



Berliner
Hochschule
für Technik

Fachbereich VI - Informatik und Medien
Technische Informatik Bachelor

Bachelorarbeit

zur Erlangung des Akademischen Grades

Bachelor of Engineering (B.Eng.)

Entwurf und Realisierung eines echtzeitfähigen Farbsortier-Demonstrators

Anton Kreß

Matrikelnr.: S872899

Betreuer: Prof. Dr.-Ing. S. Voß
Referent: Prof. Dr.-Ing. P. Gregorius

Abgabedatum: 12. Juli 2022

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Berlin, den 12. Juli 2022

(Unterschrift)

Anton Kreß

Zusammenfassung

In dieser Arbeit wird ein echtzeitfähiger Farbsortier-Demonstrator konstruiert. Dieser ist in der Lage, Testobjekte gemäß ihrer jeweiligen Farbe in unterschiedliche Fächer zu sortieren.

Dazu wurde ein Versuchsaufbau konstruiert, der diese Testobjekte mittels eines Transportbandes durch einen Tunnel bewegt. Innerhalb dieses Tunnels befindet sich ein sogenannter Detektionsbereich, in welchem Bilddaten unter standardisierten Bedingungen aufgenommen werden. Diese Bilddaten werden auf die Präsenz eines Testobjekts untersucht. Wird ein Testobjekt innerhalb der Bilddaten detektiert, durchlaufen die Bilddaten einen Algorithmus, welcher die Farbe des Testobjekts erkennt.

Am Ende des Tunnels befindet sich eine mechanische Sortierung, die auf Grundlage der erkannten Farbe gesteuert wird. Ein Schieber nimmt eine entsprechende Position ein, durch welche das analysierte Testobjekt in das korrekte Fach bewegt wird.

Während des gesamten Ablaufs unterliegt der Farbsortier-Demonstrator Zeitbedingungen, deren Einhaltung für den Erfolg dieser Arbeit grundlegend sind.

Abstract

In this thesis, a real-time capable color sorting demonstrator is constructed. This is capable of sorting test objects into different compartments according to their respective color. For this purpose, an experimental setup was constructed that moves these test objects through a tunnel by means of a conveyor belt. Inside this tunnel is a so-called detection area in which image data are recorded under standardized conditions.

These image data are examined for the presence of a test object. If a test object is detected within the image data, the image data runs through an algorithm that recognizes the color of the test object.

At the end of the tunnel is a mechanical sort that is controlled based on the detected color. A slider takes an appropriate position through which the analyzed test object is moved to the correct compartment.

Throughout the process, the color sorting demonstrator is bound by timing constraints, compliance with which is fundamental to the success of this work.

Inhaltsverzeichnis

Abbildungsverzeichnis	VIII
Tabellenverzeichnis	IX
Abkürzungsverzeichnis	X
1 Einleitung	1
1.1 Aktuelle Forschungen	2
1.2 Zielstellung	3
2 Theoretische Grundlagen	5
2.1 Farbmodelle	6
2.1.1 RGB-Farbmodell	6
2.1.2 YCbCr-Farbmodell	7
2.2 Entwicklungsplattform	10
2.2.1 Raspberry Pi	10
2.2.2 RaspiCam	12
2.2.3 General Purpose Input Output	13
2.3 System und UNIX-Prozesse	14
2.3.1 Allgemeines zu UNIX Prozessen	14
2.3.2 Verteilte Prozesse	15
2.3.3 Interprozess-Kommunikation	15
2.4 Aktorik und Sensorik	17
2.4.1 Schrittmotor NEMA 17	17
2.4.2 Motortreiber A4988	17
2.4.3 Gleichstrommotor Modelcraft RB350100-0A101R	17
2.4.4 Leuchtdiode	18
2.4.5 Relais Modul	19
2.4.6 Lichtschranke	20
2.5 Echtzeit	22
3 Anforderungsanalyse	24
4 Konstruktion Versuchsaufbau	26
4.1 Transportband	31
4.1.1 Aufbau des Transportbandes	31
4.1.2 Ansteuerung des Transportbandes	31
4.1.3 Funktionsweise des Transportbandes	31

4.2	Detektionsbereich	35
4.2.1	Aufbau des Detektionsbereichs	35
4.2.2	Ansteuerung des Detektionsbereichs	36
4.2.3	Funktionsweise des Detektionsbereichs	40
4.3	Mechanische Sortierung	46
4.3.1	Aufbau der mechanischen Sortierung	46
4.3.2	Ansteuerung der mechanischen Sortierung	49
4.3.3	Funktionsweise der mechanischen Sortierung	52
5	Software-Implementierung	57
5.1	Software Architektur	58
5.1.1	Software Komponenten	58
5.1.2	Programmablauf	61
5.2	Farberkennung	66
5.2.1	Herleitung des Farberkennungs-Algorithmus	66
5.2.2	Implementierung des Farberkennungs-Algorithmus	70
5.3	Sortiersteuerung	77
5.3.1	Herleitung des Sortiersteuerungs-Algorithmus	77
5.3.2	Implementierung des Sortiersteuerungs-Algorithmus	79
6	Ergebnisse	83
7	Ausblick	86
Literatur		87
Anhang		90

Abbildungsverzeichnis

2.1	RGB-Farbwürfel [13]	7
2.2	YCbCr-Farbwürfel [13]	7
2.3	YCbCr Datenwort	8
2.4	YCbCr-Flächen [13]	8
2.5	Blockschaltbild Cortex-A72 [5]	11
2.6	Hardware-Komponenten des Raspberry Pi's [8]	12
2.7	Prozesszustände nach UNIX Prozessmodell [20]	15
2.8	Schaltplan A4988 Motortreiber [1]	18
2.9	Steuer- und Laststromkreis einer Relais Schaltung [17]	19
2.10	Funktionsweise Einweg-Lichtschranke [4]	20
2.11	Funktionsweise Reflex-Lichtschranke [4]	21
2.12	Zeitbedingungen	22
4.1	Verortung der Teilsysteme	26
4.2	Schaltplan der elektrischen Komponenten	28
4.3	Technische Zeichnung - Gesamter Versuchsaufbau	30
4.4	Aufbau Transportband (Tunnel ausgeblendet)	32
4.5	OUT-Signale der Lichtschranken	37
4.6	Detektionsbereich Aufbau	39
4.7	Zeitverhalten des Detektionsbereichs	45
4.8	Mechanische Sortierung Aufbau	48
4.9	Hilfszeichnung für die Berechnung der Rutschen-Parameter	51
4.10	Auszug der Messung für die Ermittlung von Δt_S	53
4.11	Zeitverhalten der mechanischen Sortierung	56
5.1	Blockschaltbild des Programms	59
5.2	Zustandsmaschine des Programms	62
5.3	Sequenzdiagramm des Programms	65
5.4	CbCr-Farbskala mit Indizes der Histogrammfelder	67
5.5	Durchschnittliche Histogramme der Messobjekte auf CbCr-Farbskala	68
5.6	Ablaufdiagramm Farberkennungs-Algorithmus	71
5.7	Vergleichsbild vom Detektionsbereich	72
5.8	Bild eines Testobjekts	73
5.9	Differenzbildverfahren in verschiedenen Farbmodellen	74
5.10	Histogramm eines blauen Testobjekts	75
5.11	Vergleich der Farbanteile	76
5.12	Steuersignale an den Motortreiber des Schrittmotors (Rot: Enable, Grün: Direction, Gelb: Step)	78

5.13 Ablaufdiagramm Sortiersteuerungs-Algorithmus	80
6.1 Übersicht Validierungstests	84
6.2 Zeitverhalten des Farbsortier Demonstrators	85
7.1 Grünes Objekt	93
7.2 Blaues Objekt	93
7.3 Orangenes Objekt	94
7.4 Rotes Objekt	94
7.5 Gelbes Objekt	95
7.6 Braunes Objekt	95

Tabellenverzeichnis

1.1	Definierte Farben der Testobjekte [15]	4
2.1	Zeitbedingungen in der Echtzeit [32]	22
2.2	Kategorien der Echtzeitbedingungen	23
4.1	Bildverarbeitungszeiten des Programms <i>raspistill</i>	40
5.1	Zustände des Programms	61
5.2	Farbmasken und Farbmultiplikatoren der Zielfarben	69
7.1	Zeipunkte	90
7.2	Wegpunkte	90
7.3	Zeitintervalle	91
7.4	Winkel	91
7.5	Strecken, Distanzen, Radien, Umfänge	92
7.6	Sonstige	92

Abkürzungsverzeichnis

SoC System on a Chip

CPU Central Processing Unit

RAM Random Access Memory

AMBA Advanced Microcontroller Bus Architecture

SCU Snoop Control Unit

AES Advanced Encryption Standard

AXI Advanced eXtensible Interface

GPIO General Purpose Input Output

CSI Camera Serial Interface

HDMI High Definition Multimedia Interface

USB Universal Serial Bus

SPI Serial Peripheral Interface

I2C Inter-Integrated Circuit

COB Chip-On-Board

MIMD Multiple Instruction Multiple Data

FIFO First In First Out

1 Einleitung

Eine performante und anpassbare Bilderkennung ist aus der heutigen Welt nicht mehr wegzudenken. Fahrassistentensysteme lassen Autos automatisch die Spur halten oder leiten eine Gefahrenbremsung ein, bevor es zur Kollision kommt. In der Industrie wurde der menschliche Fließbandarbeiter vom Roboterarm abgelöst. Und autonome Roboter können sich in unbekannter Umgebung nicht nur bewegen, sondern auch mit dieser interagieren.

Auch der Blick in die Zukunft zeigt die herausragende Bedeutung der Bilderkennung. Stetig verbesserte Hardware bietet die Grundlage für die immer schnellere Verarbeitung von Bildern mit immer höheren Auflösungen. In Kombination mit künstlicher Intelligenz entstehen so weiterhin neue Anwendungsgebiete für die Bilderkennung.[10]

Ein wichtiger Teilbereich der Bilderkennung ist die Farberkennung. Durch die Erkennung von Farben können bestimmte Objekte auch in unübersichtlichen Bildausschnitten separiert und identifiziert werden. In der Praxis findet sich dies etwa bei der Bremslicht-Erkennung im Auto. Des Weiteren ermöglicht die Farberkennung eine Klassifizierung von unterschiedlichen Objekten mit ähnlicher Form.[6]

Eine verlässliche Echtzeitfähigkeit stellt die Grundlage für die genannten Anwendungen dar. Bei der Bremslichterkennung hätte eine hohe Latenz beispielsweise schwerwiegende Folgen. Auch in der Industrie entscheidet die Geschwindigkeit der Signalverarbeitung über die Produktivität und letztlich die Wirtschaftlichkeit einer Fabrik.

1.1 Aktuelle Forschungen

Zur erfolgreichen Realisierung dieser Forschungsarbeit muss die Entwicklungsplattform in der Lage sein, Objekte vom Hintergrund unterscheiden zu können. Eine geeignete Umsetzung für dieses Problem wäre das Differenzbildverfahren. In diesem werden zwei Bilder voneinander subtrahiert, um Bildunterschiede ausfindig zu machen und so ein bewegtes Objekt zu erkennen. [27]

Zur Identifikation von Farben finden sich je nach Anwendungsfall verschiedene Möglichkeiten. Dabei hängt die Methodik vom Farbraum ab, in welchen die Rohdaten der Kamera umgewandelt werden. So kann beispielsweise der YCbCr-Farbraum genutzt werden, um Zugriff auf die Helligkeitswerte zu haben und eventuell auftretende Schatten zu beseitigen.[24]

Doch auch der RGB-Farbraum findet Verwendung in der Farberkennung. Hier lassen sich die einzelnen Farbkanäle Rot, Grün und Blau leicht separieren und gut weiterverarbeiten. Es kann nach einzelnen Farben gefiltert werden, indem nicht relevante Farbkanäle auf Null gesetzt werden.[6]

Der RGB-Farbraum kann ebenfalls sehr einfach in den Normalisierten RGB-Farbraum umgewandelt werden, welcher bei bestimmten Oberflächen eine bessere Verarbeitung ermöglicht. Darüber hinaus finden noch weitere Farbräume in der Forschungsliteratur Erwähnung. Für die hier beschriebene Forschungsarbeit scheinen Versuche mit dem YCbCr- und dem RGB-Farbraum das größte Potential zu haben.[30]

Am Fachbereich VI der Technischen Hochschule Berlin findet sich zudem ein breites Band an Expertise zum Thema der Bildverarbeitung, wie etwa in der Personenerkennung [27] oder der Lichtfeldabbildung[31].

Mit dieser Arbeit möchte ich einen eigenen Beitrag leisten und die Forschung in diesem Bereich vorantreiben.

1.2 Zielstellung

Das Ziel dieser Arbeit soll sein, einen robusten, echtzeitfähigen Farbsortierer zu entwerfen und zu testen. Er hat die Aufgabe, bewegte Objekte anhand ihrer Farbe zu erkennen und anschließend eine Sortierung vorzunehmen. Der Farbsortier-Demonstrator setzt sich aus verschiedenen Bestandteilen zusammen:

- Raspberry Pi 4
- Kameramodul vom Typ RaspiCam
- Transportband, angetrieben durch Gleichstrommotor
- Sortiermechanik, angetrieben durch Schrittmotor
- LED
- Reflex-Lichtschranke (2x)

Der Raspberry Pi stellt das Kernstück des Farbsortierers dar. Er hat die Aufgabe, die gesamte Bildverarbeitung umzusetzen und die Mechanik der Sortierung zu steuern. Essentiell wird dabei eine verlässliche Bilderkennung sein, die unter Einhaltung von Echtzeitbedingungen die Testobjekte und deren Farbe identifiziert.

Zur Erkennung der Farben könnte der RGB-Farbraum genutzt werden. Pro Pixel würde der oder die Farbkanäle mit dem höchsten Wert herausgelesen werden und mit den Schwellenwerten vordefinierter Farben, welche in einem Look-Up-Table gespeichert sind, verglichen werden. Befinden sich die Werte des Pixels innerhalb der Schwellenwerte, könnte dieser auf Grund dessen klassifiziert werden.

Die konkrete Zielstellung ist die folgende: Objekte sollen einzeln auf das Transportband gelegt werden können und vom Modell eigenständig in unterschiedliche Fächer sortiert werden. Nach Abschluss der Sortierung soll es möglich sein, ein nächstes Objekt auf das Transportband zu legen und den Testablauf zu wiederholen. Den Erfolg meines Modells wird anhand der Trefferquote gemessen. Dabei soll eine Trefferquote von mindestens 70 % korrekt sortierter Objekte erreicht werden.

Es sollen sechs verschiedene Farben unterschieden und erkannt werden können. Als Testobjekte dienen farbige Schokobonbons. Diese haben die Farben Rot, Grün, Blau, Gelb, Orange und Braun. In Tabelle 1.1 sind exemplarische Fotos dieser Testobjekte mit zugeordneter Farbe. Es liegen keine Informationen über die Zusammensetzung der Farben (Farbcodes) vor, mithilfe derer Schwellwerte gebildet werden könnten.

Aufnahme Testobjekt	Farbe
	Rot
	Orange
	Gelb
	Grün
	Blau
	Braun

Tabelle 1.1: Definierte Farben der Testobjekte [15]

2 Theoretische Grundlagen

Im Folgenden werden die theoretischen Grundsätze der verwendeten Bestandteile der Arbeit erläutert. Dabei werden zuerst die Grundlagen zu den verwendeten Farbmodellen dargelegt. Hierbei werden deren jeweiligen Vor- und Nachteile vorgestellt und aufgezeigt, wie Farbwerte zwischen den Farbmodellen konvertiert werden können. Es folgt eine Vorstellung der verwendeten Entwicklungsplattform. Dabei wird besonders auf die Systemleistung und ihre Schnittstellen nach außen eingegangen. Passend dazu wird darauffolgend die UNIX-Prozesswelt beleuchtet, da diese in der Konzeption der Software eine große Rolle spielt.

Im Anschluss wird auf die verwendete Aktorik und Sensorik eingegangen. Zuletzt erfolgt eine Definition von Echtzeit und den zugehörigen Echtzeitbedingungen.

2.1 Farbmodelle

Die computergestützte Bildverarbeitung ist darauf angewiesen, Farben numerisch und in mathematischer Form darzustellen. Farbmodelle sind die digitale Repräsentation der darstellbaren Farben im meist dreidimensionalen Raum. Dabei unterscheiden sich die Farbmodelle in der Wahl ihrer darstellenden Parameter und der Berechnung dieser. Die Eigenschaften, die zur Unterscheidung verschiedener Farben verwendet werden, sind Helligkeit, Farbton und Sättigung. Diese können in die zwei Gruppen *Luminanz* oder auch *Helligkeit* sowie *Chrominanz*, also *Farbigkeit* unterteilt werden. Auf diese Weise kann jede Farbe anhand ihrer *Chrominanz*- und *Luminanz*-Werte eindeutig dargestellt werden. [14][13][11][2]

Dabei gibt es verschiedene Farbmodelle, die eine bestimmte Farbe auf unterschiedliche Weise repräsentieren können. Weiterhin ist möglich, Farben von einem Farbmodell in ein anderes zu konvertieren. Durch die unterschiedlichen Charakteristika der verschiedenen Farbmodelle ergeben sich je nach Anwendungsfall individuelle Vor- und Nachteile. Die Wahl des Farbmodells ist deshalb richtungsweisend für die Wahl der eingesetzten Methoden.

Aufgrund der großen Menge unterschiedlicher Farbmodelle, wird darauf verzichtet einen umfassenden Vergleich aller Farbmodelle vorzunehmen. Stattdessen wird nachfolgend das RGB- sowie das YCbCr-Farbmodell beleuchtet werden, da sie aus verschiedenen Gründen zentrale Komponenten dieser Arbeit ausmachen.

2.1.1 RGB-Farbmodell

Das RGB-Farbmodell setzt sich aus den monochromen Spektren Rot, Grün und Blau zusammen, welche ihm seinen Namen geben. Durch die proportionale Mischung dieser monochromen Spektren ergibt sich eine bestimmte Farbe. Das RGB-Farbmodell wird aus diesem Grund auch als additives Farbmodell bezeichnet. Es kann als RGB-Farbwürfel mit normalisierten Werten zwischen 0 und 1 für die Farbkanäle Rot, Grün und Blau dargestellt werden (siehe Abbildung 2.1).[13][16]

Kameras nehmen Eingangssignale üblicherweise in den drei Farben Rot, Grün und Blau im Wertebereich zwischen 0 und 255 auf. Mit der Verwendung einer Kamera ist demnach davon auszugehen, dass eingehende Bilddaten grundsätzlich in diesen Farben vorliegen.[13]

Im Bereich der Bildverarbeitung finden sich einige Arbeiten, die mit dem RGB-Farbmodell arbeiten. Einer der Gründe dafür ist, dass die Bilddaten bei der Aufnahme bereits in diesem Farbraum vorliegen und keine Konvertierung vorgenommen werden muss. Außerdem wird das RGB-Farbmodell als rechnerisch einfaches System betrachtet. Hinzu kommt, dass sich besonders die Farben, die durch die Ecken des RGB-Farbwürfels beschrieben werden, relativ einfach darstellen bzw. detektieren lassen.[12][16]

Sollte jedoch die Detektion mehrerer Farben in feineren Abstufungen gefordert werden, wird die Arbeit im RGB-Farbraum komplizierter. Durch die Addition der monochromen Spektren zur Generierung einer Farbe, ist es schwierig, einen bestimmten Farbton

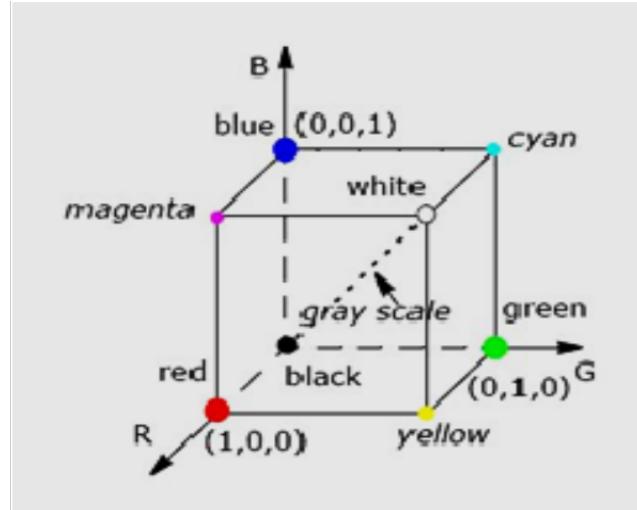


Abbildung 2.1: RGB-Farbwürfel [13]

zu bestimmen. So sind farbbasierte Bildanalysen auf Grundlage des RGB-Farbmodells in der Praxis eher unüblich. [23][14][13]

2.1.2 YCbCr-Farbmodell

Das YCbCr-Farbmodell ist Teil der YUV-Farbmodell-Familie und findet in der digitalen Bildverarbeitung eine breite Anwendung [13]. Die Luminanz wird durch die Y-Komponente dargestellt, wohingegen die Cb- und Cr-Werte die Chrominanz repräsentieren. Folglich bildet sich auch im YCbCr-Farbmodell ein dreidimensionaler Farbraum (siehe Abbildung 2.2). Üblich ist die Darstellung in 24 Bit, die sich wie nach Abbildung 2.3 aufteilen.

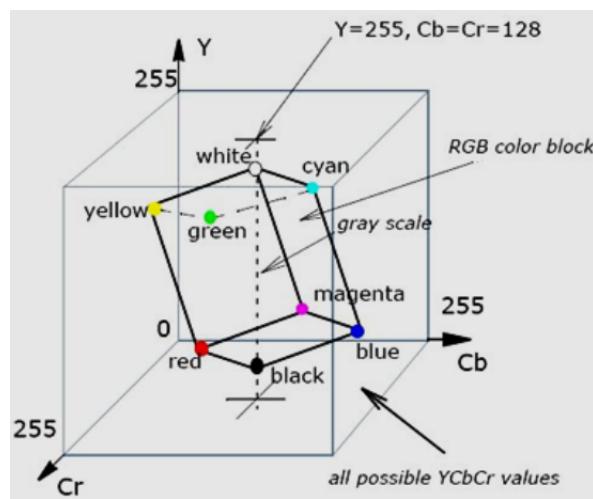


Abbildung 2.2: YCbCr-Farbwürfel [13]

23	16 15	8 7	0
Y Luminanz	Cb Chrominanz	Cr Chrominanz	

Abbildung 2.3: YCbCr Datenwort

Die Trennung zwischen Luminanz und Chrominanz stellt einen wichtigen Unterschied zum RGB-Farbraum dar. Ein Datenwort im YCbCr-Farbmodell kann problemlos auf die Chrominanzwerte Cb und Cr reduziert werden. Das Bild ist somit robuster gegenüber Änderungen in der Helligkeit [2][27]. In Abbildung 2.4 ist gut zu erkennen, dass sich die Farben auch bei großen Helligkeitsunterschieden verhältnismäßig gleichbleibend auf der CbCr-Ebene verteilen. Weiterhin verringert sich die Größe des benötigten Speicherplatzes für einen Farbwert dadurch auf 16 Bit, was wiederum zu einem höheren Datendurchsatz führt.[13]

Sollte der reine Farbwert nur noch durch die zwei Farbkanäle Cb und Cr repräsentiert werden, könnten sich die Schwellwerte zur Erkennung von Farben einfacher bestimmen lassen, als in anderen Farbmodellen. [2]

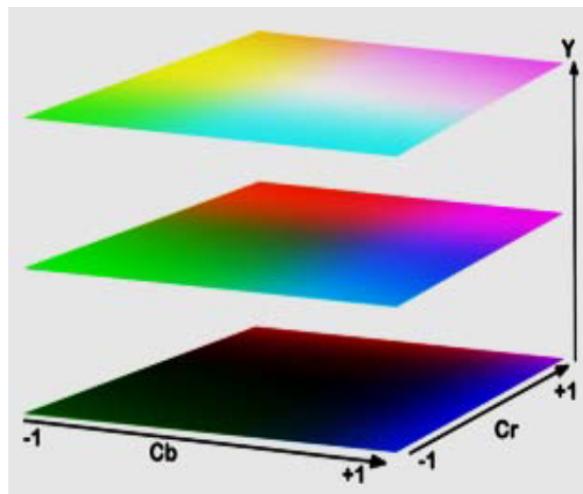


Abbildung 2.4: YCbCr-Flächen [13]

Zwischen dem RGB- und dem YCbCr-Farbmodell lässt sich linear und verlustfrei in beide Richtungen konvertieren. So zeigen Gleichung (2.1) und Gleichung (2.2) jene Konvertierungen zwischen dem RGB-Farbraum und dem YCbCr-Farbraum. Es ist zu beachten, dass in diesem Fall die Werte auf den Bereich von 0 bis 255 normiert wurden, was einem 24-Bit Datenwort entspricht.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (2.1)$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0.000 & 1.596 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} (Y - 16) \\ (Cb - 128) \\ (Cr - 128) \end{bmatrix} \quad (2.2)$$

2.2 Entwicklungsplattform

Zur erfolgreichen Realisierung des echtzeitfähigen Farbsortier-Demonstrators ist es notwendig, eine Steuereinheit zu implementieren, die verschiedene Anforderungen erfüllt. Die Steuereinheit muss in der Lage sein, Bilddaten aufzunehmen und diese innerhalb einer definierten Zeit zu verarbeiten. Weiterhin müssen Signale von externer Sensorik empfangen und an externe Aktorik gesendet werden.

Daraus ergibt sich eine breite Palette an Entwicklungsplattformen, die als Steuereinheit genutzt werden könnten. Im Zuge dieser Arbeit wurde sich für einen Raspberry Pi, Modell 4 als Entwicklungsplattform entschieden. Die Vielzahl an Schnittstellen und deren einfache Konfigurierung sowie der leistungsstarke ARM Cortex-A72 Prozessor machen den Raspberry Pi 4 zu einer geeigneten Steuereinheit für den Prototypen eines echtzeitfähigen Farbsortierers.

2.2.1 Raspberry Pi

Der Raspberry Pi ist ein sogenannter Einplatinencomputer, bei dem alle wesentlichen Komponenten wie CPU¹ oder RAM² auf einer Leiterplatte verbaut sind. Der Kern des Raspberry Pi's besteht dabei aus dem BCM2711, ein SoC³ von Broadcom. Ein SoC kombiniert den Prozessor und verschiedene Peripherieeinheiten, wie den Speicher und die Grafikeinheit, schaltungstechnisch zu einem einzigen Chip, welcher über die Leiterplatte mit den anderen Komponenten des Computers verbunden ist. Die Spannungsversorgung erfolgt über einen USB-C Adapter.[5]

Prozessor Als Prozessor kommt der ARM Cortex-A72 zum Einsatz, welcher in der gewählten Konfiguration mit vier Prozessorkernen arbeitet, die jeweils mit 1.5 GHz getaktet sind. Der ARM Cortex-A72 implementiert eine ARM-v8-Architektur, die mit einer Verarbeitungsbreite von 64 Bit arbeitet. Der Prozessor unterstützt Gleitkomma-Operationen (Floating-Points-Operations) sowie AES⁴-Verschlüsselung in Hardware. Es existieren zwei separate Cache-Speicher, ein Cache für Befehle (Instruction Cache) und ein Cache für Daten (Data Cache), wie auch bei einer x86-Architektur. Über die konfigurierbare AMBA⁵ werden On-Chip-Verbindungen und ein Management der einzelnen Funktionsblöcke in dem SoC realisiert. Die SCU⁶ verbindet den ARM Cortex-A72 mit dem Speichersystem über insgesamt drei AXI⁷-Schnittstellen. Der ARM Cortex-A72 enthält außerdem eine Out-of-Order Execution Pipeline, die es ermöglicht, Befehle in einer anderen Reihenfolge auszuführen, als vom Programmcode vorgegeben. So wird eine bessere Ausnutzung der Befehlspipeline erreicht.[5][19] Diese Architektur kann in Abbildung 2.5 nachvollzogen werden.

¹Central Processing Unit

²Random Access Memory

³System on a Chip

⁴Advanced Encryption Standard

⁵Advanced Microcontroller Bus Architecture

⁶Snoop Control Unit

⁷Advanced eXtensible Interface

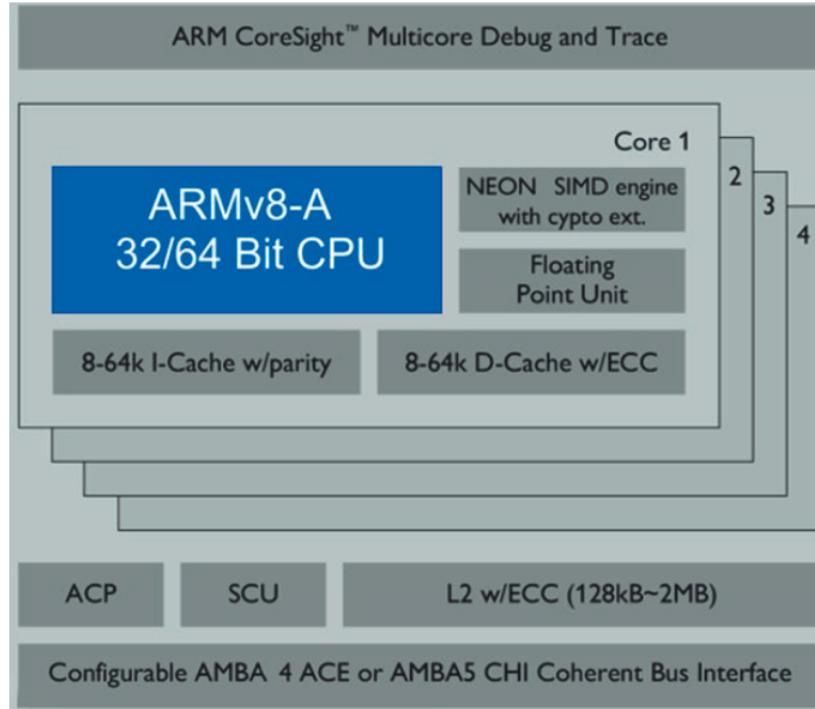


Abbildung 2.5: Blockschaltbild Cortex-A72 [5]

Speicher Der Raspberry Pi 4 ist mit einem DDR4-SDRAM-Speicher vom Typ 9FD77-D9WHV ausgestattet. Der 9FD77-D9WHV besitzt eine Größe von 4 GB und dient dem Prozessor als Arbeitsspeicher. Der Prozessor teilt sich den Speicherchip mit der Grafikeinheit, welche den DDR4-SDRAM als Graphic Memory verwendet. Die Aufteilung des Speicherchips zwischen Prozessor und Grafikeinheit kann vom Benutzer konfiguriert werden und wurde nach den Anforderungen des Systems angepasst.

Als Festwertspeicher des Systems wird eine Micro SD-Karte verwendet. Hier findet sich das Betriebssystem inklusive des Programmcodes, welcher die Steuerung des Farbsortier-Demonstrators umsetzt. Die Micro SD-Karte kann bei ausgeschaltetem Raspberry Pi gewechselt werden, so dass unterschiedlich konfigurierte Betriebssysteme schnell und unkompliziert getestet werden können.[5]

Schnittstellen Es steht eine Vielzahl verschiedener Schnittstellen zur Verfügung, über die der Raspberry Pi 4B Daten senden und empfangen kann. Für die Entwicklung des Farbsortier-Demonstrators sind besonders das CSI⁸ und die 40-polige GPIO⁹-Steckleiste von zentraler Bedeutung.

Über das CSI werden die Bilddaten der Kamera auf den Raspberry Pi übertragen. Die GPIO-Steckleiste wird hingegen für die Kommunikation des Raspberry Pi's mit externer Hardware, wie etwa den Motoren oder Lichtschranken verwendet.

⁸Camera Serial Interface

⁹General Purpose Input Output

Im weiteren Verlauf der Arbeit waren ebenfalls die integrierten Ethernet-Controller und WLAN-Adapter, die vier USB¹⁰-Buchsen sowie die zwei Micro-HDMI¹¹-Buchsen von Relevanz. Über den Ethernet-Controller (vom Typ LAN9514) und den WLAN-Adapter (vom Typ CYW43455) kann sich der Raspberry Pi mit dem Internet verbinden, wodurch eine Steuerung per Fernzugriff ermöglicht wird. Die USB-Schnittstelle, welche ebenfalls vom LAN9514 verwaltet wird, ermöglicht es dem Benutzer Maus, Tastatur und externe Speicher mit dem Gerät zu verbinden. Über die zwei Micro-HDMI-Buchsen können bis zu zwei Bildschirme verwendet werden. Die Spannungsversorgung erfolgt mithilfe eines 5 V-Steckernetzteils, das mit der USB-C-Buchse des Raspberry Pi's verbunden wird. Die Anordnung der Bestandteile des Raspberry Pi's kann Abb. 2.6 entnommen werden.

