim Zugriff auf die Netzwerkschnittstelle. Während dieser Wartezeit kann in den anderen Threads weitergerechnet werden und somit die Rechenkapazität bestmöglich ausgenutzt werden. Implizit führt dies auch dazu, dass auf Multiprozessorsystemen mehrere Kerne benutzt werden und somit die Gesamtrechenleistung stark steigt.

Häufig werden in Pipelines „Queue“-Elemente direkt hinter Elemente geschaltet, die aus einem Eingangsdatenstrom mehrere Ausgangsdatenströme generieren. Somit wird erreicht, dass die Daten fast gleichzeitig an die dahinterliegenden Threads weitergegeben werden. Anderenfalls würde zunächst ein gesamter Teilstrang abgearbeitet werden und das entsprechende Element solange blockiert sein. Erst nach Fertigstellung würden die Daten an den nächsten Teilstrang weitergegeben werden.

Tee (vorgegeben)

Das vorgegebene „Tee“-Element dient dazu, einen Datenstrom zu duplizieren. Der Name ist wie beim „Tee“-Befehl unter Unix an den Begriff T-Stück angelehnt und deutet damit an, das von einer Leitung etwas abgezweigt wird. Ein solches Element besitzt ein Eingangs-Pad und beliebig viele Ausgabe-Pads. Im Rahmen dieser Arbeit wird dieses Element benutzt, wenn Daten im weiteren Verlauf der Pipeline mehrfach benötigt werden. Bei der Weiterleitung der Daten wird wenn möglich mit Zeigern gearbeitet, also keine Kopie der Daten erzeugt. Lediglich falls von einem Element an späterer Stelle in der Pipeline auf den gleichen Datenbereich geschrieben werden soll und die Verarbeitung der Daten an anderer

Stelle noch nicht abgeschlossen ist, muss eine Kopie der Daten erzeugt werden. Dies erfolgt weitgehend automatisch, es muss lediglich im Programmcode der Schreibzugriff auf den Datenbereich angekündigt werden.

Wie bereits oben erläutert, werden üblicherweise hinter dieses Element Elemente des Typs „Queue“ geschaltet, damit die Weiterverarbeitung parallel in verschiedenen Threads erfolgt.

Input-/Output-Selector (erweitert)

Bei diesen zwei vorgegebenen und teilweise im Rahmen dieser Arbeit erweiterten Elementen handelt es sich im weiteren Sinne um Multiplexer und Demultiplexer. Der Output-Selector besitzt ein Eingangs- und beliebig viele Ausgangs-Pads. Von letzteren ist zurzeit immer nur eines aktiv; über dieses werden Daten ohne Modifikation weitergeleitet. Durch Umkonfiguration zur Laufzeit können unterschiedliche Verarbeitungspfade gewählt werden. Beispielsweise ist es so möglich auszuwählen, ob ein bestimmter Verarbeitungsschritt auf der intelligenten Kamera oder auf dem dahintergeschalteten Steuerrechner des Service-Roboters stattfinden soll. Der entsprechende Parameter ist auch über die entwickelte GUI steuerbar.

Eine weitere Funktion, die im Rahmen dieser Arbeit entwickelt wird, ist die Möglichkeit, automatisch zwischen den verschiedenen Ausgängen zu wählen. Im einfachsten Fall werden hier die verfügbaren Ausgänge nacheinander durchgeschaltet. Dies kann benutzt werden, um die Daten parallel auf verschiedenen Pfaden zu verarbeiten. So können mehrere Kerne moderner Multicore-CPUs ausgenutzt werden oder die Daten können so auch an verschiedene Systeme im Netzwerk verteilt werden. Hierbei muss jedoch berücksichtigt werden, dass sich die Daten auf verschiedenen Pfaden „überholen“ können; ebenfalls könnten je nach Konfiguration Daten verloren gehen. Daher muss über ein zusätzliches Element sichergestellt werden, dass die Daten wieder in eine korrekte Reihenfolge geordnet werden. Dies kann mit dem später vorgestellten „Multisync“-Element erfolgen; je nach Aufgabenstellung kann es auch sinnvoll sein, alte Daten direkt zu verwerfen.

Zusätzlich ist eine Option eingebaut, die die Wahl des Ausgangs abhängig vom Füllstand eines dahinterliegenden „Queue“-Elements macht. Auf diese Art kann sogenanntes Load-Balancing implementiert werden. Es wird also immer der Pfad gewählt, auf dem die Ressourcen am geringsten ausgelastet sind. Somit ist die Wahrscheinlichkeit am höchsten, dass die Daten schnellstmöglich verarbeitet werden können. Hierzu sucht das Element in den Pipeline-Abschnitten hinter den Ausgängen nach dem nächsten „Queue“-Element und liest dessen aktuellen Füllstand aus. Dieses funktioniert auch mit angebundenen TCP-Elementen, die ebenfalls Informationen zu dem aktuellen Füllstand bereitstellen.

Bei den Experimenten zeigt sich, dass die Wahl des Verarbeitungspfades mit dem geringsten Füllstand der „Queue“-Elemente zum Erzielen einer geringen Latenz nicht ausreichend ist. Es lässt sich auf diese Art zwar sicherstellen, dass der maximale Durchsatz erzielt wird, zum Erreichen einer möglichst geringen Latenz muss jedoch die Zwischenspeicherung von

Daten möglichst komplett unterbunden werden. Aus diesem Grund ist ein weiterer Betriebsmodus implementiert, der genau den entgegengesetzten Ansatz verfolgt. Es wird gezielt versucht, das Verwerfen von Dateneinheiten auszulösen, die zwischengespeichert sind, deren Verarbeitung aber noch nicht begonnen hat. In der Praxis wird folgendermaßen vorgegangen: Angenommen, es wurde die Dateneinheit N-1 an den Ausgang I gesendet, dann wird bei der Verteilung des Datenpakets N geprüft, ob die Verarbeitung des Datenpakets N-1 schon begonnen hat. Dies ist dann der Fall, wenn der Füllstand des „Queue"-Elements verbunden mit Ausgang I gleich null ist.

- Ist der Füllstand gleich eins, kann angenommen werden, dass es zu einer Verzögerung gekommen ist. Daraufhin wird die Dateneinheit N ebenfalls an Ausgang I gesendet, um im verbundenen „Queue"-Element das Verwerfen der Dateneinheit N-1 hervorzurufen. Somit wird unterbunden, dass Rechenzeit dafür verwendet wird, veraltete Daten zu verarbeiten. Ebenfalls kann in der Zwischenzeit auf den anderen Verarbeitungszweigen der wahrscheinlich auch dort vorhandene Rückstand aufgeholt werden.
- Ist der Füllstand gleich null, besagt dies, dass bereits mit der Verarbeitung begonnen wurde. In diesem Regelfall werden die Daten an den nächsten Ausgang I+1 geleitet.

Die Auswirkungen dieses Verhaltens werden in Abschnitt 4.1.3 anhand praktischer Beispiele gezeigt.

Beim Input-Selector handelt es sich um das entsprechende Gegenstück mit beliebig vielen Sink-Pads und einem Source-Pad. Dieses Element leitet die Daten von einem der Sink-Pads an den Ausgang weiter, eventuell ankommende Daten an den anderen Eingängen werden verworfen. So kann beispielsweise zwischen verschiedenen Kameras umgeschaltet werden oder die Daten können so aus verschiedenen Verarbeitungswegen wieder zusammengeführt werden. Für letzteren Anwendungsfall kann auch konfiguriert werden, dass die Daten aller Pads entgegengenommen und weitergeleitet werden. Dies ist sinnvoll, weil die Umschaltung sowieso bei dem Output-Selector vorgenommen wird und bei zusätzlicher Umschaltung des Input-Selectors Daten verloren gehen könnten, die sich gerade in der Verarbeitung befinden.

Multiqueue / Multisync (erweitert)

Das vorgegebene „Multiqueue"-Element stellt die Funktion mehrerer „Queue"-Elemente bereit. Durch dieses Element kann eine beliebige Anzahl Datenströme geleitet werden. Je nach Anzahl der Datenströme wird die entsprechende Anzahl von Eingangs-Pads angefordert. Pro angefordertem Pad wird automatisch ein entsprechendes Ausgabe-Pad bereitgestellt. Im Gegensatz zu den einzelnen „Queue"-Elementen bietet das „Multiqueue"-Element nicht die Möglichkeit, bei hohem Füllstand Daten zu verwerfen. Die Weiterverarbeitung der Daten wird wie bei den einzelnen „Queue"-Elementen in jeweils einem eigenen Thread ausgeführt.

Dieses Element wurde im Rahmen dieser Arbeit durch die Möglichkeit erweitert, verschiedene Synchronisationsmechanismen zwischen den einzelnen Datenströmen durchzuführen.

Aus Kompatibilitätsgründen wird dieses modifizierte Element unter dem neuem Namen „Multisync“ integriert. In manchen Fällen kann es in einer Pipeline notwendig sein, die Weiterleitung von Daten zu verzögern oder durch Verwerfen zu unterbinden. Wenn ein nachfolgendes Element über mehrere Kanäle Eingangsdaten erhält, muss sichergestellt sein, dass auch stets zusammengehörige Daten zur Korrelation gebracht werden. Hierbei können einige Spezialfälle auftreten. Ein Teil der Daten wird möglicherweise auf einem Verarbeitungspfad verworfen, weil neuere Daten vorliegen und die Verarbeitungseinheiten voll ausgelastet sind. Es können auch unterschiedliche Verarbeitungsgeschwindigkeiten auftreten. In diesem Fall kann es auch notwendig sein, entsprechende Datenpakete, die über andere Kanäle eintreffen, zu verwerfen, um anschließend eine synchrone Verarbeitung zu ermöglichen. Im Einzelnen sind die folgenden Synchronisationsmodi integriert:

- Weiterleitung, wenn bei allen Kanälen Daten mit gleichem Index vorliegen
- Sortierung nach aufsteigendem Index

Diese zwei Modi gehen von grundlegend unterschiedlichen Annahmen aus. Während beim ersten Modus auf allen Kanälen Daten mit demselben Index vorliegen müssen, dient der Sortierungsmodus der Wiederherstellung einer Reihenfolge, wenn auf den einzelnen Kanälen Datenpakete mit unterschiedlichen Indizes eintreffen.

Im Folgenden sollen die beiden Modi detailliert vorgestellt werden, insbesondere soll ihre Relevanz in Bezug auf die in dieser Arbeit erfolgten Implementierungen hervorgehoben werden.

Weiterleitung bei gleichem Index Ein Beispiel für die Weiterleitung bei gleichem Index könnte ein aufgenommenes Bild sein, das auf mehreren Verarbeitungseinheiten analysiert wird. Die Ergebnisse der Bildanalyse sollen nun zusammengeführt werden.

Bei der Aufteilung des Datenstroms auf verschiedene Pfade ist die Konsistenz der Daten nicht mehr sichergestellt, da diese auf den verschiedenen Verarbeitungswegen verloren gehen können. Es werden von dem „Multisync“-Element nur dann Daten weitergeleitet, wenn auf allen Eingangs-Pads Datenpakete mit übereinstimmenden Offset-Nummern eingetroffen sind. Anderenfalls werden Daten verworfen. Dies erfolgt immer dann, wenn auf den entsprechenden Eingangs-Pads neuere Daten eintreffen als die erwarteten. Dieser Maßnahme liegt die Annahme zugrunde, dass sich die Datenpakete zwar auf verschiedenen Verarbeitungswegen überholen können, jedoch nicht innerhalb eines Verarbeitungswegs.

Als Option kann zusätzlich noch die Reihenfolge festgelegt werden, in der die Daten der einzelnen Pfade ausgegeben werden. Hierbei werden jedoch alle weiteren Sende-Threads solange blockiert, bis das Senden auf einem Ausgangs-Pad abgeschlossen ist. Sinnvoll ist diese Funktion dann, wenn ein folgendes Element mit mehreren Eingangs-Pads stets seine Verarbeitung beim Eintreffen der Daten auf einem dieser Pads startet. Durch die bekannte Reihenfolge kann so sichergestellt werden, dass auf den anderen Pads stets bereits die aktuellen Daten eingetroffen sind. Auf diese Art lassen sich mit geringem Aufwand Elemente mit mehreren Eingängen implementieren, ohne dass zusätzliche Maßnahmen zur

Synchronisation erforderlich sind. Dieses Prinzip wird bei dem im Rahmen dieser Arbeit implementierten Element „Videojoin“ genutzt.

Generell ist anzumerken, dass der Fall des Verwerfens von Daten auf nur auf einem Pfad ungünstig ist, da die Daten möglichst entweder auf allen Verarbeitungswegen verworfen oder komplett verarbeitet werden sollten, da ansonsten Rechenzeit verschwendet wird. Möglichkeiten, dieses zu vermeiden, werden bei der Vorstellung des Elements „Trigger“ beschrieben.

Sortierung nach aufsteigendem Index Die Sortierung kann dann angewendet werden, wenn zuvor jeweils komplette Datenpakete abwechselnd auf verschiedene Verarbeitungspfade geleitet worden sind und nun die Daten wieder in die richtige Reihenfolge gebracht werden sollen. Ein solches Setup ist nützlich, um die Rechenlast auf verschiedene Systeme oder auch auf verschiedene Recheneinheiten zu verteilen. Voraussetzung hierfür ist, dass die verwendeten Algorithmen die Datenpakete unabhängig voneinander bearbeiten können, sodass das Gesamtergebnis nicht beeinflusst wird. Auf diese Art lässt sich vor allem die Wiederholrate bei der Bildverarbeitung steigern und somit im Endeffekt auch das durchschnittliche Alter der Informationen senken. Da das „Multisync“-Element die Daten auch in diesem Fall über mehrere Ausgangs-Pads ausgibt, bedarf es der Möglichkeit, diese in einen einzelnen Datenpfad zusammenzuleiten. Dieses ist möglich durch die Nachschaltung eines „Input-Selector“-Elements, bei dem alle Eingänge aktiv sind.

Trigger (implementiert)

Das Element Trigger dient zur Steuerung des Datenflusses auch über verteilte Systeme hinweg. An späterer Stelle beschriebene Tests zeigen, dass eine parallele Verarbeitung nur dann Vorteile aufweist, wenn sichergestellt wird, dass Daten nicht unnötig lange zwischengespeichert werden. Für lokale Pipelines kann dieses wie bei dem Element „Output-Selector“ beschrieben durch Auswerten des Füllstandes der „Queue“-Elemente erfolgen. Auf entfernten Systemen ist dies nicht ohne weitere Maßnahmen möglich. Um auch in diesen Fällen eine geringe Latenz zu erzielen, wird das Element „Trigger“ eingesetzt. Von dem Element werden für jede Verarbeitungseinheit zwei Instanzen in der Pipeline (bei verteilten Systemen in den Pipelines) platziert. Die Elemente leiten die eingehenden Daten unverändert weiter. Die erste Instanz verwirft die eintreffenden Daten, wenn die Verarbeitung der vorherigen Daten noch nicht abgeschlossen ist. Die zweite Instanz sendet zur ersten Instanz eine Nachricht, wenn ein Datenpaket durchgeleitet wird. Die erste Instanz lässt daraufhin die nächste Dateneinheit passieren, anstatt sie zu verwerfen. Dieser Mechanismus ist ursprünglich für den Fall implementiert, dass ein Verarbeitungsschritt mit großem Rechenaufwand auf einem entfernten System ausgeführt wird. Wenn nicht jedes Bild verarbeitet werden kann, liegt es nahe, nur die Bilder zu übertragen, die auch verarbeitet werden können. Zusätzlich ist eine Funktion implementiert, mit der vorab geprüft werden kann, ob die Verarbeitung der vorherigen Daten abgeschlossen ist. Auf der Basis dieser Information kann das Element „Output-Selector“ einen freien Übertragungsweg

finden. Somit kann verhindert werden, dass Daten verworfen werden, obwohl ein anderer Verarbeitungspfad möglicherweise unausgelastet ist.

Es ergeben sich jedoch auch bei diesem Ansatz Nachteile. Jedes Mal, wenn die Verarbeitung auch nur geringfügig länger dauert als der Zeitraum bis zum Eintreffen des nächsten Bildes, wird das neu eintreffende verworfen. Auf lokale Pipelines angewendet, kann dieses Verhalten auch dahingehend interpretiert werden, dass das Verwerfen eines Bildes vorgezogen wird und nicht erst dann erfolgt, wenn die Verweilzeit eines Bildes in einem „Queue“-Element zu groß wird. Es werden auf diese Art aber auch Daten verworfen, wenn das System nur kurzfristig anderweitig ausgelastet ist und es zu einem Rückstand kommt, der wieder aufgeholt werden könnte. Dies eröffnet die Fragestellung, welche der prinzipiellen Möglichkeiten am sinnvollsten ist:

- Die Bilddaten immer weiterzuleiten und das Verwerfen bei einem Überlaufen der „Queue“ auszuführen,
- immer dann die Bilddaten zu verwerfen, wenn die Verarbeitung noch nicht abgeschlossen ist,
- eine Mischform, bei der noch eine gewisse Toleranzzeit eingeräumt wird, bevor die Daten verworfen werden.

Diese Frage soll im Rahmen dieser Arbeit nicht näher betrachtet werden. In der Regel wird das Verwerfen der Bilddaten in „Queue“-Elementen ausgeführt, also immer dann, wenn neuere Bilddaten vorliegen und es ältere Bilddaten gibt, deren Verarbeitung noch nicht begonnen hat.

Das direkte Verwerfen im „Trigger“-Element ist insbesondere dann unerwünscht, wenn die Verarbeitungszeit auf verteilten Systemen länger als die Übertragungszeit ist. Hier macht es Sinn, die Übertragung des nächsten Bildes schon zu starten, bevor die Verarbeitung des vorherigen Bildes abgeschlossen ist. Aus diesem Grund ist in dem „Trigger“-Element die Möglichkeit implementiert, das nächste Bild bereits vorzeitig zu übertragen. Es wird dabei basierend auf der tatsächlichen Verarbeitungszeit des vorletzten Bildes und der vom Benutzer gewählten Zeiteinstellung entschieden, ob bereits gesendet werden kann. Auf dem von den „Trigger“-Elementen überwachten Pfad können sich dann also zeitweise zwei Dateneinheiten befinden. Diese Funktion wird in Abschnitt 4.3 verwendet, um eine höhere Bildwiederholrate zu erzielen.

Ein weiterer Parameter regelt die Zeitspanne, nach der ein „Trigger“-Element in jedem Fall die nächsten Daten passieren lässt, auch wenn die abgeschlossene Verarbeitung der vorherigen Daten noch nicht bestätigt wurde. Dieses Verhalten ist beim Systemstart (z.B. Pipeline mit zweitem „Trigger“-Element wurde noch nicht gestartet) sowie für den Fall notwendig, dass gesendete Daten verworfen wurden.

Bridge (implementiert)

Dieses im Rahmen dieser Arbeit implementierte Element dient hauptsächlich zum Testen von Algorithmen, die (noch) nicht in GStreamer implementiert sind. Aus dem Kontext der GSteamer-Pipeline können beliebige Kommandozeilenprogramme aufgerufen werden, die dann mit den Daten aus der Pipeline arbeiten. Nach Terminierung des aufgerufenen Programmes werden die Ergebnisse der Verarbeitung dem nächsten Element der Pipeline zur Verfügung gestellt.

Das Element besitzt beliebig viele Eingangs-Pads und ein Ausgangs-Pad. Vor dem Aufruf des Kommandozeilenbefehls werden die Daten der Eingangs-Pads jeweils in eine Datei geschrieben, aus der sie dann vom Kommandozeilenprogramm ausgelesen werden. Das Kommandozeilenprogramm erzeugt eine Ausgangsdatei, die dann eingelesen wird.

Diese Art der Ausführung von Algorithmen ist von ihrer Effizienz her nicht optimal, da insbesondere der Zugriff auf Massenspeicher deutlich langsamer als der Zugriff auf den Arbeitsspeicher ist. In Tests konnte jedoch von Systemtools keinerlei Festplattenaktivität beobachtet werden. Dieses liegt an der Zwischenspeicher-Strategie des Linux-Betriebssystems. Die Daten werden physikalisch erst nach einigen Sekunden auf die Festplatte geschrieben. Da zuvor schon der Befehl zum Löschen der Dateien erfolgt ist, kommt es auch nicht zum Schreibvorgang. Die Performance wird wie oben erwähnt mit dem Dateisystem des Betriebssystems, einem „ext4“-Dateisystem auf einer mechanischen Festplatte und in einem weiteren Test auch mit einem „ext2“-Dateisystem auf einer RAM-Disk getestet. Das Ergebnis hierbei ist, dass die Performance auf dem Dateisystem des Betriebssystems sogar besser ist. Hier scheint die Caching-Strategie des Betriebssystems sehr ausgereift zu sein. Trotzdem erzeugt diese Art der Implementierung Overhead durch Kopieroperationen von Daten im Speicher sowie durch das Starten und Verwalten von Prozessen.

Dieses Element wird im Rahmen eines anderen Projekts zur Erzeugung eines Panoramas aus vier Einzelbildern mit Hilfe der Kommandozeilenprogramme aus der Hugin-Programmsammlung genutzt (vgl. 4.7). Die Untersuchungen führen zur Implementierung eines eigenen Elements für diesen Einsatzzweck.

In einem weiteren Projekt, das thematisch nicht direkt mit dieser Arbeit zusammenhängt, wird das Element dazu verwendet, um aus einer in Textform ausgegebenen Matrix mit Wahrscheinlichkeitswerten unter Benutzung des Programms Octave eine dreidimensionale Darstellung zu erzeugen.

Inotifysrc (implementiert)

Das Element „Inotifysrc“ kann als universelles Quellelement gesehen werden. Analog zu dem Element „Bridge“ können hier Kommandozeilenprogramme zum Erzeugen der Dateien verwendet werden. Eine Besonderheit ist hierbei die Verwendung des „Inotify“-Systems des Linux-Kernels. Auf diese Art kann ein Programm je nach Konfiguration ein Signal

erhalten, wenn in einem Verzeichnis eine neue Datei erstellt wird. Außerdem kann von dem Element „Inotifysrc“ ein unabhängiger Prozess gestartet werden und die von diesem Prozess erzeugten Dateien werden sofort eingelesen.

Das Element kann beispielsweise benutzt werden, um über die Bibliothek Libgphoto[9] Digitalkameras anzusteuern. Die Digitalkamera, beispielsweise eine Spiegelreflexkamera von Canon, wird dazu per USB verbunden. Nun wird per USB eine Aufnahme ausgelöst und die Daten werden direkt ohne Zwischenspeicherung auf dem Flash-Speicher der Kamera über USB übertragen. So können alle im Rahmen dieser Arbeit implementierten Algorithmen auch mit einer sehr hochauflösenden Kamera getestet werden, die jedoch eine geringe Wiederholrate aufweist.

Verarbeitung von Bildausschnitten (implementiert)

Eine vielseitig einsetzbare Möglichkeit zur Übertragung von Bildausschnitten wird mit den Elementen „Pipgen“ und „Pip“ implementiert, deren Namen von der englischen Abkürzung PIP („Picture in Picture“) abgeleitet ist. Dieses Elementepaar findet Verwendung in dem Szenario zur Übertragung von ROIs (Region-of-Interest - Bildregion, die von besonderem Interesse ist, siehe Abschnitt 4.2). Im Rahmen dieses Szenarios werden Regionen, die wichtige Bildinformationen enthalten, also hier Gesichter, in bestmöglicher Auflösung übertragen, während das Gesamtbild nur in verkleinerter Auflösung übertragen wird. Auch das Element „Multivis“ stellt eine Bild-in-Bild-Funktion bereit. Mit dem Element “Multivis“ kann jedoch nur ein Bildausschnitt eingeblendet werden, wobei die Einblendung nicht an der ursprünglichen Bildposition erfolgt. Die Implementierung der Funktion im Element „Multivis“ ist eher zur Anzeige des Bildes einer zusätzlichen Kamera auf einer Schwenk-Neige-Einheit (PTU) vorgesehen, während die Funktionalität der Elemente „Pipgen“ und „Pip“ ausschließlich dazu gedacht ist, auf einem Videodatenstrom zu arbeiten. Das Ausschneiden der Daten aus einem Videodatenstrom erfolgt durch das Element „Pipgen“, das Zusammenfügen durch das Element „Pip“

Pipgen Das Element „Pipgen“ besitzt je zwei Eingangs- und zwei Ausgangs-Pads:

- Eingangs-Pad für Hauptvideodatenstrom; an dieses Eingangs-Pad wird der Videodatenstrom in voller Auflösung geleitet. Aus diesem werden dann die einzelnen Bildausschnitte extrahiert.
- Eingangs-Pad für Koordinateninformationen; an diesem Eingangs-Pad wird eine Liste von Koordinateninformationen entgegengenommen. Diese Liste beschreibt eine beliebige Anzahl von rechteckigen Bereichen, die aus dem Hauptvideo zu extrahieren sind.
- Ausgangs-Pad für Bildausschnitte; hier werden sequentiell die Bilddaten aus dem Videodatenstrom gesendet. Pro Eingangsbild werden abhängig von der Anzahl der Bildregionen beliebig viele Ausschnitte nacheinander gesendet. Es werden dabei stets die

[9] www.gphoto.org

Metainformationen (Caps) aktualisiert, um das Folgeelement über die veränderte Auflösung zu informieren.

- Ausgangs-Pad für Koordinateninformationen; um auf der Gegenseite eine Zuordnung der einzelnen Bildausschnitte zu ermöglichen, wird zu jedem gesendeten Satz Bildausschnitte die dazugehörige Koordinatenliste gesendet. Auch wenn die Koordinateninformationen direkt an das Zielelement gesendet werden können, ist diese Art der Weiterleitung der Koordinateninformationen in der aktuellen Implementierung erforderlich, um sicherzustellen, dass zu jedem Satz Bilddaten auch die passenden Koordinaten vorliegen.

In dem Element können zwei verschiedene Versendestrategien konfiguriert werden. In der synchronen Betriebsart werden nur Bildausschnitte erzeugt, wenn für genau dieses Bild eine aktuelle Liste von Koordinateninformationen vorliegt, also die Bilderkennung abgeschlossen ist. Stimmen die fortlaufenden Nummern der Koordinateninformationen und Bilddaten nicht überein, werden die älteren Daten verworfen. In vielen Konfigurationen ist die Bilderkennung aus Performancegründen nicht für jedes Bild möglich. Daher werden nicht für alle Bilder entsprechende Koordinateninformationen berechnet.

In der unsynchronisierten Betriebsart werden für alle eintreffenden Bilder die Ausschnitte basierend auf den zu diesem Zeitpunkt verfügbaren Koordinateninformationen erzeugt, auch wenn diese dann üblicherweise von einem älteren Bild stammen. Somit kann eine höhere Bildwiederholrate erzeugt werden. Es kann jedoch insbesondere bei Szenen mit viel Bewegung vorkommen, dass die Bildausschnitte dann nicht mehr das gesuchte Bilddetail enthalten.

Pip Das Element „Pip“ ist das Gegenstück zu dem Element „Pipgen“. Dieses Element fügt alle eingehenden Bildausschnitte in einen Videodatenstrom der ursprünglichen Größe ein. Das Element hat drei Eingangs-Pads und ein Ausgangs-Pad:

- Eingangs-Pad für Videodatenstrom; an dieses Eingangs-Pad wird der Videodatenstrom in ursprünglicher Auflösung geleitet. Der Bilddatenstrom der ursprünglichen Größe kann erzeugt werden, indem der Bilddatenstrom mit verringerter Auflösung wieder hochskaliert wird.
- Eingangs-Pad für Bildausschnitte; hier werden sequentiell die extrahierten Bilddaten entgegengenommen. Die Bildausschnitte werden solange zwischengespeichert, bis alle benötigten Ausschnitte eingetroffen sind, die zum Erzeugen des Ausgabebildes notwendig sind.
- Eingangs-Pad für Koordinateninformationen; an diesem Eingangs-Pad wird eine Liste von Koordinateninformationen entgegengenommen, die die aktuellen Positionen der Bildausschnitte vorgibt. Basierend auf diesen Informationen kann auch festgestellt werden, wann alle Bildausschnitte eingetroffen sind.
- Ausgangs-Pad für Videodatenstrom; auf diesem Ausgangs-Pad wird der Videodatenstrom mit den eingefügten Bildausschnitten ausgegeben.

In dem Fall, dass eine Inkonsistenz in den eingetroffenen Daten vorliegt, werden Daten verworfen. Es wird stets davon ausgegangen, dass die Daten an einem Eingang in korrekter Reihenfolge eintreffen. Wenn also neuere Daten eintreffen, obwohl noch auf ältere Daten gewartet wird, werden diese als verloren betrachtet. Es werden dann soweit notwendig andere zwischengespeicherte Daten verworfen, bis wieder ein konsistenter Zustand hergestellt ist. Kriterium für die Zuordnung der Daten ist hierbei die Offset-Nummer.

Coordgen Das Element „Coordgen“ dient zur manuellen Auswahl von Bildbereichen. Dazu werden mit der Computermaus im Ausgabevideo mehrere Rechtecke definiert, die dann in die Liste der zu extrahierenden Bildbereiche aufgenommen werden. Die rechte Maustaste setzt die Auswahl zurück. Das Element dient zum Testen des oben beschriebenen Elementepaars, kann aber auch zur Tele-Operation eingesetzt werden. Hierbei muss das „Pipgen“-Element auf unsynchronisierte Betriebsart konfiguriert werden.

Videosplitter (implementiert)

Bei dem Element „Videosplitter“ handelt es sich um ein Element, das das Eingangsbild in mehrere überlappende Abschnitte unterteilt. Es kann konfiguriert werden, ob die Unterteilung horizontal oder vertikal erfolgen soll. Die Mindestgröße des Überlappungsbereichs kann ebenfalls konfiguriert werden. Der Einsatzzweck dieses Elements besteht darin, parallel unterschiedliche Bereiche des Bildes zu verarbeiten und somit eine Leistungssteigerung zu erzielen bzw. die Rechenlast auf verschiedene Funktionseinheiten zu verteilen. Die Funktionsweise des Elements ist in Abbildung 3.13 visualisiert. Wenn es sich bei dem Ausgabedatentyp des nachfolgenden Verarbeitungsschrittes wieder um Bilddaten handelt, können die Abschnitte mit dem nachfolgend beschriebenen Element „Videojoin“ wieder zu einem Gesamtbild zusammengefügt werden. Für andere Datentypen müssen wenn möglich individuelle Funktionen zur Vereinigung mehrerer Teilergebnisse implementiert werden. Im Rahmen dieser Arbeit wird dieses für die Berechnung des SIFT-Feature-Vektors realisiert.

Der Überlappungsbereich ist vorgesehen, weil bei einigen Bildverarbeitungsalgorithmen die Nachbarpixel auf das Ergebnis der Bildverarbeitung Einfluss haben. Ist kein Überlappungsbereich vorgesehen, so kann die korrekte Berechnung des Ausgabebildes an den Übergangsbereichen nicht erfolgen, da diese dann direkt am Rand des jeweiligen Bildausschnittes liegen. Bei einem ausreichend gewählten Überlappungsbereich werden die Pixel, für die wegen ihrer Randlage keine korrekte Berechnung möglich ist, bei der späteren Zusammenführung verworfen.

Das Element hat ein Eingangs-Pad und beliebig viele Ausgangs-Pads, die zur Konfigurationszeit der Pipeline angefordert werden. Die Bildauflösung der einzelnen Ausgänge wird automatisch in Abhängigkeit von der Anzahl der Ausgänge und der Vorgabe für die Mindestgröße des Überlappungsbereichs bestimmt. Der tatsächliche Überlappungsbereich

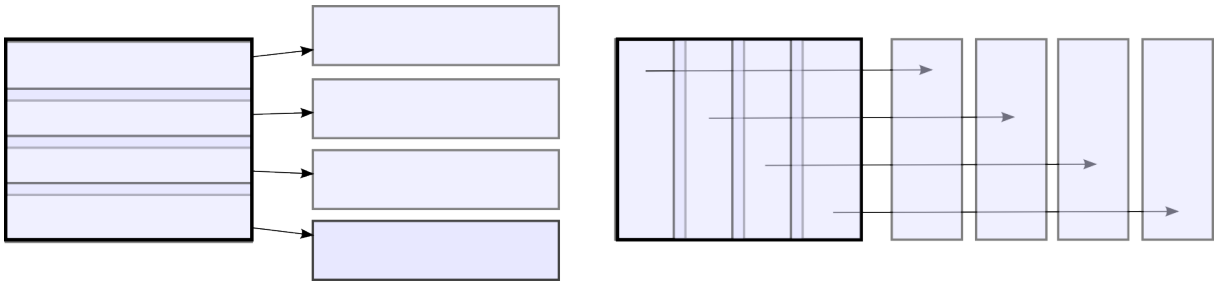


Abbildung 3.13.: Diese Abbildung visualisiert die Funktionsweise des Elements „Videosplitter“. Links ist die horizontale Unterteilung in überlappende Bildregionen dargestellt, rechts die vertikale Unterteilung. Die Art der Unterteilung sowie die Mindestbreite des Überlappungsbereichs sind konfigurierbar.

kann vom System auch größer gewählt werden, wenn sich keine mögliche Konfiguration ergibt. Nachfolgend wird die Bestimmung der tatsächlichen Bildabschnitte erläutert.

Sei N die Anzahl der Ausgangs-Pads, h die Höhe des Ausgangsbildes in Pixel und o die Mindestbreite des Überlappungsbereichs. Des Weiteren ist E_h als Einheitshöhe (E_b als Einheitsbreite) definiert. Es sind nur Bildgrößen erlaubt, die ein Vielfaches dieser Einheitsgrößen darstellen. Im Falle von RGB-Bildern sind Einheitshöhe und Einheitsbreite gleich eins, d.h., es gibt keinerlei Beschränkungen hinsichtlich der Bildgröße. Beispielsweise sind bei einigen YUV-Formaten nur Bildgrößen erlaubt, die einem Vielfachen von zwei entsprechen, wenn sich benachbarte Pixel die Farbinformationen teilen. An den vom System gewählten tatsächlichen Überlappungsbereich werden bei einer horizontalen Unterteilung folgende Anforderungen gestellt:

- $(h + o \cdot (N - 1))$ muss sich ohne Rest durch N teilen lassen.
- $(h + o \cdot (N - 1))/N$ muss ein Vielfaches der Einheitshöhe sein.
- Jedes der Teilbilder muss bei einem Vielfachen der Einheitshöhe beginnen.

In der Implementierung werden die Bedingungen für die gewählte Breite überprüft. Schlägt eine Bedingung fehl, wird die Breite solange erhöht, bis eine Lösung gefunden ist oder bis festgestellt wird, dass keine Lösung existiert. Letzteres ist genau dann der Fall, wenn die Breite des Überlappungsbereichs durch die iterative Erhöhung gleich der Größe der Dimension ist, die unterteilt werden soll (im Beispiel die Bildhöhe).

Als Beispiel soll ein Bild mit einer Bildhöhe von 480 Pixeln in vier überlappende Abschnitte unterteilt werden, die mindestens eine Überlappung von zehn Pixeln aufweisen. Es gibt keinerlei Beschränkungen hinsichtlich der Einheitshöhe. Das System erzeugt dann vier Teilbilder mit einer Höhe von 129 Pixeln und der ursprünglichen Breite, was einem tatsächlichen Überlappungsbereich von 12 Pixeln entspricht. Dieses Ergebnis kann folgendermaßen verifiziert werden:
$4 \cdot 129 = 516 \Rightarrow 36(516 - 480)$ Bildpunkte werden auf 3 (N-1) Überlappungsbereiche verteilt $\Rightarrow$ dies entspricht 12 Pixeln pro Überlappungsbereich.

Eine Besonderheit ergibt sich bei der Implementierung hinsichtlich der Effizienz der verschiedenen Unterteilungsmodi. Die horizontale Unterteilung verzichtet komplett auf Kopiervorgänge der Nutzdaten. Dies ist aufgrund der Repräsentation der Bilddaten im Speicher möglich. Die Daten werden zeilenweise von links nach rechts beginnend mit der obersten Zeile gespeichert. Somit können andere horizontale Abschnitte einfach durch Versetzen des Zeigers und eine veränderte Deklaration der Bildhöhe erfolgen. Bei der vertikalen Unterteilung des Bildes ist stets ein Kopiervorgang in einen neuen Speicherbereich notwendig.

In den Informationen zum Bilddatenstrom, den sogenannten „Caps“, wird die Information über die Breite des Überlappungsbereichs mit erfasst. Auf diese Art können die Elemente zum Zusammenfügen diese Information auslesen und automatisch die korrekte Ursprungsbildgröße rekonstruieren.

Videojoin (implementiert)

Bei dem Element „Videojoin“ handelt es sich um das Gegenstück des zuvor beschriebenen Elements „Videosplitter“. Das Element dient dazu, mehrere Bildabschnitte mit oder ohne Überlappungsbereich zu dem Ursprungsbild zusammenzufügen. Hierzu besitzt das Element mehrere Eingangs-Pads und ein Ausgangs-Pad. Es kann wahlweise horizontal oder vertikal unterteilte Abschnitte zusammenfügen.

Die Rekonstruktion der ursprünglichen Bildgröße wird von dem Element selbstständig durchgeführt. Hierzu wird aus den Informationen, den „Caps“, die Bildgröße und die Breite des Überlappungsbereichs ausgelesen. Ebenfalls beeinflusst die Anzahl der Eingänge die resultierende Bildgröße. Im Falle einer horizontalen Unterteilung ergibt sich die ursprüngliche Bildhöhe in Pixeln zu:
$h = N \cdot h_s - (o \cdot (N - 1))$,
wobei h_s die Höhe der Bildausschnitte, N deren Anzahl und o die Breite des Überlappungsbereichs ist. Für das zuvor erwähnte Beispiel ergibt sich folgende Rechnung:
$4 \cdot 129 - (12 \cdot (4 - 1)) = 480$
Dieser Wert entspricht der Ursprungsgröße.

Bei der Benutzung des Elements muss durch Vorkehrungen an anderer Stelle in der Pipeline gewährleistet sein, dass auf allen Eingangs-Pads Daten zuverlässig eintreffen. In dem Element selbst erfolgt keine Kontrolle der Offset-Werte der Dateneinheiten. Sobald auf allen Eingangs-Pads Daten eingegangen sind, beginnt die Verarbeitung. Kommen auf einem Eingangs-Pad neue Daten an, obwohl die Verarbeitung noch nicht begonnen hat, werden die alten Daten verworfen. Somit würden nicht zusammengehörende Daten fusioniert werden. Abhilfe schafft die Verwendung des modifizierten Elements „Multisync“ oder andere Vorkehrungen in der Pipeline, die sicherstellen, dass keine einzelnen Bildausschnitte auf ihrem Verarbeitungsweg verloren gehen.

Multivis (implementiert)

„Multivis“ ist ein Element, das dazu dient, verschiedene Informationen in das Bild einzublenden. Hierbei handelt es sich in der Regel um die Ergebnisse der Bildverarbeitung verschiedener Bildverarbeitungsalgorithmen. Auch Informationen zu Latenz und Bildwiederholrate können direkt ins Bild eingeblendet werden. Das Element dient darüber hinaus dazu, Benutzereingaben entgegenzunehmen, zu verarbeiten und bestimmte Aktionen zu veranlassen. Bei den verschiedenen untersuchten Szenarien wird das Element in unterschiedlichen Konfigurationen verwendet. Das Element besitzt mehrere Eingangs- und Ausgangs-Pads. Zwei dieser Pads, ein Eingangs- und ein Ausgangs-Pad, dienen zum Durchleiten des anzuzeigenden Videodatenstromes. Die Verarbeitung wird jedes Mal beim Eintreffen neuer Daten angestoßen, sämtliche zu diesem Zeitpunkt auf den anderen Eingangs-Pads bereits eingetroffene Daten werden berücksichtigt. Die einzelnen Anzeigefunktionen sind nachfolgend erläutert.

Einblenden von Bildinformationen Abhängig von der Art der extrahierten Bildinformationen kommen verschiedene Funktionen zur Anwendung, die dem Benutzer die Ergebnisse visualisieren:

Markieren von Bildpunkten: Eine Funktion des Elements besteht darin, bestimmte Punkte im Bild zu markieren, die für die aktuelle Aufgabe von Bedeutung sind. Diese Funktion wird bei der Visualisierung des SIFT-Algorithmus benutzt, wenn der Benutzer sich die vom Algorithmus detektierten Punkte anzeigen lassen möchte. Auf diese Art lässt sich die grundlegende Funktion der SIFT-Vektorberechnung überprüfen. Es lassen sich jedoch keine übereinstimmenden Features zweier Bilder anzeigen. Aus programmiertechnischen Gründen ist diese Funktion in ein anderes Element ausgelagert.

Markieren von Kategorien: Diese Funktion dient zum Darstellen von Informationen über detektierte Bildbereiche. Es wird dazu ein Rechteck variabler Größe um den entsprechenden Bildbereich gezeichnet, zusätzlich werden Text-Informationen unter dieses Rechteck eingeblendet. In den erkannten Bildbereichen befinden sich Objekte, deren Zugehörigkeit zu einer bestimmten Kategorie erkannt wurde. Die Erkennung selbst erfolgt in einem anderen Element und wird als Liste von Koordinaten dem hier beschriebenem Element „Multivis“ übergeben. Bei der Lokalisation von Gesichtern in einem Bild wird beispielsweise diese Art der Markierung verwendet, da nur das Vorhandensein und die Position eines oder mehrerer Gesichter bestimmt wird, weitere Informationen über das Gesicht jedoch nicht vorliegen.

Markieren von spezifizierten Bildbereichen: Die Markierung spezifizierter Bildbereiche unterscheidet sich von der Markierung der Kategorien hauptsächlich dadurch, dass weitere Informationen zu den im detektierten Bildbereich befindlichen Objekten in das Bild eingeblendet werden. Üblicherweise ist dies der Name des Objektes, es können zudem extrahierte Informationen zur Position des Objektes im Kamerakoordinatensystem sein. Jedes Datenpaket enthält Informationen über jeweils nur ein detektiertes Objekt.

Dies hat den Hintergrund, dass bei den gängigen Algorithmen die verschiedenen Objekte sequentiell detektiert werden. Daher ist es zum Erzielen der geringsten Latenz sinnvoll, verfügbare Informationen sofort auszugeben, anstatt auf mögliche weitere detektierte Objekte zu warten.

Eine Besonderheit ergibt sich, wenn die Erkennung der Bildinformationen an Bilddaten mit verringerter Auflösung erfolgt. In diesem Fall wird die Information über die bei der Erkennung verwendete Auflösung eingelesen und bei der Markierung mit berücksichtigt, so dass die Bildinformationen an der korrekten Position markiert werden. Es ist möglich, verschiedene Erkennungsalgorithmen unabhängig voneinander mit verschiedener Auflösung zu konfigurieren. Es ist sogar möglich, den gleichen Algorithmus zu Testzwecken mehrfach in unterschiedlichen Auflösungen auszuführen, um so die Qualität der extrahierten Bildinformationen zu testen. Ein Beispielbild mit eingeblendeten Informationen ist in Abbildung 3.14 gezeigt.

Einblendung der Latenzinformationen Die Latenzinformationen werden für das Hauptbild und sämtliche extrahierten Bildinformationen eingeblendet. Die Latenz des Hauptbildes wird anhand der Differenz zwischen dem Zeitstempel der Aufnahme und der aktuellen Systemzeit bestimmt. Neben der Latenz des aktuellen Bildes wird auch die gemittelte Latenz über eine Anzahl von Bildern ausgegeben. Hierzu wird ein Algorithmus für einen sogenannten einfachen gleitenden Mittelwert implementiert, bei dem eine konfigurierbare Anzahl an Messwerten zu gleichen Teilen in den Mittelwert einfließt. Verschiedene Optimierungen sorgen für einen geringen Rechenaufwand und vermeiden das Überlaufen bei zu großen Werten.

Durch diese Art der Implementierung wird folgendes Verhalten erreicht. Zunächst einmal kompensiert die Mittelwertbildung kurzfristige Schwankungen bei der Latenz. Dadurch, dass zu jeder Zeit nur eine begrenzte Anzahl an Werten den Mittelwert bildet, wird nur ein begrenztes Zeitfenster betrachtet. Dieses ist aus mehreren Gründen durchaus sinnvoll. Beispielsweise werden so nach einem Umschalten der Verarbeitungsstrategie nach einer gewissen Zeit nur noch Messwerte betrachtet, die basierend auf der aktuell gewählten Verarbeitungsstrategie entstanden sind. Ebenso kann sich die Lastsituation auf den beteiligten Systemen längerfristig geändert haben. In solchen Fällen wäre es unerwünscht, ein Ergebnis zu erhalten, das auf nicht mehr aktuellen Situationen beruht. Wird eine Auswertung aller gemessenen Latenzen gewünscht, so ist dies über eine Auswertung der Logdaten möglich, wie sie in 3.2.12 gezeigt wird.

Ähnlich erfolgt die Ermittlung der Bildwiederholrate: Hierfür werden die Intervalle gemessen, die zwischen zwei eintreffenden Bilddaten liegen. Die Auswertung erfolgt anhand der Systemzeit, die zu diesen Eintreff-Zeitpunkten bestimmt wird. Es erfolgt eine Mittelwertbildung wie zuvor bei der Latenzbestimmung beschrieben. Der Kehrwert des durchschnittlichen Intervalls bestimmt nun die Wiederholrate. Der Wert der Wiederholrate kann als Indiz dafür gewertet werden, ob gegenwärtig die Performance der Bildverarbeitungspipeline ausreicht. Dies ist genau dann der Fall, wenn die Wiederholrate gleich der

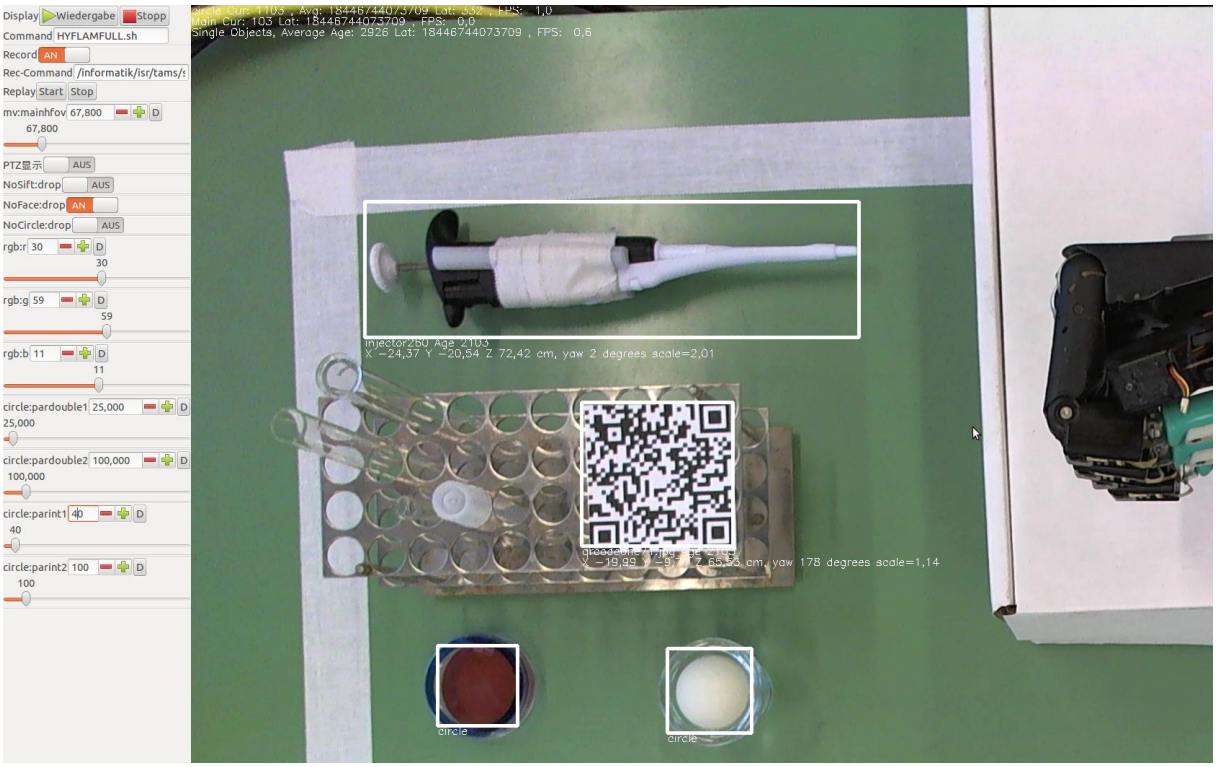


Abbildung 3.14.: Die Einblendung von detektierten Bildinformationen mit dem „Multivis“-Element. In einer Tischszene werden mittels der SIFT-basierten Elemente und der Kreisdetektion bestimmte Objekte detektiert (SIFT für Pipette und QR-Code). Die Informationen hierzu werden im Bild angezeigt. Die Pipeline wird mit der GUI-Software ausgeführt; über ein „Propexport“-Element werden bestimmte Parameter der Pipeline in die GUI eingefügt.

Wiederholrate der Quelle, also z.B. der Kamera, ist.

Bild-In-Bild-Funktion Bei der Bild-In-Bild-Funktion wird in den Videostrom ein zweiter Videostrom eingeblendet. Das „Multivis“-Element weist hierzu einen zweiten Eingang für Videodaten auf, der mit dem weiteren Quell-Element verbunden wird. Die Implementierung dieser Funktion bietet mehrere Anwendungsmöglichkeiten, die im Folgenden erläutert werden:

Eine Anwendung dieser Funktion ist, das Bild einer weiteren Pan-Tilt-Zoom-Kamera in das Hauptbild einzublenden (vgl. Abschnitt 4.7). Diese Funktion ist zur Tele-Operation vorgesehen, also für den Fall, dass ein Robotersystem von einem Benutzer ferngesteuert oder überwacht wird. Zunächst wird im Hauptbild mit der Computermaus ein Rechteck definiert. Dann versucht das System, die PTZ-Kamera auf die ausgewählte Bildregion zu justieren. Unter Berücksichtigung des zuvor angegebenen Bildwinkels werden die notwendige Orientierung der PTZ-Kamera sowie die Zoom-Einstellung berechnet. Die berechnete Kamerakonfiguration wird über ein Ausgangs-Pad ausgegeben; über ein anderes Eingangs-Pad wird die tatsächliche Konfiguration empfangen. Diese kann aufgrund physikalischer Beschränkungen von der berechneten Konfiguration abweichen. Das System probiert nun, den gesamten Ausschnitt in der bestmöglichen Qualität darzustellen. Bestmöglich heißt in diesem Fall, dass keine digitale Hochskalierung vorgenommen wird, also keine Bildpunkte ”hinzugerechnet”werden. Ebenfalls kann die Bildgröße des Hauptbildes eine Einschränkung darstellen, nämlich dann, wenn in diesem Bild nicht ausreichend Platz vorhanden ist, um den Bildausschnitt in der ursprünglichen Auflösung darzustellen. In diesem Fall wird der einzublendende Bildausschnitt so skaliert, dass er gerade noch darstellbar ist. Ist es nicht möglich, den gesamten gewählten Ausschnitt abzudecken, so wird versucht, stattdessen einen möglichst großen Teilausschnitt darzustellen. Ein Beispielbild ist in Abbildung 3.15 gezeigt.

Für den Fall, dass das Hauptbild mit einer verringerten Auflösung übertragen wird, besteht die Möglichkeit, anstatt einer physikalischen Pan-Tilt-Zoom-Kamera eine virtuelle sogenannte ePTZ-Kamera zu implementieren. Hierbei wird dann der gewählte Bildausschnitte mit der bestmöglichen Auflösung in den Videodatenstrom eingeblendet, wobei die Definition von „bestmöglich“ den oben beschriebenen Kriterien folgt. Im Pipeline-Aufbau wird hierbei das Quell-Element zweimal mit dem „Multivis“-Element verbunden: die Verbindung mit dem Pad für den Hauptvideodatenstrom erfolgt mit verringerter Auflösung, die Verbindung mit dem Pad für die Bild-in-Bild-Einblendung erfolgt mit voller Auflösung. Wenn diese Konfiguration gewählt wird, kann beispielsweise das Bild einer intelligenten Kamera in verringerter Auflösung übertragen werden. Es bleibt aber unverändert die Möglichkeit erhalten, Bildausschnitte, die von Interesse sind, in ihrer ursprünglichen Auflösung darzustellen. Dieses ist insbesondere dann sinnvoll, wenn die Bandbreite der Netzwerkverbindung stark begrenzt ist, also insbesondere bei kabellosen Verbindungen. Falls das „Multivis“-Element nicht auf dem gleichem Rechner aufgesetzt ist wie die Bildausgabe, können die notwendigen Steuerkommandos von zwei „Propexport“-Elementen übertragen werden.

Abbildung 3.15.: Hier wird in den Videodatenstrom das Bild einer PTZ-Kamera eingeblendet. Die gewünschte Bildregion wird in der GUI definiert und von dem PTZ-System automatisch abgebildet. (Die Steuerung des Zoom-Objektivs sowie der PTU ist von J. Liebrecht implementiert.)

Darüber hinaus kann auch ein zweiter Bilddatenstrom eingeblendet werden, der einen komplett anderen Bildausschnitt abdeckt oder auch eventuell von einem ganz anderen Computersystem erzeugt wird. In diesen Fällen erfolgt die Einblendung nicht über die Definition eines Rechteckes, sondern kann vom Benutzer aktiviert werden. So kann beispielsweise zeitweise das Bild einer zusätzlichen Kamera an der Roboterrückseite eingeblendet werden. Über einen zusätzlichen Input-Selector kann auch zwischen verschiedenen Kameras gewählt werden.

Die Einblendung des Bildes erfolgt stets unsynchronisiert, d.h., dass bei Verarbeitung eines Bildes des Hauptvideodatenstromes stets das neuste verfügbare Bild verwendet wird. Wenn die beiden Videodatenströme unterschiedliche Bildwiederholraten aufweisen, kann das in ungünstigen Fällen dazu führen, dass die Wiedergabe des eingeblendeten Videos nicht flüssig erscheint. In der Praxis kann dieser Effekt in den getesteten Konfigurationen bislang nicht beobachtet werden. Diese potentielle Einschränkung wird bewusst in Kauf genommen, um eine bestmögliche Darstellung des Hauptvideos zu ermöglichen.

Speichern von Einzelbildern Die Funktion, Einzelbilder zu speichern, ist ebenfalls Bestandteil des „Multivis“-Elements. Da die Verarbeitung von Benutzereingaben innerhalb dieses Elements erfolgt, ist die Steuerung der Abspeicherung von Einzelbildern ebenfalls in diesem Element implementiert. Technisch ist dieses so realisiert, dass die Bilddaten bei Bedarf auf einem von zwei zusätzlichen Pads zur Speicherung ausgegeben werden. Eines

der Pads dient zur Ausgabe des kompletten Bildes inklusive aller eingeblendeter Informationen, das andere zur Ausgabe des aktuell gewählten Bildausschnittes. Der Grund für die Verteilung auf zwei Pads liegt darin, dass auf diese Art bei der Speicherung verschiedene Verzeichnisse in der Grundeinstellung vorgeschlagen werden können. Bei der Speicherung des kompletten Bildinhalts wird lediglich durch einen GStreamer-internen Befehl eine zusätzliche Referenz erzeugt, was im Normalfall ohne Kopieren der Nutzdaten erfolgt. Bei der Objekterkennung kann die Speicherung direkt in das Verzeichnis erfolgen, das die Objektmodelle enthält. In diesem Fall erfolgt die Erkennung und Visualisierung des gewählten Objektes direkt im Anschluss.

PTUControl (implementiert)

Das Element „PTUControl“ realisiert eine aufgabenorientierte Steuerung von Schwenk-Neige-Einheiten (PTU) und der Zoom- und Fokus-Einstellung eines Objektivs. Das Element besitzt ein Eingangs- und ein Ausgangs-Pad für die Konfigurationsdaten von PTU und Objektiv. Auf dem Eingangs-Pad werden die Solldaten entgegengenommen, auf dem Ausgangs-Pad werden die Istdaten ausgegeben. Die Daten enthalten zwei Winkelwerte für die Pan- und Tilt-Achse sowie den gewünschten horizontalen Bildwinkel. Anhand dieser Werte und der vorab bekannten Sensorgröße wird die Einstellung des Zoom-Objektivs vorgenommen.

Das Element wird im Rahmen der Entwicklung eines speziellen Kamerasystems dazu verwendet, aufgabenbezogen Detailaufnahmen von bestimmten Bildbereichen zu erzeugen (s. Abschnitt 4.7). Die Entwicklung der eigentlichen Steuerung der PTU erfolgte durch J. Liebrecht im Rahmen seiner Bachelorarbeit. Die Kapselung der Funktionen in dem GStreamer-Element erfolgte im Rahmen dieser Arbeit.

3.4.1. Anbindung an ROS (implementiert)

Es soll nun gezeigt werden, wie sich die Anbindung des hier verwendeten Software-Frameworks und somit auch der intelligenten Kamera an ROS erreichen lässt. Das ROS-Framework ist in Abschnitt 2.5.1 eingeführt.

Über die bereits existierende ROS-Komponente „gscam“ lassen sich beliebige in GStreamer unterstützte Kameras innerhalb von ROS nutzen. Hierbei wird die verwendete Kamera über eine Pipeline-Beschreibung im „gst-launch“-Syntax konfiguriert. Wie in Abschnitt 3.2.9 erläutert, lassen sich durch die Verwendung des im Rahmen dieser Arbeit entwickelten Elements „Propexport“ auf entfernten Systemen Pipelines ausführen. Will man auf die Bilddaten der intelligenten Kamera unter ROS zugreifen, muss - neben einer bestehenden Netzwerkverbindung - das Daemon-Programm auf der intelligenten Kamera gestartet sein. Auf der Basler eXcite wird nicht direkt ROS installiert. Auf neueren intelligenten Kameras, insbesondere solchen mit x86-Prozessor und ausreichend Arbeitsspeicher, ist die Installation von ROS direkt auf der Kamera möglich.

