BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Entwicklung und Implementierung visuomotorischer Verfahren zur Bewegungsverfolgung, Entfernungsschätzung und Objekterkennung

von

Jonas Wurst

EDV.Nr.:810160

dem Fachbereich VII – Elektrotechnik - Mechatronik - Optometrie –
der Beuth Hochschule für Technik Berlin vorgelegte Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Engineering (B.Eng.)
im Studiengang
Elektronik und Kommunikationssysteme

Tag der Abgabe 2. September 2016

Betreuer:
Prof. Dr. M. Hild - Beuth Hochschule für Technik

Gutachter:
Prof. Dr.-Ing. T. Merkel - Beuth Hochschule für Technik

Kurzfassung

Im Rahmen dieser Arbeit wurden visuomotorische Verfahren entwickelt die eine Grundlage für die visuelle Wahrnehmung des humanoiden Roboters Myon darstellen. Zu diesen Verfahren gehört das Schätzen der Entfernung mit einer Kamera und die daraus resultierende Nah- und Fernfeldseparation, welche wiederum die Grundlage für die Verfahren zum Verfolgen von Bewegungen und dem Lokalisieren von Objekten, auf die gezeigt wurde, ist. Zusätzlich wird eine eigenständige Flächenerkennung (Blobdetection) vorgestellt, die Flächen mit gleicher Farbe lokalisiert. Alle diese Verfahren bauen auf der sogenannten Pixel-Pipeline auf, welche eine in FPGA implentierte Bildvorverarbeitungseinheit ist. Mit der im Zuge dieser Arbeit erweiterten Variante der Pixel-Pipeline sind Affine Transformationen, Differenzbilder, Histogramme und Farbklassifizierungen in Echtzeit möglich.

Abstract

In the context of this bachelor thesis visumotor procedures have been developed providing a basis for the visual perception of the humanoid robot Myon. These principles include the distance estimation with a monovision system and the resulting near and far field separation. Based on these methods further techniques for movement tracking and object localization have been designed. For this purpose the desired objects are marked with a pointing movement. Additionally a seperated blob detection based on color is presented. All these principles are based on the so-called pixel pipeline, which is a image processing unit implemented in FPGA. Moreover the pixel pipeline has been extended in order to provide affine transformations, differential images, histograms and color classifications in real time.

Erklärung

Ich versichere, dass ich diese Abschlussarbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Datum Unterschrift

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Robotik mit all ihren Teildisziplinen gewinnt an immer größerer Bedeutung, was zum einen an den wachsenden Anwendungsgebieten und zum anderen an den Forschungsergebnissen der einzelnen Disziplinen liegt. Längst sind die denkbaren Arbeitsbereiche von Roboter nicht mehr nur im industriellen Umfeld angesiedelt, sondern auch in deutlich komplexeren Umgebungen. Einfache Roboter werden bereits in zahlreichen Haushalten für das Erledigen diverser, alltäglicher Aufgaben eingesetzt. Die Komplexität der mögliche Aufgaben hängt dabei von den Wahrnehmungsmöglichkeiten dieser Systeme ab. Eine der wichtigsten ist hierbei die visuelle Wahrnehmung, die nicht nur in der Robotik eine fundamentale Rolle einnimmt. Die Steuerung verschiedenster Geräte ist in zwischen berührungsfrei über Gesten und Bewegungen möglich, was beispielsweise in einem Umfeld mit hohen hygienischen Anforderungen eingestzt werden könnte. Ebenso wird die Bildverarbeitung für zahlreiche Sicherheitstechniken verwendet, um zum Beispiel Personen zu identifizieren oder Bewegungen zu erkennen.

1.1 Motivation und Zielsetzung

Die Anforderungen an die Bildverarbeitung nehmen stetig zu, was neue und innovative Verfahren und Techniken voraussetzt. Dank steigender Rechenleistungen in der Computer basierten Verarbeitung können immer komplexere Verfahren realisiert werden. In der Robotik stehen jedoch nur deutlich stärker begrenzte Energie- und Rechenleistungsressourcen zur Verfügung. Entgegen dem Trend zu immer komplexeren Prinzipien, muss hier nach minimalistischen Verfahren geforscht werden. Von zentraler Rolle sind hierbei sinnvolle Systemarchitekturen und eine effiziente Ausnutzung derer. So ermöglichen parallele Strukturen das Bilden von vielen Teilsystemen, die für eine bestimmte Aufgabe ausgelegt sind. Für die vorliegende Arbeit wurde der humanoide Roboter Myon als Plattform verwendet, dessen verteilte Rechnerarchitektur eben solche parallelen Prozesse zu lässt.
Die in dieser Arbeit vorgestellten Verfahren bilden die Grundlage für ein visuelles Umgebungsmodell, was den Roboter dazu befähigt davon abhängige Entscheidungen zu treffen. Eine Anwendung dieses Modells ist eine Variante des sogenannten Naming Games [SL12], welches aufbauend auf den hier vorgestellten Verfahren entstehen soll. Die Variante des

Naming Games besteht unter anderem daraus Objekte zu erkennen, diese anhand des Umgebungsmodells einzuordnen und den Namen der Objekte zu lernen. Aufsetzend darauf können dem Roboter neue Objekte beigebracht werden, in dem darauf gezeigt wird, sie in das Sichtfeld des Roboters gebracht werden oder seine Blickrichtung interaktiv so verändert wird, dass er neue Objekte finden kann. Mit Algorithmen zur Objekterkennung kann anschließend untersucht werden ob bereits Objekte von diesem Typ bekannt sind, was wiederum die Grundlage für eine Art Konversation bilden könnte. Diese Arbeit besteht aus mehreren Teilzielen um das geplante Naming Game zu ermöglichen. Hierzu gehört das Verfahren zum Schätzen der Entfernung eines Objektes, was die Informationen für das Umgebungsmodell liefert. Die aus der Entfernungsschätzung resultierende Nah- und Fernfeldseparation, die eine grobe Schätzung eines Umgebungsmodells repräsentiert, ist ein weiteres Teilziel. Das Erkennen von Veränderungen und Bewegungen als eigenes Teilziel ist gleichzeitig, zusammen mit der Nah- und Fernfeldseparation, die Grundlage für die visuomotorischen Verfahren. Die Teilziele der Visuomotorik sind das Verfolgen von Bewegungen sowie das Erkennen von Objekten auf die gezeigt wird. Um Objekte gleicher Farbe näher zu untersuchen müssen potentielle Bereich im Bild vorab gefunden werden, was die Aufgabe der in dieser Arbeit vorgestellten Flächenerkennung ist. Um die genannten Teilziele effizient umsetzen zu können, wurde die in Hardware implementierte Pixel-Pipeline erweitert, so dass viele der notwendigen Rechenschritte ausgelagert werden können.

1.2 Aufbau der Arbeit

Einleitend wird die verwendete Plattform, der humanoide Roboter Myon, in Kapitel 2 vorgestellt, was gleichzeitig als wesentliche Grundlage aller folgenden Kapitel zu verstehen ist, da die Verfahren auf die Plattform ausgelegt sind und auch anhand dieser erläutert werden. Kapitel 3 stellt die Modifikationen der Plattform vor, die notwendig sind um eine effiziente und sichere Funktionsweise der darauf aufbauenden Verfahren zu ermöglichen. Teil diese Kapitels ist auch die Pixel-Pipeline, die eine fundamentale Rolle für das Verständnis der gesamten Arbeit spielt. Kapitel 4 beinhaltet die Flächenerkennung, die zwar eigenständig ist, deren Grundprinzipien im Folgenden aber immer wieder aufgegriffen werden. Die Entfernungsschätzung (Kapitel 5) baut im wesentlichen auf Kapitel 2, der verwendeten Plattform, auf. Die visuomotorischen Verfahren in Kapitel 6 führen die Inhalte aus den Kapiteln 2-5 zusammen. Im abschließenden 7. Kapitel werden mögliche Verbesserungen der einzelnen Verfahren erläutert.

Kapitel 2

Der humanoide Roboter Myon

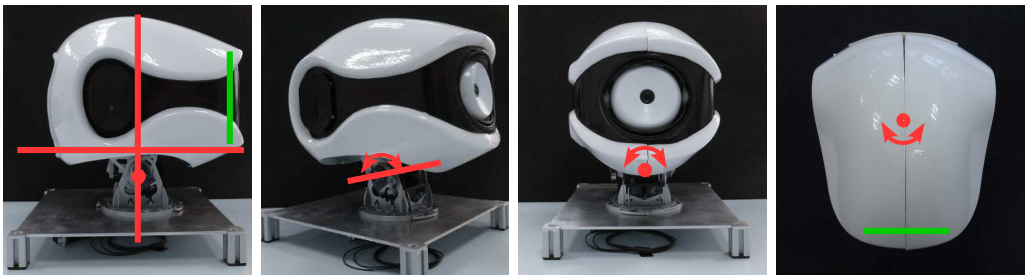
Die in dieser Arbeit vorgestellten Algorithmen wurden hinsichtlich der Verwendung auf dem humanoiden Roboter Myon konzipiert. Die plattformspezifischen Eigenschaften verändern und verbessern Teile der Algorithmen – manche werden dadurch erst ermöglicht. Dieses Kapitel bietet eine grundlegende Übersicht über die Funktionalität des Roboters und stellt relevante Eckdaten vor.

Abbildung 2.1: Der humanoide Roboter Myon ist eine Forschungsplattform, entwickelt und gefertigt im NRL der Beuth Hoschule für Technik Berlin. Myon ist 1,25 m groß und 16 kg schwer [Thi14].[HSB+11]

Myon ist im Rahmen des EU-Projekts ALEAR[1] im Forschungslabor Neurorobotik (NRL) der Beuth Hochschule für Technik Berlin entwickelt und gefertigt worden. Er dient als Forschungsplattform für diverse Algorithmen, Regelschleifen und kognitive Prozesse. Dank des modularen Aufbau des Roboters und der verteilten Rechen- und Energieversorgungsarchitektur, können Arme, Beine, Torso und Kopf eigenständig betrieben werden. Zudem können sie zur Laufzeit voneinander getrennt oder zusammengefügt werden [HSB+11]. Die in allen genannten Gliedmaßen verbauten Accelboard3Ds übernehmen neben der Steuerung der Motoren, dem Bereitstellen und Verarbeiten von Sensordaten auch Berechnungen für dezentrale Regler oder künstliche neuronale Netze [Wur16]. Jede dieser Streueinheiten ist mit dem zentralen Datenbus (Spinalcord[2]) des Roboters verbunden.

2.1 Technische Details zu Myons Kopf

Ein besonderes Augenmerk liegt auf dem Kopf des Roboters, da das Wissen über dessen Beschaffenheit und Systemstruktur für das Verständnis der Algorithmen notwendig ist und die gesamte Bildverarbeitung dieser Arbeit im Kopf vorgenommen wird. Zudem wird ausschließlich der Kopf bewegt, alle anderen Gliedmaßen werden für diese Arbeit als starr angenommen.



(a) (b) (c) (d)

Abbildung 2.2: Geometrie und Achsen von Myons Kopf. In (a) ist die Nick-, Roll- und Gierachse des Kopfes rot und die Kamera-Sensorebene grün markiert, diese korrespondiert mit der grünen Markierung in (d)(Die Länge der grünen Markierung ist nicht maßstabgerecht). In (b) ist die Nick-, in (c) die Roll- und in (d) die Gierachse jeweils rot markiert. [Wur16]

Der Kopf des Roboters ist in den wesentlichen Funktionen dem des Menschen nachempfunden. Er verfügt über zwei Ohren – Mikrofone links und rechts – , einen Lautsprecher zum Sprechen und einen Hals über den sich der Kopf nicken, neigen und drehen (Nick-, Roll- und Gierachse) lässt (Abbildungen 2.2). Das Sehvermögen bildet hierbei eine Ausnahme,

[1] ALEAR: Artificial Language Evolution on Autonomous Robots

[2] „Spinalcord ist der englische Begriff für das Rückenmark. Wegen der Analogie zur Signalübertragung von Neuronen wurde dieser Begriff gewählt.“([Thi14], S.27)

da die einzelne Kamera dem Roboter kein räumliches Sehen ermöglicht. Die zentrale Recheneinheit bildet das Brainmodul, welches dem Namen entsprechend die Anlehnung an das menschliche Gehirn ist.
Die Grundlage der visuellen Wahrnehmung des Roboters bildet die verbaute Kamera 21K15XDIG von der Firma Videology. Sie besteht aus einen 1/4“ IL CCD Bildsensor, welcher mit 752 x 582 [Vid11] aktiven Pixeln arbeitet und einem Objektiv mit einer festen Brennweite von 3,14 mm. Die Kamera befindet sich vorne am Kopf und liegt daher nicht auf der Gierachse, was für die in Kapitel 5 vorgestellte Entfernungsschätzung von besonderer Bedeutung ist.
Um die Perspektive des Roboters zu veranschaulichen wird das Kamerabild in Echtzeit über eine Chinch-Buchse am Hinterkopf ausgegeben. Zusätzlich können in diesem Bild Overlays dargestellt werden, um so beliebige Debuginformationen anzuzeigen.

Das Brainmodul die zentrale Recheneinheit
Das Brainmodul ist konzeptionell für komplexe und rechenintensive Vorgänge, wie Audio-

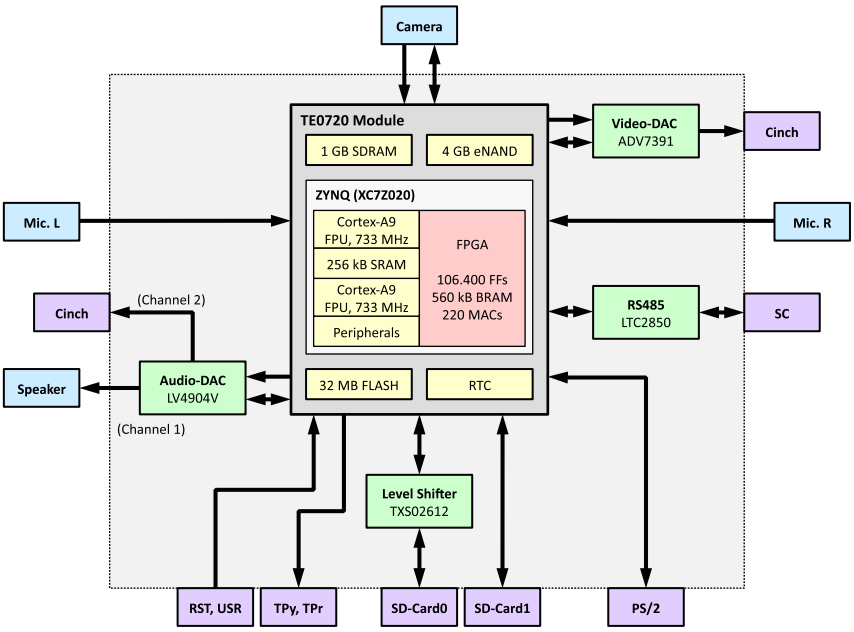


Abbildung 2.3: Struktur des Brainmoduls (hellgrau hinterlegt) mit der verwendeten Peripherie. Von SD-Karte 0 wird das Brainmodul gebootet, zudem dient sie als Langzeitspeicher. Der Video-DAC(Digital to Analog Converter) stellt die Bildaten mittels einer Chinchbuchse zur Verfügung. Die Kamera ist direkt mit dem TE07020 Module verbunden. Quelle: Labor-internes Dokument

und Videoverarbeitung oder Verhaltenssteuerungen, gedacht. So würde zum Beispiel die Regelung für das Heben eines Armes vollständig auf dem zugehörigen Accelboard3D ausgeführt werden, während die Entscheidung für das Heben, wie auch das Ermitteln und

Übertragen neuer Sollwerte, vom Brainmodul übernommen wird. Um diese Kommunikation zu ermöglichen ist das Brainmodul ein Teilnehmer des Spinalcords, ohne dabei eine übergeordnete Rolle einzunehmen.
Die Software und Hardwarekonfiguration des Brainmoduls – das Verhalten – wird von einer SD-Karte geladen. Diese Karte wird auch als Langzeitspeicher verwendet, um den sogenannten Dream - das Gedächtnis - des Roboters zu sichern. Der Dream enthält zyklisch gespeicherte Sensorwerte und grob aufgelöste Kamerabilder, was eine externe Auswertung, um zum Beispiel Lernverfahren zu Erproben, ermöglicht.
Die Kamera, der Video-Ausgang sowie die gesamte Peripherie des Kopfes sind direkt mit dem Brainmodul verbunden. Dessen Kernstück bildet das TE0720 Modul[3] auf dem sich ein Zynq7020[4] und zusätzliche Peripherie befindet. Die sich aus den genannten Komponenten ergebende Gesamtstruktur des Brainmoduls ist in Abbildung 2.3 zu sehen.

Die Bildverarbeitungsstruktur des Roboters
Die Bildverarbeitung wird teils im Prozessor und teils im FPGA[5] des Zynq7020 realisiert, was die Möglichkeit bietet diverse Verarbeitungsschritte aus dem Prozessor auszulagern und diesen so zu entlasten. Die in Hardware implementierten Module werden jeweils zu sogenannten IP[6]-Blöcken zusammengefasst. Diese Blöcke können mit anderen IPs und dem Prozessor verbunden werden. Die Basiseinheiten der Bildverarbeitung sind vollständig als solche IPs im FPGA implementiert. Ihre Einordnung in die Bildverarbeitungsstruktur sind in Abbildung 2.4 gezeigt.
Die grundlegende Einheit der Bildeingabe und -ausgabe ist das CAM-Modul. Es hat zum einen die Aufgabe die Videodaten für die bereits erwähnte Video-Ausgabe zu generieren und zum anderen ist es die erste Instanz der Bildverarbeitung, indem die Bilddaten der Kamera zum sogenannten Pixelstream gewandelt und für die folgenden Einheiten bereitgestellt werden. Eine dieser Einheiten ist die Pixel-Pipeline, welche erste Verarbeitungsschritte übernimmt und die aus dem Pixelstream erzeugten Daten in einer Pixel-RAM zur Verfügung stellt. Der Prozessor kann diese Daten auslesen und die Pixel-Pipeline konfigurieren. Die Funktion des CAM-Moduls und der Pixel-Pipeline wurden als Teil dieser Arbeit modifiziert und erweitert, nähere Erläuterungen zu den beiden Modulen sind daher in Kapitel 3 aufgeführt.

[3] Hersteller: Trenz Elektronik

[4] ARM dual-core Cortex-A9 MPCore + Field Programmable Gate Array; Hersteller: Xilinx

[5] FPGA: Field Programmable Gate Array

[6] IP: Intellectual Property

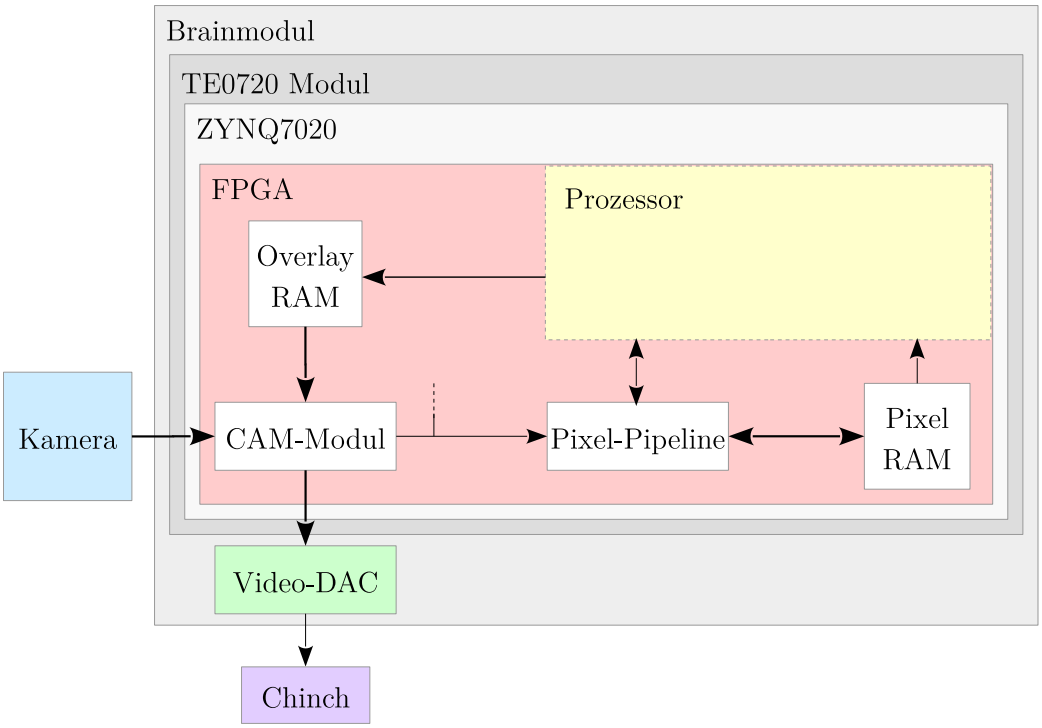


Abbildung 2.4: Das vereinfachte Diagramm zur Bildverarbeitung zeigt die relevanten Schritte der Bildvorverarbeitung (CAM-Modul, Pixel-Pipeline und PixelRAM) und Nachbearbeitung (OverlayRAM, CAM-Modul) im FPGA. Die Bilddaten der Kamera als auch der Videoausgang werden vom CAM-Modul verwaltet. Die gestrichelte Linie symbolisiert weitere Bildverarbeitungsmodule die mit dem Pixelstream arbeiten. Die Kamera lässt sich über den Prozessor konfigurieren.