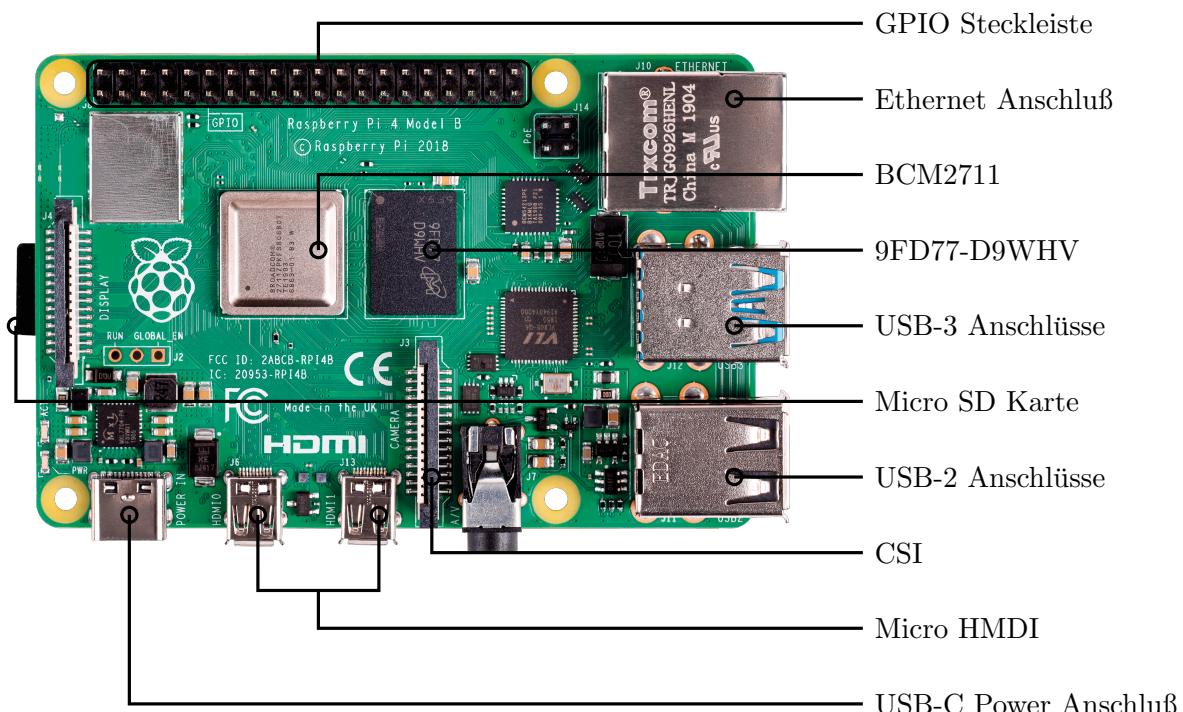


Abbildung 2.6: Hardware-Komponenten des Raspberry Pi's [8]

2.2.2 RaspiCam

Die Aufnahme der Bilddaten erfolgt über die RaspiCam, eine 8 Megapixel Kamera der Raspberry Pi Foundation. Die RaspiCam nutzt den Sensor IMX219 von Sony, welcher über eine maximale Auflösung von 3280 x 2464 Pixel verfügt. Die Übertragung der Bilddaten erfolgt seriell über das CSI. Genutzt wird ein I2C-Bus-Interface, bei dem der Kanal SCL0 als Takt- und SDA0 als Datenleitung verwendet wird.

Mittels des Programms `raspistill` können Bilder von der RaspiCam aufgenommen und im Speicher des Raspberry Pi's abgelegt werden. Die Aufnahme der Bilddaten

¹⁰Universal Serial Bus

¹¹High Definition Multimedia Interface

kann dabei über Kommandozeilen-Parameter vielfältig konfiguriert werden. So kann beispielweise mit dem Argument `-tl` der *timelapse*-Modus aktiviert werden, der eine periodische Bildaufnahme ermöglicht. Dabei beträgt der minimale Abstand zwischen zwei Bildern laut Software Spezifikation 30 ms.[9]

Da die spezifische Anpassung der Bildaufnahme von verschiedenen Rahmenbedingungen, wie etwa der Geschwindigkeit des Transportbandes, abhängig ist, wird diese unter Abschnitt 4.2.3.1 detaillierter beschrieben.

2.2.3 General Purpose Input Output

Die 40-polige Pinleiste, des Raspberry Pi's wird als GPIO zusammengefasst. GPIO bietet die Möglichkeit externe elektrische Schaltkreise mit dem Raspberry Pi zu verbinden und zu steuern. So können Signale an elektrische Bauteile gesendet und empfangen werden - je nach Konfiguration des entsprechenden GPIO-Pins. Im Rahmen dieser Arbeit stellt GPIO die Verbindungsstelle zwischen der Software und allen elektrischen Bauteilen dar.

Auf der 40-poligen GPIO-Steckleiste werden jeweils zwei Pins für eine 5 V- bzw. 3.3 V-Spannungsversorgung bereitgestellt. Acht weitere Pins sind mit Masse verbunden. Die übrigen 28 Pins sind frei als Eingang oder Ausgang konfigurierbar oder können für alternative Funktionen, wie etwa der Kommunikation mittels SPI¹² oder I2C¹³, genutzt werden.

¹²Serial Peripheral Interface

¹³Inter-Integrated Circuit

2.3 System und UNIX-Prozesse

Die Raspberry Pi Foundation stellt ein eigenes Betriebssystem für den Raspberry Pi zur Verfügung, das Raspberry Pi OS. Das Raspberry Pi OS ist ein Linux Debian-basiertes Betriebssystem und eng an die Systemvoraussetzungen der Raspberry Pi Linie angepasst. Zentrale Bestandteile von UNIX-Systemen können so auch auf dem Raspberry Pi OS genutzt werden. Als konkretes Beispiel muss an dieser Stelle die Welt der UNIX-Prozesse genannt werden. Im Verlauf der Arbeit hat sich herausgestellt, dass die Wahl der Prozessarchitektur maßgeblich die Erfolgsquote des Farbsortier-Demonstrators beeinflusst. Aus diesem Grund sollen wesentliche Kernaspekte der UNIX-Prozesse im Folgenden genauer beleuchtet werden.

2.3.1 Allgemeines zu UNIX Prozessen

Ein Prozess ist die Abstraktion eines Programms, welches gestartet wurde und sich in der Ausführung befindet. Die Verarbeitung eines Prozesses verläuft sequenziell.^[7] Prozesse werden vom Betriebssystem verwaltet. Dabei werden zu einem Prozess mehrere Informationen gespeichert, die die Arbeit mit dem Prozess erst ermöglichen. Dazu zählen unter anderem:

- PID - ganzzahliger Wert, Bezeichner des Prozesses
- PPID - wie PID, nur vom Elternprozess
- Dateideskriptoren - Tabelle, enthält Bezeichner verwendeter Dateien
- Programmreich - Programmcode
- Datenbereich - Globale Variablen
- Stack - lokale Variablen
- Priorität - Relevanz des Prozesses; für Scheduling

So kann ein Prozess durch seine PID eindeutig identifiziert und beispielsweise über ein Signal angesprochen werden. Die PPID gibt Auskunft über den Prozess, aus dem der aktuelle Prozess gestartet wurde. Sollten Dateien innerhalb eines Prozesses geöffnet worden sein, wird deren Zuordnung über Dateideskriptoren gespeichert.

Laut dem Prozessmodell [20] befindet sich ein Prozess immer in einem der folgenden Zustände: *nicht existent, erzeugt, bereit, aktiv, blockiert* oder *beendet*. Die Transition von welchem Zustand welcher Folgezustand erreicht werden kann, ist dabei konkret vorgegeben (siehe Abb. 2.7). Wichtig zu bemerken ist, dass nur **aktive** Prozesse Ressourcen auf dem Prozessor verbrauchen. Eine detaillierte Erläuterung der einzelnen Prozesszustände kann in [Tanenbaum 2009] nachgelesen werden.^[25]

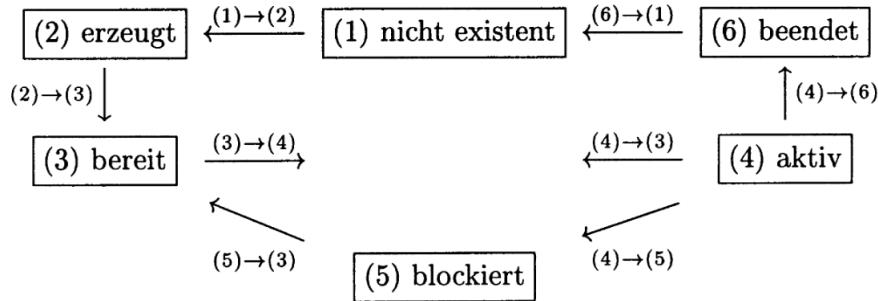


Abbildung 2.7: Prozesszustände nach UNIX Prozessmodell [20]

2.3.2 Verteilte Prozesse

UNIX-Prozesse haben die Möglichkeit mittels des Befehls `fork()` eine exakte Kopie des aufrufenden Prozesses zu erstellen. Diese Kopie ist ein neuer Prozess mit eigener PID, genannt Kind-Prozess. Der Prozess, welcher den neuen Prozess erstellt, wird Eltern-Prozess genannt. Bei der Erstellung erbt der Kind-Prozess sämtliche Informationen des Eltern-Prozesses, ausgenommen der PID und PPID. Die PID wird neu generiert, wohingegen die PPID der PID des erzeugenden Prozesses gleichgesetzt wird. Standardgemäß ist der Kind-Prozess insofern vom Eltern-Prozess abhängig, als dass sich der Kind-Prozess bei Beendigung des Eltern-Prozesses ebenfalls terminiert. Beendet sich der Kind-Prozess vor dem Eltern-Prozess, wird dieser durch das Signal `SIGCHLD` über dessen Terminierung informiert. Ein Elternprozess kann mittels des Befehls `waitpid()` auf die Terminierung eines Kindprozesses warten und sich so von der CPU nehmen. Solche verteilten Prozesse bieten die Möglichkeit, komplexe Probleme aufzuteilen, um diese dann einzeln und effizienter zu bearbeiten. Ein Konzept dieser Modularisierung von Problemen nennt sich die *Instruktionsstrom-Zerlegung* (MIMD¹⁴). Bei der Instruktionsstrom-Zerlegeung werden mehrere Prozesse mit kleineren Teilaufgaben eines größeren Problems betraut. Die benötigten Ressourcen des Programms können gemäß der UNIX-Prozess-Zustände flexibel zugewiesen werden. Prozesse, deren Teil des Programms in einem Zeitraum nicht benötigt wird, können blockiert werden und verbrauchen so keine Ressourcen des Prozessors.[7]

2.3.3 Interprozess-Kommunikation

Das Zusammenspiel mehrerer Prozesse besteht besonders aus deren Kommunikation und Synchronisation. Wird ein größeres Problem auf verschiedene Prozesse verteilt, ist es für diese Prozesse meist notwendig, Informationen untereinander zu verschicken und zu empfangen. Dabei stehen die Prozesse in einer Sender-Empfänger-Beziehung, an welcher [1..n] Sender und [1..m] Empfänger teilnehmen können.

Sender und Empfänger können direkt oder indirekt miteinander kommunizieren. Während der direkten Kommunikation blockiert der sendende Prozess solange, bis der Empfänger bereit ist eine Nachricht zu empfangen. Die Nachricht wird nicht gepuffert.

¹⁴Multiple Instruction Multiple Data

Dieses Verhalten macht die direkte Kommunikation unflexibler als die indirekte Kommunikation.

Bei dieser teilen sich Sender und Empfänger eine Art "Postfach", welches häufig als FIFO¹⁵ realisiert wird. Der Sender legt seine Nachricht in das Postfach, ohne von dem Zustand des empfangenden Prozesses abhängig zu sein. Der Empfänger liest anschließend die Nachricht aus dem "Postfach".

Ein Beispiel für die indirekte Kommunikation ist die sogenannte Nachrichtenwarteschlange oder auch *Message Queue*. Dies ist eine geordnete Nachrichtenliste im Bereich des Kernels. Eine Message Queue wird durch einen Schlüssel identifiziert. Jeder Prozess, dem der Schlüssel einer Message Queue bekannt ist, kann mit dem Befehl `msgsnd()` in diese schreiben und dem Befehl `msgrcv()` aus ihr lesen.

Wichtige Nachrichten können zudem über ein Prioritätsfeld beim Schreiben priorisiert werden. Auf diese Weise können Empfänger hoch priorisierte Nachrichten vor Nachrichten mit niedrigerer Priorität aus der Message Queue lesen. Haben alle Nachrichten die selbe Priorität, funktioniert die Message Queue gleich einem FIFO.[7]

¹⁵First In First Out

2.4 Aktorik und Sensorik

Die Aktuatoren und Sensoren sind für den Erfolg des Farbsortier-Demonstrators unabdingbar. Durch die Sensorik erhält die Software die notwendigen Eingangsdaten über die Testobjekte. Die Aktorik hingegen realisiert den Transport und die Sortierung der Testobjekte.

2.4.1 Schrittmotor NEMA 17

Der NEMA 17 Schrittmotor zählt zu den 2-Phasen-Schrittmotoren mit bipolarem Aufbau. Die Antriebsachse des Schrittmotors wird durch einen Permanentmagneten rotiert, welcher von vier Gleichstromwicklungen umgeben ist. Die Gleichstromwicklungen werden mit unterschiedlicher Polarität erregt, so dass ein sich drehendes Magnetfeld entsteht. Der Permanentmagnet im Zentrum des Motors richtet sich nach dem Magnetfeld aus und rotiert.

Ein großer Vorteil von Schrittmotoren ist, dass der Drehwinkel genau eingestellt werden kann. Der NEMA 17 Schrittmotor dreht sich pro Motorschritt um genau 1.8° . Dies erlaubt eine zielgerichtete Positionierung der angeschlossenen Mechanik. Da der Drehwinkel bekannt ist, ist auch keine zusätzliche Sensorik notwendig, um die aktuelle Position des Schrittmotors zu ermitteln.

Der NEMA 17 Schrittmotor wird mit einer Spannung von 12 V betrieben. Die Ansteuerung erfolgt durch einen Schrittmotortreiber.[29][28]

2.4.2 Motortreiber A4988

Da der Schrittmotor nicht direkt angesteuert werden kann, befindet sich zwischen Entwicklungsplattform und Schrittmotor der A4988 Motortreiber. Dieser realisiert die Abfolge an Signalen, die an den Schrittmotor gesendet werden müssen, um ein sich rotierendes Magnetfeld zu erzeugen. Er fungiert dabei als eine Art Schnittstelle. Über seine Eingangspins lässt sich der Motor aktivieren bzw. deaktivieren, die Drehrichtung einstellen sowie ein Motorschritt machen. Zusätzlich ist es möglich, die Granularität der Motorschritte auf bis zu einem sechszehntel Schritt herunterzuskalieren. Ebenso nimmt der A4988 Motortreiber die 12 V-Versorgungsspannung für den Schrittmotor auf. Ein vollständiger Schaltplan kann Abb. 2.8 entnommen werden.[1]

2.4.3 Gleichstrommotor Modelcraft RB350100-0A101R

Gleichstrommotoren werden, wie dem Namen entnehmbar, mit Gleichstrom betrieben. Es existieren viele verschiedene Möglichkeiten einen Gleichstrommotor zu konzipieren, weshalb sich in dieser Darstellung auf die Bauweise des Modelcraft RB350100-0A101R bezogen wird. Der vorliegende Gleichstrommotor besteht aus einem unbeweglichen Ständer, der einen sich rotierenden Anker umschließt. Der Ständer bildet ein konstantes Magnetfeld, bei welchem die eine Hälfte als Nordpol und die andere Hälfte als Südpol wirkt. Wird der Anker mit Strom durchflossen, wirkt die Lorentz-Kraft und der Anker

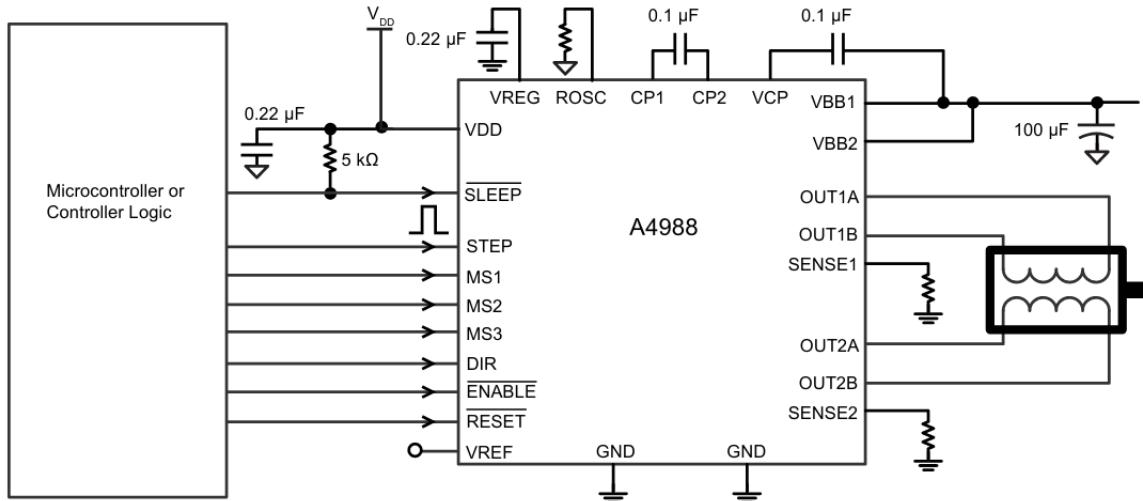


Abbildung 2.8: Schaltplan A4988 Motortreiber [1]

samt der Antriebswelle des Motors rotiert. Der Modelcraft RB350100-0A101R Gleichstrommotor wird mit einer Spannung von 12 V betrieben. Die Ansteuerung erfolgt über ein 5 V-Relais.[28][21]

2.4.4 Leuchtdiode

Leuchtdioden sind Halbleiterbauelemente, die zur Generierung von Lichtsignalen oder der Ausleuchtung von Bereichen verwendet werden können. Sie sind eine spezielle Form von Dioden, die in Durchlassrichtung betrieben werden. Bei Zuführen von elektrischer Spannung wird den Elektronen im Halbleiter Energie zugeführt. Infolge dessen bewegen sich die Elektronen vom Valenzband des Halbleiters in dessen Leitungsband. Während dieses Prozesses wird Energie frei, die in Form von Photonen, also Licht, abgegeben wird. Die Wahl des Halbleiters entscheidet über die Wellenlänge des Lichts und dementsprechend über die abgegebene Farbe der Leuchtdiode.[28]

Leuchtdioden weisen verschiedene Vorteile gegenüber konventionellen Lichtquellen auf, die auch für die hier beschriebene Arbeit relevant sind. Leuchtdioden haben eine lange Lebensdauer von bis zu 100.000 Stunden. Sie verkraften Stöße und Vibration, was in Anbetracht der eingesetzten Motoren durchaus entscheidend sein kann. Weiterhin sind sie energieeffizient, da Licht nur im sichtbaren Bereich ausgestrahlt wird und keine Energie im ultravioletten oder infraroten Bereich verloren geht. Als größter Nachteil ist die geringe Lichtstärke einer einzelnen Leuchtdiode zu nennen.[22]

Dieses Problem wird jedoch durch neuartige COB¹⁶-Leuchtdioden gelöst, die eine Vielzahl einzelner Leuchtdioden zu einem einzigen Bauelement vereinen. So kann mit minimalem Beschaltungsaufwand ein ausreichend hoher Lichtstrom erreicht werden. Ein integrierter Widerstand ermöglicht den direkten Betrieb einer COB-Leuchtdiode ohne weitere elektrische Bauteile. Folglich wurde sich für eine COB-Leuchtdiode der Firma LEDmaxx mit der Bezeichnung 27GY251 entschieden. Ihr Lichtstrom von 200 lm

¹⁶Chip-On-Board

reicht aus, um die erforderlichen Bereiche hell genug auszuleuchten. Die Farbtemperatur von 2700 K sowie der Farbwiedergabeindex von > 80 versprechen außerdem eine farbton-getreue Beleuchtung der Testobjekte.[18]

2.4.5 Relais Modul

Relais Module werden verwendet, um in elektrischen Schaltkreisen elektrische Schalter zu realisieren. Dabei ist das Relais Modul ein elektrisches Bauteil, welches von einem Steuerstromkreis mit niedriger Spannung einen Laststromkreis mit höherer Spannung steuert. Steuerstromkreis und Laststromkreis sind dabei galvanisch voneinander getrennt. Das bedeutet, dass beide Stromkreise keine elektrische Verbindung zwischeneinander haben. Die Schaltung eines Relais mittels Steuer- und Laststromkreis kann in Abbildung Abb. 2.9 nachvollzogen werden.

Die Funktionsweise eines Relais Moduls beruht dabei auf dem physikalischen Prinzips des Elektromagnetismus. Das magnetische Feld entsteht dabei durch einen fließenden Strom. Je größer dieser Strom ist, desto größer ist auch die Kraft, die das Magnetfeld ausübt.

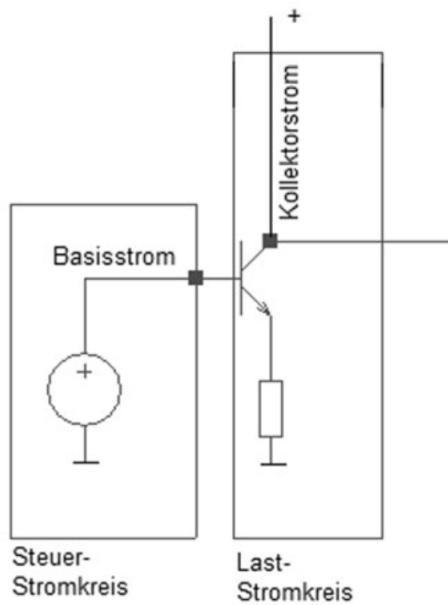


Abbildung 2.9: Steuer- und Laststromkreis einer Relais Schaltung [17]

Im Relais Modul sitzt eine Spule, welche einen ferromagnetischen Kern umwickelt. Wird über den Steuerstromkreis an der Spule eine Spannung angelegt, entsteht im Kern ein Magnetfeld. Ein mechanischer Schalter neben dem Kern wird von diesem angezogen und schließt so den Arbeitskontakt im Laststromkreis. Auf diese Weise kann der Laststromkreis unterbrochen oder geschlossen werden.[17]

Das verwendete Relais Modul ist vom Typ KF-301 der Firma AZ-Delivery. Es besteht aus zwei Leuchtdioden, drei Widerständen, einem NPM-Transistor einer Gleichrichterdiode und einem Relais. Im Gleichstrombetrieb kann es bis zu 30 V bei 5 A verarbeiten.

Die Ansteuerung des Steuerstromkreises erfolgt über die drei Pins VCC, GND und IN. VCC und GND realisieren die Stromversorgung, während über den Pin IN das Relais von der Entwicklungsplattform gesteuert werden kann.[3]

2.4.6 Lichtschranke

Lichtschranken können dazu verwendet werden, ein sich bewegendes Objekt zu erkennen und anschließend ein elektrisches Signal zu senden. Sie gehören zu der Gruppe optoelektronischer Bauelemente. Üblicherweise arbeiten Lichtschranken mit einem Lichtstrahl, dessen Unterbrechung ein bewegtes Objekt signalisiert. Es gibt verschiedene Arten von Lichtschranken, die jeweils unterschiedliche Vor- und Nachteile aufweisen.

So existieren beispielweise Einweg-Lichtschranken, bei denen Sender und Empfänger des Lichtstrahls räumlich voneinander getrennt sind. Sender und Empfänger bilden dabei sogenannte "Lichtkeulen", durch welche ein aktiver Überwachungsbereich geschaffen wird (siehe Abbildung Abb. 2.10). Diese Art von Lichtschranken ermöglicht die Detektion von besonders kleinen Objekten über größere Distanzen. Dabei steigt jedoch der Implementierungsaufwand, da Sender und Empfänger separat voneinander angesteuert werden müssen. Außerdem liegen die Einweg-Lichtschranken im Preissegment von $\approx 100\text{€}$ aufwärts, was sie verhältnismäßig teuer macht.[4]

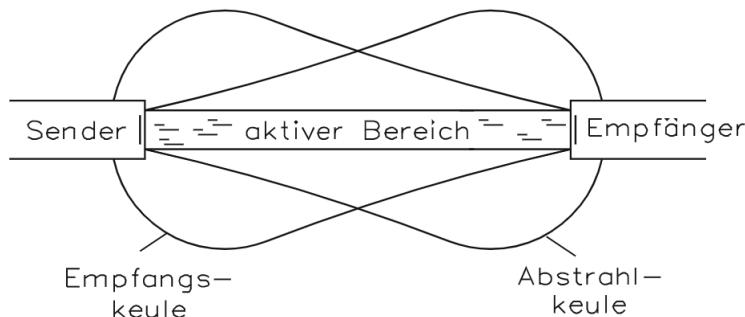


Abbildung 2.10: Funktionsweise Einweg-Lichtschranke [4]

Eine weitere Art der Lichtschranken sind Reflexions- bzw. Reflex-Lichtschranken. Hier finden sich Sender und Empfänger zusammen auf einem Bauelement. Der Sender sendet einen Lichtstrahl, welcher von einem Reflektor auf den Empfänger zurückgestrahlt wird. Ein Grundproblem von Reflex-Lichtschranken ist die unzuverlässige Detektion von reflektierenden Objekten. Der gesendete Lichtstrahl wird unter Umständen vom Objekt auf den Empfänger zurückgeworfen. Es kann deshalb passieren, dass das Objekt den Lichtstrahl nicht konstant unterbricht. Dieses Problem lässt sich beheben, indem vor Sender und Empfänger ein Polarisationsfilter installiert wird. Jedoch unterscheiden sich die Kosten für eine Reflex-Lichtschranke mit Polarisationsfilter ($\approx 80 - 100\text{€}$) erheblich von den Kosten für eine Reflex-Lichtschranke ohne Polarisationsfilter ($\approx 2\text{€}$).[4]

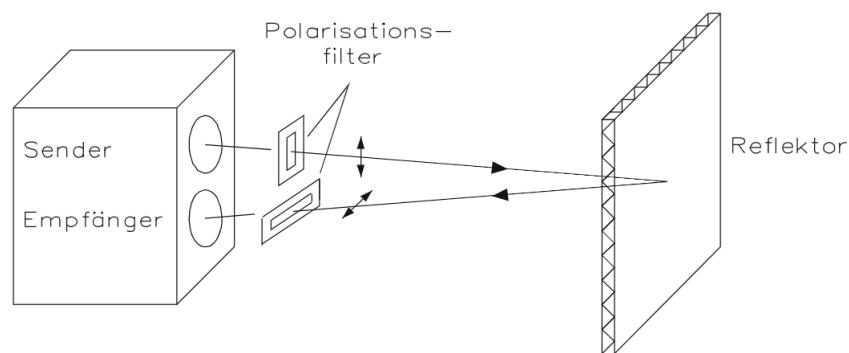


Abbildung 2.11: Funktionsweise Reflex-Lichtschranke [4]

2.5 Echtzeit

Echtzeitsysteme unterscheiden sich im Wesentlichen von Nicht-Echtzeitsystemen durch die Bedingung, dass neben der logischen Korrektheit einer Berechnung ebenfalls die zeitliche Korrektheit gegeben sein muss. Dieses Problem wird unter dem Begriff der *Rechtzeitigkeit* gefasst. Echtzeitsysteme müssen also eine zeitliche Vorhersehbarkeit besitzen, um das Einhalten von sogenannten *Zeitbedingungen* zu garantieren.

Die Zeitbedingungen legen fest, zu welchem Zeitpunkt ein Ereignis als korrekt angesehen wird. Dabei wird zwischen vier verschiedenen Fällen unterschieden, wie in Tabelle 2.1 und Abb. 2.12 ersichtlich ist.[32]

Zeitbedingung	Parameter	Erläuterung
Genauer Zeitpunkt (a)	t	Das Ereignis muss genau zum Zeitpunkt t eintreten
Spätester Zeitpunkt (b)	t_{max}	Das Ereignis darf früher, aber nicht später als Zeitpunkt t_{max} eintreten
Frühester Zeitpunkt (c)	t_{min}	Das Ereignis darf später, aber nicht früher als Zeitpunkt t_{min} eintreten
Zeitintervall (d)	t_{min}, t_{max}	Das Ereignis muss zwischen den Zeitpunkten t_{min} und t_{max} eintreten

Tabelle 2.1: Zeitbedingungen in der Echtzeit [32]

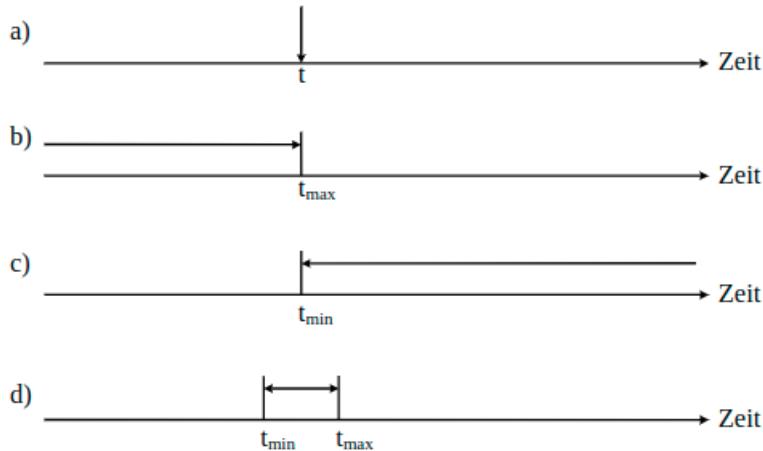


Abbildung 2.12: Zeitbedingungen

Die Zeitbedingungen können tiefergehend spezifiziert werden. So wird zwischen *periodischen Zeitbedingungen* (Zeitbedingung tritt in regelmäßigen Abständen wiederholt auf) und *aperiodischen Zeitbedingungen* (Ereignis löst Zeitbedingung aus) unterschieden.

Weiterhin können Zeitbedingungen *absolut* zu einer Uhrzeit definiert sein oder sich *relativ* auf einen anderen Auslöser, bspw. eine andere Zeitbedingung, beziehen.

Betreffend der Einhaltung von Zeitbedingungen unterscheidet man bei Echtzeitsystemen zwischen drei verschiedenen Kategorien: harten Echtzeitbedingungen, festen Echtzeitbedingungen und weichen Echtzeitbedingungen.[32] Diese können in Tabelle 2.2 nachvollzogen werden.

Echtzeitbedingung	Erläuterung
Hart	Die Zeitbedingungen müssen auf jeden Fall eingehalten werden, sonst droht Schaden.
Fest	Die Zeitbedingungen müssen eingehalten werden, sonst verfällt das Ergebnis.
Weich	Die Zeitbedingungen sind Richtwerte und können bis zu einem bestimmten Punkt überschritten werden.

Tabelle 2.2: Kategorien der Echtzeitbedingungen

Echtzeitsysteme werden nach Definition auf Mikrorechnersystemen implementiert und von sogenannten Echtzeitbetriebssystemen gesteuert. Diese haben die Eigenschaft, dass die Anzahl der Prozesse auf ein benötigtes Minimum beschränkt ist und diese vollständig vom Benutzer gesteuert werden können.[32]

Auf diese Weise sind Verhalten und Laufzeit der Prozesse exakt vorhersehbar. Da das verwendete Betriebssystem (Raspberry Pi OS) kein Echtzeitbetriebssystem ist, entfallen tiefergehende Charakteristika üblicher Echtzeitsysteme, wie etwa der Echtzeit-Scheduler bzw. das Echtzeit-Scheduling.

Nichtsdestotrotz kann ein System auf einem Nicht-Echtzeitbetriebssystem dennoch echtzeitfähig sein, sofern es die gleichen Anforderungen eines Echtzeitsystems hat und diese auch erfüllt.