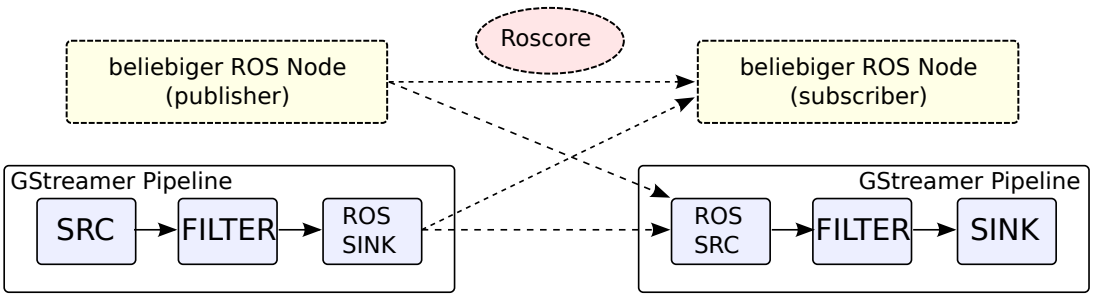


Abbildung 3.16.: Die Abbildung zeigt die entwickelte Methode zur Interaktion zwischen ROS und GStreamer. Auf diese Art kann auch die Anbindung der intelligenten Kamerasysteme an das ROS-Framework erfolgen. Viele Ergebnisse dieser Arbeit können so auch in ROS nutzbar gemacht werden.

Bei der Arbeit mit der „gscam“-Software stellen sich folgende grundlegende Beschränkungen heraus, die es erforderlich machen, die ROS-Anbindung von GStreamer neu zu entwickeln:

- unnötige Kopiervorgänge bei der Datenübergabe,
- nicht konfigurierbares „ROS-Topic“[10], daher systemweit nur eine Instanz nutzbar,
- Datenübertragung nur in Richtung von GStreamer zu ROS möglich,
- Beschränkung auf RGB-Format mit 24 Bit pro Pixel,
- Parameter der GStreamer Pipeline können nicht über das ROS-Parametersystem gesteuert werden.

Aus diesen Gründen werden zur Integration der Ergebnisse dieser Arbeit in ROS vier Elemente implementiert, die im Folgenden erläutert werden sollen. Das entwickelte Konzept der Interaktion zwischen ROS und GStreamer ist in Abbildung 3.16 gezeigt, hier jedoch nur mit den zwei wichtigsten Elementen zur Anbindung.

ROSSink

Über das Element „ROSSink“ können Bilddaten aus dem GStreamer Framework innerhalb von ROS verfügbar gemacht werden. Neben den Bilddaten werden sämtliche Meta-Daten wie Zeitstempel und Informationen zum Bildformat zwischen den Frameworks ausgetauscht. Es können Bilddaten in den Formaten RGB, YUV, Grauwertbilder, Bayer und ebenfalls komprimierte JPEG-Bilder übertragen werden. Bei Kameras, die einen MJPEG-Videodatenstrom senden (z.B. einige Webcams und IP-Kameras[11]), können die einzelnen JPEG-Bilder ohne Decodierung direkt zu ROS übertragen werden. Zur Vermeidung von unnötigen Kopieroperationen wird ein Mechanismus von GStreamer verwendet, der als

[10] In ROS werden die einzelnen Datenströme über „Topics“ identifiziert.
[11] Internet-Protocol Kamera, Kamera mit Netzwerkschnittstelle zur Datenübertragung

„Downstream-Buffer-Allocation“ bezeichnet wird. Bei dieser Methode wird von dem Zielelement direkt ein Speicherbereich dem Quellelement zur Verfügung gestellt. Hier wird also vom Element „ROSSink“ direkt der in ROS verwendete Datentyp „sensor_msgs/Image“ erstellt und der Speicherbereich den vorherigen GStreamer-Elementen zum Schreiben zur Verfügung gestellt. Zusätzlich werden von dem Element noch Daten zur Kamerakalibrierung innerhalb von ROS veröffentlicht. Diese werden über eine Konfigurationsdatei eingelesen, deren Pfad dem Element als Parameter übergeben wird.

Ein Video-Testsignal kann beispielsweise mit folgender Pipeline in das ROS-Framework gesendet werden:

```
gst-launch videotestsrc ! capsfilter caps="video/x-raw-rgb,␣bpp
   =24" ! rossink topic=testvideo
```

Dieser Aufruf führt zur Erzeugung der folgenden „ROS-Topics“, die sich dann wie andere Quellen von Bilddaten nutzen lassen:

```
/testvideo/camera_info
/testvideo/image_raw
/testvideo/image_raw/compressed
/testvideo/image_raw/compressed/parameter_descriptions
/testvideo/image_raw/compressed/parameter_updates
/testvideo/image_raw/compressedDepth
/testvideo/image_raw/compressedDepth/parameter_descriptions
/testvideo/image_raw/compressedDepth/parameter_updates
/testvideo/image_raw/theora
/testvideo/image_raw/theora/parameter_descriptions
/testvideo/image_raw/theora/parameter_updates
```

Von diesen „Topics“ werden nur die ersten beiden durch das Element „ROSSink“ erzeugt, die restlichen werden vom ROS-System erstellt und stellen die Bilddaten in komprimierter Form bereit.

Die Integration der intelligenten Basler eXcite kann beispielsweise durch folgende Pipelines erfolgen. Auf der Kamera:

```
gst-launch baslersrc ! tcpserversink port=1066 protocol=gdp
   buffer-soft-max=1
```

Auf einem PC (die Umgebungsvariable $BASLER muss auf die Netzwerkadresse der Basler eXcite gesetzt sein):

```
gst-launch tcpclientsrc host=$BASLER port=1066 protocol=gdp !
   rossink topic=excite
```

Es kann alternativ auf der eXcite eine Daemon-Instanz gestartet werden und in die Pipeline auf dem PC ein „Propexport“-Element so konfiguriert werden, dass die obere der beiden Pipelines auf der eXcite zur Ausführung gebracht wird.

Angenommen, ROS wäre auf einer intelligenten Kamera direkt installiert, so könnte nach Konfiguration der Netzwerkadresse des „Roscore“-Dienstes auch direkt auf der Kamera eine Pipeline mit dem „ROSSink“-Element gestartet werden.

ROSSrc

Durch das Element „ROSSrc“ können beliebige Bilddatenströme aus dem ROS-Framework in GStreamer verfügbar gemacht werden. Das Element agiert innerhalb des GStreamer-Frameworks als Datenquelle, vergleichbar mit einem Element zum Zugriff auf eine Kamera. Beim Start der Pipeline meldet sich das Element zum Empfang eines definierbaren „ROS-Topics“ an. Jedes Mal, wenn auf dem entsprechenden „Topic“ Daten vorliegen, wird in einer Callback-Funktion ein GStreamer-Buffer erzeugt und innerhalb von GStreamer weitergeleitet. Hierbei wird ebenfalls direkt der Speicherbereich verwendet, der von dem ROS-Framework übergeben wurde.

Es bestehen mehrere Einsatzmöglichkeiten, beispielsweise:

- Anzeige von Videodatenströmen (effizientere Alternative als das ROS-Tool Image-View)
- Zugriff auf Kameras ohne GStreamer-Treiber (z.B. Microsoft Kinect)
- Zugriff auf virtuelle Kameras (z.B. durch Gazebo simuliert)

Ausgehend von dem vorherigen Beispiel, bei dem ein Testsignal in ROS verfügbar gemacht wird, kann dieses Testsignal mit folgender Pipeline angezeigt werden:

```
gst-launch rossrc topic=/testvideo/image_raw ! ffmpegcolorspace
   ! ximagesink sync=0
```

ROSParam

Das Element „ROSParam“ ähnelt von der Funktion her dem Element „Propexport“. Die vom Benutzer ausgewählten Parameter der GStreamer-Elemente der Pipeline werden über das Parametersystem von ROS verfügbar gemacht. Auf diese Art können beispielsweise Aufnahmeparameter der Kamera über die entsprechenden ROS-Tools gesteuert werden.

ROSSiftfolder

Das Element „ROSSiftfolder“ basiert auf dem hier entwickelten „Siftfolder“-Element. Die Informationen zu den detektierten Elementen werden jedoch über ROS-Topics ausgegeben. Dies erfolgt immer dann, wenn ausreichend übereinstimmende Merkmale gefunden

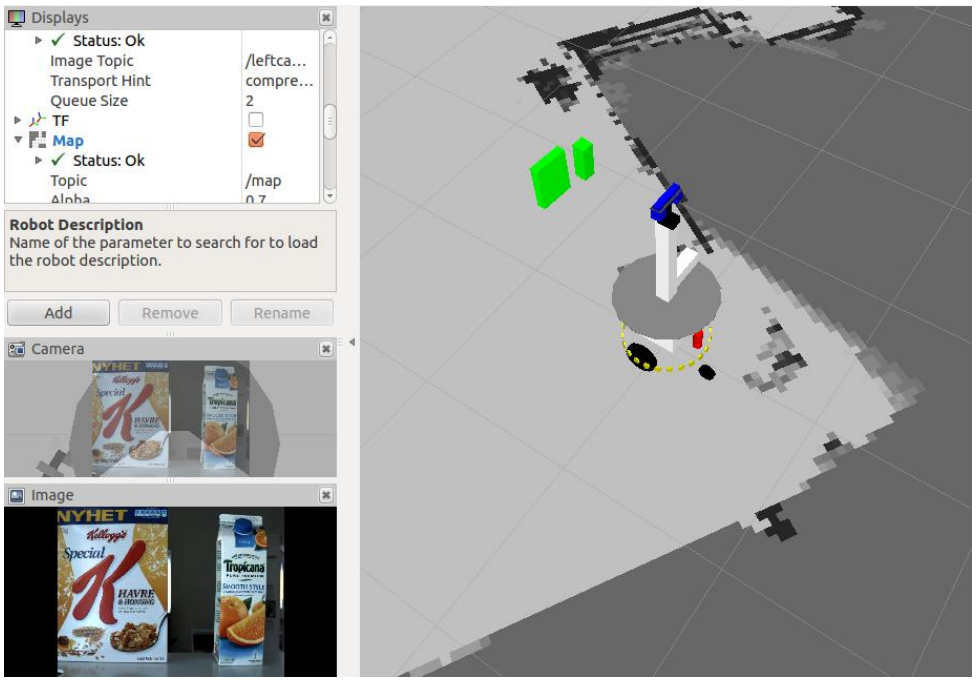


Abbildung 3.17.: Die Abbildung zeigt einen Screenshot des ROS-Tools RViz. Hier werden zwei Objekte detektiert (grün) und mitsamt dem modellierten Robotersystem in einem dreidimensionalen Modell dargestellt. Die Objektdetektion erfolgt mit den hier entwickelten Softwarekomponenten im Rahmen eines anderen Projektes, bei dem eine andere Roboterplattform benutzt wird.

werden und die Abmessungen des realen Objektes bekannt sind. Es wird dann die Pose berechnet und auf verschiedene Arten publiziert:

- als sogenannter „Marker“ zur Visualisierung,
- als Koordinatentransformation über das „Topic“ mit der Bezeichnung „/tf“.

Ein Screenshot des ROS-Tools RViz bei der erfolgreichen Detektion von zwei Objekten ist in Abbildung 3.17 gezeigt. Hier werden in einem dreidimensionalen Modell die „Marker“ angezeigt. Über die ausgegebene Koordinatentransformation wird die Transformation von dem Kamerakoordinatensystem zum Objektkoordinatensystem ausgegeben. Weitere Transformationen können durch die Mechanismen des ROS-Frameworks berechnet werden. Beispielsweise kann die Objektposition zu einem festen Bezugspunkt des Robotersystems bestimmt werden. Basierend auf diesen Koordinaten können auch Greifoperationen geplant werden.

3.4.2. Interprozesskommunikation über Shared Memory (implementiert)

Aus den Tests, die im Abschnitt 4.5 beschrieben werden, geht hervor, dass die Interprozesskommunikation sowohl unter Benutzung des TCP-basierten Protokolls von GStreamer als auch über ROS Overhead erzeugt. Auch in Forschungsergebnissen von anderer Seite zeigt sich, dass bei dem Datenaustausch innerhalb der jeweiligen Frameworks Overhead besteht [ELS+12].

Die Weiterleitung der Daten innerhalb einer GStreamer Pipeline erfolgt in der Regel mit minimalem Overhead. In bestimmten Situationen kann jedoch auch gewünscht sein, einzelne Funktionen und Algorithmen unabhängig voneinander in verschiedenen Prozessen zu starten. Neben der in GStreamer vorgegebenen TCP-Implementierung und der im Rahmen dieser Arbeit entwickelten Anbindung an das ROS-Framework werden im Rahmen dieser Arbeit zwei GStreamer-Elemente „SHMSrc“ und „SHMSink“ entwickelt, die den Shared Memory[12] Mechanismus des Linux-Systems nutzen, um Daten zwischen verschiedenen Prozessen auszutauschen. Es soll untersucht werden, ob sich mittels dieser Elemente die Kommunikation zwischen verschiedenen Prozessen auf einem System effizienter realisieren lässt. Auf diese Art könnte dem Problem begegnet werden, dass in den gängigen Frameworks die Daten erst serialisiert werden müssen und zusätzlicher Overhead durch die TCP-Kommunikation entsteht. Bei dem Shared-Memory Mechanismus wird ein Speicherbereich alloziert, auf den von verschiedenen Prozessen aus zugegriffen werden kann. Die Synchronisation erfolgt über „Mutexe“ und „Conditions“. So kann dem Empfänger mitgeteilt werden, dass neue Daten bereit stehen.

Damit der Datenaustausch tatsächlich ohne zusätzliche Kopiervorgänge abläuft, wird wieder der „Downstream-Buffer-Allocation“-Mechanismus von GStreamer verwendet.

Über beliebig viele Elemente des Typs „SHMSrc“ können die Daten dann abgerufen werden. Die Übertragung eines Testbildes kann über folgende Pipelines erfolgen, die auf einem System gestartet werden müssen.
Sender:

```
gst-launch videotestsrc ! shmsink topic=test
```

Empfänger:

```
gst-launch shmsrc topic=test ! ffmpegcolorspace ! ximagesink
   sync=0
```

Die Datenübertragung über die Elemente „SHMSrc“ und „SHMSink“ wird in Abschnitt 4.5 in die Auswertung mit einbezogen.

[12] nicht zu verwechseln mit Shared Pointern der Boost Library

3.5. Zwischenfazit

In diesem Kapitel ist das entwickelte Konzept vorgestellt, wie Bildverarbeitungsalgorithmen auf verteilten Systemen ausgeführt werden können und dabei Datenaustausch zwischen diesen Systemen erfolgt. Ebenfalls wird aufgezeigt, dass die verschiedenen Systeme von einer zentralen Instanz aus konfiguriert und gestartet werden können.

Die einzelnen Funktionen zur Bildverarbeitung sind als modulare Funktionen implementiert und werden zur Laufzeit zu Pipelines verknüpft. In diesem System sind neben grundlegenden Funktionen zur Bildverarbeitung auch solche zur Gesichtslokalisation und Objektdetektion implementiert, zusätzlich besteht die Möglichkeit, vorverarbeitete Bilddaten innerhalb des ROS-Frameworks verfügbar zu machen. Innerhalb der Softwarearchitektur sind vielfältige Möglichkeiten zur Leistungsmessung von einzelnen Funktionen und gesamten Pipelines implementiert.

Mit der hier beschriebenen Funktionalität können die im folgenden Kapitel dargestellten Experimente und Leistungstests durchgeführt werden.

4. Leistungsanalyse

Gegenstand dieses Kapitels ist eine Evaluation der im Rahmen dieser Arbeit entwickelten Methoden unter verschiedenen Gesichtspunkten. Ziel dieser Untersuchungen ist es, zu prüfen, inwieweit der Einsatz von gegenwärtigen und zukünftigen intelligenten Kamerasystemen für Service-Roboter von Nutzen ist. Zunächst werden in Abschnitt 4.1 Leistungstests des intelligenten Kamerasystems Basler eXcite bei grundlegenden Operationen der Bildverarbeitung beschrieben. Zudem wird untersucht, wie sich in der implementierten Softwarearchitektur die Parallelisierung von Bildverarbeitungsaufgaben umsetzen lässt.

Danach werden die Einsatzmöglichkeiten intelligenter Kameras exemplarisch an den Robotik-typischen Szenarien Gesichtslokalisation in Verbindung mit selektiver Übertragung von Bildregionen (4.2) und verteilte Objekterkennung basierend auf dem SIFT-Algorithmus (4.3) gezeigt. Beide Szenarien werden im Hinblick auf Latenz, Netzwerkauslastung und Bildwiederholrate analysiert.

Im darauf folgenden Abschnitt 4.4 wird eine Methode vorgestellt, wie bei gegebener Aufgabenstellung die Teilaufgaben mit Hilfe maschinellen Lernens auf die aktuell verfügbaren Systeme verteilt werden können.

Grundlegende Tests zur Performance der hier genutzten Architektur und ein Vergleich mit dem weit verbreiteten ROS-Framework werden in Abschnitt 4.5 präsentiert.

Der darauf folgende Abschnitt 4.6 führt aus, wie sich in der Software-Architektur die Recheneinheiten von Grafikkarten mittels OpenCL-basierter Algorithmen nutzen lassen. Hierbei wird auch eine Prognose über die Leistungsfähigkeit des zukünftigen intelligenten Kamerasystems Ximea CURRERA G aufgestellt, das über entsprechende Verarbeitungseinheiten verfügt.

Im Abschnitt 4.7 wird ein weiteres Kamerasystem beschrieben, das aus vier Einzelbildern ein 360°-Panorama generiert und mit den Methoden und Softwarekomponenten dieser Arbeit entwickelt wird.

4.1. Grundlegende Funktionstests

Im Folgenden werden Experimente beschrieben, die die Hauptfunktionen der implementierten Architektur aufzeigen. Zunächst soll die Leistung der intelligenten Kamera Basler eXcite getestet werden und auf bestimmte Schwachstellen näher eingegangen werden.

4.1.1. Übertragung der Bilddaten

Als Anforderung an die Kamera besteht im Bereich der Netzwerkperformance, dass die Bilddaten im Rohdatenformat bei voller Bildwiederholrate über das Netzwerk übertragen werden können. Dies ist beispielsweise für Konfigurationen erforderlich, bei denen Verarbeitungsschritte auf anderen Systemen ausgeführt werden. Bei einer Bildwiederholrate von etwa 18 Bildern pro Sekunde und einer Auflösung von 1388x1038 Pixeln ergibt sich abhängig vom eingestellten Farbmodell ein Datenaufkommen von 26 MB/s für die Rohdaten des Sensors, 52 MB/s für das YUV2-Farbmodell und 78 MB/s für RGB Farbwerte.

Die intelligente Kamera verfügt über eine Gigabit-Ethernet Schnittstelle. Diese ermöglicht theoretisch eine Übertragung von 125 MB/s, bei der Verwendung des TCP-Protokolls verbleiben abzüglich Overhead ca. 100 MB/s als nutzbare Übertragungsrate. Bei Übertragungstests stellt sich jedoch heraus, dass die in der Praxis erzielbare Datenrate im Bereich von 30 MB/s liegt, wobei der Prozessor des Kamerasystems nahezu 100-prozentig mit dem Senden der Daten ausgelastet ist.

In dem synthetischen Netzwerkbenchmark Netio[1] liegt der erzielbare Datendurchsatz bei ca. 60 MB/s, ebenfalls bei 100 % Systemauslastung. Als Ursache für den starken Unterschied stellt sich die Versendestrategie des Netio-Benchmarks heraus. In diesem Programm wird wiederholt dem Funktionsaufruf zum Versenden derselbe Speicherbereich übergeben. Es entfällt also eine Kopieroperation der Daten in den Cache des Prozessors, wie es beim Versenden realer Nutzdaten unumgänglich ist. Durch eine Manipulation des Anwendungsprogramms zum Versenden der Bilddaten können entsprechende Datenraten reproduziert werden, wobei nur nutzlose Daten verschickt werden. Um ganz auszuschließen, dass die Verzögerung von dem angebundenen PC verursacht wird, werden die Tests auf zwei äquivalenten PC-Systemen reproduziert und deutlich höhere Übertragungsraten gemessen.

Das Problem, dass Gigabit-Ethernet-Karten nicht die volle Datenübertragungsrate erreichen, ist nicht auf die Baster eXcite beschränkt, sondern betrifft auch PC mit weniger leistungsstarkem Prozessor. In [Ahl04] werden verschiedene Gigabit-Ethernet-Karten auf einem PC mit einem Intel Pentium 4 Prozessor mit 1,6 GHz getestet und Datenübertragungsraten in ähnlicher Größenordnung gemessen (ca. 60 MB/s bei Verwendung von Netio).

Im Abschnitt „Speicher-Performance“ wird die Ursache für den Einbruch der Datenrate behandelt.

Als Anforderung an zukünftige Generationen von intelligenten Kameras ergibt sich eine deutliche Erhöhung der Netzwerkperformance bei gleichzeitiger Reduzierung der Prozessorlast. Die Übertragung der Bilddaten erzeugt bei der hier getesteten Kamera eine derart hohe Systemauslastung, dass eine zusätzliche Vorverarbeitung, bei der die Datenmenge sich nicht reduziert, nicht bei voller Bildwiederholrate möglich ist (z.B. bei Korrektur von Linsenverzeichnung, Farb-, Helligkeits- und Kontrastanpassungen). Eine Verbesserung in diesem Bereich könnte durch DMA-Anbindung des Netzwerkcontrollers erfolgen.

[1] Netio Benchmark - www.ars.de/ars/ars.nsf/docs/netio

Speicher-Performance

Die Untersuchung der Speicher-Performance der Kamera Basler eXcite erfolgt unter zweierlei Aspekten. Zum einen soll sie eine Erklärung für das Verhalten der Netzwerk-Performance liefern, zum anderen kann so die maximal mögliche Arbeitsgeschwindigkeit abgeschätzt werden. In einem einfachen Testprogramm werden Daten aus einem Speicherbereich in einen anderen kopiert und die dafür benötigte Zeit gemessen. Über die Größe des Datenblockes erhält man die Datenübertragungsrate beim Kopiervorgang. Dieser Wert beträgt auf der Basler eXcite ca. 90 MB/s bei voller Auslastung des Prozessors. Der deutliche Abfall der Netzwerkperformance liegt demnach auch an der beschränkten Speicherbandbreite.

Gleichzeitig gibt diese Messung Aufschluss über die maximal mögliche Verarbeitungsleistung in Bezug auf die anzuwendenden Bildverarbeitungsalgorithmen. Unter der Annahme, das pro Sekunde 78 MB an Bilddaten zu verarbeiten sind (RGB), ergibt sich schon eine fast vollständige Systemauslastung, wenn die Daten nur einmal unverändert kopiert werden. Da bei Bildverarbeitungsalgorithmen aktiv mit den Daten gearbeitet wird, kann die Bearbeitung bei der Basler eXcite nicht für jedes Bild erfolgen. Bei zukünftigen Kameramodellen muss hier also eine deutliche Leistungssteigerung erreicht werden.

4.1.2. Performance grundlegender Operationen

In Rahmen weiterer Leistungstests werden ausgewählte Bildverarbeitungsoperationen auf der intelligenten Kamera Basler eXcite ausgeführt. Die Software erfasst die Systemzeit zu Beginn und am Ende der Operation. Die Differenz ergibt die Ausführungszeit des Algorithmus. Es ist zu beachten, dass dieser Wert gewissen Schwankungen unterliegt. So wirken sich Funktionen des Betriebssystems und insbesondere andere laufende Prozesse auf die Rechenzeit aus. Daher wird darauf geachtet, dass das System während der Messung nicht zusätzlich ausgelastet ist. Es erfolgt eine Mittelwertbildung über mehrere Messungen, um die Auswirkung der Schwankungen zu kompensieren. Die Messungen werden im Hinblick auf die angestrebte Echtzeitverarbeitung betrachtet. Des Weiteren erfolgt ein Vergleich mit der Ausführungszeit auf dem Steuerrechner des Service-Roboters mit einer Pentium 4 CPU mit 2,4 GHz.

Skalierung des Bildes

Zunächst soll die Skalierung des Bildes untersucht werden. Bei der Skalierung handelt es sich um eine grundlegende Operation der Bildverarbeitung, bei der die Auflösung des Bildes verändert wird. Von Bedeutung ist hier nur die Verringerung der Auflösung. So kann beispielsweise erreicht werden, dass die nachfolgende Bildverarbeitungsoperation mit verringerter Auflösung ausgeführt oder das Bild mit verringerter Auflösung übertragen werden kann. Die Skalierung wird durch das in GStreamer vorgegebene Element „Videoscale“

vorgenommen. Es können verschiedene Verfahren angewendet werden, die sich in ihrer Qualität und ihrem Rechenaufwand unterscheiden. Hier werden zwei Verfahren getestet:

- Das „Nearest Neighbour“-Verfahren ist ein einfaches und schnelles Verfahren zur Skalierung. Hierbei wird der Wert des Pixels verwendet, dessen Position im Eingabebild am ehesten der Position des Zielpixels entspricht. Es entstehen in Abhängigkeit von den Auflösungen Artefakte.
- Bei der bilinearen Interpolation werden zusätzlich die umliegenden Pixel bei der Zielwertbildung mit berücksichtigt. Hierdurch werden zum einen Artefakte verringert, zum anderen sinkt das Bildrauschen bei der Verringerung der Auflösung. Dieses Verfahren ist jedoch rechenintensiver.

Die Skalierung wird sowohl bei 8-bit Grauwertbildern als auch bei Farbbildern im YUV-Format (UYVY-Variante) getestet. Die Ergebnisse in Abbildung 4.1 zeigen, dass die Skalierung des Bildes auf der eXcite zwar langsamer erfolgt als auf dem PC mit Pentium 4 CPU, jedoch ausreichend schnell ist, um in Echtzeit bei voller Bildwiederholrate ausgeführt zu werden. Bei der Skalierung im YUV-Format nach VGA-Auflösung, die mit 54,2 ms die längste Rechenzeit erzeugt, liegt die rechnerisch mögliche Bildwiederholrate bei 18,5 fps, was in etwa der vollen Bildwiederholrate der Kamera entspricht. Jedoch wird in diesem Fall die gesamte Rechenkapazität durch die Skalierung verbraucht. Werden zusätzliche Algorithmen auf der Kamera ausgeführt oder wird das Bild übertragen, ist die Rechenleistung nicht mehr ausreichend, um bei voller Bildwiederholrate zu operieren. Es müssen dann Bilder verworfen werden. Auch die anderen getesteten Skalierungen verbrauchen einen gewissen Anteil der Rechenzeit, die dann nicht mehr für andere Prozesse zur Verfügung steht.

Konvertierung des Farbraumes

Die intelligente Kamera Basler eXcite bietet Unterstützung für die FPGA-basierte Konvertierung des Farbraumes. Somit kann praktisch ohne Verzögerung und zusätzliche Auslastung der CPU der Kamera auf die Bilddaten im gewünschten Format zugegriffen werden. Als Ausgabeformat werden Grauwertbilder, YUV-Farbbilder sowie Rohdaten im Bayer-Format unterstützt. In bestimmten Situationen kann es notwendig sein, das Farbformat manuell umzurechnen:

- Es werden nicht unterstützte Datenformate benötigt (z.B. RGB).
- Verschiedene Algorithmen benötigen verschiedene Formate der Eingangsdaten.
- Es sollen Grauwertbilder mit beliebigem Gewicht der Rot-, Grün- und Blau-Komponente erzeugt werden.

Für diese Fälle soll nun die Rechenzeit einiger Operationen untersucht werden. In diesem Rahmen wird ebenfalls die Performance des im Rahmen dieser Arbeit implementierten Elements „RGB2Gray“ evaluiert, das mit Fließkomma-Berechnung arbeitet. Die Ergebnisse in Abbildung 4.2 lassen erkennen, dass auf der Basler eXcite die Konvertierung des

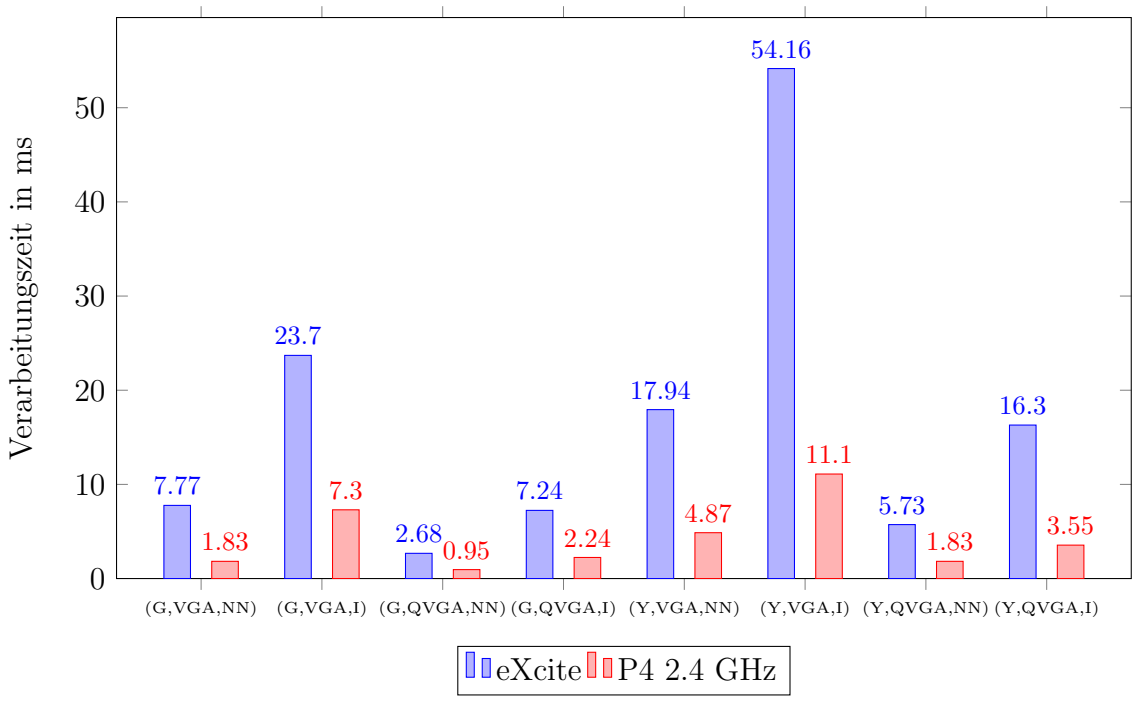


Abbildung 4.1.: Performancemessung Skalierung. Diese Grafik zeigt die Verarbeitungszeit beim Skalieren eines Bildes auf der Basler eXcite im Vergleich zu einem Pentium 4 mit 2,4 GHz. Die Auflösungen werden von 1388 x 1038 auf 640 x 480 (VGA) bzw. 320 x 240 (QVGA) Pixel verringert. Es werden Grauwertbilder (G) und Farbbilder im YUV-Format(Y) getestet und die Skalierung wird jeweils mit dem „Nearest Neighbour“-Verfahren (NN) sowie mit Interpolation (I) durchgeführt.

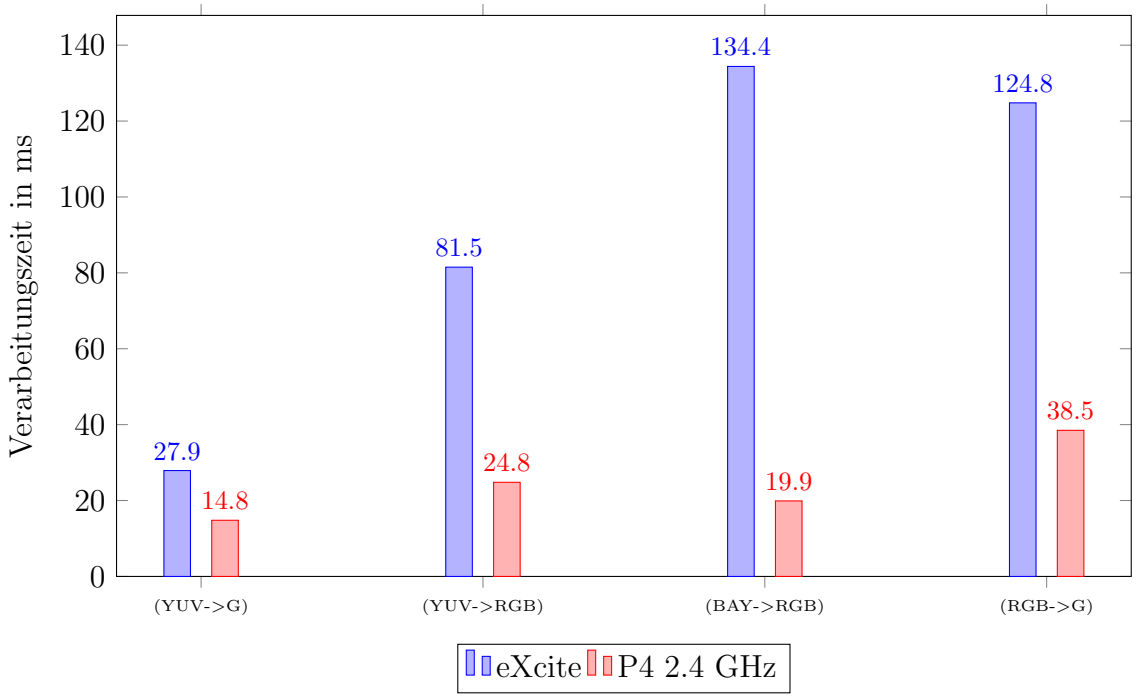


Abbildung 4.2.: Performancemessung Formatkonvertierung. Diese Grafik zeigt die Verarbeitungszeit bei der Farbraumkonvertierung. Die Auflösung beträgt stets 1388 x 1038. Die Konvertierungen YUV->G und YUV->RGB werden mit dem Element „Ffmpegcolorspace“ (vorgegeben) durchgeführt. Die Konvertierung vom Bayer- in das RGB-Format erfolgt über das vorgegebene Element „Bayer2rgb“. Bei der Konvertierung von RGB ins Grauwertbild findet das im Rahmen dieser Arbeit entwickelte Element „RGB2Gray“ Anwendung.

Farbraumes einen hohen Rechenaufwand verursacht. In Anbetracht einer möglichen Bildwiederholrate von etwa 18 fps stehen pro Bild nur etwa 55 ms für die Verarbeitung zur Verfügung. Diese Zeit wird bereits bei drei der vier getesteten Operationen überschritten. Somit zeigt sich in diesem Test einerseits, dass die Hardware-basierte Farbraumkonvertierung der Basler eXcite eine deutliche Einsparung von Rechenzeit ermöglicht. Andererseits zeigt sich jedoch auch die zu geringe Rechenleistung der Basler eXcite. Eine Lösung dieses Problems wäre, wenn die Kamera die Bilddaten gleichzeitig in mehreren Formaten bereitstellen könnte. Dieses wäre ein Ansatzpunkt bei der Entwicklung zukünftiger intelligenter Kamerasysteme.

Automatische Regelung der Bildhelligkeit

Über das Element „Adjuster“ können automatisch die Verschlusszeit und die Sensorempfindlichkeit gesteuert werden. Hierzu muss die Helligkeit des Kamerabildes ausgewertet

werden. Dabei wird ein Histogramm der Helligkeitswerte erstellt und basierend auf der Verteilung eine Regelung vorgenommen.

In der hier ausgewerteten Grundeinstellung wird die Regelung für jedes vierte Bild vorgenommen. Der Rechenaufwand pro Auswertungsvorgang hängt stark von der Auflösung ab. Neben der nativen Auflösung der Kamera (1388 x 1038 Pixel) werden eine VGA-Auflösung und eine QVGA-Auflösung getestet.

Die folgende Tabelle zeigt die Verarbeitungszeiten bei den ausgewerteten Bildern.

Auflösung	Verarbeitungszeit	Verarbeitungszeit inkl. Skalierung
1388 x 1038	114,7 ms	(114,7 ms)
640 x 480	26,7 ms	34,6 ms
320 x 240	8,2 ms	11,3 ms

Es ergibt sich, dass die Regelung von Verschlusszeit und Sensorempfindlichkeit bei verringerter Auflösung in Echtzeit möglich ist. Die Regelung der Bildhelligkeit kann durch die gewählte modulare Architektur asynchron zur Übertragung der Nutzdaten erfolgen. Das bedeutet, dass die Übertragung jedes Bildes gestartet wird, auch wenn die Analyse noch nicht abgeschlossen ist. Dennoch wirkt sich der Rechenaufwand auf die Übertragung aus, wie unten bei den Untersuchungen zu der Latenz auf Empfängerseite gezeigt wird.

Berechnung des SIFT-Vektors

Die Berechnung des SIFT-Vektors wird mit dem Element „Siftextractor“ durchgeführt. Bei der Objektdetektion ist die Berechnung des SIFT-Vektors der erste von zwei Schritten. Es bietet sich vom Prinzip her besonders an, diesen ersten Schritt auf einer intelligenten Kamera auszuführen, weil er zu einer starken Verringerung der Datenmenge führt (siehe 3.3). Bei den Untersuchungen wird auch getestet, welche Auswirkung die Bildgröße auf die Rechenzeit hat. Die Berechnung wird für verschiedene Auflösungen durchgeführt. Die Dauer der Berechnung hängt stark von dem Bildinhalt ab. Daher wird sowohl eine Tischszene (ca. 450 Merkmale bei voller Bildauflösung) als auch ein komplett verdunkeltes Bild analysiert (0 Merkmale). Es ist ersichtlich, dass die Rechenzeit bei der Tischszene deutlich höher ist. Ein entscheidender Teil der Rechenzeit ist jedoch auch abhängig von der Bildauflösung.

Die Ergebnisse dieses Tests sind in Abbildung 4.3 gezeigt. Es ist zu erkennen, dass die Verarbeitungsleistung der Basler eXcite signifikant hinter der Verarbeitungsleistung des PCs mit einem Intel Pentium 4 Prozessor zurückbleibt. Die Berechnung des SIFT-Vektors dauert bei der Tischszene in voller Auflösung über 6 Sekunden, während diese Operation auf dem Pentium 4 in 1,2 Sekunden ausgeführt werden kann. Somit ergeben sich bei Ausführung auf der Kamera erhebliche Verzögerungen im Arbeitsablauf des Service-Roboters. Die in diesem Test nicht enthaltene Suche nach übereinstimmenden Bildpunkten verursacht darüber hinaus noch zusätzliche Rechenzeit.

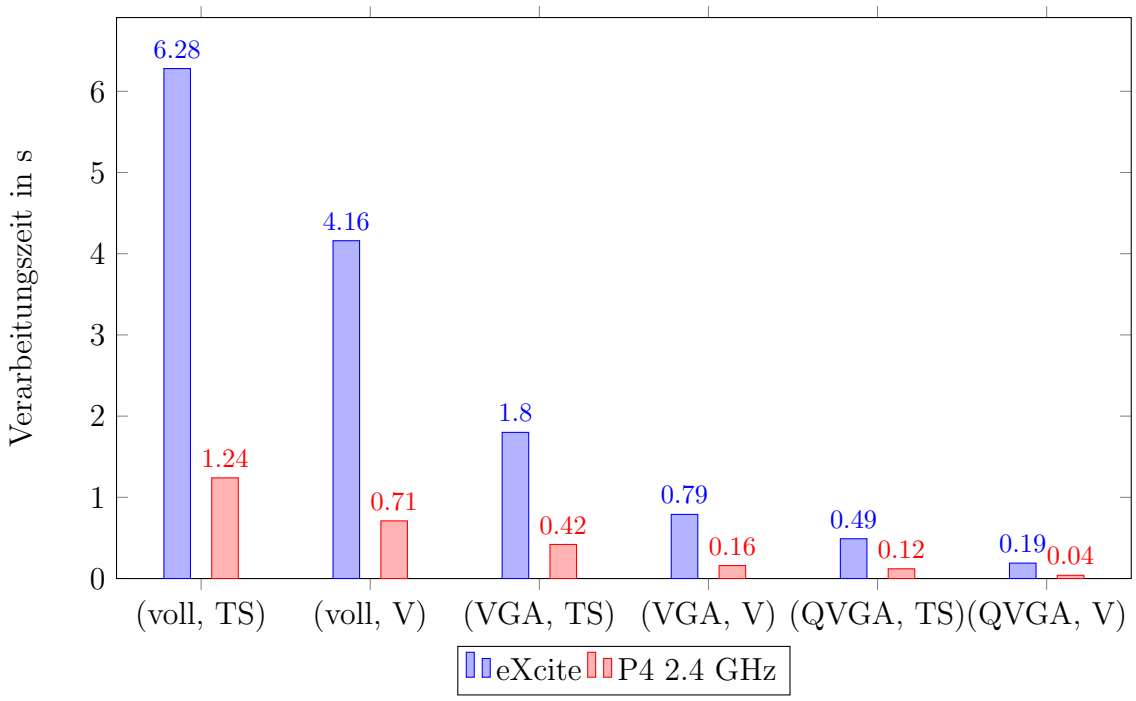


Abbildung 4.3.: Performancemessung SIFT-Vektorberechnung. Diese Grafik zeigt die Verarbeitungszeit des Elements zur Berechnung des SIFT-Vektors auf der Basler eXcite im Vergleich zu einem Pentium 4 mit 2,4 GHz. Die Verarbeitung erfolgt mit den Auflösungen 1388 x 1038 (voll), 640 x 480 (VGA) und 320 x 240 (QVGA) Pixeln. Es wird sowohl eine Tischszene (TS) als auch ein verdunkeltes Bild (V) untersucht. Die Rechenzeit zur Skalierung des Bildes ist nicht in der Zeitmessung enthalten.

Die erzielbaren Bildwiederholraten liegen auf der Basler eXcite bei der praxisrelevanten Tischszene abhängig von der Auflösung bei folgenden Werten:

- 0,16 fps (1388 x 1038 Pixel)
- 0,55 fps (640 x 480 Pixel)
- 2,04 fps (320 x 240 Pixel)

Abhängig von der gewünschten Bildauflösung und der tolerierten Latenz kann die Basler eXcite beispielsweise unter folgenden Rahmenbedingungen sinnvoll eingesetzt werden:

- Eine Auslastung des Steuerrechners vom Service-Roboter soll vermieden werden.
- Eine Auflösung von 640 x 480 Pixel wird als ausreichend angesehen und eine Verzögerungszeit von etwa 2 Sekunden wird im Hinblick auf den Arbeitsablauf der Roboterplattform als akzeptabel angesehen.

Also wird anhand dieses Tests deutlich, dass die Basler eXcite zur Berechnung des SIFT-Vektors eingesetzt werden kann. Eine Verarbeitung aller Bilder in Echtzeit ist mit dieser Kamera jedoch keinesfalls möglich.

Hough-basierte Kreiserkennung

Über die Hough-basierte Kreiserkennung können kreisförmige Gegenstände anhand ihrer Form detektiert werden. Dies kann für die Gegenstandserkennung insbesondere dann nützlich sein, wenn diese Objekte keine anderen signifikanten Merkmale aufweisen. Die Hough-basierte Kreiserkennung ist im Element „GstCV“ implementiert. Innerhalb des in OpenCV implementierten Algorithmus wird zunächst eine Canny-Funktion zur Erzeugung eines Kantenbildes ausgeführt. Die Erkennung kann über mehrere Parameter gesteuert werden, die zum einen die Filterung, zum anderen die Größe der zu suchenden Kreise in Pixel betreffen.

Da nur eine ungefähre Abschätzung der Rechenleistung erfolgen soll, werden die Parameter zur Filterung manuell auf funktionierende Werte eingestellt und im weiteren Testverlauf nicht mehr verändert. Für den Größenbereich des Radius wird im ersten Testdurchlauf ein Bereich von 20 bis 100 Pixeln gewählt, im zweiten Testdurchlauf ein Bereich von 40 bis 60 Pixeln. Werden größere Bereiche gewählt, kommt es auf der Basler eXcite mangels Arbeitsspeicher zum Programmabbruch. Es werden ebenfalls jeweils eine Tischszene und eine komplett verdunkelte Szene untersucht. Die Ergebnisse sind in der Grafik 4.4 dargestellt.

Es zeigen sich ähnliche Ergebnisse wie bei der Berechnung des SIFT-Vektors. Die Berechnung erfolgt auf dem Pentium 4 im Vergleich zur Basler eXcite ungefähr viermal so schnell. Auch hier kommt es auf die Anwendung und die Rahmenbedingungen an, um zu entscheiden, ob die hier festgestellten Verzögerungszeiten als akzeptabel betrachtet werden können.

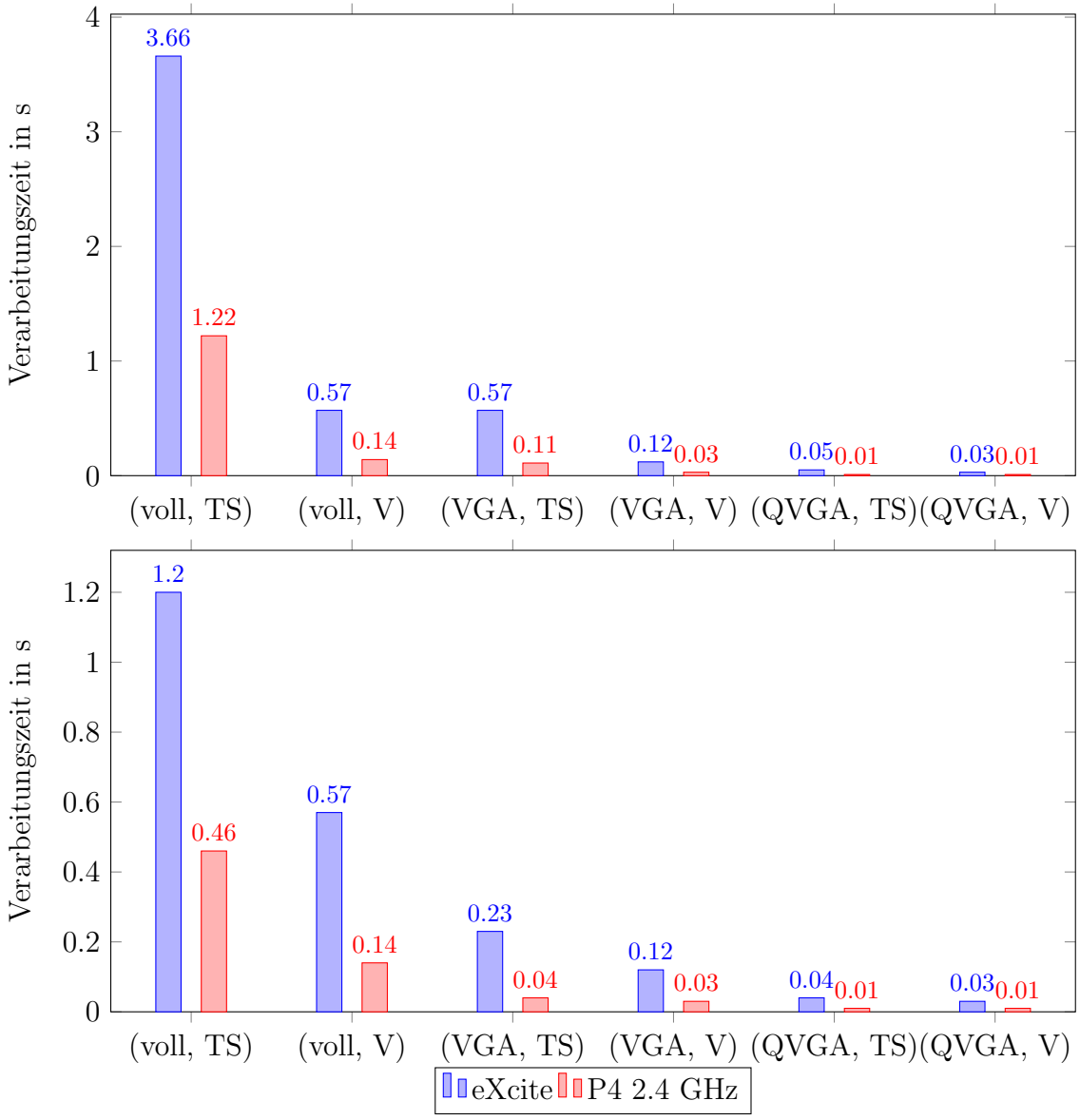


Abbildung 4.4.: Performancemessung Hough-Kreisdetektion. Diese Grafiken zeigen die Verarbeitungszeit des „OpenCV“-Elements zur Detektion von kreisförmigen Regionen mit einem Radius von 20 bis 100 Pixeln (oben) und 40 bis 60 Pixeln (unten). (TS=Tischszene, V=verdunkeltes Bild) Die Verarbeitung erfolgt mit den Auflösungen 1388 x 1038 (voll), 640 x 480 (VGA) und 320 x 240 (QVGA) Pixeln. Die Rechenzeit zur Skalierung des Bildes ist nicht in der Zeitmessung enthalten.

Gesichtslokalisation

Im Folgenden wird die Performance der implementierten Funktion zur Lokalisation von Gesichtern im Kamerabild ausgewertet. In dem später analysierten Szenario zur Übertragung von Bildregionen spielt diese Funktion eine entscheidende Rolle. Diese Funktion ist ebenfalls im Element „GstCV“ implementiert und kann über verschiedenen Parameter konfiguriert werden. Sie werden zu Beginn einmal festgelegt und bleiben im weiteren Verlauf unverändert. Über einen Skalierungsfaktor kann bestimmt werden, in welchen Schritten das Suchfenster für Gesichter vergrößert wird. Größere Schritte erzeugen geringeren Rechenaufwand, weil die Maximalgröße schneller erreicht wird. Es besteht jedoch die Gefahr, dass vorhandene Gesichter nicht erkannt werden, wenn die passende Zwischenstufe übersprungen wird. Über einen weiteren Parameter kann der Grad der Übereinstimmung gewählt werden. Hier muss eine Abstimmung gefunden werden, ob der Detektor eher zu falschen positiven oder eher zu falschen negativen Resultaten neigt. Des Weiteren kann die Mindestgröße des Suchfensters festgelegt werden. In diesem Test wird wie auch in allen folgenden Tests dieser Arbeit ein Skalierungsfaktor von 1,1, ein Übereinstimmungsgrad von 10 und eine Mindestgröße von 30 x 30 Pixeln gewählt. Als Ausgabe des Algorithmus wird eine Liste von Koordinaten der Regionen erzeugt, die Gesichter enthalten.

Es werden wieder drei Auflösungsstufen verwendet. Abbildung 4.5 zeigt die Ergebnisse der Tests. Sie ergeben, dass die Basler eXcite bei der Gesichtslokalisation gegenüber dem Pentium 4 noch deutlich schlechter abschneidet als bei den anderen getesteten Algorithmen und etwa um den Faktor fünf langsamer arbeitet. Es ist ersichtlich, dass bei Reduzierung der Auflösung die Verarbeitungszeit deutlich sinkt. Auf dem Pentium 4 kann die Gesichtslokalisation bei einer Auflösung von 320 x 240 Pixel beispielsweise in etwa 200 ms ausgeführt werden. Die Reduzierung der Auflösung wird auch bei den Tests in Abschnitt 4.2 durchgeführt.

Weitere Algorithmen

In diesem Abschnitt sollen weitere Algorithmen, die in der Softwarearchitektur implementiert sind, auf ihre Verarbeitungszeit hin untersucht werden. Die Tests erfolgen nur auf der Basler eXcite, da die bisherigen Ergebnisse das Verhältnis der Rechenleistungen zueinander bereits ausreichend verdeutlicht haben. Alle Tests werden bei voller Auflösung ausgeführt, auch werden sie immer an derselben Tischszene durchgeführt. Die Ergebnisse sind in Abbildung 4.6 zusammengefasst. Es ist ersichtlich, dass alle Algorithmen eine deutlich zu lange Ausführungszeit aufweisen, als dass bei voller Bildwiederholrate gearbeitet werden könnte. Für eine Verarbeitung bei voller Wiederholrate wäre eine Verarbeitungszeit von unter 55 ms erforderlich. Inwiefern die einzelnen Verarbeitungszeiten akzeptabel sind, muss abhängig von der jeweiligen Anwendung entschieden werden.

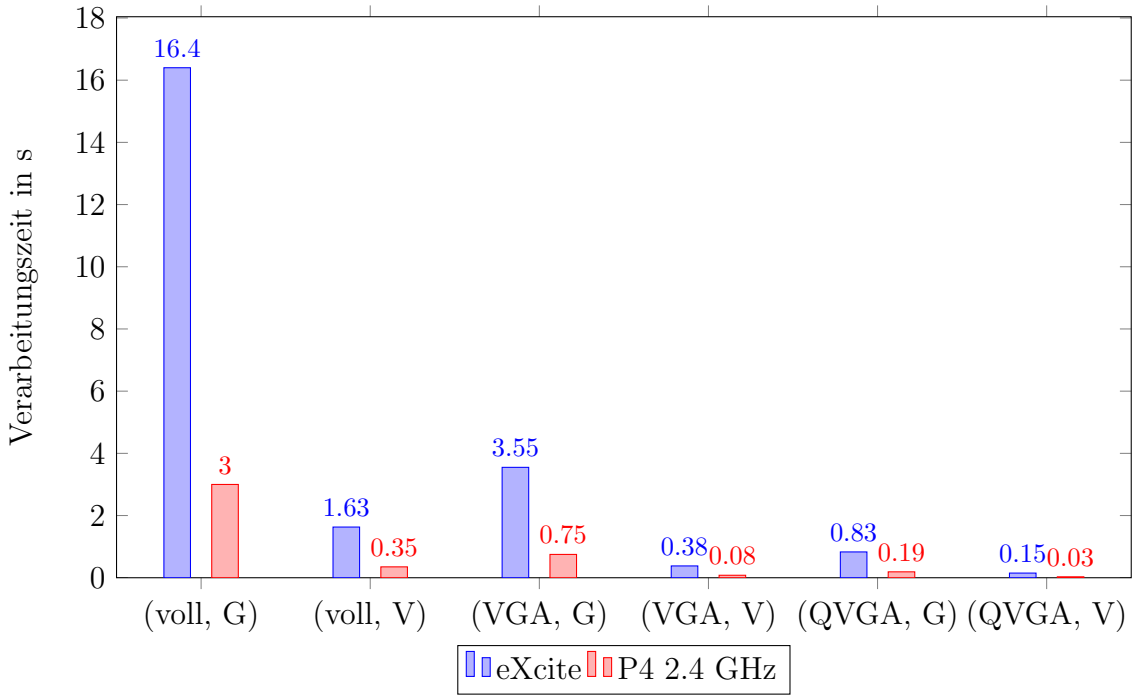


Abbildung 4.5.: Performancemessung Gesichtslokalisation. Diese Grafik zeigt die Verarbeitungszeit, die das „OpenCV“-Element zur Lokalisation von Gesichtern in Bilddaten benötigt. (G=Gesicht in Szene, V=verdunkeltes Bild) Die Verarbeitung erfolgt mit den Auflösungen 1388 x 1038 (voll), 640 x 480 (VGA) und 320 x 240 (QVGA) Pixeln. Die Rechenzeit zur Skalierung des Bildes ist nicht in der Zeitmessung enthalten.

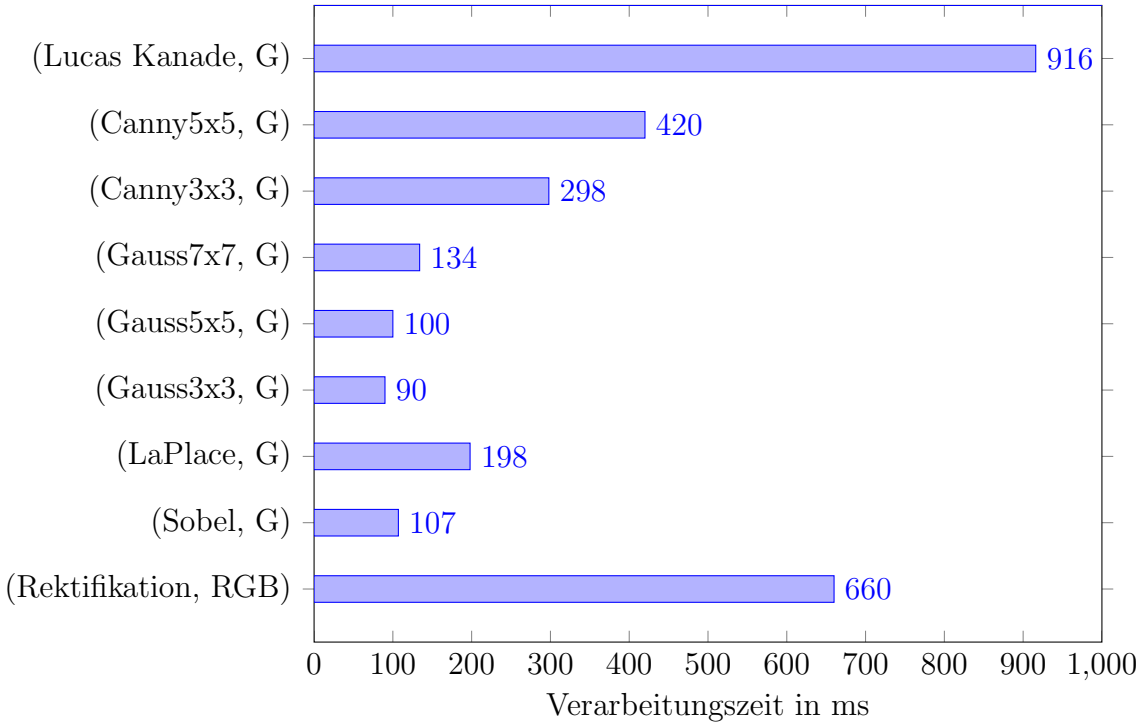


Abbildung 4.6.: Performancemessung verschiedener Operationen auf der Basler eXcite bei voller Auflösung (Tischszene). Bei den Operationen handelt es sich um die Korrektur der Linsenverzeichnung (Rektifikation), Sobel-, Laplace-, Gauß- und Canny-Filter (die beiden letzteren mit variablen Operatorgrößen) sowie Bewegungsdetektion nach Lucas & Kanade. Eine zusätzlich ausgeführte Messung für die Bewegungsdetektion nach der Methode von Horn & Schunck erzielt eine Verarbeitungszeit von 7,5 Sekunden und wird daher nicht grafisch dargestellt.

