Fachhochschule Aachen, Campus Jülich

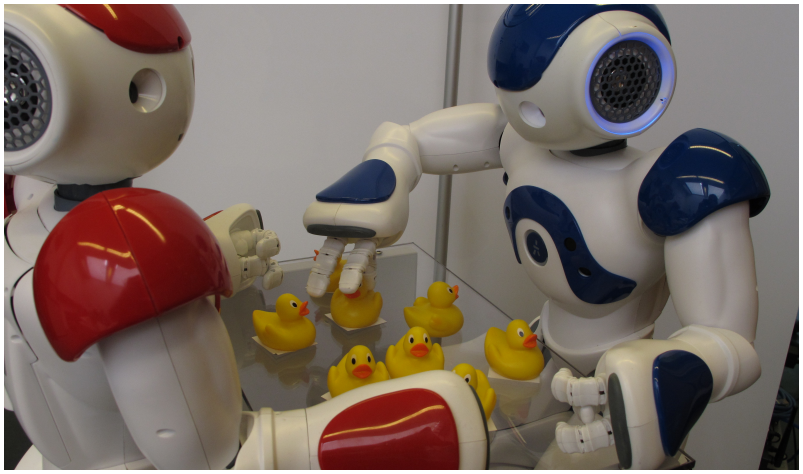
Fachbereich 9 — Medizintechnik und Technomathematik

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Bachelorarbeit
im Studiengang Scientific Programming

Ein Komponenten-Framework zur bildbasierten Steuerung von Robotern

Der memoryspielende Nao



Autor:	Marius Politze
Mat. No.	995282
Erster Prüfer:	Prof. Dr. Andreas Terstegge
Zweiter Prüfer:	Dr. Alexander Voß

Aachen, den 10. Januar 2011

Diese Arbeit wurde durchgeführt am:

Rechen- und Kommunikationszentrum, Abteilung MATSE-Ausbildung
Rheinisch-Westfälische Technische Hochschule Aachen
Dienstgebäude Seffenter Weg 23, 52074 Aachen

Diese Arbeit wurde betreut von:

Prof. Dr. Andreas Terstegge
(Fachhochschule Aachen)

Dr. Alexander Voß
(RWTH Aachen)

Erklärung

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Aachen, den 10. Januar 2011

Marius Politze

Inhaltsverzeichnis

1. Einleitung

Die digitale Bildverarbeitung ist ein interdisziplinäres Teilgebiet der Informatik, das insbesondere in den letzten zehn Jahren immer mehr Anwendungsbereiche aufgedeckt hat.

In der Produktionstechnik werden zunehmend Qualitätskontrollen mit Bildanalysemethoden durchgeführt. Dieses Verfahren kann direkt nach der Produktion durchgeführt werden, sodass Ausschussware automatisch erkannt und aussortiert werden kann. Doch nicht nur die Verbesserung bereits automatisierter Prozesse kann mit Bildanalyseverfahren erreicht werden. Durch die Bildanalyse lassen sich auch völlig neue Vorgänge finden, die automatisiert werden können. Im Maschinenbau wächst die Anzahl der Systeme, die bildverarbeitende Prozesse für die Steuerung nutzen. Insbesondere durch die Automatisierungstechnik konnten Produkte, die auf bildverarbeitende Bestandteile setzen, in den letzten Jahren ihren Marktanteil verdreifachen [VDM08]. Dennoch, oder gerade deshalb, ist die bildbasierte Steuerung insbesondere komplexer Vorgänge ein aktuelles Forschungsgebiet. Dabei steht weniger die Hardware als das Zusammenspiel der Bildeingabe, Bildanalyse und Steuerung der Maschine im Vordergrund.

Neben der industriellen Anwendung von Bildanalysemethoden gibt es auch immer mehr Anwendungen, die für den Endverbraucher direkt zugänglich sind. In diesem Zusammenhang ist vor allem die Gesichtserkennung ein weit verbreitetes Aufgabenfeld. So nutzen viele namhafte Softwarefirmen wie Google, Intel, Microsoft und Sony, aber auch kleine Start-Ups Bildverarbeitungsalgorithmen um neue Funktionsmerkmale in ihrer Software zu ergänzen.

1.1. Anwendungsbeispiele der Bildverarbeitung

Die im Folgenden gezeigten Anwendungsbereiche sind eine kleine Auswahl der aktuellen Aktivitäten rund um Systeme, die mit Hilfe von Bilddaten gesteuert werden. Neben einem Beispiel aus der Anwendungssoftware sind die anderen Beispiele aus dem industriellen Bereich und Zeigen Maschinen bzw. Roboter die durch bildverarbeitende Programme gesteuert werden.

1.1.1. Endbenutzersoftware: Photoverwaltung

Der Suchmaschinenanbieter Google bietet unter anderem die Anwendung Picasa an, mit der Photos auf einem Computer verwaltet werden können. Diese hat neben einer Photodatenbank noch zusätzliche Möglichkeiten. So werden in den Photos Gesichter gesucht. Diese können zunächst nur einem Namen zugeordnet werden. Hat das Programm zu einem Namen genug Bilder gesammelt, schlägt es zu neu eingelesenen Photos Namen vor, wenn die Gesichter in den neuen Photos wiedergefunden werden können.

Neben Anwendungen für Computer rücken Smartphones immer weiter in den Fokus der Softwareentwickler. Der schnelle Fortschritt von Prozessoren und Speicher machten es möglich, dass immer komplexere Anwendungen auf mobilen Geräten lauffähig sind. Zudem bieten so

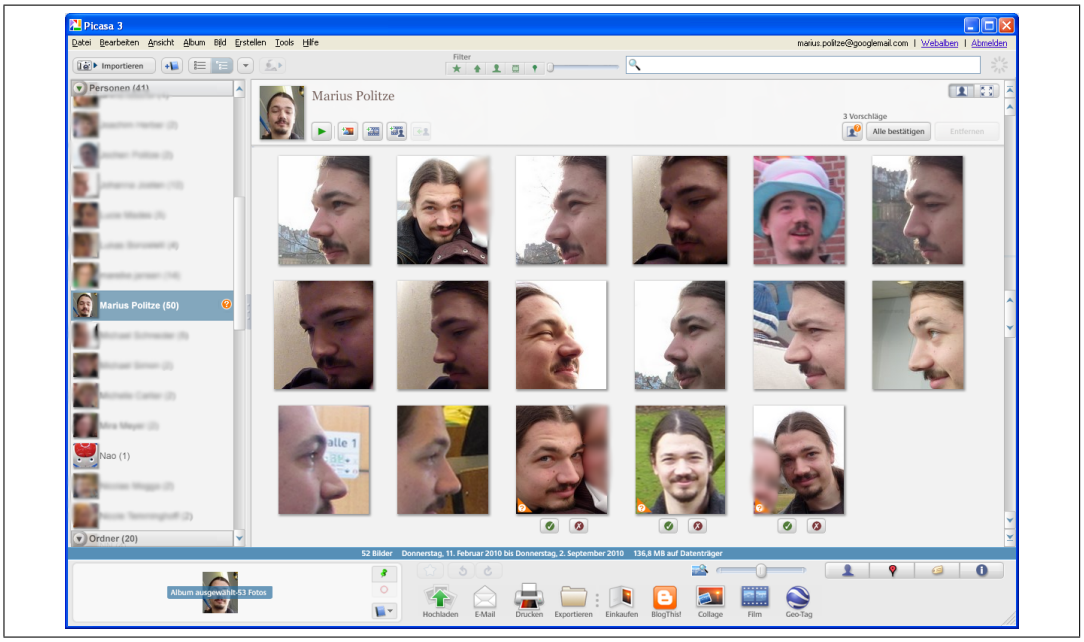


Abbildung 1.1.: Google Picasa durchsucht die Photodatenbank nach bekannten Personen und schlägt Bilder vor, auf denen diese zu sehen sind. Wie an den Schaltflächen unter den Gesichtern zu erkennen, schlägt die Software in diesem Beispiel die letzten drei Bilder vor.

gut wie alle Smartphones eine Kamera als Quelle für die Bilddaten. So lassen sich Bildanalysemethoden schon auf kleinsten Geräten zur Steuerung von Anwendungen verwenden.

1.1.2. Automatisierungstechnik: Plant Phenotyping

Ein aktuelles Forschungsgebiet in der Automatisierungstechnik ist das Plant Phenotyping. Diese Technik analysiert das Wachstum von Pflanzen anhand von Bildern. Angepasst an die aus der Bildanalyse gewonnenen Daten können die Pflanzen dann gedüngt, bewässert und belichtet werden, sodass diese optimal wachsen. Obwohl viele Verfahren dieses Anwendungsbereichs noch sehr neu sind, gibt es bereits Firmen, die Produkte wie automatisierte Gewächshäuser verkaufen. In der Region Aachen gibt es gleich zwei Einrichtungen, die sich mit dem Plant Phenotyping beschäftigen: die LemnaTec GmbH und das Forschungszentrum Jülich GmbH. Beide Einrichtungen entwickeln automatisierte Gewächshäuser, in denen die Pflanzen durch Roboterarme oder Fließbänder bewegt, gedreht und bewässert werden. So wird durch individuelle Behandlung der Pflanzen ein optimales Wachstum ermöglicht.

1.1.3. Forschungsgebiet: RoboCup

Zu den wirtschaftlich orientierten Anwendungsbereichen in Produktion und Endbenutzer-Software gibt es Projekte, die vornehmlich von akademischer Bedeutung sind. Dennoch spielen diese Projekte insbesondere in der Grundlagenforschung eine maßgebliche Rolle für die Weiterentwicklung der Bildverarbeitung. Ein Beispiel für ein solches Projekt ist der RoboCup, ein Wettbewerb, bei dem Roboter gegeneinander Fußball spielen. Bildverarbeitungsalgorithmen werden hier sowohl für die Lokalisierung des Roboters als auch für die

(a) (b)

Abbildung 1.2.: Zwei Beispiele für Aufbauten zum Plant Phenotyping: (a) Ein automatisiertes Gewächshaus der Firma LemnaTec; (b) Roboterarm zur Untersuchung von Pflanzenwachstum aus dem GARNICS-Forschungsprojekt des Forschungszentrum Jülich

Erkennung der Spielsituation genutzt.

Verschiedene Universitäten aus der ganzen Welt nehmen an den Wettkämpfen des RoboCup teil und veröffentlichen den Programmcode ein Jahr nach dem Turnier. So entsteht eine Gemeinschaft, die viel zur Forschung für die bildbasierte Steuerung von Robotern beiträgt.

1.2. Problemstellung

Zur bildbasierten Steuerung von Robotern ist es häufig nötig Komponenten verschiedener Hersteller miteinander zu kombinieren. Insbesondere aufgrund proprietärer Hardware, Treiber und Bibliotheken können Komponenten häufig nicht direkt miteinander verbunden werden und Programmcode muss auf bestimmte Zielhardware angepasst werden.

Zudem soll es ermöglicht werden, ohne lange Einarbeitungszeit Programme zur bildbasierten Steuerung von Robotern zu schreiben. Weiterhin soll der Softwareengineering Prozess von Anfang an, beim Debugging, bis zum Ende bei der Inbetriebnahme bestmöglich unterstützt werden. Dafür ist es beispielsweise nötig, dass Programmteile unabhängig voneinander zunächst einzeln auf ihre korrekte Funktionalität geprüft werden können.

Die Abteilung „MATSE-Ausbildung“ des Rechen- und Kommunikationszentrums der RWTH Aachen möchte zudem bildverarbeitende Programme auf Messen zu Werbezwecken demonstrieren. Bildverarbeitungsalgorithmen erzeugen häufig sogenannte Merkmalsbilder, aus denen mehr Informationen gewonnen werden können, als direkt aus den Eingabedaten. Für ein einfacheres Verständnis ist es notwendig, dass den Messebesuchern auch diese Zwischenschritte der Verarbeitungsalgorithmen gezeigt werden können. Anhand der Zwischenschritte ist es für die häufig fachfremden Besucher einfacher die Funktionsweise der Bildverarbeitungsalgorithmen zu verstehen, das wiederum macht die Vorführung für die Besucher interessanter. Das Visualisieren von Zwischenschritten in vorhandenem Programmcode ist jedoch nicht einfach, da Schnittstellen, die sehenswerte Zwischenergebnisse bieten, zunächst im Bildverarbeitungsalgorithmus gefunden werden müssen.

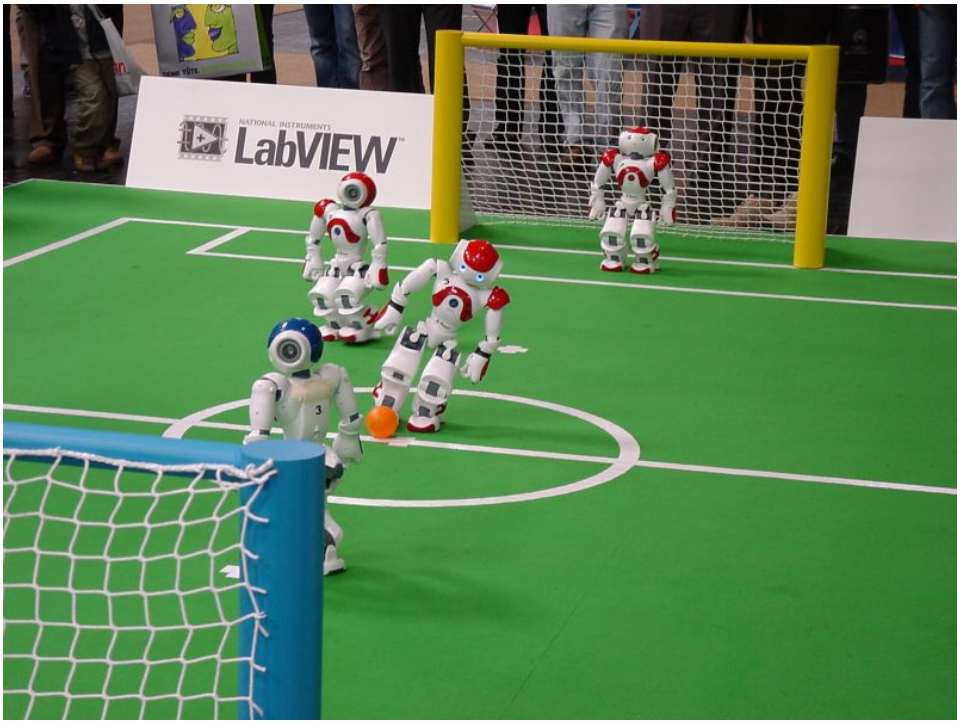


Abbildung 1.3.: In der Standard Platform League des RoboCups treten Teams aus Nao Robotern gegeneinander an.

1.2.1. Anforderungen

Im Rahmen dieser Arbeit wird ein Framework konzipiert und entwickelt, das es ermöglicht solche Bildverarbeitungsalgorithmen als Bildverarbeitungspipeline nach dem Schema Eingabe — Verarbeitung — Ausgabe zu entwickeln und Systemkomponenten verschiedener Herkunft miteinander zu kombinieren. Unter dem Begriff „Ausgabe“ kann hier nicht nur die Ausgabe des Ergebnisses eines Bildverarbeitungsalgorithmus verstanden werden, sondern auch die Umsetzung der Ergebnisse in eine Aktion eines Roboters.

Neben der Kombinationsmöglichkeit von Komponenten wird das Framework auch die Austauschbarkeit einzelner Systemkomponenten ermöglichen. Im Rahmen der Softwareentwicklung wird es durch das Framework möglich zum Beispiel die Bildquelle von einem Kamerabild durch ein Standbild oder Bilder einer Videodatei austauschen, ohne den eigentlichen Bildverarbeitungsalgorithmus zu verändern. Neben der Bildquelle wird es auch möglich sein, andere Teile des Systems auszutauschen, ohne die anderen bestehenden Programmteile verändern zu müssen.

Da auch Darstellungsebenen als Systemkomponenten betrachtet werden und diese ebenso ausgetauscht werden können wie Hardwarekomponenten, ist es dem Framework möglich, die für den Messeeinsatz geforderten Zwischenschritte einfach darzustellen. Ebenso soll durch klare Schnittstellen die Möglichkeit bestehen Verarbeitungsschritte nachträglich anzuzeigen.

Neben diesen Punkten muss das Framework weitere nicht funktionale Anforderungen erfüllen. Diese richten sich zum einen an die Programmiersprache und zum anderen an das Design des Frameworks. Als Programmiersprache wird C++ verwendet, sodass sich auf dem

Framework basierende Programme auf Windows, Linux und Mac OS X kompilieren lassen. Das Framework wird zudem objektorientiert ausgelegt um eine einfache Handhabbarkeit und Wartbarkeit der auf dem Framework basierenden Anwendungen zu erreichen.

1.2.2. Proof of Concept

Nach Entwurf und Implementierung des beschriebenen Frameworks wird eine einfache Anwendung umgesetzt. Diese Anwendung soll Bildverarbeitungsalgorithmen aus OpenCV und Bibliotheken zur Steuerung des Nao Academics Roboters von Aldebaran Robotics mit Hilfe des Frameworks kombinieren. Ziel der Anwendung ist es, dass der Roboter in der Lage ist eine abgewandelte Version des Spiels „Memory“ gegen einen menschlichen Mitspieler zu spielen. Die so entstandene Anwendung soll dann auf Messen zu Werbezwecken eingesetzt werden.

2. Grundlagen

Das zu entwickelnde Framework hat im Wesentlichen zwei Schwerpunkte: Bildverarbeitung und die davon abhängige Steuerung eines Roboters. Um die speziellen Anforderungen die daher an das Framework gestellt werden genauer nachvollziehen zu können, soll zunächst ein Überblick über diese Disziplinen verschafft werden.

2.1. Bildverarbeitung

Bildverarbeitung kann nach [Jäh02, S. 15] in mehrere Teilschritte gegliedert werden. Diese verlaufen nach dem Prinzip Eingabe — Verarbeitung — Ausgabe, wobei insbesondere im Bereich der Bildverarbeitung der zweite Schritt, die Verarbeitung, in zwei Unterschritte aufgeteilt wird: die Vorverarbeitung und die Merkmalsextraktion. In dieses Schema lassen sich auch Steuerungsalgorithmen für Roboter eingliedern, sodass die Ausgabe die Reaktion des Roboters darstellt.

Durch die Aufteilung in Einzelschritte lässt sich ein grobes Schema allgemeiner Bildverarbeitungsalgorithmen, wie in Abbildung 2.1 gezeigt, finden. Aus diesem Ablauf kann ein Konzept abgeleitet werden, nach dem die Verarbeitung von Bilddaten in einem Programm realisiert werden kann.

Im Folgenden werden neben den einzelnen Teilschritten der Bildverarbeitung einige Algorithmen und Verfahren exemplarisch vorgestellt, die in diesen Schritten häufig verwendet werden. Besonders interessant für das Design des Frameworks sind vor allem die verschiedenen Arten der Ergebnisse, die die einzelnen Teilschritte produzieren.

2.2. Eingabe: Bilderfassung

Neben der Kameraaufnahme, also der Abbildung einer realen, dreidimensionalen Szene als ein einzelnes zweidimensionales Bild, werden häufig Bilddaten mit anderen bildgebenden Verfahren erstellt. Da das Framework jedoch für den Anwendungsbereich Robotik konzipiert ist, soll hier nur auf den wesentlichen Ablauf der Kameraaufnahme mit digitalen Bildsensoren eingegangen werden.