Bezogen auf diese Arbeit bedeutet das, dass sich das System entsprechend der notwendigen Zeitbedingungen verhalten muss. Folglich müssen die Zeitbedingungen der einzelnen Teilsysteme sowie des Gesamtsystems bestimmt werden. Um das Verhalten der Teilsysteme und des Gesamtsystems gegenüber den Zeitbedingungen bewerten zu können, ist es absolut notwendig die Verarbeitungszeiten der einzelnen Teilsysteme empirisch zu erheben. Diese können anschließend mit den ermittelten Zeitbedingungen verglichen werden. Auf diese Weise kann überprüft werden, ob die Echtzeitfähigkeit des Systems robust gewährleistet wird.

3 Anforderungsanalyse

Aufgrund der Komplexität der Problemstellung erscheint es sinnvoll, das Gesamtsystem in einzelne, möglichst kleine Teilprobleme zu zerlegen. Diese können separat behandelt und deren Lösung zu komplexeren Teilsystemen zusammengesetzt werden. Die funktionierenden Teilsysteme werden abschließend miteinander verbunden und bilden die Lösung des Gesamtsystems.

Um die Teilprobleme möglichst genau zu definieren, werden alle Anforderungen zusammengetragen, die das Gesamtsystem erfüllen muss. Dabei wird versucht, die Anforderungen möglichst atomar zu formulieren. Die Anforderungen sind aus Gründen der Übersicht nummeriert.

Notwendige Anforderungen

1. Das System muss eine Komponente beinhalten, welche in der Lage ist, **Objekte bei gleichbleibender Geschwindigkeit zu bewegen** (nachfolgend *Transportband*).
2. Das System muss einen **Bereich** beinhalten, durch den Objekte hindurch bewegt werden und der **von einer Kamera überwacht** wird (nachfolgend *Detektionsbereich*).
3. Das System muss in der Lage sein, mithilfe der Kamera **Objekte zu detektieren**, die durch den Detektionsbereich bewegt werden.
4. Das System muss in der Lage sein, detektierte Objekte **einer** von sechs vordefinierten **Farben zuzuordnen** (welche sich teilweise sehr ähnlich sind).
5. Es muss eine mechanische Komponente geben, die **Objekte** in unterschiedliche Fächer **sortiert** (nachfolgend *mechanische Sortierung*).
6. Die mechanische Sortierung muss so präzise ansteuerbar sein, dass mindestens sechs **verschiedene Positionen verlässlich eingenommen** werden können.
7. Die **mechanische Sortierung muss robust bewegt** werden können (präzises Vor- und Zurückspringen in den vordefinierten Positionen, auch nach dem Durchlauf mehrerer Objekte).
8. **Umgebungslicht** im Detektionsbereich muss **unabhängig von äußeren Einflüssen** sein, um einheitliche Bedingungen zu schaffen.

9. Das System muss **mit Störungen** im Bild **umgehen** können (Rauschen, Reflexion, Schatten, Textur des Laufbands).
10. Das System muss genügend Bilder innerhalb einer zu bestimmenden Zeitspanne verarbeiten können, um ein sich durch den Detektionsbereich **bewegendes Objekt sicher zu detektieren**.
11. Das System muss in der Lage sein, den **gesamten Ablauf in Echtzeit durchzuführen**.
12. Das System muss in der Lage sein, den gesamten Ablauf **mehrfach hintereinander** durchzuführen, **ohne** dabei einen **Rückgang der Trefferquote** aufzuweisen.

4 Konstruktion Versuchsaufbau

Zur Sortierung von Objekten wird eine Vorrichtung benötigt, die Software und Hardware miteinander verbindet. Diese wird im weiteren Verlauf *Versuchsaufbau* genannt. Der Versuchsaufbau beschreibt die Kombination des konstruierten Modells mit der Aktorik und Sensorik und der Entwicklungsplattform.

Das konstruierte Modell ist nach den unter Kapitel 3 beschriebenen Anforderungen konzipiert. So finden sich mechanische Elemente, mit welchen der Transport und die Sortierung der Objekte realisiert werden können. Weiterhin ist es möglich, die Aktorik und Sensorik in dem Modell funktionsgemäß zu installieren. Beispielsweise schirmt ein Tunnel das Umgebungslicht ab, sodass Kamera und Lichtschranken mit einheitlichen Lichtverhältnissen arbeiten.

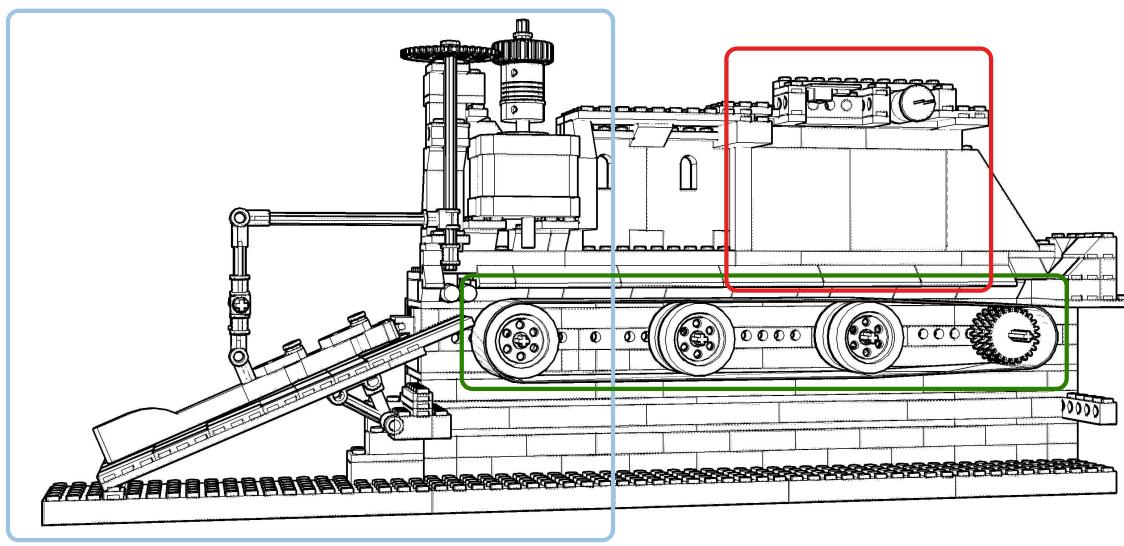


Abbildung 4.1: Verortung der Teilsysteme

Bei der Materialwahl des Modells wurde sich für Bausteine der Firma LEGO entschieden. Die Bausteine lassen sich einfach miteinander zu komplexen Gebilden kombinieren. Weiterhin kann auf Bausteine zurückgegriffen werden, die bereits spezielle Funktionen erfüllen. Als Beispiele wären hier die Lagerung von Achsen oder

die Konstruktion von Zahnrad-Getrieben zu nennen. Ein weiterer Vorteil bestand darin, dass dem Autor ein großer Bestand an Bausteinen zur Verfügung stand, sodass für die Konstruktion des Modells keine weiteren Kosten entstanden sind.

Die mechanischen, aktorischen und sensorischen Bestandteile des Versuchsaufbaus können gemäß ihrer jeweiligen Funktion zu übergeordneten Teilsystemen zusammengefasst werden. Es ergeben sich drei Teilsysteme: das *Transportband*, der *Detektionsbereich* und die *mechanische Sortierung*. Aus Abb. 4.1 ist die Positionierung dieser Teilsysteme im Versuchsaufbau ersichtlich.

Die Aktorik und Sensorik wird zu elektrischen Schaltungen mit dem Raspberry Pi verbunden. Eine Übersicht über alle elektrischen Schaltungen ist in Abb. 4.2 zu sehen.

Zur besseren Einordnung der folgenden Kapitel, zeigen die technischen Zeichnungen der Abb. 4.3 den gesamten Versuchsaufbau im Detail.

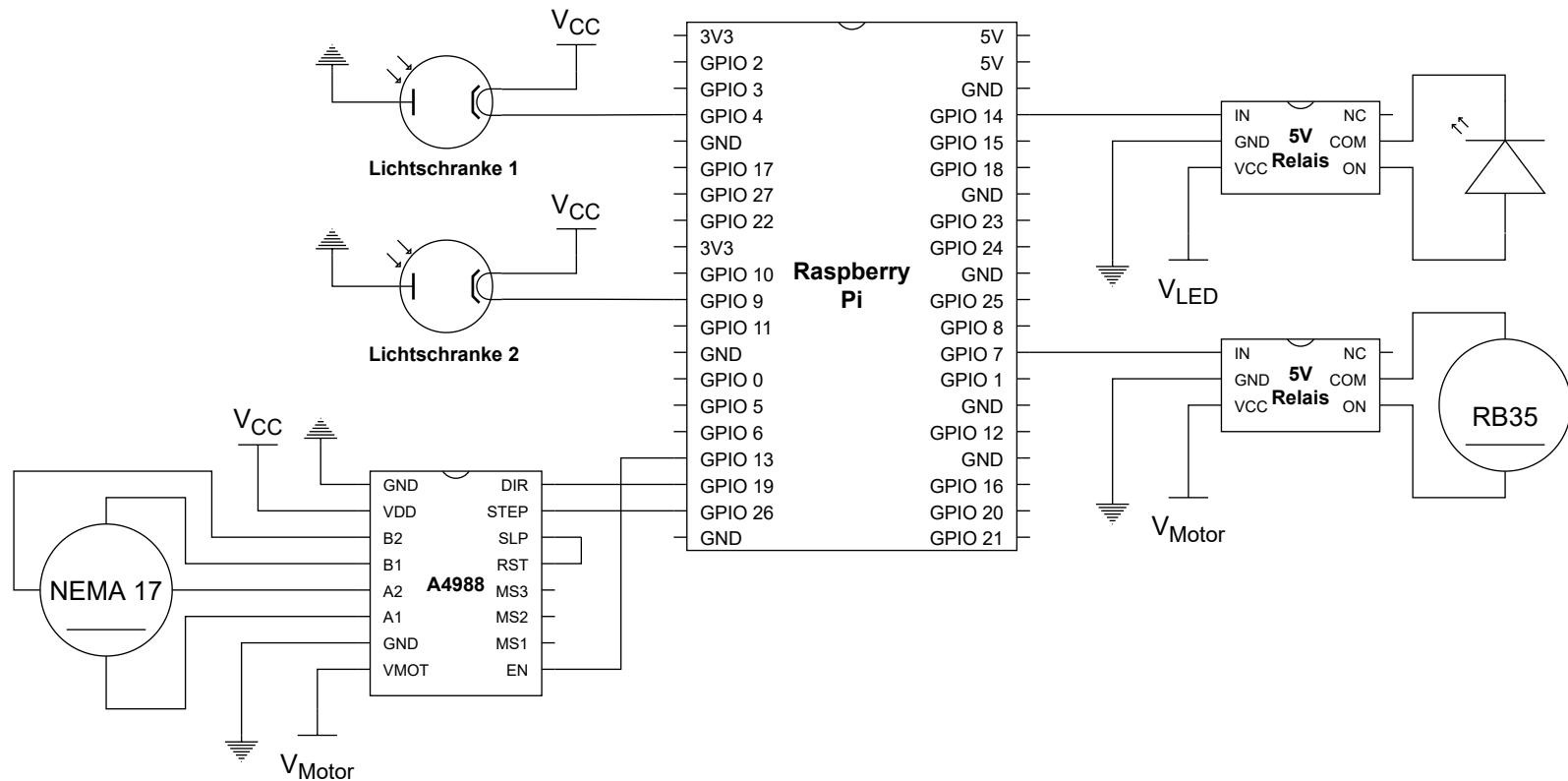
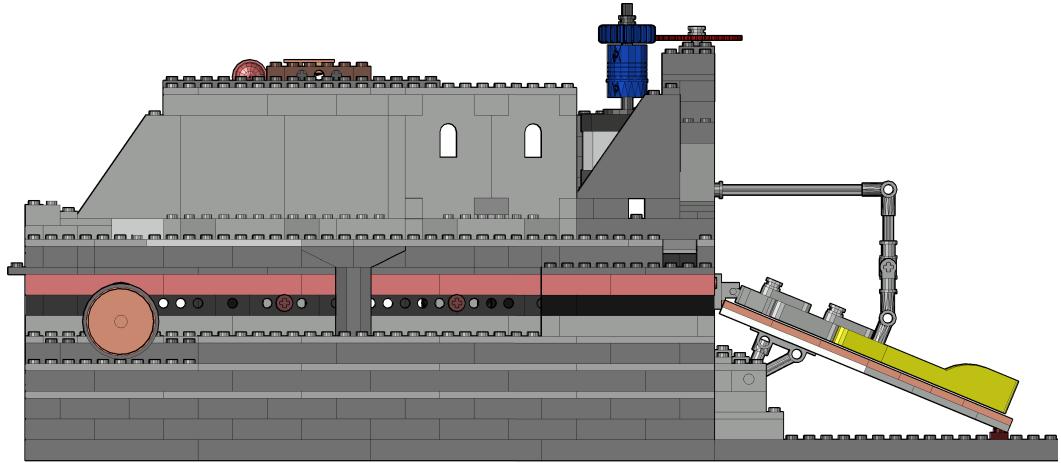
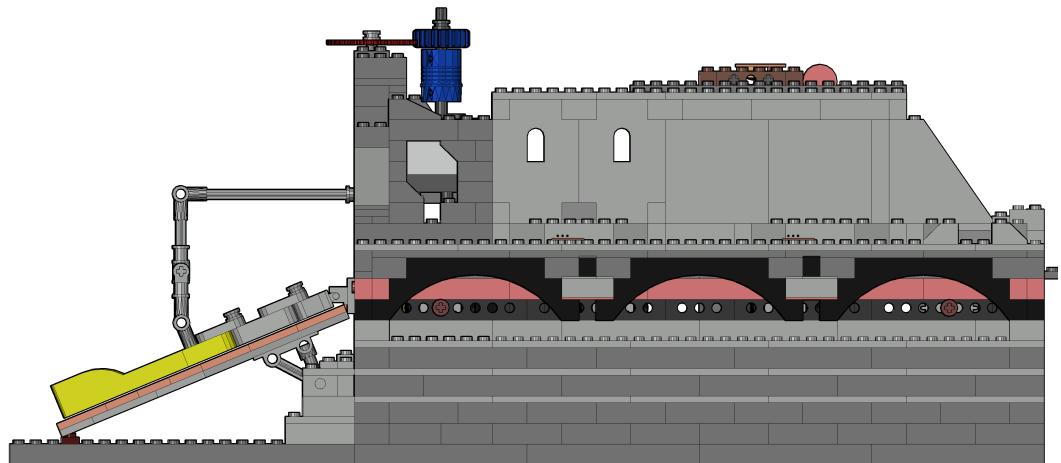


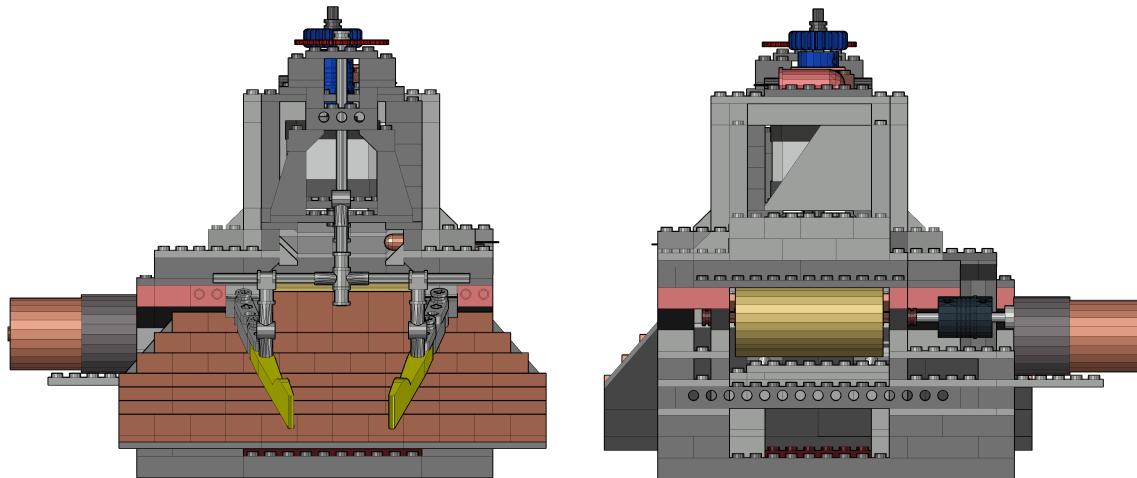
Abbildung 4.2: Schaltplan der elektrischen Komponenten



(a) Ansicht links

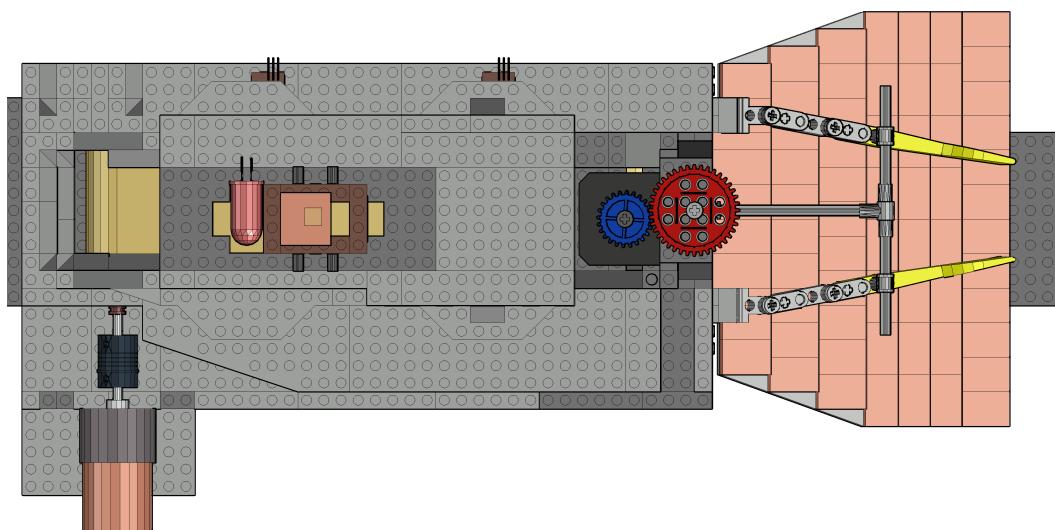


(b) Ansicht rechts



(c) Ansicht vorne

(d) Ansicht hinten



(e) Ansicht oben

Abbildung 4.3: Technische Zeichnung - Gesamter Versuchsaufbau

4.1 Transportband

Das Transportband hat die Aufgabe, die Testobjekte vom Anfang des Versuchsaufbaus bis zu dessen Ende zu transportieren. Dabei durchqueren die Testobjekte den Detektionsbereich und erreichen abschließend die mechanische Sortierung. So stellt das Transportband eine zentrale Komponente des Versuchsaufbaus dar, da durch eine Translation der Testobjekte deren Sortierung überhaupt erst möglich wird.

Für einen störungsfreien Ablauf ist es wichtig, dass die Testobjekte mit gleichbleibender Geschwindigkeit transportiert werden. Nur so können verlässliche Aussagen über die Position eines Testobjekts getätigt und somit das Verhalten des Systems sicher vorhergesagt werden.

Es existiert eine Vielzahl vorgefertigter Lösungen für Transportbänder im Internet. Diese sind meist für einen industriellen Verwendungszweck konzipiert und waren dadurch unpassend hinsichtlich der Größe des Transportbandes oder des Stückpreises. Aus diesem Grund erschien es praktikabel, ein Transportband selbst zu entwerfen, welches an die Anforderungen des Versuchsaufbaus zugeschnitten werden konnte.

4.1.1 Aufbau des Transportbandes

Das Transportband besteht aus einem länglichen Stück elastischen Materials, welches mit Montagekleber zu einem Schlauch verbunden wurde. Die Maße des Schlauchs betragen im Umfang $u_S = 58.4\text{ cm}$ bei einer Breite von $b_{Tb} = 4\text{ cm}$. Der Schlauch wird von vier Achsen getragen, von denen drei Achsen als Laufachsen dienen und eine die Antriebsachse darstellt. Die Laufachsen rotieren mit jeweils vier Rädern, die einen Radius von $r_{Lr} = 1.81\text{ cm}$ besitzen. Die Antriebsachse wird von einem Gleichstrommotor angetrieben und rotiert mit sechs Zahnrädern, welche einen ähnlichen Radius von $r_{Ar} = 1.79\text{ cm}$ aufweisen. Abb. 4.4 zeigt den Aufbau des Transportbandes im Detail.

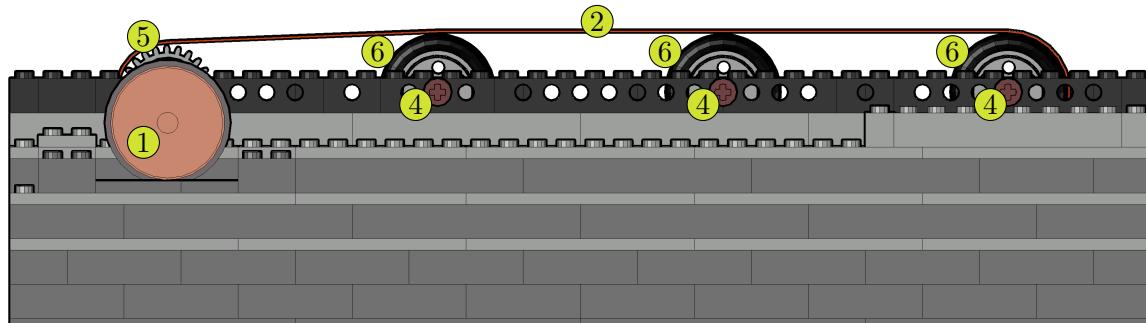
4.1.2 Ansteuerung des Transportbandes

Um das Transportband in Bewegung zu versetzen, ist ein Motor notwendig, welcher vom Raspberry Pi aus gesteuert werden kann. Dafür wurde der Gleichstrommotor RB350100-0A101R von der Firma Modelcraft ausgewählt. Dieser treibt die Antriebsachse des Transportbandes mit einer Spannung von 12 V an. Damit dieser vom Raspberry Pi gesteuert werden kann, befindet sich zwischen dem Raspberry Pi und dem Gleichstrommotor ein 5 V-Relais Modul.

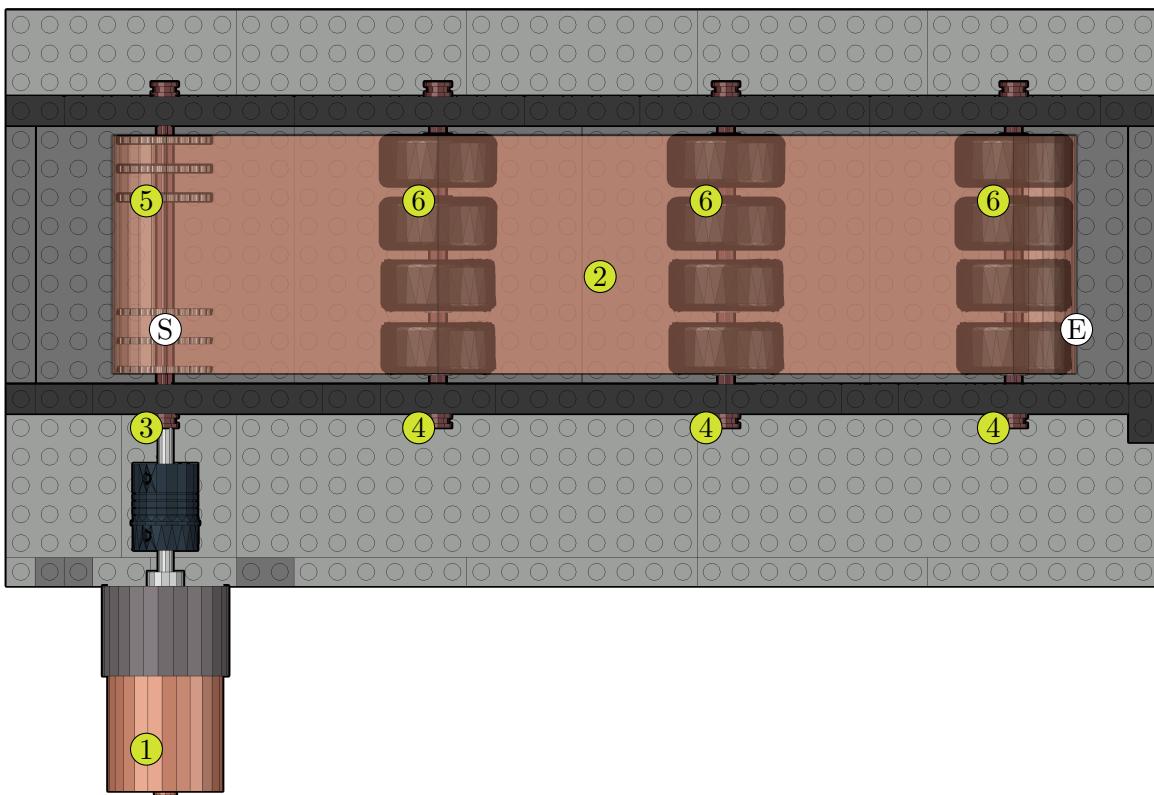
Der elektrische Schaltplan des Transportbandes ist in Abbildung Abb. 4.2 zu sehen.

4.1.3 Funktionsweise des Transportbandes

Die Geschwindigkeit, mit der sich das Transportband bewegt, entscheidet über die weitere Wahl der Programm-Parameter. Durch sie ergibt sich die Zeit, die die Testobjekte benötigten, um den Gesamtablauf der Sortierung und dessen einzelne Teilschritte zu durchlaufen. Daraus definiert sich die maximale Verarbeitungszeit



(a) Ansicht Seite links



(b) Ansicht Oben

- | | | |
|----------------------|-------------------|-------------------|
| (1) Gleichstrommotor | (2) Schlauch | (3) Antriebsachse |
| (4) Laufachse | (5) Antriebsräder | (6) Laufräder |
| (S) P_{Start} | (E) P_{Ende} | |

Abbildung 4.4: Aufbau Transportband (Tunnel ausgeblendet)

für die Aufnahme, Verarbeitung und Analyse der Bilddaten und die anschließende Steuerung der Sortierung.

Aus diesem Grund ist es wichtig, die genaue Geschwindigkeit des Transportbandes zu errechnen, um so die weiteren Parameter daraus ableiten zu können.

Die Drehzahl des Modelcraft RB350100-0A101R beträgt $n_{Gm} = 60 \frac{U}{min}$ [21]. Zur Berechnung der Geschwindigkeit v_T muss die Drehzahl n_{Gm} in die Winkelgeschwindigkeit ω_{Gm} umgerechnet werden, wie Gleichung (4.1) zeigt.

$$\omega_{Gm} = n_{Gm} \cdot \left(\frac{2\pi}{60} \right) \quad (4.1)$$

$$= 60 \frac{U}{min} \cdot \left(\frac{2\pi}{60} \right) \quad (4.2)$$

$$= 6.283 \text{ rad s}^{-1} \quad (4.3)$$

Neben der Winkelgeschwindigkeit ω_{Gm} des Gleichstrommotors ist ebenfalls der Radius r_{Lr} der Laufräder bekannt. So kann nun die Geschwindigkeit des Transportbandes v_T entsprechend (4.4) berechnet werden.

$$v_T = \omega_{Gm} \cdot r_{Lr} \quad (4.4)$$

$$= 6.283 \text{ rad s}^{-1} \cdot 0.0181 \text{ m} \quad (4.5)$$

$$= 0.112 \text{ m s}^{-1} \quad (4.6)$$

Es ist ebenfalls notwendig, die gesamte Strecke zu kennen, die das Testobjekt auf dem Transportband zurücklegt. Dafür werden die beiden Punkte P_{Start} und P_{Ende} eingeführt, die diese Strecke eingrenzen (siehe Abb. 4.4b). Der Punkt P_{Start} definiert den Startpunkt des Systems. Er liegt senkrecht über der Antriebsachse und hat einen Wert von $P_{Start} = 0$. Der Punkt P_{Ende} beschreibt den Punkt, an welchem das Testobjekt das Transportband verlässt.

Zur Berechnung von P_{Ende} werden die zwei Abschnitte d_1 und d_2 des Transportbandes miteinander addiert. Der erste Abschnitt ist die Distanz zwischen der Antriebsachse und der äußersten Laufachse. Diese lässt sich am Versuchmodell messen und beträgt $d_1 = 0.236 \text{ m}$.

Der zweite Abschnitt wird durch einen Kreisbogen beschrieben, den das Testobjekt nach der Strecke d_1 bis zum Erreichen des Punktes P_{Ende} nimmt. Dafür muss zuerst der Umfang u_{Ar} ermittelt werden.

$$u_{Ar} = 2\pi \cdot r_{Ar} \quad (4.7)$$

$$= 2\pi \cdot 0.0181 \text{ m} \quad (4.8)$$

$$= 0.112 \text{ m} \quad (4.9)$$

(4.10)

Nach der technischen Zeichnung beträgt dieser Weg $\frac{1}{8}$ des Umfangs eines Laufrads. Folglich ist $d_2 = \frac{1}{8} \cdot u_{Ar}$.

$$d_2 = \frac{1}{8} \cdot u_{Ar} \quad (4.11)$$

$$= \frac{1}{8} \cdot 0.112 \text{ m} \quad (4.12)$$

$$= 0.014 \text{ m} \quad (4.13)$$

Nun kann der Wert des Punktes P_{Ende} berechnet werden. Dabei bezieht sich dieser Wert auf den Abstand zum Punkt P_{Start} .

$$P_{Ende} = d_1 + d_2 \quad (4.14)$$

$$= 0.236 \text{ m} + 0.014 \text{ m} \quad (4.15)$$

$$= 0.25 \text{ m} \quad (4.16)$$

In einer weiteren Berechnung wird der Zeitpunkt t_{Ende} ermittelt, an welchem das Testobjekt das Transportband verlässt.

$$t_{Ende} = v_T \cdot P_{Ende} \quad (4.17)$$

$$= 0.112 \text{ m s}^{-1} \cdot 0.25 \text{ m} \quad (4.18)$$

$$= 2.223 \text{ s} \quad (4.19)$$

4.2 Detektionsbereich

Der Detektionsbereich ist ein Abschnitt des Tunnels, in welchem die Bilddaten von den Testobjekten erzeugt werden. Demnach liefert der Detektionsbereich den Input für das Farberkennungs-Programm.

4.2.1 Aufbau des Detektionsbereichs

Der Aufbau des Detektionsbereichs ist durch verschiedene Aktorik und Sensorik beschrieben. Zu den Sensoren zählen eine Kamera vom Typ RaspiCam sowie eine Reflex-Lichtschranke. Eine 12 V Leuchtdiode, die der Ausleuchtung des Detektionsbereichs dient, wird der Gruppe der Aktuatoren zugeordnet. Die Kamera und die 12 V Leuchtdiode sind zentral über dem Transportband platziert, wohingegen sich die Reflex-Lichtschranke in einer seitlichen Einlassung auf Höhe des Transportbandes befindet (siehe Abb. 4.6c).

Der Detektionsbereich bezeichnet den Bereich des Transportbandes, der von der Kamera überwacht wird und ist durch die Punkte $P_{DbStart}$ und P_{DbEnde} eingegrenzt. Die Abstände dieser Punkte zum Startpunkt P_{Start} wurden in einer Messung ermittelt. Es ergeben sich $P_{DbStart} = 0.044 \text{ m}$ sowie $P_{DbEnde} = 0.149 \text{ m}$. Folglich kann die Länge des Detektionsbereichs ermittelt werden.

$$s_{Db} = P_{DbEnde} - P_{DbStart} \quad (4.20)$$

$$= 0.149 \text{ m} - 0.044 \text{ m} \quad (4.21)$$

$$= 0.105 \text{ m} \quad (4.22)$$

Daraus kann berechnet werden, wie lange ein Testobjekt benötigt, um den Detektionsbereich zu durchqueren.

$$\Delta t_{Db} = \frac{s_{Db}}{v_T} \quad (4.23)$$

$$= \frac{0.105 \text{ m}}{0.112 \text{ m s}^{-1}} \quad (4.24)$$

$$= 0.934 \text{ s} \quad (4.25)$$

Nach den Berechnungen benötigt ein Testobjekt die Zeit $\Delta t_{Db} = 0.934 \text{ s}$ um den Detektionsbereich mit der Länge $s_{Db} = 0.105 \text{ m}$ zu durchqueren.