Latenz auf Empfängerseite

In weiteren Tests werden statistische Daten zur Latenz auf der Empfängerseite ausgewertet. Hierzu ist die eXcite über eine Gigabit-Ethernet-Verbindung an einen PC angeschlossen. Es soll bei diesen Tests insbesondere untersucht werden, wie sich die Ausführung von Verarbeitungsschritten auf die Latenz auswirkt. In folgender Tabelle sind einige Messwerte zusammengefasst:

Funktion	ϕ Lat.	$\pm$ Lat.	min. Lat.	max Lat.	Verworfen
Standardübertragung	39,2 ms	15,8 ms	34,3 ms	239,2 ms	2,4 %
VGA Übertragung (NN)	16,9 ms	0,007 ms	16,3 ms	18,0 ms	0 %
VGA Übertragung (I)	34,1 ms	14,1 ms	31,9 ms	242,8 ms	1,6 %
Regelung (volle Aufl.)	73,0 ms	41,3 ms	33,0 ms	323,1 ms	20,6 %
SIFT (VGA)	2227 ms	25 ms	2201 ms	2277 ms	97,6 %

Bei den Tests zeigen sich zum Teil starke Schwankungen bei der Latenz. Bei der „Standardübertragung“ werden die unmodifizierten Bilddaten übertragen. Die Latenz liegt wie erwartet etwa bei 40 ms. Die ebenfalls ausgewertete Standardabweichung liegt bei vergleichsweise hohen 15,8 ms. Als Grund hierfür stellt sich heraus, dass im Messintervall einige Bilder mit einer Verzögerung von über 200 ms übertragen werden. Während dieser verzögerten Übertragung treten auch die verworfenen 2,4 % Bilddaten auf.

Bei der Übertragung der Bilddaten in VGA-Qualität ist bei der „Nearest-Neighbour“-Skalierung eine wesentlich stabilere Funktion festzustellen. Es kommt zu keinem Datenverlust und zu einer nur minimalen Standardabweichung. Wird jedoch die rechenaufwändigere Skalierung mittels Interpolation verwendet, so resultiert daraus wieder eine höhere Standardabweichung aufgrund von verzögerten Bilddaten und Messdatenverlusten.

Bei Benutzung der Regelung (über das Element „Adjuster“), die hier ohne Reduzierung der Bildgröße ausgeführt wird, ergeben sich besonders starke Schwankungen, die sich in der Standardabweichung und der Relation von minimaler und maximaler Latenz äußern. Dies liegt darin begründet, dass nur jedes vierte Bild ausgewertet wird und es somit zu einer ungleichmäßigen Auslastung kommt.

Wird die rechenintensive Bestimmung des SIFT-Vektors durchgeführt, ergibt sich wiederum nur eine verhältnismäßig geringe Standardabweichung bei der Latenz. Dies beruht darauf, dass sich während der Verarbeitungszeit von über 2 Sekunden kurzzeitige Schwankungen ausgleichen.

Anhand dieser Tests werden die Probleme der Netzwerkanbindung der Basler eXcite bestätigt. Insbesondere bei gleichzeitiger Übertragung und Verarbeitung entstehen deutliche Verzögerungen. Der Grund dafür ist, dass die Datenübertragung über die Netzwerkschnittstelle eine hohe CPU-Last erzeugt. Ist die CPU gerade anderweitig ausgelastet, kommt es zu den beobachteten Verzögerungen. So steigt die Latenz von unter 40 ms auf über 70 ms bei Zuschaltung der Regelung, obwohl in beiden Fällen die Daten direkt verschickt werden. Die Auswertung zur Regelung erfolgt parallel zum Versenden.

Konfiguration	Latenz	Eintreff-Intervall	Auslastung PC	Auslastung eXcite
Canny auf eXcite	352 ms	358 ms	3 %	99 %
Canny auf P4	150 ms	85 ms	99 %	70 %
Canny auf P4 + Last	255 ms	180 ms	100 %	70 %

Abbildung 4.7.: Diese Tabelle zeigt die Ergebnisse einer Übertragungskette. Es wird die Verarbeitungszeit inklusive Netzwerktransfer gemessen. Der Canny-Filter wird auf monochrome Bilder (8-bit) mit 1388 x 1038 Pixeln angewendet. Beim 3. Test (letzte Zeile) wird das System zusätzlich mit 3 „busy-waiting-tasks“ belastet.

Untersuchungen zur Systemauslastung

In den folgenden Tests wird die Gesamtlaufzeit einer Verarbeitungsstrategie mit einer Canny-Filterung, einer Kombination aus Glättung und Kantendetektion, ermittelt. Die Ergebnisse sind in Tabelle 4.7 dargestellt. Anhand dieser Tests wird der Frage nachgegangen, inwiefern durch das Konzept der Verarbeitung auf einer intelligenten Kamera Verbesserungen gegenüber einer klassischen Anbindung von digitalen Kamerasystemen zu erwarten sind. Es werden zwei Verarbeitungsstrategien analysiert, die sich hauptsächlich dadurch unterscheiden, dass der Canny-Algorithmus bei der ersten Strategie auf der Kamera und bei der zweiten auf dem Desktop PC ausgeführt wird. In beiden Fällen wird die Zeitspanne gemessen von der Bildaufnahme bis zu dem Zeitpunkt, wo die Daten verarbeitet auf der Seite des PCs vorliegen. Zusätzlich wird das Intervall zwischen zwei verarbeiteten Bildern gemessen, so dass Informationen über das maximale Alter der Bilddaten errechnet werden können. Die Prozessorauslastung der Systeme wird ebenfalls ermittelt.

Die Untersuchungsergebnisse bestätigen wie erwartet, dass die Verarbeitungsleistung der intelligenten Kamera Basler eXcite nicht auf dem Niveau eines Desktop-PCs liegt. Für das Konzept der intelligenten Kameras spricht jedoch die geringere Systemauslastung des PCs bei der Vorverarbeitung auf dem Kamerasystem. Die Auslastung des PCs verringert sich von 99 % auf 3 %; jedoch erhöht sich das maximale Alter der Bildinformationen (kurz vor dem Eintreffen aktueller Daten) von ca. 0,25 s auf 0,7 s (jeweils Summe aus Latenz und Eintreff-Intervall) .

Ergebnisse bisheriger Performancemessungen

Im Hinblick auf das Ziel, intelligente Kamerasysteme auf einem Robotersystem nutzbar zu machen, kann die angestrebte Funktion der implementierten Software-Architektur gezeigt werden. Es besteht die Möglichkeit, durch Verändern der Auflösung den Rechenaufwand zu variieren und die Ausführung der Algorithmen somit an die aktuelle Aufgabe und die verfügbare Hardware anzupassen. Also kann die Software die zentrale Anforderung der

Skalierbarkeit erfüllen. Auf zukünftigen intelligenten Kamerasystemen lassen sich durch Umkonfiguration vorhandene Pipelines mit höherer Auflösung und höherer Verarbeitungsleistung weiter nutzen.

Aus den bisher beschriebenen Messungen ergibt sich, dass die vorhandene Rechenleistung der intelligenten Kamera Basler eXcite im Verhältnis zur Auflösung nicht ausreichend ist. Bereits bei einfachen Operationen wie der Umwandlung des Farbraumes kommt es zu deutlichen Verzögerungen. Abhängig vom Algorithmus und der gewählten Auflösung kann meist nicht bei voller Bildwiederholrate gearbeitet werden.

4.1.3. Methoden zur Parallelverarbeitung

In den zuvor betrachteten Konfigurationen wurden die Aufgaben nur entweder auf der Kamera oder auf einem PC ausgeführt. Gegenstand der folgenden Abschnitte ist es, darzulegen, welche Möglichkeiten bestehen, die Verarbeitung von Bilddaten auf mehrere Recheneinheiten zu verteilen. Hierbei können verschiedene Recheneinheiten innerhalb eines Computers (z.B. Multi-Core CPU) aber auch verteilte Systeme (z.B. intelligente Kamera und Steuer-PC) in Betracht kommen. Es ist dabei von entscheidender Bedeutung, wie die Gesamtaufgabe verteilt wird. Abhängig von der Bildverarbeitungsstrategie können sich mehrere Alternativen ergeben. Es wird gezeigt, wie die verschiedenen Methoden in dem Softwaresystem implementiert sind.

Diese Untersuchungen dienen im Rahmen dieser Arbeit dazu, die Möglichkeit zur Erhöhung der Bildverarbeitungsleistung durch Benutzung mehrerer Verarbeitungseinheiten aufzuzeigen. Das intelligente Kamerasystem stellt eine zusätzliche Verarbeitungseinheit im Gesamtsystem dar. Ebenfalls können im Hinblick auf den Entwurf zukünftiger Kamerasysteme Anforderungen an die Kamerahardware definiert werden.

Zunächst werden die Unterschiede bei der Verteilung auf einzelne Prozessorkerne und auf verteilte Systeme betrachtet. Im Anschluss daran werden Lösungsansätze zur Aufteilung der Bildverarbeitungsschritte und deren Nutzbarkeit im Hinblick auf intelligente Kamerasysteme analysiert.

Mehrkernprozessoren

In der implementierten Softwarearchitektur ist es nicht möglich, einzelne Teilaufgaben explizit einem bestimmten Prozessorkern zuzuordnen. Es besteht lediglich die Möglichkeit zu steuern, dass ab einer Stufe in der Verarbeitungspipeline alle folgenden Stufen in einem separaten Thread ausgeführt werden. Dies erfolgt durch die „Queue“-Elemente (vgl. Abschnitt 3.4), die in der Pipeline zwischen zwei Elemente gesetzt werden. Diese übernehmen die Zwischenspeicherung der Daten, bis die Verarbeitung in dem anderen Thread fortgeführt wird. Hierbei müssen theoretisch keine Daten im Hauptspeicher kopiert werden. Praktisch ergeben sich dennoch möglicherweise Nachteile bezüglich der Performance,

da die Daten in den Cache des entsprechenden Prozessorkerns kopiert werden müssen und es Verzögerungen beim Starten eines Threads geben kann. Dieses ist abhängig von der jeweiligen Prozessorarchitektur und soll im Rahmen dieser Arbeit nicht weiter erörtert werden.

Verteilte Systeme

Im Gegensatz zu der Verteilung auf Mehrkernprozessoren ist es bei der Verteilung auf verschiedene Systeme explizit möglich, einzelne Aufgaben einzelnen Systemen zuzuordnen. Zur sequentiellen Aufteilung auf verschiedene Systeme sind Paare von TCP-Elementen in den Pipelines vorgesehen. Eines der Elemente wird als Server konfiguriert, eines als Client, der dann die Verbindung zu dem Server herstellt. Aus diesem Grund muss die Pipeline, die das Server-Element enthält, zuerst gestartet werden.

Sequentielle Aufteilung der Pipeline

Die Aufteilung einer Pipeline ist immer dann möglich, wenn in dieser Pipeline mehrere Verarbeitungsschritte hintereinander ausgeführt werden. Die Verarbeitungsschritte können dann von verschiedenen Verarbeitungseinheiten ausgeführt werden. Hier lassen sich einige Parallelen zum Pipelining von aktuellen Prozessoren ziehen (vgl. [HP11]). Es ist im Allgemeinen nicht zu erwarten, dass die Latenz durch diese Form der Aufteilung sinkt. Vorteile bezüglich der Latenz können sich lediglich ergeben, wenn eine Teilaufgabe auf einer anderen Verarbeitungseinheit wesentlich schneller ausgeführt werden kann (z.B. auf einer GPU). Bei Aufteilung auf gleichartige Verarbeitungseinheiten bleibt die Latenz bestenfalls gleich, verschlechtert sich üblicherweise jedoch aufgrund des Overheads durch zusätzliche Datenübertragungen und zusätzliche Wartezeiten. Hauptvorteil einer solchen Aufteilung ist jedoch, dass der Durchsatz steigen kann, also die Anzahl der Bilder, die pro Zeiteinheit verarbeitet werden können. Theoretisch wäre es möglich, bei Verteilung auf N Recheneinheiten den Durchsatz um den Faktor N zu steigern. In diesem Idealfall wären immer N Bilder gleichzeitig in Bearbeitung. Bei der Betrachtung eines Bildes und des Verarbeitungsweges ist jedoch festzustellen, dass dieses immer nur in einer Stufe zur Zeit bearbeitet wird und sich die Gesamtlatenz nicht verbessert.

Die Auswirkungen einer verteilten gegenüber einer nicht verteilten Ausführung sollen anhand eines synthetischen Beispiels[2] genauer gezeigt werden. Es werden zwei Filter-Operationen betrachtet, die hintereinander ausgeführt werden. Um den Idealfall der Parallelisierung zu zeigen, wird zweimal der identische Canny-Filter aus dem „GstCV“-Element verwendet. Das Testbild mit FullHD-Auflösung wird mit einer Bildwiederholrate von 60 Bildern pro Sekunde gesendet. Die Anwendung der beiden Filter erfolgt im ersten

[2]Dieses Beispiel stellt keine nutzbare Funktion bereit. Es wird lediglich ein Testbild erzeugt und zweimal hintereinander gefiltert

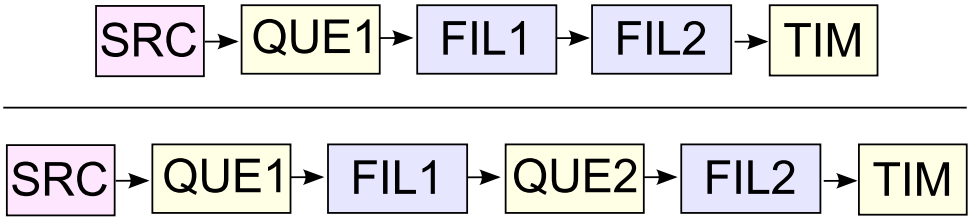


Abbildung 4.8.: Dieses Beispiel zeigt den Pipelineaufbau bei der sequentiellen Ausführung von zwei Filteroperationen. Die beiden Pipelines unterscheiden sich lediglich dadurch, dass beim zweiten Test (unten) zwischen den Elementen zur Filterung noch ein „Queue“-Element platziert ist. Erklärung der Elemente:
SRC: erstellt ein Videotestsignal
QUE: „Queue“-Elemente zur Zwischenspeicherung der Daten und zum Starten eines neuen Threads
FIL: Elemente zur Filterung, hier vom Typ „GstCV“
TIM: Elemente zur Zeitmessung

dieser Tests innerhalb eines Threads, im zweiten Test in seperaten Threads. Der Pipelineaufbau ist in Abbildung 4.8 gezeigt. Bei der zweiten Pipeline sorgt ein „Queue“-Element zwischen den beiden Elementen zur Filterung dafür, dass der zusätzliche Thread erstellt wird.

Da zum Testzeitpunkt keine frei programmierbare Kamera mit einem Mehrkernprozessor verfügbar war, wurden die Tests auf einem Intel Core i5 Prozessor mit vier Kernen ausgeführt. Die Ergebnisse der Tests sind in Abbildung 4.9 grafisch dargestellt. Es zeigt sich, dass bei der Verarbeitung innerhalb eines Threads Daten verworfen werden und nur etwa jedes zweite Bild verarbeitet werden kann. Bei der Aufteilung in zwei Threads können alle Bilder verarbeitet werden. Dies liegt daran, dass beide Elemente zur Filterung parallel arbeiten und mit der Verarbeitung des nachfolgenden Bildes schon begonnen wird, obwohl die des aktuellen Bildes noch nicht abgeschlossen ist.

Die Verarbeitungszeit eines einzelnen Bildes wird durch die Verteilung auf zwei Threads nicht verbessert. Sie wird sogar, wie die folgenden Messwerte zeigen, geringfügig erhöht:

- Latenz 28,3 ms, Wiederholrate 36,0 fps (ein Thread)
- Latenz 30,3 ms, Wiederholrate 60,0 fps (zwei Threads)

Durch die Steigerung der Bildwiederholrate wird jedoch das mittlere Alter der aktuell auf einem System verfügbaren Bildinformationen gesenkt.

Abwechselnde Verteilung auf die Verarbeitungseinheiten

Bei der zuvor erwähnten Aufteilung der Pipeline ist stets erforderlich, dass die Bildverarbeitungsstrategie eine sinnvolle Zerlegung in Teilschritte zulässt. Auch sollten die

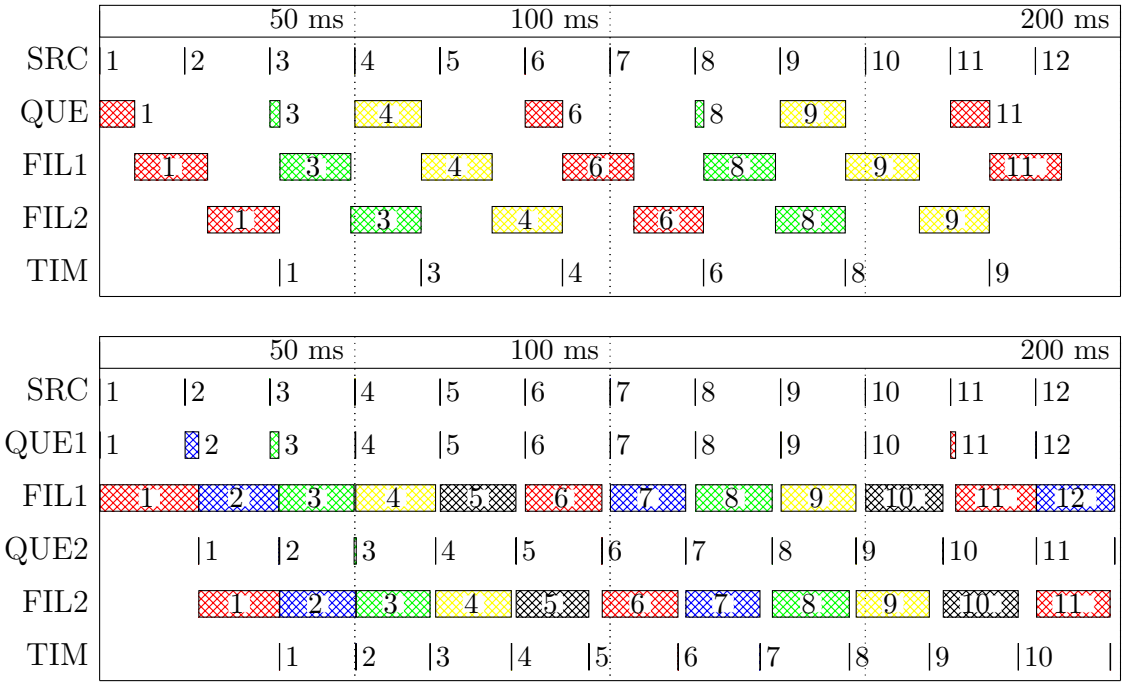


Abbildung 4.9.: Dieses Beispiel zeigt die sequentielle Ausführung von zwei Filteroperationen. Oben werden beide Filter innerhalb eines Threads ausgeführt, unten in zwei verschiedenen Threads. Es zeigt sich, dass durch die Parallelverarbeitung die Wiederholrate gesteigert werden kann. Diese Diagrammform lässt erkennen, zu welchem Zeitpunkt und wie lange von den einzelnen Elementen der Pipeline Daten verarbeitet werden. Die fortlaufende Nummer der Bilddaten ist an den Stellen vermerkt, wo dies im Hinblick auf verfügbaren Platz und Übersichtlichkeit sinnvoll ist. Zur einfacheren Orientierung ist die Farbe in Abhängigkeit von der Bildnummer gewählt. Erklärung der Elemente s. Abbildung 4.8

Teilschritte von der Ausführungszeit her ähnlich sein, damit die Latenz nicht signifikant ansteigt. Diese Annahmen sind in der Praxis nicht immer gegeben.

Eine weitere Methode der Parallelverarbeitung besteht darin, die eingehenden Bilddaten abwechselnd auf verschiedene Verarbeitungseinheiten zu verteilen.

Anhand der folgenden Untersuchungen soll die abwechselnde Zuweisung auf verschiedene Recheneinheiten gezeigt werden. Bei diesen Tests wird der SIFT-Vektor eines Videobildes berechnet. Dieser Algorithmus ist eine Teilaufgabe bei der SIFT-basierten Objekterkennung und wird hier gewählt, weil der Rechenaufwand im Vergleich zur Größe der Bilddaten sehr groß ist. So lassen sich die Auswirkungen der Parallelisierung besonders gut zeigen.

Die Tests werden wieder mit dem oben erwähnten PC mit Intel Core i5 Prozessor mit vier Kernen ausgeführt. Zunächst erfolgt die nicht parallelisierte Ausführung des Algorithmus. In der zugehörigen Pipeline wird hinter das Kamera-Quellelement ein „Queue“-Element gesetzt, das die eingehenden Daten der Kamera zwischenspeichert bzw. diese verwirft, wenn bereits neuere Daten verfügbar sind. Der Aufbau der Pipelines für die nicht parallelisierte Verarbeitung, für die Verteilung auf zwei Threads und für die Verteilung auf vier Threads ist in Abbildung 4.10 gezeigt.

Das in Abbildung 4.11 dargestellte Diagramm für die nicht-parallelisierte Verarbeitung ist folgendermaßen zu interpretieren: Beim Element „CAM“ bedeuten die einzelnen Striche die Aufnahmezeitpunkte der Bilder. Pro 100 ms werden sechs Bilder bereitgestellt, was genau der Bildwiederholrate von 60 fps entspricht. Bei der Darstellung der Bilddaten und deren Verweildauer in der „Queue“ („QUE“) werden nur die Bilddaten erfasst, die nicht verworfen werden. In dem Beispiel werden etwa drei Viertel der Bilddaten verworfen. Das Verwerfen erfolgt immer dann, wenn neuere Bilddaten eintreffen, die alten aber noch nicht an das nächste Element weitergeleitet werden konnten. Das Verwerfen der Bilddaten verhindert, dass Rechenzeit zur Berechnung veralteter Informationen verschwendet wird. Ebenfalls würde ohne Verwerfen der Daten bei diesem Beispiel der Speicherplatzbedarf des Programms immer weiter ansteigen. Das nächste Element „CSP“ dient zur Umwandlung des Farbraumes. Da die Rechenzeit um Größenordnungen geringer ist die des „SIFT“-Elements, kann die Verarbeitungsdauer anhand der Grafik nicht abgelesen werden. Im „SIFT“-Element erfolgt nun die eigentliche Berechnung. Es kann gezeigt werden, dass das System quasi permanent Daten in diesem Element berechnet. Dies führt zur vollständigen Auslastung eines CPU-Kerns. Hier zeigen sich die Nachteile der sequentiellen Verarbeitung. Während andere Recheneinheiten des Systems möglicherweise unausgelastet sind, müssen Daten verworfen werden. In Abbildung 4.12 ist das Alter der aktuell verfügbaren Bildinformationen gezeigt. Diese Grafik korrespondiert zeitlich mit Abbildung 4.11. Mit fortlaufender Zeit erhöht sich das Alter der letzten verfügbaren Daten, beim Eintreffen neuer Daten sinkt dieser Wert auf die Latenz der aktuellen Daten, die von der Verarbeitungszeit abhängt. Durch diese Art der Darstellung kann neben der Latenz einzelner Bilddaten auch der Einfluss der Wiederholrate aufgezeigt werden.

Im nächsten Beispiel werden die Bilddaten abwechselnd in einem von zwei Threads weiterverarbeitet. Das Ergebnis dieses Tests ist in Abbildung 4.13 dargestellt. Es zeigt sich,

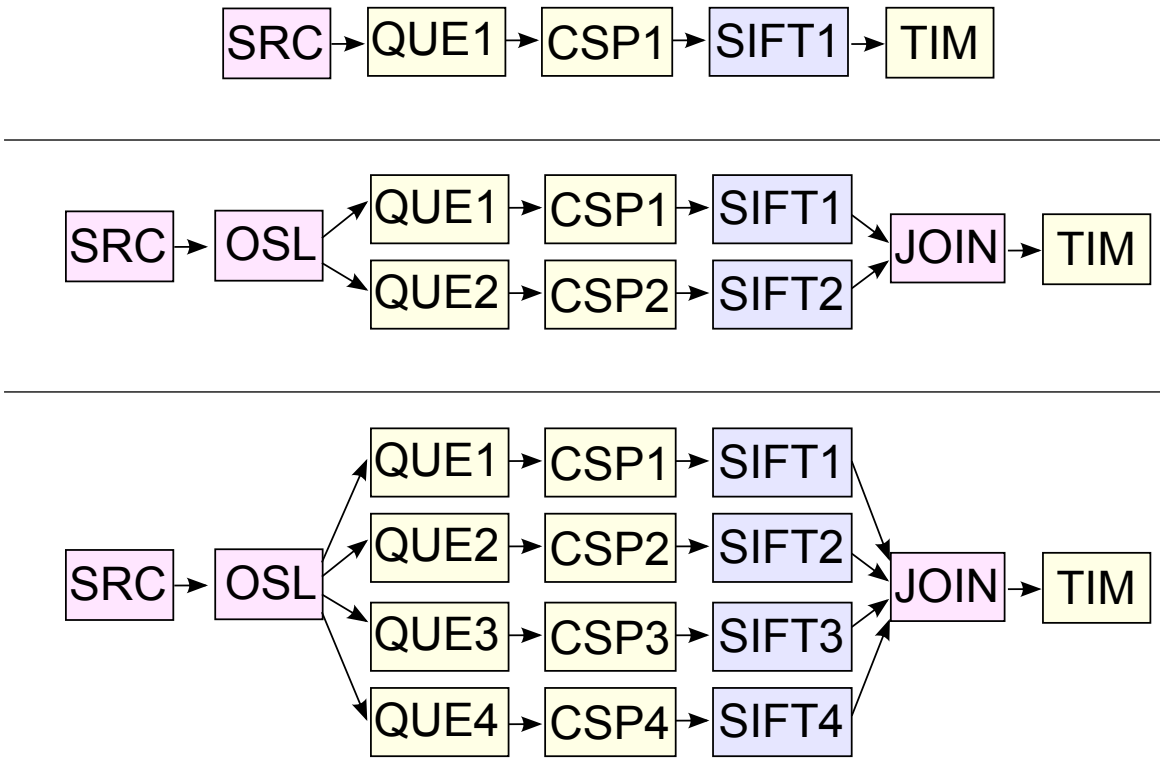


Abbildung 4.10.: Diese Abbildungen zeigen die Pipelines zur Berechnung des Feature-Vektors. In der oberen Abbildung erfolgt keine Verteilung auf verschiedene Threads.
Die mittlere Abbildung zeigt die abwechselnde Verteilung auf zwei, die untere auf vier Threads. Die Bilddaten mit der Größe von 640 x 480 Pixeln werden im YUV-Format mit einer Bildwiederholrate von 60 fps erzeugt. Die Ausführung erfolgt auf einem Prozessor mit 4 Kernen (Intel(R) Core(TM) i5-3570, 3,4 GHz). Das Element Output-Selector (OSL) verteilt die eingehenden Bilddaten auf einen der zwei bzw. vier Verarbeitungswege. Es kann davon ausgegangen werden, dass das Betriebssystem dafür sorgt, dass die Threads auch auf die verfügbaren Rechenkerne verteilt werden. Im Element Input-Selector(JOIN) werden die Bilddaten zusammengeführt. Veraltete Daten werden bei der Latenzmessung (TIM) verworfen.

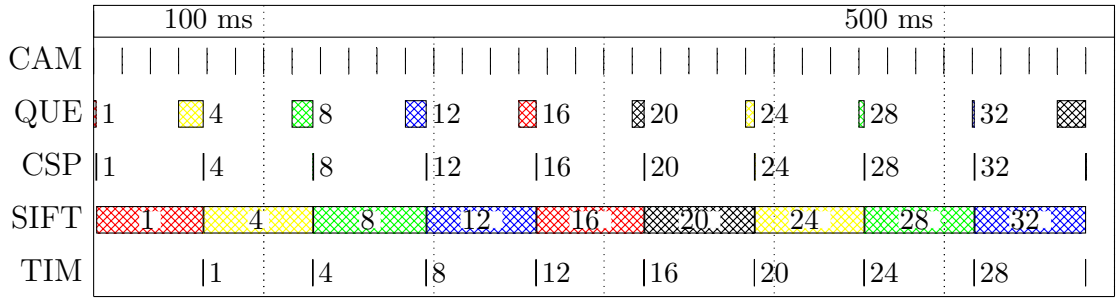


Abbildung 4.11.: Dieses Beispiel zeigt zum Vergleich die Ausführung der SIFT-Berechnung ohne Verteilung auf verschiedene Threads.

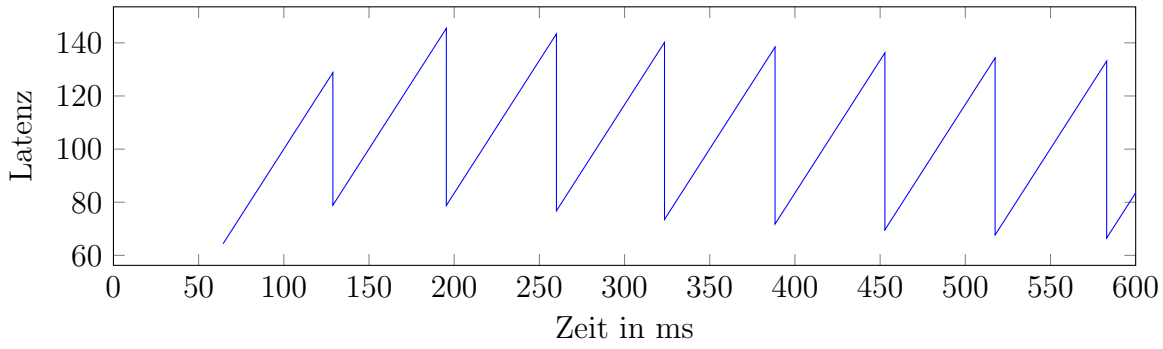


Abbildung 4.12.: Dieses Diagramm zeigt das Alter der neusten aktuell verfügbaren Daten bei der nicht parallelisierten Verarbeitung. Bei jedem Eintreffen neuer Daten verringert sich dieser Wert und erhöht sich bis zum Eintreffen der nächsten Daten.
Durchschnittliches Alter der Bildinformationen: 104 ms
Durchschnittliche Latenz beim Eintreffen: 72 ms
Wiederholrate: 16 fps

dass die Verteilung grundsätzlich funktionsfähig ist und zu dem gewünschten Ergebnis führt. Es ist ersichtlich, dass die Verarbeitung parallel erfolgt. In dem dargestellten Zeitfenster beginnen ungünstigerweise beide Elemente zur Berechnung des SIFT-Vektors ihre Verarbeitung nahezu zeitgleich. Hierbei kann es auch dazu kommen, dass sich Bilddaten „überholen“, d.h. neuere Daten werden schneller verarbeitet als alte Daten. In diesem Fall müssen die zu spät eintreffenden veralteten Bilddaten verworfen werden. Somit stellt dieses Verhalten eine Verschwendung von Rechenzeit dar. Auch der Fall, dass alte Bilddaten noch kurz vor den neueren Daten eintreffen, ist ungünstig, da die alten Daten dann ebenfalls bereits nach kurzer Zeit verworfen werden müssen. Bezüglich des Alters der Informationen, das im zeitlichen Verlauf in Abbildung 4.14 dargestellt ist, ergeben sich keine Vorteile gegenüber der Verarbeitung mit einem Thread. In bestimmten Zeitspannen ist das maximale Alter der Bilddaten sogar schlechter als bei der Verarbeitung mit einem Thread. Dies hat folgenden Grund: Die maximale Verweilzeit der Daten in den beiden „Queue“-Elementen ist größer als bei der Verarbeitung mit einem „Queue“-Element, weil die Bilder pro „Queue“ nur noch mit der halben Wiederholrate eintreffen. Daher werden alte Daten später durch neue Daten verdrängt. In anderen Zeitbereichen ergibt sich dieses Verhalten nicht, sodass sich Vorteile durch die gewählte Verteilung ergeben. Ein solches Zeitfenster ist für die Verarbeitung mit zwei Threads in Abbildung 4.15 gezeigt.

Der Test wird nun mit einer Verteilung auf vier Threads durchgeführt (Pipelineaufbau s. Abbildung 4.10). Die genaue Analyse für einen Zeitbereich ist in Abbildung 4.16 gezeigt. Der unerwünschte Effekt der Verweildauer in den „Queue“-Elementen tritt hier noch stärker hervor. Es kommt zu unregelmäßigem Eintreffen von Bilddaten und zu häufigem Verwerfen von Bilddaten. Der zeitliche Verlauf des Alters der Bildinformationen ist in Abbildung 4.17 dargestellt. Es wird sichtbar, dass die Wiederholrate weiter steigt, jedoch auch die maximale und durchschnittliche Latenz der Daten.

Die bisher analysierten Konfigurationen zur Parallelisierung zeigen zum einen ein großes Potential zur Leistungssteigerung, zum anderen tritt jedoch auch das Problem der erhöhten Latenz hervor. Bei den untersuchten Beispielen ergibt sich die ungünstige Situation, dass nicht alle Bilddaten verarbeitet werden können und es zur Zwischenspeicherung der Daten in den „Queue“-Elementen kommt. Für den Fall, dass die Verarbeitungsleistung groß genug ist, um alle Bilddaten zu verarbeiten, ist mit deutlich geringeren Latenzen zu rechnen, da der Thread sich dann bei der Übergabe der Bilddaten im Leerlauf befinden sollte und die Verarbeitung sofort beginnen kann. Dieses ist in der Praxis aber nicht immer sichergestellt. Insbesondere kann sich die zur Verfügung stehende Rechenleistung durch eine veränderte Lastsituation des Systems verringern. Es soll nun eine Strategie untersucht werden, bei der die Latenz der einzelnen Bilddaten möglichst nicht ansteigt, jedoch die Vorteile des erhöhten Durchsatzes zum Tragen kommen.

Abwechselnde Verteilung - Latenzoptimierung

Ein Ansatzpunkt für die Optimierung liegt in der Zwischenspeicherung der Daten in den „Queue“-Elementen. Bei den beschriebenen Experimenten zeigt sich, dass es zu erhöhten

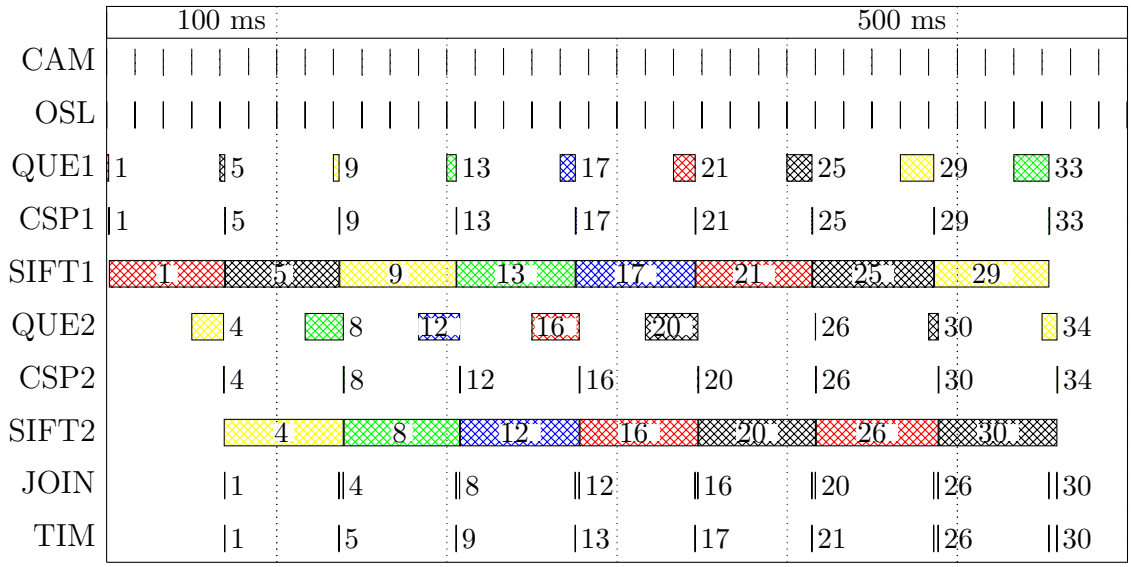


Abbildung 4.13.: Dieses Beispiel zeigt die Verteilung der SIFT-Feature-Berechnung auf zwei Threads. Von dem Element Output-Selector (OSL) werden die Bilddaten abwechselnd zu den Elementen QUE1 und QUE2 geleitet und in dem jeweiligen Thread weiterverarbeitet. In dem hier dargestellten Zeitfenster ergibt sich die ungünstige Situation, dass die beiden SIFT-Berechnungselemente fast synchron laufen. Es kommt sogar dazu, dass die neueren Bilddaten schneller verarbeitet werden als die älteren und diese daher im Element TIM verworfen werden. (z.B. Nr. 4/5)

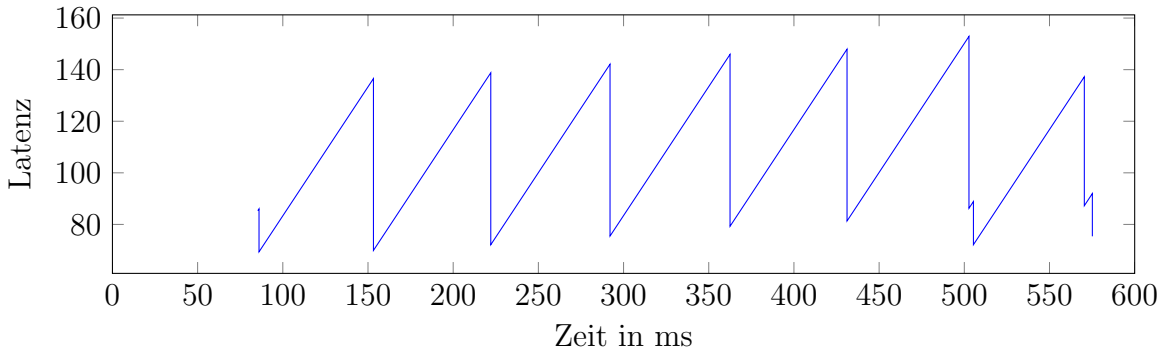


Abbildung 4.14.: Dieses Diagramm zeigt das Alter der neusten aktuell verfügbaren Daten bei der Verteilung auf zwei Threads. Am Ende des Zeitfensters ist zu erkennen, wie zwei Ergebnisse schnell nacheinander eintreffen.
Durchschnittliches Alter der Bildinformationen: 108 ms
Durchschnittliche Latenz beim Eintreffen: 86 ms
Wiederholrate: 27 fps (29 fps vor Verwerfen)

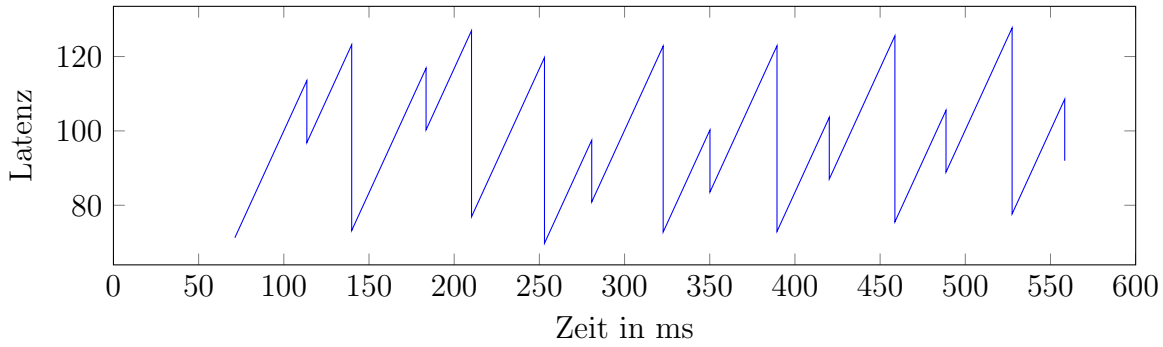


Abbildung 4.15.: Dieses Diagramm zeigt ebenfalls die SIFT-Vektor Berechnung mit 2 Threads, bezieht sich jedoch auf einen anderen Zeitbereich. Das Diagramm korreliert nicht mit Abbildung 4.13. Es ist zu erkennen, dass regelmäßig aktualisierte Daten eintreffen. Dieses führt zu einem geringeren maximalen Alter der Bilddaten, auch im Vergleich zur Verarbeitung in einem Thread. Die Verteilung ist jedoch weiterhin nicht optimal, da die Eintreffzeitpunkte nicht ganz gleichmäßig sind und die Verweildauer der Daten in der „Queue“ negativen Einfluss auf das Alter der Bilddaten hat.

Latenzen kommt, wenn die Verarbeitung länger dauert als das von der Bildwiederholrate vorgegebene Intervall. Dies liegt darin begründet, dass die Daten eine längere Verweildauer in der „Queue“ haben. Aufgrund der Aufteilung auf mehrere Threads sinkt die effektive Bildwiederholrate bei den einzelnen „Queue“-Elementen und alte Daten werden später verdrängt. Bei der Verarbeitung in nur einem Thread entspricht die maximale Verweildauer genau der Zeitspanne zwischen zwei Ausgangsbildern. Eine mögliche Lösung für dieses Problem wäre, eintreffende Daten immer dann sofort zu verwerfen, wenn auf dem Verarbeitungspfad die Verarbeitung der vorherigen Daten noch nicht abgeschlossen ist. Eine naheliegende Realisierung wäre, die „Queue“-Elemente aus der Pipeline zu entfernen. Jedoch kann in dem GStreamer-Framework nur durch die „Queue“-Elemente erreicht werden, dass die Verarbeitung auf verschiedene Threads verteilt wird.

Latenzoptimierung über modifizierte Verteilungsfunktion Eine einfache Form der Optimierung betrifft die Verteilungsstrategie des „Output-Selectors“ auf die einzelnen Threads. Neben den Modi „Abwechselnde Aktivierung der Ausgänge“ und „Wahl des Ausganges mit dem geringsten „Queue“-Füllstand“ wird nunmehr eine weitere Strategie implementiert, die vordergründig betrachtet nicht der gängigen Logik entspricht. Diese Strategie arbeitet folgendermaßen:

- Es wird der Füllstand des „Queue“-Elements geprüft, an das die letzten Daten gesendet wurden.
- Ist dieser gleich eins, erfolgt die erneute Wahl dieses Ausgangs.
- Ist dieser gleich null, wird der nächste Ausgang verwendet.

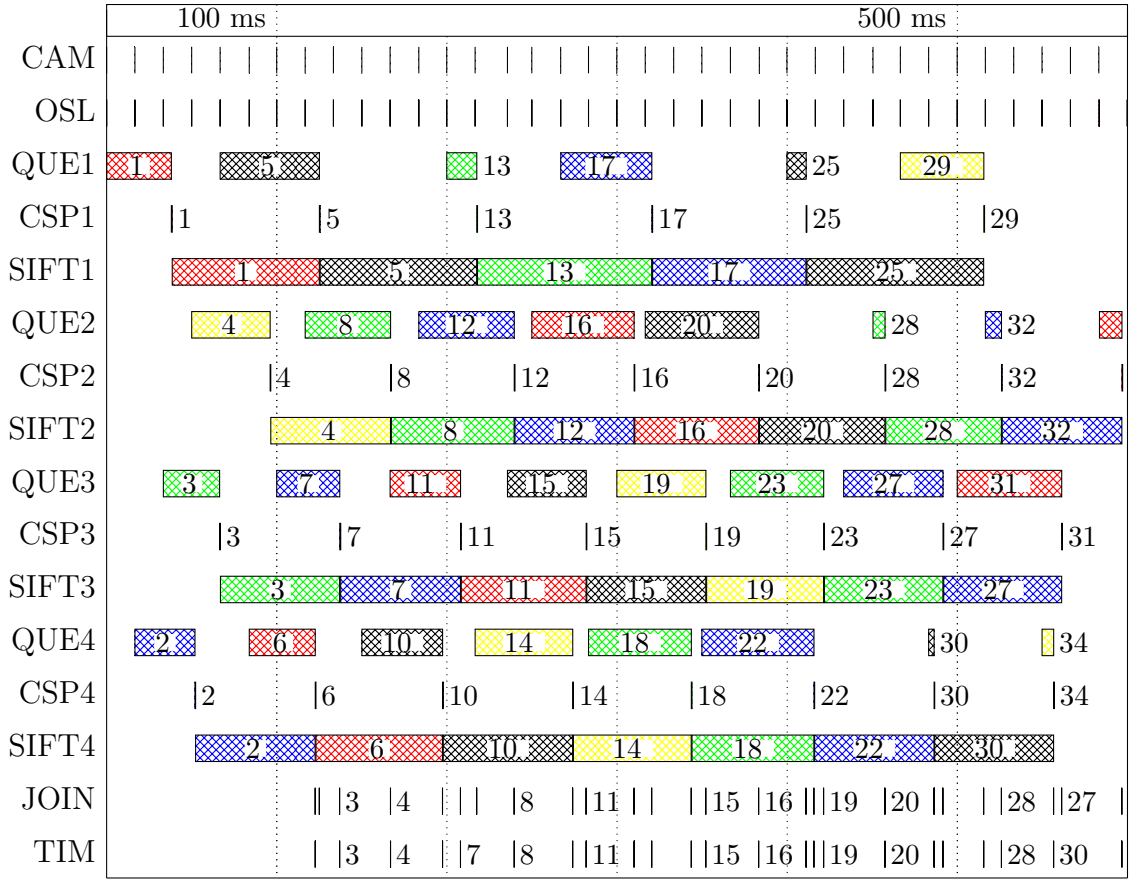


Abbildung 4.16.: Diese Abbildung zeigt die Verteilung der SIFT-Vektor Berechnung auf vier Threads. Es ist zu erkennen, dass die vier Elemente zur SIFT-Berechnung permanent arbeiten. Aus diesem Grund verweilen die Daten in den „Queue“-Elementen. Da die Verarbeitung geringfügig langsamer erfolgt, als neue Daten eingehen, steigt die Verweilzeit stetig an, bis es schließlich zum Verwerfen der Daten kommt (z.B. QUE2, Buffer Nr. 24). Aufgrund der unterschiedlichen Verweilzeiten kommt es auch dazu, dass Buffer sich gegenseitig „überholen“ und deshalb ältere Daten im Element TIM(„Timestamper“) verworfen werden (z.B. Buffer 27/30).

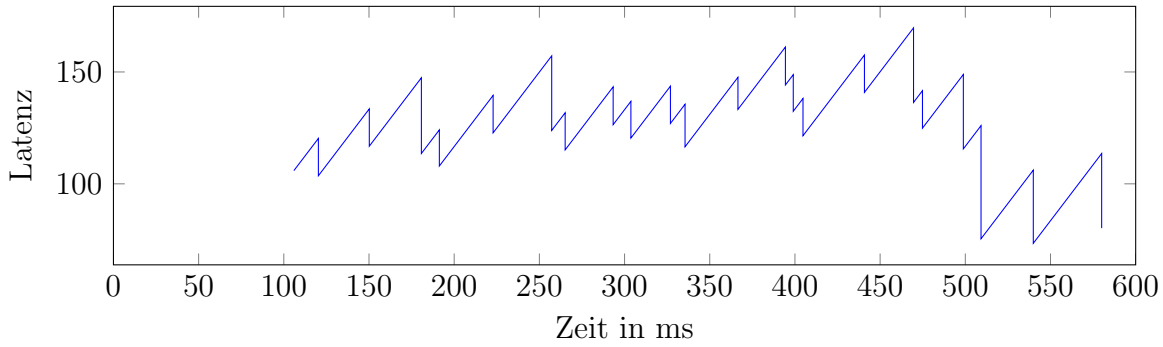


Abbildung 4.17.: Dieses Diagramm zeigt den zeitlichen Verlauf des Alters der Bilddaten bei der sequentiellen Verteilung auf vier Threads. Es ist zu erkennen, dass das Alter der Bildinformationen in weiten Bereichen zwischen 100 und 150 ms schwankt, zeitweise aber auch besser ist.
Durchschnittliches Alter der Bildinformationen: 113 ms
Durchschnittliche Latenz beim Eintreffen: 99 ms
Wiederholrate: 36 fps (53 fps vor Verwerfen) .

Hierdurch wird folgendes Verhalten erzielt: Wenn die „Queue“ einen Füllstand von eins hat, bedeutet dies, dass die Verarbeitung des vorherigen Bildes noch gar nicht begonnen hat und im Moment noch ein älteres Bild verarbeitet wird. Unter der Annahme, dass alle Rechenelemente ungefähr gleich schnell arbeiten, ergibt sich, dass sämtliche anderen Rechenelemente ebenfalls noch belegt sein müssten, da sie ihre Verarbeitung zu einem späteren Zeitpunkt gestartet haben. Durch die Wahl des gleichen Ausganges wird der darin befindliche Buffer verworfen. Auch die anderen „Queue“-Elemente bekommen hierdurch Zeit, den Rückstand aufzuholen. Erst wenn der Füllstand der „Queue“ gleich null ist, bedeutet dies, dass die Verarbeitung begonnen hat. In diesem Fall muss der nächste Ausgang gewählt werden, bei dem die Verarbeitung im Vergleich zu den anderen Ausgängen zuerst gestartet wurde. Somit ist die Chance am größten, dass die Verarbeitung auf diesem Pfad weitgehend oder bereits vollständig abgeschlossen ist.

Als Vorteil ergibt sich, dass die maximale Verweildauer eines Buffers in der „Queue“ gleich dem Eintreffintervall der Quellbilder ist. Der Nachteil ist jedoch, dass die Verarbeitungselemente zeitweise „leerlaufen“ und somit zur Verfügung stehende Ressourcen ungenutzt bleiben. Die Wiederholrate der Daten ist vermutlich bei dieser Konfiguration geringer als bei einem kontinuierlichen Wechsel des Ausganges.

Das Experiment mit Verteilung auf vier Verarbeitungseinheiten wird mit der modifizierten Strategie wiederholt. Das dazugehörige Ablaufdiagramm ist in Abbildung 4.18 dargestellt. Hierbei zeigt sich, dass es nur noch vereinzelt zu dem Verhalten kommt, dass sich Bilder „überholen“. Zudem ist festzustellen, dass die Bilddaten in relativ regelmäßigen Abständen eintreffen. Trotzdem sind die verworfenen Daten als Aussetzer zu erkennen. Des Weiteren ist auszumachen, dass die Auslastung der Verarbeitungselemente nicht mehr zu 100 % gegeben ist. Es kommt hier immer wieder kurzzeitig zum Leerlauf. Insgesamt zeigt das

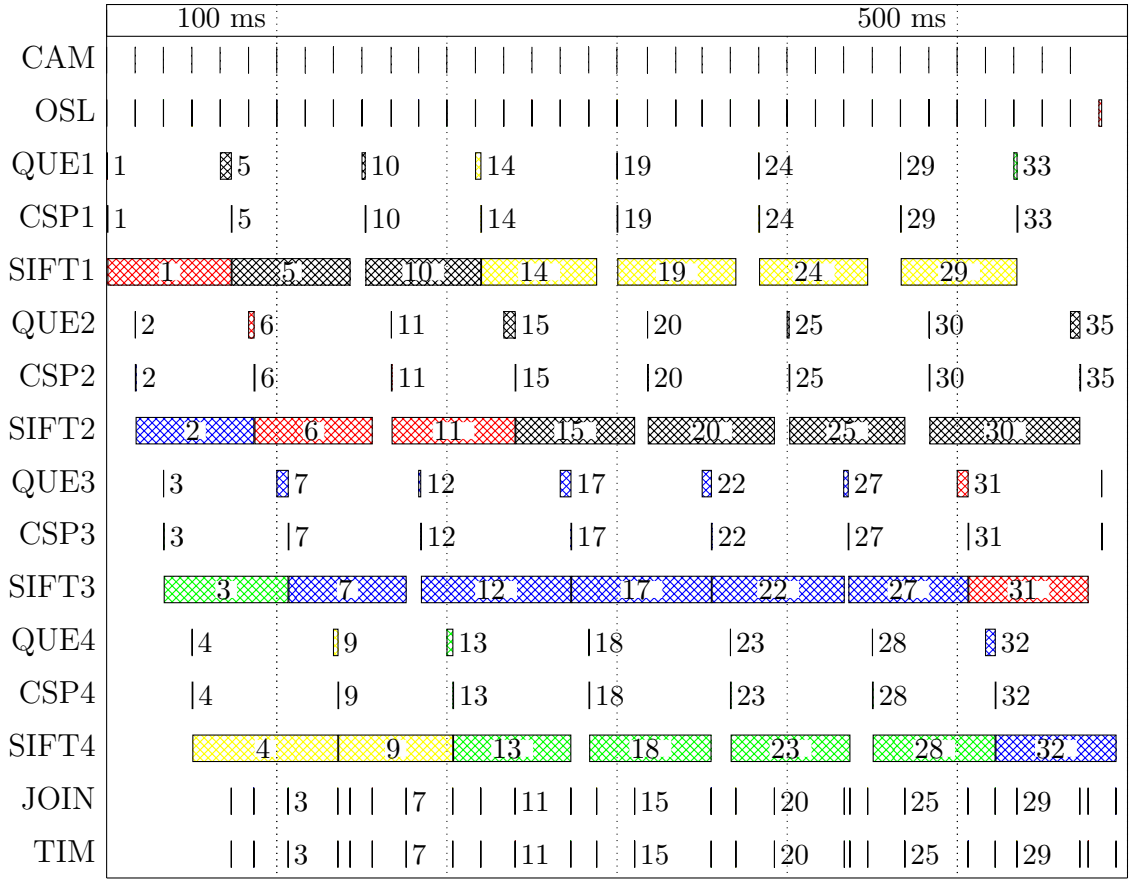


Abbildung 4.18.: Dieses Beispiel stellt die Verteilung der SIFT-Feature-Berechnung auf vier Threads mit modifizierter Verteilungsstrategie dar. Es ist zu erkennen, dass die Verweilzeit in den „Queue“-Elementen geringer ist. Zusätzlich ist die Verarbeitung deutlich regelmäßiger. Dennoch kommt es auch hier zum Verwerfen von Bilddaten (z.B. Nr. 12 und 17, da 13 und 18 schneller verarbeitet werden). Dies liegt daran, dass der Prozessor voll ausgelastet ist und daher die Threads vom Betriebssystem zeitweise unterbrochen werden. Es ist ebenfalls ersichtlich, dass in den Threads zeitweise nicht gearbeitet wird.

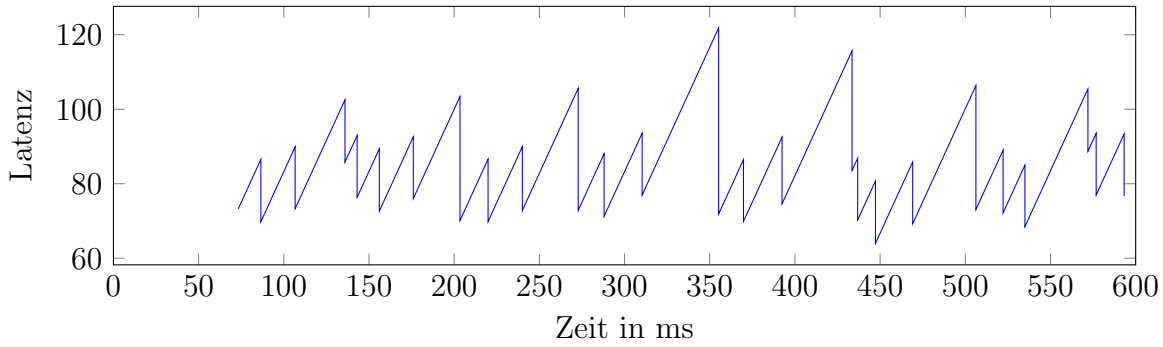


Abbildung 4.19.: Das Alter der Bildinformationen bei modifizierter Verteilungsstrategie. Die Bildinformationen sind im Vergleich zur sequentiellen Verteilung deutlich aktueller. Die absoluten Schwankungen sind ebenfalls bei Berücksichtigung des dargestellten Wertebereichs geringer.
Durchschnittliches Alter der Bildinformationen: 86 ms
Durchschnittliche Latenz beim Eintreffen: 74 ms
Wiederholrate: 48 fps (52 fps vor Verwerfen)

System das gewünschte Verhalten, da die durchschnittliche Latenz deutlich geringer ist (s. Abbildung 4.19).

Latenzoptimierung durch „Trigger"-Elemente Neben der zuvor erwähnten veränderten Verteilungsstrategie kann der Datenfluss auch über die in Abschnitt 3.4 vorgestellte „Trigger"-Elemente gesteuert werden. Diese Elemente können sowohl zur netzwerkübergreifenden als auch wie hier zur lokalen Flusskontrolle verwendet werden. Der entsprechende Aufbau der Pipeline ist in Abbildung 4.20 gezeigt. Durch den Einsatz der „Trigger"-Elemente wird die Zwischenspeicherung quasi komplett unterbunden. Es werden nur Bilddaten auf die Threads verteilt, wenn die Verarbeitung der vorherigen Daten abgeschlossen ist. Eine Auswertung dieser Art der Flusskontrolle ist in dem folgenden Vergleichstest inbegriffen.