2.2.1. Kameraaufnahme

Für die digitale Aufnahme von Bildern einer Szene wird, wie bei der analogen Fotografie, das von Objekten reflektierte Licht durch eine optische Apparatur, das Objektiv, auf einen Bildsensor projiziert. Neben der Komplexität und Qualität des Objektives unterscheiden sich die digitalen Kameras vor allem in der Funktionsweise der Bildsensoren. Die Bildsensoren sind in einzelne lichtempfindliche Segmente von einigen Mikrometern Größe aufgeteilt. Diese Segmente wandeln unabhängig voneinander die Intensität des einfallenden Lichts in

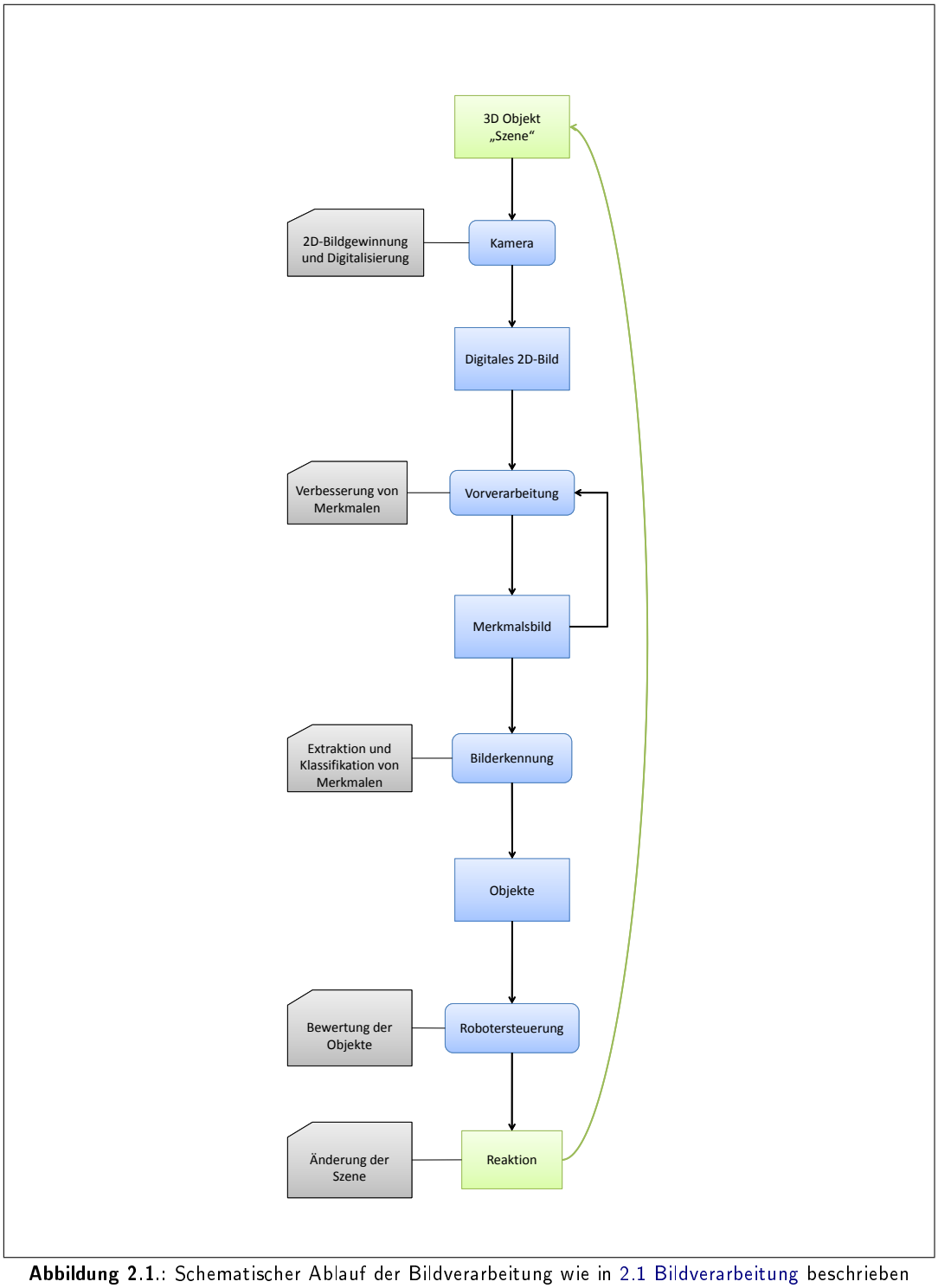


Abbildung 2.1.: Schematischer Ablauf der Bildverarbeitung wie in 2.1 Bildverarbeitung beschrieben

elektrische Signale um. Zur Aufnahme von Farbbildern wird das Licht in der Regel in den einzelnen Farbbestandteilen rot, grün und blau separat von Segmenten unterschiedlicher Farbempfindlichkeit aufgenommen.

Die elektrischen Signale sind je nach Sensortyp unterschiedlich. Neben den verschiedenen Typen gibt es auch verschiedene Anordnungen der Farbsegmente auf dem Bildsensor. Abbildung 2.2b zeigt eine Auswahl von Möglichkeiten, wie die Farbsegmente auf dem Bildsensor angeordnet sein können. Daneben gibt es noch das Drei-Chip-Verfahren, bei dem das Licht zunächst durch ein Prisma in seine Bestandteile zerlegt und dann von drei monochromen Sensoren in elektrische Signale umgewandelt wird, dieses Verfahren liefert eine bessere Bildqualität. Bei handelsüblichen Kameras hat sich aufgrund der Herstellungskosten vor allem in der unteren Preisklasse das Ein-Chip-Verfahren, in der mittleren und oberen Preisklasse das Drei-Chip-Verfahren durchgesetzt.

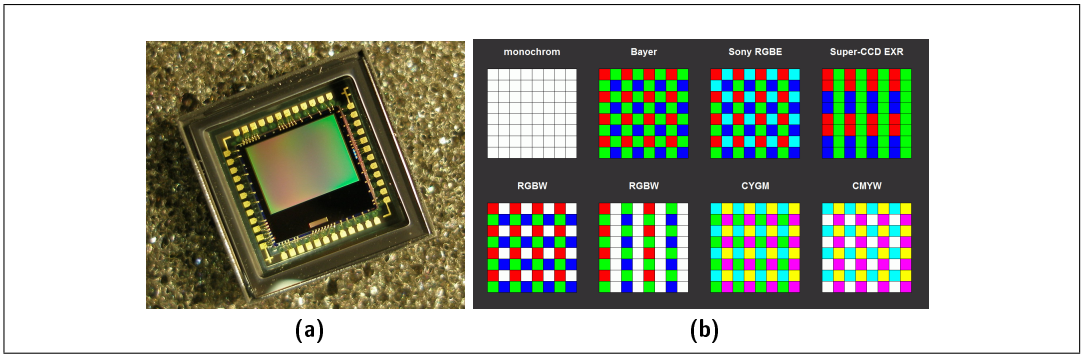


Abbildung 2.2.: Aufbau digitaler Bildsensoren: (a) Abbildung eines digitalen CMOS-Bildsensors, wie er zum Beispiel in Digitalkameras verwendet wird; (b) Eine Auswahl von Möglichkeiten die verschiedenen Farbsegmente auf den Bildsensoren anzuordnen. Die Farbe der Segmente zeigt jeweils die Farbe, für die die einzelnen Segmente am empfindlichsten sind.

2.2.2. Datenformat

Bevor ein Bild mit einem Programm verarbeitet werden kann, muss es in ein Datenformat überführt werden, mit dem das Programm Berechnungen durchführen kann. Dafür werden die elektrischen Signale für jedes Sensorsegment mit einem A/D-Wandler digitalisiert. Für jedes Segment des Bildsensors entsteht so eine Zahl, die die Intensität des Lichts auf dem Bildsensor beschreibt. Die so erstellten Messungen lassen sich dann entsprechend der Anordnung der Sensorsegmente in einer Matrix speichern.

Für die weitere Verarbeitung ist es jedoch nicht sinnvoll ein Datenformat zu wählen, das vom Aufbau des verwendeten Bildsensors abhängt. Die digitalisierten Bilddaten von der Kamera werden daher häufig zunächst in das RGB-Format konvertiert. Dieses Datenformat unterteilt das Bild in einzelne Bildsegmente, diese werden Pixel genannt. Jeder dieser Pixel setzt sich aus insgesamt drei Einträgen zusammen: den Anteilen für rot, grün und blau. Alle Einträge einer Farbe in einem Bild werden Kanal genannt. Idealerweise wird ein Pixel aus einem Sensorsegment berechnet. Die Speicherung erfolgt wiederum in einer zweidimensionalen Matrixstruktur. Abbildung 2.3 zeigt eine Möglichkeit, in welcher Reihenfolge die Werte für den roten, grünen und blauen Farbanteil der einzelnen Pixel in einer Matrixstruktur im Speicher eines Computers abgelegt werden können. Weitere Informationen über Möglichkeiten der Speicherung von digitalen Bilddaten können [Pol10, S. 5ff] entnommen werden.

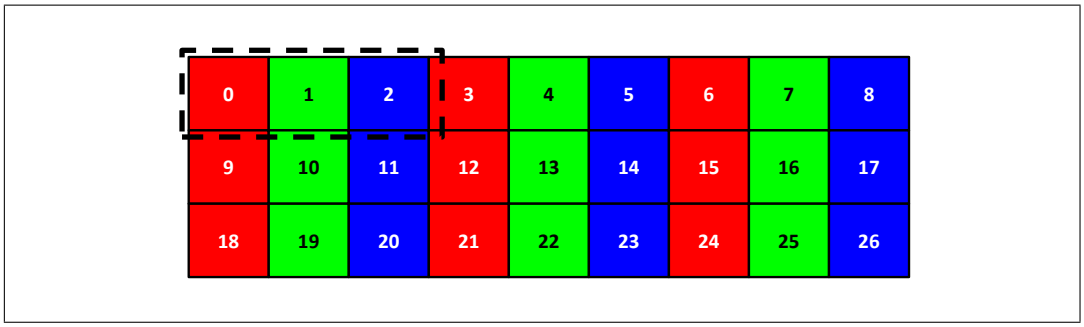


Abbildung 2.3.: Schematische Darstellung der Anordnung der Pixel und ihrer einzelnen Farbanteile im Speicher. Die gestrichelte Umrandung zeigt die Farbanteile, die zum ersten Pixel gehören. Die Rechtecke stellen je einen Farbanteil dar. Die Zahlen in den Rechtecken sind Speicherindizes relativ zum Anfang des Bildes.

Die Speicherung der einzelnen Farbanteile wird entweder im ganzzahligen Wertebereich 0-255 oder in Fließkommadarstellung mit einfacher Genauigkeit zwischen 0.0 und 1.0 realisiert. Im ersten Fall werden 24 Bit pro Pixel benötigt, im zweiten Fall sogar 96 Bit. Bei einer nicht unüblichen Auflösung von 640×480, also insgesamt 307.200, Pixeln belegt ein einzelnes Farbbild 900 bzw. 3600 kByte Speicher. In der Bildverarbeitung spielt die Menge der zu verarbeitenden Daten also eine entscheidende Rolle. Bei 10 Bildern in der Sekunde müssen damit zwischen 9 und 36 Megabyte an Daten pro Sekunde ausgewertet werden.

2.2.3. Datenquellen

Typische Datenquellen im Bereich der Bildverarbeitung sind Bilddateien, Videodateien oder USB-Webcams. Es gibt jedoch nicht nur Anwendungen, die ihre Bilddaten lokal beziehen — häufig finden zum Beispiel IP-Kameras Einsatz, die Bilddaten über ein Netzwerk übertragen. Kamerabilder unterscheiden sich in ihrer Form aber nicht von Bilddateien. Die eigentliche Herausforderung, die Digitalisierung der von der Kamera aufgenommenen Objekte, wird häufig von Kamerahardware oder -treiber übernommen und ist inzwischen selbst bei sehr günstigen Modellen hinreichend genau, sodass diese als Bildquelle für bildbasierte Programme genutzt werden können.

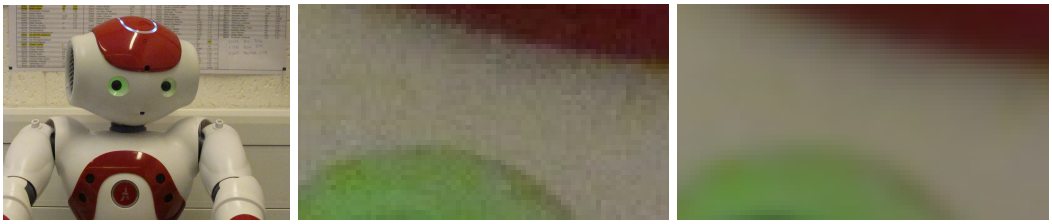
2.3. Vorverarbeitung: Kamerakalibrierung und Bildaufbereitung

Bei der Aufnahme von Objekten mit einer Kamera entstehen im Wesentlichen zwei Probleme, die eng mit der physikalischen Beschaffenheit und Qualität der Kamera zusammenhängen. Zum einen werden die aufgenommenen Objekte durch die Optik der Kamera verzerrt. Dieser Effekt kann zum einen, wie bei dem in Abbildung 2.4 gezeigten Aufbau, beabsichtigt sein ist jedoch in der Regel unerwünscht und muss mit speziellen Algorithmen ausgebessert werden.

Ein weiteres Problem bei der Aufnahme von Bilddaten ist das sogenannte Rauschen, ein Effekt, der bei fast allen Arten der Signalverarbeitung zu berücksichtigen ist. Dieser Effekt äußert sich in der Bildverarbeitung durch zufällige Störungen der Farbwerte einzelner Bildpixel.

(a) (b)

Abbildung 2.4.: Spezieller Aufbau zur Aufnahme eines 360°-Panoramas mit einer Kamera: (a) Konischer Spiegel über der Kamera; (b) Bild der Kamera



(a) (b) (c)

Abbildung 2.5.: Bildstörungen in einem Kamerabild: (a) Komplettes Bild; (b) Vergrößerung eines Bildbereichs; (c) Nach Anwendung eines Weichzeichners

Abbildung 2.5 zeigt die Auswirkungen von Rauschen in einer Kameraaufnahme. Eigentlich einheitliche Flächen zeigen einzelne Pixel, die leicht abweichen. Solche Bildstörungen durch Rauschen können z.B. durch Weichzeichnen behoben werden.

Allgemein ist die Aufgabe der Vorverarbeitung Bildmerkmale zu verstärken oder abzuschwächen. Welche Merkmale eines Bildes von der Vorverarbeitung besonders verstärkt oder abgeschwächt werden, hängt vom konkreten Algorithmus ab. Im Folgenden werden zwei Klassen von Algorithmen vorgestellt, mit denen viele verschiedene Vorverarbeitungsschritte realisiert werden können.

2.3.1. Filter und Transformationen

Die Algorithmen der Vorverarbeitung lassen sich in zwei Klassen aufteilen: Filter und Transformationen. Beide erzeugen aus den Bilddaten sogenannte Merkmalsbilder, auf denen bestimmte Merkmale deutlicher ausgeprägt sind als auf dem Ausgangsbild, sie unterscheiden sich jedoch in der Art und Weise, wie die Ergebnisse berechnet werden.

Filter

Filter sind lokale Operationen auf den Bilddaten. Beispiele für Filter sind:

- Kantenhervorhebung
- Reduzierung von Störungen
- Vereinheitlichung von Flächen.

Aus einem Eingabebild I wird Pixel für Pixel ein Merkmalsbild E berechnet. Als Grundlage für die Berechnung eines Pixels $E(x, y)$ an der Position (x, y) dient der Pixel $I(x, y)$ und seine Umgebung K in den Bilddaten. Die Umgebung K wird Kernel genannt und definiert eine Menge von Pixeln, die um einen Punkt a, den Anker, liegen. Zur Vereinfachung kann davon ausgegangen werden, dass der Kernel immer eine quadratische Umgebung mit ungerader Kantenlänge beschreibt, in deren Mitte der Anker liegt. Nach [Jäh02, S. 104] lässt sich der Filter F dann wie folgt definieren:

$$E(x, y) = F(I(i, j) \; mit \; i, j \in K(x, y))$$

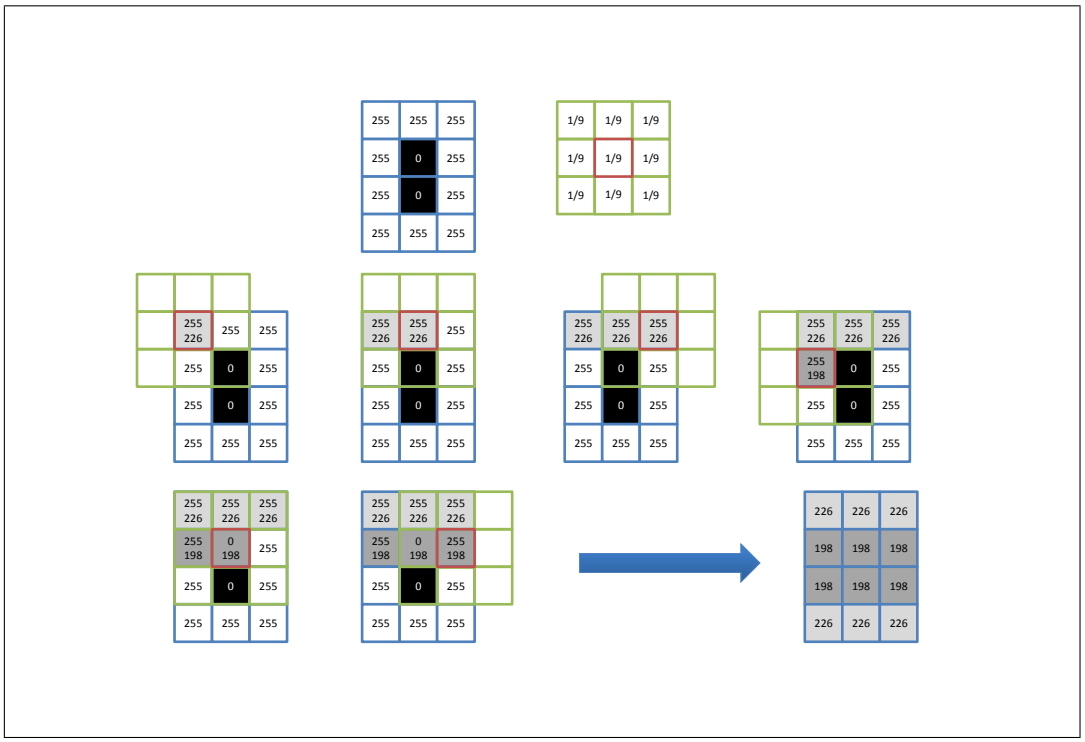


Abbildung 2.6.: Schematische Darstellung des Ablaufs einer Filteroperation. Oben ist ein Einkanalbild mit 3×4 Pixeln (blaue Umrandung) und ein Kernel der Größe 3×3 (grüne Umrandung) dargestellt.

Abbildung 2.6 zeigt beispielhaft eine Filteroperation, die den Durchschnitt der Werte der Pixel im Kernel berechnet. Der dargestellte Filter kann verwendet werden um Bildstörungen wie Rauschen zu verringern. Die Operation, die der Filter ausführt, wird häufig als eine Matrix dargestellt, deren Einträge relativ zum Anker mit den entsprechenden Pixeln multipliziert wird. Die Ergebnisse der Multiplikationen werden addiert. Der Wert des Pixels unter dem Anker wird dann auf den Wert des Ergebnisses der Addition gesetzt. Abbildung 2.7 zeigt zwei mögliche Matrizen.

Die gezeigte Darstellung findet sich zwar häufig, jedoch lassen sich mit ihrer Hilfe nur lineare zusammenhänge beschreiben. Es sind aber durchaus nichtlineare Anwendungen wie Maxima oder Minima üblich.

$$F_{smooth} = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & (\frac{1}{9}) & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \qquad F_{sobel} = \begin{bmatrix} 1 & -2 & 1 \\ 2 & (-4) & 2 \\ 1 & -2 & 1 \end{bmatrix}$$

Abbildung 2.7.: Verschiedene Filter-Matrizen, F_{smooth}: einfacher Weichzeichner, F_{sobel}: Sobel Gradient. Die Werte in Klammern stellen jeweils die Position des Ankers dar.

Transformationen

Globale Operationen, die die Eigenschaften des Bildes ändern, werden Transformationen genannt. Beispiele für diese Eigenschaften sind: Farbcodierung, Bildgeometrie und Bildgröße. Eine Bildtransformation T von einem Eingabebild I in ein Ausgabebild E kann formal dargestellt werden als

$$E(x, y) = T(I, x, y)$$

Streng genommen sind also auch alle Filter Transformationen, dennoch werden die beiden Begriffe genutzt, um lokale von rein globalen Operationen, die sich nicht auf die Benutzung eines Kernels zurückführen lassen, zu unterscheiden.

2.4. Merkmalsextraktion: Bilderkennungsverfahren

Nach der Vorverarbeitung kann die eigentliche Analyse der Bilddaten beginnen. Dafür werden die Bilddaten nach bestimmten Merkmalen durchsucht und anhand dieser Merkmale in Objekte eingeteilt. Beispiele für diese Merkmale sind: Kontrast zwischen angrenzenden Pixeln bzw. Ähnlichkeiten der Anordnung oder Farbe der Pixel.

Die Objekte werden dann analysiert und anhand ihrer Form, Lage, Größe oder Ausprägung klassifiziert. Anhand dieser Merkmale können Bildverarbeitungsalgorithmen dann Entscheidungen, beispielsweise zur Steuerung eines Roboters, treffen. Diese Klassifizierung lässt sich für jedes von den Bildverarbeitungsalgorithmen identifizierte Objekt als Menge von Punkten darstellen.

2.4.1. Vorlagenvergleich

Das generelle Vorgehen dieser Methode ist dem der Filter sehr ähnlich. Beim Vorlagenvergleich, auch Template Matching genannt, werden die Pixel eines Bildausschnitts mit denen einer Vorlage verglichen. Dies geschieht i.d.R. durch Summe der absoluten Differenzen der einzelnen Pixel. Es wird eine also Art Abstand zwischen dem Bildausschnitt und der Vorlage berechnet. Anhand dieses Abstands kann entschieden werden, wie ähnlich bestimmte Bildteile der Vorlage sind. [BK08, S. 214] Zeigt verschiedene Vorschriften zur Berechnung dieses Abstands und liefert weitere Informationen zu diesem Verfahren.

Diese Methode lässt sich gewöhnlich schnell implementieren, benötigt jedoch immer eine Vorlage, die vorher aus dem Bild extrahiert werden muss. Dazu ist diese Methode empfindlich gegenüber der Rotation der Vorlage im Verhältnis zum Bild, sodass eine Drehung der Vorlage um 90° nur noch zu zufälligen Übereinstimmungen führt.