Der gewählte Aufbau des Detektionsbereichs implementiert die *2. Notwendige Anforderung*. Die Kamera ist so positioniert, dass jedes Testobjekt, das vom Transportband bewegt wird, in ihrem Bildausschnitt erscheint. Zur Erfüllung der *8. Notwendigen Anforderung* befindet sich der Detektionsbereich innerhalb eines Tunnels. Lichteinfall von

außen wird so weit wie möglich abgeschirmt. Der komplette Aufbau des Detektionsbereichs kann Abb. 4.6 entnommen werden.

4.2.2 Ansteuerung des Detektionsbereichs

Die Aktorik und Sensorik des Detektionsbereichs ist durch separate Schaltkreise mit dem Raspberry Pi verbunden. Nachfolgend wird die Ansteuerung der einzelnen Bestandteile beschrieben.

4.2.2.1 RaspiCam

Wie bereits im Abschnitt 2.2.2 beschrieben, wird die RaspiCam über die serielle Schnittstelle CSI vom Raspberry Pi angesteuert. Sobald das Flachband-Kabel der RaspiCam mit dem CSI des Raspberry Pi's verbunden wurde, kann die RaspiCam verwendet werden. Die Raspberry Pi Foundation stellt das Programm *raspistill* zur Verfügung, über das die RaspiCam angesteuert werden kann. Mit *raspistill* können Bilddateien aufgenommen und im Dateisystem gespeichert werden. Da das Programm *raspistill* für die verwendete Hardware geschrieben wurde, erscheint es sinnvoll, diese für die Bildaufnahme zu verwenden.

4.2.2.2 Lichtschranke 1

Aufgrund der einfachen Installation und dem extrem günstigen Preis, wurde sich für die Reflex-Lichtschranke (ohne Polarisationsfilter) HW-201 der Firma Silicon TechnoLabs entschieden. Diese wird mit einer Spannung von 3.3 V - 5 V, bei einem Strom von 20 mA betrieben.

Die Ansteuerung erfolgt über die drei Pins VCC, GND und OUT. VCC und GND dienen der Spannungsversorgung. Über den OUT-Pin sendet die Reflex-Lichtschranke das Signal über den Status der Reflex-Lichtschranke. Dieser Pin ist Active-High. Dementsprechend liegt am OUT-Pin eine 1 an, solange der Lichtstrahl der Reflex-Lichtschranke unterbrochen wird, andernfalls eine 0. Auf diese Weise wird signalisiert, wenn ein Objekt die Reflex-Lichtschranke passiert. Die Reflex-Lichtschranke ist in der Lage, Objekte in einem Bereich von 0 bis 20 cm zu detektieren. Über einen verstellbaren Widerstand kann die Distanz des aktiven Bereichs eingestellt werden.[26]

Die Lichtschranke wird durch die 3.3 V Versorgungsspannung des Raspberry Pi's gespeist. Der OUT-Pin der Lichtschranke ist mit einem GPIO-Pin des Raspberry Pi's verbunden.

Prellen des Lichtschranken-Signals Wie in Abschnitt 2.4.6 bereits erwähnt, ist die verwendete Lichtschranke nicht mit einem Polarisationsfilter ausgestattet. Ein Polarisationsfilter behebt die Probleme, die Reflex-Lichtschranken mit reflektierenden Objekten haben. Reflektierende Objekte brechen den Strahl infraroten Lichts nicht, sondern

wirken selbst als Spiegel. Sie werfen den Lichtstrahl zurück und sind so einen kurzen Moment für die Lichtschranke unsichtbar.

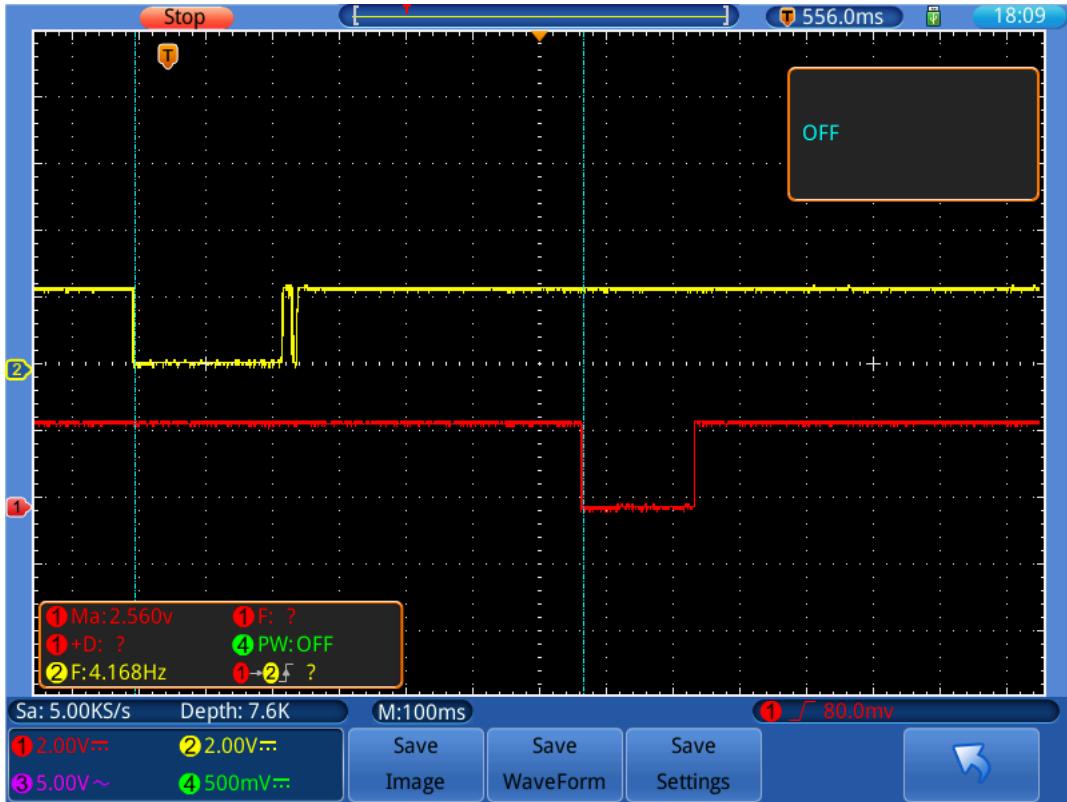
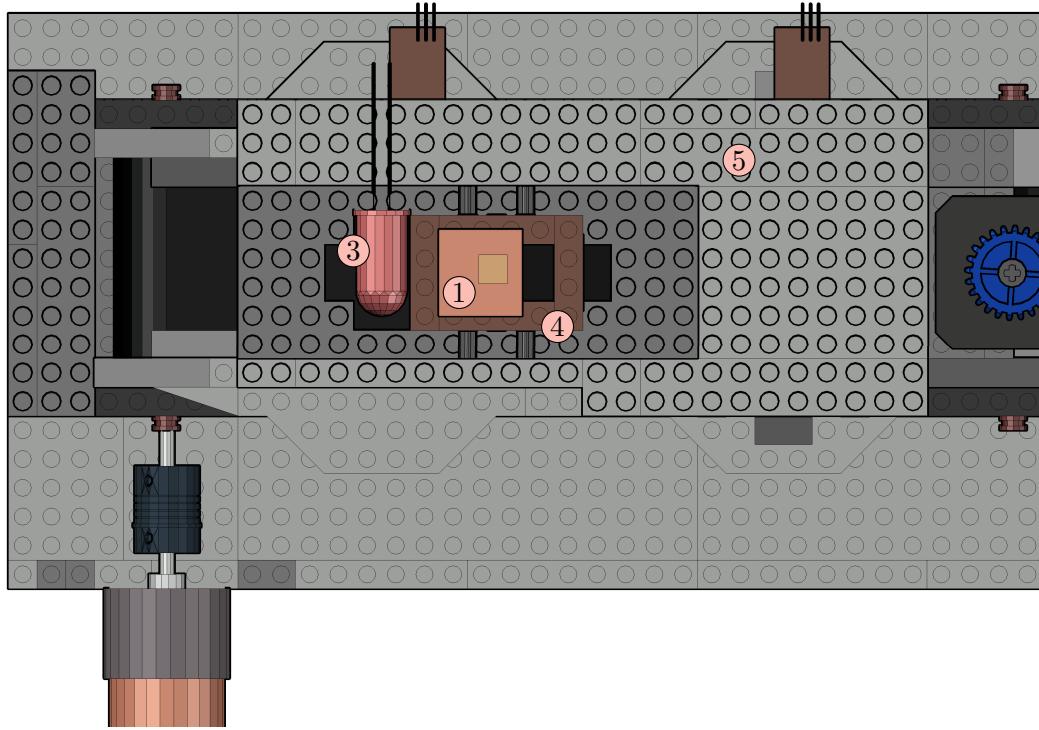


Abbildung 4.5: OUT-Signale der Lichtschranken

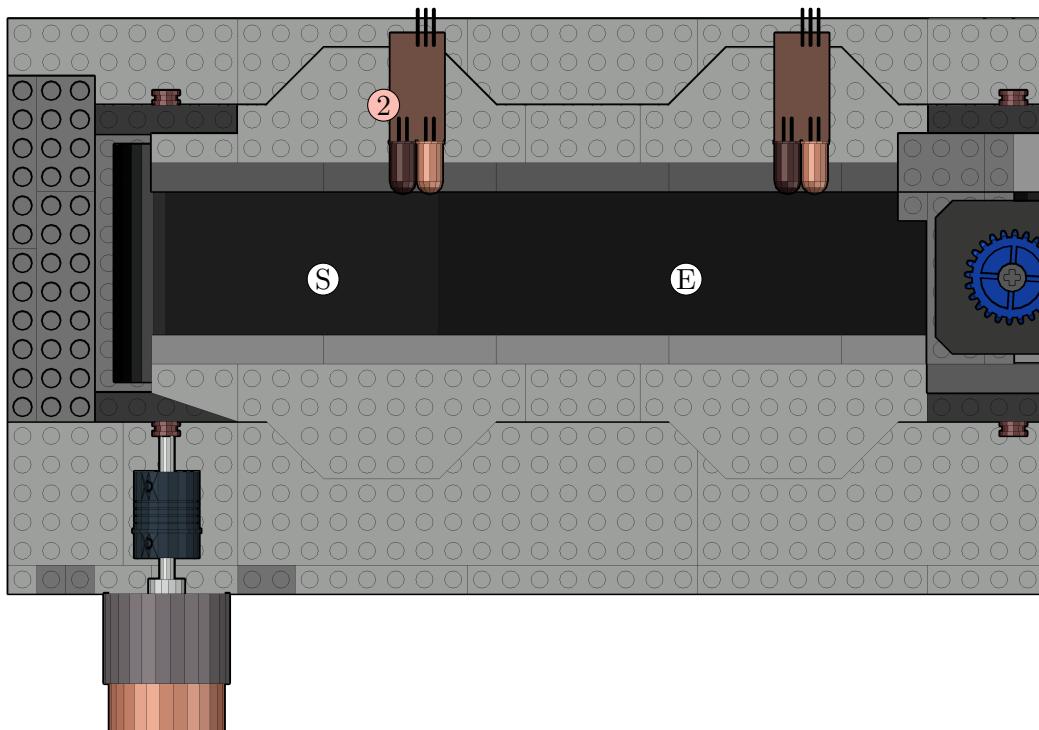
Betrachtet man das OUT-Signal der Lichtschranke mit einem Oszilloskop, ist ein sehr schneller Pegelwechsel („Prellen“) des Signals beim Durchlauf eines Testobjektes erkennbar (siehe gelbes Signal in Abb. 4.5). Die Vermutung liegt nahe, dass das Springen des Signals durch die spiegelnde Oberfläche des Testobjektes erzeugt wird. Um dem entgegenzuwirken, wird ein entsprechendes *Entprellen* im Farberkennungs-Algorithmus implementiert. Dies wird im Abschnitt 5.2.2.2 detailliert erläutert.

4.2.2.3 Leuchtdiode

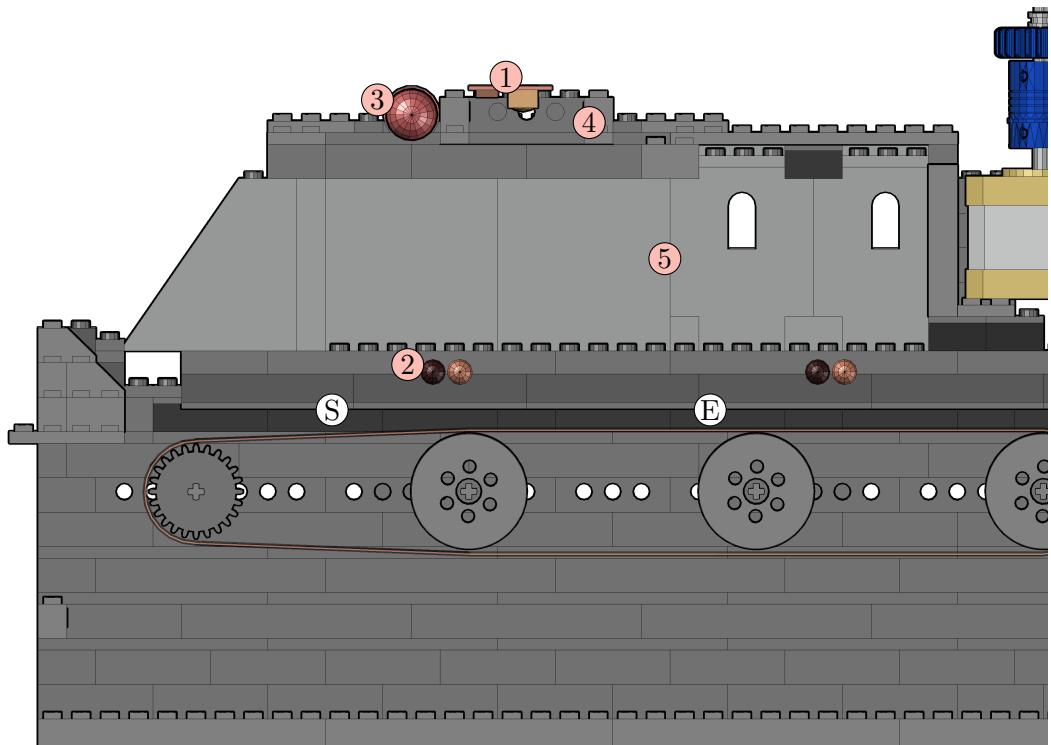
Die verwendete 12 V Leuchtdiode dient zur Ausleuchtung des Detektionsbereichs. Um die Leuchtdiode mit dem Raspberry Pi steuern zu können, befindet sich zwischen den beiden Bauteilen ein 5 V-Relais Modul. Die Ansteuerung des 5 V-Relais Moduls erfolgt entsprechend der theoretischen Beschreibung in Abschnitt 2.4.5. Da der Raspberry Pi die Versorgungsspannung von 12 V nicht liefern kann, wurde eine zusätzliche 12 V Spannungsquelle verwendet.



(a) Ansicht Oben 1



(b) Ansicht Oben 2 (Tunnel ausgeblendet)



(c) Ansicht Schnitt links

- | | | |
|-----------------------|-----------------|------------------------|
| ① RaspiCam | ② Lichtschranke | ③ Leuchtdiode |
| ④ Kamera-Stativ | ⑤ Tunnel | ④ P _{DbStart} |
| ⑤ P _{DbEnde} | | |

Abbildung 4.6: Detektionsbereich Aufbau

4.2.3 Funktionsweise des Detektionsbereichs

Im Detektionsbereich werden die Bilddaten aufgenommen, welche anschließend vom Farberkennungs-Algorithmus verarbeitet werden. Im weiteren Verlauf wird besonders das Zeitverhalten des Detektionsbereichs betrachtet. Zur besseren Veranschaulichung findet sich in Abb. 4.7 eine Übersicht über die Zeitpunkte und Zeitintervalle des Detektionsbereichs.

4.2.3.1 Bildaufnahme

Die Bilddaten müssen gemäß der 8. *Notwendigen Anforderung* unter standardisierten Bedingungen aufgenommen werden. Aus diesem Grund wird die 12 V Leuchtdiode als Lichtquelle verwendet. Umgebungslicht wird weitestgehend vom Detektionsbereich abgeschirmt. Da die 12 V Leuchtdiode die einzige Lichtquelle darstellt, muss sie in der Lage sein, den gesamten Detektionsbereich mit ausreichender Helligkeit auszuleuchten. Ein Lichtstromwert von $\phi_v = 200 \text{ lm}$ scheint dafür zu genügen. Weiterhin muss die 12 V Leuchtdiode einen hohen Farbwiedergabeindex bei passender Farbtemperatur garantieren, da die Farben der Objekte andernfalls verfälscht werden würden. Die Werte für den Farbwiedergabeindex von $Ra > 80$ und die Farbtemperatur von 2700 K genügen ebenfalls. Da die Position der Kamera statisch ist, sind alle Bedingungen erfüllt, dass jedes Bild zu den gleichen Bedingungen aufgenommen wird.

Verarbeitungszeiten der Bildaufnahme Damit der Detektionsbereich die Eingangsdaten für den Farberkennungs-Algorithmus zuverlässig und rechtzeitig zur Verfügung stellen kann, muss das Zeitverhalten der Komponenten des Detektionsbereichs gut aufeinander abgestimmt sein.

Zuerst galt es die Zeit zu ermitteln, die zwischen der Aufnahme bis zur vollständigen Abspeicherung eines Bildes vergehen. Um die Zeit t_{Bild} zwischen der Aufnahme bis zur vollständigen Abspeicherung eines Bildes zu ermitteln, wurde eine Messreihe mit je 100 Aufnahmen durchgeführt. Bei dieser wurden die Verarbeitungszeiten des Ablaufs sowohl bei der Aufnahme einzelner Bilder, wie auch bei der periodischen Aufnahme im *timelapse*-Modus gemessen. Die Messung erfolgte über die Zeitstempel der System-Log-Nachrichten des *raspistill*-Programms. Dabei wurde die Vorbereitungszeit t_{BSet} sowie die Durchführungszeit t_{BEexe} einer Aufnahme gemessen. Die arithmetischen Mittel dieser Zeiten über die je 100 Aufnahmen können Tabelle 4.1 entnommen werden.

Modus	Δt_{BSet}	Δt_{BEexe}
Einzelne Bildaufnahme	0.341 s	0.239 s
Periodische Bildaufnahme	0.187 s	0.239 s

Tabelle 4.1: Bildverarbeitungszeiten des Programms *raspistill*

Bei der Aufnahme eines einzelnen Bildes setzt sich die Verarbeitungszeit aus der Vorbereitungszeit einer Aufnahme und der Durchführungszeit einer Aufnahme zusammen.

$$\Delta t_{BildE} = \Delta t_{BSetE} + \Delta t_{BExeE} \quad (4.26)$$

$$= 0.341 \text{ s} + 0.239 \text{ s} \quad (4.27)$$

$$= 0.58 \text{ s} \quad (4.28)$$

Bei der periodischen Bildaufnahme muss die Initialisierung nur ein einziges Mal zum Start des *raspistill*-Programms erfolgen. Anschließend benötigt das Programm nur noch die Durchführungszeit, um eine Aufnahme zu tätigen und abzuspeichern.

$$\Delta t_{BildP} = \Delta t_{BExeP} \quad (4.29)$$

$$= 0.239 \text{ s} \quad (4.30)$$

Daraus ergibt sich die Notwendigkeit, das *raspistill*-Programm während der Initialisierungs-Routine zu starten und über die gesamte Programmlaufzeit laufen zu lassen.

Im weiteren Verlauf gilt demnach $t_{Bild} = t_{BildP}$.

Zeitbedingung des Detektionsbereichs Der Zeitpunkt t_{Obj} definiert den Moment, an welchem ein neues Testobjekt auf das Transportband gelegt wird. Da dieser Zeitpunkt unbekannt ist, kann auch nicht vorherbestimmt werden, zu welchem konkreten Zeitpunkt *raspistill* eine Aufnahme vom Testobjekt macht. Allerdings ist aus Abschnitt 4.2.1 das Zeitintervall Δt_{Db} bekannt, das beschreibt, wie lange ein Testobjekt benötigt, um den Detektionsbereich zu durchqueren.

Daraus ergibt sich ein Zeitfenster, in welchem *raspistill* eine Aufnahme machen muss. Dieses Zeitfenster stellt eine Zeitbedingung im Sinne der Definition von Echtzeit dar (siehe Abschnitt 2.5). Sie lässt sich als festes Zeitintervall einordnen, welches durch die Zeitpunkte $t_{DbStart}$ und t_{DbEnde} eingegrenzt wird. Tritt das Ereignis nicht innerhalb der Zeitbedingung ein, verfällt das Ergebnis, da die Kamera keine Aufnahme vom Objekt macht. Die Zeitbedingung ist aperiodisch, da sie sich nach dem Zeitpunkt t_{Obj} richtet, der nicht vorhergesagt werden kann.

Die genannte Zeitbedingung wird im weiteren Verlauf als Δt_A bezeichnet. Wie beschrieben, muss sie kleiner sein als das Zeitfenster Δt_{Db} .

Während der Erprobung kam es gelegentlich zu Durchgängen, in denen sich Testobjekte nach der Einspeisung um ein paar Millimeter auf dem Transportband bewegten. Dies ist durch die runde Form der Testobjekte zu erklären. So konnte es passieren, dass diese Testobjekte den Detektionsbereich schneller durchquerten als eingangs berechnet. Um dem entgegenzuwirken, ist ein zeitlicher Puffer von $\frac{1}{10} \cdot \Delta t_{Db}$ in die Berechnung von Δt_A mit eingeflossen.

$$\Delta t_A = \Delta t_{Db} - \frac{1}{10} \cdot \Delta t_{Db} \quad (4.31)$$

$$= 0.934 \text{ s} - 0.0934 \text{ s} \quad (4.32)$$

$$= 0.84 \text{ s} \quad (4.33)$$

Daraus folgt, dass *raspistill* im Zeitintervall von $\Delta t_A = 0.84 \text{ s}$ Aufnahmen des Detektionsbereichs machen muss, um die Zeitbedingung des Detektionsbereichs sicher einhalten zu können.

4.2.3.2 Auslösen des Farberkennungs-Algorithmus

Um dem Farberkennungs-Algorithmus mitzuteilen, dass ein Testobjekt den Detektionsbereich durchquert, ist eine Reflex-Lichtschranke innerhalb des Detektionsbereichs installiert. Wird die Lichtschranke vom Objekt durchbrochen, ändert sich das OUT-Signal der Lichtschranke, welches mit einem GPIO-Pin des Raspberry Pi's verbunden ist.

Die Pegeländerung des OUT-Signals wird von einem Interrupt Handler innerhalb des Farberkennungs-Algorithmus verarbeitet. Sobald sich der Pegel des OUT-Signals ändert, ruft der Interrupt Handler eine Interrupt Service Routine auf. Die Interrupt Service Routine liest anschließend die aktuelle Bilddatei vom Detektionsbereich ein und verarbeitet diese, um die Farbe des Testobjekts zu erkennen.

Da die Lichtschranke des Detektionsbereichs den Farberkennungs-Algorithmus startet, ist es von großer Bedeutung, an welcher Position die Lichtschranke platziert wird.

Die Position der Lichtschranke lässt sich mithilfe der gegebenen Parameter berechnen. Entscheidend sind die Werte der Transportband-Geschwindigkeit v_T , der Zeitpunkt $t_{DbStart}$ sowie der Zeitintervall Δt_{Bild} .

$$t_{DbStart} = \frac{P_{DbStart}}{v_T} \quad (4.34)$$

$$= \frac{0.044 \text{ m}}{0.112 \text{ m s}^{-1}} \quad (4.35)$$

$$= 0.391 \text{ s} \quad (4.36)$$

Aus diesen Werten kann der frühestmögliche Zeitpunkt t_{BMin} errechnet werden, zu dem die Aufnahme eines Testobjekts vollständig abgespeichert sein könnte.

$$t_{BMin} = t_{DbStart} + \Delta t_{Bild} \quad (4.37)$$

$$= 0.391 \text{ s} + 0.239 \text{ s} \quad (4.38)$$

$$= 0.63 \text{ s} \quad (4.39)$$

Damit das Testobjekt sicher erkannt wird, muss der frühestmögliche Zeitpunkt einer Bildaufnahme t_{BMin} identisch zu dem Zeitpunkt t_{Ls1} sein, an welchem das Testobjekt die erste Lichtschranke durchbricht.

$$t_{Ls1} = t_{BMin} \quad (4.40)$$

$$= 0.63 \text{ s} \quad (4.41)$$

Daraus berechnet sich die Position P_{Ls1} der ersten Lichtschranke.

$$P_{Ls1} = v_T \cdot t_{BMin} \quad (4.42)$$

$$= 0.112 \text{ m s}^{-1} \cdot 0.63 \text{ s} \quad (4.43)$$

$$= 0.071 \text{ m} \quad (4.44)$$

Die Lichtschranke des Detektionsbereich muss demnach 0.071 m ausgehend vom Startpunkt des Systems P_{Start} platziert werden. Somit ist gewährleistet, dass der Farberkennungs-Algorithmus die erste Aufnahme, die ein Testobjekt enthalten könnte, analysiert.

Weiterhin muss eingegrenzt werden, wie lange der Farberkennungs-Algorithmus die Aufnahmen auf die Präsenz von Testobjekten überprüfen muss. Dazu wird der spätestmögliche Zeitpunkt einer Bildaufnahme t_{BMax} bestimmt.

$$t_{BMax} = t_{DbStart} + \Delta t_A + \Delta t_{Bild} \quad (4.45)$$

$$= 0.391 \text{ s} + 0.84 \text{ s} + 0.239 \text{ s} \quad (4.46)$$

$$= 1.47 \text{ s} \quad (4.47)$$

Da der genaue Zeitpunkt der Bildaufnahme nicht bekannt ist, muss der Farberkennungs-Algorithmus jede neue Aufnahme zwischen den Zeitpunkten t_{BMin} und t_{BMax} auf die Präsenz eines Testobjekts untersuchen. Diese Zeitpunkte definieren folglich ein Zeitfenster Δt_P , in welchem das Bild eines Testobjekts vollständig im Speicher des Raspberry Pi's vorliegt.

$$\Delta t_P = t_{BMax} - t_{BMin} \quad (4.48)$$

$$= 1.47 \text{ s} - 0.63 \text{ s} \quad (4.49)$$

$$= 0.84 \text{ s} \quad (4.50)$$

Demnach muss der Farberkennungs-Algorithmus die erstellten Aufnahmen für den Zeintervall $\Delta t_P = 0.84 \text{ s}$ auf die Präsenz bewegter Objekte überprüfen. Eine detaillierte Beschreibung zu diesem Vorgehen findet sich im Abschnitt 5.2.2.2.

4.2.3.3 Ergebnis des Farberkennungs-Algorithmus

Sobald der Farberkennungs-Algorithmus die Präsenz eines Testobjekts in einer Aufnahme feststellt, beginnt er die Farbe des Objekts zu detektieren. Nachdem er die Farbe erfolgreich erkannt hat, schreibt er diese mit einem System-Aufruf in eine Message Queue (siehe Abschnitt 2.3.3).

Das Zeitintervall Δt_{Cd} beschreibt die Zeit, die er benötigt, um diesen Ablauf einmal komplett zu durchlaufen. Um Δt_{Cd} zu ermitteln, wurde eine Messreihe von 50 Abläufen durchgeführt. Die Verarbeitungszeiten lagen dabei zwischen $\Delta t_{CdMin} = 0.029\text{ s}$ und $\Delta t_{CdMax} = 0.037\text{ s}$. Zur weiteren Berechnung wurde Δt_{Cd} aufgerundet.

$$\Delta t_{Cd} = 0.04\text{ s} \quad (4.51)$$

Mithilfe der nun berechneten Zeiten kann genau bestimmt werden, zu welchem Zeitpunkt t_F die Farbe spätestens in die Message Queue geschrieben wurde.

$$t_F = t_{Ls1} + \Delta t_P + \Delta t_{Cd} \quad (4.52)$$

$$= 0.63\text{ s} + 0.84\text{ s} + 0.04\text{ s} \quad (4.53)$$

$$= 1.51\text{ s} \quad (4.54)$$

Abschließend kann das gesamte, maximale Zeitintervall der Farberkennung Δt_{Fe} ermittelt werden. Dieses ergibt sich von dem Zeitpunkt $t_{DbStart}$, an dem ein Testobjekt frühestens im Detektionsbereich erscheint, bis zu dem Zeitpunkt t_F , an dem die erkannte Farbe spätestens in der Message Queue steht.

$$\Delta t_{Fe} = t_F - t_{DbStart} \quad (4.55)$$

$$= 1.51\text{ s} - 0.391\text{ s} \quad (4.56)$$

$$= 1.119\text{ s} \quad (4.57)$$

Folglich benötigt das System exakt 1.119 s für den ganzen Ablauf einer Farberkennung.

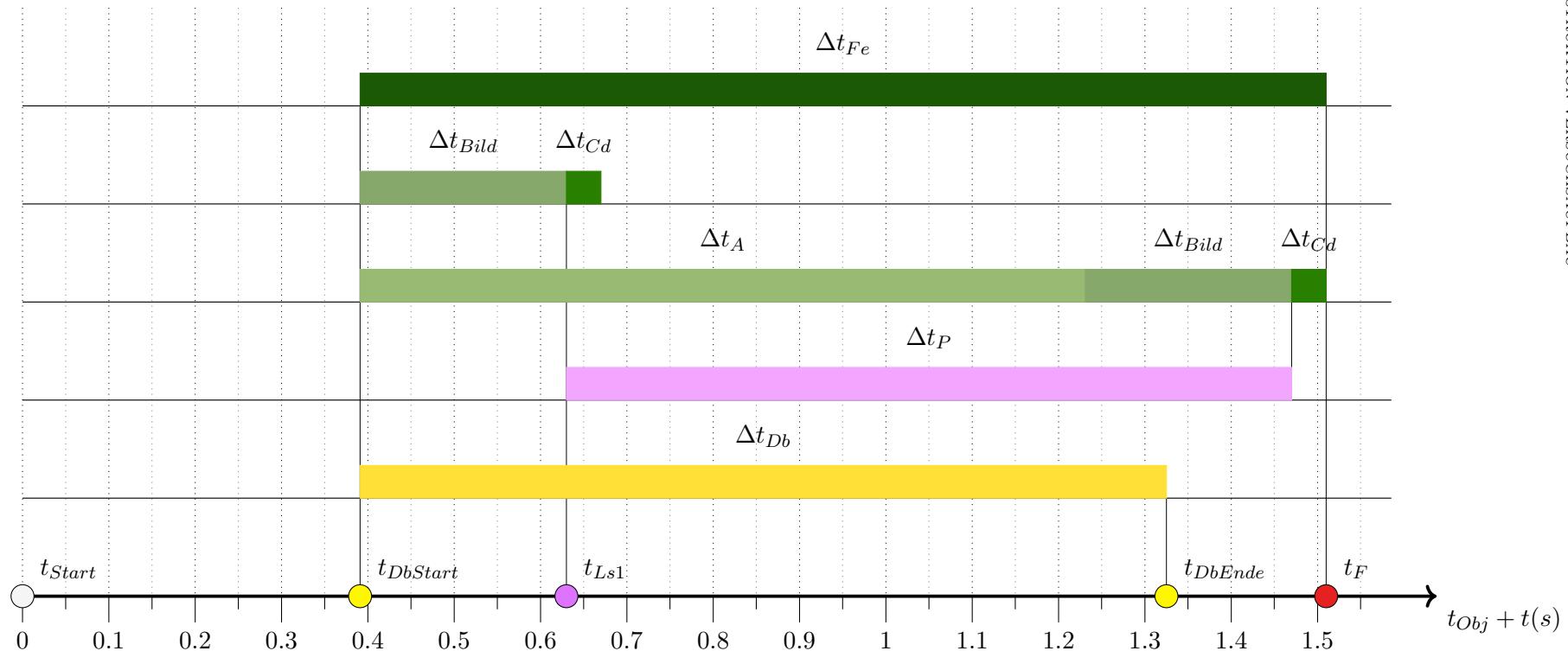


Abbildung 4.7: Zeitverhalten des Detektionsbereichs

4.3 Mechanische Sortierung

Das Teilsystem *mechanische Sortierung* hat die Aufgabe, die Testobjekte gemäß ihrer jeweiligen Farbe in unterschiedliche Fächer zu leiten. Dafür erhält sie die Information über die detektierte Farbe vom Farberkennungs-Algorithmus und steuert auf dessen Grundlage einen Schieber.