Vergleich der Strategien zur Zuweisung Die verschiedenen Verfahren zur Latenzoptimierung werden im Folgenden mit der sequentiellen Verteilung auf die einzelnen Pfade verglichen. Zur besseren Vergleichbarkeit wird ein Bild einer Tischszene (an späterer Stelle in Abbildung 4.27 gezeigt) fortlaufend mit einer Wiederholrate von 60 fps gesendet. Die absoluten Messwerte sind nicht mit denen der vorherigen Tests vergleichbar, da in den vorherigen Tests mit Live-Bildern einer Kamera gearbeitet wurde. Es werden nun die verschiedenen Verteilungsstrategien mit einer unterschiedlichen Anzahl an Threads getestet. Die unterschiedlichen Konfigurationen werden hinsichtlich Latenz und Intervall zwischen dem Eintreffen zweier Dateneinheiten untersucht. Basierend auf diesen Daten wird das

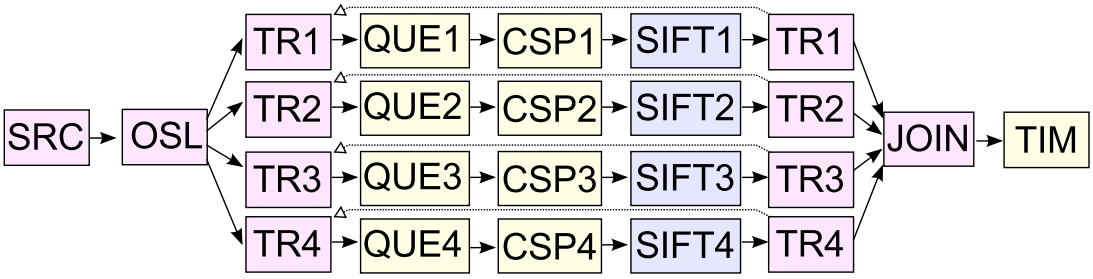


Abbildung 4.20.: Diese Grafik zeigt den Aufbau der Pipeline zur abwechselnden Verteilung auf die Verarbeitungspfade mit den verwendeten „Trigger“-Elementen. Pro Verarbeitungspfad sind zwei „Trigger“-Elemente (TR1-TR4) vorgesehen, die miteinander kommunizieren. Das jeweils erste dieser Elemente verwirft solange alle eintreffenden Daten, bis von dem zweiten Element der Abschluss der Verarbeitung der vorherigen Daten signalisiert wird.

durchschnittliche Alter der Bilddaten rechnerisch nach folgender Formel ermittelt:

$$Alter = Latenz + 0.5 \cdot Intervall$$

Die Ergebnisse dieser Tests sind in den Abbildungen 4.21, 4.22 und 4.23 dargestellt und erörtert. Die Tests ergeben, dass sich die niedrigsten Werte für das mittlere Alter der Bildinformationen mit der Flusskontrolle durch das „Trigger“-Elementepaar und der Verwendung von vier Threads erzielen lassen. Der Wert bei der Verwendung von drei Threads ist jedoch nur minimal schlechter.

Parallelisierung einzelner Operationen

Bei der Parallelisierung einer einzelnen Operation wird versucht, diese in mehrere Teilschritte aufzuteilen, die dann parallel von verschiedenen Verarbeitungseinheiten ausgeführt werden. Im Gegensatz zu den anderen beschriebenen Verfahren zur Parallelisierung wird hierbei direkt der Ablauf des Algorithmus modifiziert. Abhängig von der zugrunde liegenden Operation unterscheiden sich die notwendigen Anpassungen in ihrem Aufwand; eventuell lässt sich eine Parallelisierung auch nicht sinnvoll implementieren. Bei den bisher betrachteten Verfahren zur Parallelisierung war festzustellen, dass die Latenz der einzelnen Dateneinheit bestenfalls gleich bleibt und die Vorteile in der erhöhten Wiederholrate liegen. Bei der Parallelisierung des einzelnen Arbeitsschrittes wird hingegen auch angestrebt, den einzelnen Arbeitsschritt zu beschleunigen.

Die im Rahmen dieser Arbeit implementierte, relativ universell einsetzbare Art der Parallelisierung sieht vor, das Bild in verschiedene Abschnitte zu unterteilen, die dann jeweils unabhängig voneinander bearbeitet werden. Wie in Abschnitt 3.4 geschildert, besteht die Möglichkeit, einen beliebig großen Überlappungsbereich zu definieren, der dann jeweils in

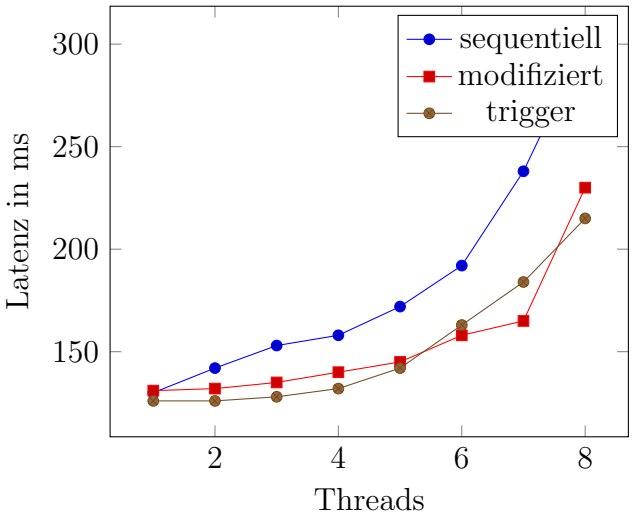


Abbildung 4.21.: Diese Abbildung zeigt die Latenz der Daten zu dem jeweiligen Eintreffzeitpunkt bei den verschiedenen Strategien zur Verteilung auf die einzelnen Verarbeitungspfade. Es ist zu erkennen, dass die Latenz der einzelnen Bilddaten jeweils stark ansteigt, wenn mehr als vier Threads ausgeführt werden. Dies liegt daran, dass die Tests auf einem PC mit einer Vier-Kern CPU durchgeführt werden. Werden mehr Threads ausgeführt, als verfügbare Kerne vorhanden sind, so „teilen“ sich die einzelnen Threads die verfügbaren CPU-Kerne und werden infolgedessen langsamer ausgeführt. Bei der sequentiellen Verteilung kommt noch hinzu, dass die Verweildauer der Daten in der „Queue“ mit der Anzahl der Threads steigt. Bei der Verwendung des „Trigger“-Elementepaars in jedem Verarbeitungspfad ist die Verweildauer im „Queue“-Element zu vernachlässigen, da jeweils nur eine Dateneinheit zur Zeit auf einem Pfad verarbeitet wird.

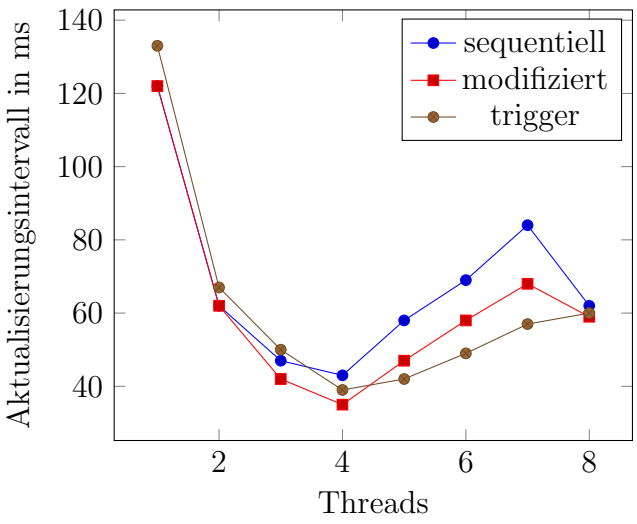


Abbildung 4.22.: Diese Abbildung zeigt die Eintreffintervalle der extrahierten Bildinformationen in Abhängigkeit von der gewählten Strategie und der Anzahl der Threads. Der Wert kann auch als Kehrwert der Bildwiederholrate betrachtet werden. Je niedriger der Wert, desto häufiger treffen neue Bilddaten ein. Theoretisches Minimum ist ein Wert von 16,7 ms, bedingt durch die Bildwiederholrate der Quelle. Es ist zu erkennen, dass bis vier Threads das Eintreffintervall deutlich sinkt und somit die Wiederholrate steigt. Bei mehr als vier Threads sinkt das Eintreffintervall nicht weiter. Im Gegenteil steigt das Intervall sogar an. Wenn alle verfügbaren CPU-Kerne ausgelastet sind, hat die Verteilung auf zusätzliche Threads zwei gegenläufige Effekte: Einerseits erhöht sich wie in Abbildung 4.21 gezeigt die Verarbeitungsdauer, andererseits werden mehr Daten parallel verarbeitet. Letztendlich erzeugt vermutlich der Overhead beim Kontextwechsel in einer CPU die Vergrößerung des Eintreffintervalls. Es ist zu erkennen, dass bei 8 Threads das Intervall wieder sinkt. Dieses Phänomen lässt sich bei mehrfachen Messungen reproduzieren, soll im Rahmen dieser Arbeit aber nicht weiter behandelt werden. Vermutlich kann der Kontextwechsel effizienter ausgeführt werden, wenn sich immer zwei Threads eine CPU teilen. Es ist ersichtlich, dass das Eintreffintervall bei Benutzung der „Trigger“-Elemente zwischen 1 bis 4 Threads größer als bei der modifizierten Verteilung ist. Dies liegt darin begründet, dass keine Zwischenspeicherung in den „Queue“-Elementen erfolgt. Ist die Verarbeitung eines Bildes abgeschlossen, so wird auf das Eintreffen des nächsten Bildes gewartet. Dagegen kann bei der sequentiellen und der modifizierten sequentiellen Verteilung gleich weitergerechnet werden, da in der „Queue“ bereits das nächste Bild gespeichert ist.

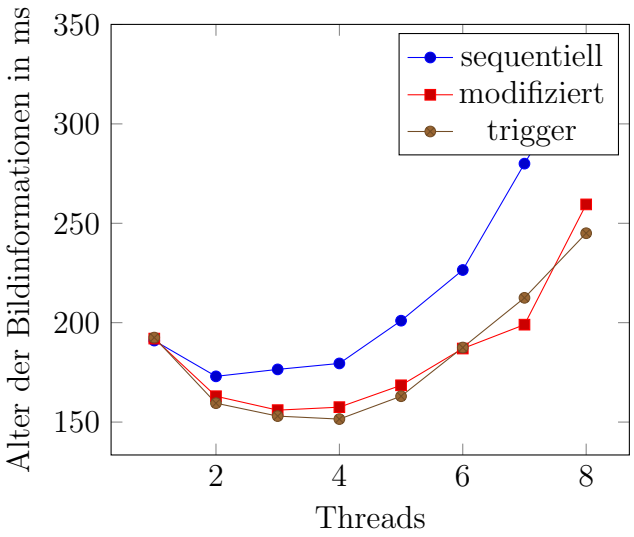


Abbildung 4.23.: Diese Abbildung zeigt das resultierende durchschnittliche Alter der Bildinformationen, das hier rechnerisch aus der Latenz und dem Aktualisierungsintervall ermittelt wird. Es zeigt sich, dass sowohl die modifizierte Verteilungsstrategie als auch die Verwendung der „Trigger“-Elemente deutliche Verbesserungen bewirken. Die niedrigsten Werte können mit der Flusskontrolle durch das „Trigger“-Elementepaar und bei der Verwendung von vier Threads erzielt werden. Der Wert bei der Verwendung von drei Threads ist jedoch nur minimal schlechter, so dass sich die Frage stellt, ob die mögliche Verbesserung den höheren Rechenaufwand rechtfertigt. Insbesondere ist bei der Entscheidung über die Threadanzahl zu bedenken, inwiefern auf dem System noch weitere rechenintensive Algorithmen ausgeführt werden. In diesem Fall sollte eher eine geringere Anzahl an Threads gewählt werden. Es ist in dieser Grafik ebenfalls deutlich zu erkennen, dass die Verwendung einer Thread-Anzahl, die höher ist als die Anzahl der (frei verfügbaren) CPU-Kerne, keinen Sinn ergibt.

zwei benachbarten Bildabschnitten abgebildet ist. Für den Fall, dass es sich bei der Ausgabe der Algorithmen um Bilddaten handelt, werden die bearbeiteten Abschnitte wieder zu einem Bild zusammengefügt und dabei die Überlappungsbereiche entfernt.

Es ist ebenfalls das Zusammenfügen von mehreren SIFT-Feature-Vektoren implementiert. Hierbei werden jeweils zwei Vektoren der Features zu einem fusioniert. Die Lage der Features im Bild muss umgerechnet werden und es müssen doppelt vorhandene Features aus den überlappenden Bereichen entfernt werden. Die Parallelisierung der SIFT-Feature-Berechnung liefert möglicherweise veränderte Ergebnisse gegenüber der nicht parallelisierten Ausführung. Dies liegt darin begründet, dass die SIFT-Features anhand ihrer Umgebung beschrieben werden. Ist die Umgebung nicht vollständig abgebildet, variiert die Beschreibung. In der Praxis zeigen sich bezüglich der Erkennungsgenauigkeit keine negativen Auswirkungen. Dies bestätigen auch Tests, in denen sich der SIFT-Algorithmus als tolerant gegenüber partieller Verdeckung erweist. Zu Problemen könnte es dann kommen, wenn eine extrem starke Skalierung eines Features vorliegen und dieses Feature sich über mehrere Teilbilder erstrecken würde. In diesem Fall könnte dieses Feature nicht mehr korrekt erkannt werden.

Bei dem hierzu durchgeführten Experiment wird die Berechnung des SIFT-Feature-Vektors eines Bildes auf vier Threads verteilt. Bei diesem Test wird ebenfalls die Tischszene analysiert, auf der auch die Tests in Abbildung 4.21 bis 4.23 beruhen.

Die Bildverarbeitungsstrategie wird wieder auf dem Quad-Core-Prozessor ausgeführt, sodass ausreichend Recheneinheiten für die Ausführung zur Verfügung stehen. Die Konfiguration der Bildverarbeitungspipeline ist in Abbildung 4.24 dargestellt und beschrieben. Es wurde bei diesem Beispiel von vornherein die Lösung gewählt, über ein Paar von „Trigger“-Elementen den Datenfluss durch die Passage zur parallelisierten Berechnung des Feature-Vektors zu steuern. Dies hat mehrere technische Gründe: In der Pipeline sind zum Erzeugen der einzelnen Threads die „Queue“-Elemente notwendig, die in dieser Konfiguration aber keine Daten verwerfen dürfen. Es würde ohne „Trigger“-Elemente zu der Situation kommen, dass stets, wenn die Verarbeitung der einer Dateneinheit startet, die nächste Dateneinheit direkt in der „Queue“ zwischengespeichert werden würde und hier nicht mehr verdrängt werden könnte, auch wenn neuere Daten vorhanden wären. Somit wäre die Latenz unnötig hoch. Ein weiteres Problem wäre, dass bei unterschiedlicher Verarbeitungszeit Daten sich innerhalb der verschiedenen Threads „überholen“ könnten und so nicht zusammengehörende Dateneinheiten fusioniert würden.

Durch die Verwendung der „Trigger“-Elemente-Paars wird jedoch in Kauf genommen, dass die Auslastung der Verarbeitungseinheiten nicht maximal ist und eine geringere Wiederholrate erzielt wird als theoretisch möglich wäre. Die Ergebnisse dieses Tests sind in den Abbildungen 4.25 und 4.26 dargestellt und erläutert. Die reine Ausführungszeit beträgt nicht parallelisiert etwa 120 ms. Es kann somit gezeigt werden, dass durch die Parallelisierung eine deutliche Leistungssteigerung erzielt werden kann. Jedoch skaliert die Verarbeitungsleistung nicht mit der Anzahl der Verarbeitungseinheiten, sondern ist bei vier Threads im Vergleich zur nicht parallelisierten Verarbeitung nur etwa doppelt so hoch.

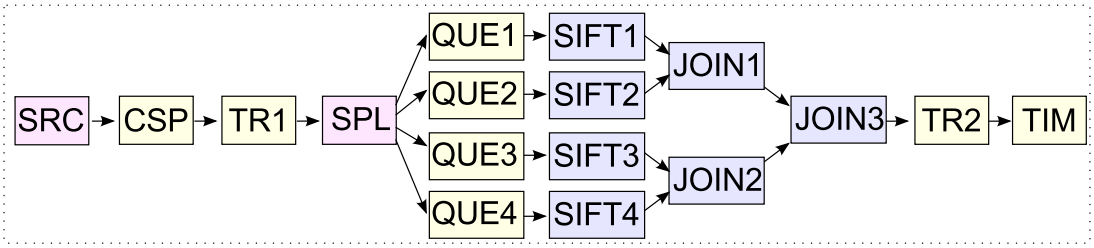


Abbildung 4.24.: Diese Grafik zeigt den Aufbau der Pipeline zur parallelen Berechnung des Feature-Vektors. Die Bilddaten werden zunächst von einem Quellelement bereitgestellt (SRC), das in diesem Fall das Bild aus einer Datei einliest und mit einer Wiederholrate von 60 fps sendet. Zunächst erfolgt die Umwandlung des Farbraumes auf das von den SIFT-Elementen geforderte Eingangsformat (CSP). Als nächstes Element in der Pipeline passieren die Daten das erste der „Trigger“-Elemente (TR1). An dieser Stelle werden die Daten entweder verworfen, falls die Verarbeitung eines vorherigen Bildes noch nicht abgeschlossen ist, oder anderenfalls weitergeleitet. Werden die Daten weitergeleitet, erfolgt die Aufteilung in vier überlappende horizontale Bildabschnitte (SPL). Es wäre ebenfalls möglich, eine vertikale Unterteilung durchzuführen, jedoch ist die horizontale Unterteilung ohne zusätzliche Kopiervorgänge möglich, da einfach vier Zeiger auf verschiedene Bildabschnitte gesetzt werden. Um das GStreamer-System anzuweisen, die Daten in neuen Threads zu bearbeiten, werden die „Queue“- Elemente QUE1- QUE4 in der Pipeline platziert. Hierbei ist von besonderer Bedeutung, dass diese „Queue“-Elemente so konfiguriert werden, dass keine Daten verworfen werden, da beim Zusammenfügen der Daten sonst Teile fehlen würden. Die berechneten Feature-Vektoren werden paarweise von Elementen des Typs „Siftfolder“ zusammengefügt (JOIN1-3). Dieses hat den Vorteil, dass somit auch das Zusammenfügen parallelisiert wird. Zudem werden unnötige Verzögerungen vermieden, da die Daten bei Verfügbarkeit schnellstmöglich zusammengefügt werden. Der hierdurch entstehende Overhead ist gering, da der eigentliche Rechenaufwand durch das Prüfen des Überlappungsbereiches entsteht. Die erzeugten Daten durchlaufen das zweite „Trigger“-Element (TR2), das dann das erste „Trigger“-Element zum Weiterleiten der nächsten Daten veranlasst. Im letzten Pipeline-Schritt erfolgt die Messung der Zeit.

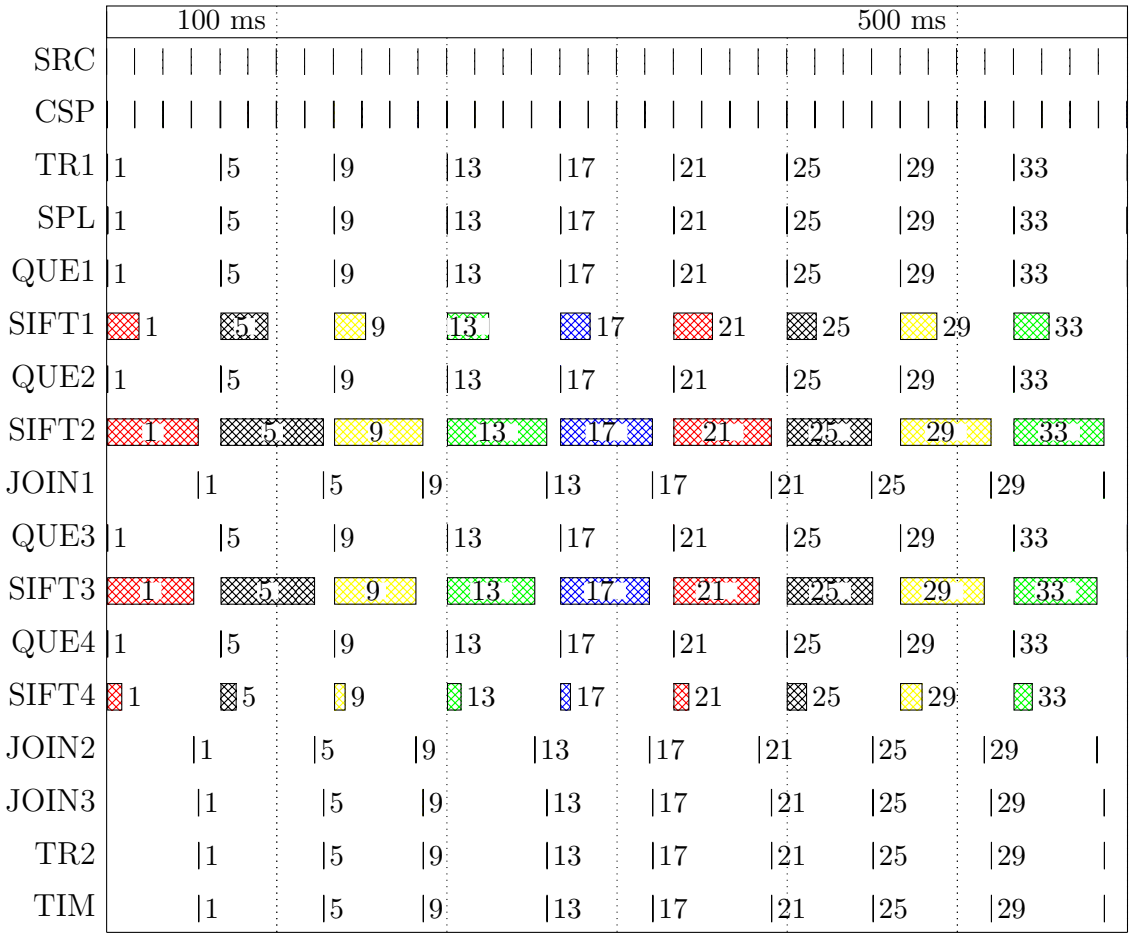


Abbildung 4.25.: Dieses Beispiel zeigt die Parallelisierung des Arbeitsschrittes zur Berechnung des SIFT-Feature-Vektors. Das Ausgangsbild wird in vier überlappende Abschnitte unterteilt (SPL), die dann unabhängig voneinander in verschiedenen Threads verarbeitet werden. An diesem Beispiel kann auch die Funktion der beiden „Trigger“-Elemente (TR1,TR2) gezeigt werden: Es werden stets die eingehenden Daten verworfen, bis die Verarbeitung der vorherigen Daten abgeschlossen ist. Auffällig an diesem Diagramm ist die unterschiedliche Verarbeitungszeit der verschiedenen Elemente zur Feature-Vektor-Berechnung (SIFT1-SIFT4). Als Ursache hierfür stellt sich der unterschiedliche Bildinhalt heraus. Während von den Elementen SIFT2 und SIFT3 jeweils etwa 200 Bildpunkte berechnet werden, werden in SIFT1 nur ca. 80 Bildpunkte detektiert, in SIFT4 keine.

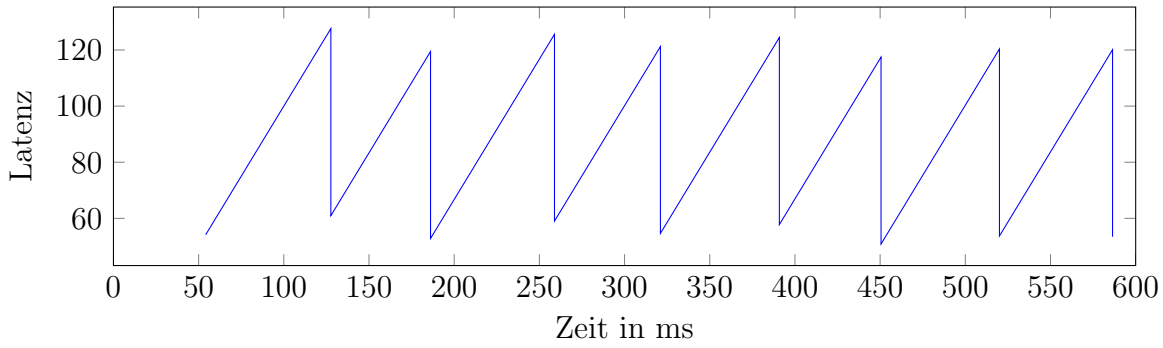


Abbildung 4.26.: Das Alter der Bildinformationen bei paralleler Verarbeitung jeweils eines Bildes. Die Bilddaten treffen regelmäßig, aber mit geringerer Wiederholrate ein.
Durchschnittliches Alter der Bildinformationen: 88 ms
Durchschnittliche Latenz beim Eintreffen: 55 ms
Wiederholrate: 15 fps
Die Verarbeitung des Bildes mit identischem Bildinhalt ohne Parallelisierung dauert etwa 120 ms und führt zu einem durchschnittlichen Alter der Bildinformationen von 190 ms.

Abbildung 4.27.: Die Tischszene mit markierten SIFT-Merkmalen. Es ist zu erkennen, dass in dem oberen der vier horizontalen Abschnitte wenige, in dem unteren Abschnitt keine Merkmale gefunden werden. Es können mehrere Merkmale dicht nebeneinander liegen, so dass diese als ein Merkmal markiert werden. Daher sind weniger Merkmale erkennbar, als bei der Beschreibung zu Abbildung 4.26 angegeben.

Der Grund hierfür liegt hauptsächlich darin, dass die Verarbeitung der einzelnen Bildabschnitte unterschiedlich komplex ist. Dieses ist auch an Abbildung 4.27 zu erkennen, in der die Tischszene mit den detektierten Feature-Punkten dargestellt ist.

Im Vergleich zur abwechselnden Verteilung auf die Verarbeitungseinheiten zeigt das Ergebnis dieses Tests das große Potential der parallelen Verarbeitung von Bildabschnitten eines Bildes. Die parallele Verarbeitung der Bildabschnitte kann auch auf vernetzten Systemen erfolgen. Dieses Verfahren wird hier jedoch nicht gesondert betrachtet, da es Teil des beschriebenen Szenarios in Abschnitt 4.3 ist.

4.1.4. Zusammenfassung der Ergebnisse zur Parallelverarbeitung

Mit den hier beschriebenen Tests kann dargelegt werden, dass von der implementierten Softwarearchitektur verschiedene Formen der Parallelverarbeitung unterstützt werden. Es stellt sich heraus, dass bei Verfügbarkeit von leistungsstarken Systemen mit mehreren CPU-Kernen signifikante Leistungssteigerungen möglich sind. So kann in Bezug auf die Entwicklung zukünftiger intelligente Kamerasysteme gezeigt werden, dass die Verwendung von parallelen Architekturen eine erfolgversprechende Möglichkeit zur Steigerung der Rechenleistung darstellt. Bei der SIFT-Vektor-Berechnung ermöglicht die parallele Verarbeitung eines Bildes die größte Leistungssteigerung. Daher wird dieses Verfahren auch bei dem in Abschnitt 4.3 behandelten Szenario angewendet. Die intelligente Kamera wird dort dazu genutzt, die Bildabschnitte direkt auf verschiedene Systeme zu verteilen.

4.2. Übertragung relevanter Bildbereiche bei Gesichtslokalisation

Bei den folgenden Anwendungsbeispielen steht die Funktion des Kamerasystems im Vordergrund, selektiv bestimmte Regionen des Bildes in voller Auflösung zu übertragen, während andere Bildbereiche nicht oder nur in geringerer Auflösung gesendet werden. Dieser Betriebsmodus erfordert Kamerasysteme mit integrierter Rechenleistung, da nur so die flexible Auswahl von Bildbereichen und die Übertragung über verschiedene Datenpfade möglich wird. Die entwickelte Softwarearchitektur unterstützt diese Verarbeitung der Bilddaten auf verschiedenen Verarbeitungspfaden. Die Auswahl von Bildbereichen kann entweder manuell durch den Nutzer oder auch automatisch über eine Gesichtslokalisation erfolgen. Als grobes Ziel kann sowohl in der Service-Robotik als auch beim Einsatz im Surveillance-Bereich die Reduktion des Datenvolumens bei gleichzeitigem Erhalt der relevanten Bildinformationen gesehen werden.

Mit Gesichtslokalisation ist wie bereits erläutert gemeint, dass Regionen im Bild bestimmt werden, in denen sich beliebige Gesichter befinden. Die Zuordnung des Gesichts zu dem System bekannten Personen soll hier nicht unter diesem Begriff zu verstehen sein. Letzteres

soll als Gesichtsdetektion bezeichnet werden, ist aber im Rahmen dieser Arbeit nicht implementiert. Die Gesichtslokalisation wird in allen Tests mit verringerter Auflösung ausgeführt. Die im Rahmen dieser Arbeit durchgeführten Tests zeigen, dass die Erkennung des Vorhandenseins eines beliebigen Gesichtes auch auf Bildern mit verringerter Auflösung zuverlässig möglich ist. Die zuverlässige Bestimmung der Person erfordert nach [UIY06] jedoch Bildmaterial mit möglichst hoher Auflösung.

Versuche zur selektiven Übertragung von Bildregionen wurde bereits zu einem früheren Zeitpunkt durchgeführt und in [BZ09] publiziert. Die dort veröffentlichten Messdaten weichen von den hier ermittelten ab, da ein anderes PC-System mit einem Zweikernprozessor verwendet wurde. Da das dort verwendete System nicht dem Steuerrechner des Service-Roboters entspricht und in der Zwischenzeit einige Optimierungen in das Softwaresystem eingearbeitet wurden, werden hier die aktuellen Versuche und Ergebnisse präsentiert.

4.2.1. Versuchsdurchführung

Ziel der in diesem Abschnitt beschriebenen Versuche ist es, alle Bildausschnitte, in denen ein Gesicht abgebildet ist, in voller Auflösung auf den Steuerrechner des Service-Roboters zu übertragen. Auf diesen Daten basierend kann beispielsweise eine Gesichtsdetektion in einem weiteren Schritt implementiert werden.

Es werden drei verschiedene Testkonfigurationen verwendet, bei denen jeweils zwei Pipelines aufgesetzt werden, und zwar eine auf dem intelligenten Kamerasystem Basler eXcite und eine auf dem Steuerrechner des Service-Roboters (s. Abbildung 4.28). Der Datentransfer zwischen den Systemen wird über TCP-Plugins realisiert.

In der ersten Konfiguration wird die Kamera wie eine konventionelle Ethernet-Kamera betrieben. Somit werden alle Bilddaten mit voller Auflösung auf den Steuerrechner übertragen, auf dem dann die Gesichtslokalisation und das Ausschneiden der relevanten Bildregionen erfolgt. Bei dem zweiten Setup wird das Gesamtbild mit verringerter Auflösung (320x240 Pixel) von der intelligenten Kamera zum Steuerrechner übertragen. Dieser führt dann auf diesem Bilddatenstrom die Gesichtslokalisation durch und sendet die Koordinaten der relevanten Bereiche zurück zum Kamerasystem. Über eine weitere Datenverbindung werden dann die relevanten Bildausschnitte in ihrer ursprünglichen hohen Auflösung übertragen. Bei der dritten Konfiguration werden alle relevanten Verarbeitungsschritte auf der intelligenten Kamera durchgeführt, so dass nur die extrahierten Regionen und die Koordinaten der entsprechenden Bereiche übertragen werden.

Die Tests werden unter verschiedenen Rahmenbedingungen durchgeführt. Die gesamte Verarbeitungspipeline wird so konfiguriert, dass Daten verworfen werden, wenn diese nicht rechtzeitig verarbeitet werden können. Daher ergibt sich die Frage, wie sich das System verhalten soll, wenn die Informationen über die Positionen der Gesichter mit verringerter Wiederholrate eintreffen. Es können entweder bei der Extraktion der relevanten Bildbereiche immer die gerade verfügbaren Positionsdaten verwendet werden (keine Synchronisation), oder es kann gewartet werden, bis die Positionsdaten zu dem aktuellen Bild

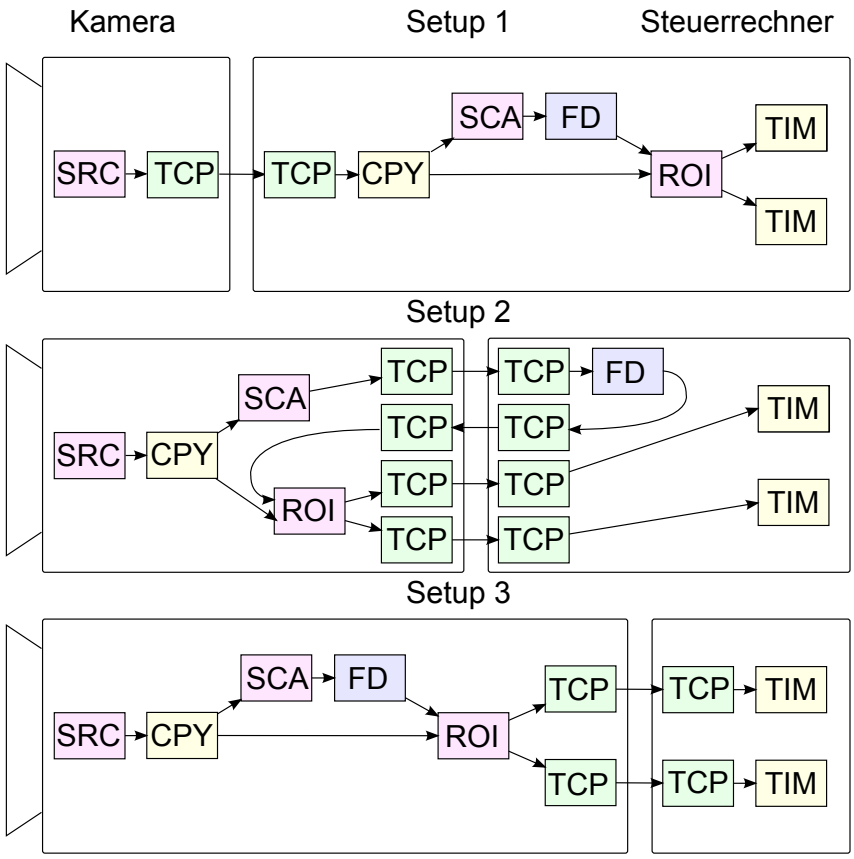


Abbildung 4.28.: Diese Grafik zeigt die Konfiguration der Systeme bei der Übertragung von Bildregionen, die Gesichter enthalten.
Legende der verwendeten Elemente:
SRC: Kapselt den Treiber der Kamerahardware
CPY: Dupliziert Daten
SCA: Skaliert Bilder, ändert die Auflösung
FD : Lokalisiert Gesichter mittels Haar-Klassifikatoren, erzeugt Koordinatenliste
ROI: Extrahiert Bildausschnitte und sendet sie sequenziell, sendet zusätzlich aktuelle Koordinaten
TCP: Überträgt Daten per TCP-Verbindung
TIM: Vergleicht Zeitstempel der Daten mit aktueller NTP-Zeit

eintreffen (Synchronisation). In letzterem Fall kommt es regelmäßig dazu, dass Bilddaten verworfen werden, da nicht für jedes Bild die Position der Gesichter rechtzeitig berechnet werden kann. Bei Betrieb ohne Synchronisation kann es jedoch insbesondere bei starken Bewegungen der Personen oder des Kamerasystems dazu kommen, dass sich in der Region wegen veralteter Koordinateninformationen keine Gesichter mehr befinden. Als weitere veränderliche Rahmenbedingung kann bei den Tests eine zusätzliche Last auf dem Steuerrechner des Roboters erzeugt werden, um das Verhalten unter realen Bedingungen zu simulieren. Die Auslastung des Steuerrechners ohne Bildverarbeitungsaufgaben beträgt dann ca. 60 %, was in etwa der Grundauslastung entspricht, die in [BWZ07] gemessen wurde.

Das Ergebnis der Experimente ist in den Abbildungen 4.29 und 4.30 wiedergegeben. Hier sind die Auswirkungen der verschiedenen Setups auf Latenz, Bildwiederholrate und Netzwerkauslastung dargestellt, jeweils über den gesamten Testzeitraum von zwei Minuten gemittelt. In der Szene befindet sich während dieser Zeit ein Gesicht. Es zeigt sich, dass durch eine geeignete Verteilung wie im zweiten Setup Latenz und Netzwerkauslastung deutlich gesenkt werden und die Bildwiederholrate steigt. Bei den Experimenten, bei denen die Koordinateninformationen nicht mit den Bilddaten synchronisiert werden, ergeben sich stets eine hohe Bildwiederholrate und eine geringe Latenz. Die Latenz ist jeweils deutlich geringer, wenn die Erzeugung der Bildausschnitte auf der intelligenten Kamera ausgeführt wird (Setup 2 und 3). Dieser Effekt tritt insbesondere dann hervor, wenn der Steuerrechner zusätzlich ausgelastet ist. Auch ist hervorzuheben, dass bei ausgelastetem Steuerrechner die komplette Verarbeitung auf der Kamera kaum Nachteile hinsichtlich der Latenz gegenüber einer Ausführung aller Schritte auf dem Steuerrechner hat.

Für eine komplette Verarbeitung auf dem Kamerasystem unter strikten Echtzeitbedingungen ist die Basler eXcite noch nicht leistungsfähig genug; es kann aber gezeigt werden, dass die Softwarearchitektur für diesen Betriebsmodus bereits vorbereitet ist. Werden an die Latenz der Daten geringere Anforderungen gestellt, ist jedoch auch diese Konfiguration sinnvoll einsetzbar, insbesondere wenn das Robotersystem eine andere Hauptaufgabe hat.

Wenn diese Testreihe auf modernerer Hardware ausgeführt wird, treten die Vorteile des zweiten Setups noch deutlicher hervor, da der Arbeitsschritt der Gesichtslokalisation schneller ausgeführt werden kann. So nimmt dann die Zeit zur Datenübertragung einen höheren Anteil an der Gesamtlatenz ein. Der prozentuale Vorteil gegenüber dem ersten Setup vergrößert sich somit.

Im folgenden Abschnitt werden einige Ergebnisse anhand von Ablaufdiagrammen genauer analysiert.

4.2.2. Analyse der Ergebnisse

Bei den zuvor beschriebenen Tests fällt insbesondere auf, dass die Latenz bei der verteilten Verarbeitung deutlich besser als bei der Verarbeitung auf dem Steuerrechner ist. Ohne

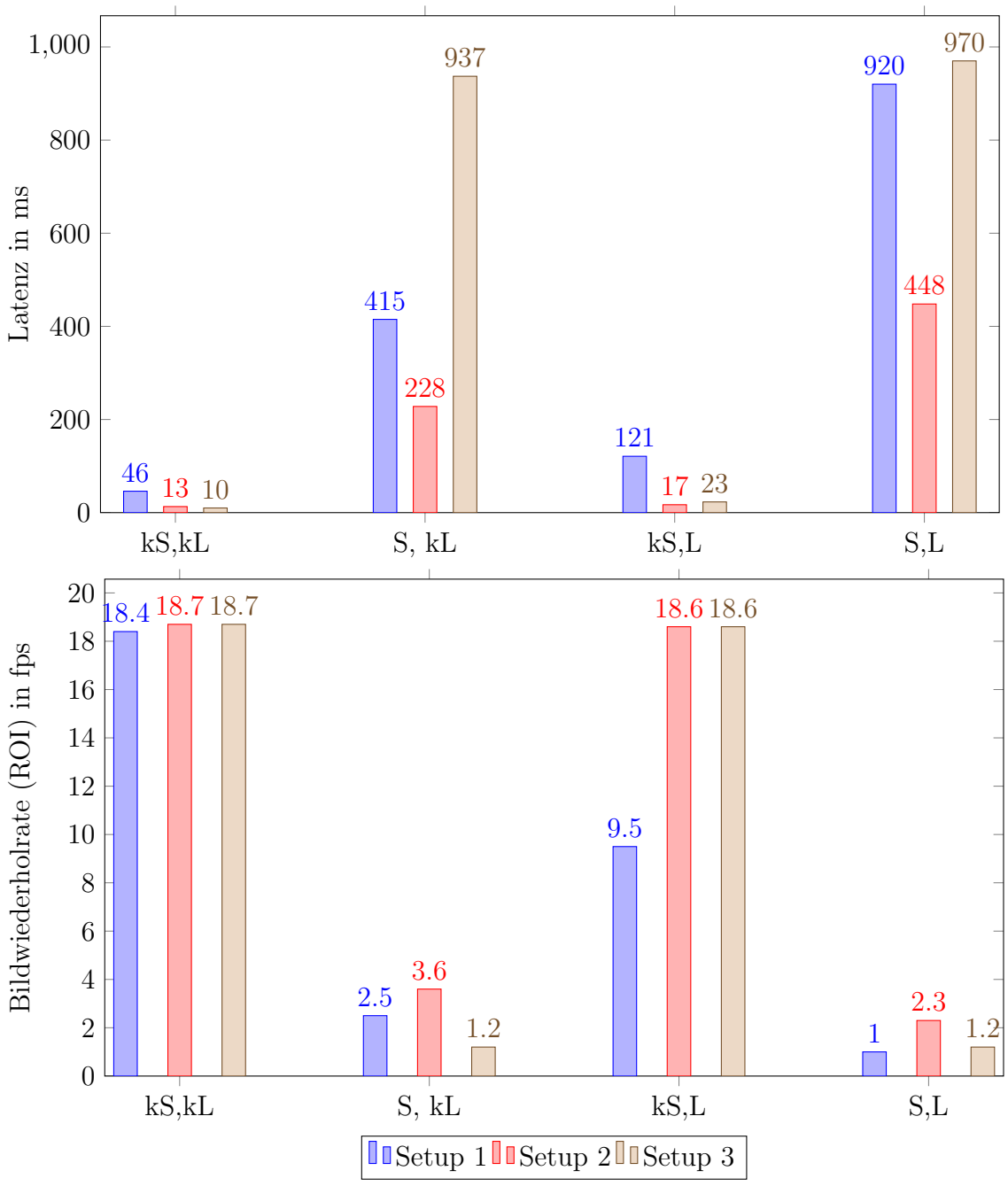


Abbildung 4.29.: Latenz der Bilddaten und Bildwiederholrate für die 3 Setups zur Übertragung von Bildregionen, die Gesichter enthalten. (Legende: S=Synchronisation, kS=keine Synchronisation, L=Last, kL=keine Last)
Setup 1: keine Verarbeitung auf intelligenter Kamera
Setup 2: Verteilte Verarbeitung
Setup 3: alle Arbeitsschritte auf intelligenter Kamera

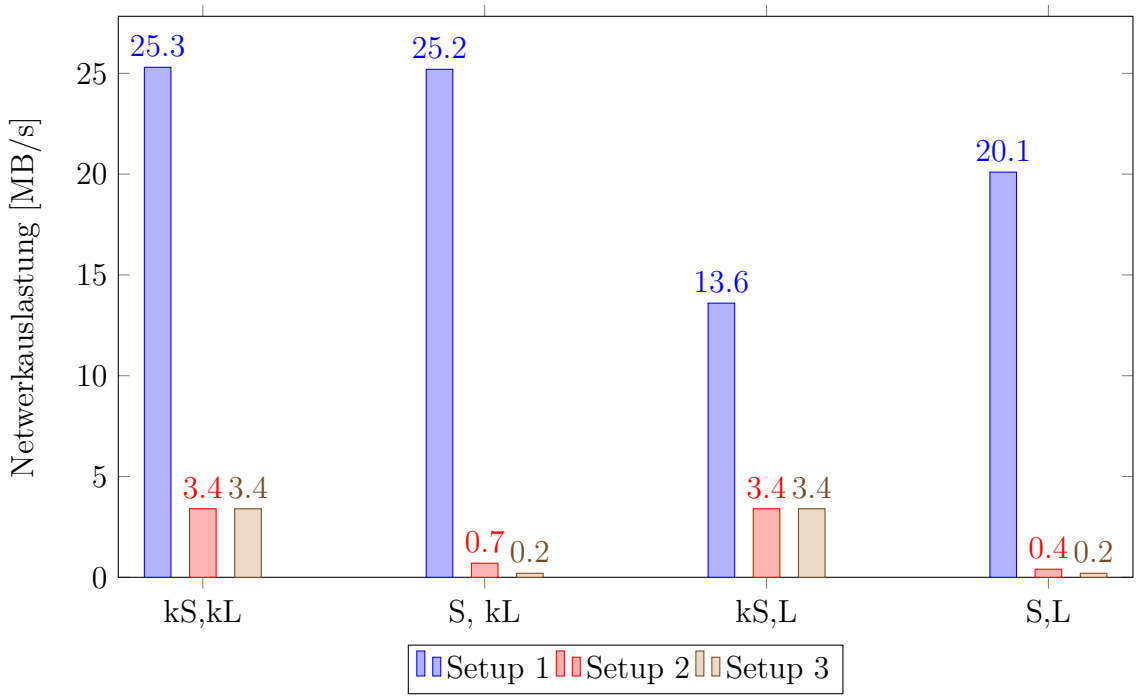


Abbildung 4.30.: Netzwerkauslastung für die 3 Setups zur Übertragung von Bildregionen, die Gesichter enthalten. (Legende: S=Synchronisation, kS=keine Synchronisation, L=Last, kL=keine Last)
Setup 1: keine Verarbeitung auf intelligenter Kamera
Setup 2: Verteilte Verarbeitung
Setup 3: alle Arbeitsschritte auf intelligenter Kamera

zusätzliche Auslastung des Steuerrechners ergibt sich bei synchroner Betriebsart ein Wert von 228 ms bei Setup 2 im Vergleich zu 415 ms bei Berechnung auf dem Steuerrechner (Setup 1). Ein derartiger Unterschied war nicht zu erwarten, da durch die Skalierung auf der intelligenten Kamera lediglich ein Teil der etwa 40 ms Übertragungszeit für das voll aufgelöste Bild eingespart wird. Auch in den Tests, die in [BZ09] publiziert sind, kommt es nicht zu einem so starken Anstieg der Latenz. Die Ablaufdiagramme zu den beiden Tests werden in den Abbildungen 4.31 und 4.32 gezeigt. Bei diesen Abbildungen handelt es sich um Aufzeichnungen, die einen bestimmten begrenzten Zeitabschnitt zeigen. Aus diesem Grund sind die Werte nicht direkt mit den gemittelten Werten in 4.29 und 4.30 vergleichbar. Es fällt beim Vergleich der beiden Diagramme sofort auf, dass der Vorgang der Gesichtslokalisation beim ersten Setup deutlich länger dauert. Alle anderen Schritte erzeugen keine zusätzliche Verzögerung, die den oben genannten Unterschied in den Latenzen erklären könnte. Als Ursache für die längere Ausführungszeit kommt somit in Betracht, dass die permanente Übertragung der voll aufgelösten Bilddaten eine zusätzliche CPU-Auslastung erzeugt, die die Ausführung der Gesichtslokalisation verzögert.

Um diesen Effekt weiter zu untersuchen, wird der Testdurchlauf wiederholt und dieses Mal eine Flusskontrolle basierend auf zwei „Trigger“-Elementen verwendet. Auf diese Art wird die Übertragung eines neuen Bildes nur dann gestartet, wenn die Verarbeitung des alten Bildes bereits abgeschlossen ist. Die Auswirkungen dieser Modifikation sind in dem Ablaufdiagramm in Abbildung 4.33 gezeigt. Wie erwartet unterscheidet sich bei dieser Konfiguration die Verarbeitungszeit nicht signifikant von der in Setup 2. Bei der Auswertung ergibt sich für die Bildregionen eine Latenz von 255 ms bei einer Bildwiederholrate von 3,6 fps. Die Netzwerkauslastung liegt bei 5,0 MB/s. Somit stellt sich im direkten Vergleich zum zweiten Setup ein gewisser Nachteil hinsichtlich der Latenz heraus, auch liegt die Netzwerkauslastung deutlich höher (Vergleichswert Setup 2: 0,7 MB/s). Diese neue Konfiguration wird nicht in die oben vorgestellte Testreihe mit aufgenommen, da sie sich nicht für die nicht synchronisierte Übertragung eignet. Auch nutzt diese neue Konfiguration die Fähigkeiten des intelligenten Kamerasystems, daher ändert dieses neue Ergebnis nichts an der Aussage der ersten Testreihe, denn dort wird das Setup 1 als solches mit einer Netzwerk-Kamera ohne Verarbeitungseinheiten betrachtet.

Der Grund, warum es in [BZ09] nicht zu dem Effekt der starken Verzögerungen bei gleichzeitigem Empfangen aller voll aufgelöster Bilddaten kommt, liegt darin, dass dort ein PC-System mit zwei CPU-Kernen verwendet wird. Eventuell ist dort auch die Implementierung der Netzwerkschnittstelle effizienter.

An einem weiteren Beispiel soll die Funktionsweise der nicht synchronisierten ROI-Erzeugung gezeigt werden. Hierzu ist in Abbildung 4.34 ein Ablaufdiagramm für das dritte Setup dargestellt. Es werden für jedes Bild die jeweiligen ROIs basierend auf den neusten verfügbaren Koordinateninformationen generiert. Bei der synchronisierten Betriebsart dauert die Gesichtslokalisation weniger als eine Sekunde (gemittelt), bei dem hier gezeigten Diagramm etwa 1,3 Sekunden. Dies liegt daran, dass bei der nicht synchronisierten Betriebsart durch die dauerhafte Generierung der ROIs und das Versenden über TCP Rechenzeit genutzt wird.

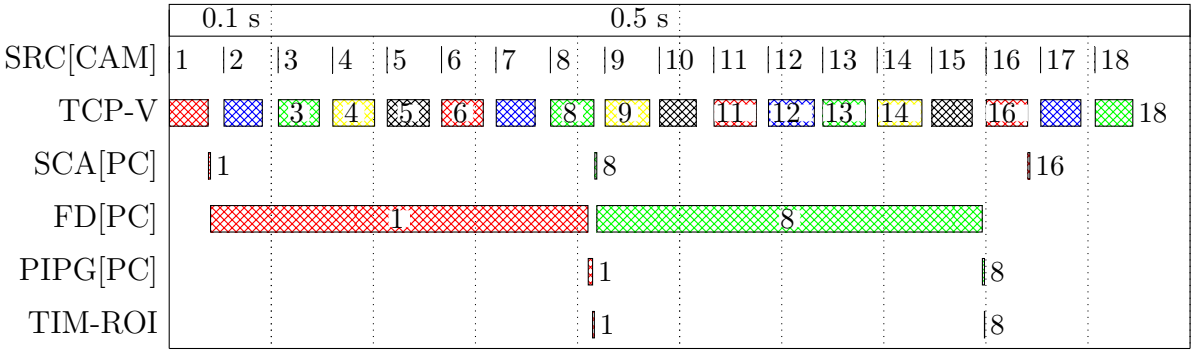


Abbildung 4.31.: Ablaufdiagramm Gesichtslokalisation - Setup 1, Verarbeitung ausschließlich auf Steuerrechner, synchronisiert, ohne zusätzliche Last. Es sind nur die wichtigsten Elemente dargestellt. Die Bilder werden vom Element „Baslersrc“ (SRC) auf der intelligenten Kamera (CAM) bereitgestellt. Alle Bilder werden in voller Auflösung über eine TCP-Verbindung (TCP-V) zu dem Steuerrechner (PC) übertragen. Nach Verringerung der Auflösung (SCA) erfolgt die Detektion von Regionen, in denen sich Gesichter befinden (FD). Im Anschluss daran werden vom Element „Pipgen“ (PIPG) aus dem voll aufgelösten Bild die Regionen extrahiert, in denen sich Gesichter befinden (in diesem Test nur ein Gesicht). Die Zeitnahme nach Abschluss der Generierung der ROIs erfolgt über ein „Timestamper“-Element (TIM-ROI).

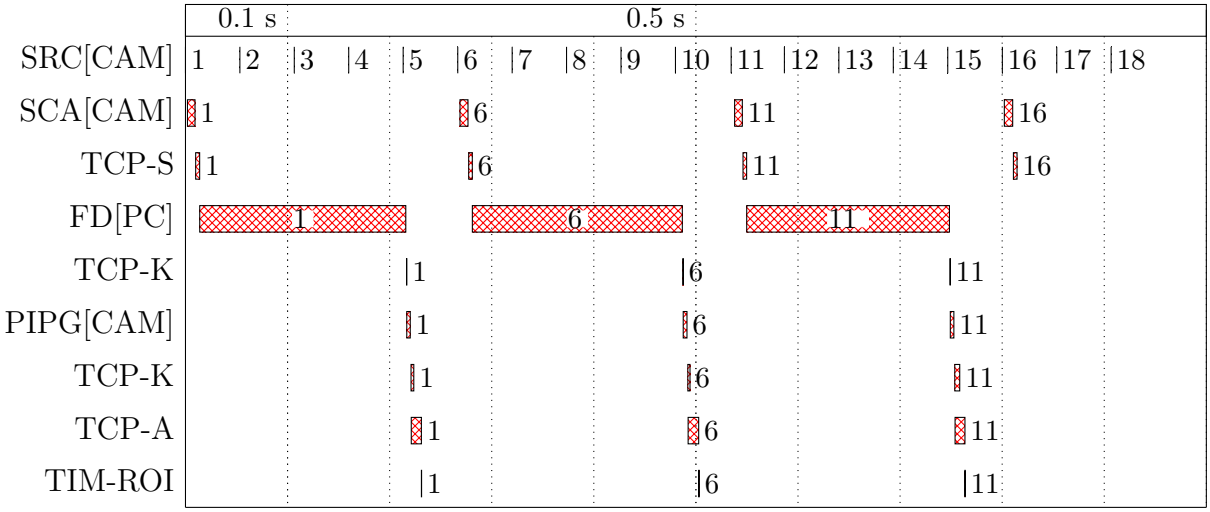


Abbildung 4.32.: Ablaufdiagramm Gesichtslokalisation - Setup 2, verteilte Verarbeitung, synchronisiert, ohne zusätzliche Last. Der Ablauf ähnelt dem in 4.31 beschriebenen Ablauf. Die Skalierung (SCA) erfolgt auf der Kamera (CAM) und die skalierten Bilder werden über eine TCP-Verbindung (TCP-S) übertragen. Nach der Gesichtslokalisation (FD) werden die Koordinateninformationen über eine TCP-Verbindung (TCP-K) zur Kamera übertragen, wo dann die ROIs erzeugt werden (PIPG). Es werden sowohl die Koordinaten (ebenfalls TCP-K) als auch die Abschnitte (TCP-A) übertragen. Diese Form der Implementierung ist notwendig, da bei der asynchronen Übertragung nur so eine Zuordnung möglich ist.

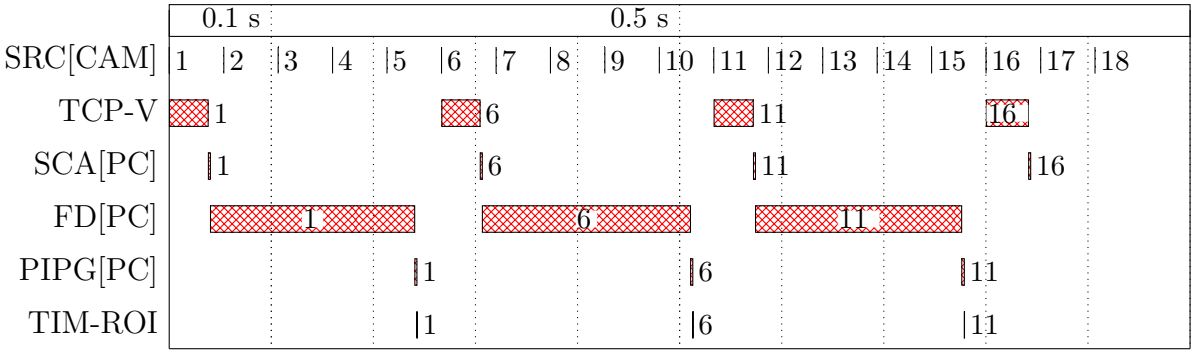


Abbildung 4.33.: Ablaufdiagramm Gesichtslokalisation - modifizierte Sendestrategie bei Setup 1, Verarbeitung ausschließlich auf Steuerrechner, synchronisiert, ohne zusätzliche Last. Die dargestellten Elemente entsprechen denen von Abbildung 4.31, die „Trigger-Elemente sind nicht dargestellt.

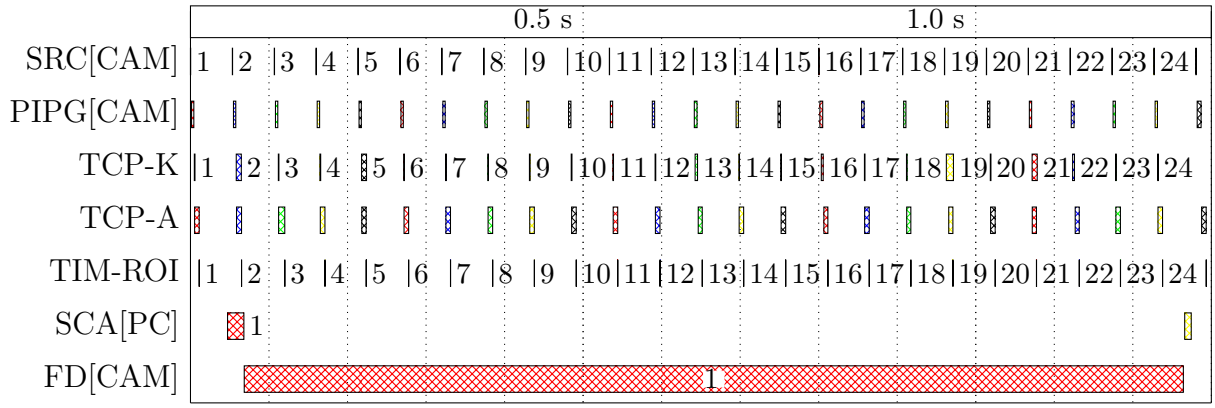


Abbildung 4.34.: Ablaufdiagramm Gesichtslokalisation - Setup 3, Verarbeitung ausschließlich auf der intelligenten Kamera, nicht synchronisiert, ohne zusätzliche Last. Die dargestellten Elemente entsprechen denen der vorherigen Abbildungen. Es zeigt sich, dass die Abschnitte kontinuierlich für jedes Bild gesendet werden.

4.3. Objekterkennung auf verteilten Systemen

Gegenstand dieses Abschnitts sind Untersuchungen zur Objekterkennung unter Verwendung des intelligenten Kamerasystems Basler eXcite und zusätzlicher Systeme. Es wird hierbei gezeigt, wie sogenanntes Cloud-Computing durch die Unterstützung von intelligenten Kameras optimal eingesetzt werden kann. Der Begriff Cloud-Computing besagt, dass eine Aufgabe von einer großen Anzahl von vernetzten Systemen ausgeführt wird.

Objekterkennung ist für die Funktion einer Roboterplattform von entscheidender Bedeutung, wenn Gegenstände mit einem Greifer manipuliert werden sollen. Hierzu müssen auch die Position der Gegenstände und ihre Orientierung bekannt sein. Neben der Planung von Greifvorgängen kann die Detektion von bekannten ortsfesten Objekten auch zur Lokalisierung der Roboterplattform verwendet werden.