2.2 Beschreibung der vorhandenen Regelstrukturen

Wie in dem Beispiel über das Heben eines Armes, wird die gesamte Regelung des Roboters dezentral auf den jeweiligen Accelboard3Ds realisiert. Der Entwurf und die Implementierung dieser Regelstrukturen erfolgt über die Software Braindesigner [Thi14]. Um die Sollwerte oder etwaige Parameter der Regler zu modifizieren wird der entsprechende Wert über den Spinalcord übertragen. Diese Werte können von anderen Accelboard3Ds, externen Spinalcordteilnehmern oder dem Brainmodul stammen. Auch die für diese Arbeit notwendigen Bewegungen des Kopfes werden dezentral geregelt, wobei neue Sollwerte vom Brainmodul an bereits implementierte Regler übertragen werden.
Die in Kapitel 6 vorgestellten Prinzipien beschränken sich auf die Verwendung der Nick- und der Gierachse, welche über eine identische Struktur geregelt werden.

Die Geschwindigkeitsregelung der verwendeten Kopfachsen
Um die Kopfbewegungen natürlich wirken zu lassen, wird für die Regelung der Nick- und Gierachse eine Geschwindigkeitsregelung verwendet. Abbildung 2.5 zeigt die verwendete Regelstruktur, wobei t für die Abtastschritte steht ($t \in \mathbb{Z}$). Der Sollwinkel φ_{soll} wird wie beschrieben vom Brainmodul übergeben. Mit dem Momentanwinkel $\varphi(t)$ wird die Sollgeschwindigkeit $\omega_{soll}(t)$ durch die Regelabweichung $\varphi_{soll} - \varphi(t)$ gewichtet mit dem Faktor p vorgegeben. Die Geschwindigkeit $\dot{\varphi}(t) = \omega(t)$ wird mit dem Faktor $-a$ multipliziert, so folgt für die faktorisierte Geschwindigkeit:

$$-a \cdot \omega(t) = -a \cdot [\varphi(t) - \varphi(t-1)] \tag{2.1}$$

Durch die Integration der Regelabweichung $p \cdot \omega_{soll}(t) - a \cdot \omega(t)$ ergibt sich die geregelte Motorspannung, welche keine bleibende Regelabweichung aufweist. Der Zusammenhang für die Motorspannung $u(t)$ ergibt sich zu:

$$u(t) = \left[p \cdot \omega_{soll}(t) - a \cdot \omega(t) + \frac{u(t+1)}{b}\right] \cdot b \tag{2.2}$$

$$u(t) = p \cdot b \cdot [\varphi_{soll} - \varphi(t)] - a \cdot b \cdot [\varphi(t) - \varphi(t-1)] + u(t+1) \tag{2.3}$$

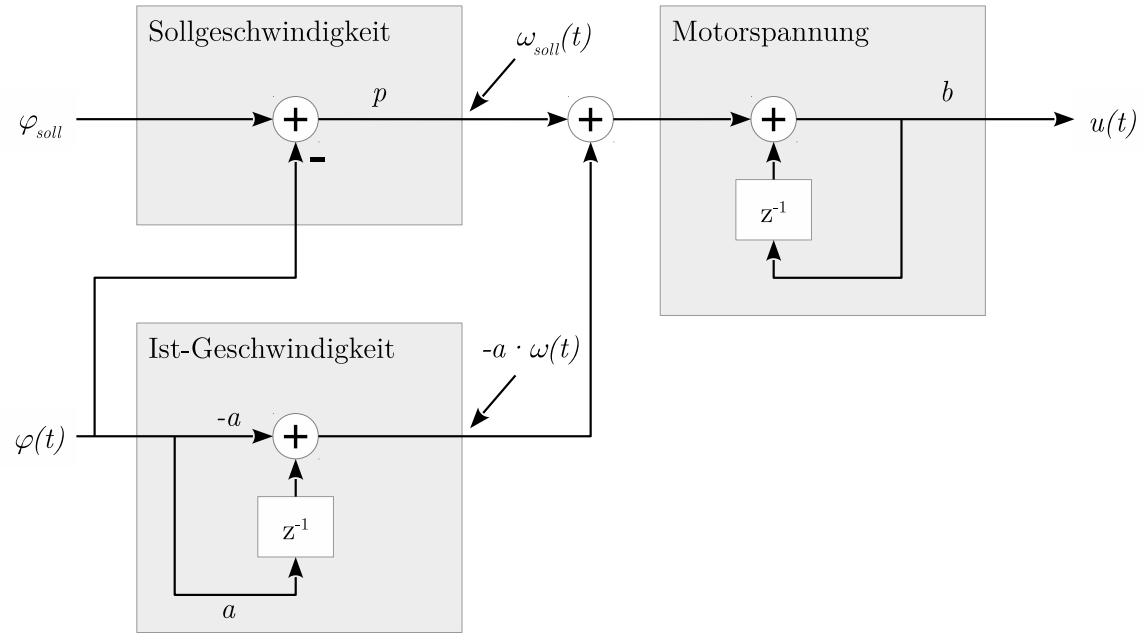


Abbildung 2.5: Die Regelschleife für die Gier- und Nickachse stellt eine Geschwindigkeitsregelung dar. Mit den Faktoren in Tabelle 2.1 ergibt sich eine langsame Bewegung ohne bleibende Regelabweichung.

Für die beiden Achsen werden unterschiedliche Faktoren verwendet:

Faktor	Gierachse	Nickachse
a	8,000	5,000
b	0,300	0,800
p	0,196	0,075

Tabelle 2.1: Faktoren der Regelstruktur für die beiden verwendeten Achsen.

Durch die kleinen b-Faktoren hat die Regelabweichung des Winkels nur einen kleinen Einfluss auf die Sollgeschwindigkeit. Die verhältnismäßig großen a-Faktoren hingegen erhöhen die Ist-Geschwindigkeit. Im Zusammenspiel mit der Integration ergibt sich eine träge und dadurch natürlich wirkende Bewegung.
Der Kopf kann sich Aufgrund von mechanischen Anschlägen nur innerhalb der in Abbildung 2.6 gezeigten Bereiche bewegen. Die Regler der beiden Achsen verfügen über eine zusätzliche Struktur die ein Anstoßen an diese Anschläge verhindert.

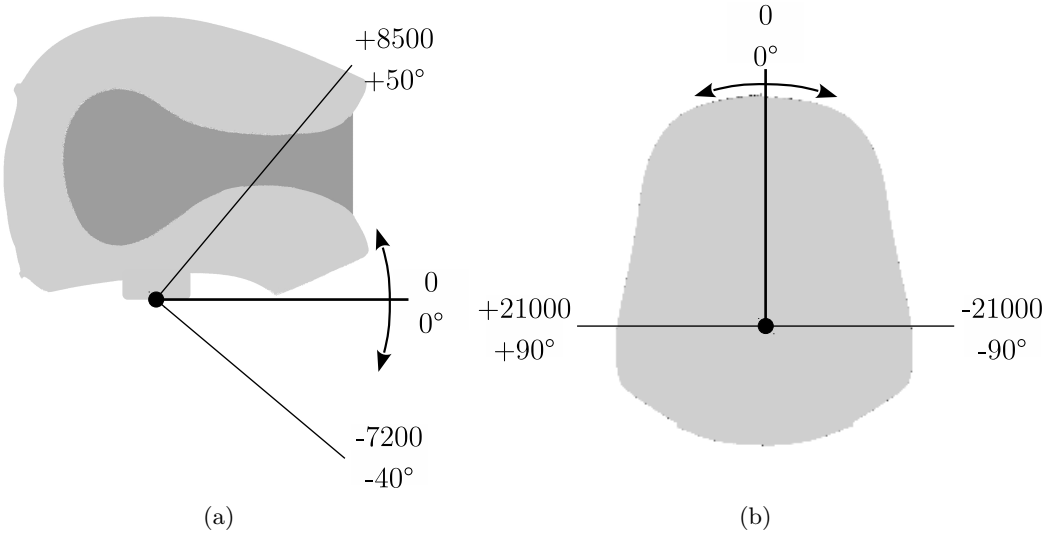


Abbildung 2.6: Die Bewegungsbereiche mit den zugehörigen Winkelangaben für die (a) Nick- und (b) Gierachse. Jeweils oben die Angaben in Winkel-Einheiten (WE) und unten die zugehörigen Werte in Grad. Die Angaben in WE werden im Folgenden als ϑ_{Gier} und ϑ_{Nick} bezeichnet.

Kapitel 3

Vorbereitende Modifikationen

Die grundlegenden Einheiten der Bildverarbeitung bilden das CAM-Modul und die Pixel-Pipeline (Abb.2.4), die beide im FPGA implementiert sind. Um die in dieser Arbeit vorgestellten Verfahren effizient realisieren zu können, musste die Funktionalität der beiden Module erhöht und verbessert werden. In diesem Kapitel werden die notwendigen Modifikationen erläutert.
Das CAM-Modul ist die Schnittstelle zur Kamera und zum Video-DAC. Außerdem werden hier die Kameradaten zum Pixelstream, für die folgenden Bildverarbeitungsschritte, aufbereitet. Das CAM-Modul wies eine fehlerhafte Synchronisierung auf, was im Rahmen der vorbereitenden Modifikationen korrigiert wurde. Die mit dem Pixelstream arbeitende Pixel-Pipeline wurde um einige Funktionen erweitert, um noch mehr rechen- und zeitaufwendige Teilschritte der Bildverarbeitung aus dem Prozessor auszulagern.

3.1 Korrektur und Erläuterung des CAM-Moduls

Die Aufgabe des CAM-Moduls ist neben dem Erzeugen des Pixelstreams, das Generieren der Videodaten für die Ausgabe mittels dem Video-DAC. Die Videodaten bestehen aus dem Kamerabild und beliebigen Overlays (s. Kap. 2.1). Der Pixelstream hingegen beinhaltet die reinen Bilddaten. Die Grundlage für die Kamera- und Videodaten bildet der ITU-BT.656 Standard[Int07], in dem die Formate und Timings festgelegt sind.

Erläuterungen zu den Empfehlungen des ITU-BT.656 Standard
Die Empfehlungen teilen sich in den 525- und den 625-Linien-Standard auf. Letzterer wird für die Realisierung des CAM-Moduls benötigt, da die Kamera in Anlehnung an diesen arbeitet. Die notwendigen Eckdaten der Empfehlung sind hier zusammengefasst um das Verständnis des CAM-Moduls und der korrigierten Synchronisierung zu erleichtern.
Die digitalen Videodaten, welche neben den Farbinformationen auch die Digitalen-Blankings und Timing-Referenz-Signale enthalten, werden über einen parallelen 8-Bit-Bus übertragen. Die Taktrate dieser Schnittstelle beträgt 27 MHz.

Die Farbinformationen werden im YUV[1] 4:2:2 Format [Int11] kodiert und setzen sich dementsprechend aus den Teilkomponenten

$$U_0, Y_0, V_0, Y_1, U_2, ...$$

zusammen, wobei der Index die Pixelnummer repräsentiert. In diesem Beispiel werden für die Pixel 0 und 1 die gleichen Farbwerte (U,V) aber unterschiedliche Luminanzwerte (Y) verwendet. Durch die Fortsetzung dieser Kodierung über alle folgenden Pixel, wird eine Datenreduzierung von 8-Bit pro Pixel erreicht.
Die Timing-Empfehlung baut auf dem Zeilensprungverfahren auf, so dass sich ein Gesamt-

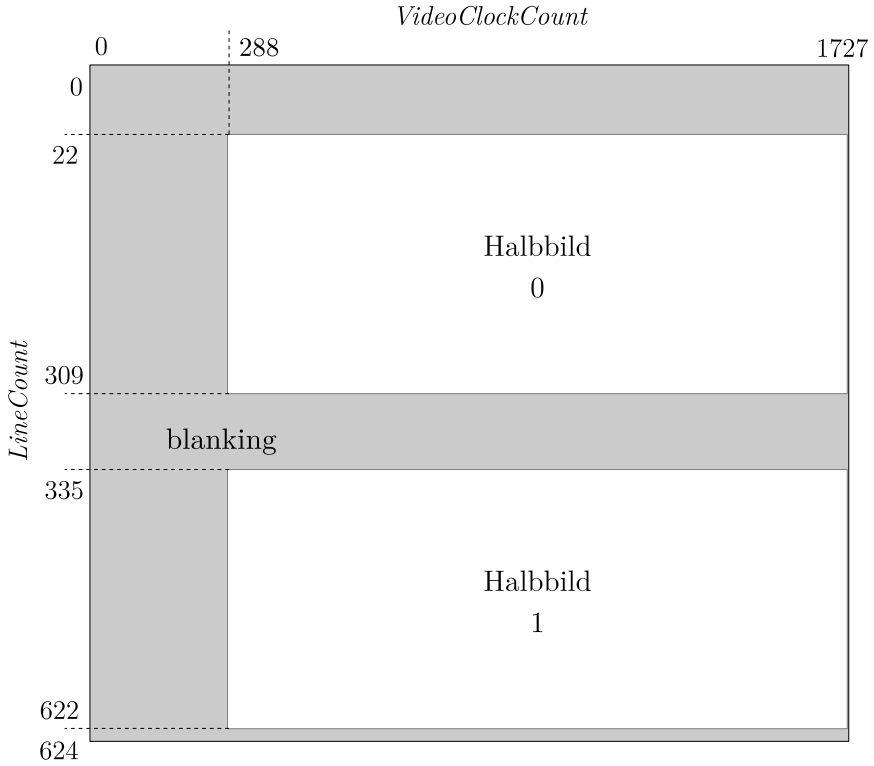


Abbildung 3.1: Der Schematische-Aufbau eines Bildes nach dem ITU-BT.656 Standard (625 Linien) zeigt die Halbbilder(weiß) und die Videoblankings (grau). Der *VideoClock-Count* wird mit 27 MHz inkrementiert. Der *LineCount* wird nach jeder Zeile erhöht.

bild aus zwei Halbbildern zusammensetzt, welche nacheinander, mit Digitalen-Blankings dazwischen, übertragen werden. Die Anzahl der Zeilen der Halbbilder und der Digitalen-Blankings sind im Standard definiert. Zudem ist auch die Größe einer Zeile bestimmt, die

[1]YUV steht für das Videosignal, die äquivalenten Videodaten werden als YC_bC_r bezeichnet. Für den Rest dieser Arbeit werden auch die digitalen Informationen vereinfacht als YUV bezeichnet.

neben dem aktiven Bereich eines Halbbildes auch aus Digitalen-Blankings besteht. Unter Berücksichtigung aller relevanten Größen ergibt sich der schematische Aufbau eines Gesamtbildes wie in Abbildung 3.1 zu sehen.

Zu Beginn jeder Zeile (*VideoClockCount* 0-4) und an der Stelle 284-287(*VideoClockCount*) werden die Synchronisations-Signale End of Active Video(EAV) und Start of Active Video(SAV) übermittelt. In den aktiven Bereichen der Halbbilder werden die Farbinformationen nach dem YUV 4:2:2 Format übermittelt, wodurch sich der beispielhafte Aufbau wie in Tabelle 3.1 ergibt. Insgesamt setzt sich der aktive Bereich eines Vollbildes aus 720x576 Pixel zusammen, mit einer Übertagungsdauer von $\frac{1}{27\text{MHz}} \cdot 1728 \cdot 625 = 40\text{ms}$ pro Bild ergibt sich eine Bildwiederholrate von 25 Hz(576i[2]/25).

VideoClockCount	0-3	4-283	284-287	288	289	290	291	292-1727
Typ	EAV	Blanking	SAV	U_0	Y_0	V_0	Y_1	...

Tabelle 3.1: Schematischer-Aufbau einer Bildzeile

Ein- und Ausgangsspezifikation des CAM-Moduls
Das CAM-Modul stellt die von der Kamera bereitgestellten Daten als Videodaten und Pixelstream zur Verfügung (s. Abb.3.2). Die Spezifikation dieser Ein- und Ausgänge ist die Grundlage für die Korrektur des CAM-Moduls.
Eingang:
Die verwendete Kamera liefert die Daten in einem Format, das zwar auf dem ITU-BT.656 Standard basiert[Vid12] jedoch in drei Punkten deutlich abweicht: Aufgrund einer höheren Anzahl an Pixeln in horizontaler Richtung (736 Pixel[Mas14]) wird eine abweichende Taktrate von 28,375 MHz [Vid12] verwendet. Auch die Anzahl der aktiven Linien weicht mit 572[Mas14] vom Standard ab.
Ausgänge:
Der Video-DAC, der das Videosignal aus den Videodaten erzeugt, setzt in der getroffenen Konfiguration die Einhaltung des ITU-BT.656 Standards voraus [Ana15].
Der Pixelstream ist die Bildquelle für die Pixel-Pipeline und andere Bildverarbeitungseinheiten. Er ist durch einen parallelen 24-Bit-Bus für die Farbinformationen, einen parallelen 18-Bit-Bus für die zugehörige Pixelkoordinate und den YUV-Takt realisiert. Zusätzlich wird über das Visiblesignal der aktive Bereich gekennzeichnet. Die Größe der Bilder ist auf 360x288 Pixel festgelegt, wodurch die gesamte Bildverarbeitung auf jedem Halbbild ausgeführt wird. Diese Spezifikation erhöht die Bildwiederholrate auf 50 Hz.

[2]i:„interlaced“, Zeilensprungverfahren

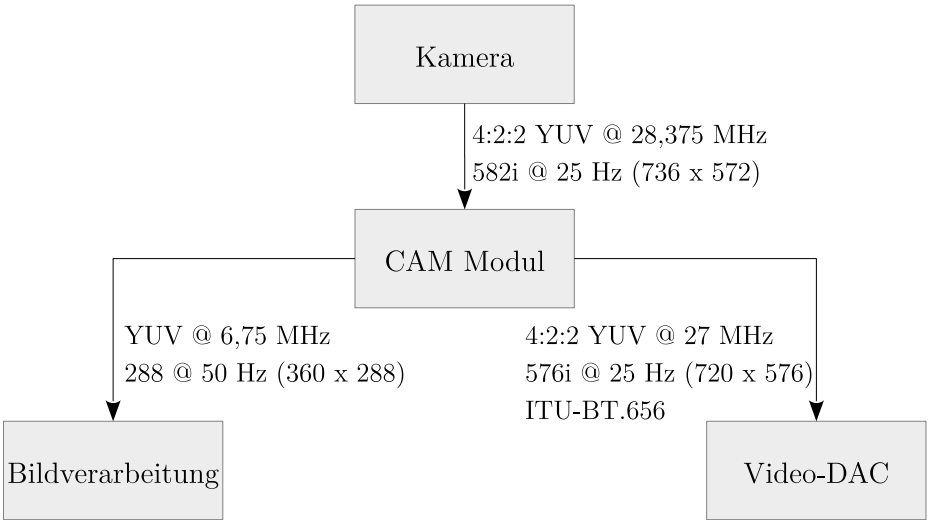


Abbildung 3.2: Die Aufgabe des CAM-Moduls ist das Interpretieren und Erzeugen der gezeigten Ein- und Ausgangsspezifikationen.

Realisierung und Korrektur des CAM-Moduls
Um die geforderten Ein- und Ausgangsspezifikationen zu erfüllen, wird das CAM-Modul wie folgt realisiert und modifiziert. Im ersten Schritt werden die Daten der Kamera eingelesen und daraus die Signale *HSync* und *VSync* erzeugt, welche im Bereich der aktiven Halbbilder "0" und sonst "1" sind. Im zweiten Schritt wird jede aktive Linie in einen Linien-Buffer gespeichert. Bei diesem Schritt werden die ersten 6 und die letzten 10 Pixel jeder Linie ignoriert, wodurch die horizontale Auflösung von 736 auf die geforderten 720 Pixel reduziert wird. Der Linien-Buffer hat 360 Speicherstellen mit jeweils 32 Bit (U_n, Y_n, V_n, Y_{n+1}) und ist in zweifacher Ausführung vorhanden. Die beiden Linien-Buffer werden wechselseitig zum Speichern und Auslesen der Werte verwendet, wodurch die Differenz der Taktraten (27 MHz; 28,375 MHz) kompensiert wird.
Die Generierung des Video-Signals nach dem Vorbild in Abbildung 3.1 erfordert eine einmalige Synchronisierung des CAM-Moduls mit der Kamera beim Starten des Roboters. In der bisherigen Variante des CAM-Moduls wurde die Synchronisierung fälschlicherweise nach dem ersten Halbbild (0) durchgeführt, was Fehler in der Bildwiedergabe und -verarbeitung verursachte. Zudem war das CAM-Modul auf die im Standard definierten 576 Zeilen ausgelegt.
Die Modifikation des CAM-Moduls korrigiert den Synchronisierungsfehler, indem der folgende Ablauf verwendet wird:

1. Warten auf Halbbild 0
2. Warten auf vertikales Blanking
3. Warten auf Halbbild 1

4. Warten auf vertikales Blanking
5. 4 Zeilen zählen
6. Synchronisiert

Die Grund für das Warten von 4 Zeilen setzt sich aus den folgenden Aspekten zusammen:

- Im Vergleich zum Standard (Abb.3.2) sind in den Bildern der Kamera die jeweils erste und letzte Linie eines Halbbildes nicht aktiv(Blanking).
  - Zeilenverzögerung: 1
- Die letzten beiden Linien laut ITU-BT.656 Standard sind Blanking.
  - Zeilenverzögerung: 2
- Der Linien-Buffer verursacht eine Verzögerung um eine Linie.
  - Zeilenverzögerung: 1

Nach der Synchronisierung wird die Erzeugung der Videodaten gestartet. Hierfür wird der ITU-BT.656 Standard mit dem regulären 27 MHz Takt verwendet. Abhängig von den Variablen *VideoClockCount* und *LineCount* werden die Digitalen-Blankings, die Sychronisations-Daten oder die Farbwerte generiert und ausgegeben. Die notwendigen Farbinformationen zum Erzeugen der Halbbilder werden aus dem Linien-Buffer ausgelesen und mit etwaigen Overlays verknüpft. Im Gesamten ergibt sich der zeitliche Verlauf der Kameradaten im Vergleich zum ITU-BT.656 Standard (Videodaten) wie in Abbildung 3.3 zu sehen. Die beiden fehlenden Linien in den Halbbildern der Kamera werden durch den Linien-Buffer ausgeglichen, indem die letzte Linie eines Halbbildes dreimal ausgegeben wird. Die dadurch resultierende Veränderung des Bildes wirkt sich gegebenenfalls auf die folgende Bildverarbeitung aus und muss, sofern notwendig, in der betroffenen Einheit korrigiert werden.