Für jede Position der Vorlage auf dem Eingabebild ergibt das Verfahren einen Wert. Die Ergebnisse können entsprechend der Vorlagenposition in einer Matrixstruktur angeordnet werden. Das Resultat kann also als Merkmalsbild angesehen werden. Auf dem Merkmalsbild können dann Maxima gefunden werden, die die wahrscheinlichste Position der Vorlage auf dem Bild darstellen.

2.4.2. Histogrammvergleich

Ähnlich wie der Vorlagenvergleich arbeitet der Histogrammvergleich. Zunächst wird das Histogramm einer Vorlage berechnet. Dann werden nacheinander von verschiedenen Bildbereichen Histogramme berechnet. Diese Häufigkeitsverteilungen der Farbwerte können mit Hilfe von stochastischen Methoden verglichen werden, sodass eine Aussage getroffen werden kann, wie ähnlich die beiden Histogramme sind. Eine Erläuterung der verschiedenen Methoden ist unter [BK08, S. 201] zu finden.

Diese Methode ist zwar aufwändiger als der unter 2.4.1 Vorlagenvergleich erwähnte Vorlagenvergleich, sie ist jedoch deutlich unempfindlicher gegenüber Rotation. Dennoch bleibt zu beachten, dass diese Methode gänzlich die Form des gesuchten Objekts vernachlässigt.

Der Histogrammvergleich ergibt einen skalaren Wahrscheinlichkeitswert für die Übereinstimmung des Histogramms der Vorlage und dem des Eingabebilds. Bei Farbbildern werden die einzelnen Farbkanäle häufig einzeln miteinander verglichen, sodass dann für jeden Farbkanal ein eigenes Ergebnis berechnet wird.

2.4.3. Konturen

Konturen sind Polygone, also Folgen von Punkten in einem Bild. Diese werden häufig durch den Kontrast aneinander grenzender Pixel bestimmt. Im Wesentlichen gibt es dafür zwei verschiedene Herangehensweisen: Konturen werden frei anhand deren Kontrast gesucht oder so, dass sie geometrischen Formen wie Linien oder Kreisen entsprechen. Über Konturen können die Formen von Objekten verglichen werden. Rotationen der Konturen können durch Drehungen oder Ausrichtung der Polygone anhand von markanten Punkten ausgeglichen werden. Eine genaue Beschreibung der Verfahren ist unter [Jäh02, S. 462f und 453f] zu finden.

Die Algorithmen, die Bilder auf Konturen untersuchen, erhalten als Eingabedaten jedoch keine normalen Bilddaten. Häufig müssen aus den Bilddaten zunächst spezielle Merkmalsbilder berechnet werden. Sogenannte Gradienten, diskretisierte Ableitungen, der Bilddaten bieten eine gute Grundlage für die Erstellung der benötigten Merkmalsbilder.

2.4.4. Segmentierung

Im Gegensatz zu den bisher vorgestellten Algorithmen versucht die Segmentierung das Bild in Teile einzuteilen, die vorher nicht genauer bekannt waren. Dies kann zum einen aufgrund von ähnlichen Eigenschaften benachbarter Pixel, aber auch durch den Unterschied zweier oder mehrerer aufeinander folgender Bilder aus einem Bilddatenstrom geschehen. Beide Ansätze liefern i.d.R. große, zusammenhängende Flächen in den Bilddaten die ähnlich wie Konturen als Menge von Punkten dargestellt werden können.

Pyramidensegmentierung

Die Pyramidensegmentierung nutzt den Ansatz Pixel mit ähnlichen Farben einer Fläche zuzuordnen. Dafür wird für zweidimensionale Bilddaten zunächst der Durchschnitt von je 16 Pixeln berechnet, sodass jedes Pixel Grundlage von vier solcher Durchschnitte wird. Jedes Pixel wird dann einem der vier Durchschnittswerte als Kindknoten zugeordnet und zwar, zu dem die Differenz am kleinsten ist. Dieses Verfahren kann für die Durchschnittswerte wiederholt werden. Durch eine oder mehrere Iterationen der Pyramidensegmentierung lassen sich Hierarchien von Pixeln in einer Baumstruktur erstellen. Auf den einzelnen Ebenen ergeben sich Bildteile verschiedener Größe — die Bilddaten wurden in Segmente aufgeteilt. Die Ränder der Segmente lassen sich dann zum Beispiel als Polygonzug darstellen. Abbildung 2.8 zeigt eine schematische Darstellung, die den Vorgang verdeutlicht. Eine detaillierte Erklärung für das Verfahren liefert [Jäh02, S. 456].

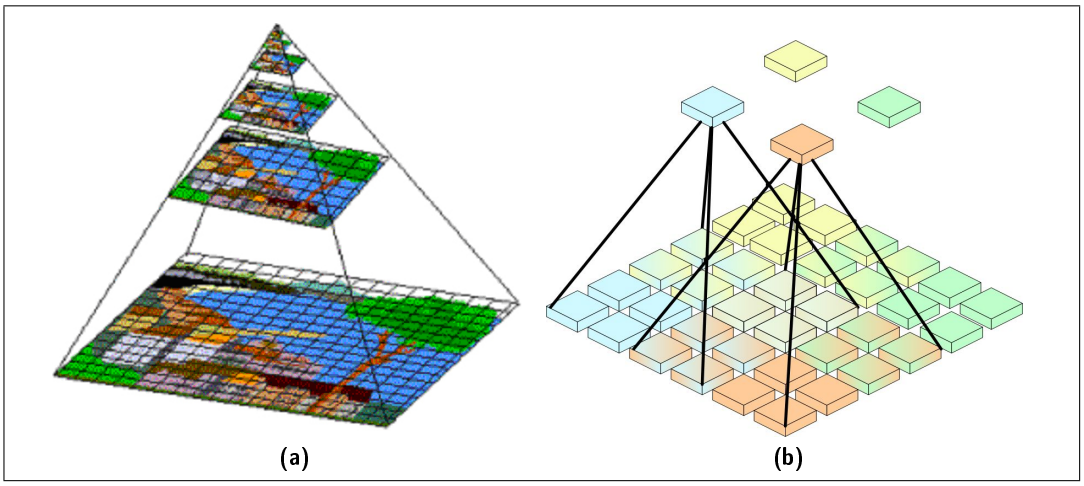


Abbildung 2.8.: Schematische Darstellung der Pyramidensegmentierung: (a) Darstellung der verschiedenen Ebenen der Pyramide, die bei der Iteration entstehen; (b) Darstellung der Vater-Kind-Beziehung zwischen den 16 Kindpixeln und dem Vaterpixel.

Hintergrundsubtraktion

Eine zweite Variante der Segmentierung ist die Hintergrundsubtraktion. Hierfür werden zwei Bilder derselben Szene benötigt. Das eine, das sogenannte Hintergrundbild, ist ein Bild von einer Szene ohne das zu findende Merkmal. Das andere Bild zeigt dieselbe Szene, jedoch mit dem Merkmal — es beinhaltet also den Vordergrund. Die Hintergrundsubtraktion findet dann die Unterschiede der beiden Bilder. Neben einem statischen Hintergrundbild kann auch der Mittelwert mehrerer Bilder verwendet werden. Diese Vorgehensweise macht es möglich periodische Bewegungen wie das Bewegen von Ästen im Wind auszugleichen. Ähnlich wie bei der Pyramidensegmentierung können nun die Umrisse der Unterschiede als Polygonzug erkannt werden. [BK08, S. 265]

2.5. Ausgabe: Roboterplattformen

Die Steuerung des Roboters soll sich nun möglichst nahtlos an die Bildverarbeitungspipeline anschließen. Dafür ist, wie auch bei den Bildverarbeitungsalgorithmen, wichtig die Arten Ein- und Ausgabedaten zu kennen um Schnittstellen für die Robotersteuerung zu identifizieren. Die generelle Aufgabe der Robotersteuerung soll dabei sein auf durch die Bildverarbeitungsalgorithmen erkannte Objekte durch Bewegungen zu reagieren.

2.5.1. Motoren und Gelenke

Damit sich der Roboter bewegen kann, müssen Motoren angesteuert werden. Normalerweise lassen sich die einzelnen Motoren eines Roboters mit einer gewissen Genauigkeit, z.B. 1°, in eine Richtung drehen. Aus der Bewegung des Motors entsteht dann eine Bewegung in einem Gelenk oder anderen Teil des Roboters. Zu beachten ist jedoch, dass wie bei den Eingabedaten immer damit gerechnet werden muss, dass die Roboterhardware nur innerhalb gewisser Toleranzen arbeitet und somit die Rotation des Motors und die daraus resultierende Positionierung des Roboters nie vollkommen exakt durchgeführt werden kann. Eine Kontrolle der Bewegung, möglicherweise ebenfalls mittels Bilderkennungsverfahren, ist daher unbedingt notwendig.

Für die Steuerung des Roboters werden demnach die Positionen der Motoren benötigt, die zu der gewünschten Aktion führen. Soll der Roboter mit einem Objekt interagieren, muss aber zunächst die Position des Objekts im Bild in die Position der Motoren umgerechnet werden: Es müssen also Bildkoordinaten in Roboterkoordinaten umgerechnet werden.

2.5.2. Bildkoordinaten zu Roboterkoordinaten

Da die Konvertierung von Bild- zu Roboterkoordinaten stark vom Aufbau des Roboters abhängig ist, ist es nicht möglich eine allgemeine Vorschrift anzugeben, mit der die Konvertierung durchgeführt werden kann. Generell ist jedoch zu unterscheiden, ob die Bildkoordinaten direkt in Positionen der Motoren umgerechnet werden, oder ob ein Zwischenschritt über Raumkoordinaten eingeführt wird. Dieser Zwischenschritt bietet sich an, wenn die Steuersoftware die Umrechnung von dreidimensionalen Koordinaten in Motorpositionen Anbietet. In 4.5.1 Karten-Lokalisierung und Greifen wird ein Beispiel gezeigt, wie für einen Bestimmten Aufbau eine Konvertierung von Bildkoordinaten in dreidimensionale Koordinaten relativ zum Roboter durchgeführt werden kann. Es gibt jedoch noch weitere Möglichkeiten die Konvertierung durchzuführen.

Zudem spielt die Berechnung von dreidimensionalen Koordinaten aus den zweidimensionalen Bilddaten eine wichtige Rolle. Die Fehlenden Informationen können dann entweder durch Kenntnis des Aufbaus gewonnen werden, oder müssen durch zusätzliche Kameras oder andere Messgeräte aufgezeichnet werden.

3. Entwurf des Frameworks

Wie unter 2.1 Bildverarbeitung beschrieben lässt sich Bildverarbeitung in Einzelschritte unterteilen. Zwischen diesen Schritten entstehen Schnittstellen, die vom Framework aufgegriffen werden können. Zudem ergeben sich aus der Bildverarbeitung weitere Anforderungen: Die Bildquelle liefert häufig einen kontinuierlichen Strom von Bilddaten, dieser muss an die Bildverarbeitungsalgorithmen weitergegeben werden. Durch diesen kontinuierlichen Eingabestrom erzeugen auch die Bildverarbeitungsalgorithmen kontinuierlich neue Ergebnisse, die von der Roboterhardware umgesetzt werden müssen. Das Framework muss demnach so entworfen werden, dass kontinuierliche Eingabedaten in korrekter Reihenfolge abgearbeitet werden.

3.1. Konzept: Pipes und Filter

Das Pipes- und Filter-Konzept beruht auf dem hintereinander Schalten von einzelnen Algorithmen. Die Ergebnisse des ersten Algorithmus sind die Eingabewerte des zweiten und so weiter. Mit diesem Konzept kann aus vielen einzelnen Algorithmen ein Programm zur Lösung des Problems erstellt werden. Das Pipes- und Filter-Konzept wird unter anderem in [Bus98, S. 54ff.] als ein Entwurfsmuster zum Design Datenstrom orientierter Software dargestellt.

Abbildung 3.1 zeigt die Übertragung des allgemeinen Schemas der Bildverarbeitung aus 2.1 Bildverarbeitung auf die Anwendung verschiedener Verarbeitungsalgorithmen. Durch die verschiedenen Komponenten entstehen unzählige mögliche Kombinationen. Jede dieser Kombinationen liefert neue Möglichkeiten für Bildverarbeitungspipelines, die unterschiedliche Probleme lösen. Beispiele für solche Kombinationen sind unter Abbildung 3.2 zu sehen.

Ausgehend von diesen Anwendungsbeispielen wird eine Struktur für das Framework entworfen. Eine Anwendung, die mit dem Framework implementiert wird, kann somit von der Wiederverwendbarkeit der einzelnen Algorithmen, aber auch von der Austauschbarkeit dieser profitieren.

3.2. Komponenten des Frameworks

Das Framework stellt zum einen Interfaces zur Verfügung, die es ermöglichen eine bildbasierte Steuerung von Robotern zu realisieren. Dies sind sowohl Interfaces für die Verarbeitung von Bildern als auch für die Steuerung von Robotern. Ein weiterer Teil des Frameworks sind einige vorgefertigte Implementierungen der Interfaces für häufig genutzte Fälle, wie beispielsweise

- Eingabedaten aus Bilddateien, Videodateien oder USB-Webcams
- Grundlegende Filter und Bildtransformationen wie Weichzeichnen oder geometrische Transformation

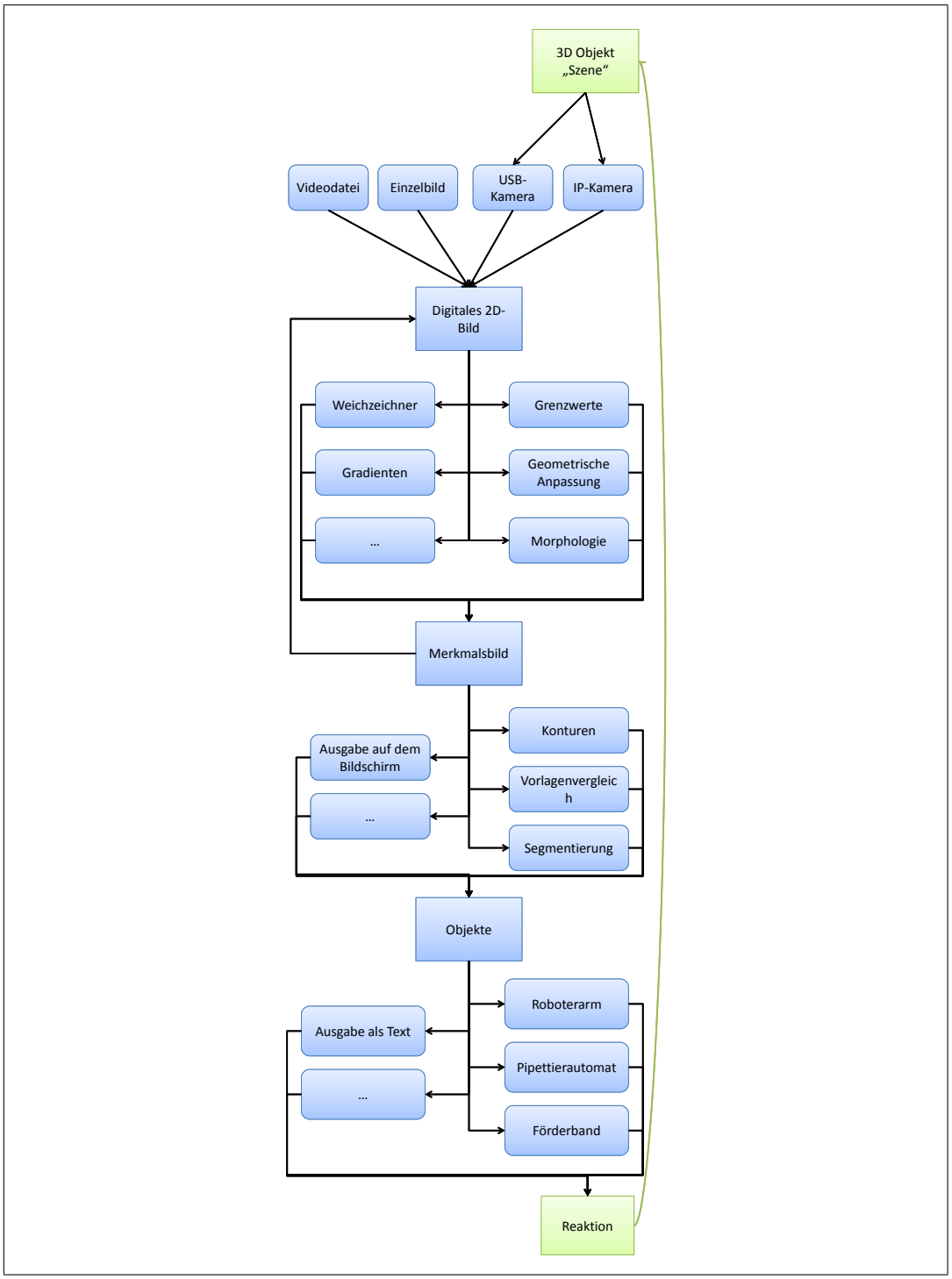


Abbildung 3.1.: Einzelne Schritte der Bildverarbeitungsalgorithmen lassen sich durch andere ersetzen oder ergänzen

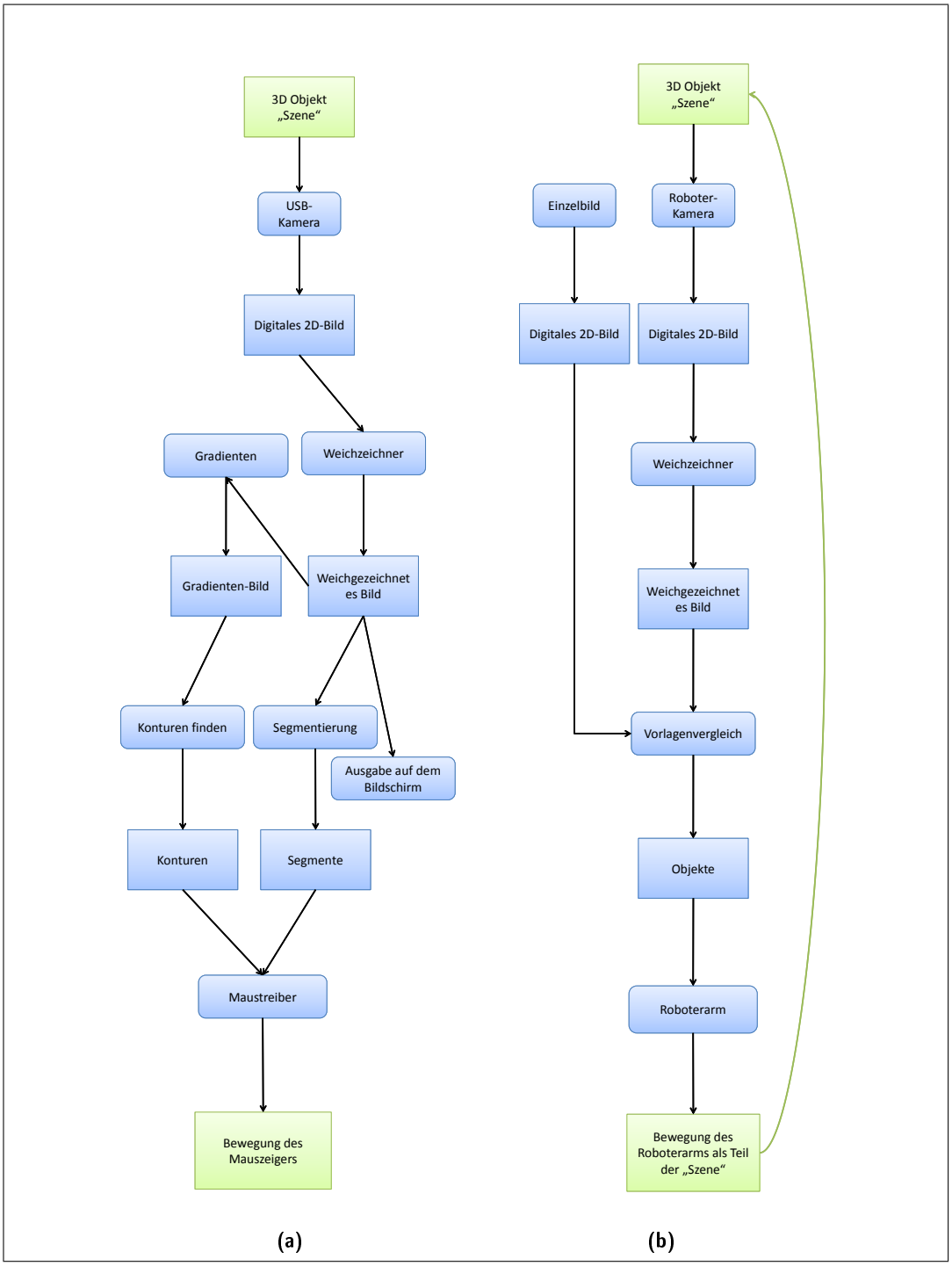


Abbildung 3.2.: Spezialfälle von Bildverarbeitungsalgorithmen anhand des gezeigten Schemas: (a) Steuerung des Mauszeigers durch Aufnahmen von einer USB-Kamera; (b) Steuerung des eines Roboterarms

- Grundlegende Bildanalysen wie Vorlagenvergleich, Extremwerte oder Hintergrundsubtraktion
- Ausgabe von Bilddaten als Datei oder auf dem Bildschirm

3.2.1. Fallstudie: „Memoryspielender Nao“

Zu den recht allgemein gehaltenen Implementierungen für Bildvorverarbeitung und Analyse werden Implementierungen für die Roboterschnittstelle des Nao Academics Roboters erstellt, sodass der Roboter zusammen mit dem Framework eingesetzt werden kann.