Da das Zeitverhalten der einzelnen Komponenten auch in der mechanischen Sortierung eine große Rolle spielt, findet sich in Abb. 4.11 eine Übersicht aller Zeitpunkte und Zeitintervalle der mechanischen Sortierung.

4.3.1 Aufbau der mechanischen Sortierung

Die mechanische Sortierung ist am Ende des Transportbandes platziert und beschreibt das letzte Teilsystem des Versuchsmodells.

Sie besteht aus einer trapezförmigen Rutsche, einem zweiteiligen Schieber, einem Getriebe und einem Auffangbehälter mit sechs Sortierfächern. Außerdem zählen ein NEMA 17 Schrittmotor, dessen A4988 Motortreiber und eine Reflex-Lichtschranke zu den Komponenten der mechanischen Sortierung. Der genaue Aufbau kann Abb. 4.8 entnommen werden.

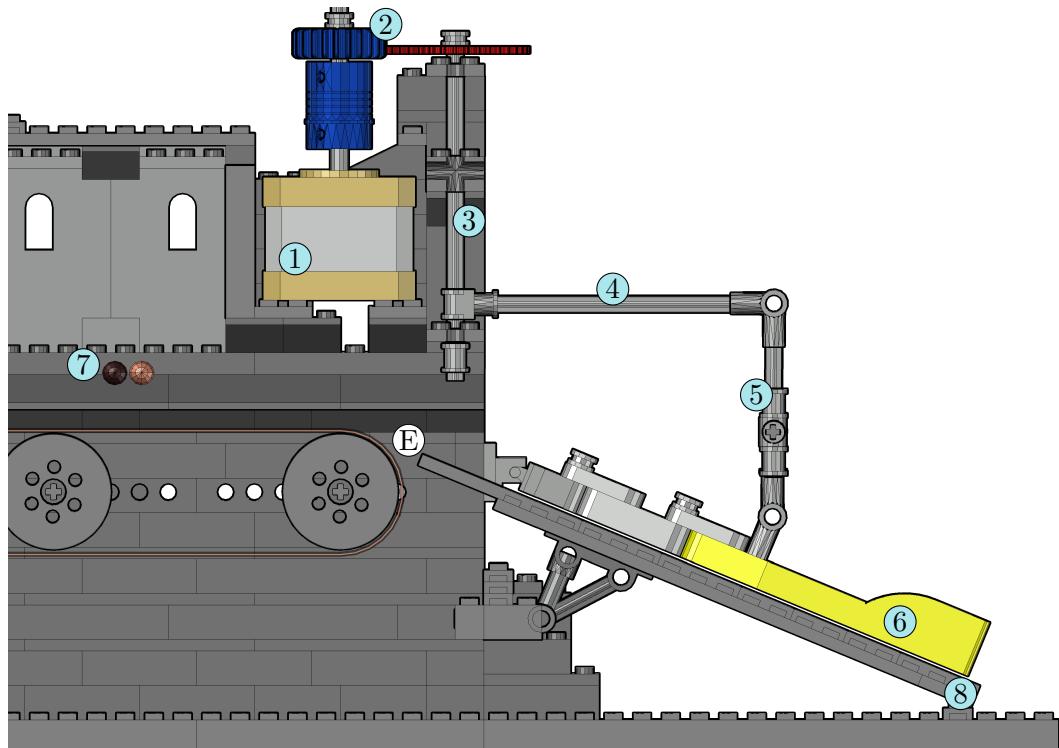
Das obere Ende der Rutsche grenzt an den Punkt P_{Ende} , der das Ende des Transportbandes definiert. Am unteren Ende der Rutsche ist ein austauschbarer Auffangbehälter mit sechs, jeweils 0.03 m breiten Sortierfächern angebracht. Nach Abschluss der Sortierung befinden sich hier die Testobjekte.

Der Schieber liegt auf der Fläche der Rutsche auf. Die zwei Seiten des Schiebers sind über Gelenke mit dem Tunnel verbunden und somit in ihrer Position verstellbar. Der Abstand der zwei Seiten beträgt zwischen den Gelenken 0.088 m. In Richtung des Auffangbehälters reduziert sich der Abstand auf eine Breite von 0.045 m, sodass jedes der sechs Sortierfelder präzise angesteuert werden kann. Die beiden Seiten des Schiebers sind durch eine Verbindungsachse miteinander verbunden. So können sie mit nur einem Motor synchron gesteuert werden.

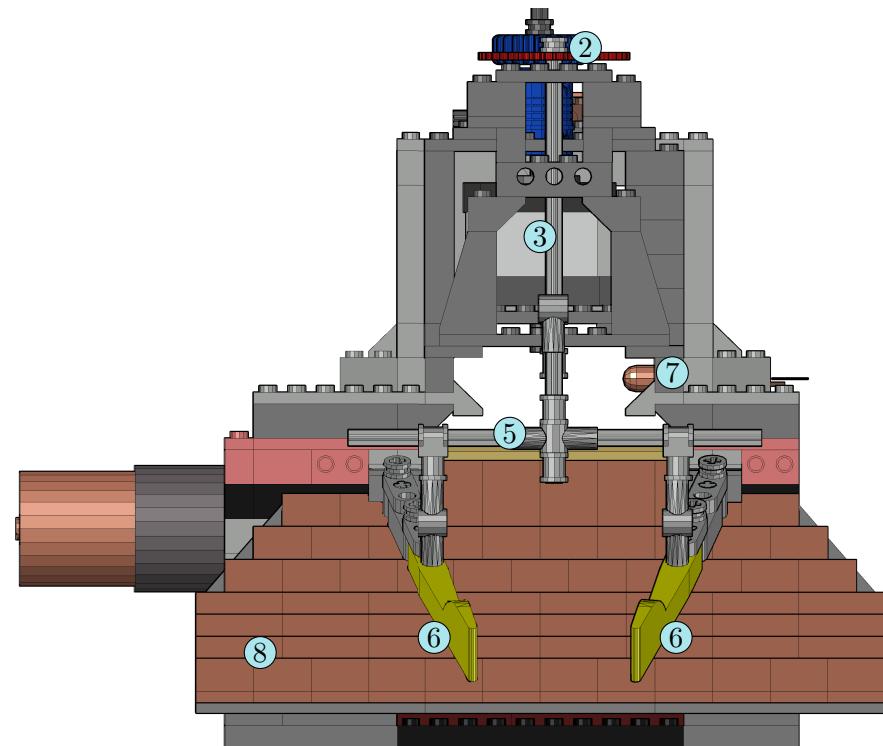
Zur Steuerung des Schiebers wird ein NEMA 17 Schrittmotor verwendet. Der Schrittmotor arbeitet in Kombination mit einem Getriebe und einer Steuerachse. Diese Bestandteile sind zentral über dem Ende des Tunnels montiert. Das Getriebe besteht aus zwei Zahnrädern. Die Steuerachse ist senkrecht montiert und über eine Verlängerung mit der Verbindungsachse der zwei Seiten des Schiebers verbunden (siehe Abb. 4.8a).

Zur Ansteuerung des Schrittmotors wird ein A4988 Motortreiber verwendet. Dieser ist auf einem Steckfeld seitlich des Modells platziert.

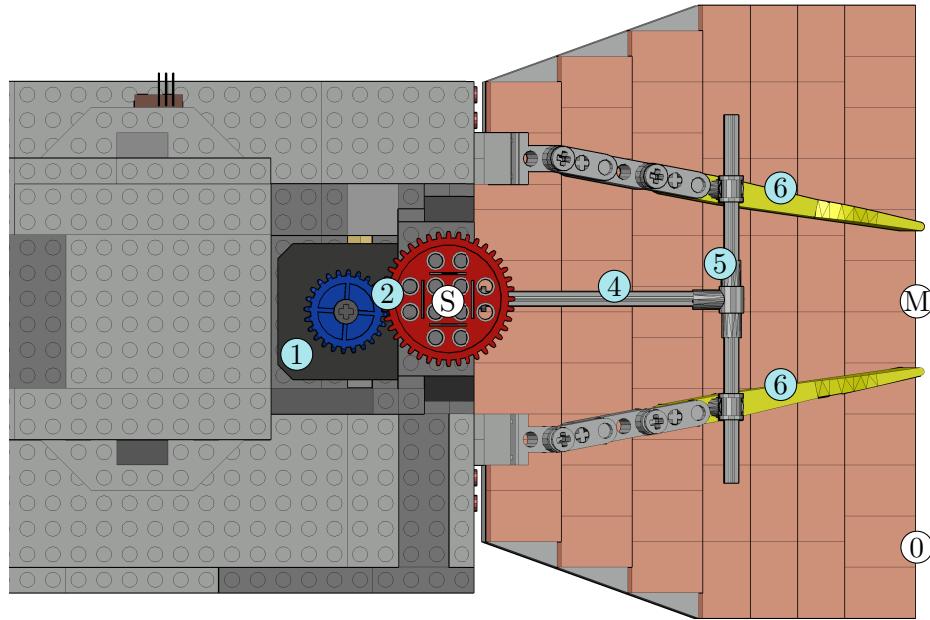
Des Weiteren kommt eine weitere Reflex-Lichtschranke zum Einsatz. Diese befindet sich seitlich innerhalb des Tunnels. Sie ist so platziert, dass sie durchbrochen wird, sobald ein Testobjekt an ihr vorbei transportiert wird.



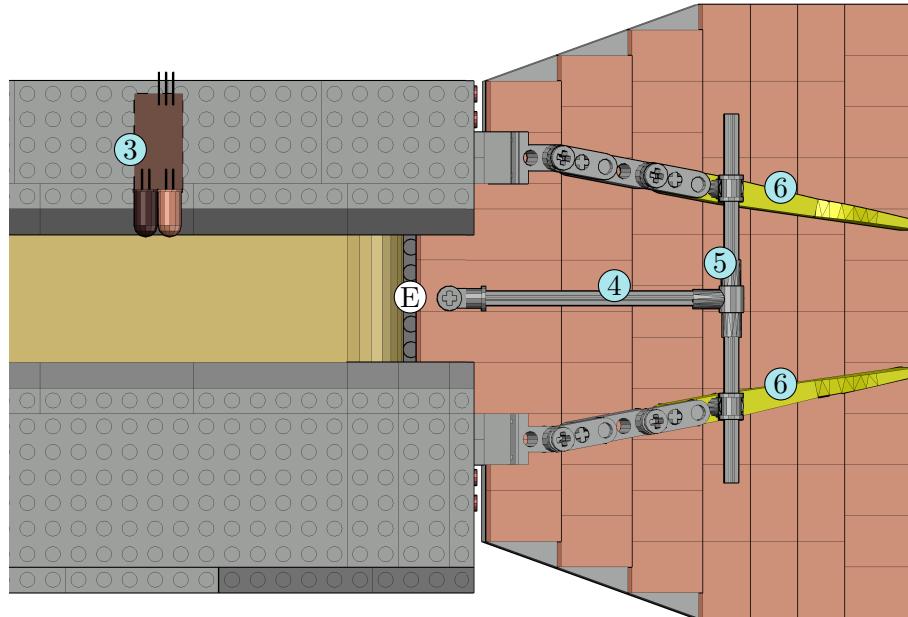
(a) Ansicht Schnitt links



(b) Ansicht Vorne



(c) Ansicht Oben 1



(d) Ansicht Oben 2 (Tunnel ausgeblendet)

- | | | |
|-----------------|--------------------|---------------|
| ① Schrittmotor | ② Getriebe | ③ Steuerachse |
| ④ Verlängerung | ⑤ Verbindungsachse | ⑥ Schieber |
| ⑦ Lichtschranke | ⑧ Rutsche | ⑭ P_{Ende} |
| ⑨ P_{Sa} | ⑩ P_0 | ⑮ P_{Mu} |

Abbildung 4.8: Mechanische Sortierung Aufbau

4.3.2 Ansteuerung der mechanischen Sortierung

Die mechanische Sortierung umfasst elektrische und mechanische Elemente, die mit dem Raspberry Pi verbunden sind und von ihm angesteuert werden.

4.3.2.1 NEMA 17 Schrittmotor

Der NEMA 17 Schrittmotor ist ein 2-phägiger Schrittmotor. Er wird über vier Eingänge angesteuert, von denen jeweils zwei intern über Spulen miteinander verbunden sind. Sobald sich zwei verbundene Eingänge im Pegel unterscheiden, fließt Strom durch die Spule und ein Magnetfeld entsteht. Ein planmäßiges, aufeinanderfolgendes An- und Abschalten von Magnetfeldern über den zwei Spulen, sorgt für eine Rotation der Antriebswelle, den sogenannten Schritt. Beim NEMA 17 Schrittmotor misst ein Schritt genau 1.8° .

Die Ansteuerung des Schrittmotors kann jedoch nicht mit dem Raspberry Pi, sondern nur über einen Motortreiber realisiert werden. Eine genauere Erklärung zur Funktionsweise des NEMA 17 Schrittmotors findet sich im Abschnitt 2.4.1.

4.3.2.2 A4988 Motortreiber

Der A4988 Motortreiber ist ein speziell für Schrittmotoren entwickelter Motortreiber. Der Motortreiber ermöglicht eine einfache Ansteuerung der vier Eingangssignale des Schrittmotors, indem er die Abfolge der Pegeländerungen als interne Schaltung realisiert. Seitens des Raspberry Pi's muss zur Erzeugung eines Schrittes nur noch das STEP-Signal von 0 auf 1 und umgekehrt geschaltet werden. Weiterhin stellt der Motortreiber die Signale **ENABLE** und **DIRECTION** zur Verfügung, deren Funktion gleich ihrer jeweiligen Namen ist.

Die Verbindung des Schrittmotors mit der geforderten Versorgungsspannung wird ebenfalls über den Motortreiber ermöglicht. Eine detaillierte Übersicht der Ansteuerung kann der Abb. 4.2 entnommen werden.

4.3.2.3 Reflex-Lichtschranke

Die in der mechanischen Sortierung eingesetzte Reflex-Lichtschranke ist gleich der Lichtschranke, die für den Detektionsbereich verwendet wird. Sämtliche Punkte der Ansteuerung sind identisch zu jenen, die in Abschnitt 4.2.2.2 beschrieben werden.

4.3.2.4 Schieber

Zur Ansteuerung des Schiebers existiert eine Kombination verschiedener Elemente zwischen der Verbindungsachse der Schieberseiten und dem Schrittmotor.

Die Antriebswelle des Schrittmotors ist über eine Achsenkupplung mit einer kreuzförmigen Welle verbunden, auf welcher das kleinere Zahnrad Z_k mit dem Radius $r_{ZrKlein} = 0.02\text{ m}$ sitzt. Das Zahnrad Z_k ist mit dem größeren Zahnrad Z_g verzahnt, mit welchem es gemeinsam das Getriebe bildet. Das Zahnrad Z_g hat einen Radius

$r_{ZrGroß} = 0.04 \text{ m}$ und sitzt auf der vertikal installierten Steuerachse. Wenn die Steuerachse rotiert, wird dadurch die Verbindungsachse der Schieber bewegt, mit der sie über eine Verlängerung verbunden ist.

Gemäß der 6. notwendigen Anforderung soll der Schieber in der Lage sein, alle sechs verschiedenen Sortierfächer präzise anzusteuern. Dazu muss die Anzahl der Motorschritte n_S berechnet werden, die benötigt werden, um die Position des Schiebers um ein Sortierfeld zu versetzen.

Zur Bestimmung von n_S muss zuerst die Rotation der Steuerachse pro Schritt des Schrittmotors berechnet werden. Es ist bekannt, dass die Antriebsachse des Schrittmotors bei einem Schritt mit einem Rotationswinkel von $\gamma_M = 1.8^\circ$ rotiert. Folglich rotiert das Zahnrad Z_k mit dem selben Rotationswinkel $\gamma_k = \gamma_M$. Zur Ermittlung des Rotationswinkels γ_g des Zahnrads Z_g muss also zuerst die Übersetzung i des Getriebes berechnet werden. Diese lautet nach [Wittel2013]:

$$i = \frac{r_{ZrKlein}}{r_{ZrGroß}} \quad (4.58)$$

$$= \frac{0.02 \text{ m}}{0.04 \text{ m}} \quad (4.59)$$

$$= 0.5 \quad (4.60)$$

Nun kann der Rotationswinkel γ_g des Zahnrads Z_g ebenfalls nach [Wittel2013] berechnet werden.

$$\gamma_g = i \cdot \gamma_k \quad (4.61)$$

$$= 0.5 \cdot 1.8^\circ \quad (4.62)$$

$$= 0.9^\circ \quad (4.63)$$

Da das Zahnrad Z_g auf der Steuerachse montiert ist, rotieren beide mit dem selben Drehwinkel.

$$\gamma_{Sa} = \gamma_g \quad (4.64)$$

Rotiert der Schrittmotor innerhalb eines Schritts um 1.8° , so rotiert das Zahnrad Z_g und die damit verbundene Steuerachse um 0.9° . Dasselbe gilt für die Verbindung, die senkrecht zur Steuerachse montiert ist.

Für die weitere Berechnung wird die Hilfszeichnung Abb. 4.9 eingeführt. Sie zeigt die Positionen der Steuerachse (P_{Sa}) sowie die Position der Mittelpunkte aller Sortierfelder (P_0 bis P_5) zentral von oben.

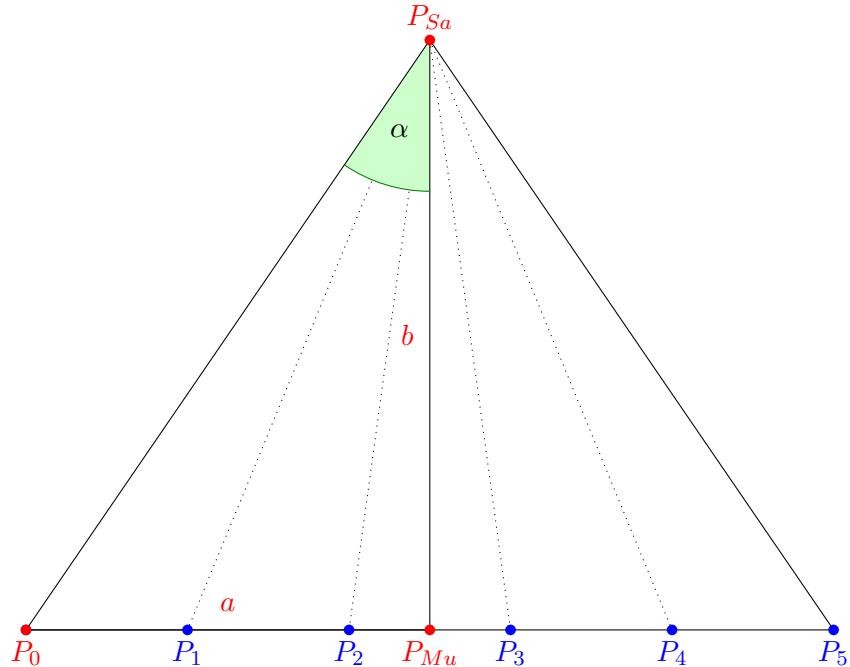


Abbildung 4.9: Hilfszeichnung für die Berechnung der Rutschen-Parameter

Um mit diesen Werten arbeiten zu können, werden die Längen der Strecken $\overline{P_0P_{Mu}}$ (nachfolgend a) und $\overline{P_{Sa}P_{Mu}}$ (nachfolgend b) benötigt. Die Werte dieser Längen wurden einer Messung entnommen.

$$a = 0.089 \text{ m} \quad (4.65)$$

$$b = 0.13 \text{ m} \quad (4.66)$$

Die Punkte P_{Sa} , P_{Mu} und P_0 bilden ein rechtwinkliges Dreieck. Da die Längen a und b bekannt sind, kann der Winkel α berechnet werden.

$$\tan \alpha = \frac{a}{b} \quad (4.67)$$

$$\alpha = \arctan \frac{a}{b} \quad (4.68)$$

$$= \arctan \frac{0.089}{0.13} \quad (4.69)$$

$$= 27.913^\circ \quad (4.70)$$

Im nächsten Schritt wird der Winkel α_5 berechnet. Dieser stellt den gesamten Bereich dar, welcher von der Sortierung abgedeckt werden soll.

$$\alpha_5 = 2 \cdot \alpha \quad (4.71)$$

$$= 2 \cdot 27.913 \quad (4.72)$$

$$= 55.826^\circ \quad (4.73)$$

Folglich muss die Steuerachse um 55.826° rotieren, um die Schieber vom ersten Sortierfeld (P_0) zum letzten Sortierfeld (P_5) zu bewegen.

Aus den ermittelten Werten kann nun das gesuchte n_S berechnet werden. Dazu wird zuerst der Abstandswinkel zwischen den Mittelpunkten zweier benachbarter Sortierfelder bestimmt. Die Anzahl möglicher Feldwechsel wird dabei durch den Parameter n_W beschrieben.

$$\alpha_1 = \frac{\alpha_5}{n_W} \quad (4.74)$$

$$= \frac{55.826^\circ}{5} \quad (4.75)$$

$$= 11.165^\circ \quad (4.76)$$

Nun kann diesem Abstandswinkel und dem Drehwinkel γ_{Sa} die Anzahl der erforderlichen Schritte n_S bestimmt werden.

$$n_S = \frac{\alpha_1}{\gamma_{Sa}} \quad (4.77)$$

$$= \frac{11.165^\circ}{0.9^\circ} \quad (4.78)$$

$$= 12.406 \approx 12 \quad (4.79)$$

Der Schrittmotor muss also 12 Schritte absolvieren, damit der Schieber seine Position um ein Feld verändert.

4.3.3 Funktionsweise der mechanischen Sortierung

Um die Echtzeitfähigkeit des Systems zu garantieren, ist es wichtig, die mechanische Sortierung eng auf das Verhalten des Farberkennungs-Algorithmus und die Position des Testobjekts abzustimmen. Der Schieber darf nicht zu früh angesteuert werden, da sonst ein vorheriges Testobjekts, das sich noch im Tunnel befindet, im falschen Sortierfach landen könnte. Er darf ebenso wenig zu spät angesteuert werden, da sich sonst das aktuelle Testobjekt bereits auf der Rutsche befinden würde, bevor der Schieber die korrekte Position vollständig eingenommen hat.

4.3.3.1 Zeitbedingung der mechanischen Sortierung

Folglich ergibt sich aus der Ansteuerung der Sortiersteuerung eine weitere Zeitbedingung. Sie hat ebenfalls die Eigenschaften einer festen Echtzeitbedingung, da ein Ereignis zum falschen Zeitpunkt zu einem falschen Endergebnis führen würde.

Die Zeitbedingung wird durch einen genauen Zeitpunkt t_{Sort} definiert, an welchem die mechanische Sortierung angesteuert werden muss. Dieser Zeitpunkt t_{Sort} sollte so spät wie möglich stattfinden, um das Ergebnis vorheriger Testobjekte nicht durch einen verfrühten Positionswechsel des Schiebers zu verfälschen.

Der Zeitpunkt t_{Sort} ist dabei abhängig von der Zeitspanne Δt_{W5} , die der Schieber für einen Positionswechsel über die maximal mögliche Distanz benötigt.

Im vorherigen Abschnitt wurde berechnet, dass für den Positionswechsel eines Feldes 12 Schritte benötigt werden. Folglich gilt es nun zu ermitteln, wie groß die Zeitspanne Δt_{W1} ist, die der Schieber für einen Feldwechsel benötigt.

Zu diesem Zweck wurde eine Messreihe von 50 Versuchen durchgeführt. In jeder der Messungen wurde die Lichtschranke der Sortiermechanik durchbrochen, wodurch der Sortiersteuerungs-Algorithmus aktiviert wurde. Dieser war so manipuliert, dass er bei jedem Signal der Lichtschranke den Schrittmotor um 12 Schritte bewegte. Dies entspricht der zuvor ermittelten Anzahl an Schritten für einen Sortierfach-Wechsel. Dabei wurden die Zeitspannen Δt_{Setup} und Δt_S gemessen. Δt_{Setup} definiert die Zeit, die der Sortiersteuerungs-Algorithmus vom Empfangen des Interrupt-Signals bis zum Beginn der Ansteuerung des Schrittmotors benötigt. Das Zeitintervall Δt_S hingegen beschreibt die Zeit für einen einzelnen Motorschritt.

```
src/sorting_control.c/motor_step          Measure Time for: 12 Steps
src/sorting_control.c/motor_step          t(us) for 1 step is: 6393
src/sorting_control.c/motor_step          t(us) for 1 step is: 6152
src/sorting_control.c/motor_step          t(us) for 1 step is: 6178
src/sorting_control.c/motor_step          t(us) for 1 step is: 6185
src/sorting_control.c/motor_step          t(us) for 1 step is: 6175
src/sorting_control.c/motor_step          t(us) for 1 step is: 6173
src/sorting_control.c/motor_step          t(us) for 1 step is: 6161
src/sorting_control.c/motor_step          t(us) for 1 step is: 6167
src/sorting_control.c/motor_step          t(us) for 1 step is: 6165
src/sorting_control.c/motor_step          t(us) for 1 step is: 6193
src/sorting_control.c/motor_step          t(us) for 1 step is: 6158
src/sorting_control.c/motor_step          t(us) for 1 step is: 6158
src/sorting_control.c/motor_step          Measure Time: Finished
```

Abbildung 4.10: Auszug der Messung für die Ermittlung von Δt_S

Die Zeiten für einen Schritt des Schrittmotors lagen zwischen $\Delta t_{SMin} = 6200 \mu\text{s}$ und $\Delta t_{SMax} = 6400 \mu\text{s}$. Aufgerundet kann $\Delta t_S = 6500 \mu\text{s}$ angenommen werden. Ein beispielhafter Ausschnitt der Messung kann Abb. 4.10 entnommen werden.

Da 12 Schritte für den Wechsel eines Feldes benötigt werden, kann nun Δt_{W1} bestimmt werden.

$$\Delta t_{W1} = n_S \cdot \Delta t_S \quad (4.80)$$

$$= 12 \cdot 6500 \mu\text{s} \quad (4.81)$$

$$= 81\,000 \mu\text{s} \quad (4.82)$$

$$= 0.081 \text{ s} \quad (4.83)$$

Eine maximale Positionsänderung des Schiebers umfasst fünf Wechsel. So ergibt sich:

$$\Delta t_{W5} = n_W \cdot \Delta t_{W1} \quad (4.84)$$

$$= 5 \cdot 0.081 \text{ s} \quad (4.85)$$

$$= 0.4032 \text{ s} \quad (4.86)$$

Bevor der Schrittmotor rotiert, muss ein gewisses Setup durchgeführt werden, um beispielsweise die Richtung der Rotation einzustellen. Das Zeitintervall Δt_{Setup} gibt die vergangene Zeit zwischen dem Durchbrechen der Lichtschranke und dem Beginn der Rotation des Motors an. In den Messungen lag dieses Zeitintervall zwischen $\Delta t_{SetupMin} = 600 \mu\text{s}$ und $\Delta t_{SetupMax} = 900 \mu\text{s}$. Aufgerundet wird $\Delta t_{Setup} = 1000 \mu\text{s}$ angenommen.

Nun kann das maximal mögliche Zeitintervall für die Sortiermechanik Δt_{Sc} errechnet werden.

$$\Delta t_{Sc} = \Delta t_{Setup} + \Delta t_{W5} \quad (4.87)$$

$$= 1000 \mu\text{s} + 403\,200 \mu\text{s} \quad (4.88)$$

$$= 404\,200 \mu\text{s} \quad (4.89)$$

$$= 0.4042 \text{ s} \quad (4.90)$$

Um die Robustheit des Systems zu erhöhen, wird außerdem ein kleiner Zeitpuffer hinzugefügt. Dieser Zeitpuffer beträgt $\frac{1}{10}$ des Zeitintervalls Δt_{Sc} . Ziel des Zeitpuffers ist es, kleine Unregelmäßigkeiten im Programmablauf abfangen zu können und die Echtzeitfähigkeit des Systems zu garantieren.

$$\Delta t_{Puffer} = \frac{1}{10} \cdot \Delta t_{Sc} \quad (4.91)$$

$$= \frac{1}{10} \cdot 0.4042 \text{ s} \quad (4.92)$$

$$= 0.0404 \text{ s} \quad (4.93)$$

$$(4.94)$$

$$\Delta t_{Ms} = \Delta t_{Sc} + \Delta t_{Puffer} \quad (4.95)$$

$$= 0.4042 \text{ s} + 0.0404 \text{ s} \quad (4.96)$$

$$= 0.4446 \text{ s} \quad (4.97)$$

Das Zeitintervall Δt_{Ms} drückt die maximale Gesamtzeit aus, die ein Testobjekt vom Durchbrechen der Lichtschranke bis zum erreichen des Punktes P_{Ende} benötigen soll. Mithilfe des Zeitpunkts t_{Ende} , an welchen das Testobjekt den Punkt P_{Ende} erreicht, kann nun auch der Zeitpunkt t_{Ls2} errechnet werden, an dem die Lichtschranke durchbrochen werden soll. Da das Durchbrechen der Lichtschranke gleichbedeutend mit dem Starten der mechanischen Sortierung ist, gilt $t_{Sort} = t_{Ls2}$.

$$t_{Ls2} = t_{Ende} - \Delta t_{Ms} \quad (4.98)$$

$$= 2.223 \text{ s} - 0.4446 \text{ s} \quad (4.99)$$

$$= 1.779 \text{ s} = t_{Sort} \quad (4.100)$$

Nun kann die Position der Lichtschranke P_{Ls2} errechnet werden.

$$P_{Ls2} = v_T \cdot t_{Ls2} \quad (4.101)$$

$$= 0.112 \text{ m s}^{-1} \cdot 1.779 \text{ s} \quad (4.102)$$

$$= 0.2 \text{ m} \quad (4.103)$$

Die Lichtschranke der mechanischen Sortierung wird demnach am Punkt $P_{Ls2} = 0.2 \text{ m}$ installiert. Auf diese Weise erreicht das Testobjekt das Ende des Transportbandes quasi im selben Moment, wenn der Schieber die korrekte Position eingenommen hat. Alle Echtzeitbedingungen der mechanischen Sortierung werden mit Sicherheit eingehalten.

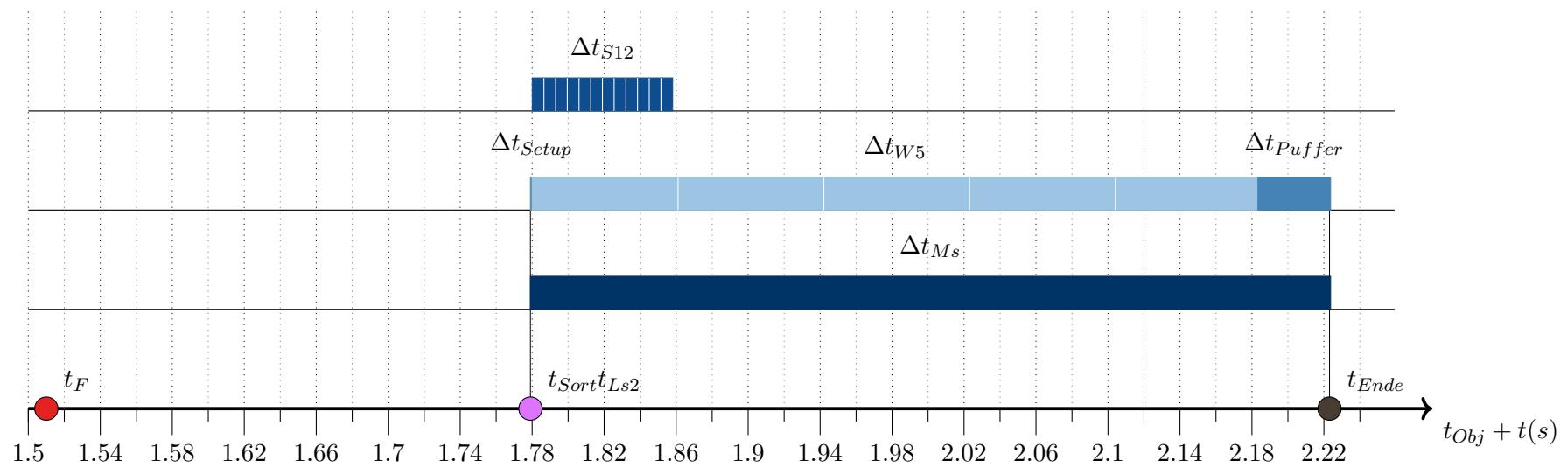


Abbildung 4.11: Zeitverhalten der mechanischen Sortierung

5 Software-Implementierung

Nachdem die Konstruktion des Versuchsaufbaus beschrieben wurde, wird nun die Implementierung der Software des Farbsortierers beleuchtet.