Parallel zu den Videodaten wird der Pixelstream erzeugt. Die Reduzierung der horizontalen Auflösung wird durch die arithmetische Mittelwertbildung eines Y-Pärchens erreicht. Die Bildinformation für ein Pixel an der Stelle n folgt somit:

$$\frac{Y_n+Y_{n+1}}{2},\ \mathrm{U_n},\ \mathrm{V_n}.$$

Die zugehörige Pixelkoordinate in horizontaler Richtung

$$x = \frac{VideoClockCount - 288}{4} \tag{3.1}$$

und vertikaler Richtung

$$y = \begin{cases} PixelCount - 22, & \text{falls } PixelCount < 320 \\ PixelCount - 335, & \text{sonst} \end{cases} . \tag{3.2}$$

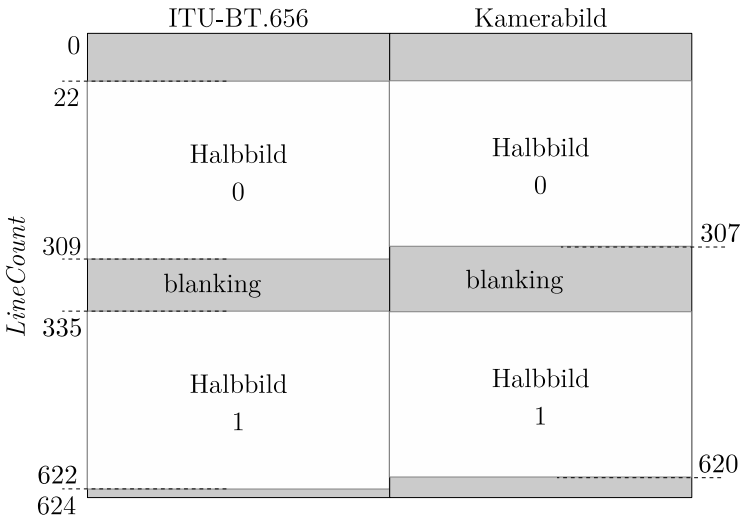


Abbildung 3.3: Durch den vorgestellten Synchronisationsprozess liegen die jeweiligen Anfänge der Halbbilder gleich auf. Durch die geringere Anzahl an Linien im Kamerabild entsteht am Ende der Halbbilder jeweils eine Lücke, die durch das dreifache Ausgeben des Linienbuffer-Inhalts kompensiert wird. Die horizontalen Blankings, wie auch die unterschiedliche horizontale Größe der Bilder werden in dieser Abbildung vernachlässigt.

Die Pixelkoordinate in vertikaler und horizontaler Richtung als auch die zugehörigen Farbwerte werden über die genannten Schnittstellen bereitgestellt. Die Bildinformationen wie auch die Pixelkoordinaten sind nur innerhalb der Halbbilder gültig, was durch das Signal *Visible* gekennzeichnet wird. Die Taktrate für den Pixelstream beträgt $\frac{27\text{MHz}}{4} = 6.75\text{MHz}$.

3.2 Modifikationen und Beschreibung der Pixel-Pipeline

Die Pixel-Pipeline ist eine konfigurierbare Echtzeit-Bildverarbeitungseinheit, welche den Pixelstream (Kap.3.1) als Bildquelle nutzt. Die Resultate der Verarbeitung werden im Pixel-RAM gespeichert und sind somit für weitere Module nutzbar. Die bisherige Variante wurde erweitert um neue Operationen zu ermöglichen. Kapitel 3.2.1 wurde zu großen Teilen aus [Wur16] übernommen und dient, durch das Beschreiben der bisherigen Variante, dem Verständnis der erweiterten Pixel-Pipeline (Kap. 3.2.2).

3.2.1 Beschreibung der existierenden Pixel-Pipeline

Die bisherige Pixel-Pipeline ermöglicht konfigurierbare geometrische und affine Bildtransformationen, deren Ergebnis ausschnittsweise im Pixel-RAM verfügbar ist. Das zugehörige Blockdiagramm ist in Abbildung 3.4 gezeigt.

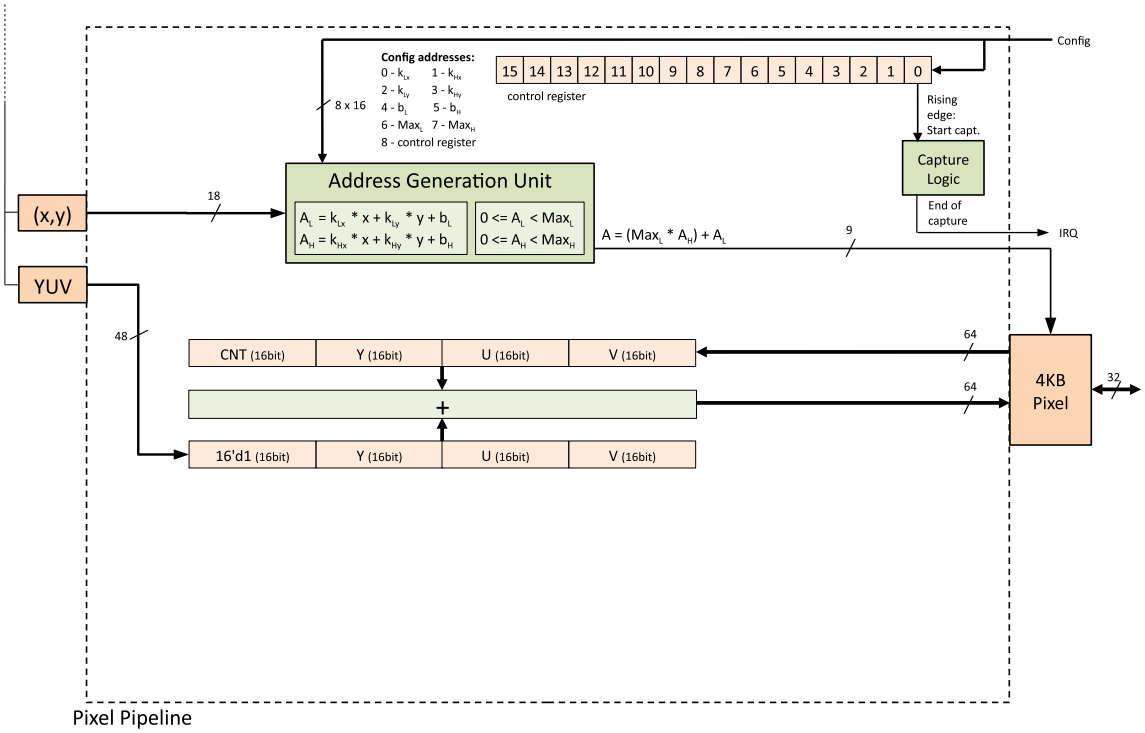


Abbildung 3.4: Durch die Address Generation Unit (AGU) in Kombination mit dem Pixel-RAM können geometrische und affine Transformationen des Eingangsbildes(Pixelstream) durchgeführt werden. Quelle: Labor-internes Dokument

Ein Pixel wird nach Abbildung 3.4 (Address Generation Unit) entsprechend durch

$$\begin{pmatrix} l \\ h \end{pmatrix} = \begin{pmatrix} k_{lx} & k_{ly} \\ k_{hx} & k_{hy} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_l \\ b_h \end{pmatrix} \tag{3.3}$$

transformiert. Die Pixelkoordinaten l und h entstehen durch die Transformation der originalen Pixelkoordinaten x und y. Die Abbildungsmatrix T (k_{lx} ... k_{hy}) beschreibt die Transformationsoperation und der Vektor $\vec{b}$ (b_l, b_h) die Verschiebung. Eine Veranschaulichung der dadurch möglichen fundamentalen Operationen ist den Abbildungen 3.5 (a-c) zu entnehmen.

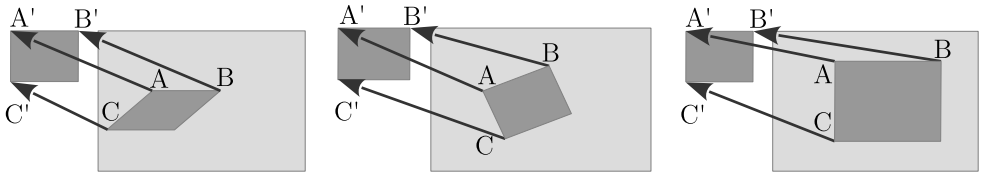


Abbildung 3.5: Das jeweils große hellgraue Feld symbolisiert das Originalbild, das jeweils links angeordnete dunkelgraue Feld das transformierte Bild, mit der Größe max_l x max_h. Die originalen Eckpunkte A-C werden jeweils zu A'-C' transformiert.[Wur16]

Abbildung 3.5 (a) zeigt eine Scherung, (b) eine Rotation und (c) eine Skalierung. Die Operationen werden durch die Abbildungsmatrizen $T_{(a)}$- $T_{(c)}$ definiert und lassen sich untereinander kombinieren. Die zusätzlichen Verschiebungen werden durch $\vec{b}$ definiert. Die zugehörigen prinzipiellen Abbildungsmatrizen [SR14] lauten:

(a) (b) (c)

$$T_{(a)} = \begin{pmatrix} 1 & k_{ly} \\ 0 & 1 \end{pmatrix} \qquad T_{(a)} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \qquad T_{(a)} = \begin{pmatrix} k_{lx} & 0 \\ 0 & k_{hy} \end{pmatrix}.$$

Über den Zusammenhang

$$A = (Max_l \cdot H) + L \tag{3.4}$$

wird die Adresse A für das Pixel-RAM berechnet. An diese Speicherstelle werden die zu der originalen Pixelkoordinate gehörenden Farbwerte gespeichert.
Die durch die Transformation entstanden Pixel werden im Folgenden Superpixel genannt. Jeder dieser Superpixel kann aus mehreren Originalpixeln bestehen, daher wird neben der aufsummierten Farbinformation auch die Anzahl der echten Pixel je Superpixel gespeichert. Um die gemittelten Farbwerte zurückzugewinnen, müssen die im RAM gespeicherten Werte durch die Anzahl der Pixel geteilt werden. Die Datenstruktur des Pixel-RAM ist demnach wie Tabelle 3.2 aufgebaut.

A	I	Wert	Superpixel
0	0	V	0
	1	U	0
	2	Y	0
	3	Pixel Anzahl	0
1	4	V	1
	5	U	1
	6	...	...

Tabelle 3.2: Die Speicherstruktur des Pixel-RAM. Pro Adresse A sind 8 Byte Speicher verfügbar. Y, U, V und die Pixel Anzahl sind somit jeweils 2 Byte groß.

Die Superpixel werden mit Hilfe des 64-Bit-Addierwerks (Abb. 3.4) berechnet, indem die Werte an der Stelle A des Pixel-RAM mit den neuen Werten addiert und wieder an die Stelle A gespeichert werden. Jede Farbinformation hat eine Größe von 16-Bit, was die maximale Anzahl an Pixeln pro Superpixel auf

$$n_{Pixel,max} = \frac{2^{16} - 1}{2^8 - 1} = 257 \tag{3.5}$$

beschränkt. Wird diese Anzahl überschritten können durch das 64-Bit-Addierwerk Überläufe in die nächst höheren Farbwerte entstehen. Eine zusätzliche Einschränkung stellt die Größe des Pixel-RAM dar, weswegen die Größe eines transformierten Bildes insgesamt nicht mehr als 512 Superpixel betragen darf.

Die Konfiguration der „Address Generation Unit“ erfolgt mittels der Transformationsmatrix M, beziehungsweise deren Elementen $k_{lx} \dots b_h$. Zusätzlich werden die Parameter max_h und max_l benötigt, welche die Größe des transformierten Bildes in Superpixeln beschreiben.

$$M = \begin{pmatrix} k_{lx} & k_{ly} & b_l \\ k_{hx} & k_{hy} & b_h \\ 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

Über die in Abbildung 3.2.1 gezeigten Register(Config addresses) werden die Parameter gesetzt. Die Elemente der Matrix M müssen zu diesem Zweck im „signed fractional“-Format vorliegen: die Parameter $k_{\dots}$ im „2.14-“ und die Parameter $b_{\dots}$ im „10.6-Format“. Bei jedem „Capture“-Befehl (Abb. 3.2.1) wird nach den oben gezeigten Regeln ein Bild zum PixelRAM hinzu addiert. Dies ermöglicht eine zeitliche Mittelwertbildung mit Hilfe der Pixelpipeline, indem die gespeicherten Bilder vor dem Aufnehmen nicht gelöscht werden.

Im Pixel-RAM bereitstehende Bilder werden durch ein Flag am IRQ-Ausgang signalisiert. Die Daten können über den 32-Bit-Bus vom Prozessor aus dem Pixel-RAM ausgelesen werden.

Mathematische Zusammenhänge
Um die Verwendung der Pixel-Pipeline zu ermöglichen, sind im Folgenden die mathematischen Zusammenhänge zwischen der Transformationsmatrix und den Bildpunkten erklärt. Die Transformationsmatrix M lässt sich aus gegebenen Punkten A-C (Matrix P)(vgl. Abb.3.5) bestimmen.

$$M = P' \cdot P^{-1} \tag{3.7}$$

$$\begin{pmatrix} k_{lx} & k_{ly} & b_l \\ k_{hx} & k_{hy} & b_h \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x'_A & x'_B & x'_C \\ y'_A & y'_B & y'_C \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{pmatrix}^{-1} \tag{3.8}$$

Für die transformierten Eckpunkte A'-C' gilt:

$$x'_A = x'_C = y'_A = y'_B = 0 \tag{3.9}$$

So folgt für die Transformationsmatrix:

$$\begin{pmatrix} k_{lx} & k_{ly} & b_l \\ k_{hx} & k_{hy} & b_h \\ 0 & 0 & 1 \end{pmatrix} = \frac{1}{|P|} \cdot \begin{pmatrix} x'_B (y_C - y_A) & x'_B (x_A - x_C) & x'_B (x_C y_A - x_A y_C) \\ y'_C (y_A - y_B) & y'_C (x_B - x_A) & y'_C (x_A y_B - x_B y_A) \\ 0 & 0 & |P| \end{pmatrix} \tag{3.10}$$

$$\text{mit: } |P| = x_A (y_B - y_C) + x_B (y_C - y_A) + x_C (y_A - y_B) \tag{3.11}$$

$$x'_B = max_l \tag{3.12}$$

$$y'_C = max_h \tag{3.13}$$

Um aus einer gegebenen Matrix M die zugehörigen Bildpunkte zu berechnen ergibt sich:

$$P = P' \cdot M^{-1} \tag{3.14}$$

Mit den Bedingungen aus den Formeln (3.9),(3.12) und 3.13 kann P' vereinfacht werden.

$$P' = \begin{pmatrix} 0 & max_l & 0 \\ 0 & 0 & max_h \\ 1 & 1 & 1 \end{pmatrix} \tag{3.15}$$

Die Eckpunkte können über die Konfigurationsparameter der Pixel-Pipeline berechnet werden.

$$\begin{pmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{pmatrix} = \frac{1}{k_{lx}k_{hy} - k_{ly}k_{hx}} \cdot \begin{pmatrix} k_{ly}b_h - k_{hy}b_l & k_{hy}\left(max_l - b_l\right) + k_{ly}b_h & -k_{ly}\left(max_h - b_h\right) - k_{hy}b_l \\ k_{hx}b_l - k_{lx}b_h & -k_{hx}\left(max_h - b_l\right) - k_{lx}b_h & k_{lx}\left(max_l - b_h\right) - k_{hx}b_l \\ k_{lx}k_{hy} - k_{ly}k_{hx} & k_{lx}k_{hy} - k_{ly}k_{hx} & k_{lx}k_{hy} - k_{ly}k_{hx} \end{pmatrix} \tag{3.16}$$

3.2.2 Erweiterung der Pixel-Pipeline

Mit der erweiterten Variante der Pixel-Pipeline sind neben Bildtransformationen auch Histogrammbildung, Differenzbildung und Farberkennungen möglich. Die Auswahl des Modus erfolgt über das „Control Register“ (Abb. 3.6) und ist somit eine Erweiterung des „Capture“-Befehls aus Kapitel 3.2.1. Die Realisierung dieser zusätzlichen Modi erfolgt über ein erweitertes Rechenwerk, den sogenannten Color-Classifier und eine alternative Pixel-RAM-Adressierung.

Erweiterung des Rechenwerks
Die Erweiterung des Rechenwerks eröffnet die Möglichkeit der Differenzbildung und somit das Erfassen von zeitliche Veränderungen. Neben dem 64-Bit-Addierwerk ist ein dreifach 16-Bit-Subtrahierer verfügbar, der die neuen Werte von den Werten aus dem Pixel-RAM abzieht. Durch die 16 Bit Segmentierung wird ein Überlaufen in höhere Bereich verhindert. Das Ergebnis der Rechnung liegt im Zweierkomplement vor, wodurch sich die maximal mögliche Anzahl an Pixeln pro Superpixel verändert. Eine kombinierte Nutzung der AGU, der Subtraktion und den regulären Farbwerten(YUV) beschränkt die Pixel Anzahl auf

$$n_{Pixel,max} = \left\lfloor \frac{2^{15} - 1}{2^8 - 1} \right\rfloor = 128. \tag{3.17}$$

Um die Information der oberen 16 Bit des Pixel-RAM an der jeweiligen Adresse zu erhalten, werden diese beim Verwenden des Subtrahiermodus nicht verändert. So bleiben beispielsweise Informationen über die Anzahl der Pixel im Superpixel erhalten.
Die Addition als auch die Subtraktion werden stets beide ausgeführt, wobei nur das mit dem Modus gewählte Ergebnis im Pixel-RAM gespeichert wird. Um den Rechenmodus zu wählen, wird das „Control Register“-Bit 1 wie folgt gesetzt:

0: Addition

1: Subtraktion

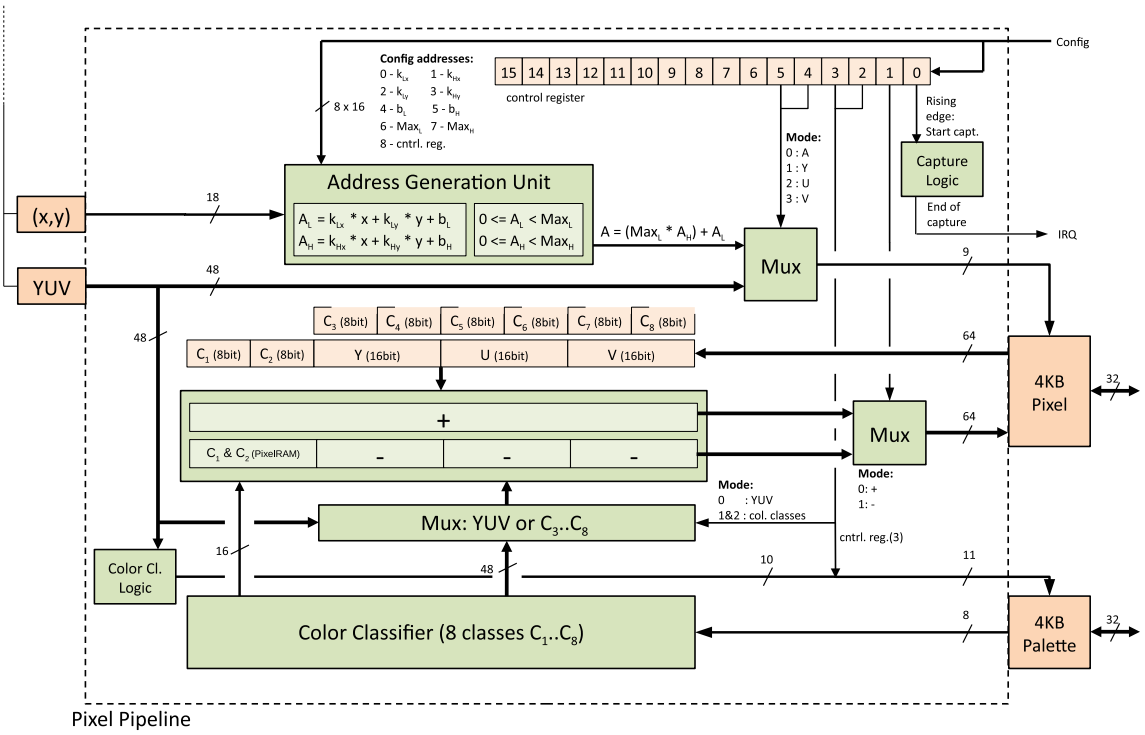
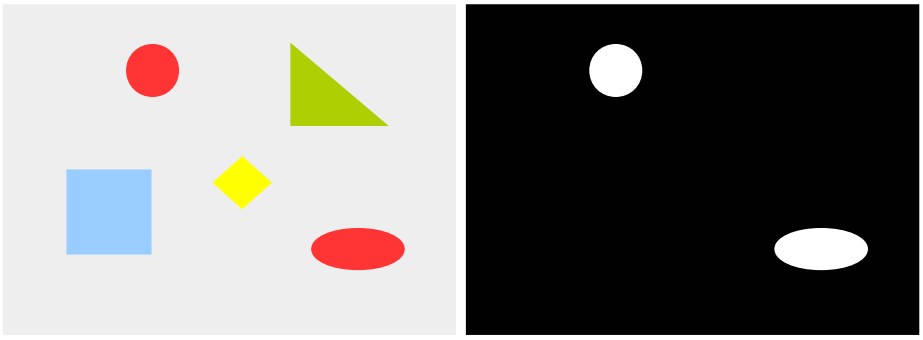


Abbildung 3.6: Die Erweiterung der Pixel-Pipeline ermöglicht durch die Wahl des Capture-Modus (cntrl. reg.) Affine Tranformationen, Differenzbildungen, Farbklassifizierungen und Histogrammbildungen.