3.3. Konzeption des Frameworks

Das Framework implementiert Bildverarbeitung als Abfolge voneinander abhängiger Verarbeitungsschritte. So entsteht eine Art Verarbeitungspipeline, die Bilddaten von der Bildquelle bis zur Ausgabe schrittweise weiterverarbeitet. Das Framework ermöglicht eine solche Bildverarbeitungspipeline aufzubauen und zu kontrollieren. Zu diesem Zweck implementiert das Framework ein Observer-Pattern, das die Weitergabe der Bilddaten zwischen den Verarbeitungsschritten realisiert. Limitierte Ressourcen, wie zum Beispiel Webcams, werden im Singleton bzw. Factory Pattern realisiert.

3.3.1. Konzeption der Programmstrukturen

Für die weitere Konzeption des Frameworks wurde eine Grundstruktur entworfen, die die Vorgänge der Bildverarbeitung und der Steuerung abbildet. Abbildung 3.3 zeigt die grundlegenden Interfaces des Frameworks. Die so modellierten Interfaces erfüllen damit folgende Funktionen:

ImageProvider Klassen, die dieses Interface implementieren sind Bildquellen. Bildquellen verteilen ihre Bilder an die sogenannten Listener, die sich mit der Methode *addSubscriber* angemeldet haben. Somit obliegt es der Bildquelle, die angemeldeten Listener mit neuen Bildern zu versorgen.

ImageListener Um sich an einer Bildquelle anzumelden muss das Interface ImageListener implementiert werden. Die Methode *refreshImage* wird immer dann von der Bildquelle aufgerufen, wenn ein neues Bild zur Verarbeitung bereit ist.

Filter Filter sind eine Zusammenfassung der beiden ersten Interfaces und bilden die Grundlage für die Bildverarbeitungspipeline. Nachdem ein Filter über ein neues Bild benachrichtigt wurde, wird das Bild verarbeitet und die am Filter Angemeldeten Listener werden informiert. Auf diese Weise lassen sich viele Filter hintereinander verketten.

CoordinateProvider Ähnlich, wie ein *ImageProvider* arbeitet auch der *CoordinateProvider*, jedoch nicht mit Bilddaten, sondern mit Tupeln von Vektoren bzw. Koordinaten.

Analyzer Während der Bildanalyse werden aus den Bilddaten Rückschlüsse auf Objekte im Bild gezogen. Diese Objekte lassen sich durch Tupel von Koordinaten beschreiben. Eine Klasse, die das Interface *Analyzer* implementiert kann demnach an das Ende einer Filter-Pipeline gehängt werden, um dort Merkmale des Bildes zu Objekten zu klassifizieren.

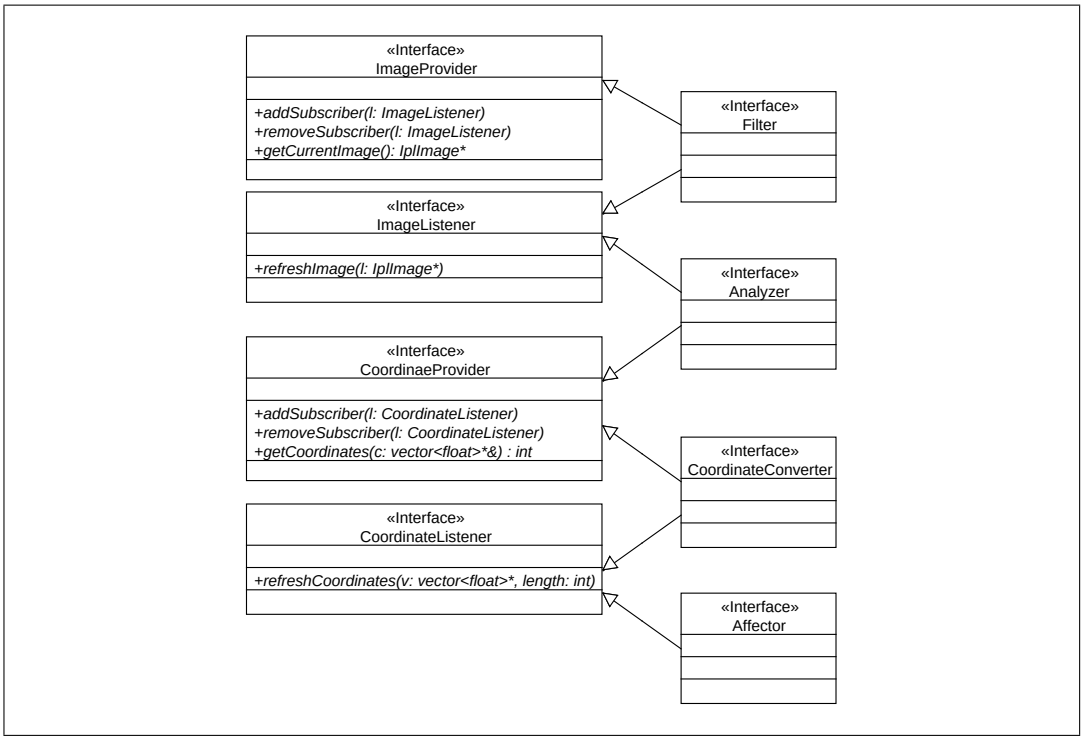


Abbildung 3.3.: Klassendiagramm der grundlegenden Interfaces des Frameworks

CoordinateListener Das anmelden an einem *Analyzer* oder einer anderen Quelle von Koordinaten wird durch Implementieren der Klasse *CoodinateListener* möglich.

CoordinateConverter Eine nachträgliche Verarbeitung der Koordinaten wird durch den *CoordinateConverter* möglich. So können beispielsweise ungültige Objekte herausgefiltert oder Koordinatentransformationen von Bildkoordinaten in Raumkoordinaten durchgeführt werden. *CoordinateConverter* können einen Großteil der Programmlogik implementieren, indem Koordinaten beispielsweise nicht in der Pipeline weitergegeben werden, wenn bestimmte Vorgaben nicht erfüllt werden.

Affector Ein *Affector* ist ein spezieller *CoordinateListener*, der Aktionen des Roboters steuert. Damit bilden Klassen, die dieses Interface implementieren, eine Möglichkeit die Bildverarbeitungspipeline abzuschließen.

3.3.2. Verwendete Pattern und Strukturen

Um das beschriebene Konzept von Pipes und Filtern umsetzen zu können werden andere Entwurfsmuster zu Hilfe genommen. Das Framework stützt sich im Wesentlichen auf zwei unterschiedliche Design Pattern: Observer und Strategy. Das Observer-Pattern wird hier mit Providern und Listenern realisiert, funktioniert aber analog. Die Grundstruktur der Entwurfsmuster stammt aus [Kle09, S. 171ff.].

Observer-Pattern

Das Observer-Pattern besteht aus zwei verschiedenen Klassen: Einem Beobachter und einem Subjekt, das beobachtet wird. Um über Änderungen informiert zu werden, registriert sich der Beobachter bei dem Subjekt. Durch einen Aufruf einer Methode aktualisiert das Subjekt bei einer Änderung die Beobachter. Übertragen auf das Framework ist das Subjekt ein *ImageProvider* oder *CoordinateProvider*. Die Beobachter, *ImageListener* oder *CoordinateListener*, können sich bei den Providern mit der Methode *subscribe()* bei den entsprechenden Providern registrieren. Ab diesem Zeitpunkt werden die Listener durch einen Aufruf der Methode *refreshImage()* über eine Änderung der Bilddaten informiert.

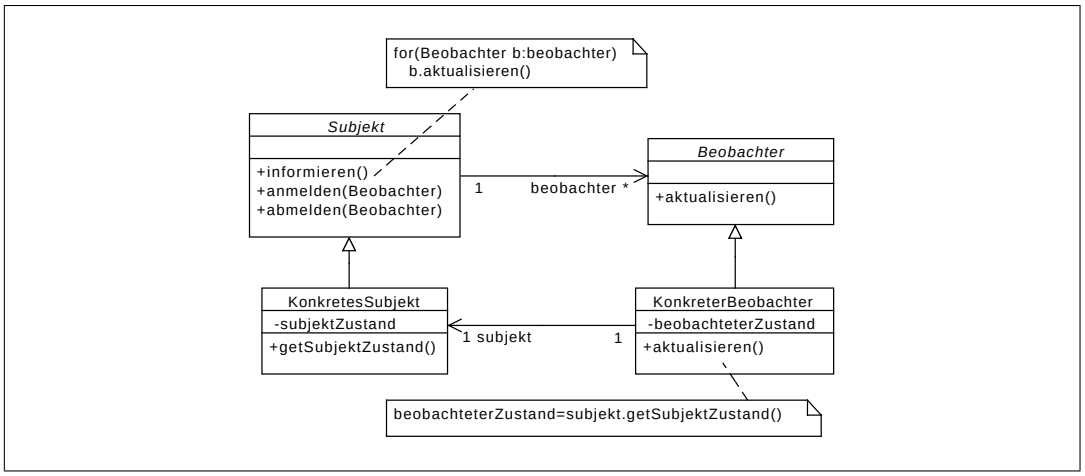


Abbildung 3.4.: Schematische Darstellung des Observer-Pattern nach [Kle09]

Strategy-Pattern

Das Strategy-Pattern bietet eine Möglichkeit für das flexible Austauschen von Algorithmen. Dafür implementieren verschiedene Klassen ein Interface „Strategie“, das von einem Anbieter genutzt wird. Auch dieses Pattern lässt sich auf die Klassen des Frameworks übertragen: Durch die Implementierung der Interfaces *Filter*, *Analyzer* oder *CoordinateConverter*

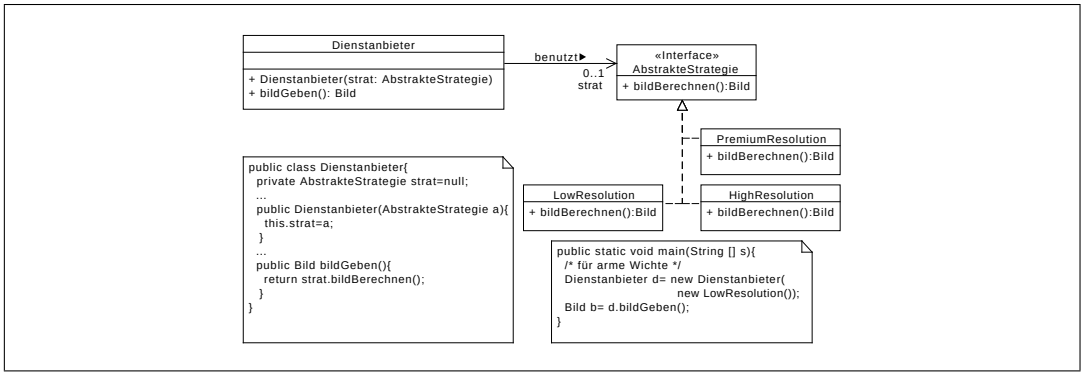


Abbildung 3.5.: Schematische Darstellung des Strategy-Pattern nach [Kle09]

können Klassen ihre Fähigkeit zu bestimmten Prozessen anzeigen. Die so entstehenden Implementierungen der Interfaces lassen sich dann nach dem Strategy-Pattern untereinander austauschen, sodass beispielsweise das Verfahren der Bildanalyse ausgetauscht werden kann ohne die Programmlogik verändern zu müssen.

Adapter-Pattern

Neben dem Observer- und Strategy-Pattern bietet es sich in Bezug auf das Framework an auch das Adapter-Pattern vorzustellen. Im Adapter-Pattern leitet eine Klasse, die ein Interface „Adapter“ implementiert, Methodenaufrufe des Interfaces direkt an ein anderes Objekt weiter. Dieses Entwurfsmuster kann insbesondere dazu eingesetzt werden um bereits vorhan-

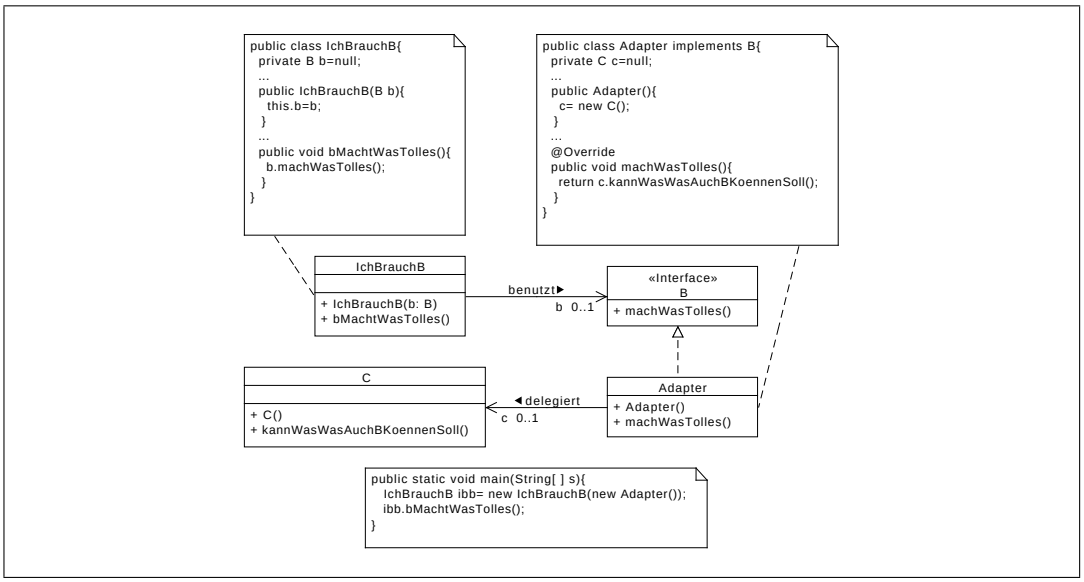


Abbildung 3.6.: Schematische Darstellung des Adapter-Pattern nach [Kle09]

dene Klassen oder Methoden in das Framework einzugliedern. Dazu wird für die vorhandene Klasse, zum Beispiel eine Steuersoftware für eine IP-Kamera, eine neue Klasse geschrieben, die das Interface *ImageProvider* implementiert und die Aufrufe für *getCurrentImage()* durch die entsprechenden Aufrufe für die IP-Kamera umsetzt.

Singleton-Pattern

Wie das Adapter-Pattern lässt sich das Singleton-Pattern nicht direkt in den Interfaces des Frameworks wiederfinden,es ist jedoch für die Implementierung der Interfaces möglicherweise interessant. Ein Singleton kann genutzt werden um den Zugriff auf auf eine knappe Ressource zu kontrollieren. Im Bereich der Bildverarbeitung ist eine solche knappe Ressource zum Beispiel eine Kamera oder ein bestimmtes Anzeigemedium.

Ein Singleton ist eine Instanz eines Objekts, die nur einmal — für den ersten Aufruf — initialisiert wird. Alle Zugriffe auf das Objekt werden durch eine meist statische Methode eine sogenannte Factorymethode realisiert. Damit keine weitere Instanz der Klasse erstellt werden kann, gibt es i.d.R. keinen öffentlichen Konstruktor.

3.3.3. Erweiterbarkeit

Das bisher gezeigte Konzept des Frameworks liefert nur eine Struktur, beinhaltet jedoch noch keinerlei Funktionalität. Diese wird erst mit der Implementierung der gegebenen Interfaces realisiert. Durch die vorgegebene Struktur ist es jedoch möglich das Framework an allen Stellen zu erweitern und so unter Wiederverwendung bestehender Strukturen neue Bildverarbeitungsalgorithmen zu realisieren.

Einige besonders häufig genutzte Bildverarbeitungsalgorithmen sollen jedoch als Implementierung mit in das Framework aufgenommen werden.

3.4. Im Rahmen des Frameworks Umzusetzende Implementierungen

Für die Implementierungen der für das Framework entworfenen Interfaces werden zunächst einige grundlegende Vorgänge der Bildverarbeitung betrachtet, die unter 2.1 Bildverarbeitung bereits vorgestellt wurden. Anhand dieser Betrachtung und der mit dem Framework zu lösenden Aufgaben, die unter 1.2.2 Proof of Concept beschrieben wurden, wird eine Auswahl von Algorithmen und Strukturen getroffen. Diese werden dann wie unter 4 Implementierung des Frameworks Implementiert.

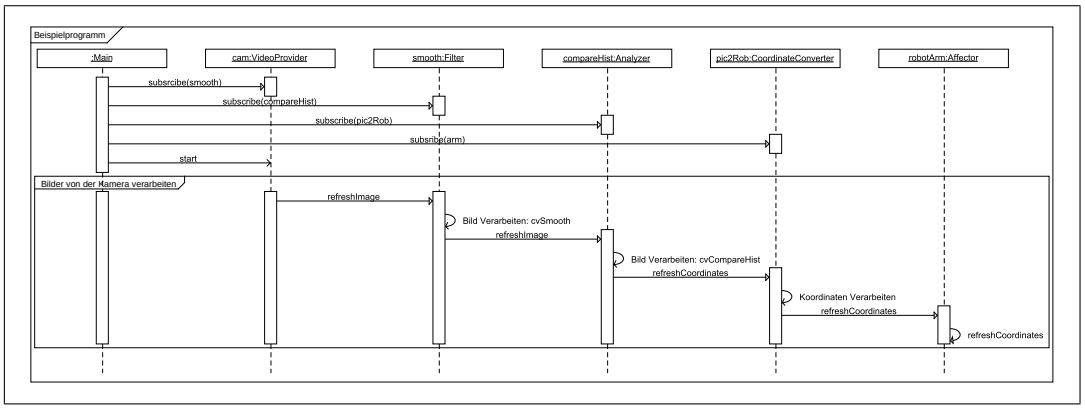


Abbildung 3.7.: Möglicher Ablauf zur Implementierung von bildverarbeitenden Programmen mit den Klassen aus dem Framework.

Mit den so erstellten Klassen kann dann ein Ablauf implementiert werden, der einen Roboter mit Hilfe von Bildverarbeitungsalgorithmen steuert. Abbildung 3.7 zeigt einen Beispielhaften Ablauf eines Solchen Programms als Sequenzdiagramm. Zunächst wird das Observer-Pattern genutzt um die einzelnen Programmbestandteile aneinander anzumelden. Nachdem die Bildverarbeitungsalgorithmen über die Verfügbarkeit eines neuen Bildes informiert werden, rufen die Klassen die Funktionen anderer Bibliotheken auf um die eigentliche Bildverarbeitung durchzuführen. Listing 4.1 Zeigt ein Beispielprogramm, dass einen Histogrammvergleich nach dem Schema auch Abbildung 3.7 durchführt.

Abbildung 3.8 zeigt die Struktur des Frameworks in Form eines UML Klassendiagramms. Die Funktionsweise der einzelnen Klassen ist unter 4 Implementierung des Frameworks beschrieben. Bevor die Funktionsweisen der einzelnen Klassen beschrieben werden, sollen diese zunächst unabhängig von konkreten Implementierungen beschrieben werden.