Zur Objekterkennung werden die in Abschnitt 3.3 erwähnten SIFT-Elemente verwendet. Die Bilddaten werden von der intelligenten Kamera erzeugt. Hierfür wird das Element „Baslersrc“ verwendet, das auf die Kameratreiber zugreift. Aus den Bilddaten wird zunächst mit dem Element „Siftextractor“ ein Feature-Vektor extrahiert. Der Feature-Vektor wird nun zu dem Element „Siftfolder“ geleitet. Dieses Element vergleicht den Vektor mit einer Datenbank von Objekten, die wie folgt erzeugt wird: Als Parameter wird dem „Siftfolder“-Element der Pfad eines Ordners mit Bildern der zu detektierenden Objekte übergeben. Alle Bilddaten werden eingelesen, und von jeder Bilddatei wird jeweils ein Feature-Vektor erzeugt. Im Element wird nun verglichen, ob zwischen den Merkmalen der Bilddaten aus dem Ordner und den Merkmalen des aktuellen Kamerabildes ausreichend

Übereinstimmungen vorhanden sind, um eine Transformation zu berechnen. Ist zudem die tatsächliche Objektgröße bekannt, kann die Position des Objektes im Raum berechnet werden. Die Informationen zu den gefundenen Transformationen werden dann von diesem Element ausgegeben. Es ist zudem möglich, zu konfigurieren, dass der Abgleich nur mit einem Teil der Datenbasis durchgeführt wird. So können mehrere Instanzen des „Siftfolder“-Elements gleichzeitig den Feature-Vektor mit jeweils unterschiedlichen Objekten vergleichen.

Der große Vorteil, der hier genutzt wird, besteht darin, dass sich die relevanten Informationen eines Bildes (ca. 1,5 MB) in einem kompakten Feature-Vektor (ca. 50 kB bei 300 Merkmalen, abhängig vom Bildinhalt) speichern lassen, wenn die im Rahmen dieser Arbeit implementierte Form der Darstellung verwendet wird. Dieser Feature-Vektor reicht aus, um in dem Bild nach bekannten Gegenständen zu suchen. Somit muss nur eine geringe Datenmenge zwischen den Elementen „Siftextractor“ und „Siftfolder“ übertragen werden.

Bei den Experimenten werden verschiedene Konfigurationen untersucht, bei denen auch von der Möglichkeit Gebrauch gemacht wird, zusätzlich zwei kompakte dedizierte Rechner auf dem Service-Roboter zu installieren und die intelligente Kamera zur Verteilung der Bilddaten auf diese Rechner zu nutzen. Dabei wird die oben beschriebene Funktion verwendet, die Berechnung des SIFT-Vektors für einzelne Bildabschnitte parallelisiert auszuführen. Das Bild wird in vier Abschnitte geteilt, dann wird für jeden Abschnitt der Vektor unabhängig berechnet und die Vektoren werden später von Elementen des Typs „Sifttool“ zusammengefügt. Auf diese Art kann die Berechnung auf zwei Systemen jeweils in zwei Threads durchgeführt werden.

In der Arbeitsumgebung des Roboters befinden sich außerdem viele Desktop-Rechner. Untersuchungen zeigen, dass solche Systeme bei normaler Büroarbeit nur gering ausgelastet sind (vgl. [Gra08]). Deren Rechenkapazität wird im Rahmen dieser Experimente ebenfalls genutzt, indem die Suche nach bekannten Objekten auf diesen Systemen parallel ausgeführt wird. Einen schematischen Aufbau der beteiligen Systeme zeigt Abbildung 4.35.

Im Folgenden sollen die in Abbildung 4.36 dargestellten Systemkonfigurationen erläutert werden.

Setup 1: Bei diesem Setup wird auf dem intelligenten Kamerasystem der SIFT-Vektor berechnet und per WLAN an acht Desktop-Systeme in der Umgebung gesendet. Auf jedem dieser Systeme erfolgt nun ein Vergleich mit jeweils einem Teil der Objekt-Datenbank. Somit kann die zeitintensive Suche nach übereinstimmenden Features parallelisiert werden.

Setup 2: Das zweite und dritte Setup unterscheiden sich lediglich dadurch, dass sich beim zweiten Setup zwei dedizierte Systeme auf dem mobilen Roboter befinden, während sie beim dritten Setup stationär im Umfeld des Roboters installiert sind. Für beide Setups wird in Grafik 4.36 dasselbe Diagramm verwendet. Die intelligente Kamera wird hier dazu genutzt, die Bilder horizontal in vier überlappende Abschnitte zu teilen. Je zwei Teilbilder

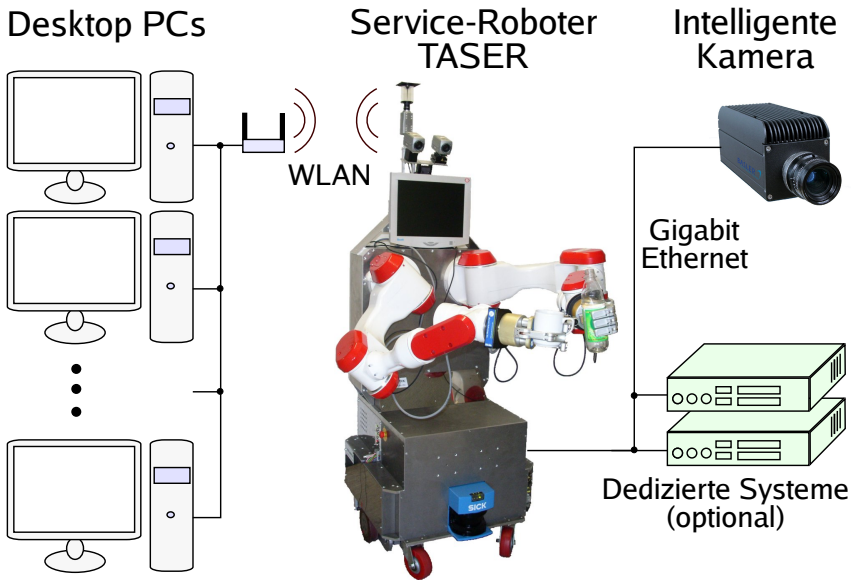


Abbildung 4.35.: Cloud-Computing mit Unterstützung durch eine intelligente Kamera

werden von jeweils einem der dedizierten Systeme verarbeitet. Es werden jeweils die SIFT-Vektoren extrahiert und dann zusammengefügt. Die weitere Verarbeitung erfolgt wie bei Setup 1.

Setup 3: Bei dem dritten Setup, bei dem die zwei dedizierten Systeme stationär installiert sind, müssen die unverarbeiteten Bilddaten über das kabellose Netzwerk gesendet werden. Im Übrigen erfolgt die Verarbeitung wie in Setup 2.

Setup 4: Bei der vierten Konfiguration erfolgen alle Arbeitsschritte auf dem Steuerrechner des Service-Roboters. Hierbei wird die Kamera so konfiguriert, dass lediglich die unverarbeiteten Bilddaten versendet werden. Zusätzlich werden weitere Software-Komponenten auf dem Steuerrechner ausgeführt, so dass dieser ohne die Bildverarbeitung bereits zu ca. 60 % ausgelastet ist.

Die Untersuchungen wurden zu verschiedenen Zeitpunkten mit jeweils unterschiedlichen Systemen und unterschiedlichen Konfigurationen durchgeführt. Somit sind die Ergebnisse nicht direkt vergleichbar, zeigen jedoch jeweils eine entscheidende Leistungssteigerung. Zunächst werden die Ergebnisse der ersten Versuchsreihe betrachtet, die mit dem Service-Roboter TASER aufgenommen wurden. Diese Ergebnisse sind auch in [BZ10] veröffentlicht.

Zu einem späteren Zeitpunkt wurden die Tests mit leistungsfähigerer Hardware wiederholt, um so die Skalierbarkeit zu zeigen und den Ablauf genauer zu untersuchen. Zu diesem Zeitpunkt war jedoch die Roboterplattform TASER aus anderen technischen Gründen nicht einsatzfähig, daher wurden die Tests mit einem äquivalenten PC durchgeführt und die kabellose Netzwerkverbindung wurde durch eine Beschränkung der Datenrate simuliert (Verzögerung in Abhängigkeit von der Datenrate).

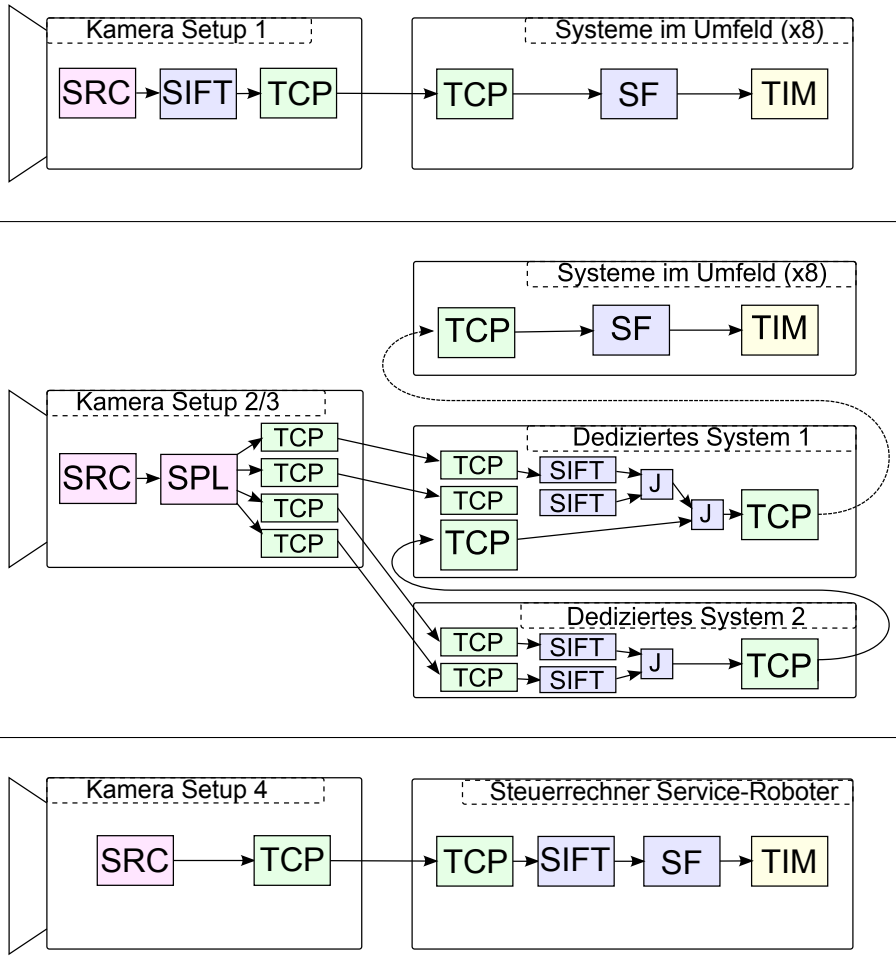


Abbildung 4.36.: Die verschiedenen Konfigurationen zur Objekterkennung. Setup 1 (oben): Feature-Vektor-Berechnung auf Kamera, Vergleich auf Systemen in der Umgebung; Setup 2/3 (Mitte): Feature-Vektor-Berechnung auf dedizierten Systemen, Vergleich auf Systemen in der Umgebung; Setup 4 (unten): alle Verarbeitungsschritte auf Steuerrechner des Service-Roboters
Legende der verwendeten Elemente:
SRC: kapselt den Treiber der Kamerahardware
SIFT: berechnet den Feature-Vektor mit dem SIFT Algorithmus
SPL: (split) teilt die Bilder in mehrere Teile auf
J: (join) Element „Sifttools“, fügt jeweils zwei Feature-Vektoren zu einem zusammen
TCP: überträgt Daten per TCP-Verbindung
TIM: vergleicht Zeitstempel der Daten mit aktueller NTP-Zeit
SF: („Siftfolder“) vergleicht Feature-Vektor mit Objekt-Datenbank

4.3.1. Versuchsdurchführung auf dem Roboter TASER

Bei den zeitlich früheren Tests auf der Roboterplattform TASER werden die in 4.36 dargestellten Konfigurationen auf den entsprechenden Systemen aufgesetzt. Es kommen hierbei neben der Basler eXcite folgende Rechner zum Einsatz:

- Steuer-PC des Service-Roboters: Intel Pentium 4, 2,4 GHz
- Dedizierte Systeme: Intel Core 2 Duo, 2,2 GHz
- Systeme in der Umgebung: Intel Core 2 Duo 2,4 GHz

Auf dem Service-Roboter wird ein Gigabit-Ethernet-Switch installiert. Die Daten, die zwischen dem Robotersystem und den Systemen in der Umgebung ausgetauscht werden, werden über den Steuerrechner des Robotersystems geroutet.

Es werden 100 Bilddateien in den vom Element „Siftfolder“ einzulesenden Ordner gespeichert. Die Objekte, die zu suchen sind, sind aus Tischszenen extrahiert und haben jeweils eine Auflösung von ca. 250 x 250 Pixeln (in Abhängigkeit von dem Seitenverhältnis in Höhe oder Breite weniger). Diese Auflösung wurde gewählt, da sie einen akzeptablen Kompromiss aus Rechenzeit und Erkennungsrate ermöglicht. Da die Objekte eventuell unscharf abgebildet sind, fällt die effektive Auflösung, die letztendlich auch die Anzahl der gefundenen Merkmale und somit den Rechenaufwand bedingt, oft geringer aus. Bei der Übertragung der Ergebnisse werden nur Informationen zu gefundenen Objekten ausgegeben.[3] Bei den Tests wird eine einfache Flusskontrolle durch „Trigger“-Elemente genutzt, die dafür sorgt, dass auf der gesamten Verarbeitungsstrecke nur ein Bild verarbeitet wird. Um die Testergebnisse besser nachvollziehen zu können, wird vorab die Datenübertragungsrate der kabellosen Netzwerkverbindung (802.11g) an verschiedenen Plätzen ausgewertet. Es ergeben sich Werte zwischen 1,4 MB/s und 2,5 MB/s. Die auf dem Steuerrechner laufenden Prozesse zur Steuerung der Roboterplattform verursachen bereits eine 60-prozentige Auslastung des Systems.

Bei den Tests wird die Verarbeitungsdauer für die verschiedenen Elemente gemessen. Die verschiedenen Operationen, die der Datenübertragung dienen, werden dabei zusammengefasst betrachtet. Anzumerken ist weiterhin, dass bei diesem Test und den weiteren Tests im Rahmen dieser Arbeit nicht die Erkennungsleistung, sondern lediglich die durch den SIFT-Algorithmus erzeugte Rechenlast untersucht wird.

Die Ergebnisse der Untersuchungen sind in den Grafiken 4.37 und 4.38 dargestellt. Es zeigt sich, dass durch die Aufteilung der Verarbeitungsschritte im zweiten Setup eine deutliche Steigerung der Verarbeitungsleistung ermöglicht wird. Für den Betrieb des Roboters bedeutet dies, dass das Robotersystem Greifvorgänge planen und ausführen kann, ohne den Arbeitsablauf unnötig lange zu unterbrechen. Ebenso kann mit der gezeigten Implementierung die Objekterkennung als Background-Task, also ohne zusätzliche Auslastung des Steuerrechners, ablaufen. So könnte beispielsweise als eine Aufgabe für das

[3] Informationen zu nicht detektierten Objekten werden nicht ausgegeben, stattdessen wird von jedem „Siftfolder“-Element pro eingehendem Feature-Vektor über eine spezielle Nachricht der Abschluss der Verarbeitung angezeigt.

Setup	Feature-Berechnung	Feature-Vergleich	Datenübertragung
1	6781 ms	381 ms	16 ms
2	269 ms	376 ms	62 ms
3	274 ms	383 ms	695 ms
4	4110 ms	10428 ms	50 ms

Abbildung 4.37.: Verarbeitungszeit der verschiedenen Setups (a). In der Tabelle werden die gemessenen Verarbeitungszeiten (gemittelt) aufgeführt. Es ist anzumerken, dass diese Werte deutlichen Schwankungen unterliegen, die durch den Bildinhalt, die Auslastung der Computersysteme und weiteren Datenverkehr auf den Netzwerkschnittstellen verursacht werden.

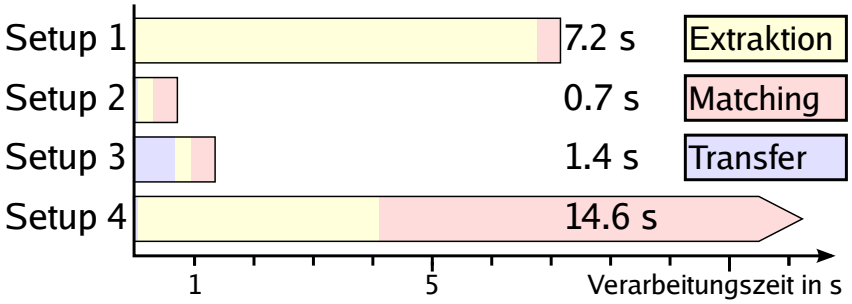


Abbildung 4.38.: Verarbeitungszeit der verschiedenen Setups (b). Diese Abbildung stellt grafisch die Gesamtlatenz dar, die sich durch Summierung der einzelnen Latenzwerte aus Abbildung 4.37 ergibt.

Robotersystem definiert werden, beiläufig und permanent nach verlorengegangenen Gegenständen Ausschau zu halten. Mit Hilfe dieser Tests kann ebenfalls gezeigt werden, dass eine Auslagerung der Aufgaben besonders dann sinnvoll ist, wenn die Daten bereits lokal vorverarbeitet werden, um die Datenmenge zu verringern. Gegenwärtig ist diese lokale Vorverarbeitung performant nur mit dedizierten Systemen möglich (Setup 2). Bei zukünftigen leistungsfähigeren intelligenten Kamerasystemen könnte auch Setup 1 nutzbringend eingesetzt werden. Aufgrund des geringen Arbeitsspeichers der Kamera kann ein Test, bei dem alle Verarbeitungsschritte auf der Kamera ausgeführt werden, nicht erfolgen.

4.3.2. Versuchsdurchführung auf modernerer Hardware und ausführliche Analyse

Die in diesem Abschnitt beschriebenen Tests sind nicht direkt mit den entsprechenden Tests im vorherigen Abschnitt vergleichbar. Dies liegt zum einen daran, dass andere Systeme zur Berechnung benutzt werden und die Software-Architektur weiter verbessert wurde, zum anderen daran, dass die zu suchenden Objekte eine deutlich höhere Auflösung haben (längere Seite auf 1000 Pixel beschränkt). Des Weiteren werden teilweise neuere Softwarebibliotheken beim Kompilieren verwendet. Trotz der höheren Auflösung ergeben sich hier sogar deutlich geringere Latenzen. Es kann so gezeigt werden, dass die im Rahmen dieser Arbeit entwickelten Methoden mit der verfügbaren Rechenleistung skalieren. Bei dem aktuellen Experiment wird die zweite Konfiguration genauer analysiert, die bei den früheren Tests die beste Latenz ermöglichte. Zusätzlich wird die Aufgabe zum Vergleich wie in der vierten Konfiguration auf einem PC ausgeführt, der von seiner Leistung her dem Steuerrechner des Service-Roboters entspricht. Die verteilte Verarbeitung wird durch mehrere unten erläuterte Maßnahmen noch weiter optimiert, um die Latenz zu verringern und die Wiederholrate zu erhöhen.

Neben der intelligenten Kamera Basler eXcite (CAM) wird folgende Hardware genutzt:

- Steuer-PC des Service-Roboters (STR): Intel Pentium 4, 2,4 GHz
- 2 dedizierte Systeme (D1/D2): Intel Core i5 3570, 3,40 GHz
- 8 Systeme in der Umgebung (C1-C8): 4x Core i5 750, 2,67 GHz, 4x Intel Xeon E31245, 3,30 GHz

Hierbei ist in Klammern die Bezeichnung der Systeme für die Abbildung 4.40 und die dazugehörige Legende 4.41 angegeben.

Zunächst werden zu Vergleichszwecken alle Aufgaben auf einem Pentium 4 PC mit 2,4 GHz ausgeführt. Die Flusskontrolle wird mittels eines Queue-Elements realisiert, das nur ein Bild zwischenspeichert und jeweils beim Eintreffen neuer Daten alte Daten verwirft, wenn deren Verarbeitung noch nicht gestartet wurde. Erst wenn der Vergleich mit allen 100 Objekten abgeschlossen ist, wird die Verarbeitung des nächsten aktuellen Bildes gestartet. Die Ergebnisse sind in Abbildung 4.39 dargestellt. Der gesamte Durchlauf dauert über 25 Sekunden, wobei der größte Anteil auf die Suche nach bekannten Objekten entfällt. Es ist

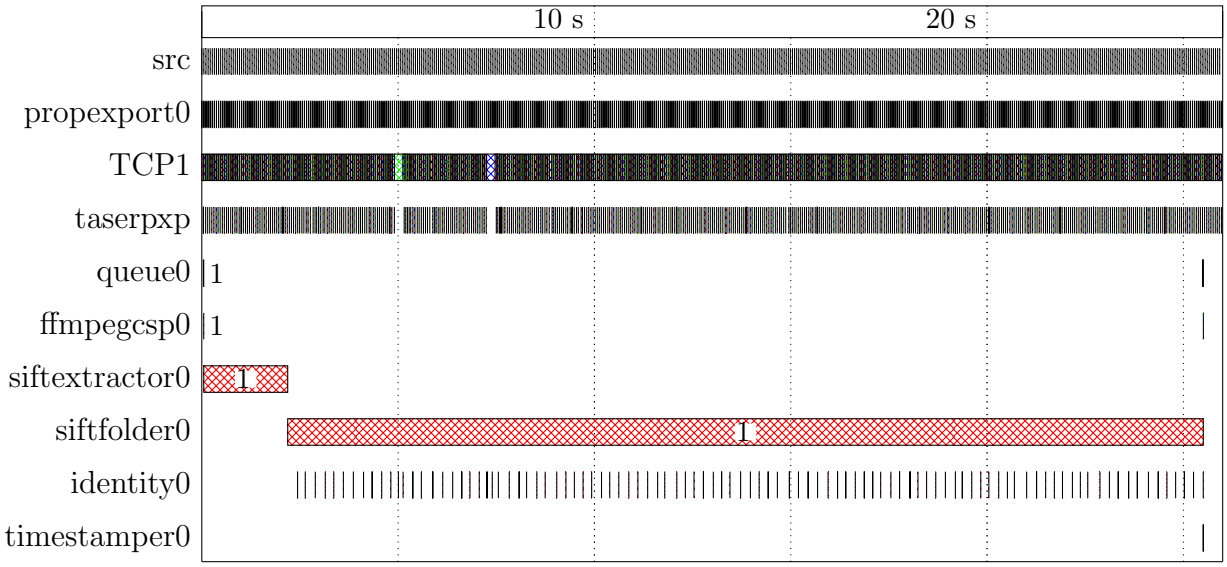


Abbildung 4.39.: Ablaufdiagramm Objekterkennung auf einem PC: Es ist zu erkennen, dass hauptsächlich die Berechnung des Feature-Vektors und der Vergleich mit bekannten Objekten die Rechenzeit verursachen. In diesem Zeitraum werden viele Bilddaten verworfen. Aufgrund des Abbildungsmaßstabes sind die einzelnen Bilder nicht mehr erkennbar. Zudem zeigen sich zeitweise Verzögerungen bei der Datenübertragung. Es ergibt sich eine Verarbeitungszeit von über 25 Sekunden, bis die Suche nach allen Objekten abgeschlossen ist.

ersichtlich, dass die Vergleichsergebnisse in regelmäßigen Abständen ausgegeben werden. Dies zeigt das große Potential der Parallelisierung, das hier genutzt werden kann, da die Suche nach jedem der Objekte auch unabhängig voneinander auf verschiedenen Systemen ausgeführt werden kann.

Die Bildverarbeitungspipeline wird nun im nächsten Test auf verschiedene Systeme verteilt und in vielerlei Hinsicht optimiert. Das entsprechende Ablaufdiagramm zeigt Abbildung 4.40. Die zugehörige Legende ist in Abbildung 4.41 aufgeführt. Durch die entwickelte Strategie wird erreicht, dass die Verarbeitungszeit bei der Suche nach bekannten Objekten von über 25 s auf unter 0,5 s gesenkt werden kann. Dieser Umstand stellt für den Arbeitsablauf eines Robotersystems einen entscheidenden Vorteil dar, da der Arbeitsablauf nur für eine deutlich kürzere Zeit unterbrochen werden muss. Im Folgenden soll das Ergebnis noch einmal detailliert analysiert werden. Dabei wird im Einzelnen erklärt, welche im Rahmen dieser Arbeit durchgeführten Implementierungen und Optimierungen dieses Ergebnis ermöglichen.

Grundlegende Voraussetzung für die beschriebene Verarbeitungsstrategie ist, dass die

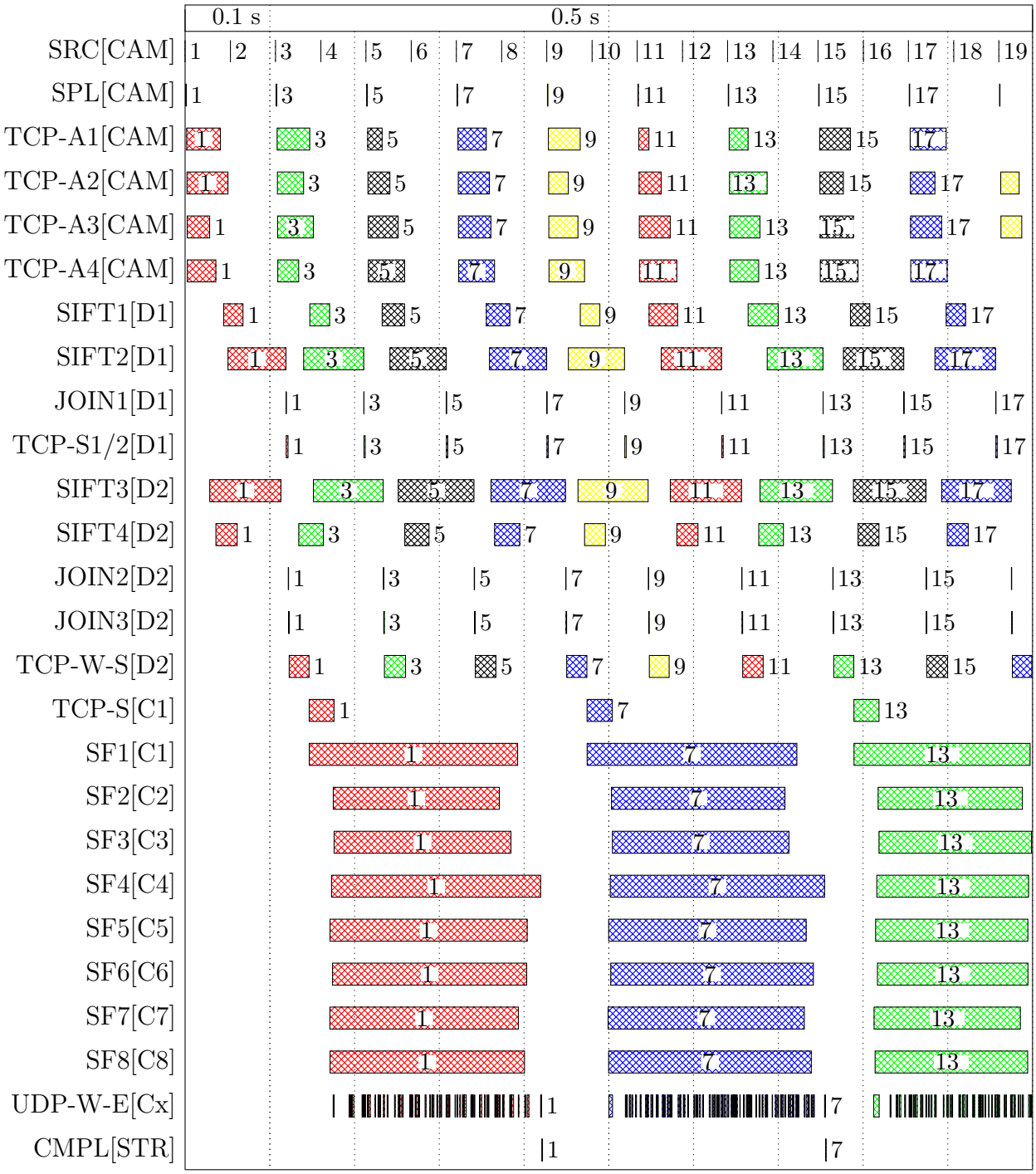


Abbildung 4.40.: Ablaufdiagramm verteilte Objekterkennung:

Die einzelnen Elemente:

SRC[CAM]:	Element „Baslersrc“, auf intelligenter Kamera
SPL[CAM]:	Element „Splitter“, teilt Bild in horizontale Abschnitte
TCP-Ax[CAM]:	TCP-Transfer der vier Abschnitte, jeweils zwei zu System D1, zwei zu D2
SIFTx[Dx]:	Vier Elemente zur Berechnung des SIFT-Vektors, jeweils zwei auf den Systemen D1 und D2
JOIN1[D1]:	Fusioniert die Ergebnisse von SIFT1 und SIFT2
TCP-S1/2[D1]:	Überträgt das Ergebnis von JOIN1 zu D2
JOIN2[D2]:	Fusioniert die Ergebnisse von SIFT3 und SIFT4
JOIN3[D2]:	Fusioniert die Ergebnisse von JOIN1 und JOIN2
TCP-W-S[D2]:	Überträgt den kompletten SIFT-Vektor zu C1 (kabellose Verbindung)
TCP-S[C1]:	Überträgt den kompletten SIFT-Vektor zu C2-C7 (Standard-Netzwerk-Verbindung)
SF1-8[C1-C8]:	Siftfolder-Elemente: Berechnen jeweils Matches mit Objekten aus einer Datenbank
UDP-W-E[Cx]:	Überträgt die Ergebnisse der Siftfolder-Elemente SF1 bis SF8 zum Steuerrechner des Roboters (STR, kabellos, UDP-Protokoll)
CMPL[STR]:	Dieses Element prüft, ob alle Daten der Elemente SF1 bis SF8 eingetroffen sind

Abbildung 4.41.: Legende zu 4.40: Bei der Aufzählung der Elemente ist das System, auf dem das jeweilige Element zur Ausführung kommt, in eckigen Klammern angegeben.

Bildverarbeitungsaufgabe in verschiedene Teilschritte zerlegt werden kann und diese Teilschritte auf verschiedene Systeme verteilt werden können. Auf diese Art können auch leistungsfähigere externe Systeme für Berechnungen herangezogen werden. Diese Voraussetzung wird hier durch die gewählte Architektur und die Art der Implementierung der einzelnen Algorithmen geschaffen. Ein entscheidender Faktor ist, dass die Berechnung des SIFT-Vektors und der Vergleich mit einer Objektdatenbank in verschiedenen Elementen gekapselt sind.

Die Bilddaten werden vom Element „Baslersrc“ auf der intelligenten Kamera Basler eXcite erzeugt. Dieses Kameratreiber-Element zur Kapselung der Kamera-Hardware auf der Basler eXcite arbeitet nach dem „Zero-Copy“-Prinzip. Dies besagt, dass die Speicherbereiche zwischen den Kameratreiber-Bibliotheken und den Elementen in der GStreamer-Pipeline geteilt werden. Eine weitere Optimierung findet sich in dem Element zum Unterteilen des Bildes in mehrere Abschnitte. Die Unterteilung in horizontale Abschnitte ermöglicht es, ohne zusätzliches Kopieren der Daten eine Teilung vorzunehmen. Dementsprechend ist in Diagramm 4.40 keine signifikante Verzögerung durch dieses Element zu erkennen.

Die Ausführung der Unterteilung auf der eXcite wirkt sich ebenfalls positiv auf die Latenz aus, da ansonsten das komplette Bild zu einem der dedizierten Systeme übertragen werden müsste und zwei der Abschnitte von diesem zu dem anderen dedizierten System gesendet werden müssten. Mittels eines weiteren (hier nicht weiter betrachteten) Tests zeigt sich, dass die hierdurch erreichte Verbesserung der Latenz etwa im Bereich von 7 ms liegt. Dieser relativ geringe Wert ergibt sich dadurch, dass die beiden dedizierten Systeme bei Kommunikation untereinander die maximale Datenübertragungsrate der Netzwerkverbindung voll ausnutzen.

Die vier zur Datenübertragung genutzten TCP-Elemente sind in GStreamer vorgegeben. Die Untersuchungen im Rahmen dieser Arbeit zeigen, dass diese im Vergleich zu den Kommunikationsmechanismen anderer Frameworks eine hohe Performance aufweisen (vgl. Abschnitt 3.4.1). Das Element zur Berechnung des SIFT-Vektors basiert auf dem Code von Hess. Im Rahmen dieser Arbeit erfolgten einige Optimierungen zur Vermeidung von unnötigen Kopieroperationen bei mehreren aufeinander folgenden Berechnungen, wie sie bei Videodatenströmen durchgeführt werden. Des Weiteren wird hier deutlich, dass die Ausführungszeit vom Bildinhalt abhängt. Das Zusammenführen der einzelnen Vektoren erzeugt nur geringe Rechenlast. Auch der Transfer des Teilvektors vom System D1 auf D2 über das Gigabit-Ethernet verursacht nur geringe Verzögerungen.

Der Ablaufsteuerung kommt ebenfalls eine besondere Bedeutung zu. Durch die Verwendung von Trigger-Elementen und eine vorgezogene Sendung lässt sich hier erreichen, dass eine hohe Wiederholrate erzielt wird und es trotzdem nicht zu erhöhten Latenzen aufgrund von Zwischenspeicherung kommt. Auch wenn die hohe Wiederholrate bei der Erzeugung der SIFT-Vektoren (9 fps) durch die Verarbeitungsgeschwindigkeit der weiteren Schritte nicht voll ausgenutzt werden kann, so führt die häufige Aktualisierung dazu, dass die Ausführung der folgenden Schritte schnell starten kann, wenn die vorherigen Daten abgearbeitet sind. Als Ausgabeformat des Elements JOIN3 wird die im Rahmen dieser

Arbeit implementierte kompakte Repräsentation gewählt (s. Abschnitt 3.3). Auf diese Art müssen weniger Daten über das kabellose Netzwerk übertragen werden. Würden andere Repräsentationen gewählt, so käme es hier zu deutlich größeren Verzögerungen.

Da Feature-Vektoren auf acht verschiedenen Systemen benötigt werden, wird auf einem der Systeme (C1) die Weiterverteilung an die übrigen Systeme (C2 - C8) über die schnellere kabelbasierte Netzwerkverbindung durchgeführt. In der Darstellung der entsprechenden Netzwerkverbindung TCP-S[C1] in Abbildung 4.40 ist der Zeitraum abgebildet, bis alle Zielsysteme (C2 - C8) die Daten erhalten haben. Die genauen Startzeitpunkte der „Siftfolder“-Elemente auf den Systemen C2 bis C8 variieren geringfügig; es ist aus dem Diagramm ersichtlich, dass das Element auf dem System C1 eher mit der Verarbeitung startet. Die Ergebnisse werden einzeln als UDP-Pakete zum Steuerrechner des Roboters gesendet. Pro Objekt und Bild werden 224 Bytes übertragen. Bei der Wiederholrate von 3 Bildern pro Sekunde ergibt sich eine Datenrate auf der kabellosen Netzwerkverbindung von ca. 68 kB/s, was als unkritisch zu betrachten ist. Eine deutliche Reduzierung dieser Datenrate wäre möglich, wenn die „Siftfolder“-Elemente dahingehend konfiguriert werden, nur für gefundene Objekte das Ergebnis auszugeben. In diesem Test werden jedoch auch negative Ergebnisse übermittelt, um in den Visualisierungen den zeitlichen Verlauf besser darstellen zu können.

Die Kombination all dieser verschiedenen Optimierungen ermöglicht diese erhebliche Verbesserung der Verarbeitungszeit. Hierbei darf natürlich nicht außer Acht gelassen werden, dass die verwendeten externen Systeme neuer und deutlich performanter als der Steuerrechner des Service-Roboters (P4 2,4 GHz) sind. Zum einen wird jedoch durch die im Rahmen dieser Arbeit implementierten Funktionen überhaupt erst die Möglichkeit geschaffen, externe Systeme zu benutzen, zum anderen verfügen auch viele heutige Robotersysteme über einen Steuerrechner mit geringer Rechenleistung (z.B. Intel Atom).

Abschließend ist festzustellen, dass die im Rahmen dieser Arbeit erarbeiteten Methoden zu einer deutlichen Steigerung der Verarbeitungsleistung bei der Objekterkennung genutzt werden können. Die intelligente Kamera Basler eXcite wird hierbei zum Verteilen der Bildausschnitte an die weiteren Systeme verwendet. Durch eine Umkonfiguration könnte problemlos auch die Rechenleistung von zukünftigen intelligenten Kameras genutzt werden. Die dedizierten Systeme auf dem Service-Roboter wären dann nicht mehr notwendig.

4.4. Automatische Zuweisung der Aufgaben

In den vorangegangenen Abschnitten 4.2 und 4.3 wird gezeigt, wie durch die Verteilung der Aufgaben auf mehrere Systeme eine Leistungssteigerung des Systems erreicht werden kann. Die verschiedenen getesteten Konfigurationen werden manuell erstellt. Als weiterer Bestandteil dieser Arbeit wird ein Ansatz aufgezeigt, wie bei gegebener Aufgabenstellung automatisch eine Zuweisung der Teilaufgaben auf die aktuell verfügbaren Systeme erfolgen kann. Dies kann in folgenden Fällen von Nutzen sein:

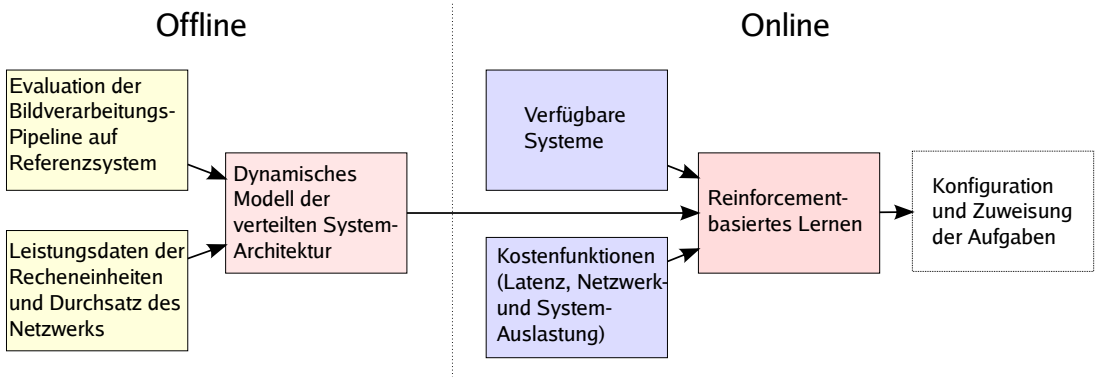


Abbildung 4.42.: Konzept der automatischen Zuweisung der Aufgaben. Die dargestellten Funktionen werden teilweise einmalig bei der Konfiguration ausgeführt (offline), teilweise auch zur Laufzeit unter Berücksichtigung aktueller Rahmenbedingungen (online).

- Zuweisung der Teilaufgaben für eine neue Aufgabenstellung
- Veränderung der Anzahl und Art der verfügbaren Systeme in einer dynamischen Umgebung
- Veränderung der Prioritäten bei der Ausführung

Vor allem kann dieses System auch dahingehend genutzt werden, um zu beurteilen, ob für eine gegebene Aufgabenstellung der Einsatz von intelligenten Kameras und auch weiteren dedizierten Systemen Vorteile bringt.

Wird das System als „Black Box“ betrachtet, so funktioniert die automatische Zuweisung der Aufgaben wie in Abbildung 4.42 dargestellt. Als Eingangsdaten werden die aktuelle Verarbeitungsstrategie, eine Liste der verfügbaren Computersysteme mit ihren jeweiligen Eigenschaften, Informationen über die Netzwerk-Verbindung zwischen den Systemen sowie Kostenfunktionen benötigt.

Mit den Kostenfunktionen wird das System angewiesen, bestimmte Prioritäten bei der Konfiguration zu setzen. Hierzu zählen:

- Latenz der Daten
- Bildwiederholrate
- CPU-Auslastung von Systemen
- Auslastung von Netzwerkschnittstellen

Die Werte für die genannten Punkte werden als reelle Zahl zwischen 0 und 1 gewählt. Ist beispielsweise der Wert für die Latenz $1,0$ und für alle anderen Punkte $0,0$, so sucht das System nach der Konfiguration mit der geringsten Latenz der Daten.

Bei der aktuellen Verarbeitungsstrategie handelt es sich um die Beschreibung einer Bildverarbeitungspipeline. In dieser sind nur die Elemente aufgeführt, die tatsächlich für die Bildverarbeitung zur Laufzeit relevant sind. Elemente, die nur zu Beginn der Laufzeit

aktiv sind und z.B. Konfigurationsdateien einlesen, werden ignoriert. Ebenso werden Elemente ignoriert, deren Funktion sich nur auf die Zwischenspeicherung und Verteilung der Daten sowie auf die Übertragung der Daten zwischen verschiedenen Systemen bezieht.

Die Liste der verfügbaren Computersysteme enthält Informationen zur Performance der einzelnen Systeme. Die Performance wird exemplarisch für einzelne Operationen der Bildverarbeitung mit den Methoden aus Abschnitt 3.2.10 gemessen. Abschließend wird ein Leistungsindex für jedes verfügbare System erstellt. Dabei wird so vorgegangen, dass die Leistung eines Referenzsystems willkürlich auf 1,0 festgelegt wird. Der Index für die weiteren Systeme ist dann entsprechend ein Vielfaches dieses Wertes (oder ein Bruchteil). Für das angewendete Verfahren ist nur das Verhältnis der Leistungsindizes zueinander ausschlaggebend.

Bei dem implementierten Verfahren zur Bestimmung einer geeigneten Aufgabenverteilung handelt es sich um so genanntes Reinforcement Learning (vgl. [SB98]). Das Hauptprinzip hierbei besteht darin, dass dem Lern-System über ein Reward-Signal mitgeteilt wird, ob die gerade betrachtete Lösung geeignet ist oder nicht. In der hier durchgeführten Implementierung wird das Reward-Signal anhand der oben erwähnten vier Kriterien berechnet. Hierbei werden die zu erwartenden Werte (modelliert) mit den realen Werten des Referenzsystems verglichen, auf dem die gesamte Bildverarbeitungspipeline testweise ausgeführt wird.

Im Folgenden werden die wesentlichen Grundlagen des Reinforcement-Learnings basierend auf der oben genannten Referenz kurz zusammengefasst, soweit dies für das Verständnis des hier entwickelten Ansatzes erforderlich ist. Die Beschreibung erfolgt an dieser Stelle und nicht in Kapitel 2, da dieses Verfahren nur für das hier vorgestellte System zur automatischen Konfiguration der Bildverarbeitungspipeline relevant ist.

4.4.1. Grundlagen zu Reinforcement Learning

Das Reinforcement-Learning [4] ist ein Verfahren, bei dem ein Agent durch Interaktion mit seiner Umgebung lernt. Dabei erhält der Agent von seiner Umgebung ein numerisches Reward-Signal. Der Agent versucht, den Reward, den er von der Umgebung erhält, auf lange Sicht zu maximieren. Dazu wird für jeden existierenden Zustand eine Bewertung vorgenommen. Diese Bewertung basiert auf den Rewards R, die ausgehend von diesem Zustand zu erwarten sind:

$$\begin{aligned} R_t = &\quad r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots \\ = &\quad \sum_{n=0}^{\infty} (\gamma^n R_{t+n+1}) \end{aligned}$$

In dieser Gleichung werden die zukünftigen Rewards durch einen sogenannten Discount-Faktor $\gamma \in [0, 1]$ abgeschwächt. Basierend auf einer solchen Zustandsbewertung entwickelt

[4] auf deutsch Verstärkungslernen

der Agent eine im Allgemeinen nicht deterministische Strategie π (Policy), mit der bei gegebenem Zustand s eine Aktion a gewählt wird, die einen Übergang in einen Folgezustand s' bewirkt. Dieser Zustandsübergang ist im Allgemeinen auch nicht deterministisch, d.h. bei gleichem Zustand und gleicher Aktion können abhängig von der Umgebung unterschiedliche Folgezustände auftreten.

Basierend auf der oberen Gleichung kann die sogenannte Bellman-Gleichung aufgestellt werden:

$$V^{\pi}(s) = \sum_{a}(\pi(s,a)\sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma V^{\pi}(s')\right])$$

Diese Gleichung beschreibt die Bewertung eines Zustandes ($V^{\pi}(s)$) (in Abhängigkeit von der Policy π) basierend auf dem unmittelbaren Reward $R_{ss'}$ und den Zustandsbewertungen der Folgezustände ($V^{\pi}(s')$). Dabei werden sämtliche möglichen Zustandsübergänge und die daraus resultierenden Rewards gewichtet mit einbezogen. Diese Gewichte hängen von den Wahrscheinlichkeiten ab, mit denen die verschiedenen Aktionen gewählt werden (Policy π), und von der Zustandsübergangswahrscheinlichkeit $P^{a}_{ss'}$, mit der in einer nichtdeterministischen Umgebung bei Wahl der Aktion a der Zustandsübergang von s nach s' ausgelöst wird. Ist die Policy π und/oder der Zustandsübergang bei gewählter Aktion deterministisch, vereinfacht sich obige Gleichung stark.

Wenn ein vollständiges Modell der Umgebung vorhanden ist und die Anzahl der Zustände endlich ist, kann die Bestimmung der Zustandswertefunktion beispielsweise mit quadratischer Programmierung erfolgen. Existiert kein Modell (Modell-freies Lernen), so kann das sogenannte Q-Learning angewendet werden, bei dem die Aktionswerte gelernt werden. Die Aktionswertefunktion wird analog zur Zustandswertefunktion bestimmt:

$$Q(s,a) = \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma \max_{a'} Q(s^{'}, a^{'})\right]$$

Diese Gleichung kann bei deterministischen Umgebungen während des Lernvorganges auch direkt für die Aktualisierung der Aktionswerte genutzt werden. Jedes Mal, wenn sich ein System in einem Zustand befindet und eine Aktion gewählt wird, wird der betreffende Aktionswert als Summe aus dem Reward und dem besten bekannten Aktionswert des Folgezustandes (multipliziert mit dem Discount-Faktor) gebildet. Nach mehreren Lerndurchläufen konvergieren die Zustandsbewertungen und es kann eine Strategie definiert werden, die immer die optimale Aktion wählt.

4.4.2. Aufgabenzuweisung als Reinforcement Learning Aufgabe

Ausgangspunkt ist, dass jede Teilaufgabe der Bildverarbeitungspipeline auf jedem der im Netzwerk verfügbaren Systeme ausgeführt werden kann. Hieraus ergibt sich in Abhängigkeit von der Anzahl der Teilaufgaben und der Anzahl der Systeme eine sehr hohe Anzahl von möglichen Konfigurationen. Von diesen sind aber nur wenige sinnvoll nutzbar. Daher

ist es im Hinblick auf die Rechenzeit unzweckmäßig, alle möglichen Konfigurationen zu analysieren. Das Ziel des Lernverfahrens besteht vielmehr darin, eine geeignete Verteilung in angemessener Zeit zu finden. Möglicherweise handelt es sich bei der gefundenen Verteilung nicht um die bestmögliche. Wie oben erwähnt, können als Kriterien für eine geeignete Verteilung Latenz der Bilddaten, mögliche Bildwiederholrate, CPU-Auslastung der Systeme und die Auslastung der Netzwerkschnittstellen definiert werden.

Innerhalb des Lernverfahrens wird die Zuweisung der Teilaufgaben als episodische Aufgabe betrachtet. Im Gegensatz zu kontinuierlichen Lernaufgaben ist die Lernprozedur hier in Episoden mit definierten Start- und Endzuständen unterteilt. Da die Auswertung der zahlreichen verschiedenen Konfigurationen auf den real existierenden Systemen zu zeitintensiv wäre, insbesondere wenn ein Robotersystem gerade im Einsatz ist, wird ein Modell für das Laufzeit-Verhalten der Bildverarbeitungspipelines erstellt, das im Abschnitt 4.4.3 erläutert wird. Im Hinblick auf das Lernverfahren wird hier trotzdem von Modell-freiem Lernen gesprochen, da das Q-Learning-Verfahren angewendet wird. Die Wahl dieses Verfahrens erfolgt deswegen, weil nicht alle möglichen Lösungen ausprobiert werden können/sollen.

Eine Reinforcement-Lernaufgabe besteht aus einer Menge Zustände S, einer Menge Aktionen A, Zustandsübergangsregeln und einem Reward-Signal R. Nachstehend erfolgt eine Definition dieser Elemente bezogen auf die hier vorliegende Lernaufgabe.

Zustände des Systems

Der Zustand des Systems gibt Auskunft darüber, welche Teilaufgaben bereits einem der verfügbaren Systeme zugeordnet sind. Dazu wird für jede der Teilaufgaben der Index des Systems gespeichert, dem diese Aufgabe zugeordnet wurde, oder -1, falls die Aufgabe noch nicht zugeordnet wurde. Somit besteht der Zustandsraum aus Vektoren, deren Länge der Anzahl der Teilaufgaben entspricht, in die die aktuelle Aufgabe unterteilt ist. Die Zustände sind folgendermaßen definiert:

$$S = \{a^n | a \epsilon \{-1, 0, 1, 2, ...k-1\}\}$$

wobei n die Anzahl der Teilaufgaben und k die Anzahl der verfügbaren Systeme ist. Für jede der Teilaufgaben werden die Abhängigkeiten definiert. Es wird also eine Liste von Teilaufgaben definiert, die abgeschlossen sein müssen, bevor die betreffende Teilaufgabe gestartet werden kann. Der Startzustand, bei dem noch keine Aufgabe zugeteilt ist, ist somit $S = \{-1^n\}$. Dieser Zustand kann auch dahingehend interpretiert werden, dass die Bilddaten auf dem Kamerasystem (bzw. den Kamerasystemen) verfügbar sind.

Endzustände sind solche, bei denen alle Teilaufgaben zugeordnet sind.

$$S_{end} = \{a^n | a \epsilon \{0, 1, 2, ...k-1\}\}$$

Der Endzustand beschreibt die Situation, bei der die Verarbeitung abgeschlossen ist und die Daten gegebenenfalls noch zu einem der Systeme übertragen werden, typischerweise zu dem Steuer-PC des Service-Roboters.

4.4.3. Modellierung der Zustandsübergänge

Bei jedem Zustandsübergang, der einem Lernschritt entspricht, wird eine der Teilaufgaben einem der verfügbaren Systeme zugeordnet. Bevor eine Teilaufgabe in dem Lernschritt einem System zugeordnet werden kann, müssen alle Datenabhängigkeiten erfüllt sein, das heißt alle Teilaufgaben, die Quelldaten für den Verarbeitungsschritt liefern, müssen bereits zugeordnet sein. Falls dieses auf mehrere Teilaufgaben zutrifft, wird diejenige ausgewählt, bei der die Abhängigkeiten zeitlich früher erfüllt sind - also dort, wo die vorherigen Elemente ihre Berechnungen zuerst abschließen.

Nach Wahl der Teilaufgabe wird das System ausgewählt, auf dem diese Teilaufgabe ausgeführt wird. Beim Reinforcement-Lernen kann man entweder explorieren (explore) oder ausnutzen (exploit). Beim Explorieren wird eine zufällige Aktion gewählt, während beim Ausnutzen die beste dem System bekannte Aktion gewählt wird. Das bedeutet hier, dass beim Ausnutzen das System ausgewählt wird, bei dem der Aktionswert am besten bewertet wird. Für den Fall, das beim Ausnutzen mehrere Aktionen gleich gut bewertet sind, wird zwischen diesen zufällig gewählt. Zu Beginn des Lernverfahrens werden alle Aktionswerte mit 0 initialisiert.

Jedes Mal, wenn ein dem System unbekannter Zustand eingenommen werden soll, werden für diesen basierend auf dem zugrunde liegenden Modell die vier oben erwähnten Attribute berechnet und anhand dieser und der Gewichtungsfaktoren wird eine Zustandsbewertung errechnet.

Die Latenz setzt sich ihrerseits aus mehreren Komponenten zusammen:

1. **Transfer-Zeit:** Die Daten müssen gegebenenfalls erst zu dem System übertragen werden, auf dem der aktuelle Verarbeitungsschritt ausgeführt wird. Dieses muss für alle Eingangsdaten erfolgen, sollten diese auf einem anderen System vorliegen. Der errechnete Wert für die Transfer-Zeit ergibt sich direkt aus der Bandbreite zwischen den Systemen und der zu übertragenden Datenmenge. Letztere wird anhand von Tests ermittelt. Da Netzwerkdatentransfer ebenfalls CPU-Last generiert, wird auch diese abgeschätzt und berücksichtigt.

2. **Warte-Zeit:** Das System könnte momentan gerade mit einer anderen Teilaufgabe ausgelastet sein. Es wird angenommen, dass erst dann mit der Bearbeitung der aktuell zugewiesenen Aufgabe begonnen wird, wenn die Bearbeitung der vorherigen Teilaufgabe abgeschlossen ist. Dies stellt eine deutliche Vereinfachung gegenüber dem tatsächlich erfolgenden Multitasking aktueller Betriebssysteme dar. Mit dieser Vereinfachung kann jedoch das Verhalten für viele Pipelines relativ gut modelliert

werden. Bei der Berechnung der Wartezeit können zudem nur solche Aufgaben berücksichtigt werden, die Teil der hier modellierten Bildverarbeitungspipeline sind. Andere Aufgaben, die auf den Systemen eventuell von anderen Nutzern ausgeführt werden, können nicht berücksichtigt werden. Die Hintergrundauslastung der Systeme muss also zuvor bekannt sein bzw. geschätzt werden. Die Zeiten, in der die Systeme belegt sind, werden in einer Liste gespeichert.

3. **Ausführungszeit:** Die Ausführungszeit ist die eigentliche Rechenzeit, die das System für die Bearbeitung der Teilaufgabe benötigt. Diese Zeitspanne wird hauptsächlich durch die Komplexität der Teilaufgabe sowie durch die Performance des Systems bestimmt. Auch die aktuelle Hintergrundlast hat Auswirkung auf diesen Wert. Es wird zur Modellierung die Ausführungszeit aus dem Testlauf zugrunde gelegt und diese durch das Verhältnis der Rechenleistungen der Systeme zueinander dividiert.
4. **End-Transfer:** Falls die aktuelle Teilaufgabe Ergebnisse bereitstellt, die zu einem bestimmten System übertragen werden müssen, wird die Netzwerk-Übertragungszeit hierfür ebenfalls diesem Verarbeitungsschritt zugerechnet. Falls der Verarbeitungsschritt nicht der letzte in der Pipeline ist, ist dieser Term nicht relevant. Ebenfalls wird dieser Term auf null gesetzt, wenn der Verarbeitungsschritt bereits auf dem Zielsystem stattfindet.

Diese Komponenten werden hintereinander modelliert und die Endzeit dieser Teilaufgabe wird in dem Folgezustand mit gespeichert. Bei der Beschreibung des Folgezustandes werden die sich ergebenen Auslastungszeiten für die Systeme und die Netzwerkschnittstellen ebenfalls gespeichert. Diese Zwischenspeicherung ist notwendig, um von diesem Zustand ausgehend in verschiedenen Lern-Episoden mehrere Folgezustände zu berechnen. Wird in einer späteren Episode ein schon bekannter Zustand eingenommen, muss die Berechnung nicht erneut erfolgen.

4.4.4. Berechnung der Rewards

Die Reward-Funktion gibt dem System eine Rückmeldung, ob die aktuelle Konfiguration sinnvoll ist oder nicht. Es werden sowohl Konfigurationen bewertet, bei denen noch nicht alle Teilaufgaben verteilt sind, als auch solche, bei denen alle verteilt sind. In Abhängigkeit von den aktuellen Anforderungen wird der Gesamt-Reward aus folgenden Komponenten gebildet:

- **Latenz:** Ein entscheidendes Ziel bei der Optimierung der Verarbeitungsstrategie ist es, die Zeit zu minimieren, die zwischen der Bildaufnahme und der Verfügbarkeit der extrahierten Bildinformation vergeht. Um den aktuell berechneten Wert der Latenz einzuordnen, wird dieser mit Daten verglichen, die zuvor bei einem Testlauf der gesamten Bildverarbeitungspipeline auf einem Referenzsystem aufgenommen wurden. Anhand dieses Vergleiches wird ein willkürlicher Wert zwischen -1 und 1 errechnet. Ein Wert von 0 bedeutet, dass die Latenzen gleich sind; 1 bedeutet, dass die Latenz

0,1 mal so groß ist wie die des Referenz-Systems (Verbesserung); -1 bedeutet, dass die Latenz 10 mal so hoch ist wie die des Referenzsystems. Dazwischenliegende Werte werden anhand folgender Formel errechnet:

$$R_{fps} = \begin{cases} -\frac{1}{9} \cdot (\frac{L_{mod}}{L_{cmp}} - 1) & \text{für } L_{cmp} < L_{mod} \\ 0 & \text{für } L_{mod} = L_{cmp} \\ -\frac{10}{9} \cdot (\frac{L_{mod}}{L_{cmp}} - 1) & \text{für } L_{cmp} > L_{mod} \end{cases}$$

Hierbei steht L_{mod} für die modellierte Latenz, L_{cmp} für den gemessenen Vergleichswert des Testsystems.