Color-Classifier

Die Grundlage vieler farb-basierter Bildverarbeitungsalgorithmen ist das Klassifizieren der Pixel in Farbzugehörigkeiten. Eine Echtzeit-Klassifizierungseinheit ist in der erweiterten Pixel-Pipeline durch den Color-Classifier realisiert.

Um Pixel nach ihrer Farbe klassifzieren zu können, ist die Defnition sogenannter Farbklassen (Color-Classes) notwendig. Eine Farbklasse ist ein bestimmter Bereich, welcher einen Farbraum in Zugehörigkeit und Nicht-Zugehörigkeit unterteilt. Pixel deren Farbwerte Teil dieses Bereiches sind werden mit "1" alle anderen mit "0" gekennzeichnet. Das Festlegen und Finden solcher Bereiche ist in [Met16] gezeigt. Ein Beispiel einer solchen Klassifizierung ist in den Abbildungen 3.7 (a) und (b) gezeigt, wobei das Originalbild(a) pixelweise auf die Farbeklasse rot geprüft wird. Bei Übereinstimmung werden in der sogenannten Colormap(b) die Pixel an dieser Stelle auf "1" (weiß) gesetzt.



(a) (b)

Abbildung 3.7: Die Klassifizierung des Originalbildes (a) mit der Farbklasse Rot ergibt die Colormap in (b). Die weißen Pixel in der Colormap signalisieren Übereinstimmung ("1").

Das für die Pixel-Pipeline verwendete Verfahren basiert auf den Ausarbeitungen in [BBV00], nach denen sich die Zugehörigkeit eines Farbwertes (Y, U, V) zu einer Farbklasse (C) nach Formel (3.18) definiert.

$$C(Y,U,V) = Y_{cc}(Y) \wedge U_{cc}(U) \wedge V_{cc}(V) \tag{3.18}$$

Für jede Farbkomponente ist ein separater Bereich definiert (Y_{cc}, U_{cc}, V_{cc}). Wenn ein Farbwert in allen drei Unterbereichen eine Übereinstimmung aufweist ist er Teil der Farbklasse ("1"). Eine mögliche Definition der Unterbereiche ist in Abbildung 3.8 zusehen.

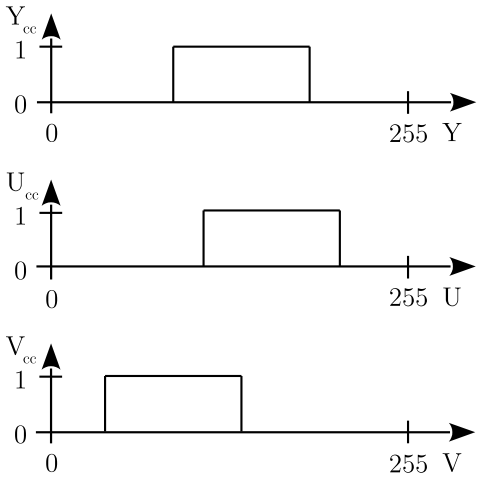
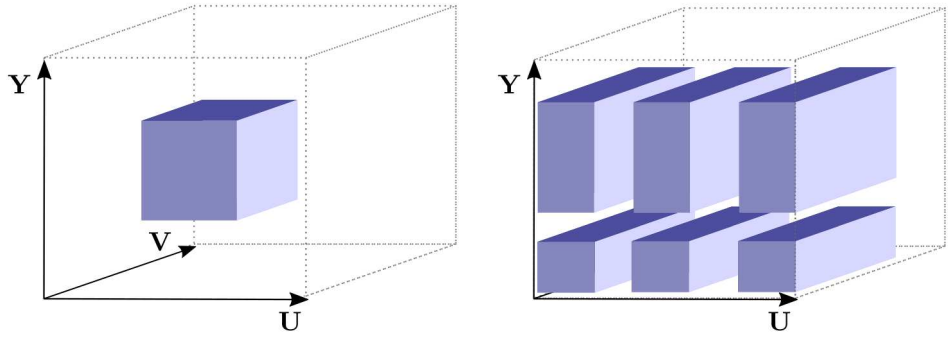


Abbildung 3.8: Die einzelnen Bereiche (Y_{cc}, U_{cc}, V_{cc}) sind "1" in der Farbklasse und "0" sonst. [BBV00]

Die Und-Verknüpfung dieser Teilbereiche ergibt die Farbklasse, einen Definitionsbereich im Farbraum. Durch die voneinander unabhängigen Prüfungen sind ausschließlich Farbklassen wie in den Abbildungen 3.9(a),(b) möglich. Sie bestehen aus einem Grundquader dessen Kanten parallel zu den Achsen des Farbraums liegen. Dieser Quader kann durch Ausschnitte getrennt sein, wobei jeder dieser Ausschnitte parallel zu einer Ebene des Farbraums verläuft und den kompletten Raum teilt (s. Abb. 3.9(b)).



(a) (b)

Abbildung 3.9: Die schematische Darstellung zweier Farbklassen ((a),(b)) im YUV-Farbraum. Der in (a)[BBV00] gezeigte Würfel setzt sich aus der Und-Verknüpfung der in Abbildung 3.8 gezeigten Teilbereiche zusammen.

Mit der Pixel-Pipeline sind bis zu 8 Farbklassifizierungen gleichzeitig möglich, welche zusätzlich logisch miteinander verknüpft werden können. Diese Option eröffnet die Möglichkeit Farbklassen mit komplexeren Bereichen zu definieren.
Die Realisierung der Farbklassifizierung erfolgt über eine konfigurierbare Look-Up-Table (LUT), die sogenannte Color-Class-Palette (Abb. 3.6), mit 2048 8-Bit-Speicherplätzen, wobei jede Adresse einen Farbwert repräsentiert. Die 8-Bit zeigen bei welchen der Farbklassen dieser Farbwert Teil der Klasse ist. Mittels der Konfiguration der Color-Class-Palette (CCP) werden die Farbklassen definiert. Der prinzipielle Aufbau der CCP ist in Tabelle 3.3 gezeigt.

$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$	$\mathbf{C_4}$	$\mathbf{C_5}$	$\mathbf{C_6}$	$\mathbf{C_7}$	$\mathbf{C_8}$	**Wert**	**Adresse**	**CCC**
1	0	0	1	0	0	1	1	**0**	**Y 0 ... 255**	**0**
...	...	...	...	...	...	...	...	**...**		
0	0	1	0	1	1	0	1	**255**		
0	1	0	1	1	0	0	1	**0**	**U 256 ... 511**	
...	...	...	...	...	...	...	...	**...**		
1	0	0	1	0	1	0	1	**255**		
0	0	0	0	0	0	0	1	**0**	**V 512 ... 767**	
...	...	...	...	...	...	...	...	**...**		
0	1	1	0	1	0	0	1	**255**		
1	1	1	0	0	0	1	0	**0**	**L 768 ... 1023**	
...	...	...	...	...	...	...	...	**...**		
1	0	0	0	0	1	0	1	**255**		
1	1	0	0	1	1	0	0	**0**	**Y 1024 ...**	**1**
...	...	...	...	...	...	...	...	**...**		
...	...	...	...	...	...	...	...	**...**	**...**	**...**

Tabelle 3.3: Der schematische Aufbau der Color-Class-Palette zeigt den Zusammenhang zwischen den Farbklassen (C_1-C_8)und den jeweiligen Farbwerten (Y,U,V). Für eine Farbwerte Kombination von (255,0,255) würde sich für den Logik Bereich (L) der Wert 9 ergeben, was der Adresse 777 des CCP entspricht. Die Einträge der Farbklassen an dieser Stelle sind das Ergebnis der Klassifizierung.

Im Detail besteht die Farbklassifizierung aus den folgenden Teilschritten:

1: $Y_{cc8} = CCP\,[Y]$

2: $U_{cc8} = CCP\,[256 + U]$

3: $V_{cc8} = CCP\,[512 + V]$

4: $C_{1...8} = CCP\,[768 + (Y_{cc8} \wedge U_{cc8} \wedge V_{cc8}))]$ ($\wedge$ hier bitweise Operation)

Der zusätzliche Index 8 soll die 8 Farbklassen der Pixel-Pipeline symbolisieren, deren Ergebnis in den jeweiligen Werten enthalten sind. $CCP\,[X]$ steht hierbei für den Inhalt der CCP an der Stelle X. Die bitweise verknüpften Teilergebnisse aus 1-3 dienen als Adresse für die Logik-LUT, wodurch sich die Farbklassen untereinander kombinieren lassen. Der Adressbereich 0-1023 (LUTs:Y,U,V,L) wird als Color-Class-Container (CCC) 0 bezeichnet. Durch den zusätzlichen CCC 1 können bis zu 16 Farbklassen definiert werden. Über die Wahl des Modus kann zwischen den CCCs umgeschaltet werden, um so die 8 Farbklassen zu bestimmen, die zur Klassifizierung verwendet werden.
Die Farbklassifizierung wird auf jeden Pixel angewendet und das Ergebnis mit Hilfe des Rechenwerks zum Pixel-RAM hinzugerechnet. Zu diesem Zweck wird der 8-Bit-Wert auf einen 64-Bit-Wert erweitert:

Bit	63-57	56	55-49	48	47-41	40	39-33	32	31-25	24	...
Wert	0	C_1	0	C_2	0	C_3	0	C_4	0	C_5	...

Die Farbklassifizierung stellt eine Alternative zum normalen Farbmodus (YUV) dar. Im YUV-Modus werden jedoch auch die oberen zwei Farbklassen (C_1,C_2) verwendet, wodurch die Anzahl der Pixel pro Superpixel gezählt und gleichzeitig eine Farbklassifizierung durch geführt werden kann. Um die Anzahl der Pixel zu zählen, muss die betreffende Farbklasse auf den kompletten Farbraum definiert werden, so dass jeder Farbwert eine Addition um Eins verursacht.
Die Wahl des Farbmodus erfolgt über die „Control Register“-Bits 3 und 2:

00: YUV

01: CC0

10: CC1

11: nicht definiert

Pixel-RAM-Adressierung
In der bisherigen Variante wird die Pixel-RAM-Adressierung vollständig von der AGU übernommen. Um Histogramme des gesamten Bildes ermitteln zu können, verfügt die erweiterte Variante über eine alternative Adressierung.
Die Farbwerte (Y,U,V) können wahlweise als Adresse für das Pixel-RAM verwendet werden. In Kombination mit der Farbklassifizierung sind so Histogrammbildungen möglich. Die verwendete Farbklasse muss zu diesem Zweck, wie auch für das Zählen der Pixeln, vollständig zu Eins definiert werden. In den Speicherstellen 0-255 der Pixel-RAM ergibt sich so die Anzahl der Pixel mit dem jeweiligen Farbwert (0-255).
Die Wahl des Adressierungsmodus wird über die die „Control Register"-Bits 5 und 4 vorgenommen:

00: AGU

01: Y

10: U

11: V

Zusammenfassung und Überblick über die Modi der erweiterten Pixel-Pipeline

Durch die Kombination der vorgestellten Teilmodi, für das erweiterte Rechenwerk, die Farbklassifizierung und die alternative RAM-Adressierung, ergibt sich eine vielfältige Funktionsweise der erweiterten Pixel-Pipeline. Der durch die Wahl der Bits resultierende Bit-Vektor wird im Folgenden dezimal interpretiert. Eine Zusammenfassung der Modi findet sich in Tabelle 3.4.
Die wesentliche Unterscheidung der Modi hängt von der Art der Pixel-RAM-Adressierung ab, so ergeben sich die Typen (Tabelle 3.4) zu affinen Bildtransformationen (Modi: 1, 3, 5, 7, 9, 11), Y-Histogrammen (21, 23, 35, 27), U-Histogrammen (37, 39, 41, 43) und V-Histogrammen (53, 55, 57, 59). Im Falle der Histogramme kann zwischen den beiden Color-Classifier-Containern gewählt werden und es lassen sich Differenz-Histogramme bilden. Die Bildung von Histogrammen baut auf der Farbklassifizierung auf. Eine Kombination mit dem normalen Farbmodus ist zu vermeiden. Durch die Wahl der AGU als Pixel-RAM-Adressierungs-Quelle werden affine Bildtransformationen ermöglicht, die entweder im normal Farbmodus (YUV) oder mit einem der beiden Color-Classifier-Containern realsiert werden. Für jede Kombination kann zusätzlich die Rechenart ausgewählt werden, um auch Differenzbilder zu ermöglichen. Die grauen Bereich in Tabelle 3.4 resultieren zum einen aus den zu vermeidenden Kombinationen zwischen Histogrammen und dem normalen Farbmodus und zum anderen aus dem nicht definierten Modus der Farbmoduswahl (11).

Übersicht über die möglichen Modi der erweiterten Pixel-Pipeline

Typ	Modus	Pixel-RAM-Adressierung				Farbmodus			Rechenart	
		V	U	Y	AGU	CCC1	CCC0	YUV	-	+
affine Bildtransformationen	1	-	-	-	X	-	-	X	-	X
	3	-	-	-	X	-	-	X	X	-
	5	-	-	-	X	-	X	-	-	X
	7	-	-	-	X	-	X	-	X	-
	9	-	-	-	X	X	-	-	-	X
	11	-	-	-	X	X	-	-	X	-
...	13-19	-	-	-	-	-	-	-	-	-
Y Histogramm	21	-	-	X	-	-	X	-	-	X
	23	-	-	X	-	-	X	-	X	-
	25	-	-	X	-	X	-	-	-	X
	27	-	-	X	-	X	-	-	X	-
...	29-35	-	-	-	-	-	-	-	-	-
U Histogramm	37	-	X	-	-	-	X	-	-	X
	39	-	X	-	-	-	X	-	X	-
	41	-	X	-	-	X	-	-	-	X
	43	-	X	-	-	X	-	-	X	-
...	45-51	-	-	-	-	-	-	-	-	-
V Histogramm	53	X	-	-	-	-	X	-	-	X
	55	X	-	-	-	-	X	-	X	-
	57	X	-	-	-	X	-	-	-	X
	59	X	-	-	-	X	-	-	X	-
...	61-63	-	-	-	-	-	-	-	-	-

Tabelle 3.4: Die möglichen Kombination der vorgestellten Erweiterungen der Pixel-Pipeline ergeben den jeweiligen Modus (hier dezimal). Grau hinterlegte Felder sind Modi die nicht definiert sind oder nicht verwendet werden sollten.

Kapitel 4

Flächenerkennung und -Lokalisierung

Rechenaufwendige Algorithmen, wie zum Beispiel Formerkennung oder Gesichtserkennung, können in ihrer Rechenzeit beschleunigt werden, indem zu untersuchende Bereiche bereits vorab eingeschränkt werden. Die Einschränkung auf mögliche Flächen kann durch die Klassifizierung eines Bildes hinsichtlich unterschiedlicher Merkmale erreicht werden. Pixel oder Bereiche des Bildes, welche die gewünschten Merkmale erfüllen werden mit "1" gekennzeichnet die restlichen mit "0" (vgl. Abb.3.7). Die so erzeugten Bilder werden im Folgenden als Bitmap (BMP) bezeichnet. Die Pixel und Flächen mit dem Wert "1" als Merkmals-Flächen und -Pixel oder weiße Flächen und Pixel bezeichnet.
Das Erkennen und Lokalisieren zusammenhängender Merkmals-Flächen wird in diesem Kapitel näher erläutert. Die notwendigen Grundlagen in Kapitel 4.1 basieren auf den Ausarbeitungen in [Wur16].

4.1 Grundlagen der Flächenerkennung und -Lokalisierung

Die meisten Algorithmen zur Flächenerkennung untersuchen jeden Pixel einer Bitmap auf "1" oder "0". Ausgehend von Merkmals-Pixeln werden die benachbarten Pixel hinsichtlich einer "1" untersucht. Diese Verfahren sind aufgrund ihrer zahlreichen Operationen pro Pixel sehr zeitintensiv. Das hier verwendetet Verfahren zur Flächenerkennung und -Lokalisierung wurde für eine schnelle und im Ablauf unterbrechbare Struktur entworfen, die auf der Bildung von Summenvektoren basiert.

Ermittlung der horizontalen Grenzen zur Flächenerkennung
Im ersten Schritt werden alle Pixel einer Spalte des Bitmaps aufsummiert und so der Vektor $x_{sum}(i)$ ermittelt.

$$x_{sum}(i) = \sum_{l=y_{min}}^{y_{max}} BMP(i,l) \tag{4.1}$$

Im zweiten Schritt wird der Vektor auf etwaige steigende und fallende Flanken geprüft,welche den Beginn beziehungsweise das Ende einer potentiellen weißen Fläche repräsentieren. Die beiden Schritte sind Teil der Funktion xGrenzenFinden (Algorithmus 4.1).

Algorithmus 4.1 Ermitteln der Flanken

```
1:  function xGrenzenFinden(xMin, xMax, yMin, yMax, xCnt, yCnt)
2:      for i = xMin → xMax do                          ▷ Berechnen des Summenvektors
3:          for j = yMin → yMax do
4:              xSum[i] ← xSum[i] + BMP[i][j]
5:          end for
6:      end for
7:      z1 ← 0
8:      n ← 0
9:      for i = xMin → xMax do                          ▷ Suchen möglicher Flanken
10:         if xSum[i] ≥ xSchwelle then
11:             z ← 1
12:         else
13:             z ← 0
14:         end if
15:         flanke ← z − z1          ▷ Flanke: 1 bei steigend; -1 bei fallend; 0 sonst
16:         if e = 1 then
17:             xGrenzeMin[n] ← i − 1
18:         else if e = 0 and z = 1 then
19:             xGrenzeMax[n] ← i
20:         else if e = −1 then
21:             yGrenzeMin[n] ← yMin
22:             yGrenzeMax[n] ← yMax
23:             xCntO[n] ← xCnt + 1
24:             yCntO[n] ← yCnt + 1
25:             n ← n + 1                                ▷ Neues Grenzpärchen
26:         end if
27:         z1 ← z
28:     end for
29:     return xGrenzeMin, xGrenzeMax, yGrenzeMin, yGrenzeMax, xCntO, yCntO
30: end function
```

Die Funktionsparameter *xMin,...,yMax* geben den zu untersuchenden Bereich an. Diese Werte werden zu Beginn so gewählt, dass das vollständige Bitmap untersucht wird. In Zeile 10 des Algorithmus wird die Variable xSchwelle verwendet, über welche sich einstellen lässt, ab wie vielen Merkamlspixel eine Fläche erkannt werden soll. Das Resultat des Algorithmus sind *n* Grenzpaare(*xGrenzeMin,xGrenzeMax*), welche die Grenzen möglicher weißer Flächen angeben. Die ermittelten Grenzpaare ergeben in Kombination mit den Eingangsparametern *yMin* und *yMax* neue zu untersuchende Bereiche, die von der Funktion zurückgegeben werden. Abbildung 4.1 (a) zeigt den Verlauf von $x_{sum}(i)$ am Beispiel von

Abbildungen 3.7.

Ermittlung der vertikalen Grenzen zur Flächenerkennung
Für jeden dieser Bereiche wird äquivalent zum gezeigten Vorgehen der Vektor $y_{sum}(j)$ über die Zeilen der Bitmap gebildet.

$$y_{sum}(j) = \sum_{l=x_{min}}^{x_{max}} BMP(l, j) \tag{4.2}$$

Die Funktion yGrenzenFinden folgt dem Aufbau in Algorithmus 4.1, wobei jedes x innerhalb des Funktionsblocks durch ein y zu ersetzen ist. Am Beispiel von Abbildung 4.1 (a) ergeben sich die beiden Verläufe $y_{sum}(j)$ in (b) und (c).

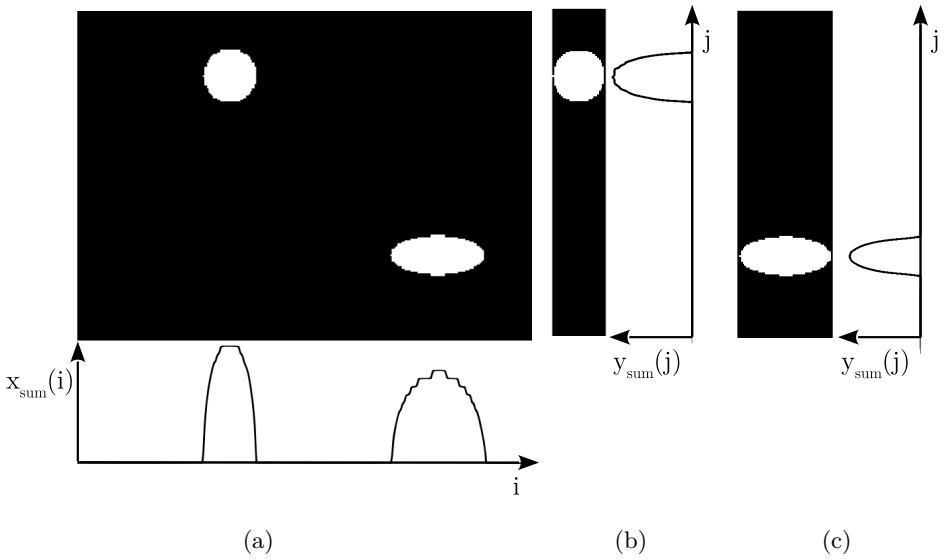


Abbildung 4.1: Das Bilden der Summenvektoren in (a)-(b) ermöglicht die Lokalisierung der weißen Flächen. Aus den Grenzen in (a) ergeben sich die Teilbereiche (b) und (c).[Wur16]

Rekursive Struktur zur Flächenerkennung
Um Bitmaps mit komplexer angeordneten Merkmals-Flächen zu analysieren müssen die die beiden Funktionen in eine rekursive Struktur eingeordnet werden. Das Kernstück dieser Struktur bildet eine Warteschlange („queue“), in welcher zu Beginn der gewünschte Anfangsbereich gespeichert werden muss. Die beiden Zählvariablen *xCnt* und *yCnt* werden für diesen ersten Bereich auf "0"gesetzt. Beim Starten des Algorithmus ist dieser Anfangsbereich als ältester Eintrag der Warteschlange zu verstehen. Im Folgenden ist der Zyklus der Struktur erklärt.
Die ältesten Einträge der Warteschlange werden hinsichtlich ihrer Zählvariablen *xCnt* und *yCnt* geprüft. Diese geben an welche der beiden Funktionen (x/yGrenzeFinden) öfter auf

den Bereich zugegriffen hat. Der Funktion mit dem kleineren Zählwert wird der Bereich zum Überprüfen übergeben, welche wiederum ihre Ergebnisse der Warteschlange übergibt. Neben den neuen ermittelten Bereichen wird auch der ursprüngliche Bereich gesichert. Bei jedem Durchlauf wird der neue Bereich mit alten Bereich verglichen. Sind diese sehr ähnlich, gilt der Bereich als lokalisierte Fläche und wird aus dem Zyklus genommen. Sind keine Flächen mehr in der Warteschlange verfügbar wird der Zyklus beendet, da alle möglichen Flächen lokalisiert wurden. Abbildung 4.2 veranschulicht dieses Vorgehen.