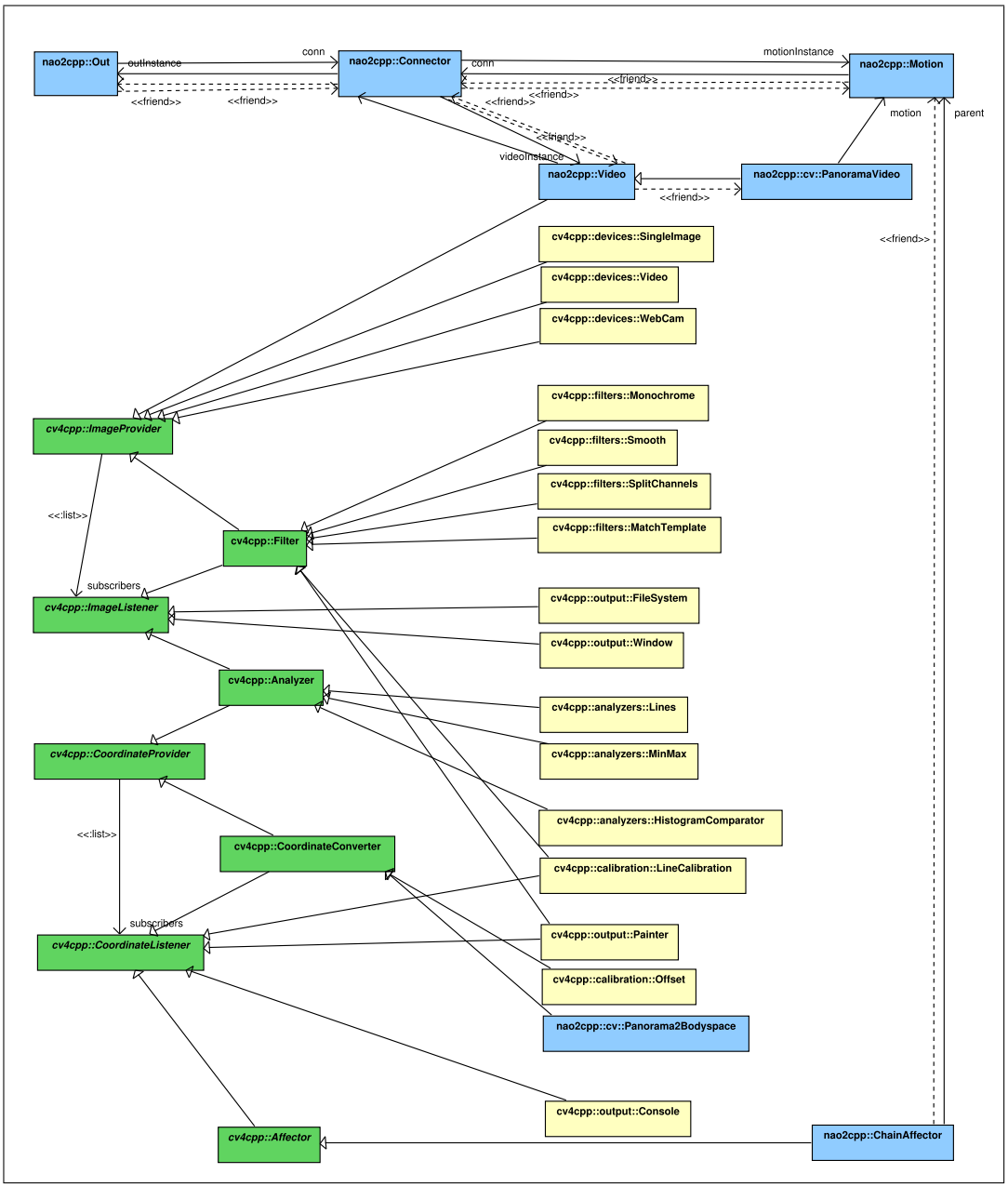


Abbildung 3.8.: Klassendiagramm aller für das Framework zu implementierenden Klassen. Die Klassen in blau wurden nur für die Anbindung an den Nao Academics Roboter implementiert. Die Klassen in grün sind die bereits vorgestellten Interfaces.

3.4.1. Bildquellen: *ImageProvider*

Die zu implementierenden Bildquellen sind vor allem durch Relevanz für die Entwicklung der Bildverarbeitungssoftware motiviert. So ist es zum Beispiel für das Testen der Software nötig, dass Bilder analysiert werden, zu denen die Ergebnisse der Algorithmen bekannt sind, um eine Vergleichbarkeit herzustellen. Zu diesem Zweck sollen *ImageProvider* für Einzelbilder wie JPEG oder PNG und Videodateien wie AVI oder MPEG implementiert werden. Zudem wird die Möglichkeit geschaffen Bilder von einer USB-Webcam als Eingabedaten zu nutzen. Diese Kameras sind durch den günstigen Anschaffungspreis ideale Bildquellen, insbesondere für Systeme, die ohnehin von einem PC gesteuert werden oder für die keine eigenen Bildquellen existieren.

Insbesondere im Hinblick auf die unter 1.2.2 Proof of Concept beschriebene Zielanwendung ist die Anbindung an die Kameras des Nao Roboters von Aldebaran Robotics notwendig. Als Beispiel für die Adaption eines proprietären Kamerasystems, wird ebenfalls eine solche Implementierung erstellt. Zudem wird die Steuerung der Gliedmaßen des Roboters mit Klassen, die Interfaces aus dem Framework implementieren, ermöglicht.

3.4.2. Bildverarbeitung: *Filter* und *Analyzer*

Neben den Implementierungen der Bildquellen werden auch einige Standardfälle von Bildverarbeitungsalgorithmen implementiert. Dies sind vor allem die unter 2.1 Bildverarbeitung aufgezeigten Filter und Bildanalysemethoden.

Weichzeichnen

Im Allgemeinen ist das Weichzeichnen eine elementare Operation der Bildverarbeitung. Es gibt jedoch nicht nur eine Variante dieses Filters. Für das Framework werden verschiedene Arten von Weichzeichnungsfiltern implementiert. Dies sind der lineare Weichzeichner, der gaußsche Weichzeichner und der mediale Weichzeichner. Genauere Angaben zu den einzelnen Weichzeichnungsalgorithmen können unter [BK08] nachgelesen werden.

Geometrische Korrektur

Die geometrische Korrektur wird genutzt um Verzerrungen des Bildes, die beispielsweise durch das Objektiv verursacht werden auszugleichen. Für diese Korrektur wird das Framework Rotation und trapezförmige Verzerrung der Bilddaten anbieten.

Farbkanäle

Wie unter 2.2.2 Datenformat beschrieben werden Bilddaten farbiger Bilder in Farbkanäle aufgeteilt. Zur Konvertierung zwischen verschiedenen Farbdarstellungen werden Filter implementiert. Vor allem die Umsetzung von Farbwerten in Grauwerte spielt in diesem Zusammenhang eine Rolle, da die Reduzierung auf Grauwerte die Menge der Eingabedaten um zwei Drittel verringert.

Vorlagenvergleich

Das Framework soll auch die Möglichkeit bieten Bilder des Bilddatenstroms mit einer definierten Vorlage zu vergleichen um die wahrscheinlichste Position der Vorlage in den Bilddaten bestimmen zu können.

Histogrammvergleich

Der Vergleich von Histogrammen bietet die Möglichkeit die Ähnlichkeit von Bilddaten zu bestimmen. Dafür wird zunächst von den Bilddaten und einer Referenz eine Häufigkeitsverteilung der Farbwerte, ein sogenanntes Histogramm, gebildet. Anschließend kann verglichen werden, ob beide Verteilungen ähnlich sind. Die verschiedenen Vergleichsmethoden können beispielsweise [BK08] entnommen werden. Das Framework wird eine Möglichkeit schaffen, mit der ein Bild mit dem Bilddatenstrom verglichen werden kann.

Konturen

Eine weitere Bildanalysemethode, die das Framework implementieren wird, ist das Auffinden von Konturen. Dafür können entweder Polygonzüge oder sogenannte Bounding Boxes genutzt werden. Bounding Boxes sind das kleinste Rechteck, in das die Kontur vollständig passt. Die Angabe von Bounding Boxes ist zwar ungenauer, dafür ist deren Darstellung jedoch wesentlich einfacher.

Segmentierung

Für die farbbasierte Segmentierung wird das Framework eine Implementierung der Pyramidensegmentierung anbieten. Wie auch im Fall der Konturen soll es möglich sein für die gefundenen Segmente des Bildes Polygonzüge und Bounding Boxes zu berechnen.

Extremwerte

Zudem wird das Framework die Möglichkeit bieten Maximalwerte in Bildern zu finden. Dies ist vor allem dann nötig, wenn zum Beispiel durch einen Vorlagenvergleich ein Merkmalsbild generiert wird, auf dem für jedes Pixel ein Ergebnis des Vergleichs berechnet wurde. Mit der Extremwert-Operation kann dann der Punkt im Bild mit dem besten Vergleichswert gefunden werden.

3.4.3. Visualisierung

Im Hinblick auf das unter 1.2.2 Proof of Concept beschriebene Einsatzgebiet ist die Visualisierung der Bilddaten an jedem Punkt der Bildverarbeitungspipeline eine wichtige Anforderung an das Framework. Durch die Konzeption wird die Möglichkeit geschaffen die Verarbeitung an den Schnittstellen zwischen den Algorithmen zu unterbrechen, beziehungsweise auf die Zwischenergebnisse zuzugreifen. Die so entstandenen Daten sollen dann möglichst anschaulich ausgegeben werden können.

Anzeige auf dem Bildschirm

Für die Visualisierung der Bilddaten wird das Framework eine Schnittstelle zum GUI-Framework Qt bereitstellen. So können Ergebnisse von Bildverarbeitungsalgorithmen in ansprechend entworfene Fenster eingebettet werden. Damit das Framework auch auf Plattformen benutzbar bleibt, auf denen keine Anzeige, beziehungsweise kein Qt, verfügbar ist, wird die beschriebene Klasse von den restlichen Teilen des Frameworks getrennt.

Speichern von Bilddaten

Neben der Anzeige der Bilder auf dem Bildschirm liefert das Framework die Möglichkeit Bilddaten als einzelne oder Folge von Bilddateien abzuspeichern. Diese Funktionalität kann beispielsweise genutzt werden um nachträglich Fehler in den Algorithmen nachvollziehen zu können, oder Sätze von realistischen aber wohl bekannten Bilddatenströmen zu erzeugen, deren Ergebnisse bekannt sind. Solche Bilddatenströme können dann beispielsweise für das Testen der Software eingesetzt werden.

Zeichnen

Zur Visualisierung von Ergebnissen wird das Framework es ermöglichen einige Standardstrukturen wie Punkte, Bounding Boxes und Polygonzüge in Bilder einzuzeichnen. Dies ermöglicht eine intuitive Darstellung von Ergebnissen der Bildanalyse.

4. Implementierung des Frameworks

Mit den unter 3.3 Konzeption des Frameworks entwickelten Konzepten kann das Framework nun umgesetzt werden. Zu diesem Zweck werden zwei Bibliotheken vorgestellt, die bereits Methoden zur Lösung von Problemen dieser Disziplinen bieten. Für die Bildverarbeitung und Analyse wird OpenCV verwendet, eine portierbare und weit verbreitete open source Bibliothek. OpenCV bietet circa 500 Funktionen für die Aufnahme, Verarbeitung und Speicherung von Bilddaten. Zur Steuerung des Nao Roboters und als Beispiel für die Anbindung proprietärer Software wird das Aldebarans Nao SDK verwendet. Die Auswahl dieser Bibliotheken stützt sich auf die unter 1.2.2 Proof of Concept beschriebene Anwendung.

Neben der Nutzung von OpenCV als eine grundlegende Bildverarbeitungsbibliothek wird das Framework die Bibliothek POSIX Threads verwenden um Multithreading zu realisieren. Details zur Funktionsweise von OpenCV können [Pol10] und [BK08] entnommen werden. Eine kurze einführung zu POSIX Threads liefert [Bla10].

4.1. Abweichungen

Die Planung des Frameworks erfolgte im Wesentlichen ohne eine genaue Zielplattform zu fokussieren. Da das Framework in C++ implementiert wird, ergeben sich jedoch sprachspezifische Möglichkeiten, die im Entwurf des Frameworks nicht berücksichtigt wurden.

Obwohl *ImageProvider* und *CoordinateProvider* als Interface konzipiert wurden, werden beide als abstrakte Klassen implementiert. Vorteil dieser Vorgehensweise ist, dass der Programmcode für das Registrieren der Listener nur einmal an zentraler Stelle gepflegt werden muss. Dieser Wechsel von Interface zu abstrakter Klasse ist ohne weitere Einschränkungen nur möglich, da C++ die Mehrfachvererbung von Klassen unterstützt.

4.2. Bildverarbeitung

Für die Bildverarbeitung wurden einige grundlegende Bildquellen, Filter und Bildanalysemethoden implementiert. Diese verschiedenen Strategien für die Bildverarbeitung können dann entsprechend dem Strategy-Pattern ausgetauscht werden, ohne bestehenden Programmcode der einzelnen Verarbeitungsschritte ändern zu müssen.

4.2.1. Bildquellen

OpenCV bietet bereits viele Möglichkeiten Bilddaten aus verschiedenen Quellen für die weitere Verarbeitung nutzbar zu machen. So unterstützt OpenCV das Laden von Bilddateien und das Extrahieren von Einzelbildern aus Videodateien und von USB-Webcams, soweit diese vom Betriebssystem angesprochen werden können. OpenCV überführt alle geladenen Daten in das einheitliche Format *IplImage*.

Da alle Methoden von OpenCV mit Bilddaten in Form von *IplImage* umgehen können, wird es als Standardformat für die *ImageProvider* und *ImageListener* gewählt. Das *IplImage* an sich ist eine Universelle und von den Algorithmen unabhängige Struktur zur Speicherung von Bilddaten. Zudem werden einige Annahmen über die Bilddaten vereinbart:

- Der Wertebereich der einzelnen Pixel ist 0-255 (*unsigned char*) pro Kanal.
- Ist das Bild ein Grauwertbild, hat es einen Kanal.
- Ist das Bild ein Farbbild, hat es drei Kanäle. Die Farbinformationen werden in BGR kodiert.

Alle implementierten Algorithmen gehen von den genannten Annahmen aus. Algorithmen, die beispielsweise andere Farbkodierungen benötigen, berechnen diese intern. Diese Vorgehen trägt erheblich zur besseren Wartbarkeit und Übersichtlichkeit des Quellcodes bei.

Einzelbild

Für das Laden von Bilddaten bietet OpenCV die Funktion *cvLoadImage*. Für die Nutzung der Funktion wurde die Adapter-Klasse *SingleImage* geschrieben, die neben dem einmaligen Laden eines Bildes auch Bilddatenströme simulieren kann. Dies geschieht, indem die angemeldeten Listener in gewissen Intervallen über ein neues Bild benachrichtigt werden — das eigentliche Bild bleibt jedoch konstant.

Videodateien und USB-Kameras

Das Extrahieren von Einzelbildern aus einer Videodatei oder einer USB-Webcam kann mit einer *CvCapture*-Struktur und der Funktion *cvGrabFrame* realisiert werden. Wie auch für die Einzelbilder wurde sowohl für USB-Kameras als auch für Videodateien ein Adapter implementiert. Diese Struktur ermöglicht das kontinuierliche Auslesen des Bilddatenstroms von diesen Quellen mit angemeldeten Listenern. Die Klassen, die das Lesen dieser Bilddatenströme ermöglichen, heißen *Video* und *Webcam*. Wie auch das Einzelbild bieten *Video* und *Webcam* die Möglichkeit den Bilddatenstrom asynchron in einem zweiten Thread auszulesen und an die Listener zu übertragen.

Die Klasse *Video* ermöglicht es ein Objekt für eine Videodatei zu erstellen. Im Gegensatz dazu implementiert die Klasse *Webcam* ein Singeleton-Pattern, mit dem für jede physikalische Webcam nur ein Objekt existiert.

Datenquellen, die nicht von OpenCV unterstützt werden, wie zum Beispiel IP-Kameras, müssen zur Weiterverarbeitung mit OpenCV in das *IplImage* Format konvertiert werden. Ein Beispiel für eine solche Kamera ist die Kamera des Nao Academics Roboters, deren Anbindung unter 4.3.3 Kamera beschrieben wird.

4.2.2. Filter

Die implementierten Filter sind beispielhafte Möglichkeiten für die Verwendung der einzelnen Interfaces, die das Framework bietet. Das Klassenlayout des Frameworks bietet bezüglich eines Verarbeitungsschrittes in der Bildverarbeitung die Möglichkeit die Algorithmen untereinander auszutauschen. Diese Vorgehensweise entspricht dem Strategy-Pattern. Die Strategie, also die eigentlichen Bildverarbeitungsalgorithmen, können einfach ausgetauscht werden und sind nicht mehr abhängig von der konkreten Implementierung eines *Filters* oder *Analyzers*.

4.2.3. Weichzeichnen

Der Filter *Smooth* bietet die Möglichkeit den Bilddatenstrom weichzuzeichnen. Zu diesem Zweck bietet die Klasse verschiedene Weichzeichnungsalgorithmen, die durch einen Parameter des Konstruktors ausgewählt werden können. Nachdem der *Smooth-Filter* über ein neues Bild benachrichtigt, wird dieses mit dem gewählten Algorithmus weichgezeichnet und dann alle angemeldeten *ImageListener* benachtichtigt.

Intern nutzt die Klasse die Funktion *cvSmooth* von OpenCV, die bereits die verschiedenen Weichzeichnungsalgorithmen beherrscht. Die Klasse *Smooth* ist somit nichts anderes als ein Adapter für die OpenCV Methode, der es ermöglicht die Funktionalität aus OpenCV mit dem Framework zu nutzen.

4.2.4. Kamerakalibrierung und geometrische Korrektur

Um Verzerrungen durch die Kameraoptik auszugleichen, liefert das Framework den *Filter LineCalibration*. Diese Klasse nutzt die Koordinaten eines Trapezes und bildet diese auf ein Quadrat ab. Mit diesem Verfahren werden alle Pixel des Eingabebildes abgebildet, sodass ein entzerrtes Bild entsteht.

OpenCV liefert die Methode *cvAffineTransform*, die Bildgeometrien global verändern kann. Die Methode benötigt eine Abbildungsmatrix, die ein Eingabeviereck E auf ein Ausgabeviereck A abbildet. Das Ein- und Ausgabeviereck berechnet *LineCalibration* aus den übergebenen Koordinaten $T = (a_x, a_y, b_x, b_y, c_x, c_y, d_x, d_y)$ *mit* $\measuredangle(\overline{ab}) \approx 90°, \measuredangle(\overline{cd}) \approx 90°$:

$$
\begin{aligned}
E &= T \\
A_{a_x} = A_{b_x} &= \max(T_{a_x}, T_{b_x}) \\
A_{c_x} = A_{d_x} &= \min(T_{c_x}, T_{d_x}) \\
A_{a_y} = T_{a_y},\ A_{b_y} &= T_{b_y} \\
A_{c_y} = T_{b_y},\ A_{d_y} &= T_{d_y}
\end{aligned}
$$

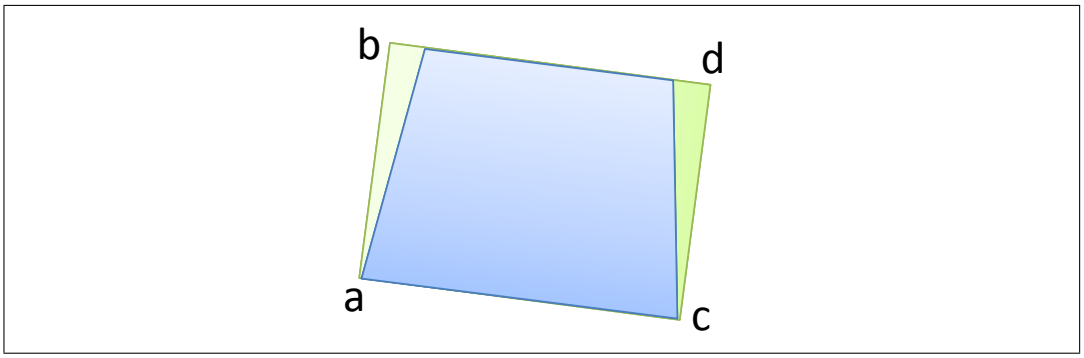


Abbildung 4.1.: Schematische Darstellung der Kamerakalibrierung mit *LineCalibration*. Das Eingabeviereck ist blau, das Ausgabeviereck grün dargestellt.

Abbildung 4.1 verdeutlicht die Berechnung von A und E aus T. Ein Ergebnis dieser Kalibrierungsmethode ist in Abbildung 4.2 zu sehen. Insbesondere perspektivische Verzerrung

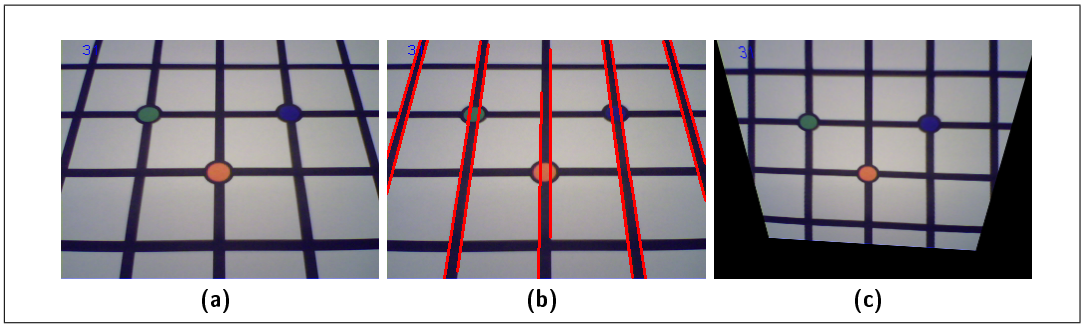


Abbildung 4.2.: Beispiel für die Anwendung der Kamerakalibrierung, wie sie die Klasse *LineCalibration* bietet.: (a) Das Eingabebild; (b) durchsuchen des Eingabebilds nach Linien; (c) Ausgabebild nach der Kalibrierung

lässt sich mit dieser Methode ausgleichen — es entsteht eine Art Vogelperspektive von den aufgenommenen Objekten.

Die Klasse *LineCalibration* ist ein Zusammenschluss von mehreren Funktionen, die auf Basis der grundlegenden Daten nicht nur die eigentliche Bildtransformation, sondern auch die Aufbereitung der Eingabedaten übernimmt.