- **Bildwiederholrate:**
  Mit der Bildwiederholrate ist hier die Anzahl an Bildern gemeint, die pro Sekunde verarbeitet werden können. Da auf verschiedenen Systemen Parallelverarbeitung stattfindet, ist die Bildwiederholrate nicht notwendigerweise der Kehrwert der Latenz. Eine höhere Wiederholrate bedeutet, dass häufiger aktuelle extrahierte Bildinformationen eintreffen. Im zeitlichen Mittel hängt das Alter der aktuell verfügbaren Bildinformationen sowohl von der Bildwiederholrate als auch von der Latenz der eintreffenden Daten ab. Eine hohe Wiederholrate sorgt dafür, dass neue Daten eintreffen, bevor die letzten Daten zu stark veralten. Die angestrebte Bildwiederholrate ist so groß wie die Wiederholrate der Kamera. Für jedes System und jede Netzwerkschnittstelle werden die Informationen über die Belegung ausgewertet und daraus die maximal mögliche Wiederholrate errechnet. Der Reward ist 0, wenn die mögliche Wiederholrate gleich der im Testlauf ermittelten Wiederholrate bei dem letzten Element der Pipeline ist. Ein Reward von 1 wird vergeben, wenn die mögliche Bildwiederholrate 10 mal so hoch ist, ein Reward von -1 bedeutet, dass nur ein $\frac{1}{10}$ der Bildwiederholrate erreicht wird. Die Bereiche zwischen diesen Werten werden interpoliert.
  Der Reward wird nach folgender Vorschrift berechnet:

$$R_{fps} = \begin{cases} \frac{10}{9} \cdot (\frac{F_{mod}}{F_{cmp}} - 1) & \text{für } F_{cmp} > F_{mod} \\ 0 & \text{für } F_{mod} = F_{cmp} \\ \frac{1}{9} \cdot (\frac{F_{mod}}{F_{cmp}} - 1) & \text{für } F_{cmp} < F_{mod} \end{cases}$$

  Hierbei steht F_{mod} für die modellierte Bildwiederholrate, F_{cmp} für den Vergleichswert des Testsystems.
- **CPU-Auslastung:** Ein mögliches Motiv bei der Wahl einer Verarbeitungsstrategie kann sein, die Last auf einem bestimmten System, beispielsweise auf dem Steuer-PC des Service-Roboters, zu reduzieren. In Abhängigkeit von der Last lässt sich je System ein negativer Reward erzeugen. Der Reward wird zwischen 0 und -1 gewählt, wobei -1 bedeutet, dass die CPU des Systems zu 100 % ausgelastet ist. Die Last-Auswertung erfolgt anhand der Belegungszeiten der Systeme.Die Belegungsdauer pro Bild wird mit dem Kehrwert der Bildwiederholrate der Kamera verglichen. Ist die Belegungsdauer pro Bild größer, so ist damit zu rechnen, dass das System voll ausgelastet ist. Werden im Rahmen der Konfiguration mehrere Systeme als relevant gekennzeichnet, kann jedes dieser Systeme einen maximalen negativen Reward von -1 zum Gesamt-Reward

beitragen. Für jedes System kann ein Gewichtsfaktor K_s zwischen 0 und 1 definiert werden, über den gesteuert wird, wie stark der negative Reward in den Gesamtreward einfließt.

- **Netzwerk-Auslastung:** Analog zur Auslastung der Systeme kann auch die Netzwerkauslastung einen negativen Reward erzeugen. Es kann beispielsweise das Ziel einer Konfiguration sein, eine bestimmte Netzwerkverbindung nicht stark auszulasten, weil diese für weitere Operationen verfügbar bleiben soll. Die Errechnung des Rewards der Netzwerk-Auslastung erfolgt wie beim Reward der CPU-Last. Erfolgt die Berücksichtigung von mehreren Netzwerkverbindungen, so kann ebenfalls jede einen Reward zwischen 0 und -1 erzeugen und multipliziert mit Gewichtsfaktoren in den Gesamtreward einfließen.

Alle zuvor erwähnten Bestandteile werden zu einem Gesamt-Reward addiert. Um Prioritäten zwischen den einzelnen Attributen definieren zu können, werden die Einzel-Rewards jeweils mit einem Gewichtsfaktor multipliziert. Auf diese Weise kann eine Anpassung an die aktuelle Aufgabenstellung des Robotersystems und weitere Bedingungen erfolgen.

Die Berechnung des Gesamt-Rewards einer Verarbeitungsstrategie erfolgt anhand folgender Formel:

$$R_{tot} = K_{lat} \cdot R_{lat} + K_{fps} \cdot R_{fps} + \sum_{s=0}^{S-1} K_s \cdot R_{cpu-s} + \sum_{n=0}^{N-1} K_n \cdot R_{net-n}$$

Bei dieser Gleichung sind $K_{lat}, K_{fps}, K_0 ... K_{S-1}, K_0 ... K_{N-1}$ die Gewichtsfaktoren, über die das Lernverfahren gesteuert werden kann.

4.4.5. Ablauf des Lernverfahrens

Damit ein Robotersystem das Lernverfahren zur Laufzeit anwenden kann, müssen vorab einige Leistungsmessungen sowie Messungen der Netzwerkperformance durchgeführt werden. Diese Leisungsmessungen sind nicht dafür vorgesehen, ein exaktes Modell des dynamischen Verhaltens zu erzeugen, sie dienen jedoch dazu, das Verhalten des Systems zur Laufzeit näherungsweise nachzubilden.

Zur weiteren Vorbereitung muss die gesamte Pipeline auf einem Referenzsystem ausgeführt werden, um so die Komplexität der einzelnen Teilaufgaben abzuschätzen und um für die Rewardberechnung einen Referenzwert für die Latenz zu erhalten, mit dem die später modellierten Latenzen dann verglichen werden. Aus einer Logdatei, wie sie in Abschnitt 3.2.12 beschrieben wird, generiert das System automatisch eine Liste der Teilaufgaben und deren Abhängigkeiten. Zunächst wird im Programm bei der Ausführung der Pipeline bei jedem Datentransfer zwischen zwei Elementen eine sogenannte Callback-Funktion aufgerufen, die folgende Informationen in diese Log-Datei speichert:

- Quellelement

- Zielelement
- NTP-Zeitstempel (aktuell)
- NTP-Zeitstempel der Bildaufnahme
- Datengröße

Die Log-Datei wird dann von dem eigentlichen Lernprogramm eingelesen und ausgewertet. Die Ausführungszeiten der einzelnen Elemente werden in diesem Schritt basierend auf diesen Daten ebenfalls analysiert. Der eigentliche Lernprozess ist als eine Variante des Q-Lernens [WD92] implementiert:

Wiederhole:

- *Setze den aktuellen Zustand auf den Startzustand.*
- *Innere Wiederholung:*
  - *Wähle eine Aktion a und führe sie aus.*
  - *Erhalte einen Reward r.*
  - *Bestimme Folgezustand s′.*
  - $\hat{Q}(s,a) \leftarrow r + \gamma \cdot \max_{a'} \hat{Q}(s',a')$
  - $s \leftarrow s'$

  Bis s ein Endzustand ist.

Bis die maximale Anzahl an Iterationen erreicht ist oder alle Endzustände evaluiert sind.

Programmintern werden die Zustände zu der Liste der bekannten Zustände hinzugefügt, wenn sie das erste Mal eingenommen werden. Alle unbekannten Aktions-Werte werden mit null initialisiert. Beim Programmablauf werden die Endzustände in einer separaten Liste gespeichert. Letztendlich wird nur unter diesen Endzuständen nach demjenigen gesucht, der am besten bewertet ist. Aus dem Endzustand lässt sich direkt die dazugehörige Verteilung ableiten. Die Zwischenzustände haben hierbei keine Bedeutung, sie dienen hauptsächlich dazu, zu steuern, welche der theoretisch möglichen Endzustände erkundet werden.

4.4.6. Beispiel 1: Skalierung des Bildes

Als einfaches Beispiel für die automatische Aufgabenverteilung wird folgende Anwendung betrachtet: Die Bilddaten der intelligenten Kamera sollen in verringerter Auflösung auf einen weiteren PC vergleichbar mit dem Steuerrechner des Service-Roboters TASER (Intel Pentium 4) übertragen werden. (s. Abbildung 4.43). Als Referenzsystem wird ein weiterer PC mit Intel Core i5 Prozessor genutzt.

Es ergeben sich zwei mögliche Konfigurationen:

- Transfer des Bildes in ursprünglicher Auflösung, Skalierung auf dem PC
- Skalierung auf der intelligenten Kamera, Transfer der verkleinerten Daten

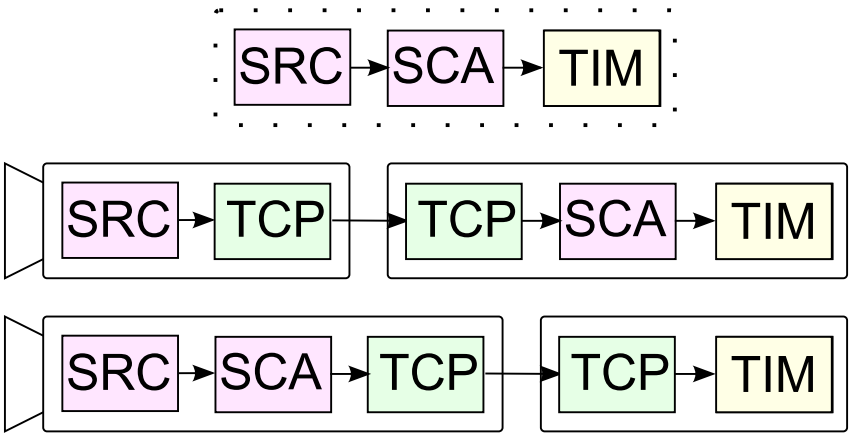


Abbildung 4.43.: Skalierung des Bildes, mögliche Konfigurationen.
Oben: Logischer Aufbau der Pipeline. Die Daten werden von dem Quellelement (SRC, auf der intelligenten Kamera) zum Element geleitet, das die Skalierung durchführt (SCA). Von dort sollen sie zum Zeitnahme-Element (TIM) auf dem PC geleitet werden.
Mitte: Konfiguration der Systeme, bei der die Skalierung auf dem PC erfolgt.
Unten: Skalierung auf der intelligenten Kamera.

Legende der verwendeten Elemente:
SRC: Kapselt den Treiber der Kamerahardware
SCA: Skaliert Bilder, ändert die Auflösung
TCP: Überträgt Daten per TCP-Verbindung
TIM: Vergleicht Zeitstempel der Daten mit aktueller NTP-Zeit

Folgende Messwerte liegen dem Lern-Modell zugrunde:

- Netzwerk-Bandbreite: 30 MB/s (bei Kommunikation zwischen Kamera und PC)
- Skalierung auf dem Referenz-PC: $0,5$ ms
- Skalierung auf der intelligenten Kamera, von 1388x1038 Pixeln 8-bit monochrom zu 640x480 Pixeln: $7,8$ ms (Leistungsindex 0,065)
- Skalierung auf dem Steuerrechner: $1,8$ ms (Leistungsindex 0,28)

Das Lernziel ist, eine Verteilung mit möglichst geringer Latenzzeit zu finden, weitere Kriterien sind nicht relevant. Somit wird $K_{lat} = 1$ gesetzt, alle weiteren Kostenfaktoren sind null. Das Lernverfahren terminiert, wenn beide möglichen Endzustände ausgewertet sind. Der Parameter zur Steuerung der Explorationstemperatur τ wird auf 0,7 gesetzt, das heißt, das System exploriert in 30 % aller Fälle und nutzt in den übrigen Fällen die beste bekannte Strategie. Für die beiden möglichen Setups ergeben sich folgende errechnete Bewertungen:

Setup	Reward (Beurteilung)	Latenz (modelliert in ms)
Skalierung auf PC	0,22	49,7
Skalierung auf eXcite	0,80	17,4

Wie anhand der Daten zu sehen ist, wird die Lösung bevorzugt, bei der die Skalierung auf der intelligenten Kamera erfolgt. Ausschlaggebend hierfür ist, dass so eine geringere Datenmenge über das Netzwerk übertragen werden muss. Der hierdurch entstehende Vorteil bezüglich der Latenz wiegt deutlich schwerer als die geringere Verarbeitungsgeschwindigkeit der intelligenten Kamera. Beide Setups werden zum Vergleich auf den realen Systemen getestet. Die Latenz beträgt 19 ms bei der Skalierung auf der Kamera und schwankt zwischen 46 und 48 ms bei der Skalierung auf dem PC. Somit können die Praxistests die vorhergesagte Latenz bestätigen. Es ergeben sich zwar gewisse Abweichungen, diese sind jedoch relativ gering und beeinflussen nicht die generelle Funktion des Lernverfahrens.

Eine komplett andere Bewertung ergibt sich, wenn der Skalierungsalgorithmus mit Interpolation ausgeführt wird (rechenintensiver, dafür bessere Qualität). Hierzu wird der oben beschriebene Test mit anders konfiguriertem Skalierungsalgorithmus wiederholt. Dieser Algorithmus benötigt ungefähr 5 ms auf dem Referenz-PC (gemessen). Folgende Bewertungen werden in diesem Fall von dem Lernsystem vorgenommen.

Setup	Reward (Beurteilung)	Latenz (modelliert in ms)
HQ-Scaling auf PC	-0,03	65,3
HQ-Scaling auf eXcite	-0.07	84,6

Aufgrund der deutlich höheren Rechenleistung ergibt sich in diesem Fall bei Skalierung auf dem PC die bessere Gesamtlatenz und somit die bessere Gesamtbewertung.

Die in diesem Abschnitt vorgestellten Experimente zeigen die grundlegende Funktion der automatischen Zuweisung der Teilaufgaben. Es ist jedoch anzumerken, dass für diese Aufgabenstellung kein Lernverfahren notwendig ist, da nur zwei verschiedene Konfigurationen

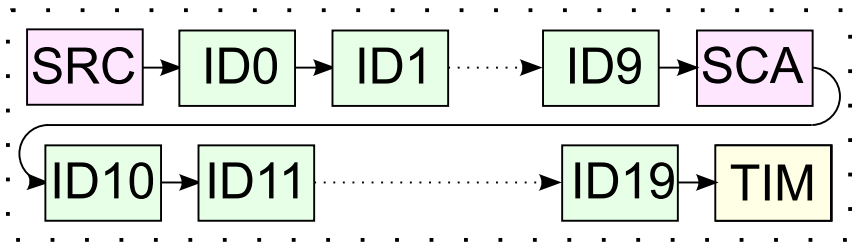


Abbildung 4.44.: Automatische Aufgabenverteilung bei langer Pipeline. Diese Abbildung zeigt den logischen Aufbau der Pipeline mit zusätzlichen Elementen, die die Daten nur weiterleiten, ohne diese zu modifizieren (ID0-ID19). Die anderen Elemente sind äquivalent zu Abbildung 4.43. An (mindestens) einer Stelle müssen in die reale Pipeline die Elemente zur TCP-Datenübertragung integriert werden, da die Daten von der Kamera zu dem PC übertragen werden müssen.

zu untersuchen sind. Der eigentliche Zweck des Lernverfahrens ist jedoch, nur erfolgversprechende Konfigurationen zu evaluieren.

4.4.7. Beispiel 2: Lange Pipeline

Da im ersten Beispiel die Vorteile des Lernverfahrens nicht zum Tragen kommen, werden diese an einem zweiten Beispiel gezeigt. Dieser Test ist von der Aufgabe her vergleichbar mit dem ersten Test, bei dem die schnelle Version des Skalierungsalgorithmus verwendet wurde. Der Unterschied besteht darin, dass jetzt in die Bildverarbeitungspipeline vor und hinter das Skalierungs-Element jeweils zehn Elemente eingefügt sind, die eingehende Daten ohne Modifikation weiterleiten. Der logische Aufbau der Pipeline ist in Abbildung 4.44 gezeigt. Diese Abbildung zeigt lediglich die verwendeten Elemente und deren Verbindung, nicht jedoch das System, auf dem sie ausgeführt werden.

Die Ausgangssituation bei der automatischen Verteilung der Teilaufgaben ist, dass 21 Elemente vorhanden sind, die jeweils entweder auf der Kamera oder auf dem PC ausgeführt werden können. Für den menschlichen Betrachter wird sofort klar, dass nur diejenigen Konfigurationen sinnvoll sind, bei denen die Daten nur einmal zwischen der Kamera und dem PC übertragen werden. Ein solches Experten-Wissen ist jedoch auf dem Lernsystem nicht vorab vorhanden. Daher soll das System „lernen“, welche von den 2^{21} möglichen Setups vorteilhaft sind und welche nicht.

In den Tests wird folgende Lösung als beste vorgeschlagen (Reward: 0,55, Latenz: 20,7ms): Alle Elemente vom ersten Identitäts-Element bis einschließlich des Skalierungs-Elements werden der intelligenten Kamera zugeordnet, alle folgenden Elemente werden dem PC zugeordnet. Bei genauerer Betrachtung fällt auf, dass das System mehrere Umstände betrachtet hat. Zunächst wurde erkannt, dass die Daten nur einmal zwischen der Kamera

und dem PC übertragen werden sollten. Der Vorteil durch die Vorverarbeitung auf der Kamera wurde ebenfalls berücksichtigt. Auch die Entscheidung, direkt nach dem Skalierungselement die Daten zu transferieren, ist begründet, da das Weiterleiten der Daten in den Identitäts-Elementen auf dem PC theoretisch minimal schneller ausgeführt wird. In der Praxis hat letzteres nur geringen Einfluss, zeigt aber sehr deutlich die Funktionsweise des Lernverfahrens.

Anhand dieser Aufgabe soll ebenfalls die Leistungsfähigkeit des Lernverfahrens evaluiert werden im Hinblick auf die Anzahl an Iterationen, die notwendig sind, um eine sinnvolle Strategie zu entwickeln. Nach jeder Lernepisode wird die aktuell beste bekannte Lösung hinsichtlich ihrer Latenz betrachtet. Das Lernverfahren wird mit verschiedenen Explorationstemperaturen verglichen, einmal mit $\epsilon = 0,7$, einmal mit $\epsilon = 0,0$. In letzterem Fall werden also die Aktionen immer zufällig ausgewählt. Das Ergebnis dieses Tests ist in Abbildung 4.45 dargestellt. Es ist erkennbar, das bei Ausnutzung des Reinforcement-Lerning-Ansatzes nach ca. 300 Iterationen eine sinnvolle, nahezu optimale Lösung gefunden wird, bei der zufälligen Evaluation der Strategien jedoch nicht. Mathematisch liegt die Chance, bei 300 zufällig gewählten Aktionen mindestens eine sinnvolle Endkonfiguration zu finden, in der Größenordnung von 0.16 %.

Diese Abschätzung beruht auf folgenden Annahmen:

- Solche Konfigurationen werden als sinnvoll betrachtet,
  - bei denen die Skalierung auf der Kamera erfolgt und
  - bei denen die Daten nur einmal zum PC übertragen werden.
- Es gibt somit 11 sinnvolle Konfigurationen.
- Insgesamt gibt es 2^{21} mögliche Konfigurationen.
- Die Wahrscheinlichkeit des Gegenereignisses, dass in 300 Episoden keine sinnvolle Konfiguration gefunden wird, ergibt sich zu:
  $(\frac{2^{21}-11}{2^{21}})^{300} = 0.9984$
- Somit ergibt sich eine Wahrscheinlichkeit von 0.16 %.

4.4.8. Lernziel: ROI Übertragung basierend auf Gesichtslokalisation

Die folgenden Untersuchungen beziehen sich auf die Pipelines zur Übertragung der Regions-of-Interests, die anhand von Gesichtslokalisation ausgewählt werden (vgl. Abschnitt 4.2). Die Aufgabe des Systems besteht wie zuvor in der Lokalisierung von Gesichtern in dem Videodatenstrom mit einer verringerten Auflösung von 320 x 240 Pixel. Die Regionen, in denen Gesichter erkannt werden, werden dann aus dem Videodatenstrom mit voller Auflösung (1388 x 1038 Pixel) ausgeschnitten. Die Videodaten liegen beim Startzustand auf der intelligenten Kamera vor und der Endzustand tritt ein, wenn alle Daten auf dem Steuer-PC vorliegen. Der logische Aufbau der Pipeline und die Ergebnisse des Lernverfahrens sind in Abbildung 4.46 gezeigt.

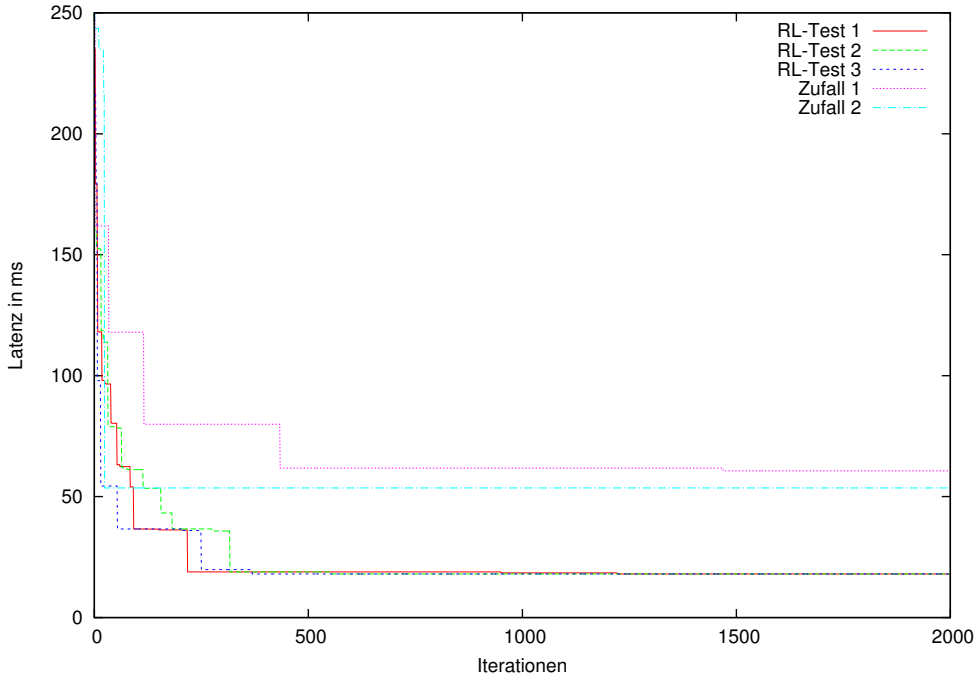


Abbildung 4.45.: Die Abbildung zeigt die Latenz der besten bekannten Konfiguration in Abhängigkeit von der Zahl der Lernepisoden. Es sind drei Verläufe dargestellt, bei denen Reinforcement Learning benutzt wird (70% ausnutzen, 30% explorieren), und zwei Verläufe mit zufälliger Aktionswahl. Nach ungefähr 300 Episoden wird mithilfe der Reinforcement-basierten Lernverfahrens eine sinnvolle Lösung gefunden.

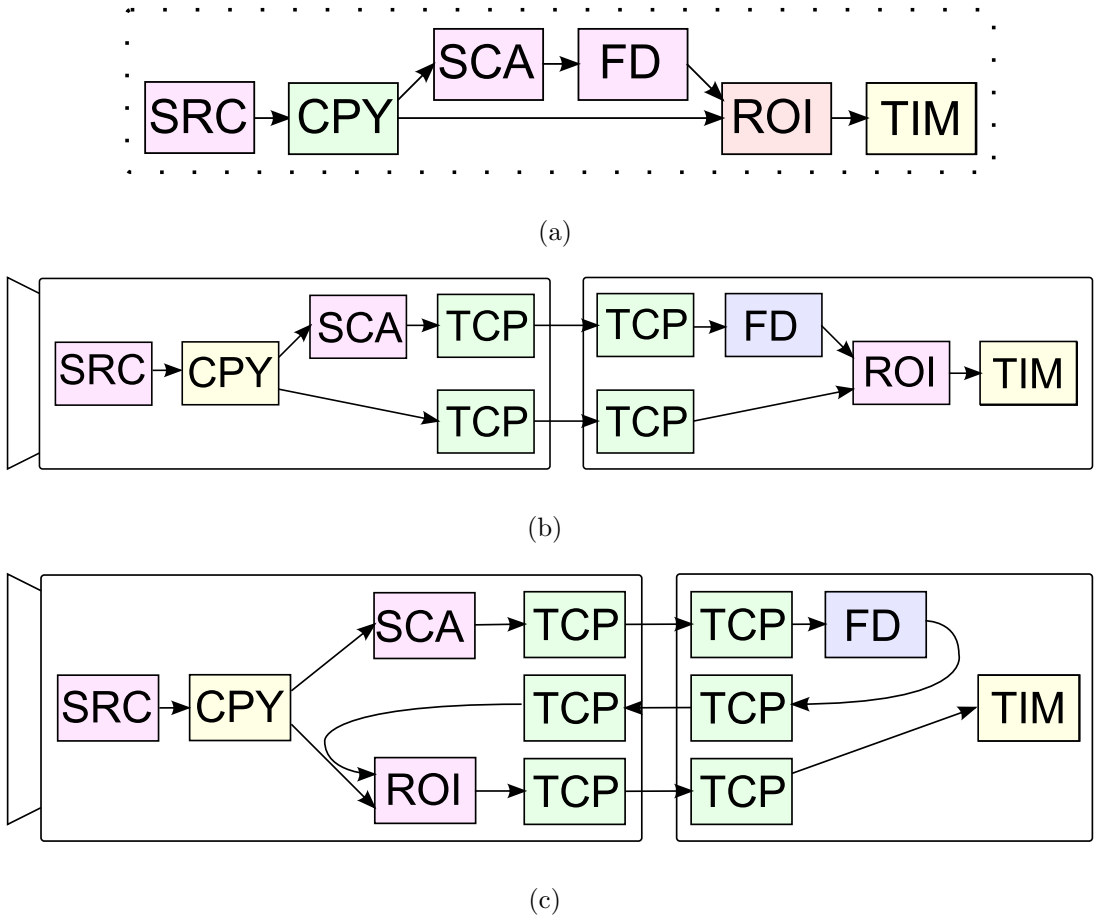


Abbildung 4.46.: Automatische Aufgabenverteilung: Lernziel ROI Übertragung basierend auf Gesichtslokalisation. Diese Abbildungen zeigen die verschiedenen Pipeline-Konfigurationen für die ROI-Extraktion basierend auf Gesichtslokalisation. Der logische Pipelineaufbau ist in (a) gezeigt, (b) zeigt die vorgeschlagene Aufteilung, wenn nur die Latenz Einfluss auf den Reward hat. Konfiguration (c) wird gewählt, wenn sowohl Latenz als auch Netzwerkauslastung den Reward beeinflussen. Zusätzliche Elemente:
FD: „Face Detection“, dient zur Lokalisation von Gesichtern
ROI: extrahiert ROIs aus einem Bild.

Im ersten Durchlauf des Lernverfahrens hat nur die Gesamtlatenz Einfluss auf den Reward. Als Ergebnis wird eine Konfiguration vorgeschlagen, bei der die intelligente Kamera sowohl die Bilder in verringerter Auflösung als auch die voll aufgelösten Bilder überträgt. Auf dem Steuer-PC erfolgen dann die Gesichtslokalisation und die Extraktion der Regions-of-Interest (vgl. Abbildung 4.46(b)). Dieses Ergebnis ist unerwartet, da es entgegen der Logik ist, redundante Daten zu senden. Die Latenz für diese Konfiguration wird auf 212 ms geschätzt, was einem Reward von -0,005 entspricht. Bei der Modellierung des Laufzeitverhaltens werden von dem Lernsystem folgende Annahmen zugrunde gelegt: Erst nachdem das verkleinerte Bild komplett übertragen wurde, wird der Transfer des Bildes in voller Auflösung gestartet. Während auf dem Steuer-PC also die Bildverarbeitung zur Gesichtslokalisation läuft, wird im Hintergrund das voll aufgelöste Bild empfangen. Das eigentliche Ausschneiden der Regionen kann dann auf dem Steuerrechner mit deutlich höherer Performance erfolgen als dieses auf der intelligenten Kamera möglich wäre.

An diesem Beispiel zeigen sich auch die Grenzen des implementierten Lernmodells. Die tatsächliche Pipeline würde sich auf diese Art konfiguriert zur Laufzeit anders verhalten: Die Übertragung des voll aufgelösten Bildes und die Erzeugung des niedrig aufgelösten Bildes sowie dessen Übertragung würden auf der Kamera parallel erfolgen. Daher wäre die tatsächliche Latenz deutlich schlechter. Durch ein zusätzliches Paar von „Trigger"-Elementen kann jedoch genau der vom Lernsystem vorgeschlagene Ablauf herbeigeführt werden. So kann auf dem Steuer-PC festgestellt werden, wann die Übertragung des niedrig aufgelösten Bildes abgeschlossen ist. Erst dann wird die Übertragung des voll aufgelösten Bildes durch das „Trigger"-Element auf der intelligenten Kamera gestartet. Auf diese Art könnte der Vorschlag des Lernsystems in die Praxis umgesetzt werden.

In den Tests, die im Rahmen dieser Arbeit erfoltgen und auch Gegenstand der Veröffentlichung [BZ09] sind, wird eine manuell erstellte Konfiguration, wie sie in Abbildung 4.46(c) gezeigt ist, als beste evaluiert. Bei dieser Konfiguration erfolgt die Gesichtslokalisation auf dem Steuerrechner, die Information über die Regions-of-Interest wird jedoch an die intelligente Kamera zurückgesendet, auf der diese dann extrahiert und übertragen werden. In dem oben beschriebenen Lernverfahren, bei dem nur die Latenz ausschlaggebend ist, wird diese Konfiguration als zweitbeste mit einer Latenz von $213,5\ ms$ und einem Reward von -0,006 bewertet.

Das Lernverfahren wird erneut gestartet, jedoch wird diesmal auch die Auslastung des Netzwerks in gleichem Maße wie die Latenz berücksichtigt. Es wird nunmehr die Konfiguration wie in Abbildung 4.46(c) mit einem Reward von -0,03 favorisiert, während die zuvor vorgeschlagene Konfiguration aus Abbildung 4.46(b) nur noch einen Reward von -0,58 erreicht.

In einem dritten Test wird der Gewichtsfaktor für die Latenz auf null gesetzt. Somit hat nur noch die Netzwerkauslastung Einfluss auf den Reward. Hierbei wird nun vorgeschlagen, dass alle Teilaufgaben auf der intelligenten Kamera ausgeführt werden sollten (ohne Abbildung).

Mit diesen Tests kann gezeigt werden, dass das Reinforcement-basierte Lernverfahren in

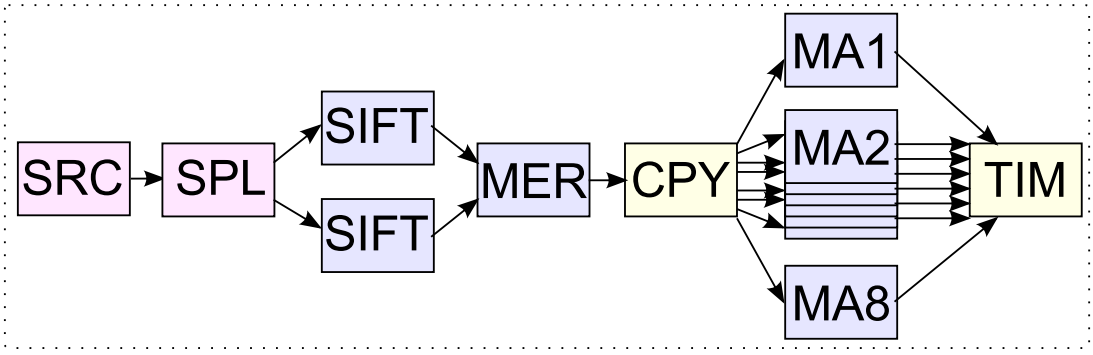


Abbildung 4.47.: Automatische Aufgabenverteilung: Lernziel Objektdetektion. Diese Grafik zeigt die relevanten Elemente der Pipeline zur verteilten SIFT-basierten Objekterkennung. Das Bild wird zunächst in zwei überlappende Teilbilder geteilt (SPL), die Feature-Vektoren werden für jedes dieser Teile unabhängig voneinander berechnet (SIFT) und zu einem einzigen Feature-Vektor verrechnet (MER), dupliziert (CPY) und mit den zu erkennenden Objekten verglichen (MA1-MA8).

der Lage ist, für praxisrelevante Aufgaben eine Aufteilung auf die verschiedenen Teilsysteme zu errechnen, und hierbei auch Nutzervorgaben berücksichtigen kann. Die errechneten Latenzwerte liegen in einem Bereich, der auch in den Tests in Abschnitt 4.2 ermittelt wird.

4.4.9. Lernziel: Objektdetektion

Wie im Abschnitt 4.3 gezeigt wird, ermöglicht Cloud-Computing bei der Objekterkennung deutliche Leistungssteigerungen. Der Feature-Vektor wird hierbei auf zwei lokalen dedizierten PCs auf dem Service-Roboter berechnet und dann über eine kabellose Netzwerkverbindung zu acht Desktop-PCs in der Büroumgebung des Roboters gesendet. Dort werden die bekannten Objekte durch einen Vergleich mit jeweils einem von acht Teilen einer Datenbank ermittelt (insgesamt 100 Objekte). Nunmehr wird mit Hilfe des beschriebenen Lernalgorithmus versucht, automatisch eine sinnvolle Verteilung der Teilaufgaben zu bestimmen. Der Aufbau der Pipeline ist noch einmal - diesmal bezogen auf die automatische Verteilung - in Abbildung 4.47 gezeigt. Da in dem Modell noch keine Mehrkern-CPUs berücksichtigt werden, wird hier ein Pipeline-Aufbau verwendet, bei dem nur zwei Teilbilder erzeugt werden, von denen dann jeweils eines auf einem der dedizierten Systeme verarbeitet werden sollte.

In diesem Fall gibt es $13^{12} = 23,3 \cdot 10^{12}$ mögliche Systemkonfigurationen, von denen nur die wenigsten sinnvoll sind. Eine Ausführung auf dem Referenzsystem ergibt insgesamt eine Verarbeitungszeit von 10,1 Sekunden. Basierend auf den gemessenen Verarbeitungszeiten

erfolgt die Ausführung des Lernverfahrens. Vom Lernsystem wird dabei angenommen, dass die gleichen (leistungsfähigeren) Systeme verfügbar sind wie in der späteren Testreihe der in Abschnitt 4.3 beschriebenen Tests.

Die vom Lernsystem gewählte Lösung (Reward 0,99 , Latenz 1,34 Sekunden) unterscheidet sich von der manuell gewählten und als optimal anzusehenden Verteilung (Setup 2) lediglich dadurch, dass auf einem der acht Systeme in der Umgebung zwei „Siftfolder"-Elemente ausgeführt werden und eines der Systeme ungenutzt bleibt.

Als Ursache hierfür stellen sich in den aufgezeichneten Testdaten stark unterschiedliche Ausführungszeiten der verschiedenen „Siftfolder"-Elemente heraus. Zum einen bestimmt die Anzahl der Merkmalspunkte die Verarbeitungszeit, zum anderen wird zusätzliche Rechenzeit benötigt, wenn eine Transformation gefunden und bestimmt wird. In dem vorliegenden Fall ist die Rechenzeit eines Siftfolder-Elements größer als die Summe der Rechenzeiten zweier anderer Elemente. Aus diesem Grund werden letztere zwei Elemente einem System zugeordnet.

In einem weiteren Test werden dieselben Bilddaten für alle Siftfolder-Elemente verwendet. Das Ergebnis dieses Tests ist, dass exakt die gleiche Verteilung vorgeschlagen wird, wie sie manuell gewählt wird (Reward 1,0 Latenz 0,8 Sekunden).

Bei wiederholter Ausführung ergibt sich die Situation, dass gelegentlich eine andere Lösung favorisiert wird, die geringfügig schlechtere Ergebnisse liefert. Es werden jedoch folgende Strategien zuverlässig gelernt:

- Verwendung der intelligenten Kamera zur Aufteilung der Bilddaten
- parallele Berechnung des SIFT-Vektors unter Benutzung beider dedizierter Systeme
- einmaliges Senden des SIFT-Vektors über WLAN und Weiterverteilung im Kabelbasierten Netzwerk

Bezüglich der teilweise nicht gefundenen optimalen Lösung sind in Zukunft noch weitere Optimierungen des Lernverfahrens notwendig. Des Weiteren ist einschränkend anzumerken, dass die Parallelität vom Benutzer vorgegeben werden muss, also hier die Aufteilung der Berechnung des SIFT-Vektors in zwei Teile und die Durchführung des Feature-Vergleichs in acht Teilen. Ebenso werden noch keine Multiprozessorsysteme modelliert.

4.4.10. Möglichkeiten zur Weiterentwicklung der automatischen Zuweisung

Im Rahmen der Arbeit werden nur die grundlegenden Methoden erarbeitet. Um ein vollständig autonomes System zu implementieren, müssten noch folgende Funktionen realisiert werden:

- Automatisches Benchmarking der verschiedenen Systeme
- Service Discovery-System, um automatisch nach verfügbaren Systemen im lokalen Netzwerk zu suchen, beispielsweise basierend auf Avahi oder Roblets

- Echtzeit Monitoring der verschiedenen Rechnerauslastungen und Bewertung der Priorität sonstiger Aufgaben
- Implementierung einer automatischen Umschaltung der Konfigurationen zur Laufzeit

Im Hinblick auf das Ergebnis bei der Objektdetektion ergibt sich noch Optimierungspotential am Lernalgorithmus. Auch könnte im System vorab sogenanntes „Expertenwissen“ implementiert werden, so dass beispielsweise versucht wird, nicht genutzte Systeme einzubeziehen.

4.5. Overhead bei Interprozesskommunikation

In den folgenden Tests soll der Frage nachgegangen werden, wie sich die in dieser Arbeit gewählte Methode zur Implementierung von modularen Verarbeitungsfunktionen im Vergleich zu anderen Frameworks im Hinblick auf den Overhead verhält, der entsteht, wenn verschiedene Programmkomponenten Daten austauschen. Hier wird ein direkter Vergleich zu dem ROS-Framework durchgeführt, da dieses Framework die größte Verbreitung aufweist. Von anderer Seite sind bereits Ergebnisse zum Vergleich von verschiedenen Frameworks publiziert. Die Ergebnisse werden hier herangezogen, um eine Bewertung durchzuführen, wie sich die Performance der hier getesteten Methoden im Vergleich zu anderen Frameworks verhält [ELS+12].

Die Tests erfolgen stets innerhalb des GStreamer-Frameworks unter Benutzung verschiedener Mechanismen zur Datenübertragung:

- ROS über „gstros“ (ohne Downstream Buffer Allocation)
- ROS über „gstros“
- GStreamer TCP
- GStreamer Shared Memory (ohne Downstream Buffer Allocation, nur lokal)
- GStreamer Shared Memory (nur lokal)
- Weiterleitung innerhalb einer Pipeline (nur innerhalb eines Prozesses, nur lokal)

Die Setups der Pipelines für die Tests sind in den meisten Fällen ähnlich, mit Ausnahme des Tests zur Weiterleitung innerhalb einer Pipeline. Die Setups werden in Abbildung 4.48 und 4.49 erläutert. In jedem Testdurchlauf wird die Latenz von 1000 Bildern evaluiert und der Mittelwert gebildet. Bei diesen Tests wird ebenfalls die CPU-Auslastung gemessen, die durch den Sender und den Empfänger verursacht wird.

Die Ergebnisse der Tests, die auf einem PC mit Intel Core i5-3570 CPU durchgeführt werden, sind in den Abbildungen 4.50 und 4.51 dargestellt. Es ergibt sich, dass der Datentransfer innerhalb des ROS-Frameworks die höchste Latenz sowie die höchste CPU-Auslastung erzeugt. Dennoch können im Vergleich zur in ROS existierenden GStreamer-Anbindung durch die im Rahmen dieser Arbeit implementierte Anbindung ungefähr 5 % CPU Last und 2 ms Latenz bei einem FullHD Bilddatenstrom mit 30 fps eingespart werden (Die

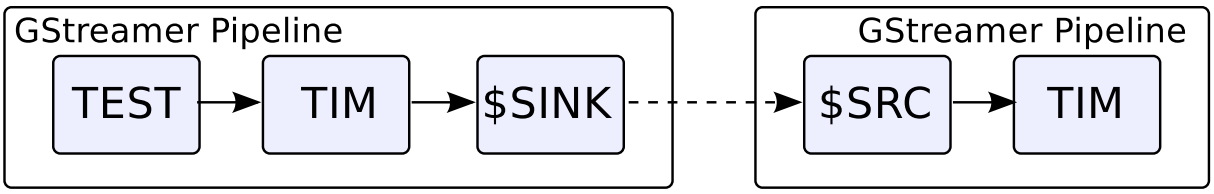


Abbildung 4.48.: Diese Abbildung zeigt die Testkonfiguration zur Ermittlung des Overheads bei Interprozesskommunikation verschiedener Instanzen auf einem System. Die Elemente $SINK und $SRC stehen für die Elemente der jeweiligen Übertragungsmethode. Über „Timestamper“-Elemente wird der Zeitstempel gesetzt und ausgelesen. Auf diese Art wird die Zeitspanne gemessen, die vergeht, bis die Daten übertragen worden sind. Es bestünde auch die Möglichkeit, mehrere Empfängerpipelines zu starten.

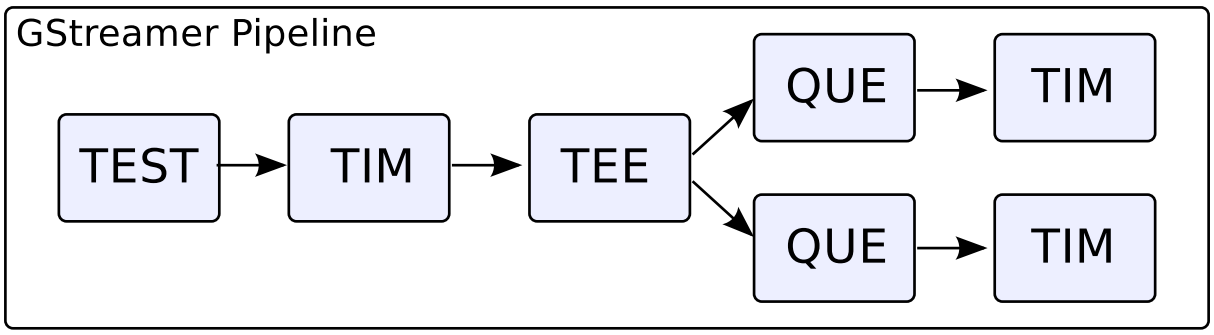


Abbildung 4.49.: Diese Abbildung zeigt den Testaufbau zur Intraprozesskommunikation, wie sie innerhalb von GStreamer verwendet wird, wenn mehrere Elemente auf dieselben Daten zugreifen müssen. Über das „Tee“-Element werden die Daten dupliziert und auf die einzelnen Pfade verteilt. Hierbei werden nur Pointer auf den jeweiligen Datenbereich weitergegeben. Für jeden Pfad, auf dem die Daten verarbeitet werden, wird ein eigener Thread gestartet.

existierende „gscam“-Software entspricht von der Performance her dem „gstros“-Element ohne „Downstream Buffer Allocation“).

Der Datentransfer erfolgt signifikant schneller mittels der anderen Übertragungsmethoden. Der von GStreamer verwendete TCP-basierte Mechanismus versendet die Daten ungefähr drei- bis viermal so schnell wie der ROS-Mechanismus. Da in den Tests auch mit 1 MB großen Nutzdaten gearbeitet wird, können die Ergebnisse direkt mit denen aus [ELS+12] verglichen werden, wo ein etwas schnellerer PC mit Intel Core i7 Prozessor genutzt wird. Die Latenz bei ROS beträgt dort etwa 3 ms und als schnellstes Framework schneidet YARP [MFN06] mit einer Latenz von etwa 0,8 bis 0,9 ms ab. Die hier durchgeführten Tests bestätigten die Messung bezüglich ROS (2,6 ms bzw. 3,0 ms ohne „Downstream Buffer Allocation“). Der TCP-basierte Kommunikationsmechanismus von GStreamer ist mit einer hier gemessenen Latenz von 0,7 ms somit bereits in der Lage, sämtliche anderen dort evaluierten Frameworks zu übertreffen.

Der hier zusätzlich implementierte Mechanismus basierend auf Shared Memory arbeitet im Vergleich mit einer Latenz von 0,2 ms nochmals deutlich schneller. Diese Performance kann jedoch nur erreicht werden, wenn mit „Downstream Buffer Allocation“ gearbeitet wird. Anderenfalls fällt die Performance hinter die TCP-basierte Methode von GStreamer zurück. Somit ist als Ergebnis dieser Tests festzuhalten, dass die Vorteile der Shared Memory basierten Kommunikation nur dann voll zum Tragen kommen, wenn das vorherige Element direkt in den entsprechenden Speicherbereich schreiben kann. Der geringste Overhead ergibt sich erwartungsgemäß, wenn sich die verschiedenen Software-Komponenten, also bei GStreamer die Elemente, alle innerhalb einer Pipeline befinden. Hier spielt die entstehende Verzögerung bei der Aufteilung in Threads keine Rolle. Es ist jedoch notwendig, dass schon vor dem Startzeitpunkt der Pipeline sämtliche Software-Komponenten konfiguriert werden. Über Elemente des Typs „Valve“ können je nach Bedarf einige Pfade zur Laufzeit deaktiviert werden.

Die gemessene CPU-Auslastung bestätigt das Ergebnis, dass der Datenaustausch innerhalb des ROS-Frameworks die ineffizienteste der hier getesteten Methoden ist. In der gemessenen CPU-Auslastung auf der Senderseite ist die etwa 25 % betragende Last zur Erzeugung des Testsignals mit enthalten. Somit ergibt sich, dass der Datentransfer über ROS eine etwa viermal so hohe zusätzliche CPU-Auslastung wie die TCP-basierte Datenübertragung von GStreamer oder die Kommunikation über Shared Memory aufweist (CPU-Last durch den Empfänger eingerechnet).

In einer weiteren Testreihe werden CPU-Last und Latenz ermittelt, wenn die Daten von einem Sender zu mehreren Empfängern übertragen werden. Da hierbei auch die im Rahmen dieser Arbeit implementierte Methode getestet werden soll, JPEG-komprimierte Daten von GStreamer an ROS zu übertragen, wird die Webcam Logitech C910 verwendet, die mittels des „video4linux2“-Elements (vorgegeben) hierzu in der Lage ist. Die Kamera hat eine variable Bildwiederholrate, die bei Beleuchtung unter den gegebenen Testbedingungen etwa 15 fps beträgt. Die Dekodierung nach RGB erzeugt etwa 30 % CPU-Auslastung und dauert pro Bild etwa 17 ms. Im Testverlauf wird das Verhalten zunächst ohne, dann

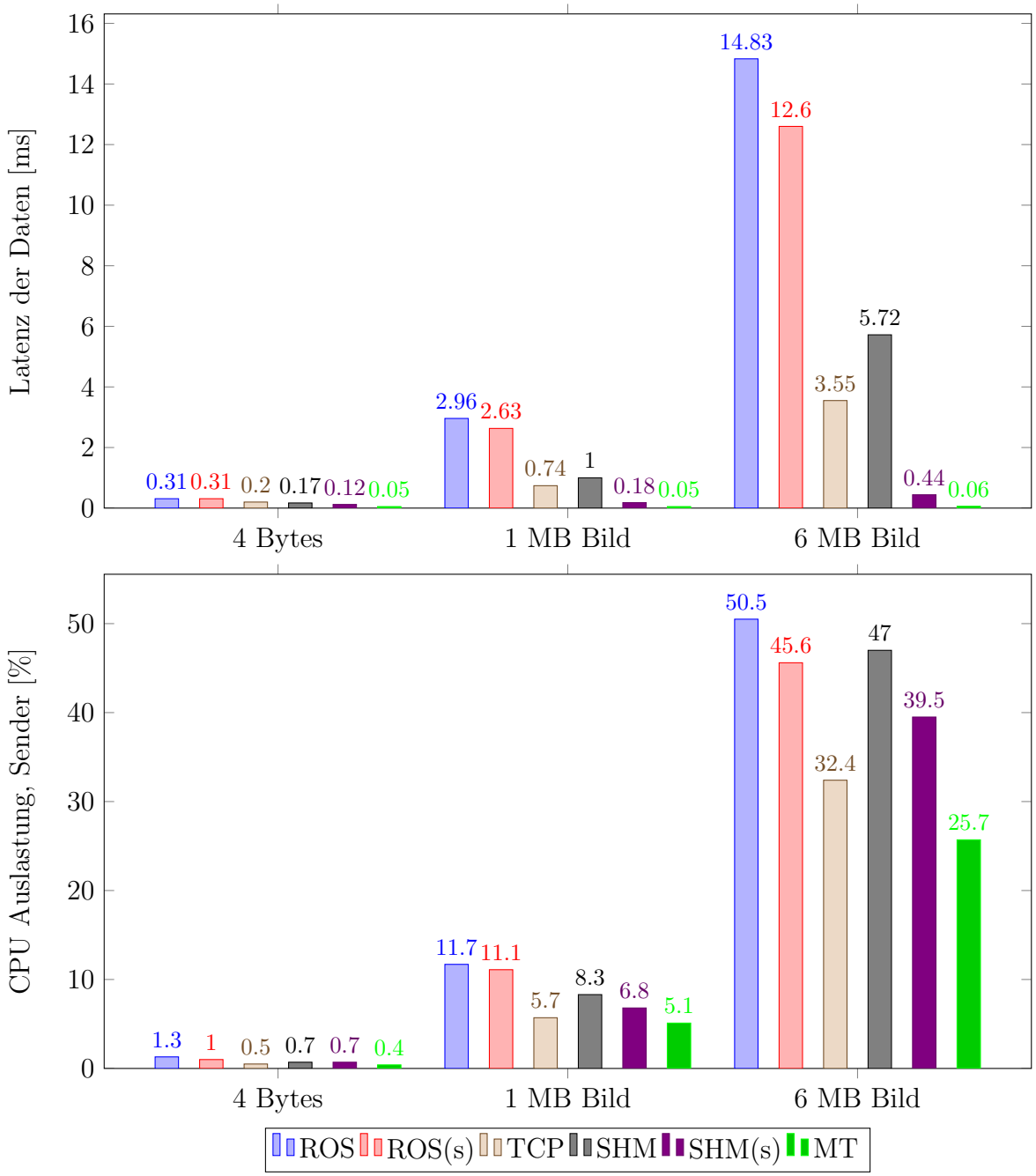


Abbildung 4.50.: Die Abbildungen zeigen Latenz und Systemauslastung auf Senderseite bei Übertragung verschieden großer Nutzdaten. Weitere Beschreibungen befinden sich bei Abbildung 4.51, die sich ebenfalls auf diesen Test bezieht.

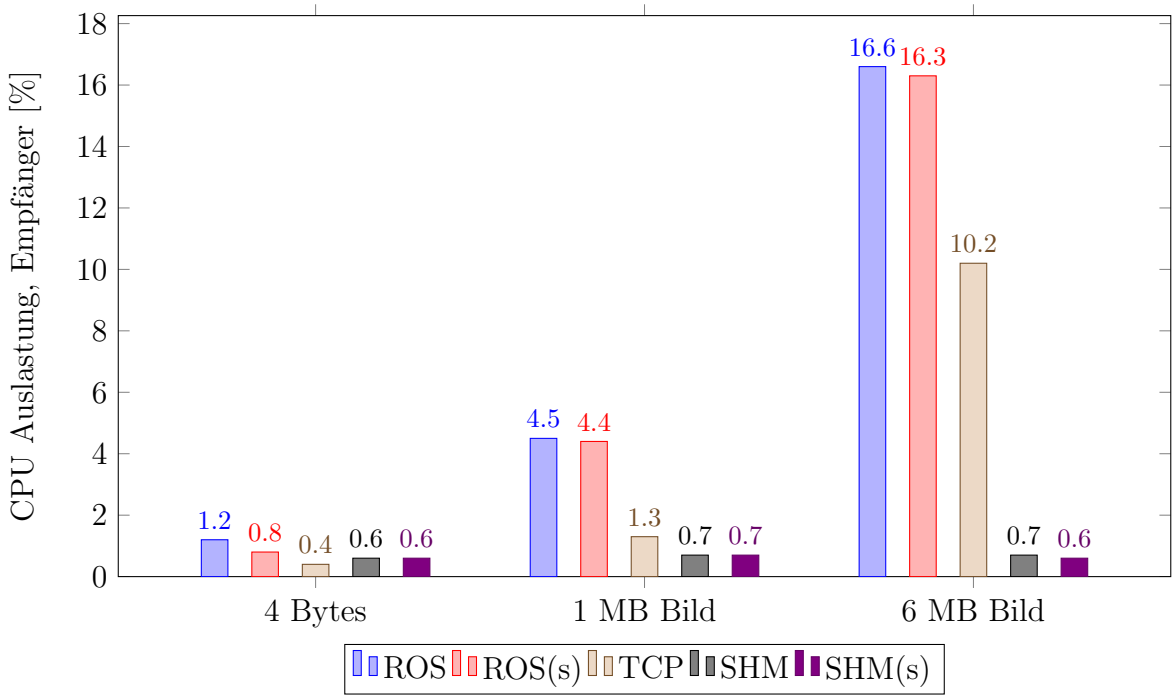


Abbildung 4.51.: Die Abbildung zeigt ergänzend zu den Abbildungen 4.50 die Systemauslastung auf Empfängerseite bei Übertragung verschieden großer Nutzdaten. Die Tests werden auf einem PC mit Intel Core i5-3570 CPU durchgeführt und es werden 30 Dateneinheiten pro Sekunde übertragen. Die CPU-Auslastung auf dem Server beinhaltet die Grundlast zur Erzeugung der Nutzdaten, die abhängig von der Größe der Nutzdaten etwa 0,4 % / 5,2 % / 25,4 % beträgt.
ROS: GStreamer ROS Element „ROSSrc“, ohne „Downstream Buffer Allocation“ (Performance vergleichbar mit „gscam“)
ROS(s): GStreamer ROS Element „ROSSrc“
TCP: GStreamer TCP Übertragung, proprietäres GDP-Protokoll, von GStreamer vorgegeben
SHM: GStreamer Shared Memory Element „SHMSrc“, ohne „Downstream Buffer Allocation“
SHM(s):GStreamer Shared Memory Element „SHMSrc“, mit „Downstream Buffer Allocation“
MT: Intraprozesskommunikation innerhalb von GStreamer (Multi-Thread, nur für Abbildung 4.50 relevant)

mit einem, zwei und vier Empfängern untersucht, die alle in separaten Prozessen laufen.

Es werden folgende Übertragungswege getestet, soweit möglich immer mit „Downstream-Buffer-Allocation“:

- GStreamer ROS Element
- GStreamer ROS Element, Versenden als JPEG
- GStreamer TCP (vorgegeben)
- GStreamer Shared Memory Element

Die Ergebnisse sind in Abbildung 4.52 gezeigt. Es ergibt sich, dass das Verteilen der Daten über Shared Memory den anderen Verfahren hinsichtlich CPU-Auslastung und Latenz überlegen ist. Lediglich wenn weniger als zwei Empfänger angemeldet sind, erzeugt das Versenden als JPEG-komprimierte Daten eine geringere CPU-Last. Als nachteilig stellt sich der Umstand heraus, dass der komprimierte Datenstrom von ROS für jeden Empfänger separat dekodiert zu werden scheint. Somit verhält sich die CPU-Auslastung proportional zur Anzahl der Empfänger. Da auf dem verwendeten Computersystem mit einer Vierkern-CPU ausreichend Rechenleistung zur Verfügung steht, wirkt sich dieser Umstand bei der komprimierten Übertragung nicht auf die Latenz aus. Werden mehr Empfänger gestartet als freie CPU-Kerne zur Verfügung stehen, kommt es auch hier zu einer erhöhten Latenz. Ein weiterer Test zeigt den Anstieg der Latenz von 20 ms auf 26 ms, wenn sechs Empfänger gestartet werden (ohne Abbildung).

Alle in diesem Abschnitt durchgeführten Tests zeigen, dass die im Rahmen dieser Arbeit gewählte Form der Implementierung der Bildverarbeitungsalgorithmen in GStreamer sich durch eine sehr hohe Effizienz auszeichnet. Innerhalb von GStreamer können die Daten mit minimalem Overhead von einem Element zum nächsten übertragen werden. Auch zur Interprozesskommunikation und Netzwerkkommunikation bietet GStreamer hoch effiziente TCP-Elemente, die die Mechanismen anderer Frameworks übertreffen. Als weiterer Ansatz zur Interprozesskommunikation wird im Rahmen dieser Arbeit eine Methode basierend auf Shared Memory entwickelt, die in Kombination mit der „Downstream-Buffer-Allocation“ von GStreamer nochmals eine deutlich höhere Effizienz erzielt.

Der mögliche Realisierungsweg, alle im Rahmen dieser Arbeit entwickelten Funktionen unter ROS zu implementieren, würde deutliche Nachteile bezüglich der Effizienz hervorbringen, insbesondere wenn hoch aufgelöste Bilddaten zwischen verschiedenen Software-Komponenten ausgetauscht werden sollen. Die Tests zum Vergleich der verschiedenen Mechanismen zur Datenübertragung wurden auf einem leistungsstarken PC mit Intel Core i5 Prozessor ausgeführt. Kommen wie auf intelligenten Kamerasystemen oder mobilen Robotern entsprechend leistungsschwächere Systeme zum Einsatz, ergeben sich deutlich größere Verzögerungen. In einem exemplarisch durchgeführten Test beträgt die Verzögerung beim Übertragen eines 6 MB großen Bildes über das ROS-Framework auf einem Intel Atom N270 132 ms, was als nicht akzeptable Verzögerung zu betrachten ist.