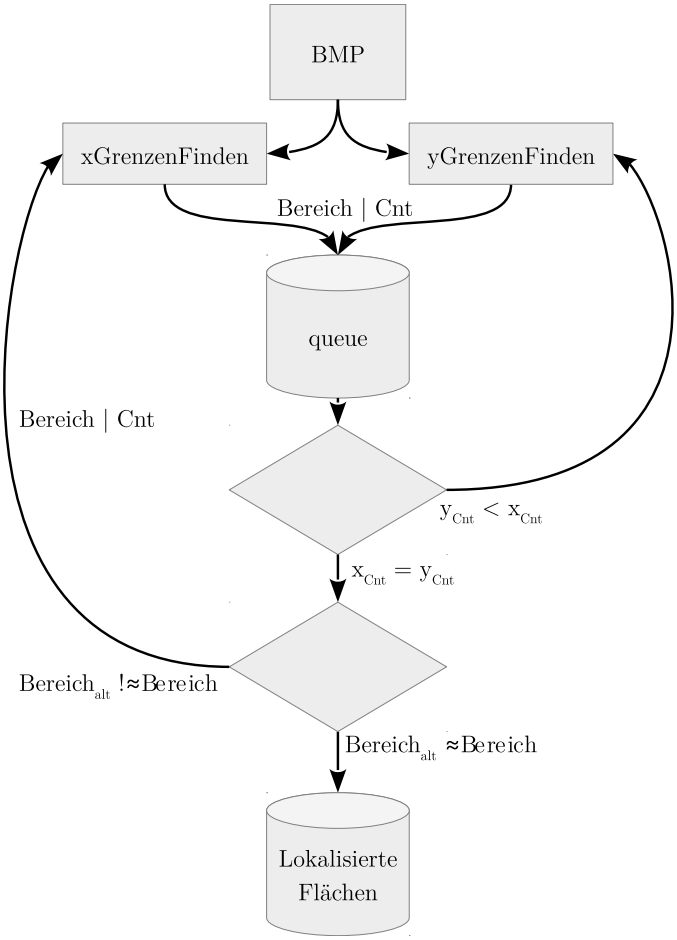


Abbildung 4.2: Die rekursive Struktur ermöglicht das Lokalisieren beliebig vieler Merkmalsflächen in einer Bitmap. Die hier gezeigt Struktur ist eine Weiterentwicklung aus dem in [Wur16] gezeigten Verfahren.

4.2 Realisierung der Flächenerkennung und -Lokalisierung

Das theoretische Verfahren zur Flächenlokalisierung wird im Folgenden auf den humanoiden Roboter Myon angewendet. Im Bild sollen Flächen einer bestimmten Farbe gefunden

werden. Die Farbklassifizierung wird von der Pixel-Pipeline übernommen, die zu diesem Zweck im Modus 5 (+,CCC0,AGU) betrieben wird. Durch die geeignete Wahl der Transformationsmatrix M und den Parametern $\max_l$ $\max_h$ wird auch das Bilden der benötigten Summenvektoren von der Pixel-Pipeline übernommen. Die zu verwendende Farbklasse wird als gegeben vorausgesetzt.
Für die realisierte Variante entfallen die für die Summenbildung notwendigen Zeilen 2-6 des Algorithmus 4.1. Stattdessen werden die Konfigurationsparameter berechnet und der Pixel-Pipeline übergeben. Der resultierende Vektor in der Pixel-RAM wird als *x/ySum(i)* verwendet.

Konfigurationsparameter xSum
Um die Summenbildung für xSum umzusetzen, wird das Eingangsbild auf eine Zeile (in Superpixeln) bei horizontaler Vollauflösung konfiguriert. Ausgehend von den Bereichsgrenzen aus Kapitel 4.1 und der Transformationsmatrix in Formel 3.10 ergeben sich mit

$$X_A = X_C = x_{min} \tag{4.3}$$
$$Y_A = Y_B = y_{min} \tag{4.4}$$
$$X_B = x_{max} \tag{4.5}$$
$$Y_B = y_{max} \tag{4.6}$$
$$max_l = x_{max} - x_{min} \tag{4.7}$$
$$max_h = 1 \tag{4.8}$$

die Konfigurationsparameter zu:

$$\begin{pmatrix} k_{lx} & k_{ly} & b_l \\ k_{hx} & k_{hy} & b_h \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -x_{min} \\ 0 & \frac{1}{y_{max}-y_{min}} & \frac{-y_{min}}{y_{max}-y_{min}} \\ 0 & 0 & 1 \end{pmatrix} \tag{4.9}$$

Da die resultierenden Superpixel bis zu 320 Pixel enthalten können, sollte bei der Definition der Farbklasse darauf geachtet werden, dass die nächst höhere Farbklasse stets "0" ergibt.

Konfigurationsparameter ySum
Analog dazu ergeben sich die Konfigurationsparameter für ySum mit den Bedingungen aus (4.3)-(4.6) und

$$max_l = 1 \tag{4.10}$$
$$max_h = y_{max} - y_{min} \tag{4.11}$$

zu:

$$\begin{pmatrix} k_{lx} & k_{ly} & b_l \\ k_{hx} & k_{hy} & b_h \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{x_{max}-x_{min}} & 0 & \frac{-x_{min}}{x_{max}-x_{min}} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{pmatrix} \tag{4.12}$$

Modifizierung und Ergebnis
Neben den genanten Änderungen im Algorithmus 4.1 müssen weitere Anpassungen getroffen werden. Die veränderten Zeilen sind in Algorithmus 4.2 gezeigt.

Algorithmus 4.2 Ermitteln der Flanken

```
1:  function xGrenzenFinden(xMin, xMax, yMin, yMax, xCnt, yCnt)
2:      kLX ← 16384                                  ▷ *16384 wegen Zahlenformat
3:      kHY ← 16384/(yMax − YMin)
4:      kLY ← 0, kHX ← 0
5:      bL ← −xMin ∗ 64                              ▷ *64 wegen Zahlenformat
6:      bH ← −yMin ∗ 64/(yMax − yMin)
7:      maxL ← xMax − xMin
8:      maxH ← 1
9:      PixelPipelineKonfigurieren(...)
10:     PixelPipelineCapture(5)
11:     ...
12:     for i = 0 → maxL − 1 do                      ▷ Suchen möglicher Flanken
13:         ...
14:         if e = 1 then
15:             xGrenzeMin[n] ← i − 1 + xMin
16:         else if e = 0 and z = 1 then
17:             xGrenzeMax[n] ← i + xMin
18:         else if e = −1 then
19:             ...
20:         end if
21:         ...
22:     end for
23:     ...
24: end function
```

Das Ergebnis des Verfahrens sind die lokalisierten Flächen, die zu Illustrationszwecken weiß umrahmt wurden. Ein so entstandenes Originalbild aus dem Ausgangsvideo des Roboters ist in Abbildung 4.3 zu sehen. Über die Schwellen-Parameter und über zusätzliche Bedingungen können die zu lokalisierenden Flächen bezüglich ihrer Größe und ihres Abstands zueinander eingeschränkt werden. Das Verfahren weist eine hohe Unempfindlichkeit gegenüber Störungen wie Rauschen und Fehl-Detektionen auf. Die Laufzeit des Verfahrens steigt mit der Anzahl der Merkmals-Flächen.
Das implementierte Verfahren eröffnet die Möglichkeit, den Ablauf zu unterbrechen und auf Grundlage der Bereiche eine Interaktion zu starten. Wenn zum Beispiel eine erste Prüfung ergäbe, dass zwei potentielle Bereiche (links und rechts) vorhanden sind, könnte eine Verhaltenssteuerung entscheiden welche der Bereiche zuerst oder überhaupt geprüft werden soll.

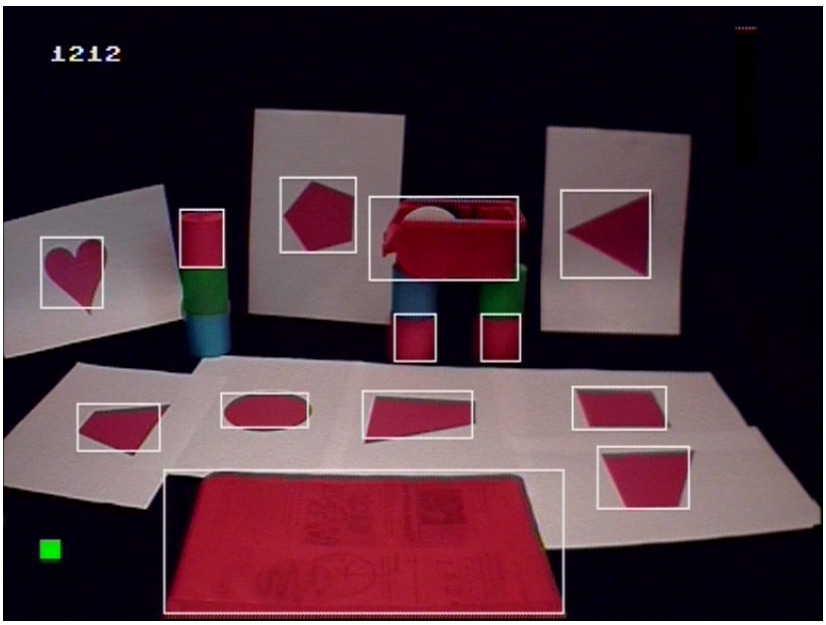


Abbildung 4.3: Das Ergebnis der Flächenlokalisierung für die Farbe Rot mit dem humanoiden Roboter Myon. Jeweils weiß umrahmt sind die gefunden Bereiche.

Kapitel 5

Visuomotorisches Verfahren zur Entfernungsschätzung

Um erkannte Flächen und Objekte in ein Umgebungsmodell einordnen zu können, werden Informationen zum Abstand und der Orientierung, relativ zum Körper des Roboters, benötigt. Mit Hilfe dieser Daten kann die reale Größe der Objekte ermittelt werden. Diese Klassifizierungen können als Indikatoren für eine mögliche Verhaltenssteuerung benutzt werden.
Durch die Drehung des Kopfes um die Gierachse (Abb.2.2 (d)) können Bilder aus verschiedenen Blickwinkeln aufgenommen werden, welche als Grundlage für die Entfernungsschätzung benötigt werden. In den aufgenommenen Bildern ändern potentielle Objekte ihre Position. Aus dieser Positionsänderung und der Änderung des Gierwinkels kann die Entfernung eines Objekts geschätzt werden.

5.1 Geometrische Grundlagen der Entfernungsschätzung

Um eine Ermittlung der Distanz zu ermöglichen werden in diesem Kapitel die geometrischen Zusammenhänge hergeleitet und erklärt. Grundlage des Verfahrens ist das Modellieren des Kamerasystems.

Lochkameramodell
In erster Näherung wird das optische System des Roboters als einfache Lochbildkamera angenommen. Ziel ist das Finden eines geometrischen Zusammenhangs zwischen dem Änderungswinkel des Kopfes und der Positionsänderung zweier korrespondierender Pixel. Der Strahlengang einer Lochkamera ist in Abbildung 5.1 veranschaulicht. Der mathematische Zusammenhang folgt dem Strahlensatz in Formel (5.1).

$$\frac{B}{b} = \frac{G}{g} \tag{5.1}$$

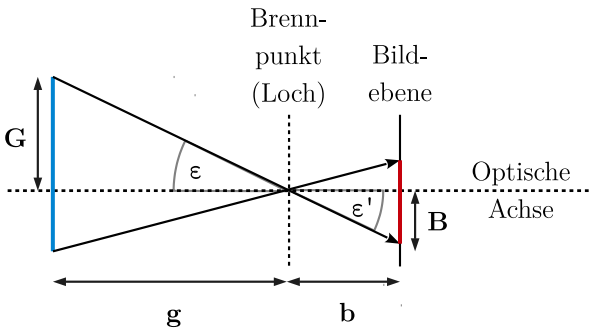


Abbildung 5.1: Strahlengang des Lochkameramodells. Das Objekt G (blau) wird durch den Brennpunkt auf die Bildebene projiziert, wodurch das Bild B (rot) entsteht. Die Objektentfernung g vom Brennpunkt und die Brennweite b definieren den Zusammenhang zwischen G und B.

Wird in die Bildebene ein Bildsensor eingesetzt folgt mit dem Übersetzungsfaktor

$$k = \frac{b_{Sensor}}{Pixel_{Horizontal,Sensor}} \tag{5.2}$$

die Größe der Abbildung B in Abhängigkeit der Größe in Pixel B_p zu:

$$B = B_p \cdot k \tag{5.3}$$

Die Bildwinkel ε und ε' ergeben sich mit (5.1) zu

$$\varepsilon = \varepsilon' = \arctan\left(\frac{B_p \cdot k}{b}\right) \tag{5.4}$$

Rotierendes Lochkameramodell

Das Rotieren des Roboterkopfes wird auf das Lochkameramodell angewendet. Abbildung 5.2 zeigt die geometrischen Zusammenhänge in der Draufsicht. Um die Lage des Punktes P zu beschreiben wird ein zwei-dimensionales Koordinatensystem mit Ursprung im Rotationspunkt M angesetzt. Die beiden Bildwinkel ε_1 und ε_2 berechnen sich aus den Bildpunkten B_1 und B_2 (hier in Pixel) wie in Formel 5.4. Um die Distanz eines Objektes vom Drehpunkt M zu ermitteln, wird hier vereinfacht ein bestimmter Punkt des Objekts verwendet. Eine mögliche gedrehte Lage des Objekts wird so vernachlässigt, was für eine Schätzung der Entfernung ausreichend ist. Der Punkt P, die linke Ecke des Objekts, ergibt sich somit nach Abbildung 5.2 zu:

$$P_1 = \begin{pmatrix} r \cdot \cos(\varphi) \\ r \cdot \sin(\varphi) \end{pmatrix} + \begin{pmatrix} g_1 \cdot \cos(\varphi + \varepsilon_1) \\ g_1 \cdot \sin(\varphi + \varepsilon_1) \end{pmatrix}, \tag{5.5}$$

$$P_2 = \begin{pmatrix} r \cdot \cos(\varphi + \beta) \\ r \cdot \sin(\varphi + \beta) \end{pmatrix} + \begin{pmatrix} g_1 \cdot \cos(\varphi + \beta + \varepsilon_2) \\ g_1 \cdot \sin(\varphi + \beta + \varepsilon_2) \end{pmatrix}. \tag{5.6}$$

Für ein ruhendes Objekt gilt $P_1 = P_2$. Aus dieser Bedingung folgt für g_1:

$$g_1 = \frac{r \cdot [\sin(\varphi + \beta) + \tan(\varphi + \beta + \varepsilon_2) \cdot [\cos(\varphi) - \cos(\varphi + \beta)] - \sin(\varphi)]}{\sin(\varphi + \varepsilon_1) - \cos(\varphi + \varepsilon_1) \cdot \tan(\varphi + \beta + \varepsilon_2)} \tag{5.7}$$

Mit $g_1 = r \cdot g$ ergibt sich für g unter Anwendung des Additionstheorems:

$$g = \frac{-\sin(\varepsilon_2) + \sin(\beta + \varepsilon_2)}{\sin(\varepsilon_1 - \varepsilon_2 - \beta)} \tag{5.8}$$

Der Punkt P lässt sich demnach wie folgt beschreiben.

$$\begin{pmatrix} x_P \\ y_P \end{pmatrix} = \begin{pmatrix} r \cdot \cos(\varphi) \\ r \cdot \sin(\varphi) \end{pmatrix} + \begin{pmatrix} r \cdot g \cdot \cos(\varphi + \varepsilon_1) \\ r \cdot g \cdot \sin(\varphi + \varepsilon_1) \end{pmatrix} \tag{5.9}$$

Der Abstand des Punktes P zum Rotationspunkt M und der Winkel α zur Bezugsachse x ergeben sich nach:

$$D = \sqrt{x_P^2 + y_P^2} \tag{5.10}$$

$$= r \cdot (1 + g)\,, \tag{5.11}$$

$$\alpha = \begin{cases} \arccos\left(\dfrac{x_P}{D}\right) & \text{für: } y \geq 0 \\ -\arccos\left(\dfrac{x_P}{D}\right) & \text{für: } y < 0 \end{cases} \tag{5.12}$$

$$= \begin{cases} \arccos\left(\dfrac{\cos(\varphi) + \cos(\varphi + \varepsilon_1) \cdot g}{1 + g}\right) & \text{für: } y \geq 0 \\ -\arccos\left(\dfrac{\cos(\varphi) + \cos(\varphi + \varepsilon_1) \cdot g}{1 + g}\right) & \text{für: } y < 0 \end{cases}\,. \tag{5.13}$$

Das vorgestellte Verfahren unterliegt, für eine Drehung des Kopfes gegen den Uhrzeigersinn, der Bedingung:

$$\varphi + \beta\varepsilon_2 < \varphi + \varepsilon_1 \tag{5.14}$$

Bei einer Drehung mit dem Uhrzeigersinn:

$$\varphi + \beta\varepsilon_2 > \varphi + \varepsilon_1 \tag{5.15}$$

Diese Bedingungen stellen die Grenzen dar, bei denen sich die Geraden g_1 und g_2 gerade noch schneiden, beziehungsweise wie weit sich der Kopf von einem zum anderen Bild drehen darf.

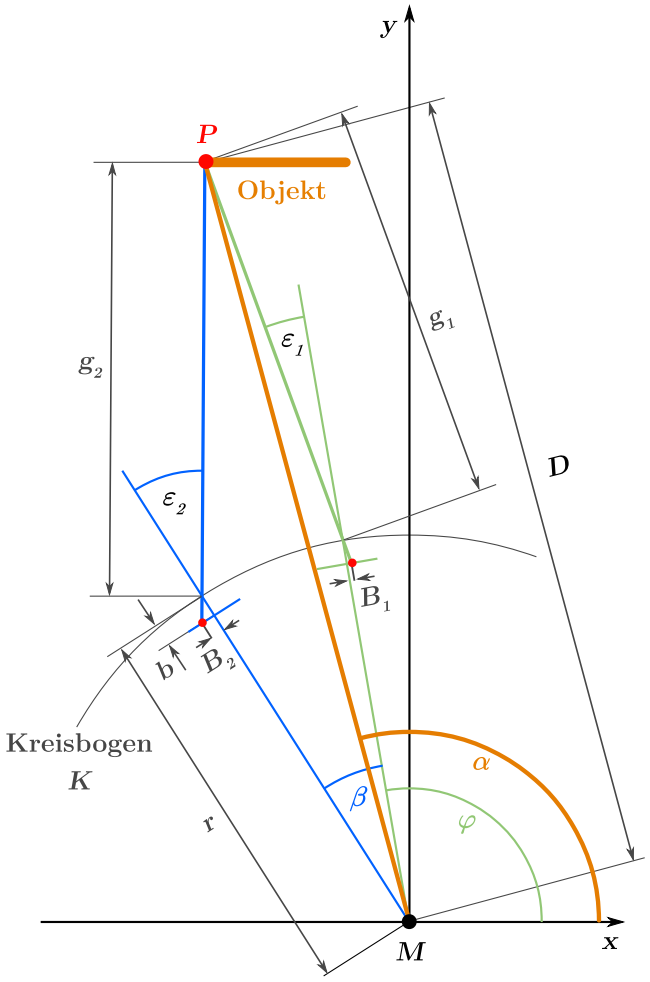


Abbildung 5.2: Das Kamerasystem wird um den Punkt M auf dem Kreisbogen K rotiert. Das Objekt (orange) mit dem Punkt P wird zweimal von der Kamera aufgenommen. Die grüne Konstellation steht für das erste Bild, wobei der Winkel φ die Auslenkung der optischen Achse zur x- Achse beschreibt. Die blaue Anordnung steht analog dazu für das zweite Bild. Die beiden Bildwinkel ε_1 und ε_2 beziehen sich auf die jeweilige optische Achse. Aus den beiden Bildpunkten (B_1, B_1), der Brennweite b, dem Radius r und den beiden Winkeln (φ, β) kann die Position des Punktes P ermittelt werden.

5.2 Systemparameter und Messung der Entfernungsschätzung

Die hergeleiteten Zusammenhänge werden im Folgenden hinsichtlich der verwendeten Plattform konkretisiert. Die darauf aufbauende Messreihe als auch die daraus resultierenden Systemgrenzen sind Teil dieses Unterkapitels.