Die Software liest Daten der Sensoren ein, verarbeitet diese und steuert die Aktuatoren. Alle Teilsysteme (Transportband, Detektionsbereich, Mechanische Sortierung) sind mit der Software verbunden und werden von ihr gesteuert.

Um die Ressourcen der Entwicklungsplattform möglichst effizient zu nutzen, wurde die Software in der Programmiersprache C geschrieben. Die Programmiersprache C ermöglicht einen direkten Zugriff auf die Speicheradressen von Variablen und Funktionen. So können im Programmablauf die *Adresszeiger* größerer Daten, anstatt deren Werte, übergeben werden. Aufgrunddessen kann die Verarbeitungszeit des Programms erhöht werden.

Ein weiterer Vorteil ist die Verwendung von C in der gesamten UNIX-Prozesswelt (siehe Abschnitt 2.3). In der Programmiersprache C können Prozesse gezielt erschaffen, kontrolliert und terminiert werden. Diese Eigenschaft hat einen hohen Bedeutungswert für den Erfolg dieser Arbeit.

Da die Programmiersprache C eine hardwarenahe Programmierung erlaubt, bietet sie eine gute Grundlage für anknüpfende Arbeiten, beispielsweise für die Implementierung auf einem Mikrocontroller oder FPGA. Bei der Konzeption der Software wurde bestmöglich auf das Einbinden externer Bibliotheken verzichtet, um eine möglichst einfache Portierung des Systems auf Hardware zu ermöglichen.

In den folgenden Kapiteln werden anfangs die allgemeine Software-Architektur und die Funktionsweisen der einzelnen Komponenten vorgestellt. Anschließend werden die zwei zentralen Algorithmen, die Farberkennung und Sortiersteuerung, im Detail vor gestellt.

5.1 Software Architektur

In diesem Abschnitt werden die einzelnen Bestandteile der Software des Farbsortier-Demonstrators behandelt. Dabei werden die einzelnen Komponenten vorgestellt und deren Funktion im Gesamtablauf erläutert.

Nachfolgend wird der zeitliche Ablauf des Programms beschrieben. Dabei wird auf die einzelnen Schritte eingegangen, die durchgeführt werden müssen, um eine erfolgreiche Sortierung der Testobjekte zu realisieren.

5.1.1 Software Komponenten

Das Farberkennungsprogramm besteht aus verschiedenen Software Komponenten. Diese sind gemäß ihrer Funktion in einzelne Dateien aufgeteilt. Zu diesen Software Komponenten gehören das Hauptprogramm *main.c*, die Farberkennung *color_detection.c*, die Sortiersteuerung *sorting_control.c* sowie verschiedene Hilfsfunktionen *utils.c*. Zentrale Konfigurations-Variablen werden als preprozessor Direktive in einer Konfigurationsdatei *config.h* definiert. Dies ermöglicht eine einfache und zentrale Anpassung der Programmparameter.

Eine Übersicht der Software Architektur kann dem Blockschaltbild in Abb. 5.1 entnommen werden.

Obwohl versucht wurde, auf externe Bibliotheken zu verzichten, war es dennoch sinnvoll die Bibliothek *pigpio* sowie das Programm *raspistill* zu verwenden. Die Bibliothek *pigpio* ist ebenfalls in C geschrieben und existiert, um die GPIO-Steckerleiste des Raspberry Pi's zu kontrollieren. Mithilfe des Programms *raspistill* wird die Kamera des Raspberry Pi's angesteuert, um Bilddaten aufzunehmen und zu speichern.

5.1.1.1 Hauptprogramm

Das Hauptprogramm findet sich in der Datei *main.c*. Es übernimmt die Initialisierungs- und Terminierungsroutine. In diesen wird die Schnittstelle zu den GPIO-Pins mithilfe der *pigpio*-Bibliothek geöffnet und geschlossen. Die 12 V-Leuchtdiode und der Gleichstrommotor werden direkt über das Hauptprogramm kontrolliert. In der Initialisierungsroutine wird außerdem mittels *raspistill* die Aufnahme des Vergleichsbildes für das Differenzbildverfahren behandelt.

Das Hauptprogramm startet und verwaltet die drei Kindprozesse *Farberkennung*, *Sortierkontrolle* und *raspistill*.

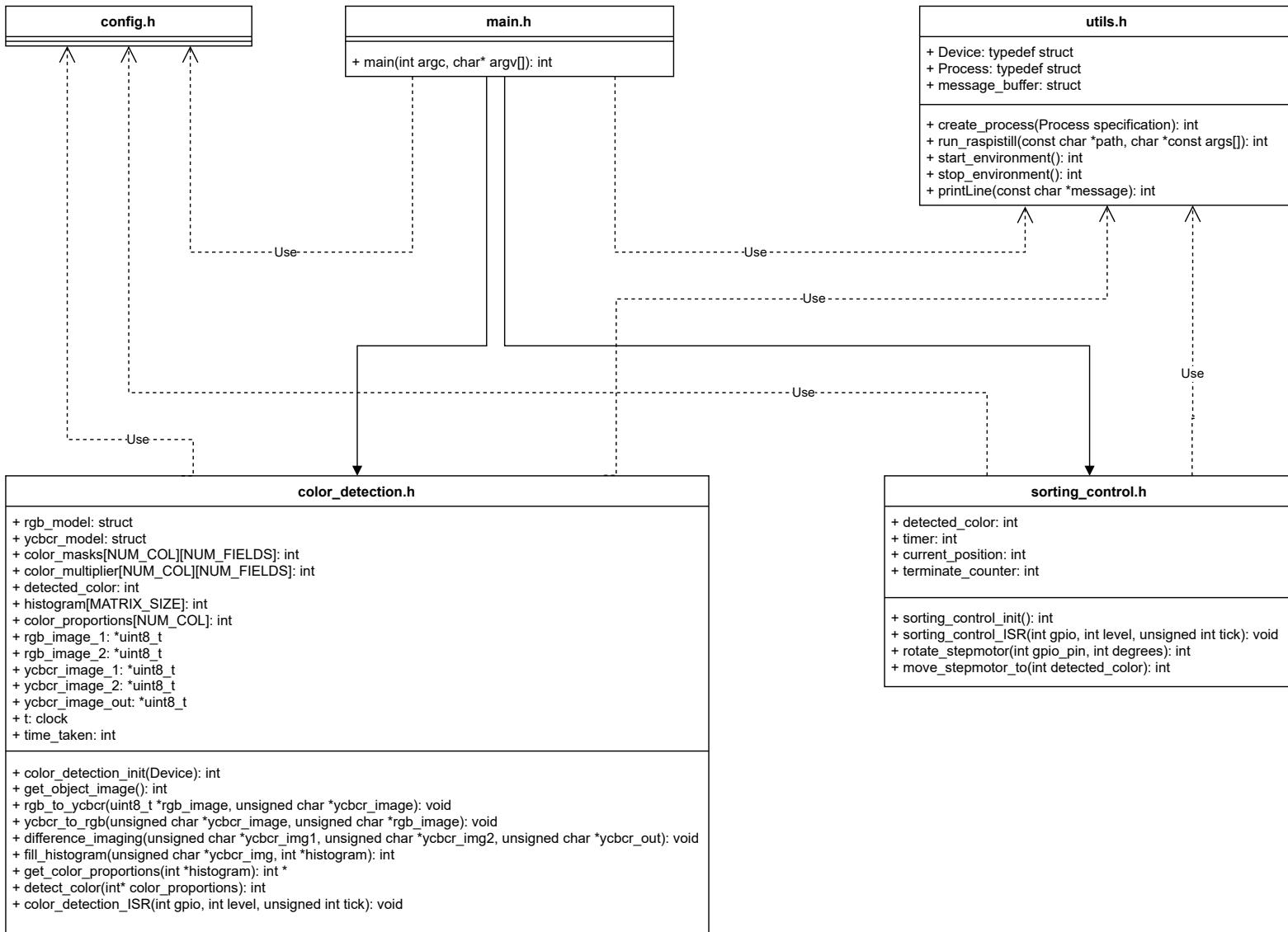


Abbildung 5.1: Blockschaltbild des Programms

5.1.1.2 Farberkennung

Die Farberkennung ist für die Farbdetektion der Testobjekte verantwortlich. Sie findet sich in der Datei *color_detection.c*. Sie ist ein Kindprozess vom Hauptprogramm und wird von diesem verwaltet.

Der Detektionsbereich liefert Bilddaten, welche die Farberkennung als Eingangsdaten nutzt. Innerhalb der Farberkennung werden diese Bilddaten in verschiedenen Teilschritten verarbeitet und transformiert.

Die chronologische Abfolge der Teilschritte sind:

1. Initialisierung
2. Aufruf der Interrupt Service Routine
3. Einlesen der Bilddaten
4. Farbraumkonvertierung
5. Differenzbildverfahren
6. Bilden des 2D-Histogramms
7. Berechnen der Farbanteile
8. Farbdetektion mittels Vergleich der Farbanteile
9. Kommunizieren der Farbinformation

Um möglichst wenig Ressourcen zu verbrauchen, wartet die Farberkennung auf das Lichtschrankensignal des Detektionsbereichs. Erst nachdem sie dieses Signal mittels Interrupt erhalten hat, wird sie aktiv. Andernfalls befindet sich der Prozess im Status *blockiert* und verbraucht keine Prozessorleistung.

5.1.1.3 Sortiersteuerung

Die Sortiersteuerung ist der Teil in der Software, der für die Sortierung der Testobjekte zuständig ist. Sie ist ebenfalls ein Kindprozess vom Hauptprogramm und findet sich in der Datei *sorting_control.c*.

Sie liest die detektierte Farbe aus der Message Queue und steuert daraufhin den Schrittmotor an, damit die Testobjekte in die korrekten Sortierfächer geleitet werden.

Der Prozessablauf unterteilt sich ebenfalls in verschiedene Teilschritte:

1. Initialisierung
2. (Dekrementieren des Terminierungs-Counters)
3. Aufruf der Interrupt Service Routine
4. Einlesen der detektierten Farbe
5. Vorbereitung der Motordrehung

6. Durchführen der Motordrehung

Die Sortiersteuerung enthält außerdem einen sogenannten Terminierungs-Counter, welcher beständig dekrementiert. Bei Ablauf des Counters wird das Hauptprogramm benachrichtigt, um die Terminierung des Programms einzuleiten.

Wie auch die Farberkennung, reagiert die Sortiersteuerung auf das Signal einer Lichtschranke bevor sie aktiv wird. Abgesehen von kurzen Unterbrechungen, die dem Dekrementieren des Counters dienen, befindet sich die Sortiersteuerung die restliche Zeit ebenfalls im Prozesszustand *blockiert*.

5.1.1.4 Hilfsfunktionen

Zu den Hilfsfunktionen gehören Funktionen, die von mehreren Software-Komponenten genutzt werden. Sie sind in der Datei *util.c* gesammelt. Zu diesen gehören beispielsweise eine modifizierte Print-Funktion oder die Ermittlung der Systemzeit für Latenzmessungen.

5.1.2 Programmablauf

Der Programmablauf lässt sich in fünf übergeordnete Zustände gemäß Tabelle 5.1 unterteilen.

Nr.	Zustand
1	Initialisierung
2	Warten auf Testobjekt
3	Farberkennung
4	Sortiersteuerung
5	Terminierung

Tabelle 5.1: Zustände des Programms

Abb. 5.2 zeigt die Zustandsmaschine des Programms, in welcher die Zustandswechsel deutlich werden. Im Sequenzdiagramm Abb. 5.3 sind die konkreten Verhalten der einzelnen Software-Komponenten ersichtlich, die während des Programmablaufs auftreten.

5.1.2.1 Zustand 1: Initialisierung

Die 1. Zustand *Initialisierung* wird von der `start_environment()`-Funktion des Hauptprogramms gestartet. In dieser werden die GPIO-Pins mithilfe der `pigpio`-Bibliothek initialisiert und anschließend die 12 V-Leuchtdiode sowie der Gleichstrommotor in Betrieb genommen.

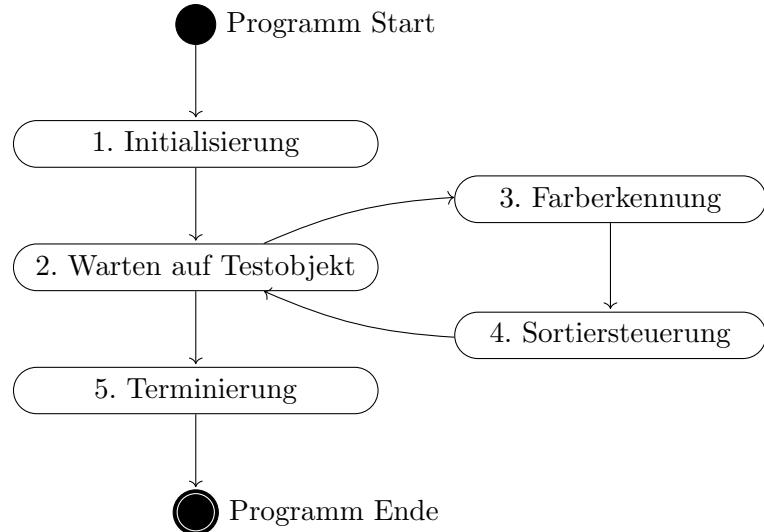


Abbildung 5.2: Zustandsmaschine des Programms

Um die Kommunikation zwischen den verschiedenen Prozessen zu ermöglichen, wird eine *Message Queue* erstellt (siehe Abschnitt 2.3.3). Deren Identifikationsnummer wird vom Hauptprogramm an die Kindprozesse vererbt, sodass alle verwendeten Prozesse auf diese Message Queue zugreifen können.

Mittels des Programms `raspistill` wird eine einzelne Aufnahme vom ausgeleuchteten und leeren Detektionsbereich gemacht. Diese dient im 3. Zustand *Farberkennung* als Vergleichsbild zu den Aufnahmen der Testobjekte und ermöglicht das Differenzbildverfahren.

Zum Abschluss der Initialisierung werden die notwendigen Kindprozesse des Hauptprogramms erstellt.

Das Programm `raspistill` wird erneut gestartet, dieses Mal jedoch im `timelapse`-Modus. Auf diese Weise werden periodisch Aufnahmen des Detektionsbereichs erstellt, die ebenfalls im 3. Zustand genutzt werden. Der Kindprozess `raspistill` läuft über die gesamte Laufzeit des Programms und wird später vom Elternprozess terminiert.

Ein weiterer Kindprozess, der innerhalb des 1. Zustands gestartet wird, ist der Prozess *Farberkennung*. Dieser Prozess übernimmt die Aufgaben des gleichnamigen 3. Zustands. Nachdem der Farberkennungs-Prozess gestartet wurde, durchläuft er seine eigene prozess-spezifische Initialisierungs-Routine. Während dieser Initialisierung wird das Vergleichsbild geladen und Speicher alloziert. Anschließend definiert er seine entsprechende Interrupt Service Routine und aktiviert den Interrupt-Handler. Der Interrupt-Handler achtet nun auf den GPIO-Pin, welcher mit der Lichtschranke des Detektionsbereichs verbunden ist.

Der *Sortiersteuerungs*-Prozess ist der dritte Kindprozess, der vom Hauptprogramm gestartet wird. Er ist an den 4. Zustand *Sortiersteuerung* geknüpft. Das Verhalten dieses Prozesses nach der Erstellung gleicht dem des Farberkennungs-Prozesses. Er durchläuft

ebenfalls eine eigene prozess-spezifische Initialisierung, die den Terminierungs-Counter mit einem Startwert initialisiert, die Interrupt Service Routine setzt und den Interrupt-Handler aktiviert. Der Interrupt-Handler der Sortiersteuerung achtet dabei auf den GPIO-Pin, welcher mit der Lichtschranke der mechanischen Sortierung verbunden ist.

5.1.2.2 Zustand 2: Warten auf Testobjekt

Das Hauptprogramm befindet sich nach Abhandlung der Initialisierung im Zustand *blockiert*. Anders als die beiden Kindprozesse *Farberkennung* und *Sortiersteuerung*, wartet es jedoch nicht auf das Signal eines GPIO-Pins, sondern auf die Terminierung des Prozesses *Sortiersteuerung*. Solange dieser Prozess läuft, setzt es seinen Programmablauf nicht fort.

Der Prozess *Farberkennung* wartet nach seiner Initialisierungs-Routine auf einen Interrupt und ist währenddessen im Zustand *blockiert*. Sollte ein Testobjekt die Lichtschranke des Detektionsbereichs durchbrechen, wird der Prozess *Farberkennung* aktiv und das Programm wechselt in den 3. Zustand *Farberkennung*.

Auch der Prozess *Sortiersteuerung* wartet auf einen Interrupt und ist für die meiste Zeit ebenfalls im Zustand *blockiert*. Anders als der *Farberkennungs*-Prozess wird er jedoch jede Sekunde für einen kurzen Moment wach, um den Terminierungs-Counter zu dekrementieren. Sollte der Counter ablaufen, terminiert der Prozess *Sortiersteuerung*. Über diese Terminierung wird das Hauptprogramm informiert, das daraufhin seinen Programmablauf fortsetzt. In diesem Fall wechselt das Programm anschließend in den 5. Zustand *Terminierung*.

Da die Lichtschranke der mechanischen Sortierung nicht durchbrochen werden kann, ohne vorher den Farberkennungs-Algorithmus zu aktivieren, ist ein Wechsel vom Zustand 2 in den Zustand 4 nicht möglich.

5.1.2.3 Zustand 3: Farberkennung

Durch das Durchbrechen der Lichtschranke des Detektionsbereichs wird der Farberkennungs-Prozess aktiv. Er verarbeitet die Bilddaten der Kamera und errechnet daraus die Farbe des transportierten Testobjekts. Die detektierte Farbe schreibt er in eine Message Queue. Anschließend wechselt das Programm in den 4. Zustand *Sortiersteuerung*.

Der 3. Zustand *Farberkennung* und der Prozess der Farberkennung sind quasi identisch. Aus diesem Grund wird auf Abschnitt 5.2 verwiesen, wo der gesamte Ablauf der Farberkennung ausführlich beschrieben wird.

5.1.2.4 Zustand 4: Sortiersteuerung

Sobald die Lichtschranke der mechanischen Sortierung durchbrochen wurde, wird die Sortiersteuerung aktiv. Sie liest die detektierte Farbe aus der Message Queue und

steuert den Sortiermechanismus entsprechend dieser Information. Außerdem wird der Terminierungs-Counter bei jedem Interrupt zurück auf einen Startwert gesetzt. Nach Abschluss der Sortierung wechselt das Programm zurück in den 2. Zustand *Warten auf Testobjekt*.

Da sich der 4. Zustand *Sortiersteuerung* ebenfalls mit dem gleichnamigen Prozess *Sortiersteuerung* gleicht, wird auch hier auf eine ausführliche Beschreibung des 4. Zustands verzichtet. Diese findet sich im Abschnitt 5.3.

5.1.2.5 Zustand 5: Terminierung

Erhält das Hauptprogramm die Information über die Terminierung des Prozesses *Sortiersteuerung*, ruft es die Terminierungs-Routine auf. Diese beendet die Kindprozesse *Farberkennung* und *raspistill*. Es werden sämtliche Aktuatoren ausgeschaltet und die verwendeten GPIO-Pins geschlossen. Des Weiteren wird die Message Queue gelöscht und Speicher freigegeben.

Nach Ablauf der Terminierungs-Routine beendet sich das Hauptprogramm.

Anmerkung zum Zustandsmodell Da mehrere Prozesse laufen, können zeitgleich auch mehrere Zustände aktiv sein. So kann beispielsweise die Sortiersteuerung den Schrittmotor steuern, während die Farberkennung ein nächstes Testobjekt verarbeitet. Diese Eigenschaft des Programms wurde gezielt so implementiert und ist für einen robusten Ablauf notwendig. Um die verschiedenen Zustände des Programms anschaulich erklären zu können, wurde auf die Erläuterung parallel arbeitender Zustände verzichtet.

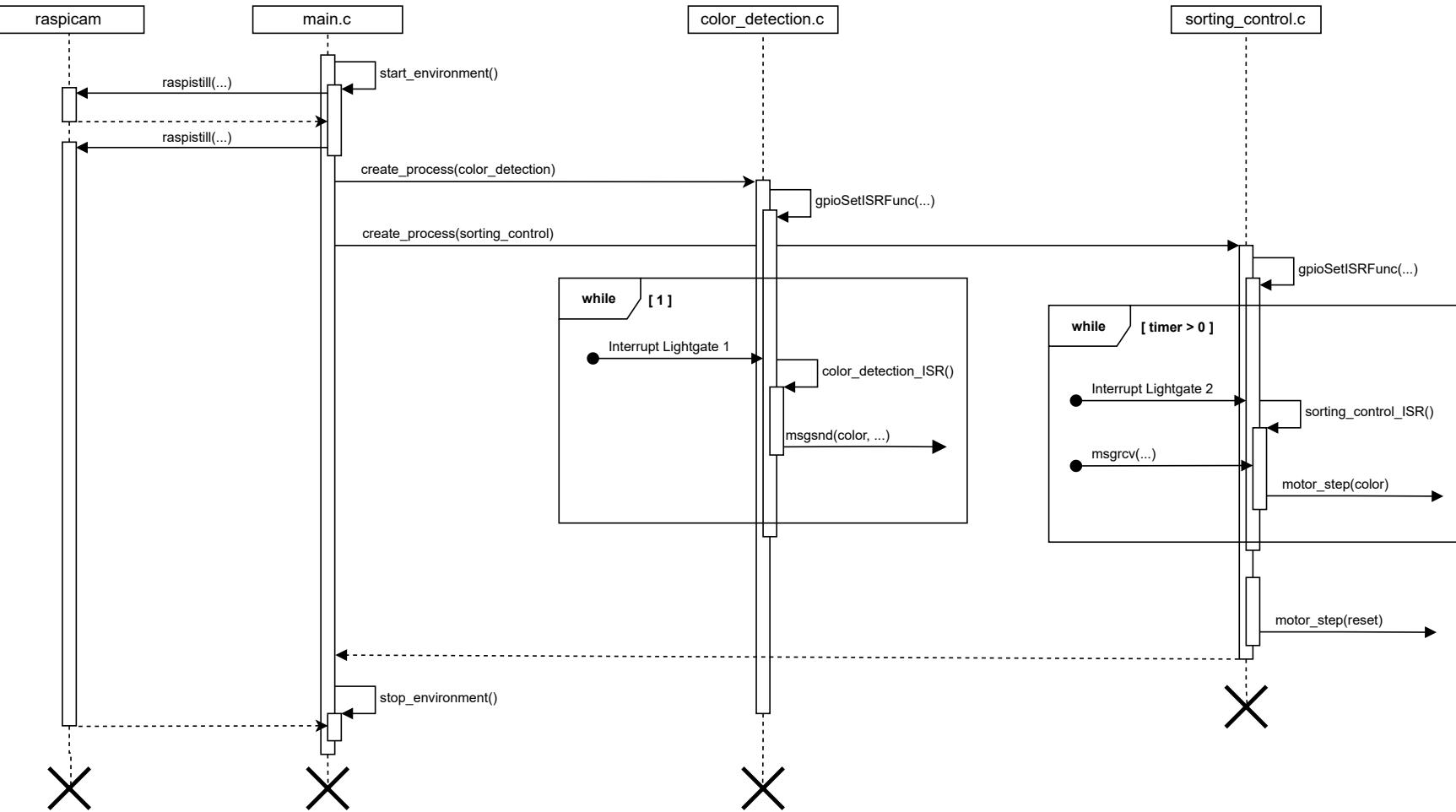


Abbildung 5.3: Sequenzdiagramm des Programms

5.2 Farberkennung

Der Farberkennungs-Algorithmus ist das Kernstück des Farbsortier-Demonstrators. In diesem Algorithmus wird aus den eingehenden Bilddaten des Detektionsbereichs das transportierte Testobjekt extrahiert und anschließend dessen Farbe erkannt. Diese Farbe wird dann der Sortiermechanik für die weitere Verarbeitung zur Verfügung gestellt.

5.2.1 Herleitung des Farberkennungs-Algorithmus

Die entscheidende Fragestellung bei der Implementierung einer Farberkennung ist, auf welche Weise Farben gruppiert und voneinander abgegrenzt werden sollen. Erste Versuche unter dem Einsatz von Schwellwerten erzielten keine zufriedenstellenden Ergebnisse, da sich die Pixelwerte der unterschiedlichen Farben teilweise stark überlappend auf der CbCr-Farbskala¹⁷ verteilen. So teilen sich beispielsweise braune, rote und orangene Objekte größere Bereiche im Zentrum der CbCr-Farbskala. Eine klare Abgrenzung der Zielfarben auf der CbCr-Farbskala ist deshalb so nicht möglich.

In der Folge musste eine eigene Methode entwickelt werden, um ähnlich gefärbte Testobjekte sicher voneinander unterscheiden zu können.

5.2.1.1 Histogrammbildung

Um die charakteristischen Bereiche der einzelnen Farben auf der CbCr-Farbskala empirisch ermitteln zu können, wurde eine Messung durchgeführt. In dieser wurden je 100 Aufnahmen von Messobjekten der sechs Zielfarben erhoben. Anschließend wurde jede Aufnahme in das YCbCr-Farbmodell konvertiert, wobei der Luminanzwert Y in der weiteren Verarbeitung nicht mehr berücksichtigt wurde. Nach der Farbraumkonvertierung wurde das Objekt mittels Differenzbildverfahrens vom Hintergrund extrahiert, um der anschließenden Analyse ausschließlich Farbwerte des Messobjekts zu übergeben.

Nun wurde die Verteilung der Pixelwerte auf der CbCr-Farbskala analysiert. Die CbCr-Farbskala wurde dabei in ein Raster mit zehn mal zehn Feldern unterteilt (siehe Abb. 5.4). Die Pixel eines Messobjekts wurden entsprechend ihrer Cb- und Cr-Werten in die jeweiligen Felder des Rasters eingesortiert. Die akkumulierten Werte der einzelnen Felder bildeten so eine Häufigkeitsverteilung aller Farbwerte des Messobjekts auf der CbCr-Farbskala. Diese Häufigkeitsverteilung entspricht einem zweidimensionalen Histogramm des Messobjekts auf der CbCr-Farbskala. Aus den Histogrammen aller Messobjekte einer Farbe wurde anschließend ein Histogramm mit der durchschnittlichen Häufigkeitsverteilung auf der CbCr-Farbskala errechnet.

Das Ergebnis der Messung waren demnach sechs Histogramme, die das arithmetische Mittel der Häufigkeitsverteilung der sechs Farben repräsentieren. Diese sind in Abb. 5.5

¹⁷CbCr-Farbskala meint im Folgenden einen zweidimensionalen Raum mit Cb auf der x-Achse und Cr auf der y-Achse

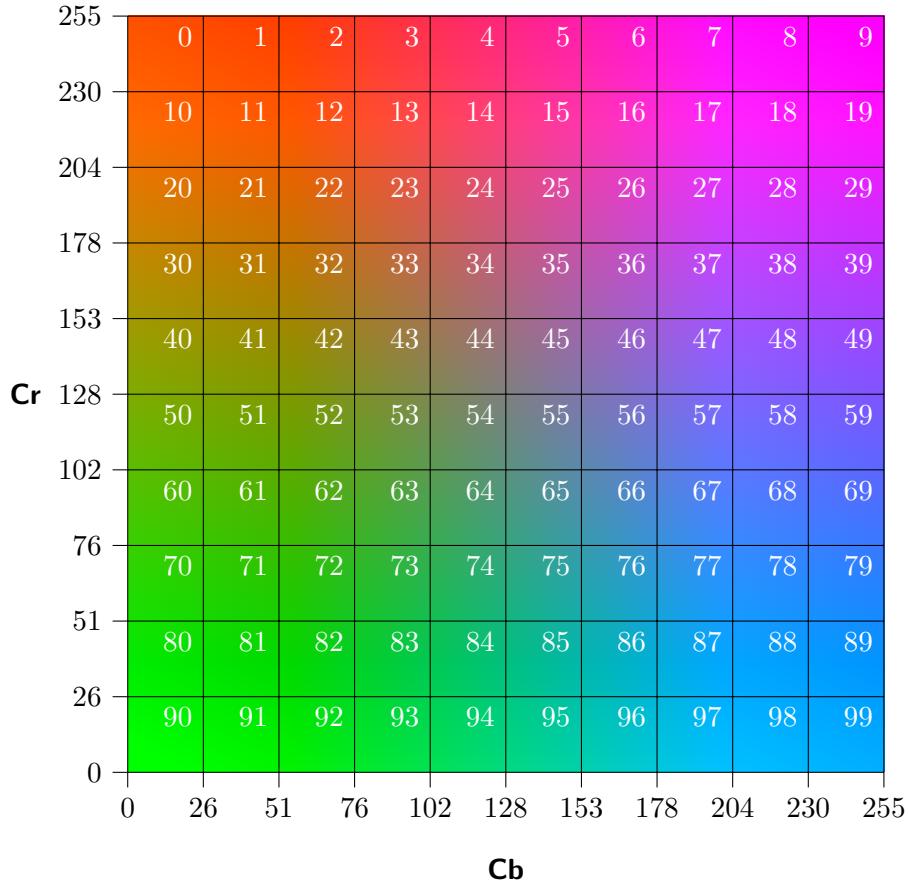


Abbildung 5.4: CbCr-Farbskala mit Indizes der Histogrammfelder

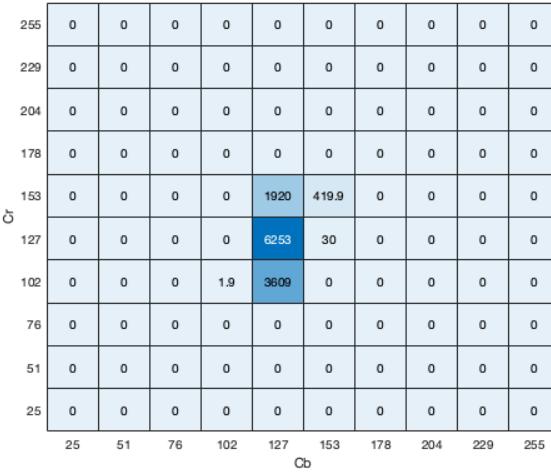
zu sehen. Mithilfe dieser Histogramme konnten die sechs Zielfarben nun klar definiert und miteinander verglichen werden.

5.2.1.2 Definition der Farbmasken

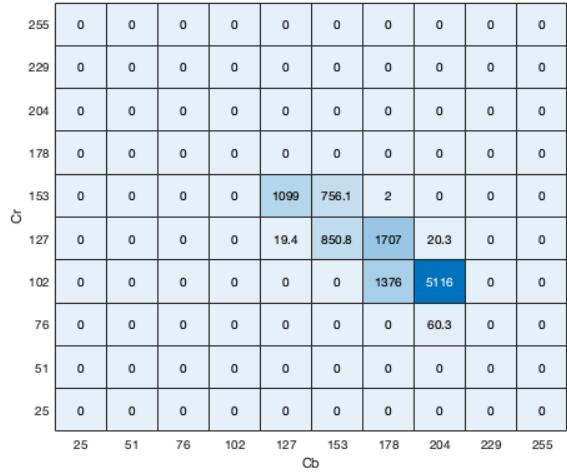
Die Histogrammfelder einer Farbe, die eine besonders hohe Konzentration von Pixelwerten aufweisen, werden als *charakteristisch* für diese Farbe angesehen. Als *charakteristische Felder* gelten die n_H größten Felder einer Zielfarbe, deren akkumulierten Werte mindestens 97 % der gesamten Anzahl an Pixeln ausmachen. Je nach der Verteilung auf der CbCr-Farbskala ergeben sich so unterschiedliche Werte für n_H der jeweiligen Farben (siehe Tabelle 5.2).

Folglich kann eine Zielfarbe mithilfe weniger Felder des Histogramms nahezu vollständig beschrieben werden. Dies wird sich zunutze gemacht, um die einzelnen Zielfarben möglichst kurz, aber eindeutig zu definieren.