Vorlagenvergleich

OpenCV bietet für diesen Zweck die Funktion *cvMatchTemplate*, die genutzt werden kann, um Objekte in einem Bild zu finden. Der Vorlagenvergleich positioniert die Vorlage nacheinander über jedem Pixel des Originalbilds und berechnet den Unterschied zu den darunter liegenden Pixeln im Originalbild. Je kleiner dieser Unterschied ist, umso höher ist die Wahrscheinlichkeit, dass das Bild an dieser Stelle im Bild liegt.

Mit diesem Verfahren entsteht eine Wahrscheinlichkeitsmatrix, die zu jeder Position der Vorlage die berechnete Wahrscheinlichkeit enthält. Die so erstellte Matrix kann wiederum als Bild angesehen werden — auf einem solchen Bild lässt sich dann durch Bestimmung des Maximums eine mögliche Position des Objekts finden. Da der Vorlagenvergleich so implementiert wurde, dass quasi Bilddaten berechnet werden, passt der Algorithmus nicht mehr zum Datentyp *Analyzer*, sodass die Klasse *MatchTemplate* als *Filter* implementiert wurde.

4.2.5. Bildanalyse

Histogrammvergleich

Die Klasse *HistogramComparator* führt einen Histogrammvergleich einer Vorlage mit dem Bilddatenstrom durch. Es können Histogrammvergleiche auf Grundlage der verschiedenen Darstellungsformen von Bilddaten geschehen:

Einkanal: Grauwerte Dieser Histogrammtyp setzt voraus, dass die Daten als Grauwertbild vorliegen. Das Histogramm wird dann als eindimensionale Häufigkeitsverteilung über dem Kanal berechnet. Als Ergebnis gibt der *HistogramComparator* dann einen skalaren Wert.

Dreikanal: RGB Für den Vergleich von RGB-Histogrammen müssen die Daten im unter 4.2.1 Bildquellen beschriebenen BGR-Format vorliegen. Die Klasse berechnet dann

Analog zum Grauwertbild für jeden Kanal ein Histogramm. Die Kanäle werden dann unabhängig voneinander verglichen. Als Ergebnis gibt der *HistogramComparator* drei Vergleichswerte, einen pro Farbkanal.

Dreikanal: RGB kombiniert Anders als im vorherigen Fall berechnet der *HistogramComparator* für das kombinierte Histogramm ein dreidimensionales Histogramm der Bilddaten. Mit dieser Methode lassen sich die einzelnen Farbkanäle in Abhängigkeit voneinander betrachten. Das Ergebnis dieser Operation ist dann wieder nur ein skalarer Wert für das gesamte Histogramm.

Dreikanal: HSV Für die Berechnung des HSV-Vergleichs werden die Bilddaten zunächst in das HSV-Format konvertiert. Der Vergleich der Histogramme erfolgt dann wie im Fall 'RGB' kanalweise. Analog gibt es drei Rückgabewerte.

Zweikanal: HS kombiniert Wie auch für das HSV-Histogramm müssen die Bilddaten für das HS-Histogramm zunächst in das HSV-Format konvertiert werden. Dann wird ein zweidimensionales Histogramm erstellt, wobei der V-Kanal nicht berücksichtigt wird. Dieses Verfahren bietet sich an um Störungen durch wechselnde Beleuchtung zu umgehen, da diese im Wesentlichen den V-Kanal beeinflussen. Für das kombinierte HS-Histogramm liefert der *HistogramComparator* wieder nur ein skalares Ergebnis.

Als Vergleichsmethode nutzt *HistogramComparator* die Korrelation zwischen den Histogrammen. Ein guter Vergleichswert ist demnach nahe an 1.0, eine zufällige Korrelation ist ca. 0.7. Der schlechtestmögliche Vergleichswert ist -1.0 für ein inverses Histogramm. Dieser kann aber wiederum auf eine Verschiebung des Histogramms um eine Konstante hinweisen. Weitere Informationen zu dem Vergleichsverfahren liefert [BK08, S. 201f].

Um den Histogrammvergleich durchzuführen fasst die Klasse *HistogramComparator* die Vorverarbeitung wie das Überführen in das entsprechende Format, das Berechnen des Histogramms und das eigentliche vergleichen der Histogramme als eine funktionale Einheit zusammen. Die Klasse geht dabei von den unter 4.2.1 Bildquellen genannten Rahmenbedingungen aus und konvertiert das Bild entsprechend in das benötigte Format zur Berechnung des Histogramms. Die Klasse *HistogramComparator* nutzt somit auch das vorgestellte Adapter-Pattern um Funktionen aus OpenCV mit dem Framework nutzbar zu machen.

Segmentierung

Eine weitere Analysemethode, die im Framework implementiert wurde, ist die Pyramidensegmentierung. Das Framework bietet dafür den *Analyzer PyramidSegmentation*. Neben den Koordinaten der gefundenen Segmente wird das Bild zurückgegeben, dass durch die Nachbarschaftsoperationen der Pyramidensegmentierung erstellt wurde.

Intern nutzt die Pyramidensegmentierung die Funktion *cvPyradmidSegementation* aus der Bibliothek OpenCV. Diese liefert neben den Konturen auch das Bild, das aus den Operationen der Pyramidensegmentierung entsteht.

Neben der eigentlichen Segmentierung ist auch die Anzahl der Ebenen, für die die Segmentierung durchgeführt wird, von entscheidender Bedeutung. Da das Bild in jeder Ebene auf 50% der Bildgröße herunter skaliert wird, muss die Seitenlänge des Bildes entsprechend oft durch zwei teilbar sein, damit sie von der Funktion *cvPyramidSegmentation* verarbeitet werden kann. Die Klasse *PyramidSegmentation* übernimmt eine entsprechende Überprüfung und vergrößert das Bild gegebenenfalls, indem ein schwarzer Rand hinzugefügt wird.

Aufgrund von Limitierungen der Funktion *cvPyramidSegmentation* gibt die Klasse *PyramidSegementation* nicht die Konturen, sondern die Bounding Boxes der Konturen zurück.

Konturen

Die Behandlung von Konturen im Framework wurde in zwei Klassen aufgeteilt. Die Klasse *Lines* sucht gerade Linien in den Bilddaten, die Klasse *Conturs* sucht Polygone. In beiden Fällen werden durch das Adapter-Pattern die verschiedenen Abläufe aus OpenCV im Framework nutzbar gemacht.

Die Klasse *Lines* nutzt die Funktion *cvHoughLines* zum Auffinden von geraden Linien in den Bilddaten. Dabei kann die Klasse zwischen Horizontalen (mit absolutem Winkel $< 45°$) und Vertikalen (mit absolutem Winkel $> 45°$) unterscheiden. Neben der Lage der Linie filtert *Lines* die gefundenen Linien zusätzlich nach der Länge — zu kurze Linien werden nicht beachtet. Die nach der Filterung übrig gebliebenen Linien werden als Tupel von Anfangspunkt a und Endpunkt e im Format (a_x, a_y, e_x, e_y) zurückgegeben.

Zum Finden von Konturen in Bildern kann die Klasse *Conturs* verwendet werden. Diese Klasse nutzt die OpenCV Funktion *cvFindContours* zum Finden der Konturen in den Bilddaten. Neben den Konturen als Polygonzug kann *Contours* auch die Bounding Boxes der Konturen als Ergebnis weitergeben. Bounding Boxes werden jedoch behandelt wie Polygone aus vier Punkten. *Contours* filtert die gefundenen Konturen dann nach den im Konstruktor angegebenen minimalen und maximalen Größen der Bounding Boxen der Konturen. Die Weitergabe der Konturen als Polygone erfolgt so, dass ein Polygon in einem Vektor linearisiert ist. Für ein Polygon p aus n Punkten ergibt sich folgende Form: $p = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$.

Extremwerte

Für die Analyse von Merkmalsbildern ist es häufig nötig maximale und/oder minimale Bildbereiche zu finden. Der *Analyzer MinMax* sucht solche lokalen Extrempunkte in den Bilddaten. Hat der *Analyzer* einen Extrempunkt gefunden, wird der Bildbereich rund um diesen maskiert — weitere Extrempunkte müssen also einen Mindestabstand zu den vorherigen haben. Die Klasse sucht nach einer vorgegebenen Anzahl von Minima, Maxima oder beidem und gibt diese an die registrierten *CoordinateListener* weiter.

Über den Konstruktor lassen sich die Parameter von *MinMax*, die Anzahl und Art der zu findenden Extrempunkte und die um die gefundenen Extrempunkte zu maskierenden Bereiche einstellen. Einen gefundenen Extrempunkt e gibt *MinMax* in der Form $e = (x, y, v)$, wobei x, y den Koordinaten des Punktes und v dem Wert der Bilddaten an (x, y) entspricht, weiter.

4.2.6. Visualisierung

Bildschirmanzeige

OpenCV bietet die Möglichkeit mittels einer einfachen GUI-Implementierung Bilder auf dem Bildschirm anzuzeigen. Diese Methode eignet sich allerdings nicht um die Bilder innerhalb einer bestehenden Benutzeroberfläche anzuzeigen. Um Bilder in einer bestehenden Benutzeroberfläche anzeigen zu können muss ein Steuerelement für ein UI-Framework geschrieben werden. Zu diesem Zweck liefert das Framework ein *QWidget*, das das *ImageListener*-Interface implementiert und das Bild auf der Zeichenfläche des Steuerelements anzeigt. Dieses *QWidget* kann dann in Benutzeroberflächen, die mit dem Qt Framework implementiert wurden, eingebettet werden. Ein Beispiel für eine solche Anwendung ist unter Abbildung 4.3 zu sehen.

Wird die Methode *refreshImage* des Steuerelements aufgerufen, wird das von der Bildquelle übergebene Bild in das Qt eigene Datenformat *QImage* konvertiert. Dann wird der Bereich des Steuerelements in der Qt-Ereignisverarbeitung für ungültig erklärt und Qt zeichnet das Steuerelement mit dem neuen Bild.

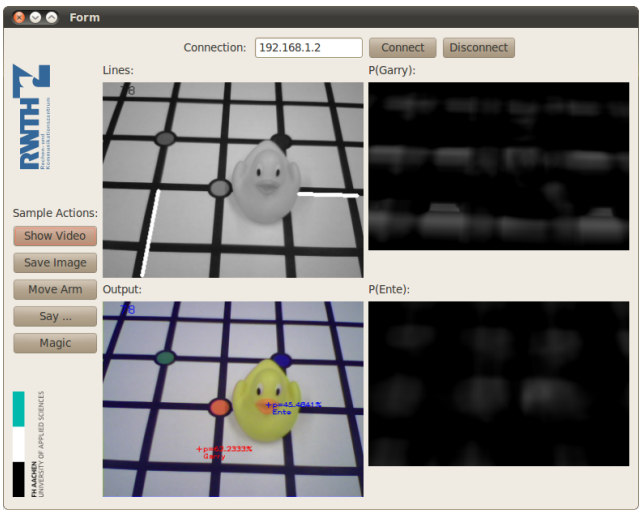


Abbildung 4.3.: Darstellung von Bilddaten, die mit dem Framework verarbeitet werden in einem Qt Fenster.

Visualisierung der Ergebnisse

Um die Ergebnisse der Bildanalyse auf dem Bildschirm auszugeben, wurde die Klasse *Painter* implementiert. Diese Klasse ist sowohl ein *ImageListener* als auch ein *CoordinateListener*. Sobald Bild und Koordinaten aktualisiert wurden, zeichnet *Painter* die Ergebnisse farbig in den Bilddaten ein und benachrichtigt die angemeldeten *ImageListener* über ein neues Bild.

Die Implementierung erkennt verschiedene Objekte anhand der Anzahl ihrer Koordinaten.

Punkte Eine einzige Koordinate wird als Punkt interpretiert, zu der Position (x, y) kann noch ein Wert angegeben werden, sodass ein Tripel (x, y, v) entsteht. Einen solchen Punkt zeichnet die Klasse als Kreuz an der Position.

Linien Linien werden aus zwei Koordinaten, dem Anfangs- und Endpunkt der Linie, gezeichnet. Das Koordinatentupel muss demnach die Form (a_x, a_y, e_x, e_y) haben.

Polygone Alle Koordinatentupel, die mehr als vier Einträge haben, werden als Polygon interpretiert. Für ein Tupel der Form $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$ zeichnet die Klasse die Verbindungslinien zwischen den aufeinander folgenden Punkten sowie die Verbindung zwischen dem letzten und dem ersten Punkt.

4.3. Robotersteuerung

Die Robotersteuerung wird über das Interface *Affector* realisiert. So können für die zu steuernden Teile des Roboters Adapter-Klassen geschrieben werden, die Kommandos in Funktionsaufrufe für die Roboterhardware umsetzen. Im Hinblick auf die unter 1.2.2 gezeigte

Problemstellung wurden Klassen für die Steuerung des Nao Academics Roboters der Firma Aldebaran Robotics implementiert.

4.3.1. Plattform: Nao Academics

Abbildung 4.4.: Nao Roboter der Firma Aldebaran Robotics

Der Hersteller Aldebaran Robotics beschreibt den Nao Academics Roboter auf seiner Homepage wie folgt:

> „NAO is the *most used humanoid robot* for academic purposes worldwide. Aldebaran Robotics has chosen to make NAO's technology available to any higher education program. Fully interactive, fun and permanently evolving, NAO is a *standard* platform for teaching students of all levels.
>
> Complete with a *user-friendly programming environment*, students and teachers can use at any programming level. It is really easy to start working on NAO, and our educational kits will get you teaching with NAO in no time!
>
> From simple visual programming to elaborate embedded modules, the *versatility* of NAO and his programming environment enables users to explore a wide variety of subjects at whatever level of programming complexity and experience."
>
> — *Aldebaran Robotics* [Ald10a]

4.3.2. Nao SDK

Zur Programmierung des Nao Academics Roboters liefert der Hersteller eine Programmbibliothek. Diese bietet zwei Möglichkeiten der Kommunikation mit dem Roboter:

Lokales Programm Das Programm, das den Roboter steuern soll, wird direkt auf dem Roboter ausgeführt. Die Programmbibliothek gibt die Befehle direkt an die Roboterhardware weiter.

Entferntes Programm Das Programm wird entfernt auf einem anderen Rechner ausgeführt. Die Kommunikation und die Steuerung der Roboterhardware erfolgt über ein TCP/IP-Netzwerk mittels SOAP.

Die Programmbibliothek kann die beiden Betriebsmodi selbstständig erkennen, sodass prinzipiell ein und dasselbe Programm auf beide Arten ausgeführt werden kann. Ein Vorteil der entfernten Ausführung ist, dass beispielsweise während der Entwicklung ein grafischer Debugger genutzt werden kann. Zudem besteht die Möglichkeit den Status des Roboters auf einem Bildschirm darzustellen — diese Möglichkeiten entfallen, da der Roboter keine Möglichkeit zum Anschluss von Anzeigemedien bietet. Dennoch bleibt zu beachten, dass die entfernte Ausführung der Programme diese um ein vielfaches verlangsamt, da alle Steuerkommandos über das Netzwerk gesendet werden müssen. Insbesondere für die Bildverarbeitung sind die Wartezeiten für Kamerabilder nicht unerheblich.

4.3.3. Implementierungen für den Nao Roboter

Zur Anbindung an das Framework wurden für die Klassen und Funktionen aus dem Nao SDK Adapter-Klassen geschrieben. Diese ermöglichen es die Kamerabilder vom Roboter in das *IplImage*-Format zu konvertieren und weiterzuverarbeiten. Zudem werden Affektoren implementiert, mit denen am Ende der Bildverarbeitungspipeline die Steuerung der Gliedmaßen des Roboters ermöglicht wird.

Kamera

Der Nao Academics Roboter verfügt über insgesamt zwei Kameras. Diese sind übereinander am Kopf angeordnet. Abbildung 4.5 zeigt eine schematische Darstellung der Kamerapositionen und der Blickwinkel. Um die Kameras des Nao Academics Roboters nutzbar zu machen

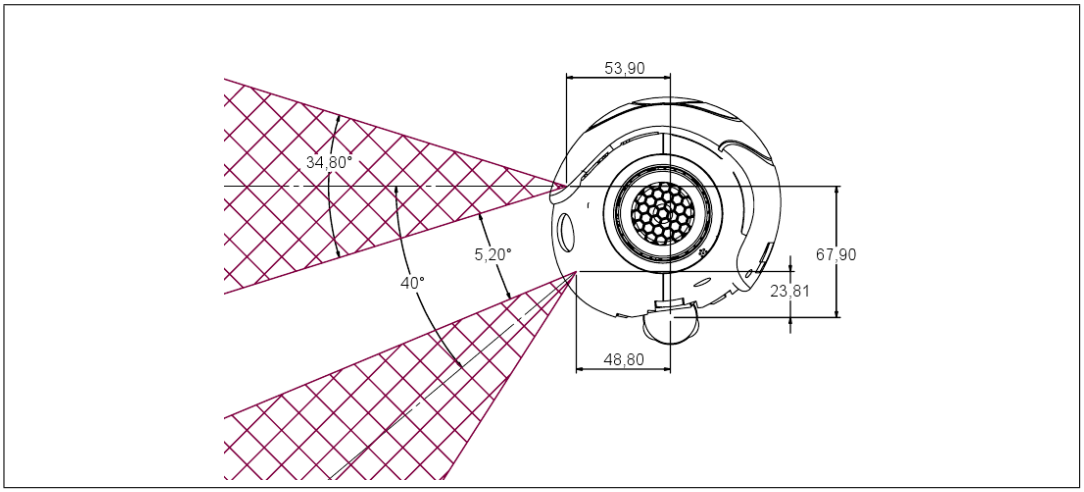


Abbildung 4.5.: Schematische Darstellung des Kopfes des Nao Academics Roboters. Die schraffierten Flächen zeigen die Blickwinkel der beiden Kameras.

wurde die Klasse *nao::Video* implementiert, die das *ImageProvider*-Interface implementiert. Aufgrund technischer Beschränkungen des Roboters ist es nur möglich das Kamerabild einer der beiden Kameras auszulesen, sodass, wie auch bei den USB-Webcams in 4.2.1 Videodateien und USB-Kameras beschrieben, für den Zugriff auf das Objekt ein Singeleton-Pattern realisiert wurde.

Die Klasse *nao::Video* konvertiert das Bild von einem Aldebaran-spezifischen Format in das *IplImage*-Format. Das Bild kann dann in der Bildverarbeitungspipeline analysiert werden. Die Klasse bietet zudem die Möglichkeit den Bilddatenstrom vom Roboter asynchron, in einem anderen Thread, an die registrierten *ImageListener* zu übertragen.

Steuerung der Gliedmassen

Für die Ausgabe der Algorithmen müssen die Gliedmaßen des Nao Academics Roboters angesteuert werden. Dieser Prozess kann im Wesentlichen in zwei Teilschritte gegliedert werden:

Konvertierung der Koordinaten Die Ergebnisse der Bildverarbeitungsalgorithmen sind Koordinaten auf dem Bild, also sind die gefundenen Koordinaten nur die Projektion auf die Bildebene. Diese müssen in Koordinaten konvertiert werden, die im Raum liegen, die der Roboter beeinflussen kann. Für diese Konvertierung der Koordinaten muss ein *CoordinateConverter* implementiert werden, der speziell an die Szene auf dem Bild angepasst ist. 4.5 Fallstudie: Memoryspiel zeigt eine mögliche Vorgehensweise bei der Erstellung eines solchen Konverters.

Bewegung des Affektors Nachdem die Zielkoordinaten umgesetzt wurden, muss der Affektor zu der Zielposition bewegt werden. Diese Funktionalität erfordert genaue Kenntnis des Roboters und wurde im Falle des Nao Academics Roboters vom Hersteller implementiert.

Der Nao Academics Roboter bietet im Wesentlichen Affektoren für die Arme, die Beine und den Kopf. Für den Zugriff auf diese mit den Möglichkeiten des Frameworks wurden Adapter-Klassen geschrieben, die das Interface *Affector* implementieren. Der Zugriff auf diese Klassen ist ebenfalls im Singleton-Pattern organisiert. Die Affektoren bieten die Möglichkeit die Gliedmaßen des Roboters relativ zu seinem Mittelpunkt zu bewegen. Abbildung 4.6 zeigt die Lage des Nullpunkts und der Achsen dieses Koordinatensystems.

4.4. Beispielimplementierung

Als Beispiel für eine Implementierung eines Programms, dass auf dem Framework basiert, wir hier ein Stück Programmcode vorgestellt, dass analog zu dem unter 3.7 vorgestellten Ablauf einen Histogrammvergleich durchführt. Listing 4.1 zeigt den Quellcode des Beispiels.

Zunächst einige Bilder aus Bilddateien geladen: Eine Referenz und vier, die mit der Referenz verglichen werden. Nachdem die weiteren Verarbeitungsschritte: Weichzeichnen, Histogrammvergleich und Ausgabe initialisiert wurden, wird die Bildverarbeitungspipeline aufgebaut. Dafür werden die einzelnen Schritte nach dem Observer-Pattern verknüpft.