Systemparameter
Das im Roboter verwendete optische System verfügt im Gegensatz zu der getroffenen Annahme über ein Objektiv, welches sich nicht über den Zusammenhang $\varepsilon = \varepsilon'$ modellieren lässt. Es gilt $\varepsilon < \varepsilon'$, wobei der Zusammenhang zwischen Objekt- und Bildwinkel hier vereinfacht als proportional angenommen wird:

$$\varepsilon = \arctan\left(\frac{B_P \cdot k}{b \cdot f}\right) \tag{5.16}$$

Der Faktor f ist als Korrekturfaktor für das Objektiv anzusehen, welcher rechnerisch die Brennweite und somit den Bildwinkel beeinflusst. Für diese Anwendung wurde der Korrekturfaktor experimentell bestimmt.

$$f = 0,646 \tag{5.17}$$

Der verwendete 1/4" Sensor arbeitet auf einer Fläche von 3,2 x 2,4 mm [Vid11]. Der Übersetzungsfaktor für diesen Sensor ergibt sich zu:

$$k = \frac{3,2 \text{ mm}}{736 \text{ Pixel}} = 4,35\frac{\mu\text{m}}{\text{Pixel}} \tag{5.18}$$

Da die Bildverarbeitung mit einer horizontalen Auflösung von 360 Pixeln arbeitet, ergibt sich der Übersetzungsfaktor zu:

$$k = \frac{3,2 \text{ mm}}{360 \text{ Pixel}} = 8,89\frac{\mu\text{m}}{\text{Pixel}} \tag{5.19}$$

Da das Bild nach dem optischen System noch einmal gedreht wird (natürliche Wiedergabe), müssen alle Pixel rechts der vertikalen Bildmitte als negativ betrachtet werden.
Für die Berechnung der Distanz wird zudem der Radius r benötigt. Der Abstand zwischen dem Rotationsmittelpunkt M und dem Bildsensor b beträgt:

$$r = 11,5 \text{ cm} \tag{5.20}$$

Das in Abbildung 5.2 vorgestellte Koordinatensystem wird wie in Abbildung 5.3 für den Roboter angewendet, wobei es den Anforderungen entsprechend Torso starr ausgelegt ist.

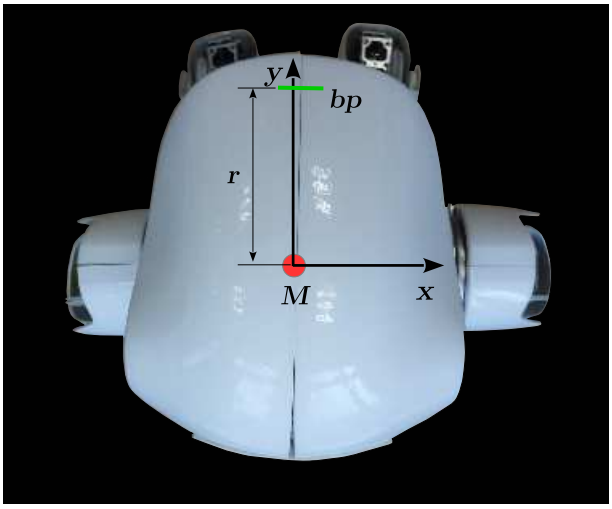


Abbildung 5.3: Das Koordinatensystem wird auf Myon angewendet. Die x-Achse verläuft über der Schulterachse, y zeigt somit in Blickrichtung geradeaus. Rot markiert der Drehpunkt M des Kopfes, grün der Brennpunkt bp des optischen Systems. Der Abstand zwischen M und bp ist der Radius r. In diesem Fall ist der Kopf leicht nach rechts ausgerichtet.

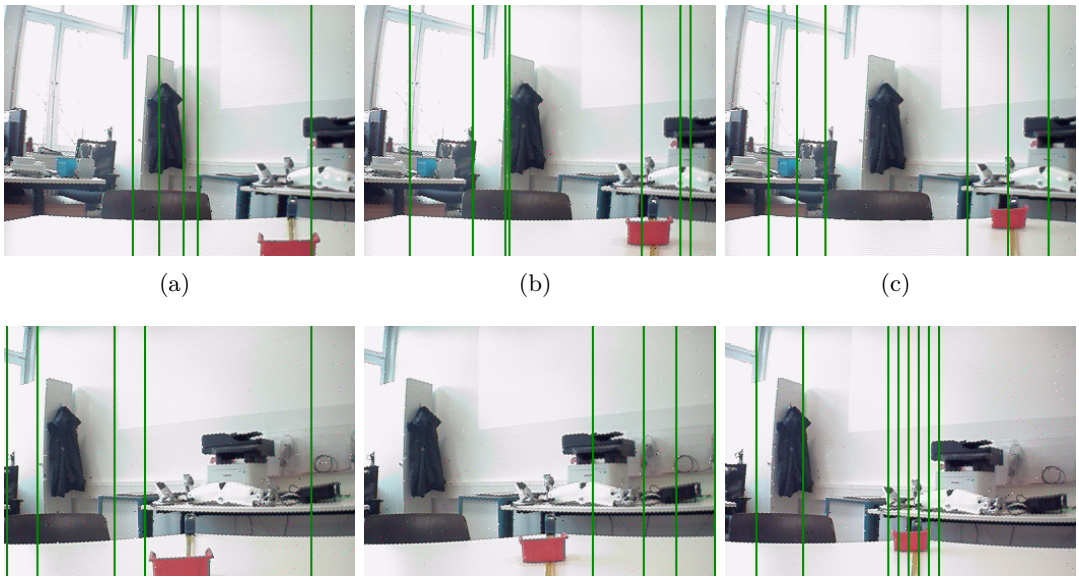
Messreihe

Um das vorgestellt Verfahren zu überprüfen wurde eine Testreihe mit dem Roboter Kopf durchgeführt. Der Roboterkopf wurde mit Hilfe eines Stativs auf einem Tisch platziert und für drei Entfernungen eines Objektes jeweils zwei Bilder aufgenommen. Der Roboter kann über den sogenannten Pieper[1] ferngesteuert werden, was unter anderem ein manuelles Aufnehmen von Bildern ermöglicht. Die Nick- und die Rollachse sind für diesen Aufbau mit einem P-Regler waagerecht gehalten worden. Die Winkel (φ,β) wurden manuell abgelesen. Aus den Abbildungen 6.2.2 (a)-(f) folgen die jeweiligen Positionen für die linke untere Ecke des roten Objekts in Pixeln. Tabelle 5.1 zeigt die Pixelwerte der Bildpaare und die daraus resultierende Distanz.

[1] Pieper: erzeugt akustische Signale die vom Roboter erfasst werden. Die Signale werden zum Fernsteuern des Roboters verwendet.

Bild	P[Pixel]	φ [°]	β [°]	D_b[cm]	D[cm]
(a)	-82	-	20	60,3	60
(d)	25	90	-		
(b)	-87	-	20	79,3	80
(e)	15	90	-		
(c)	-92	-	19,4	96,5	100
(f)	5	90,6	-		

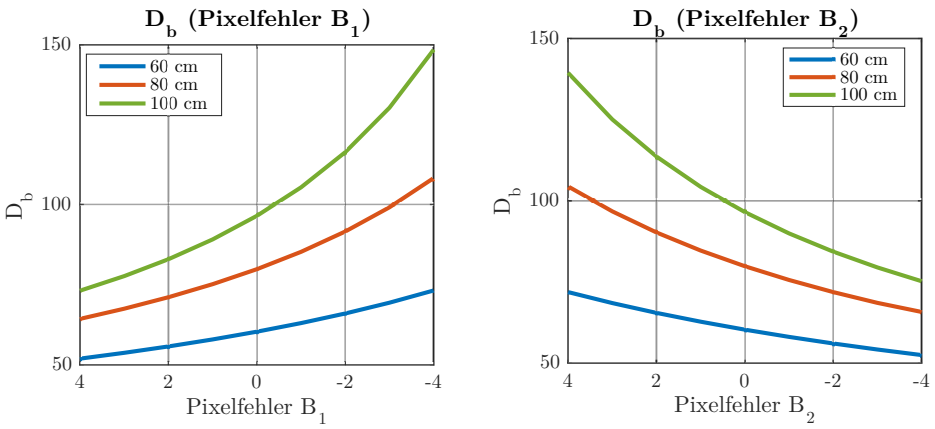
Tabelle 5.1: Messergebnisse für die Abb. 5.4 (a)-(f), wobei D die tatsächliche, D_b die aus den Messungen berechnete Distanz ist.



(a) (b) (c)

(d) (e) (f)

Abbildung 5.4: Für das rote Objekt wurde jeweils eine Entfernungsschätzung durchgeführt. Die Abstände betragen 60 ((a),(d)), 80 ((b),(e)) und 100 cm ((c),(f)). Der Winkel für die oberen Bilder ist jeweils 110° und für die unteren jeweils 90° ($\varphi = 90°$ und $\beta = 20°$). Bild (f) weicht jedoch leicht ab, hier gilt $\varphi \approx 90,6°$ und somit $\beta = 19,4°$. Die grünen Linien stellen den optischen Fluss dar und haben keine auswertungstechnische Relevanz.



(a) (b)

Abbildung 5.5: Die berechnete Distanz als Funktion von Pixelfehlern von B_1 und B_2. Der Richtungssinn der x-Achsen ist bewusst umgekehrt, um die Fehlerrichtung zu veranschaulichen. Negative Werte für x bedeuten eine rechtere Position als der Sollwert (0).

Durch Rauschen, ungenaue Objekterkennung oder optische Verzerrung am Sensor können in der Angabe der Pixelposition Fehler entstehen. Die Abhängigkeit der Distanz von einem möglichen Pixelfehler ist in den Abbildungen 5.5 gezeigt. Negative Pixelfehler bedeuten, dass der Punkt um x- Werte zu weit rechts vom Sollwert liegt. Wie zu erwarten wächst der Fehler mit der Distanz und nur für Rechts- (B2) bzw. Linksverschiebungen (B1). Die Kurven stellen jeweils einen Ausschnitt aus dem Systemgraphen dar, welcher in Abbildung 5.6 zu sehen ist.

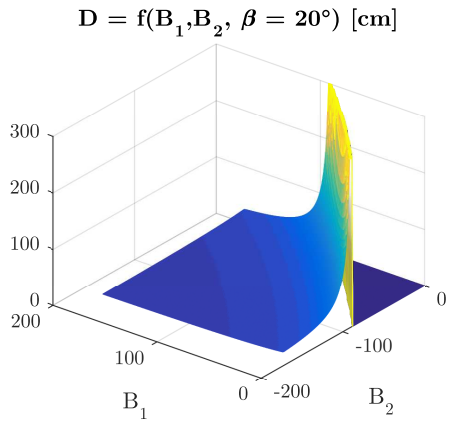


Abbildung 5.6: Berechnete Distanz als Funktion der Pixelwerte B_1 und B_2: Die linke Fläche mit D = 0 liegt nach Bedingung (5.14,5.15) außerhalb des Definitionsbereiches. Die Funktion wurde bei 300 cm gesättigt um eine bessere Darstellung zu ermöglichen. Für die restlichen Pixelwerte von B_1 (negativ) und B_2 (positiv) ist der Graph symmetrisch. Für steigende Werte von β verschiebt sich der „Wellenberg“ entlang der Diagonalen von (0,0) nach (200,-200). Somit wird auch der undefinierte Bereich immer größer.

Systemgrenzen und weiterführende Betrachtungen

Das System wird durch die Bedingungen in den Formeln 5.14 und 5.15 begrenzt. Aus diesen Grenzen kann ein Rückschluss auf den Mindestabstand in Abhängigkeit des Änderungswinkels β getroffen werden. Hierzu werden die Pixelpositionen, die ein Punkt P verursacht, als maximal positiv und negativ angenommen. Die sich ergebende minimale Distanz eines Objekts $D_{min}(\beta)$ ist in Abbildung 5.7 gezeigt. Objekte, die sich näher am Roboter befinden, sind nach der Drehung um β außerhalb des Bildes.
Die maximal schätzbare Entfernung wird durch die gewünschte Genauigkeit und den Änderungswinkel β eingeschränkt. Die sich ergebende Kurvenschar für $D_{max}(\beta)$ ist ebenso in Abbildung 5.7 veranschaulicht. Die Genauigkeit in cm gibt die zulässige Toleranz beim Berechnen der Entfernung an. Die Einhaltung einer geringen Toleranz ist nur für nahe Objekte und eine relativ große Änderung des Winkels möglich. Eine Verbesserung würde durch eine höhere Auflösung erreicht werden. Eine Auflösung von 1920 horizontalen Pixeln (1080i) würde die Kurvenschar um ca.100 cm anheben.

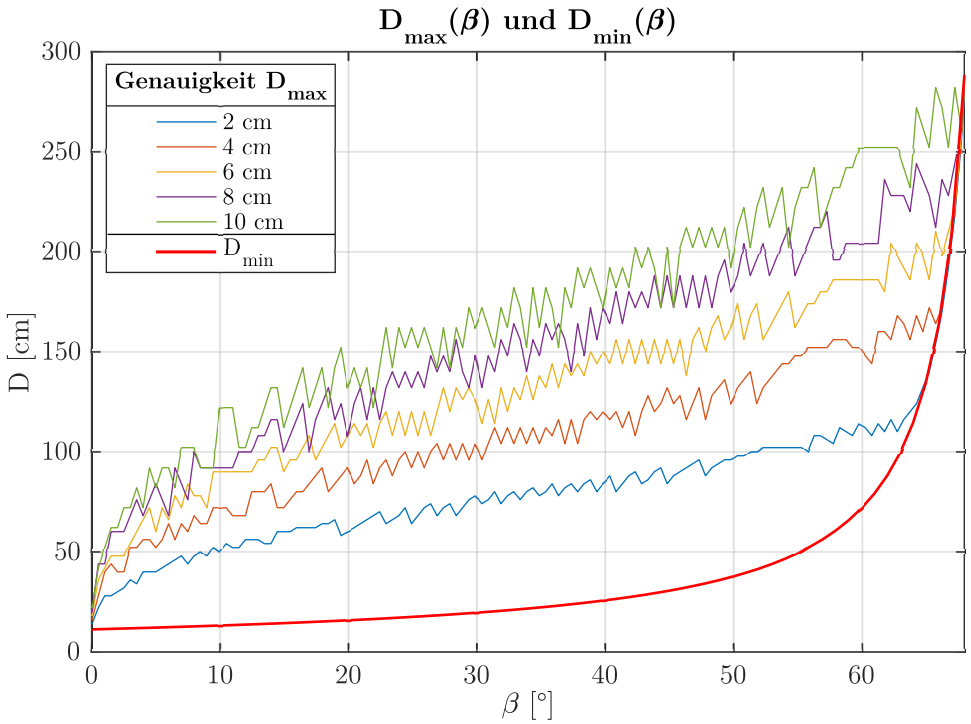


Abbildung 5.7: Die minimale und die maximale Distanz eines Objektes zum Roboter in Abhängigkeit vom Änderungswinkel β. Die Distanz des Objekts muss größer sein als $D_{min}(\beta)$ und kleiner als $D_{max}(\beta)$, in Abhängigkeit der gewünschten Genauigkeit.

Ist die Distanz eines Objekts bekannt, kann die Position im Bild berechnet werden nachdem der Kopf gedreht wurde. Bei gegebenem $\varepsilon_1(B_{1,Pixel})$ und β verschiebt sich der Punkt zu $B_{2,Pixel}$ in Abhängigkeit der Distanz. Der zugehörige Winkel $\varepsilon_2(D)$ berechnet sich zu:

$$\varepsilon_2(D) = \arctan\left(\frac{\sin(\beta) - \sin(\varepsilon_1 - \beta)\cdot\left(\frac{D}{r} - 1\right)}{-\cos(\beta) - \cos(\varepsilon_1 - \beta)\cdot\left(\frac{D}{r} - 1\right) + 1}\right) \tag{5.21}$$

Die Verschiebung in Pixeln ergibt sich zu:

$$\Delta B_P(D) = \frac{b\cdot f}{k}\cdot[\tan(\varepsilon_1) - \tan(\varepsilon_2(D))] \tag{5.22}$$

Für weit entfernte Objekte gilt $D \to \infty$; für die Änderung des Bildwinkels folgt somit:

$$\lim_{D\to\infty} \Delta\varepsilon\,(D) = -\varepsilon_1 + \arctan\left(\frac{\cancel{\sin(\beta)} - \sin(\varepsilon_1 - \beta)\cdot\infty}{\cancel{-\cos(\beta)} - \cos(\varepsilon_1 - \beta)\cdot\infty + \cancel{1}}\right) \tag{5.23}$$

$$= -\varepsilon_1 + \arctan\left(\frac{-\sin(\varepsilon_1 - \beta)\cdot\cancel{\infty}}{-\cos(\varepsilon_1 - \beta)\cdot\cancel{\infty}}\right) \tag{5.24}$$

$$= -\varepsilon_1 + \varepsilon_1 - \beta \tag{5.25}$$

$$= -\beta \tag{5.26}$$

Was einer Verschiebung in Pixeln von

$$\lim_{D\to\infty} \Delta B_P\,(D) = \frac{b\cdot f}{k}\cdot[\tan(\varepsilon_1) - \tan(\varepsilon_1 - \beta)] \tag{5.27}$$

entspricht. Kleine Werte für ε_1 und β führen zu der Vereinfachung in Formel 5.29, da der Tangens für kleine Werte näherungsweise der Identität entspricht.

$$\lim_{D\to\infty} \Delta B_P\,(D) = \frac{b\cdot f}{k}\cdot[\varepsilon_1 - \varepsilon_1 - \beta] \tag{5.28}$$

$$= -\frac{b\cdot f}{k}\cdot\beta \tag{5.29}$$

Auf diesen Vereinfachungen basiert die in Kapitel 6.2 vorgestellte Nah- und Fernfeldseparation.

Kapitel 6

Visuomotorische Verfahren zur Bewegungsverfolgung und Objekterkennung

Die Koordination zwischen visuellen Informationen und der durch die Motoren hervorgerufenen Eigenbewegung –die Visuomotorik– ist der Hauptbestandteil dieses Kapitels. Neben dem Bewegen des Kopfes aufgrund von visuellen Informationen nehmen auch umgekehrt die Kopfbewegungen Einfluss auf die Bildverarbeitung. Als visuelle Informationen dienen Bewegungen im Bild und im Besonderen das Zeigen auf Objekte, wobei der Kopf so gedreht werden soll, dass das Zentrum des Zeigens beziehungsweise der Bewegung in die Bildmitte wandert. Beim Drehen entstehen selbst verursacht Bewegungen im Bild, welche für eine davon unabhängige Bildverarbeitung gegen gerechnet werden müssen. Das Erkennen und Verfolgen von Bewegungen (Kap. 6.2) sowie das Erkennen von zeigenden Bewegungen (Kap. 6.3) wird in dieser Arbeit auf der Grundlage von Differenzbildern (Kap. 6.1) realisiert.

6.1 Grundlagen der Differenzbildanalyse

Grundlagen zur Bewegungserkennung, wie die Analyse von Differenzbildern, basieren auf der Annahme, dass sich Bewegungen als Veränderungen in einer Bildfolge darstellen. In dieser Arbeit wird die erste Auswertung von möglichen Veränderungen auf das Klassifizieren in Bewegung und Nicht-Bewegung eingeschränkt. In Anlehnung an die in Kapitel 3.2.2 beschriebene Colormap wird die aus der Bewegungsklassifizierung erzeugte Bitmap als Movingmap bezeichnet.
Unter der Annahme, dass sich Veränderungen im Bild im Wesentlichen in der Helligkeit auswirken, werden zum Erstellen eines Differenzbildes, aus zwei zeitlich aufeinander folgenden Bildern, nur die Luminanzwerte (Y) verwendet. Die beiden einzelnen Bilder können daher näherungsweise als Graustufenbilder interpretiert werden. Die Helligkeitsänderung

der Grauwertbildfolge an der Stelle (*x,y*) ist formal als

$$\frac{\partial Y\left(x,y,t\right)}{\partial t} \tag{6.1}$$

definiert. Mit der Zeit t diskret ergibt sich die Helligkeitsänderung zu

$$\Delta Y\left(x,y,t\right) = Y\left(x,y,t+1\right) - Y\left(x,y,t\right). \tag{6.2}$$

Das alte Graustufenbild (t) wird von dem neuen Graustufenbild (t+1) abgezogen, wodurch das Differenzbild $\Delta Y\left(x,y,t\right)$ erzeugt wird. Für Pixel deren Helligkeitswert in beiden Graustufenbildern nicht gleich ist, ist das Differenzbild ungleich Null. Dementsprechend gilt, dass für alle diejenigen Pixel, die sich über die Zeit in ihrem Helligkeitswert verändert haben, das Differenzbild ungleich 0 ist. Für das Klassifizieren des Differenzbildes in Bewegung und Nicht-Bewegung und somit das Erzeugen der Movingmap, wird eine zusätzliche Schwelle eingeführt. Die Pixel deren Wert im Differenzbild betragsmäßig größer als diese Schwelle sind, werden als Bewegung klassifiziert und somit in der Movingmap auf Eins gesetzt. Durch die Schwelle können fehlerhafte, durch Rauschen und globale Helligkeitsänderung verursachte, Änderungen im Differenzwert ignoriert werden. Die Schwelle stellt die minimale Helligkeitsänderung dar, bei der ein Pixel noch als Bewegt klassifiziert wird. Um die Bilder voneinander abzuziehen wird die Pixel-Pipeline verwendet, die durch das erweiterte Rechenwerk Differenzbildungen zu lässt. Zu diesem Zweck wird zuerst ein Bild im Modus 1 (+,YUV,AGU) aufgenommen und anschließend ein Bild im Modus 3 (-,YUV,AGU). Zwischen diesen Aufnahmen wird der Inhalt der Pixel-RAM nicht verändert. Durch die in Formel 3.17 beschriebene maximale Pixelanzahl pro Superpixel von 128, kann pro Aufnahme jedoch nur ein Halbbild verarbeitet werden, was eine entsprechende Konfiguration der AGU voraussetzt, daher wird die Differenzbildung erst für die linke Hälfte und dann für die rechte Hälfte des Bildes durch geführt. Die zugehörigen Konfigurationsparameter für die linke Seite ergeben sich zu

$$\begin{pmatrix} k_{lx} & k_{ly} & b_l \\ k_{hx} & k_{hy} & b_h \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1640 & 0 & 0 \\ 0 & 1640 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.3}$$

$$max_l = 17 \tag{6.4}$$

$$max_h = 27. \tag{6.5}$$

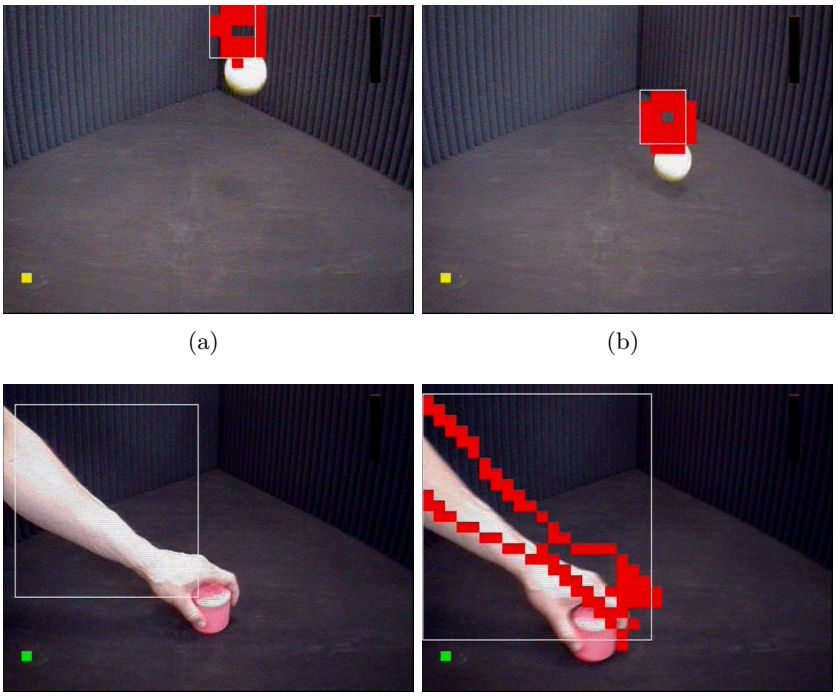
Die rechte Seite wird über

$$\begin{pmatrix} k_{lx} & k_{ly} & b_l \\ k_{hx} & k_{hy} & b_h \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1640 & 0 & -1087 \\ 0 & 1640 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.6}$$

$$max_l = 17 \tag{6.7}$$

$$max_h = 27 \tag{6.8}$$

definiert. Der aus dem nacheinander Aufnehmen resultierende zeitliche Versatz zwischen der linken und der rechten Hälfte wird als vernachlässigbar angenommen.
Aus den beiden Differenzbildern wird mittels der genannten Schwelle eine Movingmap für das Gesamtbild erzeugt. Die Kanten von bewegten Objekten werden in der Movingmap



(a) (b)

(c) (d)

Abbildung 6.1: Die Abbildungen (a) und (b) zeigen einen sich bewegenden Ball, (c) und (d) einen bewegenden Arm. Die Superpixel der Movingmap die als bewegt klassifiziert wurden sind rot markiert. Die weiße Umrahmung stellt die Grenzen einer Flächenlokalisierung für die jeweilige Movingmap dar.

als Flächen sichtbar, durch schnelle Bewegungen werden diese Flächen größer, was zu einem möglichen Verbinden der Flächen führen kann. Auch Kanten kleiner Objekte sind aufgrund der Größe als eine Gesamtfläche zu sehen. Ausgehend von der Movingmap können potentielle Bewegungen lokalisiert werden.