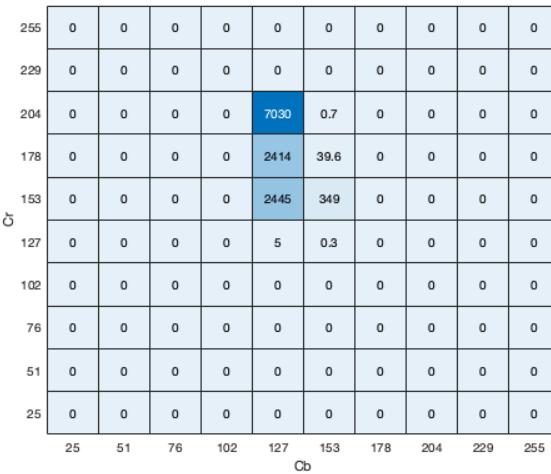
Aus diesem Grund wurden sogenannte *Farbmasken* aus den n_H charakteristischen Feldern der jeweiligen Zielfarbe gebildet. Die Farbmaske einer Zielfarbe beinhaltet die Indizes der Histogrammfelder, auf welche sich die meisten Pixel verteilen. Die Farb-



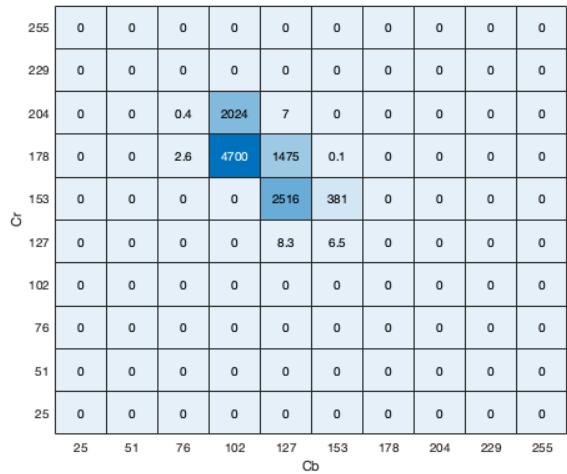
(a) Grünes Objekt



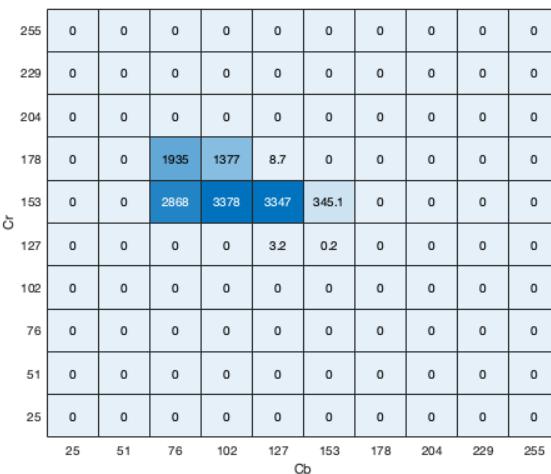
(b) Blaues Objekt



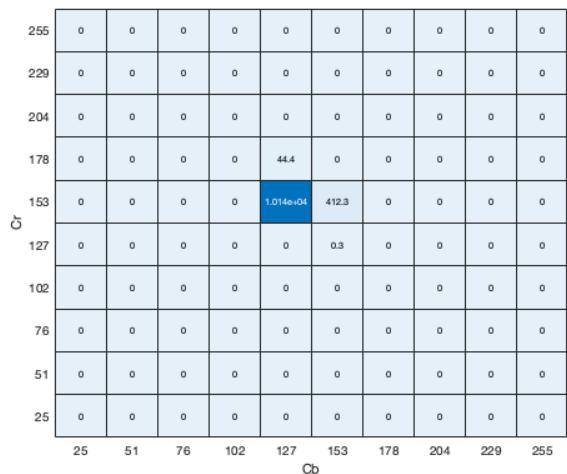
(c) Rotes Objekt



(d) Orangenes Objekt



(e) Gelbes Objekt



(f) Braunes Objekt

Abbildung 5.5: Durchschnittliche Histogramme der Messobjekte auf CbCr-Farbskala

Farbe	n_H	Farbmaske	Farbmultiplikatoren
Grün	4	44, 45, 54, 64	1.9, 0.4, 6.2, 3.6
Blau	6	44, 45, 55, 56, 66, 67	1, 0.7, 0.8, 1.7, 1.3, 5.1
Rot	4	24, 34, 44, 45	7, 2.4, 2.4, 0.3
Orange	5	23, 33, 34, 44, 45	2, 4.7, 1.4, 2.5, 0.3
Gelb	6	32, 33, 42, 43, 44, 45	1.9, 1.3, 2.8, 3.3, 3.3, 0.3
Braun	2	44, 45	6.1, 0.4

Tabelle 5.2: Farbmasken und Farbmultiplikatoren der Zielfarben

masken der einzelnen Zielfarben sind aus Tabelle 5.2 ersichtlich.

Als Vergleich dazu dient Abb. 5.5. Dort werden Histogrammfelder, die eine hohe Dichte an Pixeln aufweisen, dunkler gefärbt. Diese bilden gleichzeitig die Farbmasken der jeweiligen Zielfarben.

5.2.1.3 Einführen der Farbmultiplikatoren

Für die Klassifikation der Zielfarben sind jedoch nicht nur die Indizes der Histogrammfelder interessant. Auch die Gewichtungen der einzelnen Histogrammfelder sind von entscheidender Bedeutung.

Dies kann am Beispiel der Zielfarbe Braun nachvollzogen werden. Braun wird einzig durch die beiden Histogrammfelder mit den Indizes 44 und 45 definiert. Diese beiden Indizes sind jedoch ebenso in den Farbmasken aller weiteren fünf Zielfarben vertreten. Wie später in Abschnitt 5.2.2.7 erklärt wird, würde ein braunes Objekt vom Algorithmus nie als solches erkannt werden, wenn die Farberkennung einzig auf den Werten der Farbmasken basieren würde.

Um dieses Problem zu beheben, werden sogenannte *Farbmultiplikatoren* verwendet. Diese repräsentieren die normierten Gewichtungen der Felder der Farbmasken. Jedem Wert einer Farbmaske ist genau ein Farbmultiplikator zugewiesen. Im weiteren Verlauf sind Farbmaske und Farbmultiplikatoren nicht voneinander zu trennen. Beide sind für die Definition einer Farbe auf der CbCr-Farbskala erforderlich.

Die Farbmaske beschreibt, welche Bereiche der CbCr-Farbskala eine Farbe definieren, während die Farbmultiplikatoren beschreiben, wie wichtig diese Bereiche für die Definition sind.

Auf diese Art kann auch das oben beschriebene Problem gelöst werden. Der braune Farbmultiplikator enthält für das Feld 44 einen vergleichsweise sehr hohen Wert, wohingegen die Farbmultiplikatoren der anderen Farben für das selbe Feld einen eher niedrigen Wert aufweisen. Wird ein Bild analysiert, das viele Werte im Feld 44 vorortet, führen diese so zu einem höheren Wert für die Farbe Braun, als für die anderen Farben.

Mit der Bildung der Histogramme gelingt es, analysierte Farbwerte auf der CbCr-Farbskala zu gruppieren. Die Farbmasken ermöglichen in Kombination mit den Farbmultiplikatoren eine kurze, aber sehr aussagekräftige Definition einer Farbe auf der genannten Farbskala.

Diese Hilfskonstrukte erlauben nun die Implementierung des eigentlichen Farberkennungs-Algorithmus.

5.2.2 Implementierung des Farberkennungs-Algorithmus

Der gesamte Farberkennungs-Algorithmus findet sich in der Datei *color_detection.c*. Er besteht aus verschiedenen Teilschritten, die, abgesehen von der Initialisierungs-Routine, für jedes eingespeiste Objekt erneut aufgerufen werden.

Ziel des Farberkennungs-Algorithmus ist es, die Farbe eines transportierten Testobjekts zu identifizieren und diese Information an den Prozess *Sortiersteuerung* zu übermitteln.

Der Algorithmus ist als eigener Kindprozess des Hauptprogramms implementiert, wodurch er unabhängig vom Verhalten der restlichen Prozesse arbeiten kann. Um die Auslastung des Prozessors so gering wie möglich zu halten, ist der Prozess nur für den Zeitraum der Farberkennung aktiv. Für die restliche Zeit befindet er sich im Zustand *blockiert*.

Er ist eng verbunden mit der Komponente *Detektionsbereich* des Versuchsaufbaus. Die Kamera des Detektionsbereichs liefert ihm die benötigten Bilddaten der Testobjekte. Außerdem sendet die Lichtschranke des Detektionsbereichs ein Interrupt Signal, durch welches der Prozess aktiviert wird.

Im Folgenden werden die einzelnen Teilschritte des Farberkennungs-Algorithmus vorgestellt und deren Implementierung erläutert. Zur besseren Übersicht kann Abb. 5.6 die Struktur des Algorithmus entnommen werden.

5.2.2.1 Initialisierung

Wie eingangs erwähnt, wird die Initialisierung nur einmal nach der Erstellung des Prozesses aufgerufen. In ihr wird der spätere Programmablauf vorbereitet.

Anfangs liest der Algorithmus das *Vergleichsbild* vom Detektionsbereich ein (siehe Abb. 5.7). Dieses ist notwendig, um das später durchgeführte Differenzbildverfahren zu ermöglichen.

Danach aktiviert der Algorithmus einen Interrupt Handler. Dieser wird so konfiguriert, dass er bei steigender Flanke des Lichtschrankensignals reagiert. Trifft dieses Ereignis ein, wird die Interrupt Service Route aufgerufen, die den weiteren Ablauf der Farberkennung startet.

Anschließend wird der Prozess bis zum Signal der Lichtschranke inaktiv.

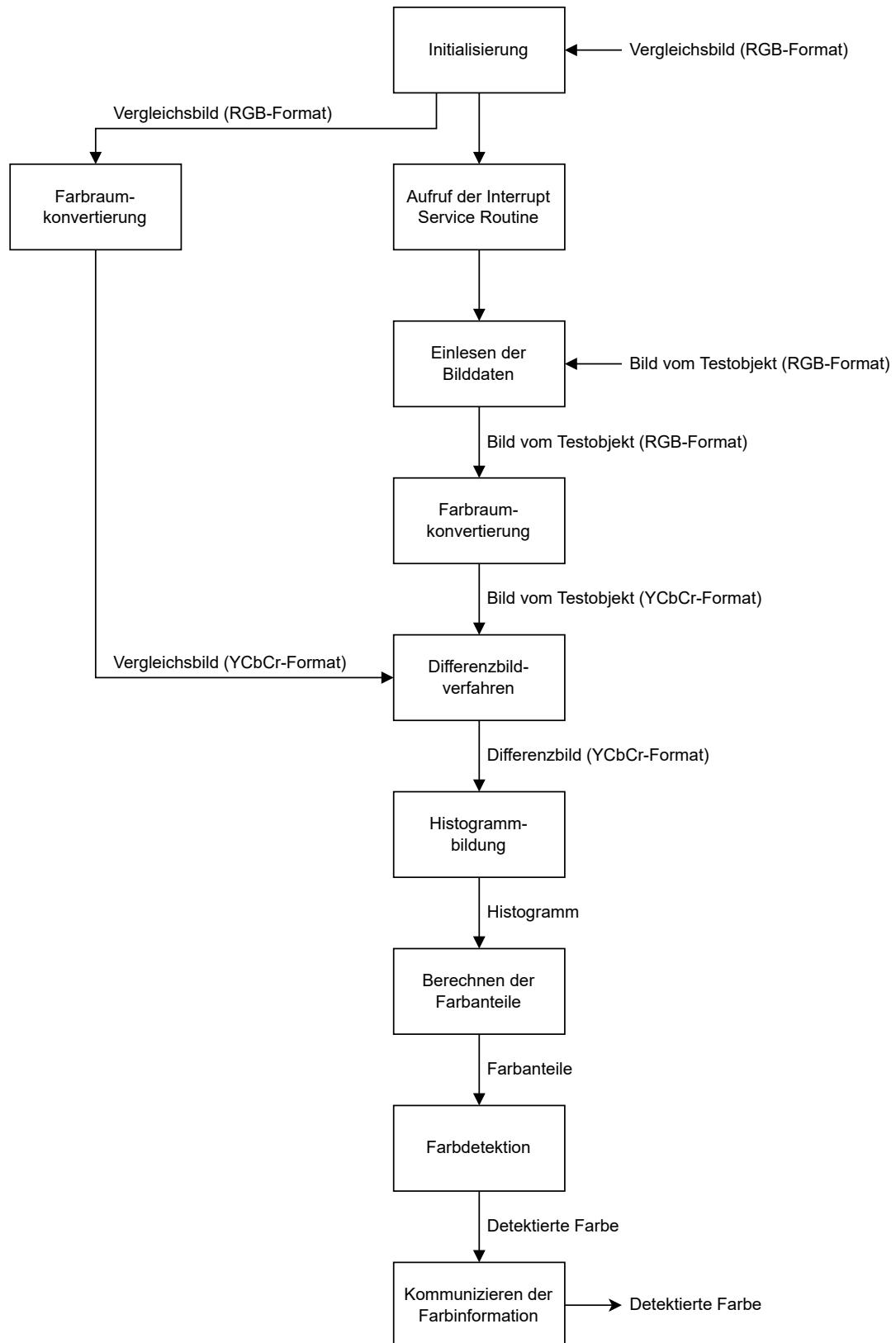


Abbildung 5.6: Ablaufdiagramm Farberkennungs-Algorithmus



Abbildung 5.7: Vergleichsbild vom Detektionsbereich

5.2.2.2 Aufruf der Interrupt Service Routine

Sobald der Interrupt Handler ein Interrupt Signal der Lichtschranke empfängt, erwacht der Prozess und die Interrupt Service Routine wird aufgerufen. Die Interrupt Service Routine startet den eigentlichen Ablauf der Farberkennung.

Wie in Abschnitt 4.2.3.2 berechnet, muss der Farberkennungs-Algorithmus die einge henden Bilddaten für den darauffolgenden Zeitintervall von $\Delta t_P = 0.84\text{ s}$ analysieren. Es kann passieren, dass die erste Aufnahme des Detektionsbereichs noch keine Bild daten des Testobjekts enthält. In diesem Fall werden die Teilschritte *Einlesen der Bilddatei* und *Differenzbildverfahren* innerhalb des Zeitintervalls Δt_P bis zur Objekt detektion wiederholt aufgerufen. Wurde nach Ablauf von Δt_P kein Objekt detektiert, terminiert die Interrupt Service Routine und das Programm wird erneut inaktiv.

Entprellen des Lichtschrankensignals Da die eingesetzte Lichtschranke ohne Polarisationsfilter arbeitet, springt das empfangene Signal gelegentlich (siehe Abschnitt 4.2.2.2). So können beim Durchlauf eines Testobjekts mehrere Interrupt Signale gesendet werden, die die Interrupt Service Routine wiederum mehrmals aufrufen. Da der Algorithmus auf einem Raspberry Pi und nicht auf einem Mikrocontroller im plementiert ist, können einzelne Interrupts nicht vom Benutzer gesperrt werden. Aus diesem Grund ist es notwendig das Lichtschrankensignal im Algorithmus zu *entprellen*. Mit der Entprellung des Lichtschrankensignals ist garantiert, dass die weiteren Teilschritte zur Farberkennung nicht öfter als nötig aufgerufen werden. Um diese Ent prellung zu implementieren, wurde die Methode der Zeitmessung verwendet. Nach dem Starten der Interrupt Service Routine wird der Zeitpunkt des Interrupt Aufrufs anhand der momentanen Systemzeit gemessen. Anschließend wird der Zeitpunkt des aktuellen Interrupts mit dem Zeitpunkt des letzten Interrupts verglichen. Sollte die Differenz der beiden Zeiten einen Schwellwert unterschreiten, beendet sich die Interrupt Service Rou

tine. Nur bei ausreichend großem zeitlichen Abstand, werden die folgenden nächsten Schritte durchlaufen.

5.2.2.3 Einlesen der Bilddaten

Im ersten Schritt nach dem Start der Interrupt Service Routine muss die aktuelle Aufnahme des Detektionsbereichs eingelesen werden. Die Aufnahmen des Detektionsbereichs werden vom Programm `raspistill` periodisch mit dem Zeitintervall $\Delta t_A = 0.84\text{s}$ erhoben und liegen im RGB-Farbraum vor. Abb. 5.8 zeigt eine beispielhafte Aufnahme eines Testobjekts im Detektionsbereich.

Zur Einsparung von Rechenzeit wird der Farbraumkonvertierung nur ein Zeiger auf die Adresse des ersten Pixels übergeben.



Abbildung 5.8: Bild eines Testobjekts

5.2.2.4 Farbraumkonvertierung

Die Farbraumkonvertierung erhält das eingelesene Bild vom Detektionsbereich. Die Übergabe der Farbinformationen erfolgt über einen Adresszeiger auf den Speicherbereich des Bildes.

In diesem Teil der Datenverarbeitung werden die Pixelwerte vom RGB-Farbmodell in das YCbCr-Farbmodell konvertiert. Die Berechnungsvorschrift, nach welcher dies geschieht, findet sich in Abschnitt 2.1.2.

5.2.2.5 Differenzbildverfahren

Beim Differenzbildverfahren wird das aktuell eingelesene Bild vom Detektionsbereich mit dem Vergleichsbild aus der Initialisierungs-Routine verglichen. Aus dem Vergleich beider Bilder resultiert ein neu generiertes Ausgangsbild.

Durch das Differenzbildverfahren wird eine Objekterkennung realisiert. Nach Abschluss des Differenzbildverfahrens weiß der Farberkennungs-Algorithmus, ob die aktuellen Bilddaten relevante Informationen enthalten oder nicht. Sollte dies nicht der Fall sein, kann der Algorithmus die Farberkennung an diesem Punkt abbrechen und Ressourcen sparen.

Kann im aktuellen Bild ein Objekt detektiert werden, dient das Differenzbildverfahren der Datenvorverarbeitung. Durch die Trennung von Testobjekt und Hintergrund müssen nachfolgende Bestandteile des Farberkennungs-Algorithmus nur Daten, die eine Farbinformation enthalten, verarbeiten. Auf diese Weise werden ebenfalls Ressourcen und Rechenzeit gespart.

Das Differenzbildverfahren arbeitet dabei wie folgt. Alle Pixel des aktuellen Bildes werden von den jeweiligen Pixeln des Vergleichsbildes, die an der selben Position liegen, subtrahiert. Die Differenzen dieser Pixelpaare werden anschließend gegen einen Schwellwert geprüft. Dieser Schwellwert dient dazu, kleinere Ungleichheiten, wie etwa eine unterschiedliche Struktur des Transportbandes, herauszufiltern. Unterschreitet die Differenz zweier Pixel den Schwellwert, bedeutet dies, dass sich die beiden Pixel ähnlich sind. Ähnliche Pixel werden als Hintergrund interpretiert und im Ausgangsbild mit dem Wert 0 gesetzt. Überschreitet die Differenz den Schwellwert, deutet dies auf die Präsenz eines Testobjekts hin. In diesem Fall wird der Farbwert des aktuellen Bildes in die äquivalente Position des Ausgangsbildes übernommen.

Nach Abschluss des Differenzbildverfahrens liegt ein Ausgangsbild im selben Format wie das Vergleichs- bzw. aktuelle Bild vor. Enthielt das aktuelle Bild ein Testobjekt, finden sich im Ausgangsbild nur noch die Farbwerte des Testobjekts vor einer weißen Fläche. War kein Testobjekt vorhanden, ist das Ausgangsbild komplett weiß. Dies wird zum Abschluss ebenfalls überprüft. Enthält das Ausgangsbild nicht genügend farbige Pixel, interpretiert der Farberkennungs-Algorithmus dies als Bild ohne Testobjekt und terminiert.



(a) RGB-Farbmodell



(b) YCbCr-Farbmodell

Abbildung 5.9: Differenzbildverfahren in verschiedenen Farbmodellen

Aus Gründen der Datenreduktion wurde versucht, das Differenzbildverfahren vor der Farbraumkonvertierung durchzuführen. So hätten nur die Farbpixel des Testobjekts umgewandelt werden müssen, anstatt zwei komplette Aufnahmen. Abb. 5.9 zeigt eine Messung, bei der dasselbe Bild eines Testobjekts in unterschiedlichen Farbmodellen als Eingangsdaten des Differenzbildverfahrens verwendet wurden. Es ist zu sehen, dass ein Differenzbildverfahren im RGB-Farbraum (Abb. 5.9a) ein wesentlich schlechteres Ergebnis erzielt, als wenn Bilddaten im YCbCr-Farbraum vorliegen (Abb. 5.9b). Aus diesem Grund müssen die Bilddaten konvertiert werden, bevor das Differenzbildverfahren auf sie angewendet werden kann.

5.2.2.6 Bilden des 2D-Histogramms

Die vorverarbeiteten und konvertierten Bilddaten sollen nun analysiert werden. Zu diesem Zweck werden sie nach der gleichen Methode, die schon in Abschnitt 5.2.1.1 beschrieben wurde, verarbeitet. Die Pixel der Bilddaten werden entsprechend ihrer Cb- und Cr-Werte den Feldern des Histogramms zugeordnet.

So entsteht das zweidimensionale Histogramm des Testobjekts für die CbCr-Farbskala. Ein beispielhaftes Histogramm eines blauen Testobjekts zeigt Abb. 5.10.

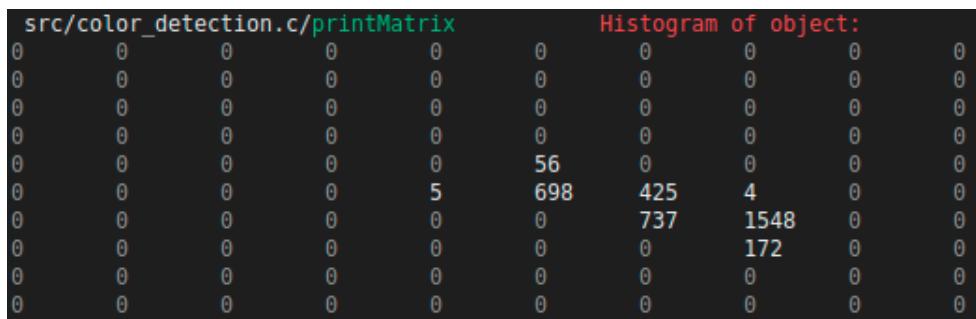


Abbildung 5.10: Histogramm eines blauen Testobjekts

5.2.2.7 Berechnen der Farbanteile

Der Begriff Farbanteil beschreibt den Anteil einer Zielfarbe im analysierten Bild. Um den Farbanteil einer Zielfarbe zu berechnen, werden nur die Werte der Histogrammfelder betrachtet, die in der Farbmaske der Zielfarbe enthalten sind. Die einzelnen Werte der Histogrammfelder werden mit ihren jeweiligen Farbmultiplikatoren multipliziert. Abschließend ergibt sich der Farbanteil aus der Summe dieser Produkte.

Dabei sei auf das Beispiel mit der Farbe Braun verwiesen. Das Bild eines braunen Testobjekts enthält über 97 % seiner Farbinformationen in den Histogrammfeldern 44 und 45. Würde der Farbanteil einzlig aus der Summe der Histogrammfelder der Farbmasken gebildet werden, so wäre in diesem Fall der Farbanteil für alle Farben gleich hoch. Dies hat den Grund, da die Felder 44 und 45 in allen Farbmasken enthalten sind. Die Farbe Braun würde sich demnach nicht hervortun und könnte nicht zuverlässig erkannt werden. Durch die Farbmultiplikatoren wird der Wert der Felder 44 und 45

bei der Bildung des braunen Farbanteils jedoch wesentlich höher gewichtet. Dadurch erzeugt ein braunes Testobjekt auch einen entsprechend hohen braunen Farbanteil.

Demnach werden in diesem Schritt die Farbmasken aller Zielfarben auf die Bilddaten des Testobjekts angewendet. Als Ergebnis liegen anschließend sechs Werte vor, die die Farbanteile der sechs Zielfarben im aktuellen Bild ausdrücken.

5.2.2.8 Farbdetektion mittels Vergleich der Farbanteile

Um das Endergebnis der Farbdetektion zu ermitteln, müssen die unterschiedlichen Farbanteile nun miteinander verglichen werden. Die Zielfarbe mit dem höchsten Farbanteilswert definiert die detektierte Farbe des Testobjekts. Dies kann exemplarisch der Abb. 5.11 entnommen werden.

```
src/color_detection.c/detect_color          Color Proportions:
src/color_detection.c/detect_color          green: 53
src/color_detection.c/detect_color          blue: 10171
src/color_detection.c/detect_color          red: 16
src/color_detection.c/detect_color          orange: 16
src/color_detection.c/detect_color          yellow: 16
src/color_detection.c/detect_color          brown: 22
src/color_detection.c/detect_color          Detected color: blue
src/color_detection.c/detect_color          Pixels counted: 10171
```

Abbildung 5.11: Vergleich der Farbanteile

Es ist davon auszugehen, dass das Histogramm eines Testobjekts eine ähnliche Häufigkeitsverteilung aufweist, wie in der Farbmaske seiner Zielfarbe definiert. Auf diese Weise fließt ein Großteil der Histogrammwerte in die Berechnung des „richtigen“ Farbanteils ein. Bei der Berechnung der „falschen“ Farbanteile überschneiden sich die Farbmasken mit dem Histogramm des Testobjekts nur partiell. Dies führt zu einem wesentlich niedrigeren Farbanteil für die restlichen Zielfarben.

5.2.2.9 Kommunizieren der Farbinformation

Zum Abschluss der Farbdetektion muss die Information über die Farbe des Testobjekts zur Sortiersteuerung bereitgestellt werden. Dies erfolgt über eine sogenannte *Message Queue*. Die Message Queue funktioniert nach dem Prinzip First In First Out. Sollten mehrere Testobjekte mit engem zeitlichen Abstand zueinander analysiert werden, hat dies den Vorteil, dass die detektierten Farben in Form einer Warteschlange vorliegen. Die Sortiersteuerung kann die detektierten Farben so nacheinander aus der Message Queue auslesen. Eine detaillierte Beschreibung der Funktionsweise von Message Queues findet sich in Abschnitt 2.3.3.

Nachdem die detektierte Farbe erfolgreich in die Message Queue geschrieben wurde, ist der Ablauf zur Farberkennung abgeschlossen. Der Farberkennungs-Algorithmus wechselt anschließend zurück in den Status *blockiert* und wartet auf das nächste Interrupt Signal.

5.3 Sortiersteuerung

Der Sortiersteuerungs-Algorithmus wird als Kindprozess vom Hauptprogramm erzeugt. Er ist für die Ansteuerung der *mechanischen Sortierung* zuständig. Über ihn wird somit die eigentliche Sortierung über die Hardware umgesetzt. Da er von dem Ergebnis des Farberkennungs-Algorithmus abhängig ist, wird er erst aktiv, nachdem die Farbe eines Testobjekts detektiert wurde. Dieses Zeitverhalten ist durch die gezielte Platzierung der Lichtschranke der mechanischen Sortierung gewährleistet, wie in Abschnitt 4.3.3 beschrieben wurde.

Des Weiteren ist die Terminierung des gesamten Programms von dem Sortiersteuerungs-Algorithmus abhängig. Dieser enthält einen Counter, welcher sekündlich dekrementiert. Läuft dieser Counter ab, beendet sich die Sortiersteuerung. Das Hauptprogramm wird über die Terminierung des Sortiersteuerungs-Prozesses informiert, wodurch die Terminierungs-Routine des Hauptprogramms zum Laufen kommt.

5.3.1 Herleitung des Sortiersteuerungs-Algorithmus

Um eine präzise Sortierung der Testobjekte zu realisieren, ist die Ansteuerung des in Abschnitt 2.4.2 beschriebenen Motortreibers notwendig. Dieser Motortreiber empfängt die Signale **Enable**, **Direction** und **Step**, mithilfe derer er den zugehörigen Schrittmotor und somit die Sortiermechanik steuert.

5.3.1.1 Steuerung der Sortiermechanik

Zur Steuerung der Sortiermechanik, muss der Sortiersteuerungs-Algorithmus diese Signale setzen und an den Motortreiber schicken. Dies geschieht unter Abhängigkeit der detektierten Farbe des Testobjekts.

Der Sortiersteuerungs-Algorithmus muss wissen, um wie viele Felder n_{Felder} er sich bewegen muss, damit sich der Schieber auf der Position P_{Ziel} des Sortierfeldes der detektierten Farbe befindet. Dafür ist das Wissen über die aktuelle Position $P_{Schieber}$ des Schiebers notwendig. Daraus ergibt sich für n_{Felder} :

$$n_{Felder} = |P_{Ziel} - P_{Schieber}| \quad (5.1)$$

Wie in Abschnitt 4.3.2.4 berechnet, werden für den Positionswechsel des Schiebers um ein Feld $n_{Schritt} = 12$ Schritte benötigt. Somit kann die Anzahl $n_{StepSignal}$ berechnet werden, mit der der Sortiersteuerungs-Algorithmus das **step**-Signal setzen muss, um den Schieber um die gewünschte Anzahl an Feldern zu versetzen.

$$n_{StepSignal} = n_{Felder} \cdot n_{Schritt} \quad (5.2)$$

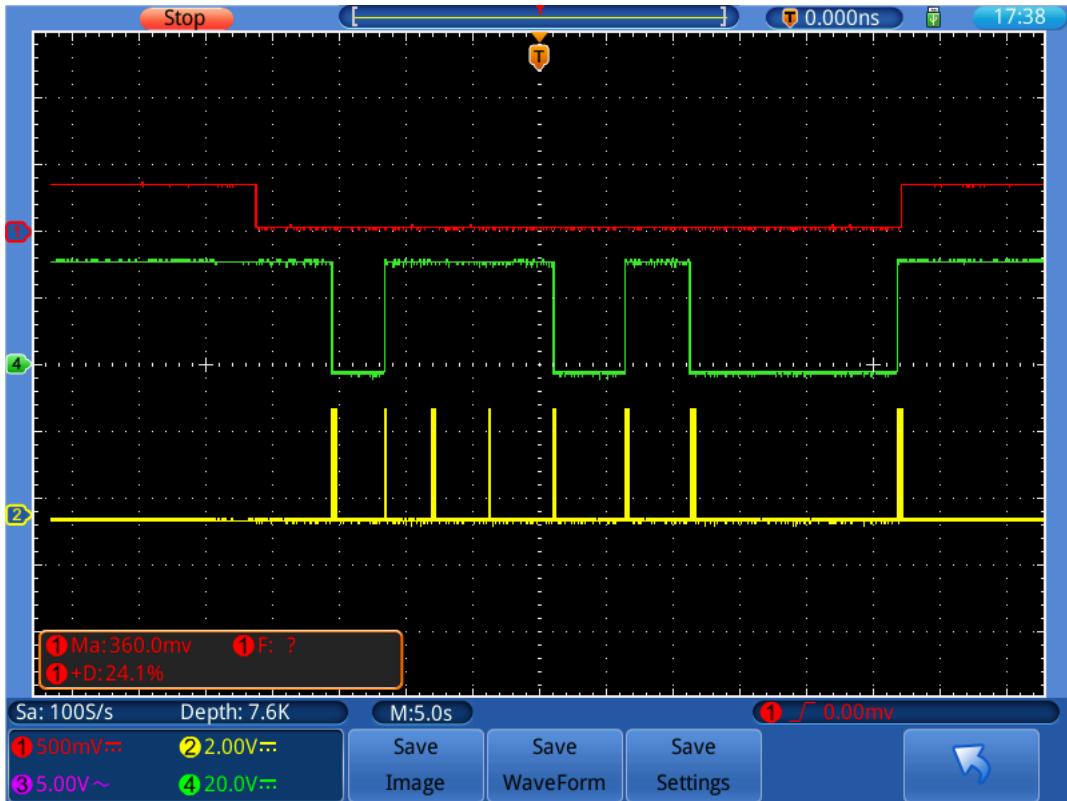


Abbildung 5.12: Steuersignale an den Motortreiber des Schrittmotors (Rot: Enable, Grün: Direction, Gelb: Step)

Zur Ermittlung der Richtung R , in welche der Schrittmotor rotieren soll, wird ebenfalls die aktuelle Position des Schiebers $P_{Schieber}$ und die Position des Zielfeldes P_{Ziel} verwendet.

$$R = P_{Ziel} - P_{Schieber} \quad (5.3)$$

Ist $R > 0$ wird das **Direction**-Signal auf 0 gesetzt. Der Motor dreht sich entgegen des Uhrzeigersinns und der Schieber bewegt sich vorwärts. Sollte $R < 0$ sein, ist das Verhalten des Motors und Schiebers umgekehrt. Der Fall $R = 0$ kann vernachlässigt werden, da hier der Motor nicht bewegt wird.