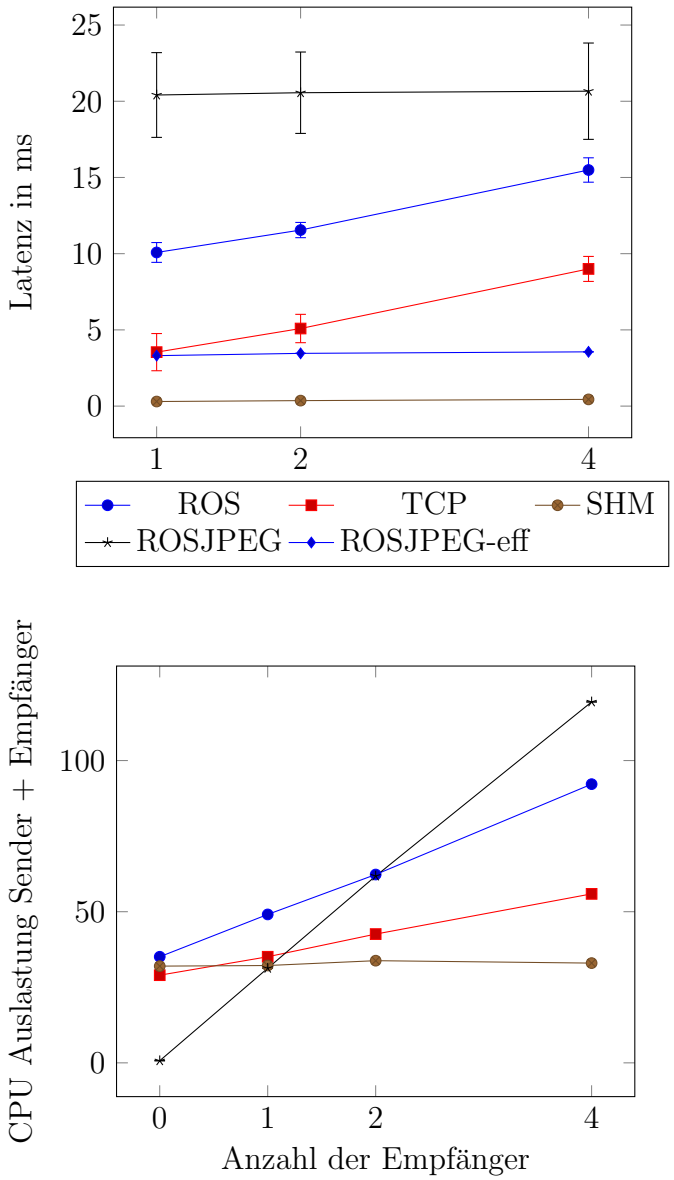


Abbildung 4.52.: Diese Grafik zeigt die Auswirkung der Anzahl der Empfänger auf die Latenz und die gesamte CPU-Auslastung. Die Standardabweichung wird markiert, außer wenn sie zu gering ist, als dass sie darstellbar wäre. Da die gemessene Latenz bei der Übertragung der JPEG-komprimierten Daten auch die Zeit zum Dekodieren beinhaltet, wird zusätzlich die effektive Latenz dargestellt (ROSJPEG-eff), die sich durch Subtraktion der Zeit zur Dekodierung ergibt. Eine CPU-Auslastung von über 100 % ergibt sich daraus, dass mehrere CPU-Kerne benutzt werden. Weitere Details zu den Abkürzungen s. Abbildung 4.51.

Auch in ROS gibt es mittlerweile Ansätze, diese Probleme zu umgehen. Durch sogenannte „Nodelets“ wird ebenfalls die Möglichkeit geschaffen, zwischen verschiedenen Softwarekomponenten Daten ohne unnötige Kopiervorgänge auszutauschen. Diese „Nodelets“ können aber nur innerhalb eines „Nodes“ gestartet werden und Daten austauschen. Soll zwischen verschiedenen „Nodes“ kommuniziert werden, fällt wieder der hier gemessene Overhead an.

Als Ergebnis der Untersuchung zur Interaktion mit ROS ergibt sich, dass es als sinnvoll erscheint, Videodaten mit hoher Bandbreite innerhalb von GStreamer zu verarbeiten und die Ergebnisse der Bildanalyse, die ein geringeres Datenvolumen aufweisen, in ROS zu veröffentlichen. Auf diese Art können die Vorteile der beiden Frameworks vereint werden:

- GStreamer bietet eine Vielzahl an Bildverarbeitungsfunktionen und zeichnet sich durch eine sehr hohe Effizienz aus.
- ROS stellt eine Vielzahl von Funktionen zur Robotersteuerung bereit. Zudem bietet ROS eine hohe Flexibilität hinsichtlich der Ausführung von „Nodes“ auf verteilten Systemen.

4.6. OpenCL beschleunigte Bildverarbeitung

In diesem Abschnitt wird dargelegt, dass in der entwickelten Architektur neben gewöhnlichen Prozessoren auch beliebige OpenCL-fähige Recheneinheiten verwendet werden können und somit Flexibilität hinsichtlich der verwendeten Hardwarearchitektur besteht. Die Einbindung der Recheneinheiten von Grafikkarten (GPU) in die bestehende Softwarearchitektur ist ohne weitgehende Umstrukturierungen möglich. Diese Integration von GPU-Verarbeitung eröffnet zum einen die Möglichkeit zur Nutzung zusätzlicher Systeme mit leistungsfähiger Grafikhardware in Szenarien vergleichbar mit den in Abschnitt 4.3 beschriebenen. Zum anderen könnten spätere Generationen von intelligenten Kameras auf entsprechenden Architekturen basieren. Beispielsweise verfügt die angekündigte intelligente Kamera Ximea CURRERA G über einen AMD-Prozessor mit integrierter GPU (vgl. Abschnitt 2.3.4). Diese Architekturen versprechen für viele Anwendungen ein günstiges Verhältnis von Rechenleistung zu Energieverbrauch.

Im Gegensatz zu den in den Abschnitten 4.2 und 4.3 beschriebenen Tests, bei denen komplette Anwendungen implementiert werden, soll bei der Behandlung dieses Themenbereichs nur die Realisierbarkeit einzelner Funktionen getestet werden. Im Einzelnen handelt es sich um folgende Funktionen:

- Sobel-Filter
- Laplace-Filter
- Korrektur der Linsenverzeichnung

- Erzeugen eines Panoramabildes aus vier Einzelbildern

Die Implementierung dieser Funktionen soll auf OpenCL basieren [G+08]. Hierbei handelt es sich um einen einheitlichen Standard zur Programmierung von verschiedenen Hardwarearchitekturen (GPU, Signalprozessoren, CPU). Auch für Mobiltelefone oder Tablet-PCs sind mittlerweile Grafikprozessoren mit OpenCL Unterstützung angekündigt [mal12]. Die Syntax ist an die Programmiersprache C angelehnt. Im Gegensatz zu klassischen Programmiersprachen werden hier parallel ausführbare Arbeitsschritte direkt vom Programmierer vorgegeben. Dies erfolgt in der Regel über die Definition von Arrays, bei denen für jedes Datenfeld eine Funktion definiert ist, die in Abhängigkeit von der Position des Datenfeldes im Array ein Feld in einem Ausgabearray berechnet. Zur Laufzeit werden die Funktionen dann auf die vorhandenen Rechenwerke verteilt. Bezüglich GPUs ergeben sich so zwei entscheidende Vorteile:

- große Anzahl an Recheneinheiten pro GPU (z.B. 1536 bei NVIDIA GeForce GTX 680, vgl. [nvc12])
- schnelle Anbindung an den Hauptspeicher der Grafikkarte (z.B. 192.2 GB/s Speicherbandbreite; Quelle s.o.)

Als Nachteil dieser Art der Implementierung von einzelnen Bildverarbeitungsfunktionen muss jedoch festgehalten werden, dass der Datentransfer vom Hauptspeicher des Computersystems zum Speicher der GPU eine Verzögerung verursacht. Im Rahmen dieser Arbeit erfolgt die Implementierung der OpenCL-basierten Algorithmen ebenfalls innerhalb von GStreamer-Elementen, die zu einer Pipeline verknüpft werden.

Bei den Tests wird eine GPU mit einbezogen, die von ihren Recheneinheiten her der Ximea CURRERA G ähnelt. Durch diese Tests soll abgeschätzt werden, welche Performance von einer aktuellen intelligenten Kamera (geplante Markteinführung 2013) erwartet werden kann. Folgende Tabelle stellt die Recheneinheiten der Ximea CURRERA G denen der im Test verwendeten AMD Radeon 5450 gegenüber:

Hardware	Recheneinheiten	Taktrate GPU	Speicher
AMD Radeon 5450	80	400 MHz	DDR3-1333
Ximea CURRERA G (AMD G-T56N)	80	500 MHz	DDR3-1333

Diese Angaben beruhen auf der GUI des Treibers (AMD Radeon 5450) und auf dem Produktdatenblatt des Herstellers Ximea.

4.6.1. OpenCL beschleunigte Algorithmen auf Grauwertbildern

Bei der Implementierung der Filter, die auf Grauwertbildern arbeiten, wird der Umstand ausgenutzt, dass das OpenCL-System den Programmcode, der auf der GPU zur Ausführung kommt, zur Laufzeit lädt. Das implementierte GStreamer-Element „gstcl“ stellt nur

die Funktion zum Up- und Download sowie zum Laden und Ausführen eines beliebigen Kernels bereit. Abhängig von dem geladenen Kernel können verschiedene Bildverarbeitungsfunktionen ausgeführt werden. Im Rahmen dieser Arbeit werden Kernel zur Sobel- und zur Laplace-Filterung implementiert und getestet. Die Einschränkung hierbei ist, dass der Kernel als Ein- und Ausgabewerte ein Array von 8-bit-Werten hat.

4.6.2. OpenCL beschleunigte Algorithmen auf Farbbildern

In einem weiteren GStreamer-Element wird analog zu dem Element zur Verarbeitung von Grauwertbildern ein Element zur Verarbeitung von Farbwertbildern implementiert. Hierbei wird ein spezieller Datentyp von OpenCL genutzt, der für Bilddaten vorgesehen ist. Seitens der Grafikhardware und des Treibers erfolgen einige Optimierungen hinsichtlich der Organisation der Daten im GPU-Speicher, um bei typischen Zugriffsmustern eine möglichst hohe Performance zu erzielen. Des Weiteren stehen dem Programmierer zusätzliche Funktionen zur Verfügung, mit denen Bilddaten interpoliert werden können. In den Leistungsmessungen wird dieses Element nicht getestet. Stattdessen wird ein Element zur Rektifikation von Farbbildern implementiert und evaluiert, das ähnlich arbeitet, jedoch noch zusätzliche Eingabedaten benötigt.

4.6.3. OpenCL beschleunigte Rektifikation

In [Qur11] wird ein OpenCL-basierter Algorithmus zur Rektifikation vorgestellt. Dort wird eine Verarbeitungszeit von 47 ms für ein Farbbild mit einer Auflösung von 2592 x 1944 Pixeln ermittelt (ohne Up- und Download, Testhardware AMD Radeon 5800). Da eine direkte Ausführung des dort veröffentlichten Algorithmus auf der hier verwendeten Hardware nicht möglich ist, werden nur einige Konzepte dieses Algorithmus übernommen und andere Teile neu implementiert.

Der hier genutzte Algorithmus hat folgenden Ablauf: Zu Beginn wird eine Lookup-Tabelle erzeugt, die steuert, welcher Pixel vom Eingabebild auf welchen Pixel im Ausgabebild abgebildet wird. Für jeden Bildpunkt im auszugebenden Bild werden die Koordinaten im Eingangsbild in Subpixel-Genauigkeit dargestellt. Diese Lookup-Tabelle wird in den Speicher der GPU geladen. Zur Laufzeit wird dann für jeden Pixel der entsprechende RGB-Wert durch Interpolation bestimmt. Hierfür wird eine in OpenCL vorgegebene Funktion verwendet. Da diese Funktion nur für RGB-Bilder in dem zuvor erwähnten Format implementiert ist, müssen die Bilddaten gegebenenfalls vorher umgewandelt werden. Hierzu wird im Rahmen dieser Arbeit auch eine OpenCL-Funktion implementiert, die die Bilddaten vom YUV- in das RGB-Format umwandelt. Die Vorteile hierbei bestehen in der höheren Verarbeitungsgeschwindigkeit, der Entlastung der CPU sowie darin, dass weniger Daten zur Grafikkarte übertragen werden müssen. Das Ausgabeformat des Elements ist unabhängig vom Eingabeformat stets RGB.

4.6.4. Ergebnisse der OpenCL-beschleunigten Bildverarbeitung

Die Leistungsmessungen werden auf unterschiedlicher Testhardware ausgeführt, darunter auch auf der Grafikkarte von AMD, die ähnlich leistungsfähig wie die GPU der intelligenten Kamera Ximea CURRERA G ist. Es wird zum Vergleich ebenfalls die Verarbeitungsleistung auf zwei verschiedenen CPUs mit ausgewertet. Im Einzelnen handelt es sich um folgende Systeme:

- Intel Core i5 3570 (Beispiel für aktuelle CPU)
- Intel Pentium 4 mit 2,4 GHz (Beispiel für ältere CPU)
- AMD Radeon 5450 (Leistung entspricht in etwa der intelligenten Kamera Ximea CURRERA G)
- Nvidia Quadro NVS 295 (entspricht NVidia 8400, Einsteigergrafikkarte von etwa 2007)
- Nvidia Quadro NVS 600 (entspricht NVidia GT 430, untere Mittelklasse von etwa 2010)
- Nvidia Quadro NVS 2000 (entspricht NVidia GTS 450, Mittelklasse von etwa 2010)[5]
- Nvidia GTX 670 (untere Oberklasse, von etwa 2012)

In den Tests werden folgende Auflösungsstufen untersucht:

- 640 x 480 VGA
- 1384 x 1032 (Auflösung der Basler eXcite, minimal verringert)[6]
- 2592 x 1944 (zum Vergleich mit [Qur11])

Bei den Testergebnissen soll zunächst die Korrektur der Abbildungsverzeichnung bei einer Auflösung von 1384 x 1032 Pixeln genauer analysiert werden. Das Ergebnis für dieses Beispiel ist grafisch in Abbildung 4.53 dargestellt. Es ist ersichtlich, dass die AMD 5450 zwar langsamer als die Mittelklassegrafikkarten ist, jedoch eine Bildwiederholrate von etwa 30 fps erreichen würde. Da die Shader der Ximea CURRERA G mit 500 MHz noch etwas schneller getaktet sind, könnte die bei der CURRERA G erreichbare Bildwiederholrate noch höher ausfallen. Eine Verarbeitung in Echtzeit und ein sinnvoller Einsatz in diesem Szenario sind somit möglich. Dieses stellt im Vergleich zur Basler eXcite (Verarbeitungszeit über 600 ms, etwa 1,5 fps, vgl. 4.1.2) eine entscheidende Verbesserung dar. Es zeigt sich jedoch auch, dass die AMD-GPU aktuellen Mittelklasse GPUs deutlich unterlegen ist. Auch eine aktuelle CPU der oberen Mittelklasse arbeitet schneller als die AMD GPU. Des Weiteren lässt der Test erkennen, dass die GPU-basierte Verarbeitung deutliche Vorteile gegenüber einer Verarbeitung auf einer CPU haben kann. Die drei leistungsstärkeren GPUs von NVidia arbeiten deutlich schneller als die Intel CPU. Somit kann gezeigt werden, dass durch den Einsatz von GPUs eine deutliche Leistungssteigerung erzielt werden

[5] Vergleichbar leistungsfähige Grafikkarten (GTS 450) für den Heimbereich sind für ca. 90 Euro erhältlich. Die ebenfalls getestete Quadro NVS 295 ist vom Chipsatz her mit der NVidia 8400 identisch, die eine der leistungsschwächsten aktuell verfügbaren dedizierten Grafikkarten ist (ca. 30 Euro). Stand August 2012. Quelle: wikipedia.de und nvidia.de, Preise ebay.de

[6] Die Auflösung wird aus technischen Gründen minimal verringert, damit jede Seite ein Vielfaches von 8 ist.

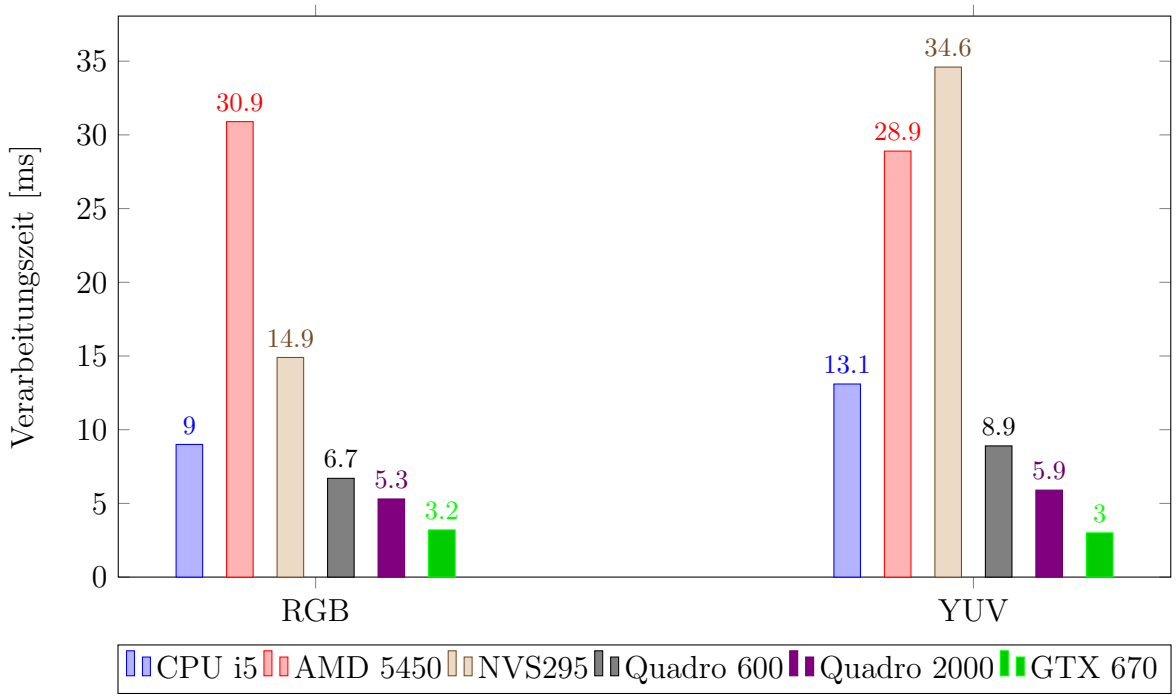


Abbildung 4.53.: Diese Grafik zeigt die Verarbeitungszeit bei Korrektur der Abbildungsverzeichnung. Der Messwert für die CPU-basierte Verarbeitung auf dem Pentium 4 mit 2,4 GHz ist nicht mit abgebildet (RGB 169 ms, YUV 132 ms). Der Up- und Download ist in der Zeit inbegriffen. Bei YUV-Eingangsdaten ist ebenfalls die Konvertierung in der Zeit enthalten.

Algorithmus	Auflösung	CPU i5	P4	AMD5450	NVS295	Q600	Q2000	GTX670
NOOP	640 x 480	n.M.	n.M.	3,3	0,5	0,4	0,3	0,2
NOOP	1384 x 1032	n.M.	n.M.	7,0	1,6	1,3	1,2	0,8
NOOP	2592 x 1944	n.M.	n.M.	17,4	5,1	3,6	3,5	2,6
Sobel	640 x 480	1,6	6,7	7,1	10,9	1,3	0,7	0,4
Sobel	1384 x 1032	7,7	32,1	23,3	51,0	4,5	2,6	1,1
Sobel	2592 x 1944	27,8	112	75,5	174	14,5	8,6	3,4
Laplace	640 x 480	1,9	12,1	6,8	5,9	0,7	0,4	0,3
Laplace	1384 x 1032	9,1	55,8	22,5	29,2	2,8	2,8	1,0
Laplace	2592 x 1944	31,7	196	73,1	96,9	9,4	5,0	2,9
Rekt. RGB	640 x 480	1,7	37,2	9,5	3,4	1,6	1,2	0,8
Rekt. RGB	1384 x 1032	9,0	169	30,9	14,9	6,7	5,3	3,2
Rekt. RGB	2592 x 1944	31,0	230	90,5	A.	23,2	19,0	10,9
Rekt. YUV	640 x 480	2,7	47,7	9,7	7,7	2,1	1,4	0,8
Rekt. YUV	1384 x 1032	13,1	230	28,9	34,6	8,9	5,9	3,0
Rekt. YUV	2592 x 1944	45,5	706	88,5	A.	31,4	20,5	10,0
Panorama	4x FullHD	n.M.	n.M.	123	A.	33,4	21,9	8,9

Abbildung 4.54.: Diese Tabelle zeigt die Verarbeitungszeit der verschiedenen GPUs und CPUs bei verschiedenen Algorithmen in ms. Es wird ebenfalls getestet, wie lange die Verarbeitung mit einem Kernel dauert, der keine Operation enthält (NOOP). Hiermit soll die Zeit zum Up- und Download bestimmt werden. Einige Messungen konnten nicht ausgeführt werden (gekennzeichnet mit „n.M.“), weil keine geeignete Implementierung bereitsteht (z.B. Panorama-Stitching auf CPU). Operationen, die aufgrund von zu geringem GPU-Speicher abgebrochen sind, sind mit „A.“ gekennzeichnet.

kann. Ein interessanter Effekt ist bei der Nvidia GTX 670 zu verzeichnen: Wenn neben der Rektifikation noch die Konvertierung des Farbraumes auf der GPU ausgeführt wird, sinkt die gesamte Verarbeitungszeit sogar von 3,2 ms auf 3,0 ms. Dieses liegt daran, dass beim Upload von YUV-Bilddaten nur die Hälfte an Daten vom System-Speicher zum GPU-Speicher übertragen werden muss. Aufgrund der schnellen Ausführung der eigentlichen Bildverarbeitungsalgorithmen ist hier der Anteil der Zeit zur Datenübertragung deutlich höher. Dieses kann auch anhand der Tabelle 4.54 nachvollzogen werden, die nachfolgend erklärt wird.

In dieser Tabelle werden alle Testergebnisse dargestellt, einschließlich des in Abbildung 4.53 dargestellten Tests. In der Tabelle ist auch der OpenCL-basierte Algorithmus aufgeführt, der für die Erzeugung von Panoramas aus vier FullHD Einzelbildern implementiert ist (vgl. Abschnitt 4.7). Es zeigt sich, dass bei den meisten Tests die Verarbeitung auf der NVidia GTX 670 ca. dreimal so schnell ist wie auf einer Intel Core i5 CPU. Auf der CPU könnte jedoch durch Parallelisierung die Verarbeitungsleistung noch erhöht werden. Durch die Messung der Zeit zum Up- und Download kann deutlich gemacht werden, dass gerade bei den leistungsfähigen GPUs und den weniger rechenintensiven Algorithmen die Übertragungszeit den größten Anteil an der Gesamt-Verarbeitungszeit einnimmt.

Bezüglich der Implementierungen, die im Rahmen dieser Arbeit erfolgen, ergeben sich noch einige Verbesserungsmöglichkeiten. In jedem Element, das die GPU benutzt, erfolgt sowohl der Upload als auch der Download der Daten zur Grafikkarte. Wenn nun mehrere solcher Elemente zu einer Pipeline verknüpft werden, würden somit überflüssige Operationen ausgeführt werden. Dieses könnte dadurch behoben werden, dass zwischen solchen Elementen nur ein Zeiger auf die Datenbereiche weitergegeben wird, wie dies auch bei zwei klassischen Elementen in der Regel erfolgt. Hierzu müsste neben der Definition von Datentypen noch eine zentrale Instanz implementiert werden, um die Gültigkeit der Datenbereiche und somit die Verwendbarkeit in verschiedenen Elementen sicherzustellen. Des Weiteren wäre es von Vorteil, wenn erzeugte Bilddaten direkt ausgegeben werden könnten, ohne vorher in den Hauptspeicher des PCs zurück kopiert zu werden. Dieser Mechanismus ist prinzipiell in OpenCL implementiert. In OpenCL erzeugte Speicherbereiche könnten direkt in OpenGL Speicherbereiche umgewandelt werden. Hierauf basierend könnte dann die Anzeige erfolgen. Die Verwendung von bestimmten Spezialfunktionen von GPUs, z.B. das Decodieren von Videodatenströmen, ist ebenfalls aktuell in Verbindung mit den OpenCL basierten Elementen nur nutzbar, wenn die Daten zwischendurch in den Hauptspeicher zurückgeschrieben werden.

Im Hinblick auf das im Rahmen dieser Arbeit entwickelte Gesamtkonzept kann belegt werden, dass seitens der Softwarearchitektur die Möglichkeit gegeben ist, einzelne Algorithmen durch eine leistungsfähigere GPU-basierte Implementierung zu ersetzen. Der Aufbau einer Pipeline und sämtliche andere Funktionen können weiter genutzt werden. Auf diese Art wird die Möglichkeit geschaffen, zukünftige intelligente Kamerasysteme mit entsprechender Hardware direkt zu unterstützen.

Bezüglich der Leistungsabschätzung der Ximea CURRERA G ergibt sich, dass diese leistungsfähig genug sein müsste, um viele Algorithmen in Echtzeit auszuführen, die auf der Basler eXcite nur mit verringerter Wiederholrate nutzbar sind. Wenn jedoch auch die Auflösung erhöht wird, ist auch hier mit Einschränkungen zu rechnen. Das Grundproblem, dass eine intelligente Kamera in ihrer Leistung hinter moderner Desktop-Hardware zurückfällt, wird bei Zugrundelegung der hier erfolgten Messungen auch bei der Ximea CURRERA G bestehen.

Die verschiedenen Systeme haben sehr unterschiedliche Voraussetzungen und Einsatzbedingungen. Während beispielsweise eine NVidia Geforce GTX eine Leistungsaufnahme von über 150 Watt hat und auch entsprechende Maßnahmen zur Kühlung erforderlich sind, nimmt die kombinierte CPU/GPU der CURRERA G weniger als 20 Watt auf und wird passiv gekühlt.

4.7. Intelligentes omnidirektionales Kamerasystem

Im Folgenden wird ein intelligentes Kamerasystem vorgestellt, das im Rahmen einer Kooperation des Arbeitsbereichs TAMS mit dem Institut Robotechn aus Kunshan (China) entwickelt wird. Bei der Entwicklung dieses Systems werden viele Erkenntnisse aus den im Rahmen dieser Arbeit erfolgten Untersuchungen mit einbezogen. Bei der Implementierung steht die Systemarchitektur im Mittelpunkt. An diesem praktischen Beispiel soll gezeigt werden, dass mit den hier entwickelten Methoden und Softwarekomponenten auch ein weiteres komplexes intelligentes Kamerasystem aufgebaut werden kann. Nicht Bestandteil des Systementwurfs ist die hardwareseitige Implementierung als Eingebettetes System. Stattdessen werden zunächst Standardkomponenten aus Desktop-PCs verwendet. Die Entscheidung, einen dedizierten PC im Desktop-Format zu verwenden, erfolgt aufgrund der Testergebnisse, die bei der hier vorliegenden Arbeit erarbeitet wurden.

Eine Besonderheit des Systems ist, dass es über mehrere Kameras verfügt. Vier Kameras dienen dazu, ein direkt in dem System berechnetes 360° Panorama der Umgebung zu erzeugen. Für dieses Panorama können die implementierten Algorithmen beispielsweise zur Objekt- und Personenerkennung angewendet werden. Eine zusätzliche Kamera generiert mit Hilfe einer Schwenk-Neige-Einheit (PTU) und eines Zoom-Objektivs Detailaufnahmen einer *Region of Interest*. Diese Kamera wird im Folgenden auch PTZ-Kamera genannt. Die Regionen können über eine GUI durch Definieren eines Rechtecks bestimmt werden. Das System berechnet dann automatisch die notwendige PTU-Position und die Zoom-Einstellung, um den Bereich möglichst komplett abzudecken. Die Detailaufnahme wird direkt in das Ausgabebild eingeblendet. Die Steuerung der PTU und der Zoom- und Fokus-Einstellung des Objektivs auf der PTZ-Kamera erfolgt ebenfalls automatisch von dem System.

Das System ist sowohl für den Einsatz auf mobilen Robotern, insbesondere für Tele-Operation, als auch für den Surveillance-Bereich konzipiert. Für viele Anwendungen bietet eine möglichst weitwinklige Aufnahme signifikante Vorteile. Im Surveillance-Bereich beispielsweise ermöglicht ein solches System, sämtliche signifikanten Bildinformationen in einem Bilddatenstrom darzustellen, der leichter zu überwachen ist als mehrere unabhängige Videoströme. Im Bereich der Robotik ermöglichen Rundumsichtsysteme eine Lokalisierung in der Umgebung.

4.7.1. Hardwareaufbau

Die Hardware des Systems besteht aus mehreren Komponenten, die über einen Gigabit Ethernet Switch verbunden sind (siehe Abbildung 4.55). Ein leistungsstarker Standard-PC mit einer dedizierten Grafikkarte dient zur Bildverarbeitung, zur Anzeige der Videodaten und zur Steuerung des gesamten Systems. Zur Aufnahme des Panoramabildes ist ein Verbund aus vier Kameras des Typs Sanyo HD2100P vorhanden. Diese liefern

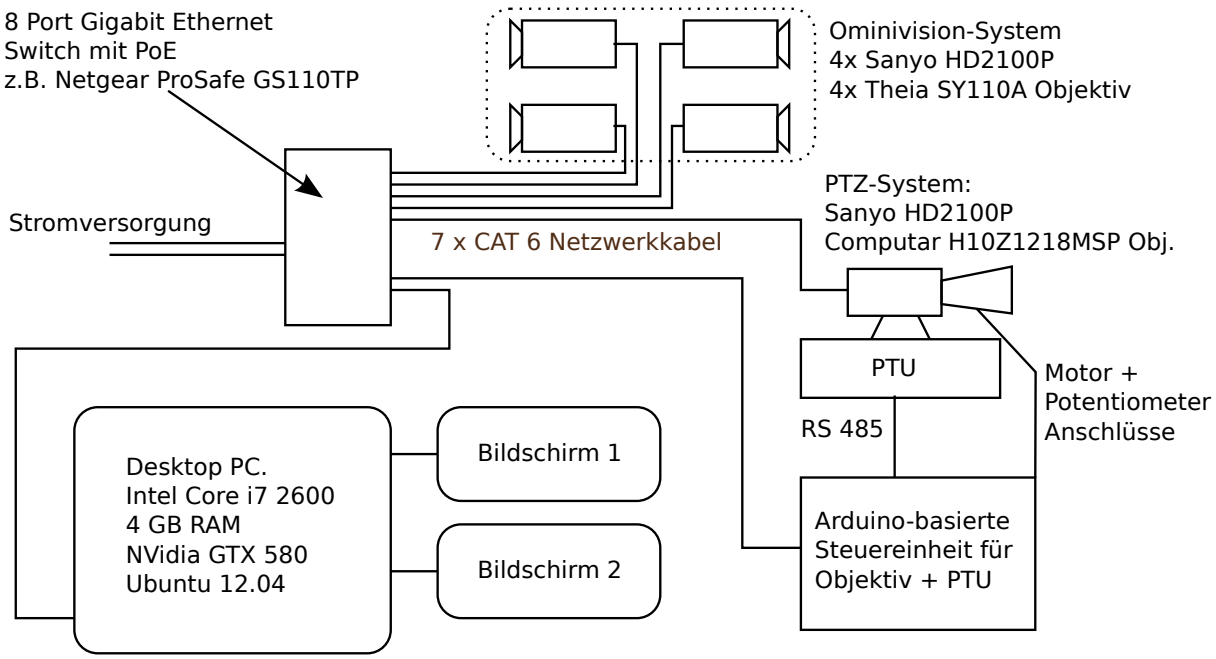


Abbildung 4.55.: Die Hardware des intelligenten omnidirektionalen Kamerasystems.

einen h264-kodierten Bilddatenstrom, so dass die Datenübertragungsrate des Gigabit-Netzwerkes ausreichend ist.

Ein weitere Kamera des gleichen Typs ist auf einer PTU montiert und mit einem Zoom-Objektiv ausgestattet. Dieses System liefert Detailaufnahmen von einer ROI. Zur Steuerung der PTZ-Kamera wurde von J. Liebrecht (Universität Hamburg) im Rahmen seiner Bachelorarbeit [Lie12] ein System entwickelt, das auf einem Arduino Microcontroller basiert und folgende Funktionen bereitstellt:

- Anbindung über Ethernet
- Ansteuerung der Motoren für Zoom und Fokus
- Auswertung der Potentiometer zum Auslesen der aktuellen Zoom- und Fokus-Stellung
- automatisches Anfahren von Zielpositionen
- Ansteuerung der PTU über RS485 Schnittstelle (Pelco-D Protokoll)
- Ethernet-zu-Seriell-Adapter zur Ansteuerung beliebiger anderer PTU-Typen mit serieller Schnittstelle

Auch wenn im Rahmen oben genannter Bachelorarbeit ein Autofokus-Algorithmus implementiert ist, wird hier der Fokus-Wert anhand einer Lookup-Tabelle eingestellt, die für jede Zoom-Einstellung einen entsprechenden Wert speichert.

Die entscheidenden Hardware-Komponenten des Systems sind in 4.56 abgebildet.

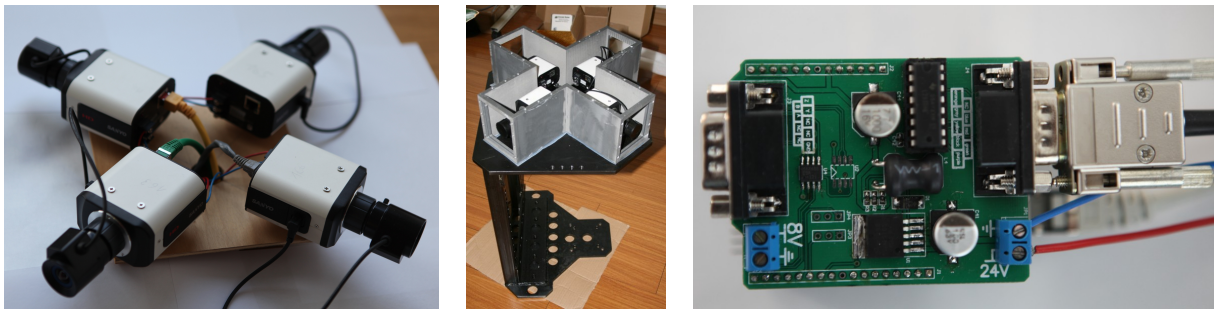


Abbildung 4.56.: Diese Bilder zeigen den Aufbau der vier Kameras (links), das Gehäuse (Mitte, entworfen von Kunshan Robotechn Intelligent Technology Co.) und das Eingebettete System zur Steuerung von PTU und Zoom (Schaltung entworfen von J. Liebrecht, Board Layout von Kunshan Robotechn Intelligent Technology Co.)

4.7.2. Softwarekonfiguration

Das Software-System des omnidirektionalen Kamerasystems entspricht in weiten Teilen dem System, dass im Rahmen dieser Arbeit entwickelt wurde. Es werden ebenfalls das GStreamer-Framework und viele der im Rahmen dieser Arbeit implementierten Funktionen zur Bildverarbeitung verwendet. Die konfigurierte Pipeline enthält die Funktionen zur Panoramaerzeugung, zur Detektion von Objekten und Personen sowie zur Visualisierung (siehe Abbildung 4.57). Die Bilddaten jeder Kamera werden zunächst dekodiert; danach werden von einem „Multisync“-Element immer vier zusammengehörige Bilder weitergeleitet. Sollten keine vier Bilder eintreffen, so werden die anderen Bilddaten verworfen. In dem folgenden Schritt erfolgt die eigentliche Erzeugung des Panoramas aus den vier Einzelbildern.

Die Verwendung einer OpenCL-basierten Funktion zur Erzeugung des Panoramas erweist sich als notwendig, da die Ausführung auf einer CPU eine deutlich zu hohe Rechenzeit verursacht, als dass bei voller Bildwiederholrate gearbeitet werden könnte. Die nicht weiter auf Videodatenströme optimierten Funktionen von Hugin [hug13] benötigen folgende Ausführungszeiten[7]:

System	Ausführungszeit
Intel Core i5 750	0.9 sec
Intel Xeon E31245	0.6 sec
Intel Atom N270	11.5 sec

Die Generierung des Panoramas erfolgt durch ein OpenCL-basiertes Element, das wie das Element zur Korrektur der Abbildungsverzeichnung mit einer Lookup-Tabelle arbeitet, die neben dem Quellpixel auch die Information enthält, welches der Eingangsbilder verwendet wird. Die Lookup-Tabelle wird für Objekte erstellt, die sich in weiter Ferne

[7]Nicht in GStreamer integriert, von Kommandozeile mit dem „time“-Befehl gestartet.

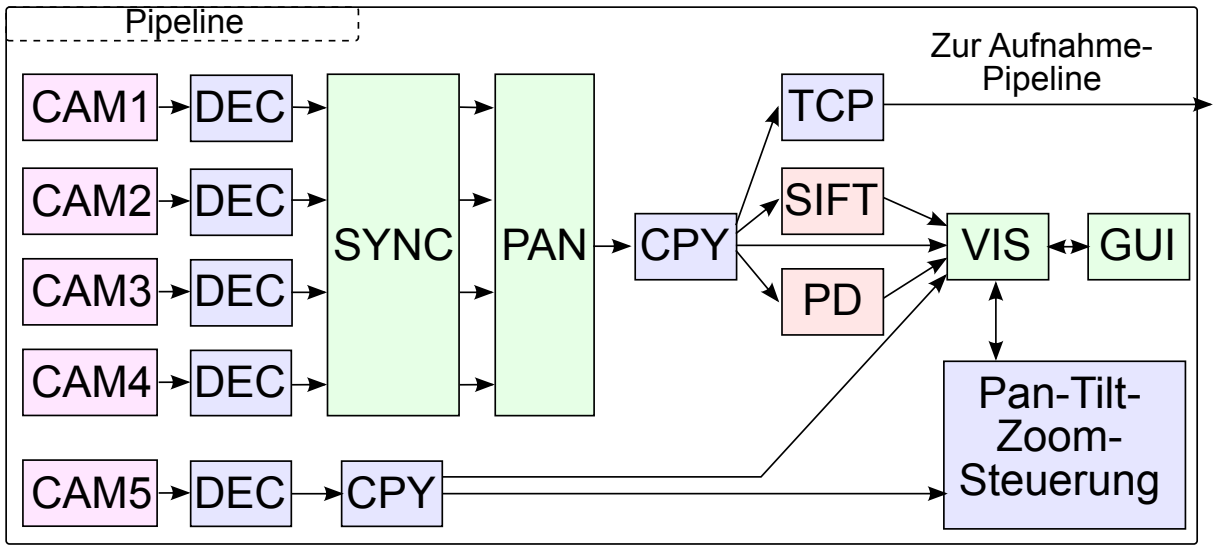


Abbildung 4.57.: Diese Abbildung zeigt die Konfiguration der GStreamer-Pipeline beim intelligenten omnidirektionalen Kamerasystem. Es werden nur Funktionsblöcke gezeigt, die in der Praxis auch oft aus mehreren GStreamer-Elementen bestehen. Die einzelnen Funktionsblöcke und Elemente:
CAMx: Zugriff auf IP-Kameras
DEC: Parsen des RTSP-Protokolls, Dekodieren von h264
SYNC: Weiterleitung von jeweils vier Bildern, gegebenenfalls verwerfen
PAN: erzeugt ein Panorama aus vier Ausgangsbildern
CPY: erzeugt Kopien der Bilder (Element Typ „Tee“)
TCP: versendet Daten via TCP
SIFT: SIFT-basierte Objekterkennung
PD: Detektion von Personen
VIS: Markierung detektierter Ergebnisse, Einblendung der PTZ-Kamera
GUI: Bereitstellung von Steuerelementen

befinden. Bei näheren Objekten weist der Übergang Fehler auf, die hier zwecks einer höheren Performance aber in Kauf genommen werden. Eine Korrektur dieser Fehler würde bedeuten, dass in den überlappenden Regionen jedes Satzes Einzelbilder eine Suche nach korrespondierenden Bildpunkten durchgeführt werden müsste.

Die Erzeugung der Lookup-Tabelle erfolgt durch halbautomatische Kalibrierung über die Open-Source Software Hugin [hug13], die zur Erstellung von Panoramen vorgesehen ist. Der Kalibrierungsvorgang für das Testsystem wurde von Dr. A. Mäder (Universität Hamburg) durchgeführt. Die Ausgabedateien von Hugin müssen in ein anderes Format umgewandelt werden. Hier wird ein Tool eingesetzt, das von G. Cheng (Universität Hamburg) entwickelt ist.

Die Ablauf bei der OpenCL-basierten Verarbeitung ist folgender:

- Initialisierung (bei Pipelinestart)
  - GPU Speicher allozieren
    - 4 YUV Eingangsbilder
    - 4 RGB Bilder (Zwischenschritt)
    - 1 RGB Ausgabebild
    - 1 Lookup-Tabelle
  - Lookup-Tabelle hochladen
- Verarbeitung (pro Frame)
  - 4 YUV-Bilder zu GPU-Speicher übertragen
  - 4 x Farbraumkonvertierung YUV zu RGB
  - Panoramaerzeugung mit Subpixelgenauigkeit
  - Panorama-Ausgabebild herunterladen

Aus den durchgeführten Tests zur Leistungsmessung ergibt sich, dass mit der verwendeten Grafikkarte NVidia GTX 580 die Erzeugung des Panoramas nur 7 ms benötigt.[8] Es kann damit gezeigt werden, dass die Erzeugung eines Panoramas bei voller Bildwiederholrate auf einer leistungsfähigen GPU möglich ist.

Das erzeugte Panorama wird über ein Element des Typs „Tee“ an vier weitere Elemente geleitet. Der Datenstrom wird über ein TCP-Element für weitere Pipelines verfügbar gemacht. So lässt sich bei Aufnahme des Videodatenstromes die Aufnahme-Pipeline unabhängig von der Haupt-Pipeline starten und stoppen. Das Panoramabild wird an die beiden Element-Gruppen zur Objektdetektion und zur Personenlokalisation geleitet. Letztere funktioniert wie die Gesichtslokalisation, jedoch wird ein andere sogenannte Kaskade zur Detektion von Oberkörpern verwendet. Die Ergebnisse der Bildverarbeitung sowie eine weitere Kopie des Panoramabildes werden dann zu einem Element des Typs „Multivis“ geleitet, das hier folgende Aufgaben hat:

- Markieren von erkannten Objekten

[8] Spätere Tests auf einer eigentlich schnelleren NVidia GTX 670 zeigen einen Wert von 9 ms, die Ursache hierfür liegt vermutlich in einer höheren Auflösung des Ausgabebildes bei einer neuen Konfiguration.

Abbildung 4.58.: Dieses Bilder zeigen das Ergebnis der Panoramaberechnung und haben in dieser Konfiguration eine Auflösung von etwa 2500 x 500 Pixeln. Es können auch beliebig höhere Ausgabeauflösungen konfiguriert werden. Das Kamerasystem ist beim unteren Bild auf dem Mast eines Schiffes montiert. Bei dem unteren Bild wird ein Farbkorrektur-Algorithmus verwendet, der zur Zeit noch nicht in dem GPU-beschleunigten Code integriert ist. Eine bessere Kalibrierung der Kameras könnte das Ergebnis im Hinblick auf die teilweise sichtbaren Übergänge verbessern.

- Markieren von lokalisierten Personen
- Einblenden von Latenzinformationen
- Verarbeiten von Benutzereingaben
- Berechnen von Sollwerten für PTU + Zoom
- Entgegennahme von Istwerten für PTU + Zoom und basierend hierauf
- Einblendung des Bildes der PTZ-Kamera

Bilder, die durch das Panoramasystem erzeugt werden, sind in Abbildung 4.58 gezeigt.

4.8. Zwischenfazit

In diesem Kapitel sind die im Rahmen der Arbeit durchgeführten Experimente beschrieben. Die anvisierte Funktion der implementierten Softwarearchitektur kann anhand der durchgeführten Tests bestätigt werden.

Das Ziel, die Bildverarbeitung einerseits als modulare Funktionen zu realisieren, andererseits hierbei nur minimalen Overhead zu erzeugen, darf als erreicht betrachtet werden.

Dies zeigt insbesondere ein Vergleich mit dem etablierten Framework ROS, aus dem hervorgeht, dass die hier verwendeten Methoden nur einen Bruchteil des Overheads erzeugen, der bei Datenaustausch innerhalb des ROS-Frameworks entsteht.

Die Tests zur Leistungsmessung an dem Kamerasystem Basler eXcite ergeben, dass dieses in seiner Leistungsfähigkeit zwar hinter den zum Vergleich getesteten Desktop-Systemen zurückbleibt, sein Einsatz durch eine geeignete Verteilung der Teilaufgaben aber dennoch zu deutlichen Vorteilen im Hinblick auf die Latenz führen kann. So kann bei den evaluierten Szenarien zur Gesichtslokalisation die Verarbeitungszeit deutlich gesenkt und die über das Netzwerk übertragene Datenmenge verringert werden.
Bei der Objektdetektion ermöglicht die lokale Vorverarbeitung auf dem Robotersystem das Versenden eines kompakten Merkmalsvektors über WLAN, der dann von vielen Systemen in der Umgebung des Roboters ausgewertet wird. Diese Form des Cloud-Computings führt zu einer entscheidenden Leistungssteigerung.

Des Weiteren wird die Möglichkeit gezeigt, durch ein Reinforcement-Learning-basiertes System automatisch eine Verteilung der modularen Bildverarbeitungsschritte auf verfügbare Systeme vorzunehmen.

Es wird zudem ausgeführt, wie die angekündigte intelligente Kamera Ximea CURRERA G durch GPU-basierte Bildverarbeitung deutliche Leistungssteigerungen ermöglichen könnte. Sie wird auch bei vielen Anwendungen sinnvoll einsetzbar sein, bei denen die Basler eXcite eine noch zu geringe Bildwiederholrate liefert.

Basierend auf den Ergebnissen dieses Kapitels wird im folgenden Kapitel eine abschließende Bewertung im Hinblick auf die Fragestellung durchgeführt, inwiefern intelligente Kamerasysteme auf mobilen Service-Robotern nutzbringend eingesetzt werden können.

5. Zusammenfassung und Ausblick

Als Fazit der durchgeführten Arbeiten kann festgehalten werden, dass intelligente Kamerasysteme das Potential zu deutlichen Leistungssteigerungen bei mobilen Service-Robotern haben. Der Problematik der großen Datenmenge, die von aktuellen hochauflösenden Sensoren generiert wird, lässt sich durch direkte Verarbeitung auf den Kamerasystemen wirksam begegnen. Die Resultate dieser Arbeit können ebenfalls auf andere verteilte Systeme im Kontext der Bildverarbeitung in der mobilen Robotik übertragen werden. Es wird gezeigt, dass Leistungssteigerungen von vielen verschiedenen Faktoren abhängen. Neben der Rechenleistung der Systeme kommt der Bandbreite der Verbindungen und der Art der Bildverarbeitungsaufgabe eine besondere Rolle zu.

Die Versuchsergebnisse haben jedoch auch bestätigt, dass durch die Verwendung verteilter Systeme zusätzlicher Overhead entsteht und dieser nur dann ausgeglichen werden kann, wenn die dedizierten Recheneinheiten der Kamerasysteme im Vergleich zu den Recheneinheiten des verbundenen Steuerrechners des Roboters nicht deutlich schwächer sind. Trotzdem ist auch für solche Konfigurationen, bei denen die intelligente Kamera deutlich leistungsschwächer ist, eine Leistungssteigerung bei bestimmten Szenarien erreichbar, insbesondere wenn das Datenvolumen in einem frühen Schritt der Bildverarbeitung stark reduziert werden kann.

Des Weiteren wird gezeigt, dass mit Hilfe geeigneter Algorithmen die Verteilung der Bildverarbeitungsfunktionen auf verteilte Systeme auch automatisch möglich ist. Auch wenn im Rahmen des zugrunde liegenden Modells nicht alle Eigenschaften der Systeme mit einbezogen werden können, ergeben die Praxistests, dass die von dem System gewählten Konfigurationen weitgehend den manuell für sinnvoll erachteten Konfigurationen entsprechen und Vorteile gegenüber einer nicht parallelisierten Verarbeitung haben.

5.1. Resümee der Ergebnisse

5.1.1. Performancemessung der Basler eXcite

Die intelligente Kamera Basler eXcite wurde in verschiedenen Szenarien getestet, die für mobile Service-Roboter von Bedeutung sind. Als Ergebnis dieser Tests ist festzustellen, dass für die meisten Anwendungsfälle eine Verarbeitung der Bilddaten bei voller Bildwiederholrate direkt auf der Kamera nicht möglich ist. Für viele Szenarien konnten dennoch Möglichkeiten erarbeitet werden, die Basler eXcite nutzbringend einzusetzen. Insbesondere

wenn die eXcite zur Auswahl von Bildbereichen eingesetzt wird, zeigen sich entscheidende Vorteile. Auch bei der Unterteilung des Bildes in mehrere Bereiche zur anschließenden Parallelverarbeitung auf mehreren Systemen können Vorteile erzielt werden. Bei dem Einsatz von Bildverarbeitungsalgorithmen auf einer höheren Ebene (Gesichtslokalisation, SIFT-Merkmalsberechnung) muss abhängig von der Anwendung entschieden werden, ob die Verarbeitungsleistung und die dadurch entstehenden Latenzen akzeptabel sind. Eine Steigerung der Bildwiederholrate ist hier durch eine Verringerung der Auflösung möglich. Es konnte gezeigt werden, dass zumindest Software-seitig die Möglichkeit besteht, diese Anwendungen auf intelligenten Kameras auszuführen. Bei Portierung auf zukünftige Generationen intelligenter Kameras kann die im Rahmen dieser Arbeit entwickelte Software also direkt für entsprechende Anwendungen eingesetzt werden. Bei der Auswertung der Ergebnisse muss berücksichtigt werden, dass der Rechenaufwand bei einigen Funktionen konstant ist, bei anderen vom Bildinhalt abhängt. Somit sind einige Ergebnisse nur bedingt reproduzierbar und können als grobe Einschätzung aufgefasst werden.

5.1.2. Probleme der Basler eXcite

In den Tests konnten mehrere Probleme der Basler eXcite ausgemacht werden. Das Hauptproblem besteht darin, dass die Rechenleistung in Anbetracht der Auflösung und der Komplexität der Algorithmen nicht ausreichend ist. Eine weitere Einschränkung stellt die Speicherausstattung von 128 MB dar. Aufgrund des begrenzten Arbeitsspeichers kann beispielsweise bei der Objekterkennung nur die Berechnung des Merkmalsvektors, nicht jedoch der Vergleich mit einer Objektdatenbank auf der Kamera ausgeführt werden. Zusätzlich verursacht die Anbindung der Netzwerkschnittstelle eine Einschränkung. Beim Versenden der Daten über das Netzwerk ist der Prozessor der Kamera stark ausgelastet. Somit steht die Rechenleistung nicht mehr für die Bildverarbeitungsalgorithmen zur Verfügung.

Die Kombination der beschriebenen Nachteile ergibt eine besonders ungünstige Situation: Wenn nur Bildverarbeitungsalgorithmen auf einer niedrigen Ebene ausgeführt werden, resultiert daraus meist nicht eine Verringerung der Datenmenge. Genau eine solche Verringerung der Datenmenge wäre jedoch notwendig, um ein effizientes Versenden der Daten zu ermöglichen. Für viele Verarbeitungsschritte, die eine solche Verringerung der Datenmenge erzeugen würden, ist die Kamera jedoch gar nicht erst leistungsfähig genug.

5.1.3. Bedeutung der Rechenleistung des Steuerrechners

Falls wie in vielen der vorgestellten Szenarien die Aufgaben der Bildverarbeitung auf mehrere Systeme verteilt werden, muss immer auch das Verhältnis der Rechenleistungen zueinander betrachtet werden. Gerade wenn ein System eine deutlich höhere Rechenleistung aufweist, ist es ein naheliegender Ansatz, alle Funktionen auf diesem System auszuführen. Eine solche Situation liegt zum Beispiel dann vor, wenn die Basler eXcite, die aus

dem Jahr 2006 stammt und einen eingebetteten Prozessor hat, an ein modernes Desktop-System mit einem Core i5 Prozessor der dritten Generation aus 2012 angebunden ist. Bei einer solchen Konfiguration ist stets als Ergebnis zu erwarten, dass das Kamerasystem zu keiner nennenswerten Entlastung des Prozessors beitragen kann. Selbst wenn die Kamera voll ausgelastet ist, ist die hierdurch auf dem Steuerrechner eingesparte Rechenzeit nur geringfügig. Als zusätzlicher negativer Effekt ergibt sich, dass die Latenz der Bilddaten durch die länger dauernde Verarbeitung und die Netzwerkdatenübertragung erhöht wird. Abhängig von der Anwendung kann es eine bessere Lösung sein, eine konventionelle digitale Kamera direkt an den leistungsfähigen PC anzuschließen. Mit den im Rahmen dieser Arbeit entwickelten Methoden lässt sich für beliebige Anwendungen evaluieren, welcher Weg zu bevorzugen ist.

5.1.4. Ausblick

Zwar kann die im Rahmen dieser Arbeit untersuchte intelligente Kamera Basler eXcite die Anforderungen hinsichtlich einer Verarbeitung in Echtzeit nicht erfüllen, jedoch kann sich bei zukünftigen intelligenten Kamerasystemen diese Situation verändern. Durch den technischen Fortschritt sind immer leistungsfähigere Recheneinheiten möglich. Bei Zugrundelegung des Mooreschen Gesetzes (vgl. [Sch97]) kann davon ausgegangen werden, dass sich die Integrationsdichte auf Computerchips alle 18 Monate verdoppelt. Aufgrund dieses Fortschritts ist absehbar, dass es irgendwann intelligente Kameras geben wird, die für die untersuchten Anwendungen ausreichend Leistungsreserven bereitstellen. Allerdings wird - insbesondere ohne Optimierung der Algorithmen - in der Praxis nicht direkt eine Verdoppelung der Leistungsfähigkeit durch die Verdoppelung der Integrationsdichte erreicht werden können. Trotzdem soll hier eine Prognose aufgestellt werden, zu welchem Zeitpunkt welche Algorithmen mit welcher Bildwiederholrate möglich wären. Die Tabelle ist in Abbildung 5.1 aufgeführt.

Ausgehend von der gemessenen Bildwiederholrate der Basler eXcite (2006) wird errechnet, wie hoch die Bildwiederholrate auf intelligenten Kameras mit Technik aus dem Jahr 2013 wäre. Dieser errechnete Wert wird gegebenenfalls mit den Tests auf GPU-Hardware verglichen, die der Kamera Ximea CURRERA G entspricht. Die prognostizierte Bildwiederholrate wird errechnet durch:

$$F(m) = f_t \cdot 2^{\frac{m}{18}}$$

Hierbei ist f_t die gemessene Bildwiederholrate und m die Anzahl an vergangenen Monaten seit dem Vergleichszeitpunkt. Es ergibt sich, dass sich die Rechenleistung vom Jahr 2006 zum Jahr 2013 um den Faktor 25,4 erhöhen könnte. Soll der Zeitpunkt prognostiziert werden, zu dem in Zukunft ein bestimmter Algorithmus mit einer Wiederholrate f_z ausgeführt werden kann, ergibt sich dieser in Monaten zu:

$$T(f_z) = 18 \cdot ld(\frac{f_z}{f_t})$$

Algorithmus	eXcite (2006)	Prognose (2013)	CURRERA G (2013, simul.)	Prognose (20 fps)
Sobel	9,3 fps	237 fps	42,9 fps	2008
LaPlace	5,0 fps	128 fps	44,4 fps	2009
Rektifikation	1,5 fps	38 fps	32,4 fps	2012
SIFT-Vektor	0.16 fps	4,0 fps	-	2016
Gesichtslok.	0.06 fps	1,5 fps	-	2019

Abbildung 5.1.: Prognose zur Leistungssteigerung intelligenter Kamerasysteme. Die Tabelle zeigt basierend auf den Testergebnissen der Basler eXcite, welche Bildwiederholrate eine intelligente Kamera mit der Technik von 2013 gemäß dem Mooreschen Gesetz erreichen könnte (Prognose - 2013). Wo eine entsprechende Simulation unter Benutzung einer ähnlich leistungsstarken GPU erfolgt ist, sind die Ergebnisse für die Ximea CURRERA G angegeben (Kehrwert der Verarbeitungszeit in Sekunden). In der letzten Spalte ist angegeben, wann gemäß dem Mooreschen Gesetz (wiederum basierend auf den Testergebnissen der Basler eXcite) mit intelligenten Kameras zu rechnen ist (bzw. war), die bei gleichbleibender Auflösung eine Bildwiederholrate von 20 fps erreichen.

Es wird hier willkürlich eine Bildwiederholrate von 20 fps festgesetzt und der Zeitpunkt berechnet, wann entsprechende Hardware als Eingebettetes System für eine intelligente Kamera verfügbar sein könnte (bzw. hätte verfügbar sein können).

Des Weiteren ist zur Tabelle anzumerken, dass die Prognose nur als ungefähre Schätzung aufgefasst werden darf. Bei der tatsächlichen Leistungssteigerung spielen neben der Erhöhung der Integrationsdichte, wie sie vom Mooreschen Gesetz vorhergesagt wird, auch noch die Weiterentwicklung der Rechnerarchitektur und der Algorithmen eine Rolle. Im Fall der Rektifikation ergibt sich sogar tatsächlich ein Wert in Größenordnung der Prognose, bei den Filtern (Sobel und Laplace) ist die erreichte Wiederholrate deutlich geringer als die Prognose, was hier auch an dem größeren Anteil der Zeit für den Up- und Download der Bilder auf den GPU-Speicher liegen könnte. Generell kann jedoch prognostiziert werden, dass sich in den nächsten Jahren immer neue Möglichkeiten für intelligente Kamerasysteme bieten und immer mehr Algorithmen auf diesen Systemen sinnvoll nutzbar sein werden. Es ist jedoch auch weiterhin mit einer Steigerung der Auflösung zu rechnen, was wiederum höhere Anforderungen an die Rechenleistung stellt. Bezüglich der Steigerung der Auflösung sind jedoch im Hinblick auf das Bildrauschen und das tatsächliche Auflösungsvermögen der Optik physikalische Grenzen gesetzt, die eine weitere Steigerung nicht mehr sinnvoll machen.

Potential kabelloser intelligenter Kameras

Ein besonders großes Potential lässt sich bei intelligenten Kameras ausmachen, die ihre Daten per Funkdatenverbindung übertragen (vgl. [Kle10]). Solche Kameras können Bestandteil einer intelligenten Umgebung sein und selbständig Informationen extrahieren. Während bei der kabelgebundenen Datenübertragung die Datenrate dank der Verfügbarkeit von Schnittstellen wie Gigabit Ethernet, CameraLink und USB 3.0 weitgehend unkritisch ist und die Verarbeitung oft auf PC-basierten Systemen erfolgen kann, bestehen bei der Datenübertragung per Funk in Abhängigkeit von der verwendeten Technologie starke Beschränkungen.