6.2 Verfolgen von sichtbaren Bewegungen

Die erste Instanz einer visuomotorischen Aktion in dieser Arbeit ist das Drehen des Kopfes in Richtung von Bewegungen. Durch diese Funktion kann die Blickrichtung des Roboters interaktiv beeinflusst werden, um zum Beispiel ein Objekt, das sich außerhalb des aktuellen Blickfeld befindet, für den Roboter sichtbar werden zu lassen.

Durch das Drehen werden Bewegungen im Bild verursacht, welche zum Verfolgen von externen, nicht selbstverursachten Bewegungen gegengerechnet werden müssen. Aus Basis der in Kapitel 6.1 vorgestellten Movingmap werden externe Bewegungen erkannt und lokalisiert.

Lokalisieren externer Bewegungen
Um die Bewegung zu lokalisieren wird die Movingmap ausgewertet. Hierzu wird der geometrische Schwerpunkt aller Bewegungen im Bild ermittelt, indem im ersten Schritt, wie in Kapitel 4.1, die Summenvektoren

$$x_{sum}(i) = \sum_{l=0}^{26} MMP(i,l) \text{ und} \tag{6.9}$$

$$y_{sum}(j) = \sum_{l=0}^{33} MMP(l,j) \tag{6.10}$$

für die Movingmap (*MMP*) gebildet werden. Die Grenzen der Summe für x_{sum} ergibt sich aus der Formel 6.5, der Höhe der Movingmap und für y_{sum} aus den Formeln 6.4 und 6.7, die Gesamtbreite der Movingmap. Anhand dieser Summenvektoren wird der geometrische Schwerpunkt in horizontaler Richtung

$$x_{SP} = \sum_{i=0}^{33} \frac{x_{sum}(i) \cdot i}{34} \tag{6.11}$$

und in vertikaler Richtung

$$y_{SP} = \sum_{j=0}^{26} \frac{y_{sum}(j) \cdot j}{27} \tag{6.12}$$

ermittelt. Die Schwerpunkte müssen noch auf die tatsächliche Pixelanzahl korrigiert und eine Verschiebung vorgenommen werden, so dass die Mitte des Bildschirms dem Nullpunkt entspricht.

$$x_{SPA} = x_{SP} \cdot \frac{360}{34} - \frac{360}{2} \tag{6.13}$$

$$y_{SPA} = y_{SP} \cdot \frac{288}{27} - \frac{288}{2} \tag{6.14}$$

Die daraus resultierenden Koordinaten kennzeichnen den geometrischen Schwerpunkt aller als bewegt klassifizierter Superpixel.

Drehen des Kopfes in Richtung des Bewegungsschwerpunktes
Ausgehend von den Koordinaten wird der Kopf in Richtung des Schwerpunktes der Bewegung gedreht. Die Ermittlung der neuen Sollwerte basiert hierbei auf der Annahme, dass sich die Bewegung in einem immer gleichen Abstand vom Kopf abspielt. Der Grund für diese Annahme ist, dass sich die Verschiebung eines Objektes im Bild in Abhängigkeit vom Kopfwinkel über die Entfernung definiert. Umgekehrt ergibt sich für das Drehen

des Kopfes, dass je nach Abstand der Bewegung eine unterschiedliche Verschiebung im Bild resultiert, was eine exakte Berechnung der Sollwerte verhindert. Daher wird der angenommene Abstand vom Kopf so gewählt, dass der Drehfehler maximal ein drittel des überstrichenen Winkels beträgt.
Für Objekte im Abstand von $2r$ ergibt sich die Änderung des Bildwinkels näherungsweise zu

$$\Delta\varepsilon = -2\beta. \tag{6.15}$$

Nach der Bedingung in Formel 5.26 ergibt sich die Änderung des Bildwinkels zu $\Delta\varepsilon = -\beta$. Für die Berechnung wird von einer mittleren Winkeländerung mit $\Delta\varepsilon = -1,5\beta$ ausgegangen. Mit den in den Abbildungen 2.6 (a) und (b) gezeigten Zusammenhängen zwischen den Winkelwerten in Grad und den Winkelwerten in Winkel-Einheiten ergibt sich für den Gierwinkel

$$\beta = \frac{\vartheta_{Gier} \cdot \pi}{42000}. \tag{6.16}$$

Mit der Vereinfachung aus Formel 5.29 und der Annahme $\Delta\varepsilon = -1,5\beta$ ergibt sich der Gierwinkel in Winkel-Einheiten wie in Abbildung 2.6 (b) zu

$$\Delta\vartheta_{Gier}\left(x_{SPA}\right) = x_{SPA} \cdot -\frac{k \cdot 42000}{b \cdot f \cdot \pi \cdot 1,5} \tag{6.17}$$

$$= x_{SPA} \cdot -39. \tag{6.18}$$

Gleichermaßen folgt für den Nickwinkel

$$\Delta\vartheta_{Nick}\left(y_{SPA}\right) = y_{SPA} \cdot -\frac{k \cdot 15700}{b \cdot f \cdot \frac{\pi}{2} \cdot 1,5} \tag{6.19}$$

$$= y_{SPA} \cdot -29. \tag{6.20}$$

Die so ermittelten Winkel stellen die Änderung der Sollwerte in Abhängigkeit des Schwerpunkts der Bewegung dar und müssen daher noch mit der aktuellen Position verrechnet werden:

$$\vartheta_{Gier}(t+1) = \vartheta_{Gier}(t) + \Delta\vartheta_{Gier} \tag{6.21}$$

$$\vartheta_{Nick}(t+1) = \vartheta_{Nick}(t) + \Delta\vartheta_{Nick}. \tag{6.22}$$

Diese Sollwerte werden wie in Kapitel 2.2 erläutert über den Spinalcord an das relevante Accelboard3D übermittelt, wobei die vorgestellte Regelung für das Einnehmen der Position sorgt.

Korrektur der selbst verursachten Bewegung im Bild: Nah- und Fernfeldseparation

Auch während dem Drehen sollen Bewegungen erfasst und ausgewertet werden, was jedoch aufgrund der selbst verursachten Bewegung im Bild einer zusätzlichen Anpassung bedarf. Wie in Kapitel 5 erläutert verschiebt sich das Bild in Folge des Drehens des Kopfes mindestens um $-\beta$ (Formel 5.26). Auch Objekte die mehr als 2 Meter vom Roboter entfernt

sind folgen näherungsweise dieser Verschiebung. Mit der zusätzlichen Annahme, dass der arctan für die auftretenden Winkel der Identität entspricht, kann die von Objekten, welche weiter als 2 Meter entfernt sind, verursachte Bewegung in der Movingmap verhindert werden. Dazu müssen die Grauwertbilder vor der Differenzbildung so gegeneinander verschoben werden, dass die Verschiebung der selbst verursachten Bewegung kompensiert wird. Die Pixel-Pipeline ermöglicht dieses Verschieben mit Hilfe der AGU in Echtzeit.
Das jeweils erste Halbbild (links,rechts) wird ohne zusätzliche Verschiebungen aufgenommen, die jeweils Zweiten werden in Abhängigkeit vom überstrichenen Winkel verschoben. Daraus ergibt sich der schematische Verlauf wie folgt:

1. Winkelwerte erfassen
2. erstes Grauwertbild aufnehmen (Modus 1)
3. Winkelwerte erfassen
4. – erforderliche Verschiebung berechnen
   – Pixel-Pipeline konfigurieren
5. zweites Grauwertbild aufnehmen (Modus 3)

Die Berechnung der erforderlichen Verschiebungen ergibt sich, mit der durch die Bewegung verursachten Verschiebung in Pixeln in vertikaler Richtung

$$\Delta B_{PY}\left(\Delta\vartheta_{Nick}\right) = -\frac{b \cdot f \cdot \frac{\pi}{2}}{k \cdot 15700} \cdot \Delta\vartheta_{Nick} \tag{6.23}$$

$$= -0,0228 \cdot \Delta\vartheta_{Nick} \tag{6.24}$$

und in horizontaler Richtung

$$\Delta B_{PX}\left(\Delta\vartheta_{Gier}\right) = -\frac{b \cdot f \cdot \pi}{k \cdot 42000} \cdot \Delta\vartheta_{Gier} \tag{6.25}$$

$$= -0,017 \cdot \Delta\vartheta_{Gier}, \tag{6.26}$$

zu

$$b_l\left(\vartheta_{Gier}, \vartheta_{Nick}\right) = \frac{64 \cdot \left(h_{Pixel} - 1\right) \cdot max_l \cdot \Delta B_{PX}\left(\Delta\vartheta_{Gier}\right)}{\left(l_{Pixel} - 1\right) \cdot \left(\left(h_{Pixel} - 1\right) + 2 \cdot \Delta B_{PY}\left(\Delta\vartheta_{Nick}\right)\right)} \tag{6.27}$$

$$= -29,5 \cdot \frac{\Delta\vartheta_{Gier}}{287 - 0,0456 \cdot \Delta\vartheta_{Nick}} \tag{6.28}$$

und zu

$$b_h\left(\vartheta_{Nick}\right) = \frac{64 \cdot \left(l_{Pixel} - 1\right) \cdot max_h \cdot \Delta B_{PY}\left(\Delta\vartheta_{Nick}\right)}{\left(l_{Pixel} - 1\right) \cdot \left(\left(h_{Pixel} - 1\right) + 2 \cdot \Delta B_{PY}\left(\Delta\vartheta_{Nick}\right)\right)} \tag{6.29}$$

$$= -39,4 \cdot \frac{\Delta\vartheta_{Nick}}{287 - 0,0456 \cdot \Delta\vartheta_{Nick}}. \tag{6.30}$$

Für das rechte Halbbild muss das Ergebnis aus Formel 6.28 zu dem Wert b_l aus Formel 6.6 addiert werden. Bei der Auswertung der Gesamt-Movingmap ist zudem darauf zu Achten, dass sich durch das Verschieben Definitionslücken an den Rändern der Halbbilder ergeben können. Die zu ignorierenden Spalten und Zeilen in Abhängigkeit zu den oben genannten Verschiebungen ergeben sich für das linke Halbbild zu

$$x_{ignore,low}\left(b_l\right) = \left\lceil -\frac{b_l\left(\vartheta_{Gier}, \vartheta_{Nick}\right) \cdot \left(287 - 0,0456 \cdot \Delta\vartheta_{Nick}\right)}{17340} \right\rceil \tag{6.31}$$

$$x_{ignore,high}\left(b_l\right) = \left\lfloor 17 + \frac{b_l\left(\vartheta_{Gier}, \vartheta_{Nick}\right) \cdot \left(287 - 0,0456 \cdot \Delta\vartheta_{Nick}\right)}{17340} \right\rfloor \tag{6.32}$$

$$y_{ignore,low}\left(b_h\right) = \left\lceil -\frac{b_h\left(, \vartheta_{Nick}\right) \cdot \left(287 - 0,0456 \cdot \Delta\vartheta_{Nick}\right)}{15360} \right\rceil \tag{6.33}$$

$$y_{ignore,high}\left(b_h\right) = \left\lfloor 24 + \frac{b_l\left(\vartheta_{Nick}\right) \cdot \left(287 - 0,0456 \cdot \Delta\vartheta_{Nick}\right)}{15360} \right\rfloor . \tag{6.34}$$

Für das rechte Halbbild werden die selben Werte verwendet, wobei die Werte $x_{ignore,low}\left(b_l\right)$ und $x_{ignore,high}\left(b_l\right)$ um 17 erhöht werden.
Die sich so ergebende Movingmap beinhaltet neben externen Bewegungen auch alle Objekte die sich näher als circa zwei Meter am Roboter befinden. Davon ausgehend, dass keine externe Bewegung vorhanden ist, wird mit dem genannten Vorgehen eine reine Nah- und Fernfeldseparation erreicht. Eine so erzeugt Movingmap für die Nah- und Fernfeldseparation ist in Abbildung 6.2 zu sehen.

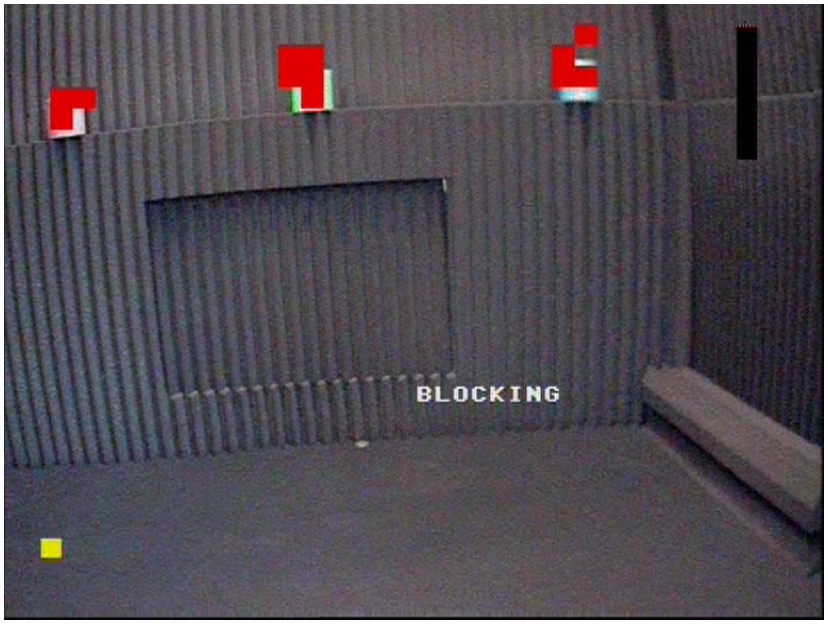


Abbildung 6.2: Visualisierung einer Movingmap die durch die vorgestellte Korrektur-Verschiebung eine Nah- und Fernfeldseparation darstellt. Die rot markierten Superpixel wurden als bewegt eingestuft. Der schwarze Hintergrund hat keinen Einfluss auf die Movingmap, da keine Helligkeits-Kanten enthalten sind.

Für das Verfolgen von externen Bewegungen kann durch die Wahl einer hohen Schwelle zur Movingmap-Erzeugung auch der Einfluss für nahe Objekte reduziert werden.

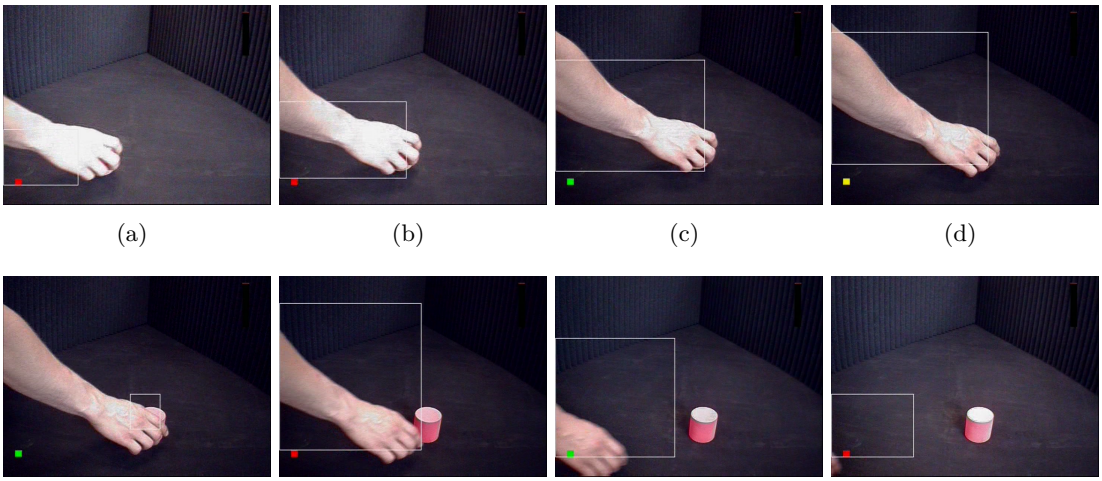
6.3 Objekterkennung in Abhängigkeit von Zeigebewegungen

Nonverbale Gestiken wie das Zeigen stellen eine fundamentale Rolle der visuellen Kommunikation dar. Um ein solche Kommunikation auch mit dem Roboter zu ermöglichen, sollen zeigende Bewegungen erkannt werden. In dieser Anwendung wird davon ausgegangen, dass sich das Zeigen auf Objekte bezieht, die beim Zeigen berührt oder beinahe berührt werden. Wird eine zeigende Bewegung erkannt, soll der bis dahin starre Kopf so gedreht werden, dass das Ziel des Zeigens in die Bildmitte wandert. Während dieser Drehung wird die in Kapitel 6.2 vorgestellte Nah- und Fernfeldseparation durchgeführt, um nahe positionierte Objekte an der Ziel-Stelle zu Erfassen.

Zeigen erkennen
Zum Erkennen von zeigenden Bewegungen wurden zwei Verfahren entwickelt, welche zwei wesentlichen Bedingungen unterliegen. Zum einen wird davon ausgegangen, dass das Zeigen durch einen Trigger angekündigt wird, was zum Beispiel durch eine Sprachanweisung

erfolgen kann. Zum anderen werden mögliche Bewegungen im Bild auf die des Zeigens eingeschränkt. Um eine Zeigegestik zu erkennen ist es notwendig deren Charakteristik zu kennen. Für diese Arbeit wird davon ausgegangen, dass ein Zeigen immer mit einer Bewegung verbunden ist und dass sich das Ende beziehungsweise das Ziel einer solchen Bewegung durch ein Verweilen an dieser Stelle ausdrückt. Das erste Verfahren basiert zudem auf der Annahme, dass sich die genannten Ziele zusätzlich durch eine Richtungsänderung im Bewegungsablauf, vor und nach dem Verweilen, kennzeichnen.

Erstes Verfahren zum Erkennen von zeigenden Bewegungen
Durch den erwähnten Trigger wird eine zyklische Erstellung der in Kapitel 6.1 vorgestellten Movinmap gestartet. Jede dieser Movingmaps wird mit Hilfe der Flächenerkennung (Kap. 4) auf sich bewegende Bildinhalte untersucht. Die so gefundenen Grenzen dienen als Anhaltspunkt für die Auswertung möglicher Richtungsänderungen. Die Abbildungen 6.3 (a)-(h) zeigen einen Bewegungsablauf mit den visualisierten Grenzen.



(a) (b) (c) (d)

(e) (f) (g) (h)

Abbildung 6.3: Der Bewegungsablauf ist zeitlich von (a) nach (h) angeordnet. Die weißen Boxen kennzeichnen die Bereiche die in der jeweiligen Movingmap als Bewegt klassifiziert wurden.