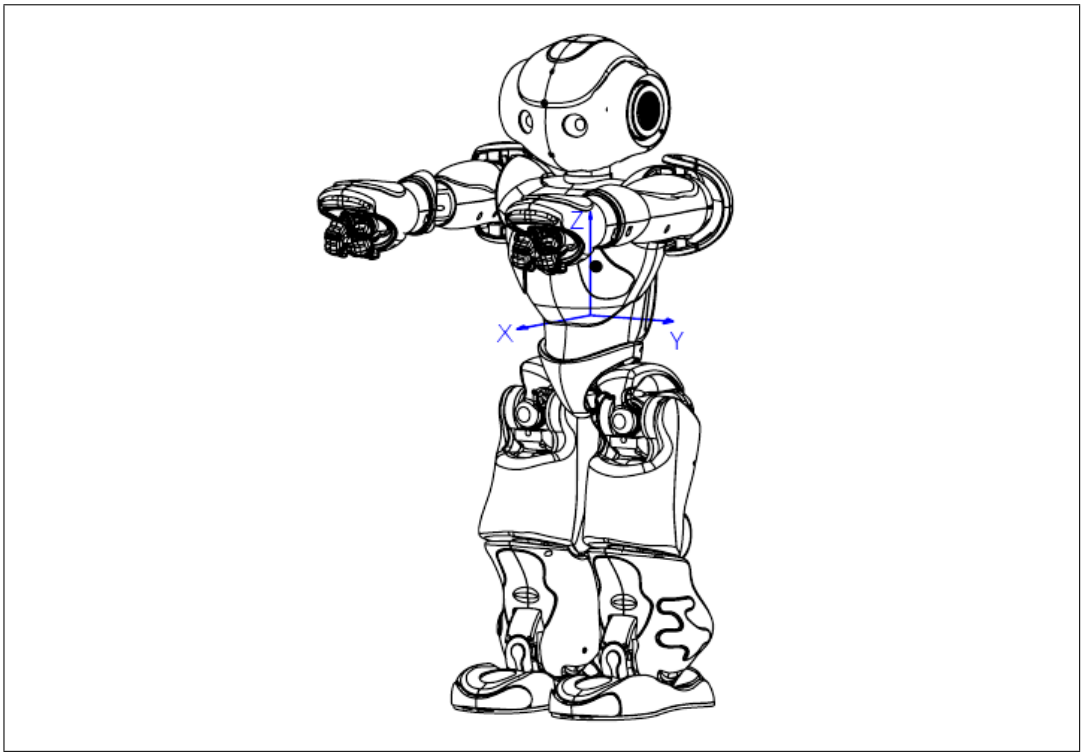


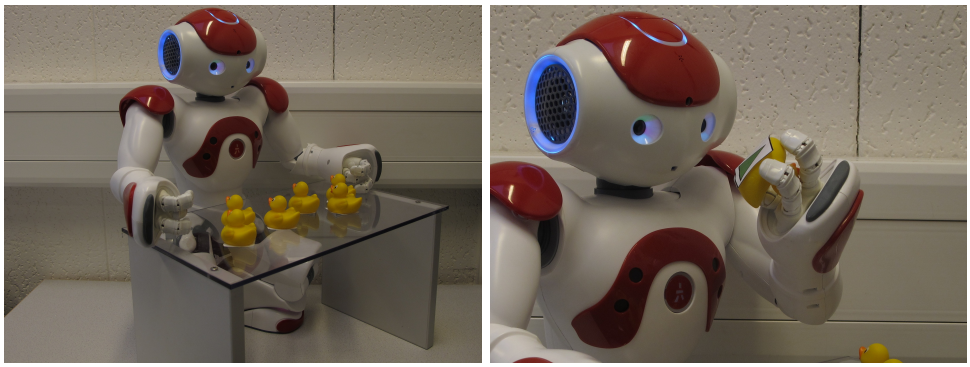
Abbildung 4.6.: Schematische Darstellung des Nao Academics Roboters. Die Pfeile zeigen das Koordinatensystem relativ zur Robotermitte, in dem die implementierten Affektoren zur Steuerung der Gliedmaßen bewegt werden können.

```
1  int main(int argc, char* argv)
2  {
3    //Bilder initialisieren
4    cv4cpp::devices::SingleImage reference("../Testing/images/shapes/circle_blue.png");
5    cv4cpp::devices::SingleImage compare1("../Testing/images/shapes_bad/circle_blue.png");
6    cv4cpp::devices::SingleImage compare2("../Testing/images/shapes/star_yellow.png");
7    cv4cpp::devices::SingleImage compare3("../Testing/images/shapes/square_red.png");
8    cv4cpp::devices::SingleImage compare4("../Testing/images/shapes/triangle_green.png");
9
10   //Verarbeitung initialisieren
11   cv4cpp::output::Console stdout;
12   cv4cpp::filters::Smooth smooth(cv4cpp::filters::Smooth::SMOOTH_GAUSSIAN, 5);
13   cv4cpp::analyzers::HistogramComparator analyzer(smooth.getCurrentImage(),
         cv4cpp::analyzers::HistogramComparator::HIST_RGB_COMBINE);
14
15   //Bildverarbeitungspipeline aufbauen
16   reference.subscribe(&smooth);
17   smooth.subscribe(&analyzer);
18   analyzer.subscribe(&stdout);
19
20   //Eingabebilder nacheinander in die Pipeline geben
21   compare1.subscribe(&smooth);
22   compare2.subscribe(&smooth);
23   compare3.subscribe(&smooth);
24   compare4.subscribe(&smooth);
25
26   return 0;
27 }
```

Listing 4.1: Beispielprogramm, dass die Histogramme mehrerer Bilder vergleicht und die Vergleichswerte auf der Konsole ausgibt.

4.5. Fallstudie: Memoryspiel

Neben den recht allgemein gehaltenen Implementierungen für die Bildverarbeitung und Steuerung des Roboters wurden auch Methoden implementiert, die speziell für die Realisierung des unter 1.2.2 Proof of Concept beschriebenen Ziels „Memoryspielender Nao“ zugeschnitten sind. Zunächst soll jedoch kurz der Versuchsaufbau für das Memoryspiel beschrieben werden:



(a) (b)

Abbildung 4.7.: Versuchsaufbau Memoryspiel. Der Nao Academics Roboter sitzt an einem Tisch. Damit der Roboter die Karten greifen kann sind die Memorykarten unter Gummienten geklebt: (a) Der Aufbau des Roboters am Tisch; (b) Der Roboter greift eine Memorykarte und identifiziert das Bild unter der Karte.

Der Roboter sitzt vor einem ca. 20cm hohen Tisch. Der Tisch hat eine Fläche, die ungefähr der eines DIN A4 Blattes entspricht. Auf der Tischplatte werden die Memorykarten verteilt, sodass alle im Interaktionsradius des Roboters liegen. Damit der Roboter die Memorykarten greifen kann, wurden diese an Gummienten geklebt. Der gezeigte Aufbau kann so auf einem Tisch platziert werden und ermöglicht das Spielen mit einem menschlichen Mitspieler.

4.5.1. Spielphasen

Das Memoryspiel gliedert sich in drei Spielphasen, das Erkennen des Spielfeldes, das Greifen der Spielkarten und das Erkennen und Vergleichen der einzelnen Spielkarten.

Spielfelderkennung

Zu Beginn des Spiels müssen die Karten auf dem Tisch erkannt werden. Dies geschieht mit der Kamera des Nao Academics Roboters. Die Kameras des Roboters sind jedoch eher für die Aufnahme von Objekten ausgelegt, die weiter vom Roboter entfernt sind. Abbildung 4.8a zeigt das Bild der Kamera. Das ganze Spielfeld kann also nicht mit einem einzelnen Kamerabild erkannt werden.

Um dennoch Objekte auf dem ganzen Spielfeld erkennen zu können, werden zwei Bilder mit verschiedenen Positionen des Kopfes gemacht und diese dann zusammengesetzt. Abbildung 4.8b zeigt eine solche Kameraaufnahme. Zur Erstellung solcher Bilder wurde eine Klasse

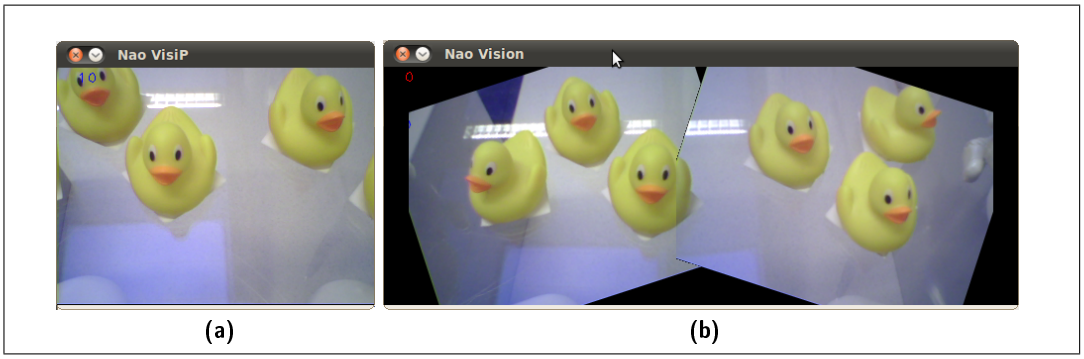


Abbildung 4.8.: Kamerabilder des Nao Academics Roboters während des Memoryspiels: (a) Blickwinkel des Roboters auf den Tisch mit den Memorykarten; (b) Zwei Bilder werden zusammengesetzt um einen größeren Bildbereich abzudecken.

nao::PanoramaVideo implementiert, die sowohl die Bewegung des Kopfes und die Aufnahme der Bilder koordiniert als auch die beiden aufgenommenen Bilder zu einem zusammenfügt.

Karten-Lokalisierung und Greifen

Auf einem Bild, das wie im vorherigen Abschnitt beschrieben erstellt wurde, können nun die Memorykarten lokalisiert werden. Der Arm des Roboters muss dann so positioniert werden, dass er die Ente greifen kann. Wie bereits unter 4.3.3 Steuerung der Gliedmaßen beschrieben, müssen dafür die Koordinaten auf dem Bild in Koordinaten relativ zum Roboter umgerechnet werden. Zu diesem Zweck wurde eine Klasse *nao::PanoramaConverter* geschrieben, die diese Konvertierung übernimmt.

Die Klasse *nao::PanoramaConverter* nutzt eine einfache, aber dennoch effektive Methode für die Umrechnung von Koordinaten: Zunächst werden die Parameter der Postion des Roboters festgehalten. Dies sind vor allem: Winkel der Beine, Neigung und Drehung des Kopfes. Aus dieser Postion werden dann verschiedene Aufnahmen eines Objekts bekannter Größe und Position gemacht. Beispielhaft wurde dafür ein Lineal verwendet, da eine Aufnahme direkt mehrere Messpunkte liefert.

Zusammen mit der Höhe des Tisches, auf dem die Objekte liegen, kann dann aus den gemessenen Punkten ein überbestimmtes Gleichungssysteme aufgestellt werden. Aus diesen lassen sich dann Koeffizienten für Gleichungen berechnen, mit denen die Bildpixel in Entfernungen relativ zum Roboter umgerechnet werden können. Abbildung 4.9 zeigt beispielhaft Bilder, aus denen solche Datenpunkte gewonnen werden können.

Kartenerkennung und Zuordnung

Nachdem die Memorykarte auf dem Spielfeld lokalisiert wurde, kann diese vom Roboter gegriffen und vor die Kamera geführt werden. Abbildung 4.7b zeigt den Roboter mit der Memorykarte in der Hand. Das Bild auf der Unterseite der Memorykarte kann dann von der Steuersoftware verarbeitet werden, sodass eine dem Spielverlauf entsprechende Reaktion erfolgt.

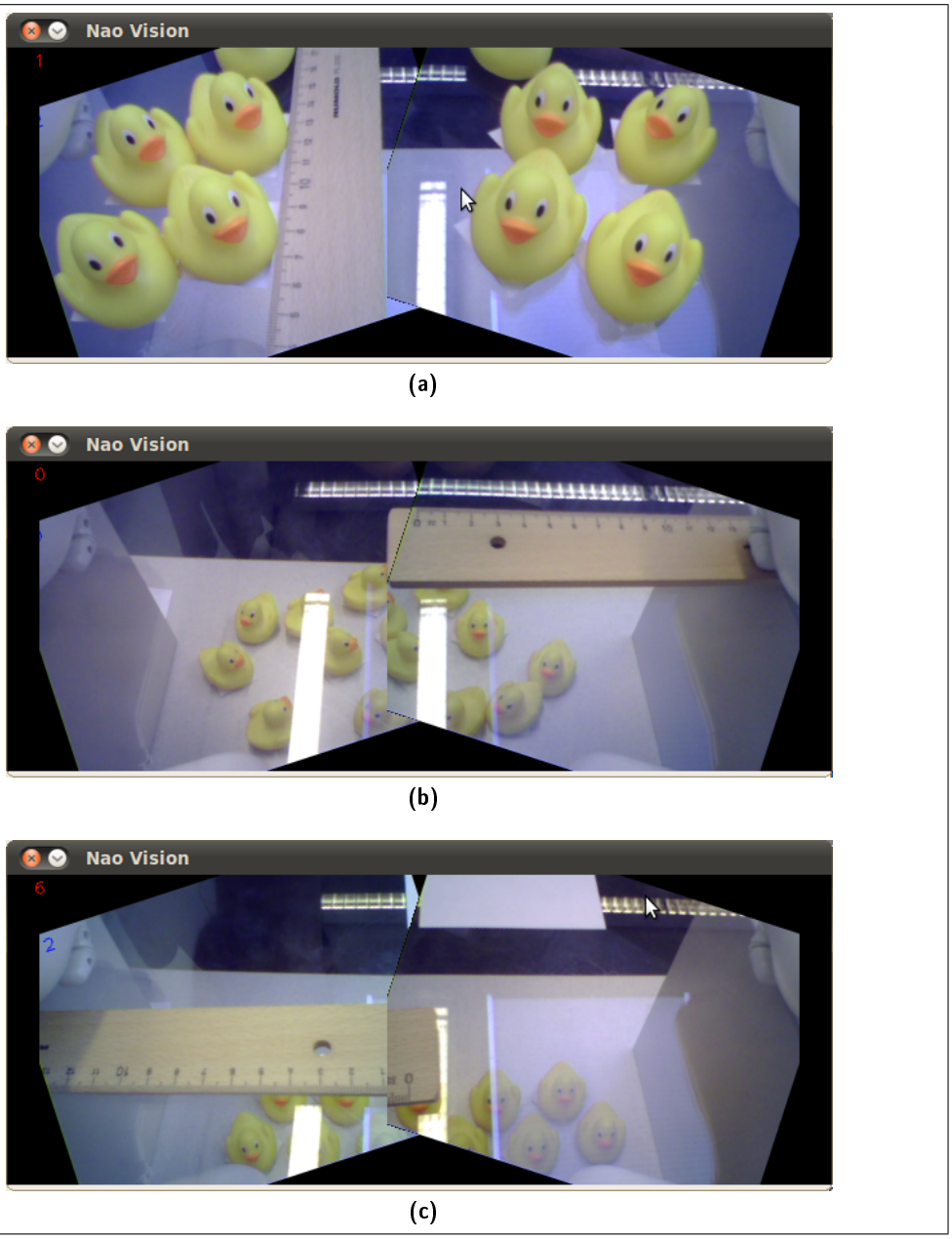


Abbildung 4.9.: Kamerabilder des Nao Academics Roboters zur Kalibrierung der Kamera: (a) Die vertikale Bildachse wird möglichst mittig vermessen; (b), (c) Die horizontale Bildachse wird mehrfach an verschiedenen Positionen vermessen.

5. Software Tests

Da das Framework ein Bestandteil von anderen Programmen sein wird, ist es besonders wichtig, dass die implementierten Funktionen systematisch getestet werden. Fehler in der Implementierung des Frameworks lassen sich bei der späteren Nutzung in anderen Programmen möglicherweise nicht mehr erkennen. Die Klassen werden in zwei Schritten getestet:

Komponenten Tests Zunächst werden die einzelnen Klassen alleine mit synthetischen Eingabedaten, deren Ergebnisse bekannt sind, getestet. Da die Ergebnisse bekannt sind, können diese automatisiert ausgewertet werden.

Systemtests Anschließend werden mehrere Klassen zusammen in realitätsnahen Anwendungsfällen getestet. Das Ergebnis dieser Tests muss in der Regel manuell kontrolliert werden.

5.1. Komponenten Tests

Für den Test der einzelnen Komponenten kommt das QtTest Framework zum Einsatz. Dieses Framework bietet eine Möglichkeit sogenannte Unit Tests für C++ Bibliotheken zu erstellen und durchzuführen. Die Ergebnisse des Unit Testings können dann als XML-Datei oder auf dem Bildschirm ausgegeben werden. Abbildung 5.1 zeigt die grafische Ausgabe eines solchen Testlaufs.

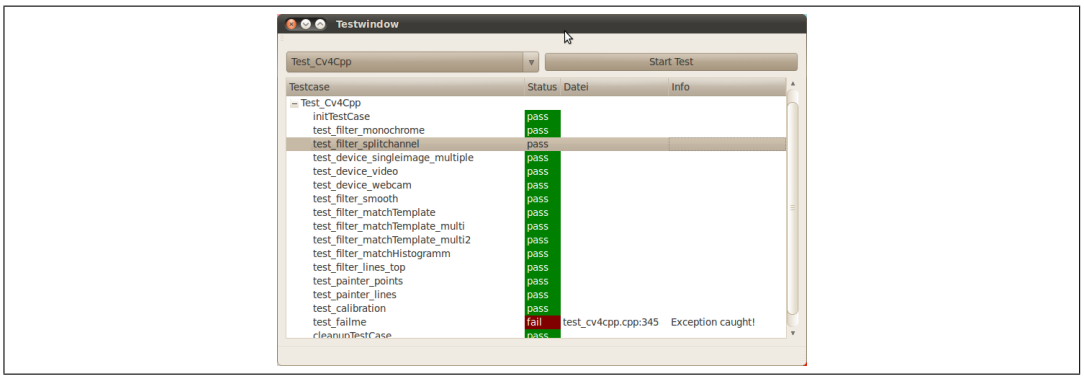


Abbildung 5.1.: Screenshot der grafischen Ausgabe der Komponenten Tests.

Während der Implementierung der einzelnen Klassen, wurde für jede dieser Klassen mindestens eine Methode geschrieben, die die Funktionalität dieser Klasse prüft. Werden die Unit-Tests regelmäßig durchgeführt kann beispielsweise überprüft werden, ob eine nachträgliche Änderung am Quellcode einer Klasse das Verhalten der Klasse verändert hat.

5.2. System Tests

Neben den Tests der einzelnen isolierten Komponenten wurden insbesondere hinsichtlich der geplanten Umsetzung des „Memoryspielenden Nao“ Software Tests implementiert, die als ein kleines Programm aus mehreren Klassen des Frameworks ausgeführt werden können. Diese Tests bieten vor allem die Möglichkeit die einzelnen Klassen auf Threadsicherheit und Toleranz für die Verzögerung durch Warten auf Benutzerinteraktion oder Netzwerkübertragung zu überprüfen. Zu diesem Zweck wurden einzelne Programmteile des Memoryspiels umgesetzt:

Identifikation und Greifen von Memorykarten Dieser Testfall besteht darin, dass der Nao Academics Roboter an einem wie unter 4.5 Fallstudie: Memoryspiel beschriebenen Tisch mit Memorykarten sitzt. Nachdem der Roboter das Bild analysiert hat, muss er die als Memorykarten identifizierten Objekte greifen und hochheben.

Vergleich von Karten Der Roboter startet in der Situation wie in Abbildung 4.7b zu sehen. Der Roboter muss das Bild unter der Memorykarte erkennen, mit einer Menge von vorgegebenen Bildern vergleichen und den besten Treffer ausgeben.

Neben dem reinen Test der Software helfen die implementierten Testszenarien auch bei der späteren Umsetzung der Anwendung „Memoryspielender Nao“.

6. Fazit und Ausblick

Das im Rahmen dieser Arbeit entwickelte Framework bietet eine Möglichkeit für die bildbasierte Steuerung von Robotern. Dafür wurden sowohl Aspekte der Bildverarbeitung als auch Aspekte der Steuerung von Robotern berücksichtigt. Das Framework liefert durch eindeutige Schnittstellen zudem die Möglichkeit einzelne Bestandteile wie Algorithmen, Bildquellen oder Roboterhardware auszutauschen. Zudem können durch Implementierung der Interfaces weitere Algorithmen, Bildquellen und Steuerungssoftware für andere Roboter hinzugefügt werden.

6.1. Optimierung der Implementierten Algorithmen

Der Fokus bei der Entwicklung des Frameworks lag vor allem auf der klaren Strukturierung des Programms und der Bildverarbeitungsalgorithmen. Da das Framework keine explizite Zielplattform hat, kann entsprechend auch keine Optimierung für eine solche vorgenommen werden. Zudem sollte wahrscheinlich auch folgendes beachtet werden:

> „The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet."
>
> — *Michael A. Jackson*

Neben der Spezialisierung auf bestimmte Zielplattformen, würden die Optimierungen den Quellcode der Implementierung wahrscheinlich schlechter wartbar machen. Da das Framework zudem auf die Anwendung als Demonstrationswerkzeug auf Messen konzipiert wurde, müssten durchgeführte Optimierungen eine Verbesserung für den Menschlichen Betrachter bringen.