Im Gegensatz zu der Verarbeitungsleistung einer intelligenten Kamera, die gemäß dem Mooreschen Gesetz steigen könnte, bestehen bezüglich der Datenrate einer Funkverbindung physikalische Grenzen (Shannon-Hartley-Grenze, vgl. [PSZL94]). Die Shannon-Hartley-Grenze besagt, dass bei gegebener Übertragungsstrecke, festgelegter Sendeleistung und gegebenem Frequenzbereich die maximal mögliche Datenrate einen bestimmten Wert nicht überschreiten kann. Bei gegebener Übertragungsstrecke kann nur eine Erhöhung der Sendeleistung oder eine Erhöhung der Bandbreite im physikalischen Sinne[1] zu einer höheren Datenübertragungsrate führen. Dieses ist oft aus Gründen von Reglementierungen unzulässig oder es ist aufgrund des damit verbundenen höheren Energieverbrauchs nicht erwünscht. Die aktuell verfügbaren Funktechnologien (ZigBee, Bluetooth, WLAN) liegen von ihrer Effizienz her nahe an dem theoretischen Optimum (vgl. [Kle10]). Somit sind bezüglich der Funktechnologie keine signifikanten Fortschritte zu erwarten.

Hinsichtlich der Verarbeitungsleistung von Recheneinheiten ist jedoch gemäß dem Mooreschen Gesetz in den kommenden Jahren noch mit erheblichen Fortschritten zu rechnen[2]. Somit wird sich das Verhältnis von verfügbarer Rechenleistung zur verfügbaren maximalen Datenrate dahingehend entwickeln, dass die Vorverarbeitung direkt in der Kamera zunehmend Vorteile bringen wird.

Auch bei Verwendung von kabelgebundener Datenübertragung kann eine Vorverarbeitung - zusätzlich zur Entlastung anderer Systeme - deutliche Vorteile hervorbringen. Insbesondere wenn vorhandene Datenleitungen und Netzwerktechniken genutzt werden (z.B. bestehende 100 MBit Netzwerktechnik, Powerline), sind aber auch hier Beschränkungen hinsichtlich der Datenübertragungsrate gegeben. Es kann darüber hinaus auch im Hinblick auf den Energieverbrauch sinnvoll sein, einen langsameren Standard zur Datenübertragung zu integrieren. So findet sich in vielen intelligenten IP-Kameras im Surveillance-Bereich nur eine 100 MBit Netzwerkschnittstelle, obwohl der Stand der Technik bereits seit einigen Jahren schnellere Verbindungen ermöglicht.

[1] Differenz der oberen und unteren Grenzfrequenz bei Funkdatenübertragung

[2] Inwieweit der Gültigkeit des Mooreschen Gesetzes in Zukunft physikalische Grenzen entgegenstehen, soll an dieser Stelle nicht weiter behandelt werden.

5.2. Weitere Forschungsmöglichkeiten

5.2.1. Portierung auf Android

In den letzten Jahren hat die Linux-basierte Android-Plattform im mobilen Bereich stark an Bedeutung gewonnen. In gleichem Maße sind Kameras in Mobiltelefonen deutlich besser geworden und auch mobile Prozessoren sind leistungsfähiger geworden. Daher liegt es nahe, den Einsatz von Mobiltelefonen als intelligente Kameras zu erproben. Hierzu müsste die Software auf Android portiert werden. Für die GStreamer-Bibliotheken ist dies schon teilweise erfolgt. Die im Rahmen dieser Arbeit entwickelten Elemente und Teile der Steuersoftware müssten ebenfalls portiert werden. Von der GUI der Steuersoftware aus könnten nun eine oder auch mehrere Daemon-Instanzen auf Mobiltelefonen gesteuert werden. Neben der reinen Übertragung der Bilddaten könnten so Funktionen zur Bildverarbeitung auf den Mobiltelefonen ausgeführt werden. Der Einsatzzweck der Geräte wäre sehr vielfältig. Die Geräte könnten auf mobilen Robotern betrieben werden und dort eventuell auch noch andere Steueraufgaben übernehmen. Ebenfalls denkbar ist die Verwendung als Bestandteil einer intelligenten Umgebung, die selbständig Informationen aus dem Umfeld extrahiert und verfügbar macht. Es könnte beispielsweise die Position von Objekten auf einer Arbeitsfläche überwacht werden oder auch erkannt werden, ob sich in einem Raum Personen befinden. Aufgrund des geringen Gewichts könnten solche Systeme auch auf unbemannten autonomen Fluggeräten installiert werden und dort beliebige Bildverarbeitungsalgorithmen ausführen. Beispielsweise ließe sich eine FPV-Steuerung[3] realisieren, es könnten auch relevante Bilddetails vergrößert übertragen werden.

Bei der Portierung wäre auch von Interesse, inwiefern sich OpenCL-basierte GStreamer-Elemente nutzen lassen. Grafikchips mit OpenCL-Unterstützung sind auch für den mobilen Bereich bereits angekündigt. Auf diese Art ließe sich trotz der begrenzten Ressourcen von mobilen Geräten bezüglich Rechenleistung und Energieverbrauch möglicherweise für manche Algorithmen eine ausreichend hohe Rechenleistung erzielen.

Eng mit dieser Erweiterung verbunden wäre auch die Möglichkeit, die GUI-Komponente auf Android zu portieren. Insbesondere auf Tablet-Geräten ließe sich so die Bildverarbeitung auf mehreren gekoppelten Systemen steuern. Die hier entwickelte Softwarearchitektur ist dafür ausgelegt, auch mehrere GUI-Instanzen gleichzeitig zu unterstützen. Bei der Portierung müsste vor allem untersucht werden, wie sich der Videodatenstrom am besten auf das Tablet-Gerät übertragen ließe. Das Format muss von den verfügbaren GStreamer-Elementen unterstützt werden und es sollte auf dem Tablet-Gerät stromsparend und flüssig angezeigt werden. Zudem sollte die Datenrate bei guter Bildqualität möglichst gering sein, um die Bandbreite der mobilen WLAN-Datenverbindung nicht zu überschreiten. Insbesondere wäre dies auch für 3G- und LTS-Mobilfunkverbindungen zu testen.

[3] first-person-view, Sicht aus der Perspektive des zu steuernden Objekts

5.2.2. Webbasierte GUI-Steuerung

Neben der Portierung der GUI-Komponente auf das Android-Betriebssystem bestünde auch die Möglichkeit, eine Web-basierte Oberfläche zu erstellen. Hierzu müsste ein Web-Server mit entsprechenden Software-Komponenten aufgesetzt werden. Auf dem System, das das Webinterface dann anzeigen soll, müsste lediglich im Browser die Netzwerk-Adresse des Servers geöffnet werden, dann würde sich ein Interface zur Steuerung der Bildverarbeitungspipeline aufbauen. Nebeneffekt dieser Weiterentwicklung wäre ebenfalls die Nutzung auf beliebigen Plattformen, darunter auch Windows, Linux, MacOS und mobile Systeme wie Android und iOS. Ein weiterer Vorteil wäre, dass die Steuerung ohne Installation einer zusätzlichen Software auskommen würde.

Eine Herausforderung bestünde darin, eine Echtzeit-Videoanzeige mit möglichst geringer Verzögerung zu erreichen. Bestehende Mechanismen zur Videowiedergabe im Browser dienen hauptsächlich dazu, auf einem Server gespeicherte Videodaten wiederzugeben. Dementsprechend werden mehrere Sekunden Video zwischengespeichert, um auch bei Schwankungen der Netzwerkdatenrate eine unterbrechungsfreie Wiedergabe zu ermöglichen. Im Hinblick darauf wurden kürzlich bedeutende technische Neuerungen vorgestellt, die eine Echtzeitvideokommunikation zwischen zwei Webbrowsern ermöglichen [BBJN12]. Inwiefern dieses auch für das Streaming von Videodaten nutzbar ist und inwiefern die einzelnen verfügbaren Browser die notwendigen Standards unterstützen, wäre gesondert zu untersuchen.

5.2.3. Untersuchungen zur intelligenten Steuerung der Aufnahmeparameter

Im Rahmen dieser Arbeit wurde die technische Möglichkeit implementiert, die Aufnahmeparameter der Kamera (Verschlusszeit, Gain) abhängig von extrahierten Bildinformationen zu steuern. Über eine Bewegungsdetektion kann gewählt werden, ob die notwendige Bildhelligkeit durch Erhöhung der Verschlusszeit oder des Gain erzielt werden soll. Somit wird entweder Bildrauschen oder Bewegungsunschärfe in Kauf genommen.

Diesbezüglich wäre noch eine quantitative Auswertung notwendig. Es müsste ein Testverfahren entwickelt werden, um konkret die Auswirkungen dieser Methode zu bestimmen. Eine Möglichkeit wäre es, dieses am Beispiel der Objekterkennung zu evaluieren. Voraussetzung hierfür ist, dass die Kamera auf eine Arbeitsfläche mit regelbarer homogener Beleuchtung gerichtet ist. Es darf keine Fremdlichtquellen geben, zudem muss die Beleuchtung in einem sehr weiten Bereich regelbar sein. Des Weiteren muss eine Vorrichtung bereitgestellt werden, um einen Gegenstand mit regelbarer konstanter Geschwindigkeit durch den Bildbereich zu bewegen. Hierzu könnte beispielsweise eine sogenannte Luftschiene benutzt werden, auf der sich Objekte auf einem Schlitten praktisch mit konstanter Geschwindigkeit bewegen können. Der Beschleunigungsvorgang müsste komplett

außerhalb des Sichtfeldes der Kamera stattfinden und die Beschleunigungsenergie müsste präzise einstellbar sein.

Ziel könnte es nun sein, bei verschiedenen Kombinationen von Objektgeschwindigkeiten und Beleuchtungsstärken das Objekt von dem Kamerasystem erfolgreich bestimmen zu lassen. Dies stellt das Kamerasystem vor folgende Problematik: Durch die Objektbewegung entsteht Bewegungsunschärfe und durch eine Erhöhung der Gain-Einstellung wird hohes Bildrauschen erzeugt. Es müsste untersucht werden, wie sich die Erkennungsrate bei der bewegungsgesteuerten Abstimmung im Vergleich zu einer fest vorgegebenen Abstimmung zwischen Gain und Verschlusszeit verhält.

Das erhoffte Ergebnis wäre, dass die Berücksichtigung der Bewegungserkennung zu einer erhöhten Erkennungsrate führt. Insbesondere kann in folgenden Situationen mit Verbesserungen gerechnet werden: Bei ausreichender Beleuchtung könnte auch bei Objekten mit hoher Geschwindigkeit eine erfolgreiche Erkennung möglich sein, was eventuell bei klassischer Regelung infolge von Bewegungsunschärfe scheitern würde. Bei sehr langsamen Objekten und schwacher Beleuchtung könnte erreicht werden, dass eine Erkennung trotzdem erfolgreich wäre, da das Bildrauschen durch eine längere Verschlusszeit und geringere Gain-Werte vermindert werden könnte.

5.2.4. Intelligente RGB+D-Kameras

In den vergangenen Jahren haben sich RGB+D-Kameras in der Robotik zu einem beliebten Sensor entwickelt. Es wäre daher ein naheliegender Forschungsansatz, diesen Kameratyp ebenfalls mit Recheneinheiten zur Bildverarbeitung auszustatten. Ziel der Forschung könnte hierbei die Entwicklung einer intelligenten RGB+D-Kamera sein. Als mögliche Algorithmen, die zu untersuchen wären, kämen Background-Subtraction, Objekterkennung oder auch Gestenerkennung in Frage.

5.3. Schlussbetrachtung

Die Frage, inwiefern der Einsatz intelligenter Kameras auf mobilen Service-Robotern sinnvoll ist, kann nicht pauschal beantwortet werden. Vielmehr müssen unterschiedlichste Faktoren berücksichtigt werden, wie die Verarbeitungsleistung der intelligenten Kamera im Vergleich zur Verarbeitungsleistung des Steuerrechners, die Anforderungen bezüglich der Algorithmen und der maximal zulässigen Latenz und die Möglichkeit, den Steuerrechner mit Schnittstellen für Kameras auszustatten. Im Rahmen dieser Arbeit wurden die Werkzeuge entwickelt, mit denen diese Frage für viele Szenarien beantwortet werden kann. In verschiedenen Szenarien konnte gezeigt werden, dass es durchaus die Möglichkeit gibt, auch leistungsschwächere intelligente Kamerasysteme wie die hier untersuchte Basler eXcite nutzbringend einzusetzen. Der technische Fortschritt wird dazu beitragen,

dass intelligente Kamerasysteme immer leistungsfähiger werden und der Einsatz in immer mehr Szenarien möglich sein wird. Insbesondere kann die Verwendung von parallelen Architekturen zu einer hohen Leistungssteigerung intelligenter Kamerasysteme beitragen. Durch die Einführung eines einheitlichen Standards für die Programmierung (OpenCL) wird der Einsatz paralleler Architekturen zunehmend vereinfacht und lohnenswert. Im Rahmen dieser Arbeit konnte gezeigt werden, dass die entwickelte Software-Architektur für die OpenCL-basierte Verarbeitung bereits ausgelegt ist.

Beitrag dieser Arbeit zur Forschung

Die Ergebnisse der Arbeit können insbesondere als Konzept aufgefasst werden, wie intelligente Kamerasysteme in die Umgebung eines Robotersystems integriert werden können. Die hier vorgestellte Methode ist durch ihre modularen Softwarekomponenten sehr flexibel, ohne jedoch Overhead in einem Ausmaß zu erzeugen, wie ihn gängige modulare Frameworks hervorrufen. Darüber hinaus wird dargelegt, wie die Interaktion mit dem weit verbreiteten Framework ROS erreicht werden kann und so die Vorteile des hier entwickelten Konzepts mit den Möglichkeiten von ROS kombiniert werden können. Bei den Untersuchungen zur Interprozesskommunikation wird ferner ein Verfahren entwickelt, das den Overhead im Vergleich zu gängigen Frameworks auf ein Minimum reduziert.

Ein weiterer Beitrag dieser Arbeit liegt in der Präsentation einer Methode zum Testen und Bewerten von intelligenten Kamerasystemen bei gängigen Bildverarbeitungsaufgaben eines mobilen Robotersystems.

Des Weiteren wird im Rahmen dieser Arbeit anhand konkreter Beispiele demonstriert, wie intelligente Kameras in bestimmten Szenarien gewinnbringend eingesetzt werden können. Dabei werden sowohl die Einsatzmöglichkeiten jetziger leistungsschwächerer als auch zukünftiger leistungsstärkerer Kameras herausgearbeitet. Ferner wird aufgezeigt, wie sich das Konzept der verteilten Verarbeitung auf Steuerrechner und intelligenter Kamera zum Cloud-Computing erweitern lässt. Für die untersuchten Beispiele kann eine deutliche Reduzierung der Verarbeitungszeit erreicht werden.

Darüber hinaus werden Ansätze präsentiert, wie bei einer bestimmten Anzahl von verfügbaren Systemen eine bestehende Bildverarbeitungsaufgabe automatisch verteilt werden kann. Für viele Beispiele ist bereits eine sinnvoller Einsatz möglich, während die Methode für einige Fälle noch weiter optimiert werden müsste.

5.4. Publikationen

Im Rahmen dieser Arbeit konnten Veröffentlichungen auf nationaler und internationaler Ebene erreicht werden. Diese sollen hier kurz mit ihrer thematischen Relevanz zu der Arbeit vorgestellt werden. In [BPWZ07] werden grundlegende Untersuchungen an dem

Robotersystem TASER veröffentlicht. Es wird dort auf die Möglichkeit eingegangen, verschiedene Restriktionen im Hinblick auf Hardwaredesign, Software-Architektur und Performance durch die Benutzung intelligenter Sensoren zu überwinden. In einer weiteren Publikation zu diesen Themenkomplex [BWZ07] werden weitere Implementierungsdetails hinsichtlich intelligenter Sensoren sowie konkrete Messwerte veröffentlicht. Verschiedene an der Universität Hamburg entwickelte Ansätze zur Realisierung von Rechenkapazität in intelligenten Kameras werden in der Publikation [MBZ07] präsentiert. Mit weiteren Messergebnissen und den bis dahin erfolgten Änderungen an der Architektur konnte in [MBZ08] über das Themengebiet eine Journal-Veröffentlichung erzielt werden. Eine Anwendung von selektiver Übertragung von Bilddetails basierend auf Bewegungserkennung konnte auf der 3rd International Conference on Safety and Security Systems in Europe [BVZ09] gezeigt werden. Detaillierte Messergebnisse zur selektiven Übertragung von Regionen, in denen menschliche Gesichter abgebildet sind, sind in [BZ09] publiziert. In diesem Beitrag wird eine deutliche Reduktion von Datenvolumen, Prozessorauslastung und Latenzzeiten dargelegt. In [BZ10] wird aufgezeigt, dass die Vorverarbeitung bei der Objekterkennung zu einer deutliche Reduktion des Datenvolumens führt, so dass die WLAN-Verbindung des Robotersystems benutzt werden kann, um die Daten an beliebig viele weitere Rechnersysteme in der Umgebung zu schicken. Diese können dann die Objekterkennung basierend auf einer großen Datenbank mit bekannten Objekten durchführen. Auf diese Art werden die Restriktionen bezüglich Platzbedarf und Energieverbrauch eines mobilen Robotersystems umgangen. Das in Abschnitt 4.7 vorgestellte omnidirektionale Kamerasystem wird in [BZ12] behandelt.

Ferner wird in [BWS+10] die Verwendung der im Rahmen dieser Arbeit implementierten SIFT-basierten Elemente zur Erkennung von übereinstimmenden Bildausschnitten in Zusammenhang mit Bildern eines Rasterelektronenmikroskops beschrieben.

Ein Beitrag über die implementierte ROS-GStreamer Anbindung sowie die Messwerte zum Overhead bei der Interprozesskommunikation sind bei der Konferenz IROS 2013 eingereicht worden und befinden sich im Review-Prozess[4].

5.5. Projektverlauf

Während der Projektarbeiten zeichnete sich ab, dass das Marktpotential für intelligente Kameras seitens der Basler AG noch als gering eingeschätzt wurde. Aus diesem Grund wurde die Weiterentwicklung und Markteinführung dieses Kameratyps zunächst auf unbestimmte Zeit verschoben. Einen Grund für den bisherigen Misserfolg der schon verfügbaren intelligenten Kamerasysteme am Markt sah die Basler AG in der Auslieferung des Systems ohne passende Bildverarbeitungssoftware. Die Kunden hingegen wünschten einsatzfertige Lösungen. Für den Surveillance-Bereich wurde in der Projektlaufzeit von der Basler AG eine weiterentwickelte intelligente Kamera auf den Markt gebracht, die auch Ergebnisse aus

[4]Stand Mai 2013

den Forschungen zu intelligenten Kameras für Service-Roboter berücksichtigt. Aufgrund der Beschränkung auf Überwachungsfunktionen kann dieser Kameratyp nicht sinnvoll im Robotik-Bereich eingesetzt werden. Eine freie Programmierung durch den Anwender ist nicht möglich.

Die Zusammenarbeit mit der Basler AG wurde fortgeführt und ein Ergebnisaustausch fand weiterhin statt, insbesondere im Bereich der Bildverarbeitungsalgorithmen, die sowohl für die Service-Robotik als auch für den Surveillance-Bereich von Interesse sind. Dazu zählen beispielsweise die Personen- und Gesichtslokalisation, Bewegungserkennung, priorisierte Übertragung von relevanten Bildbereichen und die automatische Steuerung der Aufnahmeparameter der Kamera.

Verzeichnis eigener Publikationen

[BPWZ07] Bistry, H. ; Poehlsen, S. ; Westhoff, D. ; Zhang, J.: Development of a smart laser range finder for an autonomous service robot. In: *Integration Technology, 2007. ICIT '07. IEEE International Conference on* (2007), March, S. 799–804

[BVZ09] Bistry, H. ; Vietze, F. ; Zhang, J.: Towards intelligent high resolution surveillance cameras. In: Künzel, M. (Hrsg.) ; Michel, B. (Hrsg.): *Safety and Security Systems in Europe*, Micro Materials at Fraunhofer IZM Berlin and Fraunhofer ENAS Chemnitz, June 2009 (Micromaterials and Nanomaterials 10). – ISSN 1619–2486, S. 76–79

[BWS+10] Bistry, H. ; Wolter, B. ; Schütz, B. ; Wiesendanger, R. ; Zhang, J.: An Approach for Automated Scale Invariant STM-Scan Matching using SIFT. In: *10th International Conference on Nanotechnology, IEEE NANO 2010* IEEE, 2010

[BWZ07] Bistry, H. ; Westhoff, D. ; Zhang, J.: A smart interface-unit for the integration of pre-processed laser range measurements into robotic systems and sensor networks. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on* (2007), November, S. 358–363

[BZ09] Bistry, H. ; Zhang, J.: Task Oriented Control of Smart Camera Systems in the Context of Mobile Service Robots. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on* (2009), Oktober, S. 3844–3849

[BZ10] Bistry, H. ; Zhang, J.: A Cloud Computing Approach to Complex Robot Vision Tasks using Smart Camera Systems. In: *Intelligent Robots and Systems, 2010. IROS 2010. IEEE/RSJ International Conference on* (2010), October, S. 3195–3200

[BZ12] Bistry, H. ; Zhang, J.: Development of an Intelligent Omnivision Surveillance System. In: *Proceedings of the IEEE First International Conference on Cognitive Systems and Information Processing*, 2012

[MBZ07] Maeder, A. ; Bistry, H. ; Zhang, J.: Towards intelligent autonomous vision systems - smart image processing for robotic applications -. In: *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on* (2007), Dec., S. 1081–1086

[MBZ08] Maeder, A. ; Bistry, H. ; Zhang, J.: Intelligent Vision Systems for Robotic Applications. In: *International Journal of Information Acquisition (IJIA)* 5 (2008), September, Nr. 3, S. 259 – 267

Literaturverzeichnis

[AB89] Adams, A. ; Baker, R.: *Die Kamera*. Christian Verlag GmbH, 1989 (Adams, Ansel: Die neue Ansel-Adams-Photobibliothek). – ISBN 9783884720707

[Ahl04] Ahlers, E.: Hochdruckanschluss: Gigabit-Ethernet-Karten zum Nachrüsten. In: *Magazin für Computer Technik (c't)* 4 (2004), S. 170–175

[AMK+01] Altendorfer, R. ; Moore, N. ; Komsuoglu, H. ; Buehler, M. ; Brown Jr, H. ; McMordie, D. ; Saranli, U. ; Full, R. ; Koditschek, D. E.: RHex: A biologically inspired hexapod runner. In: *Autonomous Robots* 11 (2001), Nr. 3, S. 207–213

[AXZ12] Adler, B. ; Xiao, J. ; Zhang, J.: Towards Autonomous Airborne Mapping of Urban Environments. In: *Multisensor Fusion and Information Integration (MFI), 2012 IEEE International Conference on*, 2012

[Aya91] Ayache, N.: *Artificial vision for mobile robots: stereo vision and multisensory perception*. The MIT Press, 1991

[Bas06] Basler AG (Hrsg.): *Basler eXcite User's Manual*. An der Strusbek 60-62 22926, Ahrensburg: Basler AG, May 2006. `www.baslerweb.com`

[BBJN12] Bergkvist, A. ; Burnett, D. C. ; Jennings, C. ; Narayanan, A.: WebRTC 1.0: Real-time communication between browsers. In: *Working draft, W3C* (2012)

[BCJK05] Broers, H. ; Caarls, W. ; Jonker, P. ; Kleihorst, R.: Architecture study for smart cameras. In: *Proceedings of the EOS Conference on Industrial Imaging and Machine Vision*, 2005, S. 39–49

[BDM+06] Bramberger, M. ; Doblander, A. ; Maier, A. ; Rinner, B. ; Schwabach, H.: Distributed embedded smart cameras for surveillance applications. In: *Computer* 39 (2006), Nr. 2, S. 68–75

[Bel09] Belbachir, A.: *Smart cameras*. Springer Verlag, 2009

[Ben75] Bentley, J. L.: Multidimensional binary search trees used for associative searching. In: *Commun. ACM* 18 (1975), Nr. 9, S. 509–517. – ISSN 0001–0782

[BH09] Bellotto, N. ; Hu, H.: Multisensor-based human detection and tracking for mobile service robots. In: *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 39 (2009), Nr. 1, S. 167–181

[BK08] Bradski, G. ; Kaehler, A.: *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Incorporated, 2008

[BMMC12] Belbachir, A. ; Mayerhofer, M. ; Matolin, D. ; Colineau, J.: Real-time 360° panoramic views using BiCa360, the fast rotating dynamic vision sensor to up to 10 rotations per Sec. In: *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on* IEEE, 2012, S. 727–730

[BN08] Bradshaw, D. ; Ng, K.: Tracking conductors hand movements using multiple Wiimotes. In: *Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS'08. International Conference on* IEEE, 2008, S. 93–99

[Bou04] Bouguet, J.: Camera calibration toolbox for matlab. (2004)

[BPWZ07] Bistry, H. ; Poehlsen, S. ; Westhoff, D. ; Zhang, J.: Development of a smart laser range finder for an autonomous service robot. In: *Integration Technology, 2007. ICIT '07. IEEE International Conference on* (2007), March, S. 799–804

[Bra00] Bradski, G.: The openCV library. In: *DOCTOR DOBBS JOURNAL* 25 (2000), Nr. 11, S. 120–126

[Brä08] Bräunl, T.: *Embedded robotics: mobile robot design and applications with embedded systems*. Springer-Verlag New York Inc, 2008

[BTVG06] Bay, H. ; Tuytelaars, T. ; Van Gool, L.: Surf: Speeded up robust features. In: *Computer Vision–ECCV 2006* (2006), S. 404–417

[BU01] Balasuriya, A. ; Ura, T.: Autonomous underwater vehicles for submarine cable inspection: experimental results. In: *Systems, Man, and Cybernetics, 2001 IEEE International Conference on* Bd. 1, 2001. – ISSN 1062–922X, S. 377–382 vol.1

[BVZ09] Bistry, H. ; Vietze, F. ; Zhang, J.: Towards intelligent high resolution surveillance cameras. In: Künzel, M. (Hrsg.) ; Michel, B. (Hrsg.): *Safety and Security Systems in Europe*, Micro Materials at Fraunhofer IZM Berlin and Fraunhofer ENAS Chemnitz, June 2009 (Micromaterials and Nanomaterials 10). – ISSN 1619–2486, S. 76–79

[BWS+10] Bistry, H. ; Wolter, B. ; Schütz, B. ; Wiesendanger, R. ; Zhang, J.: An Approach for Automated Scale Invariant STM-Scan Matching using SIFT. In: *10th International Conference on Nanotechnology, IEEE NANO 2010* IEEE, 2010

[BWZ07] Bistry, H. ; Westhoff, D. ; Zhang, J.: A smart interface-unit for the integration of pre-processed laser range measurements into robotic systems and sensor networks. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on* (2007), November, S. 358–363

[BZ09] Bistry, H. ; Zhang, J.: Task Oriented Control of Smart Camera Systems in the Context of Mobile Service Robots. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on* (2009), Oktober, S. 3844–3849

[BZ10] Bistry, H. ; Zhang, J.: A Cloud Computing Approach to Complex Robot Vision Tasks using Smart Camera Systems. In: *Intelligent Robots and Systems, 2010. IROS 2010. IEEE/RSJ International Conference on* (2010), October, S. 3195–3200

[BZ12] Bistry, H. ; Zhang, J.: Development of an Intelligent Omnivision Surveillance System. In: *Proceedings of the IEEE First International Conference on Cognitive Systems and Information Processing*, 2012

[Can86] Canny, J.: A computational approach to edge detection. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (1986), Nr. 6, S. 679–698

[Cat05] Catsoulis, J.: *Designing embedded hardware*. O'Reilly Media, Inc., 2005

[CB92] Coombs, D. ; Brown, C.: Real-time smooth pursuit tracking for a moving binocular robot. In: *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on* IEEE, 1992, S. 23–28

[CBC07] *Your View: How would you define a robot?* `cbc.ca/technology/technology-blog/2007/07/your_view_how_would_you_define.html`. Version: July 2007. – Zuletzt abgerufen: Juli 2012

[CC11] Cherubini, A. ; Chaumette, F.: Visual navigation with obstacle avoidance. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on* IEEE, 2011, S. 1593–1598

[CGP89] Cox, P. ; Giles, F. ; Pietrzykowski, T.: Prograph: a step towards liberating programming from textual conditioning. In: *Visual Languages, 1989., IEEE Workshop on* IEEE, 1989, S. 150–156

[CH06] Chaumette, F. ; Hutchinson, S.: Visual Servo Control, Part I: Basic Approaches. In: *IEEE Robotics and Automation Magazine* 13 (2006), December, Nr. 4, S. 82–90

[CH07] Chaumette, F. ; Hutchinson, S.: Visual Servo Control, Part II: Advanced Approaches. In: *IEEE Robotics and Automation Magazine* 14 (2007), March, Nr. 1, S. 109–118

[cmu13] *CMUcam*. `http://www.cmucam.org`. Version: January 2013. – Zuletzt abgerufen: Januar 2013

[ČN04] Čapek, K. ; Novack, C.: *RUR (Rossum's universal robots)*. Penguin Group USA, 2004

[Cou10] Cousins, S.: Ros on the pr2 [ros topics]. In: *Robotics & Automation Magazine, IEEE* 17 (2010), Nr. 3, S. 23–25

[cur13] *Ximea - GPU-beschleunigte PC-Kamera CURRERA-G.* `http://www.ximea.com/de/products-news/gpu-currera-g`. Version: April 2013. – Zuletzt abgerufen: April 2013

[DCK04] Davison, A. ; Cid, Y. ; Kita, N.: Real-time 3D SLAM with wide-angle vision. In: *Proc IFAC Symposium on Intelligent Autonomous Vehicles Lisbon* Citeseer, 2004

[def12] *Robotics Law & Legal Definition.* `http://definitions.uslegal.com/r/robotics/`. Version: July 2012. – Zuletzt abgerufen: Juli 2012

[DHPS07] Dixon, M. ; Heckel, F. ; Pless, R. ; Smart, W.: Faster and more accurate face detection on mobile robots using geometric constraints. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on* (2007), November, S. 1041–1046

[DI09] Dinh, H. ; Inanc, T.: Low cost mobile Robotics Experiment with camera and sonar Sensors. In: *American Control Conference, 2009. ACC'09.* IEEE, 2009, S. 3793–3798

[DLEMSA97] De La Escalera, A. ; Moreno, L. ; Salichs, M. ; Armingol, J.: Road traffic sign detection and classification. In: *Industrial Electronics, IEEE Transactions on* 44 (1997), Nr. 6, S. 848–859

[DRMS07] Davison, A. ; Reid, I. ; Molton, N. ; Stasse, O.: MonoSLAM: Real-time single camera SLAM. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29 (2007), Nr. 6, S. 1052–1067

[DT05] Dalal, N. ; Triggs, B.: Histograms of oriented gradients for human detection. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* Bd. 1 Ieee, 2005, S. 886–893

[ELS+12] Einhorn, E. ; Langner, T. ; Stricker, R. ; Martin, C. ; Gross, H.-M.: MIRA-middleware for robotic applications. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* IEEE, 2012, S. 2591–2598

[FB81] Fischler, M. ; Bolles, R.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. (1981)

[FLZ12] Fang, Y. ; Liu, X. ; Zhang, X.: Adaptive Active Visual Servoing of Nonholonomic Mobile Robots. In: *IEEE Transactions on Industrial Electronics* 59 (2012), Nr. 1

[FMK+04] Frew, E. ; McGee, T. ; Kim, Z. ; Xiao, X. ; Jackson, S. ; Morimoto, M. ; Rathinam, S. ; Padial, J. ; Sengupta, R.: Vision-based road-following using

a small autonomous aircraft. In: *Aerospace Conference, 2004. Proceedings. 2004 IEEE* Bd. 5 IEEE, 2004, S. 3006–3015

[Fra04] Fraden, J.: *Handbook of modern sensors: physics, designs, and applications*. Springer Verlag, 2004

[G+08] Group, K. O. W. ; others: The opencl specification. In: *A. Munshi, Ed* (2008)

[GG07] Gijsenij, A. ; Gevers, T.: Color Constancy by Local Averaging. In: *Image Analysis and Processing Workshops, 2007. ICIAPW 2007. 14th International Conference on* (2007), Sept., S. 171–174

[GLC00] Guo, G. ; Li, S. Z. ; Chan, K.: Face recognition by support vector machines. In: *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on* IEEE, 2000, S. 196–201

[gog12] *Google Goggles*. `http://www.google.com/mobile/goggles`. Version: November 2012. – Zuletzt abgerufen: November 2012

[Gra08] Gray, J.: Distributed computing economics. In: *Queue* 6 (2008), Nr. 3, S. 63–68

[gst13] *GStreamer Editor Website*. `http://code.google.com/p/gst-editor/`. Version: Mai 2013. – Zuletzt abgerufen: Mai 2013

[GVS+01] Gerkey, B. ; Vaughan, R. ; Stoy, K. ; Howard, A. ; Sukhatme, G. ; Mataric, M.: Most valuable player: a robot device server for distributed control. In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on* 3 (2001), S. 1226–1231

[H+00] Hirata, K. ; others: Development of experimental fish robot. In: *Sixth International Symposium on Marine Engineering*, 2000, S. 235–240

[hal13] *Halcon*. `http://www.mvtec.com/halcon/`. Version: February 2013. – Zuletzt abgerufen: Februar 2013

[HBN+08] Hinterstoisser, S. ; Benhimane, S. ; Navab, N. ; Fua, P. ; Lepetit, V.: Online learning of patch perspective rectification for efficient object detection. In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* IEEE, 2008, S. 1–8

[HCO+87] Huang, Y. ; Cao, Z. ; Oh, S. ; Kattan, E. ; Hall, E.: Automatic operation for a robot lawn mower. In: *Cambridge Symposium_Intelligent Robotics Systems* International Society for Optics and Photonics, 1987, S. 344–354

[HF07] Hess, R. ; Fern, A.: Improved video registration using non-distinctive local image features. In: *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2007

[HFFW06] Huang, W. ; Fajen, B. ; Fink, J. ; Warren, W.: Visual navigation and obstacle avoidance using a steering potential function. In: *Robotics and Autonomous Systems* 54 (2006), Nr. 4, S. 288–299

[HGTH04] Hada, Y. ; Gakuhari, H. ; Takase, K. ; Hemeldan, E.: Delivery service robot using distributed acquisition, actuators and intelligence. In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* Bd. 3 IEEE, 2004, S. 2997–3002

[HHC+12] Huang, Y. ; Hu, M. ; Chong, H. ; Jia, X. ; Ma, J. ; Liu, W.: A Survey of Robot Visual Tracking Algorithm. In: *Key Engineering Materials* 501 (2012), S. 577–582

[Hin07] Hinton, G. E.: Learning multiple layers of representation. In: *Trends in cognitive sciences* 11 (2007), Nr. 10, S. 428–434

[Hor86] Horn, B.: *Robot vision*. The MIT Press, 1986

[Hou62] Hough, P.: *Method and means for recognizing complex patterns*. Dezember 18 1962. – US Patent 3,069,654

[HP11] Hennessy, J. L. ; Patterson, D. A.: *Computer architecture – A quantitative approach*. Fifth. San Mateo, CA : Morgan Kaufmann Publishers Inc., 2011. – ISBN 978–0–12–383872–8

[HQZ12] Hu, X. ; Qi, P. ; Zhang, B.: Hierarchical k-means algorithm for modeling visual area v2 neurons. In: *Neural Information Processing* Springer, 2012, S. 373–381

[HS81] Horn, B. ; Schunck, B.: Determining optical flow. In: *Artificial intelligence* 17 (1981), Nr. 1-3, S. 185–203

[HS94] Horn, B. ; Schunck, B.: Determining optical flow-a retrospective. In: *Artif. Intell. v59* (1994), S. 81–87

[hug13] *Hugin - Panorama photo stitcher*. `http://hugin.sourceforge.net/`. Version: Mai 2013. – Zuletzt abgerufen: Mai 2013

[ifr13] *Definition von Service-Robotern*. `http://www.ifr.org/service-robots`. Version: April 2013. – Zuletzt abgerufen: April 2013

[Ing08] Ings, S.: *Das Auge. Meisterstück der Evolution*. Hoffmann und Campe, 2008. – ISBN 9783455500721

[IYDT07] Ishii, I. ; Yamamoto, K. ; Doi, K. ; Tsuji, T.: High-speed 3D image acquisition using coded structured light projection. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on* (2007), Oktober, S. 925–930

[JHM04] Johnston, W. ; Hanna, J. ; Millar, R.: Advances in dataflow programming languages. In: *ACM Computing Surveys (CSUR)* 36 (2004), Nr. 1, S. 1–34

[Joh97] Johnson, G.: *LabVIEW graphical programming: practical applications in instrumentation and control*. McGraw-Hill School Education Group, 1997

[JR11] Joly, C. ; Rives, P.: Self calibration of a vision system embedded in a visual SLAM framework. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on* IEEE, 2011, S. 3320–3326

[KBK08] Kolb, A. ; Barth, E. ; Koch, R.: ToF-sensors: New dimensions for realism and interactivity. In: *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on* IEEE, 2008, S. 1–6

[Kir85] Kirsch, S.: *Detector for electro-optical mouse*. Oktober 8 1985. – US Patent 4,546,347

[Kle10] Kleihorst, R.: Designing a Wireless Smart Camera with a High-Performance Vision System. In: *Smart Cameras*. Springer, 2010, S. 227–244

[KM66] Karp, R. ; Miller, R.: Properties of a model for parallel computations: Determinancy, termination, queueing. In: *SIAM Journal on Applied Mathematics* (1966), S. 1390–1411

[KMLR05] Klose, R. ; Meier, M. ; Linz, A. ; Ruckelshausen, A.: Reihenführung autonomer Roboter mit der Low-Cost-Kamera CMUCam. In: *Bornimer Agrartechnische Berichte* 40 (2005), S. 131–137

[KRZ12] Klimentjew, D. ; Rockel, S. ; Zhang, J.: Towards Scene Analysis based on Multi-Sensor Fusion, Active Perception and Mixed Reality in Mobile Robotics. In: *Proceedings of the IEEE First International Conference on Cognitive Systems and Information Processing*, 2012

[KUI+06] Konno, A. ; Uchikura, R. ; Ishihara, T. ; Tsujita, T. ; Sugimura, T. ; Deguchi, J. ; Koyanagi, M. ; Uchiyama, M.: Development of a High Speed Vision System for Mobile Robots. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* (2006), Oct., S. 1372–1377

[Lee08] Lee, J.: Hacking the nintendo wii remote. In: *Pervasive Computing, IEEE* 7 (2008), Nr. 3, S. 39–45

[LGTB97] Lawrence, S. ; Giles, C. L. ; Tsoi, A. C. ; Back, A. D.: Face recognition: A convolutional neural-network approach. In: *Neural Networks, IEEE Transactions on* 8 (1997), Nr. 1, S. 98–113

[Lie12] Liebrecht, J.: *Entwicklung eines eingebetteten Systems zur intelligenten Steuerung einer Einheit aus Kamera und PTU*. Mai 2012. – Bachelorarbeit

[LJ11] Li, S. ; Jain, A.: *Handbook of face recognition*. Springer, 2011

[LK81a] Lucas, B. ; Kanade, T.: An iterative image registration technique with an application to stereo vision. In: *International joint conference on artificial intelligence* Bd. 3 Citeseer, 1981, S. 3

[LK81b] Lucas, B. ; Kanade, T.: An iterative image registration technique with an application to stereo vision. In: *International Joint Conference on Artificial Intelligence* 3 (1981)

[Lov05] Love, R.: Kernel korner: Intro to inotify. In: *Linux Journal* 2005 (2005), Nr. 139, S. 8

[Low99] Lowe, D.: Object recognition from local scale-invariant features. In: *International Conference on Computer Vision* 2 (1999), S. 1150–1157

[LRM+11] Le, Q. V. ; Ranzato, M. ; Monga, R. ; Devin, M. ; Chen, K. ; Corrado, G. S. ; Dean, J. ; Ng, A. Y.: Building high-level features using large scale unsupervised learning. In: *arXiv preprint arXiv:1112.6209* (2011)

[LS11] Ling, W. H. ; Seng, W. C.: Traffic sign recognition model on mobile device. In: *Computers Informatics (ISCI), 2011 IEEE Symposium on*, 2011, S. 267 –272

[mal12] *ARM submits ARM Mali-T604 GPU for Full Profile OpenCL Conformance.* `www.arm.com/about/newsroom/arm-leads-gpu-computing-trend-improving-systems-performance-and-energy-efficiency.php`. Version: August 2012. – Zuletzt abgerufen: August 2012

[Man08] Manley, J. E.: Unmanned surface vehicles, 15 years of development. In: *OCEANS 2008* IEEE, 2008, S. 1–4

[Mar84] Marzullo, K. A.: *Maintaining the time in a distributed system: an example of a loosely-coupled distributed service (synchronization, fault-tolerance, debugging).* Stanford, CA, USA, Stanford University, Diss., 1984

[mat13] *Matrox Iris GR.* `http://www.matrox.com/imaging/en/products/smart_cameras/iris_gt`. Version: February 2013. – Zuletzt abgerufen: Februar 2013

[MBZ07] Maeder, A. ; Bistry, H. ; Zhang, J.: Towards intelligent autonomous vision systems - smart image processing for robotic applications -. In: *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on* (2007), Dec., S. 1081–1086

[MBZ08] Maeder, A. ; Bistry, H. ; Zhang, J.: Intelligent Vision Systems for Robotic Applications. In: *International Journal of Information Acquisition (IJIA)* 5 (2008), September, Nr. 3, S. 259 – 267

[MFN06] Metta, G. ; Fitzpatrick, P. ; Natale, L.: Yarp: Yet another robot platform. In: *International Journal on Advanced Robotics Systems* 3 (2006), Nr. 1, S. 43–48

[MG11] Mell, P. ; Grance, T.: The NIST definition of cloud computing (draft). In: *NIST special publication* 800 (2011), S. 145

[Möl03] Möller, D.: Eingebettete Systeme. In: *Rechnerstrukturen* (2003), S. 347–366

[nis13] *NI Vision Development Module*. `http://www.ni.com/vision/d/vdm.htm`. Version: February 2013. – Zuletzt abgerufen: Februar 2013

[NKLL10] Nimmagadda, Y. ; Kumar, K. ; Lu, Y.-H. ; Lee, C.: Real-time moving object recognition and tracking using computation offloading. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2010. – ISSN 2153–0858, S. 2449 –2455

[nvc12] *GeForce GTX 680 Grafikkarte*. `http://www.nvidia.de/object/geforce-gtx-680-de.html#pdpContent=2`. Version: August 2012. – Zuletzt abgerufen: August 2012

[OC03] Orebäck, A. ; Christensen, H. I.: Evaluation of architectures for mobile robotics. In: *Autonomous robots* 14 (2003), Nr. 1, S. 33–49

[OP96] O'Donoghue, K. ; Plunkett, T.: Development and validation of network clock measurement techniques. In: *Parallel and Distributed Real-Time Systems, 1996. Proceedings of the 4th International Workshop on* (1996), Apr, S. 65–68

[ope13] *Camera Calibration and 3d Reconstruction*. `http://opencv.willowgarage.com/documentation/python/camera_calibration_and_3d_reconstruction.html`. Version: Mai 2013. – Zuletzt abgerufen: Mai 2013

[PB06] Poehlsen, S. ; Bistry, H.: Entwicklung eines Eingebetteten Systems zur ressourcenschonenden und plattformunabhaengigen Anbindung von SICK-Lasermesssystemen. (2006)

[PHZ+06] Pang, Y. ; Huang, Q. ; Zhang, W. ; Hu, Z. ; Rajpar, A. H. ; Li, K.: Real-time Object Tracking of a Robot Head Based on Multiple Visual Cues Integration. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* (2006), Oct., S. 686–691

[PJ95] Parker, S. ; Johnson, C.: SCIRun: a scientific programming environment for computational steering. In: *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)* ACM, 1995, S. 52

[PK07] Paik, D. ; Kim, B.-S.: A Selective Interpolation Scheme for Mobile Camera Sensors. In: *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on* (2007), Aug., S. 510–514

[Poy12] Poynton, C.: *Digital Video and HD: Algorithms and Interfaces*. Morgan Kaufmann, 2012

[PPC12] Piccinini, P. ; Prati, A. ; Cucchiara, R.: Real-time Object Detection and Localization with SIFT-based Clustering. In: *Image and Vision Computing* (2012)

[PSZL94] Proakis, J. G. ; Salehi, M. ; Zhou, N. ; Li, X.: *Communication systems engineering*. Bd. 94. Prentice Hall New Jersey, 1994

[Puc96] Puckette, M.: Pure Data: another integrated computer music environment. In: *Proceedings of the Second Intercollege Computer Music Concerts* (1996), S. 37–41

[QBG+09] Quigley, M. ; Batra, S. ; Gould, S. ; Klingbeil, E. ; Le, Q. ; Wellman, A. ; Ng, A.: High-accuracy 3d sensing for mobile manipulation: Improving object detection and door opening. In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on* IEEE, 2009, S. 2816–2822

[QGC+09] Quigley, M. ; Gerkey, B. ; Conley, K. ; Faust, J. ; Foote, T. ; Leibs, J. ; Berger, E. ; Wheeler, R. ; Ng, A.: ROS: an open-source Robot Operating System. In: *Open-Source Software workshop of (ICRA)*, 2009

[Qur11] Qureshi, S.: Computer Vision Acceleration Using GPUs. In: *AMD Fusion Developer Summit 11* AMD, 2011. – Zuletzt abgerufen: August 2012

[RB10] Real, F. D. ; Berry, F.: Smart cameras: Technologies and applications. In: *Smart cameras*. Springer, 2010, S. 35–50

[RC11] Rusu, R. B. ; Cousins, S.: 3d is here: Point cloud library (pcl). In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on* IEEE, 2011, S. 1–4

[Ric] Richtlinie, V.: 2860: Montage-und Handhabungstechnik. In: *VDI, Düsseldorf*

[RWPD05] Reinhard, E. ; Ward, G. ; Pattanaik, S. ; Debevec, P.: *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics) (The Morgan Kaufmann Series in Computer Graphics)*. Har/Dvdr. Morgan Kaufmann, 2005. – ISBN 9780125852630

[RZH02] Rossler, B. ; Zhang, J. ; Hochsmann, M.: Visual guided grasping and generalization using self-valuing learning. In: *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on* 1 (2002), S. 944–949

[SB98] Sutton, R. ; Barto, A.: *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998

[Sch75] Schneidermann, R.: Smart Cameras Clicking with Electronic Functions. In: *Electronics* 48 (1975), S. 74–81

[Sch97] Schaller, R. R.: Moore's law: past, present and future. In: *Spectrum, IEEE* 34 (1997), Nr. 6, S. 52–59

[Sch12] Schreiner, R.: *Computernetzwerke, 4. Auflage*. Hanser Verlag, 2012. – ISBN 3446431179

[SEI11] Shibata, M. ; Eto, H. ; Ito, M.: Visual tracking control for stereo vision robot with high gain controller and high speed cameras. In: *Access Spaces (ISAS), 2011 1st International Symposium on* IEEE, 2011, S. 288–293

[SH07] Shimizu, K. ; Hirai, S.: Realtime and Robust Motion Tracking by Matched Filter on CMOS+FPGA Vision System. In: *Robotics and Automation, 2007 IEEE International Conference on* (2007), April, S. 788–793. – ISSN 1050–4729

[SHB07] Sonka, M. ; Hlavac, V. ; Boyle, R.: *Image Processing, Analysis, and Machine Vision*. 3. Cengage-Engineering, 2007. – ISBN 9780495082521

[SKI05] Shimoda, S. ; Kuroda, Y. ; Iagnemma, K.: Potential field navigation of high speed unmanned ground vehicles on uneven terrain. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on* IEEE, 2005, S. 2828–2833

[SPHB08] Schlömer, T. ; Poppinga, B. ; Henze, N. ; Boll, S.: Gesture recognition with a Wii controller. In: *Proceedings of the 2nd international conference on Tangible and embedded interaction* ACM, 2008, S. 11–14

[Sut66] Sutherland, W.: ON-LINE GRAPHICAL SPECIFICATION OF COMPUTER PROCEDURES. / DTIC Document. 1966. – Forschungsbericht

[SWA+02] Sakagami, Y. ; Watanabe, R. ; Aoyama, C. ; Matsunaga, S. ; Higaki, N. ; Fujimura, K.: The intelligent ASIMO: System overview and integration. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on* Bd. 3 IEEE, 2002, S. 2478–2483

[TBW+08] Taymans, W. ; Baker, S. ; Wingo, A. ; Bultje, R. ; Kost, S.: *GStreamer Application Development Manual (0.10.21.3)*. October 2008

[TCP+99] Torres, F. ; Candelas, F. ; Puente, S. ; Jiménez, L. ; Fernández, C. ; Agulló, R.: Simulation and scheduling of real-time computer vision algorithms. In: *Computer Vision Systems* (1999), S. 98–114

[TCPO00] Torres, F. ; Candelas, F. ; Puente, S. ; Ortiz, F.: Graph models applied to specification, simulation, allocation, and scheduling of real-time computer vision applications. In: *International Journal of Imaging Systems and Technology* 11 (2000), Nr. 5, S. 287–291

[TMD+06] Thrun, S. ; Montemerlo, M. ; Dahlkamp, H. ; Stavens, D. ; Aron, A. ; Diebel, J. ; Fong, P. ; Gale, J. ; Halpenny, M. ; Hoffmann, G. ; others: Stanley: The robot that won the DARPA Grand Challenge. In: *Journal of field Robotics* 23 (2006), Nr. 9, S. 661–692

[TP91] Turk, M. ; Pentland, A.: Eigenfaces for recognition. In: *Journal of cognitive neuroscience* 3 (1991), Nr. 1, S. 71–86

[Tsa87] Tsai, R.: A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. In: *Robotics and Automation, IEEE Journal of* 3 (1987), Nr. 4, S. 323–344

[UIY06] Utsumi, Y. ; Iwai, Y. ; Yachida, M.: Performance Evaluation of Face Recognition in the Wavelet Domain. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* (2006), Oct., S. 3344–3351

[UMN97] Ulrich, I. ; Mondada, F. ; Nicoud, J.-D.: Autonomous vacuum cleaner. In: *Robotics and autonomous systems* 19 (1997), Nr. 3, S. 233–245

[VJ01] Viola, P. ; Jones, M.: Rapid Object Detection Using a Boosted Cascade of Simple Features. In: *IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* 1 (2001)

[VLDS+08] Vadakkepat, P. ; Lim, P. ; De Silva, L. ; Jing, L. ; Ling, L.: Multimodal approach to human-face detection and tracking. In: *Industrial Electronics, IEEE Transactions on* 55 (2008), Nr. 3, S. 1385–1393

[Wal09] Walsh, A.: Quick response codes and libraries. In: *Library Hi Tech News* 26 (2009), Nr. 5/6, S. 7–9

[WD92] Watkins, C. ; Dayan, P.: Q-learning. In: *Machine learning* 8 (1992), Nr. 3, S. 279–292. – ISSN 0885–6125

[WGR11] Wu, J. ; Geyer, C. ; Rehg, J.: Real-time human detection using contour cues. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on* IEEE, 2011, S. 860–867

[Win07] Winner, H.: Mit aktiven Sensoren das Kfz-Umfeld erfassen-Funktion und Leistungsfähigkeit von Radar & Co (Detecting the Automotive Surrounding with Active Sensors-Function and Capabilities of Radar & Co.). In: *it-Information Technology* 49 (2007), Nr. 1, S. 17–24

[WOHK10] Waidyasooriya, H. ; Okumura, D. ; Hariyama, M. ; Kameyama, M.: Task Allocation with Algorithm Transformation for Reducing Data-Transfer Bottlenecks in Heterogeneous Multi-Core Processors: A Case Study of HOG Descriptor Computation. In: *IEICE- Transactions on Fundamentals of Electronics, Communications and Computer Sciences* (2010), Nr. 12

[wRo12] *Robot*. `en.wikipedia.org/wiki/Robot`. Version: July 2012. – Zuletzt abgerufen: Juli 2012

[WSZ06] Westhoff, D. ; Stanek, H. ; Zhang, J.: Distributed Applications for Robotic Systems using Roblet-Technology. In: *Proceedings of the 37th International Symposium on Robotics and Deutsche Fachtagung Robotik 2006*. Munich, Germany, Mai 2006

[WWW+08] Wang, W. ; Wang, Y. ; Wang, K. ; Zhang, H. ; Zhang, J.: Analysis of the kinematics of module climbing caterpillar robots. In: *Advanced Intelligent*

Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on IEEE, 2008, S. 84–89

[WZ08] Westhoff, D. ; Zhang, J.: A Unified Robotic Software Architecture for Service Robotics and Networks of Smart Sensors. In: *Autonome Mobile Systeme 2007*, Springer Berlin Heidelberg, 2008 (Informatik aktuell). – ISBN 978–3–540–74763–5 (Print) 978–3–540–74764–2 (Online), 126–132

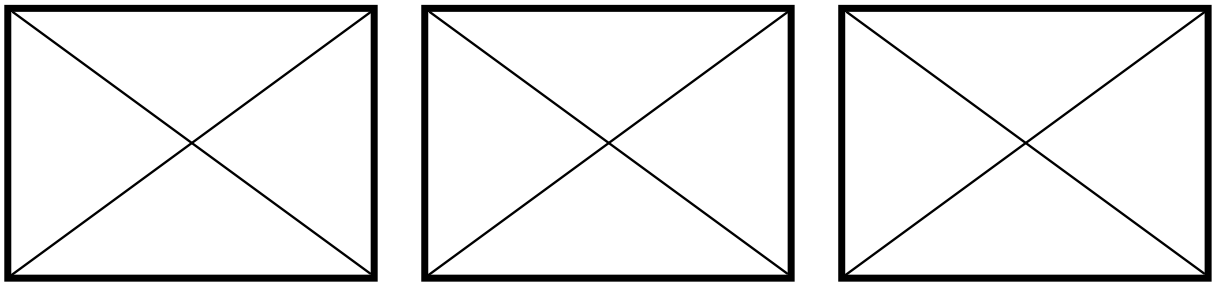
[XJ05] Xu, R. ; Jin, J.: Scheduling latency insensitive computer vision tasks. In: *Parallel and Distributed Processing and Applications* (2005), S. 1089–1100

[Ylo96] Ylonen, T.: SSH–secure login connections over the Internet. In: *Proceedings of the 6th USENIX Security Symposium*, 1996, S. 37–42

[Zha12] Zhang, Z.: Microsoft kinect sensor and its effect. In: *Multimedia, IEEE* 19 (2012), Nr. 2, S. 4–10

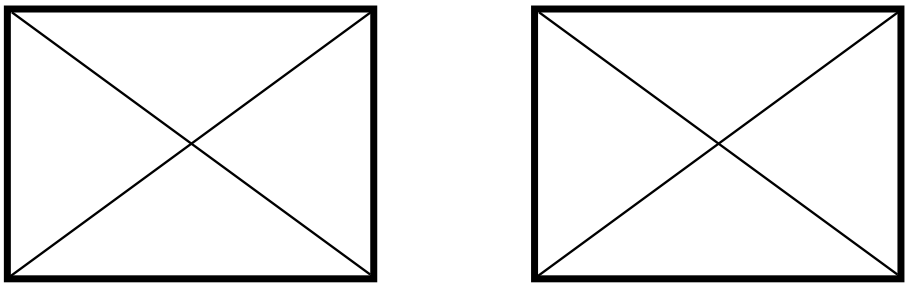
A. Weitere Abbildungen

Die folgenden Abbildungen können aus urheberrechtlichen Gründen in der veröffentlichten Version nicht abgedruckt werden und sind daher durch Links ersetzt. (Gültigkeit überprüft Juni 2013)



Abbildungen aus [mat13] (Seite 2, 6 und 7), Download `http://www.matrox.com/imaging/media/pdf/products/iris_gt/b_iris_gt.pdf`

Abbildung A.1.: Die intelligente Kamera Matrox Iris GT (links), die Entwicklungsumgebung Matrox Design Assistant (Mitte) und das Webinterface (rechts), über das zur Laufzeit Informationen abgerufen werden können, hier beim Einsatz in der Qualitätskontrolle von Backwaren. (Quelle jeweils [mat13])



Abbildungen aus [Lee08] (*Figure 1* und *Figure 2*), Download `http://www.cs.cmu.edu/~15-821/CDROM/PAPERS/lee2008.pdf`

Abbildung A.2.: Der Spielecontroller Wiimote und die darin verbaute intelligente IR-Kamera der Firma PixArt Imaging. (Quelle jeweils [Lee08])

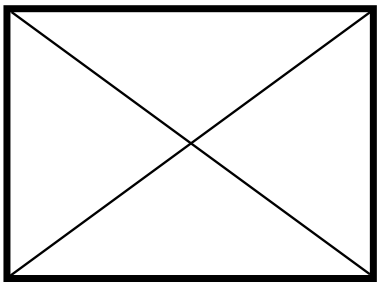
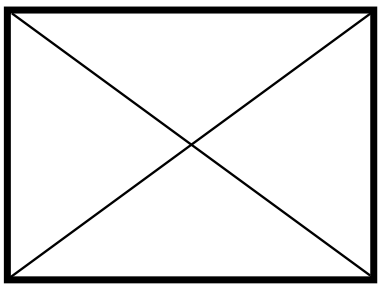
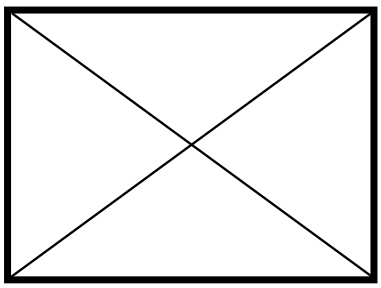


Abbildung von `www.cmucam.org`, direkter Link `www.cmucam.org/attachments/516/CMUcam4_B.JPG`

Abbildung A.3.: Das kompakte intelligente Kameramodul CMUcam in der vierten Version. Mittig auf der Platine ist ein Kameramodul des Herstellers Omnivision integriert. (Quelle: www.cmucam.org)





Abbildungen aus [BMMC12] (*Figure 1* und Bildausschnitt von *Figure 4*), Download (ggfs. Registrierung erforderlich)
`http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6272139`

Abbildung A.4.: Die rotierende Zeilenkamera mit integrierten Verarbeitungseinheiten zur Erzeugung von Panoramabildern (links). Die rechte Abbildung zeigt einen Ausschnitt des Ausgabebildes des Kamerasystems bei einer Umdrehung pro Sekunde und Kantendetektion. (Quelle jeweils [BMMC12])