Anhand der sich verschiebenden Grenzen kann die zugehörige Geschwindigkeit ermittelt werden:

$$v_{x,\dots}(t) = x_{Grenze}(t) - x_{Grenze}(t-1) \tag{6.35}$$
$$v_{y,\dots}(t) = y_{Grenze}(t) - y_{Grenze}(t-1) \tag{6.36}$$

Die so entstehenden Geschwindigkeitsverläufe können hinsichtlich der Kriterien der Richtungsänderung und dem Verweilen untersucht werden. Zu diesem Zweck wird nach der in Bewegungsrichtung vorn liegenden Grenze gesucht, da diese einen gleichmäßigen Verlauf in

Bezug auf die Kriterien aufweist. Wenn zum Beispiel mit einem Zeigestab, der eine gleichmäßige Form und Farbe hat, eine geradlinige Bewegung durchgeführt wird, entstehen in der Movingmap ausschließlich an der Spitze als Bewegt klassifizierte Superpixel. In dem Fall, dass der Stab nicht exakt geradlinig bewegt wird, können an beliebigen Stellen des Stabes als bewegt klassifizierte Superpixel entstehen. Diese Superpixel verursachen eine sprunghafte Veränderung der hinteren Grenzen, weswegen diese nicht für eine Auswertung geeignet sind.
Jedoch können sprunghafte Änderung ebenso für die vornliegenden Grenzen entstehen, was durch Klassifzierungsfehler oder langsame Bewegungen verursacht werden kann. Daher wird zusätzlich die Beschleunigung der Grenzen bestimmt:

$$a_{x,\dots}(t) = v_{x,\dots}(t) - v_{x,\dots}(t-1) \tag{6.37}$$
$$a_{y,\dots}(t) = v_{y,\dots}(t) - v_{y,\dots}(t-1). \tag{6.38}$$

Bei zu großen Beschleunigungswerten wird der betroffene Geschwindigkeitswert gleich Null gesetzt. So entstandene Geschwindigkeits- und Beschleunigungsverläufe für zeigende Bewegungen sind in Abbildung 6.4 gezeigt.

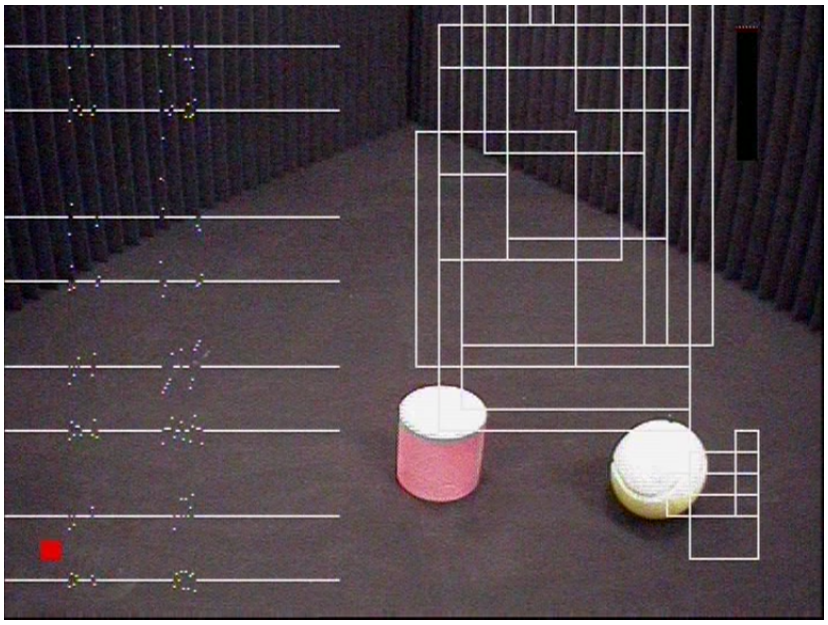


Abbildung 6.4: Die Geschwindigkeits- und Beschleunigungsverläufe der Grenzen der weißen Rahmen sind links im Bild zusehen, von oben nach unten: v_{links}, a_{links}, v_{rechts}, a_{rechts}, v_{unten}, a_{unten}, v_{oben} und a_{oben}. Die rechten Ausschläge in den Verläufen gehören zu den Rahmen am Ball und die Linken zu denen am Zylinder. Einzig für den Verlauf der unteren Grenze beim Zeigen auf den Zylinder lässt sich der erwartete Verlauf erahnen.

Für zeigende Bewegungen lässt sich ein Umkehrpunkt nur unsicher bestimmen, da die erwarteten Verläufe, bestehend aus einem Umkehren und einem Verweilen, nicht sicher

aufzufinden sind. Der Grund für diese Unsicherheit sind sprunghafte Änderungen der Grenzen und zu wenige Messwerte für kleine Bewegungsräume. Eine mögliche Lösung dieser Problematiken ist in Kapitel 7 erläutert.

Zweites Verfahren zum Erkennen von zeigenden Bewegungen
Um dennoch eine zeigende Bewegung zu Erkennen wurde ein weiteres Verfahren entwickelt, was jedoch deutlichen Einschränkungen unterliegt. Neben den bereits genannten Bedingungen basiert dieses Verfahren zum einen auf der Annahme, dass eine Zeigebewegung immer vom Bildschirmrand ausgehend beginnt und dass zum anderen eine Zeigebewegung immer geradlinig und direkt erfolgt. Wie auch das erste Verfahren basiert dieses auf der zyklischen Erstellung von Movingmaps. Sobald in einer dieser Movingmaps eine Bewegung klassifiziert wurde, werden die darauffolgenden Movingmaps aufsummiert. Aufgrund der Bedingung das ein Zeigen immer mit einem Verweilen am Zielpunkt verbunden ist, wird die Summenbildung solange ausgeführt, bis für einen Zeitraum von circa einer Sekunde keine Bewegung klassifiziert wurde. Die so erzeugte Summen-Movingmap beinhaltet den gesamten Bewegungsablauf vom Bildrand bis zur Zielposition. Durch das anschließende Ermitteln einer zusammenhängenden Fläche (nach Kap. 4) ergeben sich die Grenzen des Bewegungsablaufs. Mit der Bedingung, dass eine Zeigebewegung am Bildrand beginnt ergeben sich zwei mögliche Typen von Flächen.

1: Eine Grenze ist deckungsgleich mit dem Bildrand.

2: Zwei Grenzen sind deckungsgleich mit dem Bildrand beziehungsweise eine Ecke der Fläche ist deckungsgleich mit einer Ecke des Bildes.

Unterliegt die resultierende Fläche dem Typ 2 wird als Ziel für das Zeigen die Ecke gewählt, die der deckungsgleichen Ecke gegenüberliegt. Ergibt sich die Fläche wie nach Typ 1 wird der Bereich, welcher der deckungsgleichen Grenze gegenüberliegt, untersucht. Zu diesem Zweck wird der Bereich, senkrecht zur deckungsgleichen Grenze, in zwei Teilbereiche unterteilt. Die äußere Ecke des Teilbereichs, in dem die Summen-Movingmap mehr als bewegt klassifizierte Superpixel enthält, wird als Zielpunkt angenommen. Bei Gleichverteilung wird die Mitte als Zielpunkt gewählt. Abbildung 6.5 veranschaulicht die möglichen Typen wie auch die jeweils resultierenden Zielpunkte.

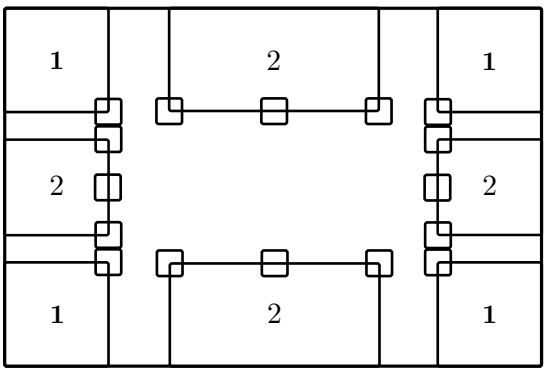


Abbildung 6.5: Prinzipielle Typen für das Erkennen von zeigenden Bewegungen sind die beschrifteten mittelgroßen Bereiche. Die überlappenden kleinen Bereiche symbolisieren die möglichen Zielpunkte.

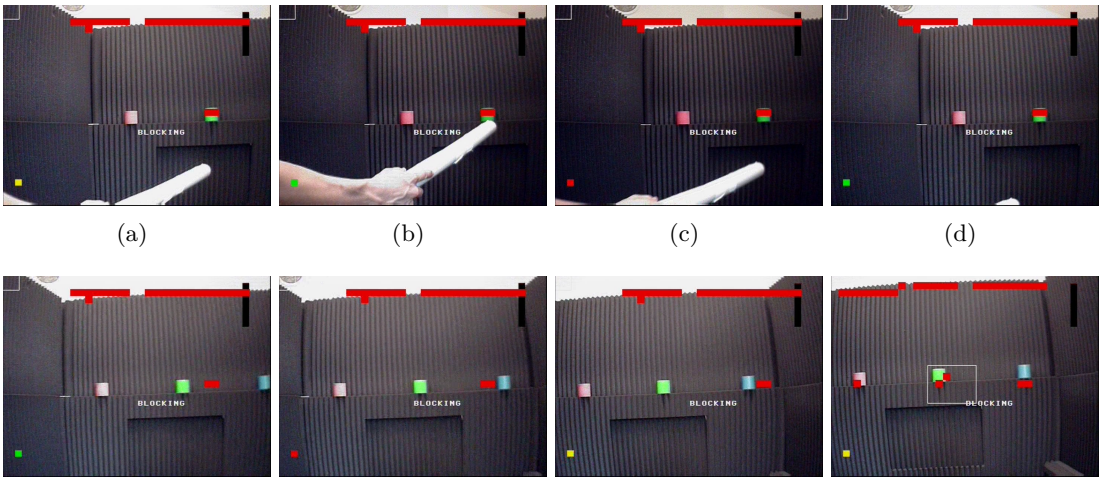
Dieses statische Verfahren dient im wesentlichen dem Anschauungszweck für das Drehen des Kopfes in Richtung des Zielpunktes und die währenddessen durchzuführende Nah- und Fernfeldseparation. Für weiterführende Arbeiten ist das universellere erste Verfahren besser geeignet, sofern die Lösungsansätze aus Kapitel 7 berücksichtigt werden.

Drehen des Kopfes in Richtung des Zielpunktes mit Nah- und Fernfeldseparation

Unabhängig von der Detektionsweise einer zeigenden Bewegung, ergibt sich für einen bekannten Zielpunkt das Drehen des Kopfes in diese Richtung. Währenddessen wird eine Nah- und Fernfeldseparation durchgeführt. Der vollständige Ablauf ist im Folgenden erläutert.

Wie schon bei der Bewegungsverfolgung wird der Kopf so gedreht, dass sich der Zielpunkt in die Mitte des Bildschirms verschiebt. Die Berechnung der notwendigen Sollwerte in Abhängigkeit des Zielpunktes wird wie in den Formeln 6.18 und 6.20 durchgeführt. Während der Drehung des Kopfes werden die Abweichungen der Momentanwinkel von den Sollwinkeln ermittelt. Sobald beide Abweichungen betragsmäßig unter 1000 (Winkeleinheiten) liegen, wird die in Kapitel 6.2 vorgestellte Nah- und Fernfeldseparation durchgeführt, wobei die resultierenden Movingmaps bis zum vollenden der Kopfdrehung aufsummiert werden. Die Einschränkung auf kleine Abweichungen der Winkelwerte (1000) ist notwendig, da die Nah- und Fernfeldseparation auf eine Verschiebung von $-\beta$ ausgelegt ist und sich nahe positionierte Objekte mit bis zu -2β verschieben. Würden die Movingmaps für die gesamte Drehung aufsummiert werden, so ergäbe sich für große Drehungen eine großer Bewegungsverlauf, was die Lokalisierung des Objektes erschweren würde. Die Summen-Movingmap, welche die begrenzte Nah- und Fernfeldseparation beinhaltet, wird nach dem Erreichen der Sollwerte ausgewertet. Da sich das potentielle Zielobjekt nach dem Drehen näherungsweise in der Mitte des Bildschirms befinden sollte, wird die Mitte der Summen-Movinmap in einem Bereich von 8x8 Superpixeln auf klassifizierte Bereiche untersucht.

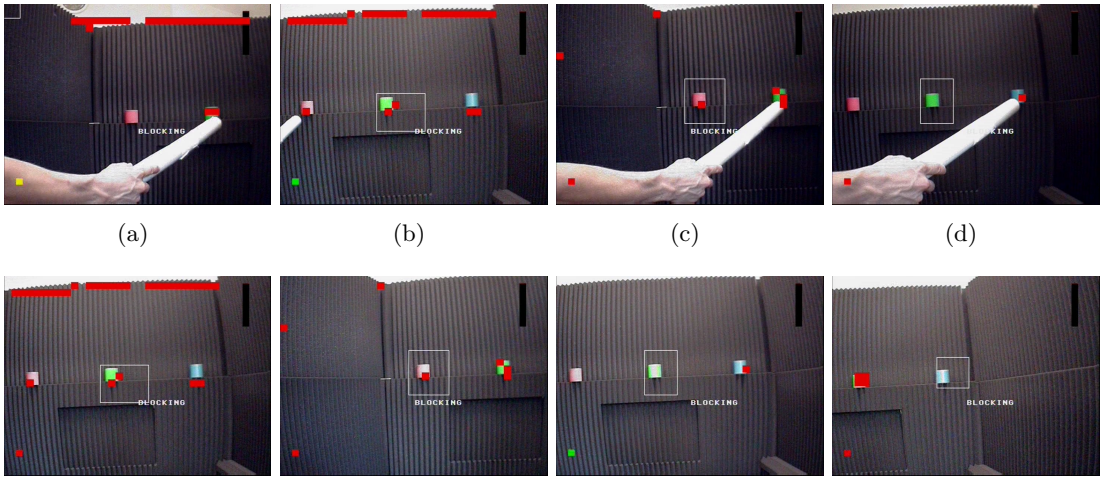
Ein klassifizierter Bereiche wird Lokalisiert und als Zielobjekt des Zeigevorgangs interpretiert. Ein vollständiger Ablauf vom Zeigen bis zum Lokalisieren eines Objektes ist in den Abbildungen 6.6 (a)-(h) gezeigt. In den Abbildungen 6.7 (a)-(h) sind in den oberen Bildern (a-d) jeweils die Zielpositionen beim Zeigen zu sehen. Die unteren Bilder (e-h) zeigen das zugehörige lokalisierte Objekt weiß umrahmt. Die Grenzen lokalisierter Objekte können von weiteren Bildverarbeitungsalgorithmen benutzt werden. So könnte der resultierende Bereich auf Farbverteilungen untersucht werden, um die Farbe des Objekts zu bestimmen. Für den Fall, dass sich mehrere potentielle Flächen innerhalb des zu untersuchenden Bereichs befinden, muss noch eine zusätzliche Funktion entwickelt werden, die gegebenenfalls überprüft, ob nähere Informationen (z.B. Fragen oder Auffordern zum erneuten Zeigen) gewonnen werden können.



(a) (b) (c) (d)

(e) (f) (g) (h)

Abbildung 6.6: Die Bilder zeigen ausschnittsweise den zeitlichen Ablauf vom Zeigen auf ein Objekt (a-d) über die Drehung des Kopfes (e-g) und schlussendlich das lokalisierte Objekt (h). Der weiße Rahmen um das Objekt wird aus den gefunden Grenzen generiert und ist gleichzeitig als resultierender Bereich des Verfahrens zu verstehen. Die rot markierten Superpixel sind das Ergebnis der Nah- und Fernfeldseparation, wobei hierfür eine andere Schwelle wie für das Finden der Grenzen verwendet wurde.



(a) (b) (c) (d)

(e) (f) (g) (h)

Abbildung 6.7: In den oberen Abbildung ist jeweils die Zielposition einer zeigenden Bewegung zu sehen (a-d). Die unteren Abbildungen zeigen die zugehörigen lokalisierten Objekte nach dem Drehen des Kopfes (e-h).

Kapitel 7

Ausblick

Die vorgestellten Verfahren können durch weiterführende Arbeiten verbessert und erweitert werden. Zudem kann die Eingliederung in das in Kapitel 1 erwähnte Naming Game vorgenommen werden. Im Folgenden wird eine Auswahl an möglichen Ansatzpunkten der einzelnen Verfahren erörtert.

Pixel-Pipeline
Um die Debugeigenschaften der Pixel-Pipeline zu erhöhen, könnte eine Overlayfunktion entwickelt werden, die zum Beispiel ausgewählte Farbklassen aufgreift und die Pixel in den Video-Daten entsprechend einfärbt, so dass in Echtzeit sichtbar wird welcher Pixel in welche Farbklasse fällt.
Der nicht definierte Modus (siehe Color-Classifier Kap. 3) bei der Wahl des Farbmodus könnte durch einen zusätzlichen HSV-Kanal belegt werden. Bei einer Addition müssen jedoch die Farbwerte H im Winkelmaß gesondert verrechnet werden.

Flächenerkennung und -Lokalisierung
Um die Geschwindigkeit zu erhöhen und Prozessor zu entlasten, könnte die Auswertung der Summenvektoren in den FPGA ausgegliedert werden. Die rekursive Struktur würde so nur noch mit dem Verwalten der externen Berechnungen beschäftigt sein. Das wiederum ermöglicht eine Kommunikation mit höheren Instanzen wie einer Verhaltensteuerung, welche ihrerseits das Unterbrechen des Ablaufs wie auch das priorisieren beziehungsweise ignorieren bestimmter Objekte veranlassen könnte.

Entfernungsschätzung
Die Entfernungsschätzung kann bereits bei kleinen Pixel-Fehlern, beim Bestimmen der Verschiebung in Pixeln, sehr große Fehler in der Distanz aufweisen. Aus diesem Grund bietet sich hier die Möglichkeit ein geeignetes Verfahren zu entwickeln, das diese Schätzfehler gering hält. Bei der Eingliederung in das Naming Game kann ein möglicher Objekt-Manager dahingehend konzipiert werden, dass er die in Abbildung 5.7 gezeigten Grenzen berücksichtigt und so der jeweiligen Entfernungsschätzung einen Zuverlässigkeits-Wert zuordnen kann.

Visuomotorische Verfahren
Die Entstehung der Movingmap hängt maßgeblich von der Helligkeitsdifferenz-Schwelle ab. Um auch langsame Bewegungen oder Bewegungen von Objekten mit geringer Helligkeitsdifferenz zu erfassen, sollte diese Schwelle dynamisch gestaltet werden. Zusätzlich könnten neben Helligkeitswerten auch die Farbwerte für die Erzeugung einer Movingmap herangezogen werden. Die Farbdifferenz müsste jedoch im HSV-Farbraum bestimmt werden, um eine Aussage unabhängig von der Sättigung zu erzielen.
Um das Erkennen von zeigenden Bewegungen mittels dem Finden von Umkehrpunkten zu ermöglichen, wäre der erste sinnvolle Schritt das Eingrenzen des Bereichs zum Bilden der Movingmap. Diese Einschränkung sollte so gewählt werden, dass sich die Anzahl der notwendigen Aufnahmen halbiert (z.B. nur die linke Hälfte des Bildes). Die Anzahl der Messwerte für die Geschwindigkeitsverläufe der Grenzen würde sich so verdoppeln, was das Finden der vornliegenden Grenzen und der Umkehrpunkte erleichtern würde.
Das eher statische summen-basierte Verfahren zum Erkennen von Objekten auf die gezeigt wurde, würde durch zwei einfache Erweiterungen deutlich flexibler werden. Zum einen über ein gewichtetes Aufsummieren der Movingmaps, wobei das Gewicht mit jeder neuen Movingmap steigt. Zum anderen könnte diese Gewichtung bei einer Richtungsänderung oder einem Verweilen von Null beginnend mit umgekehrten Vorzeichen gestartet werden. Mit diesem Prinzip können ganze Bewegungsabläufe inklusive einer Richtungs- und Pause-Information festgehalten werden. Die Bereiche mit den jeweils minimalen beziehungsweise maximalen Werten repräsentieren Ruhe- und Umkehrpunkte, was als mögliche Zeige-Ziele verstanden werden kann.

Literaturverzeichnis

[Ana15] Analog Devices. *Low Power, Chip Scale, 10-Bit SD/HD Video Encoder Data Sheet ADV7390/ADV7391/ADV7392/ADV7393*, 2015. Rev I.

[BBV00] James Bruce, Tucker Balch, and Manuela Veloso. Fast and Cheap Color Image Segmentation for Interactive Robots. In *Proceedings of WIRE-2000, Worskshop on Interactive Robotics and Entertainment*, May 2000.

[HSB+11] Manfred Hild, Torsten Siedel, Christian Benckendorff, Matthias Kubisch, and Christian Thiele. Myon: Concepts and Design of a Modular Humanoid Robot Which Can Be Reassembled During Runtime. In *Proceedings of the 14th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Paris, France, September 2011.

[Int07] International Telecommunication Union. *Recommendation ITU-R BT.656-5*, 2007.

[Int11] International Telecommunication Union. *Recommendation ITU-R BT.601-7*, 2011.

[Mas14] Ivan Masár. An Image Processing System for Visual Servoing of Mobile Robots and Object Tracking , 2014.

[Met16] Evgeni Metzler. Entwicklung und Implementierung eines interaktiven Lernverfahrens für humanoide Roboter. Bachelorarbeit, 2016.

[SL12] Luc Steels and Martin Loetzsch. The grounded naming game. *Experiments in cultural language evolution*, 3:41–59, 2012.

[SR14] Herbert Süße and Erik Rodner. *Bildverarbeitung und Objekterkennung: Computer Vision in Industrie und Medizin (German Edition)*. Springer Vieweg, 2014.

[Thi14] Christian Thiele. Design der verteilten Echtzeit-Systemarchitektur DISTAL und Implementierung am Beispiel des humanoiden Roboters Myon. Diplomarbeit, 2014.

[Vid11] Videology Imanging Solutions Inc. *PDS-20K15XDIG*, 2011. Rev C.

[Vid12] Videology Imanging Solutions Inc. *Application Note 20/21K13XDIG(B&W) 20/21K14XDIG(Standard Resolution) 20/21K15XDIG(High Resolution)*, 2012. Rev J.

[Wur16] Jonas Wurst. Praxisbericht: Grundlagen der visuellen Warnehmung des humanoiden Roboters Myon. Beuth Hochschule für Technik Berlin, 2016.