6.2. Portierung auf andere Plattformen

Im Wesentlichen wurde das Framework im Zusammenspiel mit dem Nao Academics Roboter auf Basis von Linux getestet. Zur weiteren Verwendung des Frameworks ist es wahrscheinlich sinnvoll dieses auch auf anderen Plattformen zu testen. Sowohl im Bereich der Entwicklungsplattform als auch für die Zielhardware sollen in Zukunft weitere Betriebssysteme, Roboter und Geräte zusammen mit dem Framework getestet werden. Dies sind beispielsweise:

Lego Mindstorms NXT Steuerung des Roboters über Bluetooth, die Aufnahme und Auswertung der Bilder erfolgt über einen Computer.

Smartphones und Tablets (Android) Steuerung von Applikationen auf einem Smartphone oder Tablet. Über integrierte Kameras lassen sich die Bilder direkt mit dem Smartphone aufnehmen und dort verarbeiten. Mit inzwischen über 1GHz Takfrequenz können die Bilder direkt und quasi in Echtzeit auf dem Smartphone ausgewertet werden.

Seagate Dockstar ist eigentlich ein NAS Server, kann aber auch frei mit Software bespielt werden. Mit mehreren USB-Anschlüssen und LAN-Port auf 8×8cm bietet das Dockstar eine kleine und mobile Lösung. Über die Anschlüsse kann dann die zu steuernde Hardware, aber auch eine oder mehrere USB-Webcams angeschlossen werden.

Durch die Veröffentlichung als OpenSource Projekt könnte das Framework weiterhin eine Vielzahl von Entwicklern und Interessenten finden. Da es im Rahmen der „MATSE-Ausbildung“ entstanden ist, könnte dadurch sogar ein weiterer Werbeeffekt für die Ausbildung entstehen.

6.3. Vollständige Implementierung des Szenarios: „Memoryspielender Nao“

Für die Entwicklung des Frameworks wurden bereits einige der Spielphasen des Fallbeispiels „Memoryspielender Nao“ implementiert. Zur endgültigen Realisierung des Programms werden die Programmteile zusammengefügt und getestet. Neben der reinen Funktionalität soll zudem eine für fachfremde Messebesucher ansprechende GUI entworfen werden, die den Status des Roboters und die Zwischenschritte der Bildverarbeitungsalgorithmen darstellen kann.

Die für die Systemtests implementierten Testfälle konnten bereits als Vorführmaterial auf Messen gezeigt werden, sodass bereits eine Abschätzung der Reaktion der Messebesucher ermöglicht wird. Trotz fehlender Aufmachung für die Besucher konnten die bisher realisierten Programmteile in der Entwicklerversion bereits die Aufmerksamkeit vieler zukünftiger Auszubildender, deren Eltern, Lehrer und anderer Interessenten für die „MATSE-Ausbildung“ gewinnen.

A. Terminologie

A/D Wandler Analog-Digital Wandler: Wandelt ein Analoges Signal in ein Digitales Signal. Dabei wird das Signal innerhalb eines Wertebereiches Quantisiert und dann digitalisiert.

AVI Audio Video Interleave: Dateiformat zur verlustbehafteten Speicherung von digitalen Video- und Audiodaten.

Bounding Box Kleinstes Rechteck, dass alle Eckpunkte einer geometrischen Form komplett einschließt.

BGR Blue Green Red, Blau Grün Rot: Name eines Farbraumes zur Speicherung von Farbinformationen in Bilddaten. Der Name spiegelt die Reihenfolge der einzelnen Farbkanäle innerhalb der Datenstruktur wieder.

Histogramm Diskrete Häufigkeitsverteilung: Im der Bildverarbeitung: Häufigkeitsverteilung der Werte eines Kanals oder mehrerer Kanäle.

JPEG Joint Photographic Experts Group: Dateiformat zur verlustbehafteten Speicherung von digitalen Bilddaten.

Kanal Teil der Bilddaten der die Informationen für eine Farbe speichert.

LAN Local Area Network: Rechnernetz, das nur lokale Ausdehnung hat — i.d.R. einige Meter bis wenige Kilometer.

MPEG Moving Picture Experts Group: Dateiformat zur verlustbehafteten Speicherung von digitalen Video- und Audiodaten.

PNG Portable Network Graphic: Verlustfreies Format zur Speicherung von Bilddaten als Datei auf einem Computer.

Rauschen Signalstörung bei der Aufzeichnung von Messdaten, die sich durch zufällige Verfälschung der Messdaten auswirkt. In der Bildverarbeitung: Fehler, die bei der Aufnahme von einheitlich gefärbten Flächen auftreten.

RGB Red Green Blue, Rot Grün Blau: Name eines Farbraumes zur Speicherung von Farbinformationen in Bilddaten. Der Name spiegelt die Reihenfolge der einzelnen Farbkanäle innerhalb der Datenstruktur wieder.

SOAP Simple Object Access Protocol: Netzwerkprotokoll zum Aufruf von Methoden über ein Netzwerk. SOAP wird genutzt um die Kommunikation zwischen Objekten zu ermöglichen, die im Speicher von verschiedenen Rechnern liegen. Die Methodenaufrufe werden dafür in einer Netzwerknachricht codiert und an ein entferntes System gesendet. Auf dem System wird die Methode ausgeführt, der Rückgabewert wird dann, ebenfalls als Netzwerknachricht, zurück zum Ursprugsystem gesendet.

TCP/IP Transmission Control Protocol / Internet Protocoll: Protokolle die zum Aufbau von Rechnernetzen genutzt werden. Die Protokolle werden zur Verbindungsorientierten Übertragung zwischen den Rechnern und Adressierung der einzelnen Rechner genutzt.

USB Universal Serial Bus: Schnittstelle am PC über die sich verschiedene Peripheriegeräte anschließen lassen.

B. Entwicklerdokumentation

Der Folgende Abschnitt zeigt die, aus den Quellcodes der Klassen generierte, Dokumentation. Da die Klassen des Frameworks einer ständigen Weiterentwicklung unterliegen, ist eine Dokumentation innerhalb des Quellcodes unerlässlich um die Aktualität der Dokumentation zu garantieren. Die hier gezeigten Dokumentationen der Grundlegenden Interfaces wurden mit Doxygen [Hee10] erstellt.

B.1. Dokumentation der Klassen

B.1.1. cv4cpp::Affector Klassenreferenz

A Affector can be used to apply a (usually physical) effect at given coordinates.

`#include <affector.h>`

Öffentliche Methoden

virtual void **moveToCoordinates** (std::vector< float > &destination)=0

Move the affector to given coordinates.

virtual void **refreshCoordinates** (std::vector< float > *coords, int coordl)

Refresh this CoordinateListeners coordinate.

Ausführliche Beschreibung

A Affector can be used to apply a (usually physical) effect at given coordinates.

Dokumentation der Elementfunktionen

virtual void cv4cpp::Affector::moveToCoordinates (std::vector< float > & *destination*) [pure virtual]

Move the affector to given coordinates.

Parameter

destination destination where the affector should move to

Implementiert in **nao::ChainAffector** und **nao::cv::PanoramaAffector** .

void cv4cpp::Affector::refreshCoordinates (std::vector< float > * *coords*, int *coordl*) `[virtual]`

Refresh this CoordinateListeners coordinate.

Parameter

coords the new coordinates for this listener.

coordl number of new coordinates

Implementiert **cv4cpp::CoordinateListener** .

Erneute Implementation in **nao::cv::PanoramaAffector** .

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/affector.h

cv4cpp/affector.cpp

B.1.2. CV4CPP::ANALYZER KLASSENREFERENZ

An Analyzer extracts information from an Image.

`#include <analyzer.h>`

ÖFFENTLICHE METHODEN

Analyzer ()

Create a new Analyzer.

FREUNDBEZIEHUNGEN

cv4cpp::Analyzer & **operator>** (**cv4cpp::ImageProvider** &provider, **cv4cpp::Analyzer** &listener)

AUSFÜHRLICHE BESCHREIBUNG

An Analyzer extracts information from an Image. An Analyzer might be added to any ImageProvider to extract coordinates of features from the image data.

BESCHREIBUNG DER KONSTRUKTOREN UND DESTRUKTOREN

cv4cpp::Analyzer::Analyzer ()

Create a new Analyzer.

This implementation will not provide any coordinates.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/analyzer.h

cv4cpp/analyzer.cpp

B.1.3. cv4cpp::CoordinateConverter Klassenreferenz

A CoordinateConverter convters coordinates from one space to an other.

`#include <coordinateconverter.h>`

Freundbeziehungen

cv4cpp::CoordinateConverter & **operator>** (**cv4cpp::CoordinateProvider** &provider, **cv4cpp::CoordinateConverter** &listener)

Ausführliche Beschreibung

A CoordinateConverter convters coordinates from one space to an other.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/coordinateconverter.h

cv4cpp/coordinateconverter.cpp

B.1.4. cv4cpp::CoordinateListener Klassenreferenz

A CoordinateListener may be subscribed to a CoordinateProvider to do some kind of processing on the coordinates delivered by the Analyzer.

`#include <coordinatelistener.h>`

Öffentliche Methoden

virtual void **refreshCoordinates** (std::vector< float > *coords, int coordl)=0

Refresh this CoordinateListeners coordinate.

Ausführliche Beschreibung

A CoordinateListener may be subscribed to a CoordinateProvider to do some kind of processing on the coordinates delivered by the Analyzer. This class provides no processing of the coordinates this can be defined by overriding the refreshImage() method.

Dokumentation der Elementfunktionen

virtual void cv4cpp::CoordinateListener::refreshCoordinates (std::vector< float > * *coords*, int *coordl*) `[pure virtual]`

Refresh this CoordinateListeners coordinate.

Parameter

coords the new coordinates for this listener.

coordl number of new coordinates

Implementiert in **cv4cpp::Affector** , **cv4cpp::calibration::LineCalibration** , **cv4cpp::calibration::Offset** , **cv4cpp::output::Console** , **cv4cpp::output::Painter** , **memory::VisionMemory** , **nao::cv::Panorama2Bodyspace** und **nao::cv::PanoramaAffector** .

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/coordinatelistener.h

cv4cpp/coordinatelistener.cpp

B.1.5. cv4cpp::CoordinateProvider Klassenreferenz

A CoordinateProvider delivers corrdinates from an arbitrary source.

`#include <coordinateprovider.h>`

Öffentliche Methoden

virtual int **getCoordinates** (std::vector< float > *&coords)=0

Get the coordinates resulting from the last image analysis.

virtual int **getDimension** ()=0

The dimension the result of this Analyzer has.

virtual void **subscribe** (**CoordinateListener** *l)

Add a Coordinate Listener to the subscribers.

virtual void **unsubscribe** (**CoordinateListener** *l)

Remove a CoordinareListener from the subscribers.

Geschützte Methoden

virtual void **notifySubscribers** ()

Refresh all subscribers with current coordinates.

Freundbeziehungen

cv4cpp::CoordinateProvider & **operator>=** (**cv4cpp::CoordinateProvider** &provider, **cv4cpp::CoordinateListener** &listener)

cv4cpp::CoordinateProvider & **operator-=** (**cv4cpp::CoordinateProvider** &provider, **cv4cpp::CoordinateListener** &listener)

Ausführliche Beschreibung

A CoordinateProvider delivers corrdinates from an arbitrary source. A CoordinateProvider may frequently call a CoordinateListeners refreshCoordinates() method. A CoordinateListener might subscribe to the CoordinateProvier, causing the CoordinateProvider to call the CoordinateListeners refreshCoordinates() method. This class delivers a basic implementation for subscribing and unsubscribing. Calling the notifySubscribers() method will cause the ImageProvider to notify all subscribers.

Dokumentation der Elementfunktionen

virtual int cv4cpp::CoordinateProvider::getCoordinates (std::vector< float > *& *coords*) `[pure virtual]`

Get the coordinates resulting from the last image analysis.

The dimension of the returned points might be obtained by the getDimension() method. Previous contents of coords will not be deallocated! Do not deallocate coords!

Parameter

coords destination of the coordinates (vector of points)

Rückgabe number of coordinates returned

Implementiert in **cv4cpp::analyzers::HistogramComparator** , **cv4cpp::analyzers::Lines** , **cv4cpp::analyzers::MinMax** , **cv4cpp::calibration::Offset** , **memory::VisionMemory** und **nao::cv::Panorama2Bodyspace** .

virtual int cv4cpp::CoordinateProvider::getDimension () `[pure virtual]`

The dimension the result of this Analyzer has.

Usual values for this are 1 if the Analyzer counts something, 2 or 3 if the results are points within 2D or 3D space.

Rückgabe dimension count.

Implementiert in **cv4cpp::analyzers::HistogramComparator** , **cv4cpp::analyzers::Lines** , **cv4cpp::analyzers::MinMax** , **cv4cpp::calibration::Offset** , **memory::VisionMemory** und **nao::cv::Panorama2Bodyspace** .

void cv4cpp::CoordinateProvider::subscribe (CoordinateListener * *l*) `[virtual]`

Add a Coordinate Listener to the subscribers.

Parameter

l new subscriber

void cv4cpp::CoordinateProvider::unsubscribe (CoordinateListener * *l*) `[virtual]`

Remove a CoordinareListener from the subscribers.

Parameter

l the subscriber to be removed

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/coordinateprovider.h

cv4cpp/coordinateprovider.cpp

B.1.6. CV4CPP::FILTER KLASSENREFERENZ

A Filter processes image data.

`#include <filter.h>`

FREUNDBEZIEHUNGEN

cv4cpp::Filter & **operator>** (**cv4cpp::ImageProvider** &provider, **cv4cpp::Filter** &listener)

AUSFÜHRLICHE BESCHREIBUNG

A Filter processes image data. The image data is converted it but keeps genral image character (usually a two dimensional set of data).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/filter.h

cv4cpp/filter.cpp

B.1.7. CV4CPP::IMAGELISTENER KLASSENREFERENZ

A ImageListener may be subscribed to a ImageProvider to do some kind of processing with the image data delivered by the ImageProvider.

`#include <imagelistener.h>`

ÖFFENTLICHE METHODEN

virtual void **refreshImage** (IplImage *img)=0

Refresh this ImageListeners image.

AUSFÜHRLICHE BESCHREIBUNG

A ImageListener may be subscribed to a ImageProvider to do some kind of processing with the image data delivered by the ImageProvider. The class ImageListener provides no implmentation of processing the image. The image processing can be defined by overriding the refreshImage() method.

DOKUMENTATION DER ELEMENTFUNKTIONEN

virtual void cv4cpp::ImageListener::refreshImage (IplImage * *img*) `[pure virtual]`

Refresh this ImageListeners image.

Parameter

img the new image for this listener.

Implementiert in **cv4cpp::analyzers::HistogramComparator** , **cv4cpp::analyzers::Lines** , **cv4cpp::analyzers::MinMax** , **cv4cpp::calibration::LineCalibration** , **cv4cpp::filters::MatchTemplate** , **cv4cpp::filters::Monochrome** , **cv4cpp::filters::Smooth** , **cv4cpp::filters::SplitChannels** , **cv4cpp::output::Console** , **cv4cpp::output::FileSystem** , **cv4cpp::output::Painter** , **cv4cpp::output::Window** , **memory::VisionMemory** und **qt4cv::Image** .

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/imagelistener.h

cv4cpp/imagelistener.cpp

B.1.8. cv4cpp::ImageProvider Klassenreferenz

A ImageProvider delivers image data from an arbitrary source.

`#include <imageprovider.h>`

Öffentliche Methoden

ImageProvider ()

Create a new ImageProvider.

virtual void **subscribe** (**ImageListener** *l)

Add an ImageListener to the subscribers.

virtual void **unsubscribe** (**ImageListener** *l)

Remove an ImageListener from the subscribers.

virtual IplImage * **getCurrentImage** ()=0

Get the current image from this ImageProvider.

Geschützte Methoden

virtual void **notifySubscribers** ()

Refresh all subscribers with current image data.

Freundbeziehungen

cv4cpp::ImageProvider & **operator>=** (**cv4cpp::ImageProvider** &provider, **cv4cpp::ImageListener** &listener)

cv4cpp::ImageProvider & **operator-=** (**cv4cpp::ImageProvider** &provider, **cv4cpp::ImageListener** &listener)

IplImage * **operator!** (**cv4cpp::ImageProvider** &provider)

Ausführliche Beschreibung

A ImageProvider delivers image data from an arbitrary source. A ImageProvider may frequently call a ImageListeners refreshImage method once it subscribed the ImageProvider. The class ImageProvider delivers a basic implementation of subscribtion and unsubscribtion. The image source can be defined by overriding the getCurrentImage method. Calling the notifySubscribers() method will cause the ImageProvider to notify all subscribers.

Beschreibung der Konstruktoren und Destruktoren

cv4cpp::ImageProvider::ImageProvider ()

Create a new ImageProvider.

This implementation will not provide any image.

Dokumentation der Elementfunktionen

void cv4cpp::ImageProvider::subscribe (ImageListener * *l*) `[virtual]`

Add an ImageListener to the subscribers.

Parameter

l new subscriber

Erneute Implementation in **cv4cpp::devices::SingleImage** .

void cv4cpp::ImageProvider::unsubscribe (ImageListener * *l*) `[virtual]`

Remove an ImageListener from the subscribers.

Parameter

l the subscriber to be removed

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

cv4cpp/imageprovider.h

cv4cpp/imageprovider.cpp

C. Literaturverzeichnis

[Ald10a] Aldebaran-Robotics (Hrsg.): *Nao for Education*. http://aldebaran-robotics.com/en/naoeducation. Version: 2010. – Abgerufen am 14.12.2010

[Ald10b] Aldebaran-Robotics (Hrsg.): *Nao Documentation*. 1.8.16. Paris: Aldebaran-Robotics, 2006-2010. http://academics.aldebaran-robotics.com/docs/site_en/index_doc.html. – Abgerufen am 14.12.2010

[BK08] Bradski, Gary ; Kaehler, Adrian: *Learning OpenCV*. First. Sebastopool, CA : O'Reilly Media Inc., 2008 http://oreilly.com/catalog/9780596516130. – ISBN 978–0–596–51613–0

[Bla10] Blaise, Barney ; Lawrence Livermore National Laboratory (Hrsg.): *POSIX Threads Programming*. Webseite. https://computing.llnl.gov/tutorials/pthreads/. Version: 2010. – Abgerufen am 22.06.2010

[Bus98] Buschmann, Frank: *Pattern-orientierte Software-Architektur: ein Pattern-System*. Addison-Wesley, 1998 http://books.google.de/books?id=o2nuK0qpo3QC. – ISBN 9783827312822

[FW04] Fawer, Markus ; Wysssmann, Adrian: *OCT3D plus*, Fachhochschule Bern, Diplomarbeit, Dezember 2004. http://diploma.wyssmann.com/index.html

[FZJ10] Forschungszentrum Jülich GmbH (Hrsg.): *Phytoshäre Enabling Technologies: GARNICS*. http://www.fz-juelich.de/icg/icg-3/imaging/garnics. Version: 2010. – Abgerufen am 20.12.2010

[Hee10] Heesch, Dimitri van: *Doxygen Manual*. 1.7.0, 2010. http://www.stack.nl/~dimitri/doxygen/manual.html

[Jäh02] Jähne, Prof. Dr. B.: *Digitale Bildverarbeitung*. 5. Springer-Verlag Berlin, 2002. – ISBN 3–540–51260–3

[Kle09] Kleuker, Stephan: *Grundkurs Software Engeneering mit UML*. 1. Wiesbaden : Vieweg + Teubner, 2009 http://www.cs.hs-rm.de/~kleuker/SoftwareEngineering.html. – ISBN 978–3–8348–0391–7

[Lem10] LemnaTec GmbH (Hrsg.): *Scanalizer 3D Plant Prototyping*. http://www.lemnatec.com/product/scanalyzer-3d-plant-phenotyping. Version: 2010. – Abgerufen am 20.12.2010

[Pol10] Politze, Marius: *OpenCV in a Nutshell*. Oktober 2010

[VDM08] VDMA Robotik und Automation: Branche erwartet für 2008 Rekordergebnis. In: *PresseBox* (2008), Juni. http://www.pressebox.de/pressemeldungen/verband-deutscher-maschinen-und-anlagenbau-ev-frankfurtmain/boxid/180265

[Wik10a] Wikipedia Foundation Inc. (Hrsg.): *Active pixel sensor*. http://en.wikipedia.org/w/index.php?title=Active_pixel_sensor&oldid=400443552. Version: 2010. – Abgerufen am 10.12.2010

[Wik10b] Wikipedia Foundation Inc. (Hrsg.): *CCD-Sensor*. http://de.wikipedia.org/w/index.php?title=CCD-Sensor&oldid=79948241. Version: 2010. – Abgerufen am 10.12.2010

D. Abbildungsverzeichnis