Das **Enable**-Signal des Motortreibers ist Active Low. Folglich muss an diesem Signal eine 0 anliegen, um den Motor zu aktivieren.

Abb. 5.12 zeigt eine Oszilloskop-Messung der beschriebenen Steuersignale, die vom Sortiersteuerungs-Algorithmus an den Motortreiber gesendet werden. Dabei zeigt der rote Graph das Enable Signal, welches Active Low ist. Der grüne Graph steht für das Direction- und der gelbe Graph für das Step-Signal.

Die Messung zeigt einen vollständigen Programmverlauf mit insgesamt sieben Testobjekten. Anhand des Enable-Signals kann gesehen werden, dass der Motor nur innerhalb des Programmverlaufs aktiv ansteuerbar ist. Es sind außerdem mehrere Richtungsänderungen zu beobachten. Diese können durch Pegeländerungen des Direction-Signals erkannt werden.

Weiterhin fällt auf, dass das Step-Signal unterschiedlich breite Ausschläge aufweist, die durch eine unterschiedliche Anzahl an zurückgelegten Schritten erzeugt wurden. Jeder Ausschlag des Step-Signals repräsentiert dabei eines der Testobjekte - mit Ausnahme des letzten Ausschlages. Dieser letzte Ausschlag des Step-Signals zeigt die Bewegung des Schiebers zurück auf dessen Ausgangsposition. Dies geschieht zum Ende jedes Programmdurchlaufs.

5.3.2 Implementierung des Sortiersteuerungs-Algorithmus

Der Sortiersteuerungs-Algorithmus findet sich in der Datei *sorting_control.c*. Auch er besteht aus verschiedenen Teilschritten, die nacheinander abgearbeitet werden. Dabei werden manche Teile nur einmal (Initialisierung), andere periodisch (Dekrementieren des Counters) und wieder andere Teile immer nach Eintreffen eines Interrupts (restliche Bestandteile) aufgerufen. Die Übersicht des Ablaufs kann Abb. 5.13 entnommen werden.

5.3.2.1 Initialisierung

Die Initialisierungs-Routine wird aufgerufen, nachdem der Sortiersteuerungs-Prozess vom Hauptprogramm erstellt wurde. In dieser werden die Signale des Motortreibers initialisiert. Darauffolgend wird das **Enable**-Signal des Motortreibers auf 0 gesetzt und somit der Motor aktiviert.

Es ist zwingend erforderlich, den Schrittmotor über die gesamte Laufzeit des Programms zu aktivieren. Dies hat den Grund, dass sich die Position der Antriebswelle bei Aktivieren und Deaktivieren des Motors leicht verstellt. Wird das **Enable**-Signal des Schrittmotors vor einem Schritt aktiviert (also 0 gesetzt) und nach diesem Schritt wieder deaktiviert (also 1 gesetzt), kann das System nicht robust laufen.

Weiterhin wird Terminierungs-Counter mit einem Startwert initialisiert und die Initialisierungs-Routine setzt den Interrupt-Handler mit der Interrupt Service Routine. Anschließend wechselt der Prozess in den Status *blockiert*.

5.3.2.2 Dekrementieren des Counters

Sollten für 15 s keine neuen Testobjekte in das System eingespeist werden, soll sich der Farbsortier-Demonstrator abschalten. Aus diesem Grund wurde der Terminierungs-Counter eingeführt. Dieser wird vom Sortiersteuerungs-Prozess sekündlich dekrementiert. Sobald ein Objekt die Lichtschranke durchbricht, wird der Wert des Terminierungs-Counters wieder auf seinen Startwert gesetzt. Auf diese Weise kann das System eine Inaktivität des Benutzers registrieren.

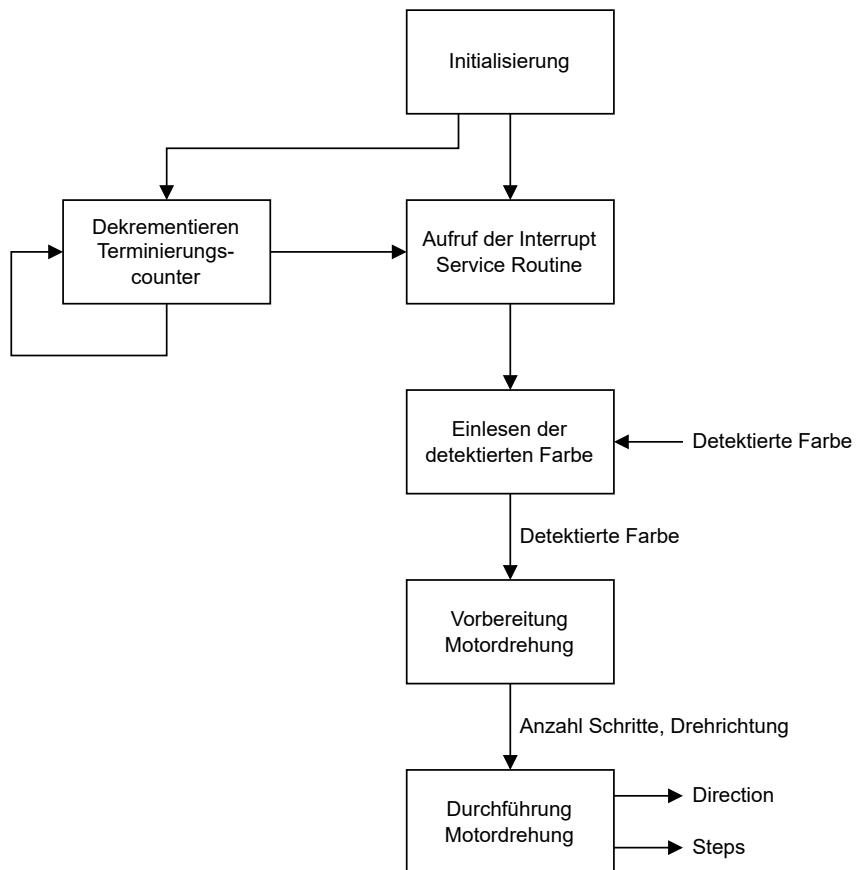


Abbildung 5.13: Ablaufdiagramm Sortiersteuerungs-Algorithmus

Sobald der Terminierungs-Counter den Wert 0 erreicht, ist der Schwellwert dieser Inaktivität erreicht. In der Folge terminiert erst der Sortiersteuerungs-Prozess und leitet dadurch die Terminierung und Abschaltung aller restlichen Prozesse und Komponenten ein.

5.3.2.3 Aufruf der Interrupt Service Routine

Durchbricht ein Objekt die Lichtschranke, wird die Interrupt Service Routine des Sortiersteuerungs-Algorithmus aufgerufen. In dieser muss, wie auch im Farberkennungs-Algorithmus, ein Entprellen des Lichtschrankensignals implementiert werden. Dieses verhindert das mehrfache Starten der Interrupt Service Routine. Die Implementierung ist dabei identisch zu jener im Farberkennungs-Algorithmus (siehe Abschnitt 5.2.2.2).

Nach dem Entprellen wird der Terminierungs-Counter, wie eben beschrieben, auf seinen Startwert gesetzt und anschließend die Interrupt Service Routine aufgerufen. Durch diese wird der weitere Ablauf der Sortiersteuerung gestartet.

5.3.2.4 Einlesen der detektierten Farbe

Die Information über die detektierte Farbe ist absolut notwendig für eine erfolgreiche Sortierung des Testobjekts. Durch den Farberkennungs-Algorithmus wurde die detektierte Farbe in eine Message Queue geschrieben. Auf diese Message Queue greift der Sortiersteuerungs-Algorithmus nun lesend zu. Sollten sich mehrere Einträge in der Message Queue befinden, wird nur der erste Eintrag ausgelesen.

5.3.2.5 Vorbereiten der Motordrehung

Bevor der Motortreiber vom Sortiersteuerungs-Algorithmus angesteuert werden kann, muss diese Ansteuerung vorbereitet werden. In dieser Vorbereitung wird die Farbinformation in die Ansteuerungs-Parameter des Motortreibers übersetzt.

Der Sortiersteuerungs-Algorithmus kennt seine aktuelle Position. Wie in Abschnitt 5.3.1 beschrieben, errechnet der Algorithmus die Anzahl der Felder, die er sich bewegen muss. Multipliziert mit der Anzahl der Schritte für einen Feldwechsel, ermittelt er so die Anzahl der **Step**-Signale, die er dem Motortreiber sendet.

Damit der Schieber in die korrekte Richtung rotiert, muss anschließend noch die Drehrichtung R berechnet werden. Dies wird ebenfalls in Abschnitt 5.3.1 erläutert.

5.3.2.6 Durchführen der Motordrehung

Nachdem alle Parameter für die Motordrehung berechnet wurden, werden diese Werte an die Signale **Direction** und **Step** des Motortreibers gesendet.

Um die Richtung des Motors zu steuern, muss das **Direction**-Signal entsprechend dem Wert von R gesetzt werden. Zur Durchführung eines vollen Motorschritts muss das **Step**-Signal zuerst auf den Wert 1 und anschließend auf den Wert 0 gesetzt werden.

Dieses Vorgehen wird so häufig wiederholt, wie es der Wert für die Anzahl an Schritten vorgibt. Dabei ist es notwendig zwischen dem Pegelwechsel eine gewisse Zeit zu warten. So wird den Spulen im Schrittmotor Zeit gegeben, ihr Magnetfeld vollständig aufzubauen und die Antriebsachse um einen vollen Schritt zu rotieren.

Nachdem alle geplanten Schritte erfolgreich durchgeführt wurden, ist die Ansteuerung des Schrittmotors beendet. Der Sortiersteuerungs-Prozess wechselt nun wieder in den Status *blockiert* und wartet auf das nächste Interrupt Signal.

6 Ergebnisse

Es wurde ein System konstruiert und programmiert, das die in Kapitel 3 festgelegten Anforderungen vollständig erfüllt.

Der Farbsortier-Demonstrator ist durch das Transportband in der Lage, Testobjekte aufzunehmen und durch das System zu transportieren. Dabei werden die Testobjekte mit gleichmäßiger Geschwindigkeit transportiert, sodass diese Transportgeschwindigkeit als Grundlage für die Berechnung der weiteren Systemparameter zulässig ist.

Mithilfe der Kombination aus Detektionsbereich und Farberkennungs-Algorithmus ist es dem Farbsortier-Demonstrator möglich, bewegte Testobjekte als solche zu erkennen und mittels Differenzbildverfahrens vom restlichen Bild zu extrahieren. Das Differenzbildverfahren wirkt dabei wie ein Filter, der mit Bildrauschen, Schatten oder der Textur des Transportbandes umgehen kann. Durch diese Extraktion wird die Anzahl der relevanten Pixel reduziert, wodurch die Bildverarbeitungszeit gesenkt wird.

Der Detektionsbereich ist durch dessen Konstruktion unabhängig von äußeren Lichteinflüssen und sorgt somit für einheitliche Bedingungen bei der Bildaufnahme.

Zur Realisierung der Farberkennung wurde ein eigener Algorithmus entwickelt, welcher auf der Farbdarstellung des YCbCr-Farbmodells basiert. Dazu werden Histogramme aus den Cb- und Cr-Werten der eingehenden Bilddaten erstellt, die dann mit vordefinierten Farbmasken der sechs Zielfarben kombiniert werden und so die Anteile der einzelnen Zielfarben im Bild ergeben. Somit ist das System in der Lage, die Testobjekte einer der sechs vordefinierten Zielfarben zuzuordnen.

Die Kombination von mechanischer Sortierung und Sortiersteuerungs-Algorithmus gewährleistet die Sortierung der Testobjekte gemäß der detektierten Farbe. Sie ist in der Lage sechs verschiedene Sortierfächer präzise anzusteuern. Die mechanische Sortierung kann beliebig viele Positionswechsel vollziehen, ohne dabei an Genauigkeit zu verlieren.

Der Ablauf des Farbsortier-Demonstrators unterliegt dabei gewissen Echtzeitbedingungen. So steht beispielsweise die mechanische Sortierung in zeitlicher Abhängigkeit zur Bildaufnahme. Es ergibt sich eine Vielzahl an verschiedenen Parametern, die zur Einhaltung der Echtzeitbedingungen notwendig sind. Diese können in gesammelter Form dem Kapitel 7 entnommen werden. Eine Übersicht des Zeitverhaltens wird in Abb. 6.2 visualisiert.

Um die Latenzen des Systems so gering wie möglich zu halten, arbeitet der Farbsortier-Demonstrator mit Interrupts, die auf die Eingangssignale von Lichtschranken reagieren. So können die Ressourcen des Systems optimal ausgenutzt und die Echtzeitbedingungen eingehalten werden. Folglich ist die Echtzeitfähigkeit des Systems gegeben.

Der Farbsortier-Demonstrator ist in der Lage den gesamten Ablauf beliebig oft zu wiederholen, ohne dabei einen Rückgang der Trefferquote zu produzieren. Da die Trefferquote die höchste Aussagekraft über die Qualität des Farbsortier-Demonstrators hat, wurden mehrere Validierungstests durchgeführt, um diese empirisch zu erheben.

Für die Validierungstests wurde eine Messreihe mit zehn Versuchen durchgeführt. Pro Versuch wurden insgesamt 100 Testobjekte mit einem kurzen zeitlichen Abstand in das System eingespeist. Die durchschnittliche Trefferquote nach zehn Versuchen lag bei exakt 95 %. Somit wurde die ausgegebene Zielquote von 75 % weit übertroffen. Die detaillierten Aufzeichnungen der Validierungstests finden sich in Abb. 6.1

Test Datum	Grün			Blau			Rot			Orange			Gelb			Braun			Total			Acc.
	Σ	T	F																			
10.05.22	12	10	2	26	26	0	13	11	2	24	24	0	12	12	0	13	12	1	100	95	5	95%
10.05.22	12	11	1	26	26	0	13	9	4	24	24	0	12	12	0	13	11	2	100	93	7	93%
10.05.22	12	12	0	26	25	1	13	13	0	24	23	1	12	11	1	13	13	0	100	97	3	97%
27.05.22	15	13	2	19	19	0	16	13	3	25	23	2	14	14	0	11	10	1	100	92	8	92%
27.05.22	15	15	0	19	19	0	16	15	1	25	25	0	14	14	0	11	10	1	100	98	2	98%
30.05.22	11	9	2	21	21	0	22	20	2	19	19	0	18	18	0	9	9	0	100	96	4	96%
30.05.22	11	11	0	21	21	0	22	20	2	19	18	1	18	16	2	9	9	0	100	95	5	95%
30.05.22	11	7	4	21	18	3	22	22	0	19	19	0	18	18	0	9	8	1	100	92	8	92%
30.05.22	11	10	1	21	19	2	22	22	0	19	18	1	18	15	3	9	9	0	100	93	7	93%
27.06.22	14	14	0	24	23	1	15	15	0	21	21	0	14	14	0	12	12	0	100	99	1	99%
	90.32%			96.88%			91.95%			97.72%			96.00%			94.50%			100			95.00%

Abbildung 6.1: Übersicht Validierungstests

Allerdings weist das System auch kleinere Schwächen seitens der Hardware auf. Die geringe Anzahl falsch sortierter Testobjekte ist auf die zwei verwendeten Reflex-Lichtschranken zurückzuführen. Während der Validierungstests war auffällig, dass die Lichtschranken von den Testobjekten gelegentlich nicht durchbrochen wurden. Dies könnte durch die reduzierte Ausstattung der Lichtschranken verursacht werden. Da diese keinen Polarisationsfilter integriert haben, wird die Objekterkennung reflektierender Objekte beeinträchtigt. Weiterhin ist die Konfiguration der detektierten Distanz nur sehr grob einstellbar. Es kann also davon auszugehen sein, dass der Einsatz von Lichtschranken höherer Qualität ein besseres Ergebnis erzielen würde.

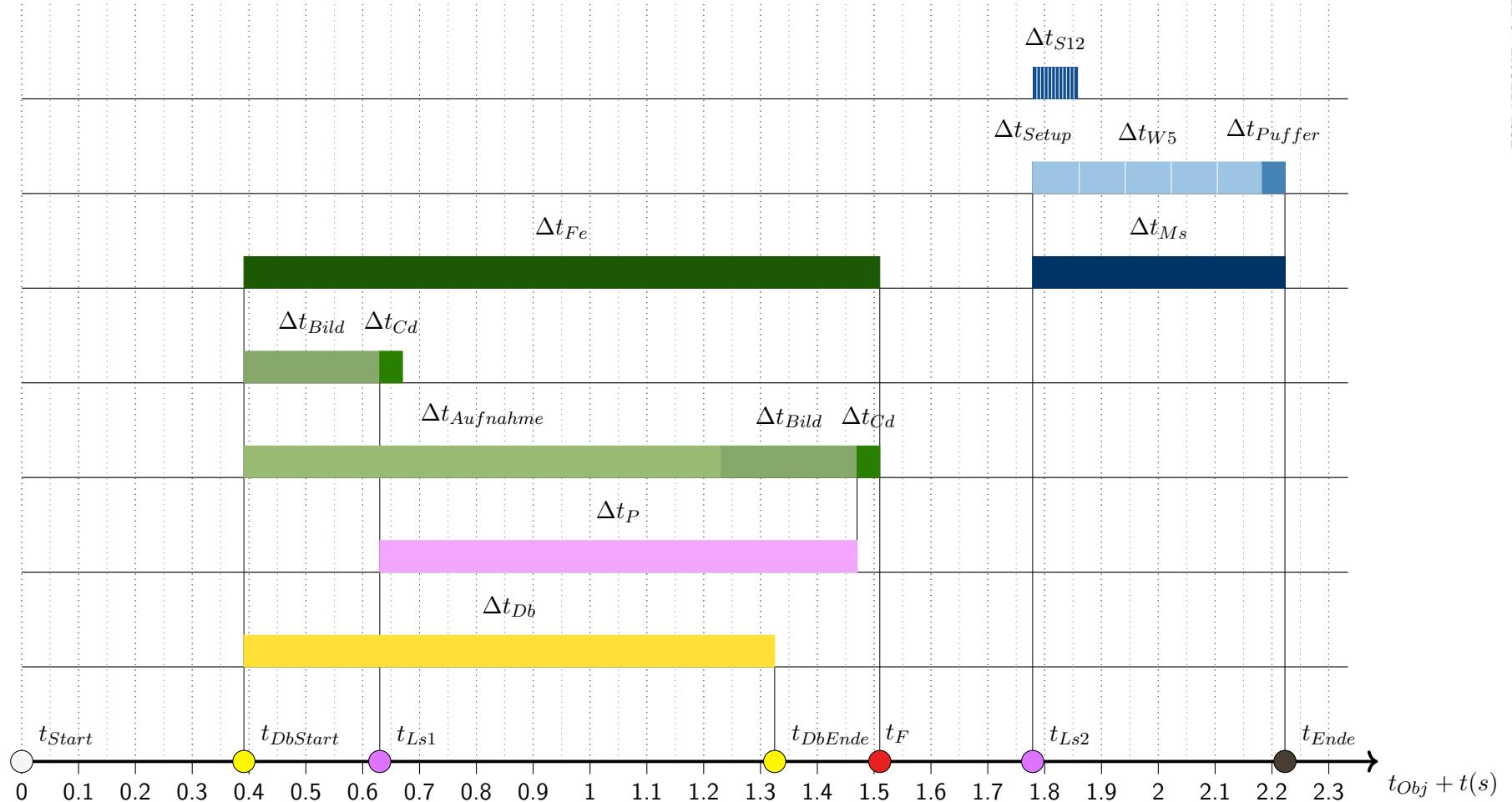


Abbildung 6.2: Zeitverhalten des Farbsortier Demonstrators

7 Ausblick

Die Arbeit legt einen guten Grundstein für weitere Arbeiten. Der Farberkennungs-Algorithmus hat sich als robust erwiesen und erzielt exzellente Ergebnisse. Es bleibt zu erproben, inwieweit dieser Algorithmus auch auf komplexere Probleme anwendbar ist.

So ist beispielsweise vorstellbar, dass die Unterscheidung mehrfarbiger Objekte ebenfalls mithilfe des Farberkennungs-Algorithmus realisierbar wäre. Da in dieser Arbeit nur einfarbige Objekte verarbeitet werden sollten, liegen die Felder der Farbmasken direkt aneinander. Es bilden sich logischerweise zusammenhängende Flächen auf der CbCr-Farbskala. Durch den Farberkennungs-Algorithmus könnten jedoch auch mehrere unabhängige Bereiche der CbCr-Farbskala in den Farbmasken festgelegt werden. So könnten beispielsweise rotblaue Testobjekte von rotgrünen Testobjekten einfach unterschieden werden. Unter Hinzunahme der Farbmultiplikatoren wäre außerdem eine Unterscheidung mehrfarbiger Testobjekte mit unterschiedlich großen Farbflächen möglich. Es ist denkbar, dass der Farberkennungs-Algorithmus rotblaue Testobjekte mit großem Blauanteil von rotblauen Testobjekten mit großem Rotanteil unterscheiden könnte.

Ein weiteres Ziel wäre es außerdem, eine Entwicklungsumgebung mit komplett vorhersagbarem Verhalten zu verwenden. Das Zeitverhalten des Farberkennungs- und Sortiersteuerungs-Algorithmus kann ausreichend, jedoch nicht vollständig berechnet werden. Demnach würde die Effizienz und Geschwindigkeit des System erhöht werden, wenn als Entwicklungsplattform ein Mikrocontroller mit einem Echtzeitbetriebssystem oder ein FPGA verwendet werden würde. Da diese Arbeit mit Hinblick auf eine solche Implementierung erarbeitet wurde, sollte der Transfer zu einem solchen System gelingen.

Literaturverzeichnis

- [1] LLC Allegro MicroSystems. A4988 datasheet. https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf, 2014. Accessed: 2022-07-11.
- [2] Zena Ahmed Alwan, Hamid Mohammed Farhan, and Siraj Qays Mahdi. Color image steganography in ycbcr space. International Journal of Electrical and Computer Engineering, 10:202–209, 2020.
- [3] AZ-Delivery. Kf-301 relais-modul datasheet. https://cdn.shopify.com/s/files/1/1509/1638/files/KF-301_1-Relais_Modul_Datenblatt_AZ-Delivery_Vertriebs_GmbH_2fbfb0f0-d405-4fd8-b04e-1c3ae81a242f.pdf?v=1646995909. Accessed: 2022-07-11.
- [4] Herbert Bernstein. Optische sensoren und ihre anwendungen, 2016.
- [5] Klaus Dembowski. Raspberry Pi - Das technische Handbuch. Springer Verlag, 2019.
- [6] Michael Derryberry and Jeremiah McCarthy. Pedestrian Detection Image Processing with FPGA. 2014.
- [7] Erich Ehses, Lutz Köhler, Petra Riemer, Horst Stenzel, and Frank Victor. Systemprogrammierung in unix / linux. 2012.
- [8] Raspberry Pi Foundation. Raspberry pi manual. <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>, 2021. Accessed: 2022-07-11.
- [9] Raspberry Pi Foundation. Raspicam manual. <https://www.raspberrypi.com/documentation/accessories/camera.html#raspistill>, 2021. Accessed: 2022-07-11.
- [10] V. Ganesh and S. Vaidyanathan. Image Processing in Artificial Intelligence with Sensors. 2012.
- [11] Rafael C. Gonzalez and Richard E. Woods. Digital image processing. Prentice Hall, 2008.
- [12] Behic Guven. Building a color recognizer in python. <https://towardsdatascience.com/building-a-color-recognizer-in-python-4783dfc72456>, 2020. Accessed: 2022-07-11.

- [13] Mokhtar Hasan, Rafiqul Zaman Khan, Noor A Ibraheem, Mokhtar M Hasan, Rafiqul Z Khan, and Pramod K Mishra. Arpn journal of science and technology:: Understanding color models: A review. ARPN Journal of Science and Technology, 2, 2012.
- [14] G D Hastings and A Rubin. Colour spaces-a review of historic and modern colour models. 71:133–143, 2012.
- [15] Mars Incorporated. Our mms colors. <https://www.mms.com/en-gb/shop-products/our-colors>, 2022. Accessed: 2022-07-11.
- [16] Deborah T. Joy, Gitika Kaur, Aarti Chugh, and Shalini B. Bajaj. Computer vision for color detection. International Journal of Innovative Research in Computer Science and Technology, 9, 5 2021.
- [17] Cihat Karaali. Grundlagen der Steuerungstechnik. Springer Fachmedien Wiesbaden, 2013.
- [18] LEDmaxx. Led stiftsockellampe cob datenblatt. https://www.ledmaxx.de/mediafiles/Datenblaetter/27GY251_db.pdf, 2022. Accessed: 2022-07-11.
- [19] ARM Ltd. Arm cortex-a72 mpcore processor technical reference manual. <https://developer.arm.com/documentation/100095/0003/?lang=en>, 2016. Accessed: 2022-07-11.
- [20] Christian Maurer. Prozessmodell. Springer Berlin Heidelberg, 1999.
- [21] Modelcraft. Rb350100-0a101r datasheet. <https://asset.conrad.com/media10/add/160267/c1/-/en/000227560DS01/datenblatt-227560-modelcraft-rb350100-0a101r-getriebemotor-12-v-1100.pdf>, 2014. Accessed: 2022-07-11.
- [22] Fernando Puente, Frese Christian, and Beyerer Jürgen. Licht. Springer Berlin Heidelberg, 2012.
- [23] Khamar Basha Shaik, P. Ganesan, V. Kalist, B. S. Sathish, and J. Merlin Mary Jenitha. Comparative study of skin color detection and segmentation in hsv and ycbcr color space. volume 57, pages 41–48. Elsevier, 2015.
- [24] Guibin Sun, Zhi Weng, Peng Zhao, Dongxu Guo, Yu Tian, and Lei Xiao. Design of color recognition system based on fpga. pages 322–325, 2016.
- [25] A.S. Tanenbaum. Moderne Betriebssysteme. Pearson Studium - IT. Pearson Deutschland, 2009.
- [26] Silicon TechnoLabs. Infrared proximity sensor datasheet. www.siliconTechnolabs.in. Accessed: 2022-07-11.
- [27] Michael Thiele. Entwicklung eines kamerabasierten bildverarbeitungsgestützten Präsenzdetektors in VHDL. 2015.
- [28] K. Tkotz and P. Bastian. Fachkunde Elektrotechnik. Europa-Fachbuchreihe für

- elektrotechnische Berufe. Verlag Europa-Lehrmittel, 2006.
- [29] Schneider Electric Motion USA. Nema 17 datasheet. <https://www.novantaims.com/downloads/quickreference/NEMA17.pdf>, 2018. Accessed: 2022-07-11.
- [30] V. Vezhnevets, V. Sazonov, and Alla Andreeva. A Survey on Pixel-Based Skin Color Detection Techniques. Nederlands Tijdschrift voor de Klinische Chemie, 21:238–244, 1996.
- [31] Sven-Hendrik Voß and Marc Geh. Real-time processing for light field imaging using an FPGA-based approach. 2020.
- [32] Heinz Wörn and Uwe Brinkschulte. Echtzeitsysteme. 2005.

Anhang

Systemparameter

Beschreibung	Parameter	Wert	Kap.	Gl.
Zeitpunkt Einspeisung Objekt	t_{Obj}	-	4.2.3.1	-
Startzeitpunkt System	t_{Start}	0 s	4.1.3	-
Objekt betritt Db	$t_{DbStart}$	0.391 s	4.2.1	-
Ls Db wird durchbrochen	t_{Ls1}	0.63 s	4.2.3.2	4.40
Objekt verlässt Db	t_{DbEnde}	1.325 s	4.2.1	-
Farbe wurde erkannt	t_F	1.51 s	4.2.3.2	4.52
Ls Ms wird durchbrochen	t_{Ls2}	1.779 s	4.3.3	4.98
Ms wird angesteuert	t_{Sort}	1.779 s	4.3.3	4.98
Objekt verlässt Tb	t_{Ende}	2.223 s	4.1.3	4.17

Tabelle 7.1: Zeitpunkte

Beschreibung	Parameter	Wert	Kap.	Gl.
Startpunkt System	P_{Start}	0 m	4.1.3	-
Anfang Detektionsbereich	$P_{DbStart}$	0.044 m	4.2.1	-
Position Ls Db	P_{Ls1}	0.071 m	4.2.3.2	4.42
Ende Detektionsbereich	P_{DbEnde}	0.149 m	4.2.1	-
Position Ls Ms	P_{Ls2}	0.2 m	4.3.3	4.101
Endpunkt System	P_{Ende}	0.25 m	4.1.3	4.14

Tabelle 7.2: Wegpunkte

Beschreibung	Parameter	Wert	Kap.	Gl.
Durchqueren Db	Δt_{Db}	0.934 s	4.2.1	4.23
Bildspeicherung	Δt_{Bild}	0.239 s	4.2.3.1	4.29
Abstand zwischen zwei Aufnahmen	Δt_A	0.84 s	4.2.3.1	4.31
Bild auf Objekt überprüfen	Δt_P	0.84 s	4.2.3.2	4.48
Farberkennungs-Algorithmus	Δt_{Cd}	0.04 s	4.2.3.2	4.51
Farberkennungs-Algorithmus	Δt_{CdMin}	0.029 s	4.2.3.2	-
Farberkennungs-Algorithmus	Δt_{CdMax}	0.037 s	4.2.3.2	-
Gesamte Fe	Δt_{Fe}	1.119 s	4.2.3.2	4.55
Setup Ms	Δt_{Setup}	0.001 s	4.3.3	-
Motorschritt	Δt_S	0.0065 s	4.3.3	-
Positionswechsel über 1 Feld	Δt_{W1}	0.081 s	4.3.3	4.80
Positionswechsel über 5 Felder	Δt_{W5}	0.4032 s	4.3.3	4.84
Von Ls Ms bis abgeschlossene Sc	Δt_{Sc}	0.4042 s	4.3.3	4.87
Puffer für Ms	Δt_{Puffer}	0.0404 s	4.3.3	4.91
Gesamte Ms	Δt_{Ms}	0.4446 s	4.3.3	4.95

Tabelle 7.3: Zeitintervalle

Beschreibung	Parameter	Wert	Kap.	Gl.
Drehwinkel pro Schritt Schrittmotor	γ_M	1.8°	4.3.2.4	-
Drehwinkel pro Schritt Steuerachse	γ_{Sa}	0.9°	4.3.2.4	4.64
Winkel der Steuerachse / 2	α	27.913°	4.3.2.4	4.67
Winkel der Steuerachse	α_5	55.826°	4.3.2.4	4.71
Abstandswinkel für einen Feldwechsel	α_1	11.165°	4.3.2.4	4.74

Tabelle 7.4: Winkel

Beschreibung	Parameter	Wert	Kap.	Gl.
Umfang Transportband	u_S	0.584 m	4.1.1	-
Breite Transportband	b_{Tb}	0.04 m	4.1.1	-
Radius Antriebsrad	r_{Ar}	0.0179 m	4.1.1	-
Radius Laufrad	r_{Lr}	0.0181 m	4.1.1	-
Umfang Laufrad	u_{Lr}	0.112 m	4.1.3	4.7
Distanz Achsen	d_1	0.236 m	4.1.3	-
$\frac{1}{8}$ Umfang u_{Lr}	d_2	0.014 m	4.1.3	4.11
Länge Detektionsbereich	s_{Db}	0.105 m	4.2.1	4.20
Breite oben Schieber	d_{Bo}	0.088 m	4.3.1	-
Breite unten Schieber	d_{Bu}	0.045 m	4.3.1	-
Breite Sortierfächer	d_{Bf}	0.03 m	4.3.1	-
Radius kleines Zahnrad	$r_{ZrKlein}$	0.02 m	4.3.2.4	-
Radius großes Zahnrad	$r_{ZrGroß}$	0.04 m	4.3.2.4	-
Strecke $P_0 P_{Mu}$	S_a	0.074 m	4.3.2.4	4.65
Strecke $P_{Sa} P_{Mu}$	S_b	0.14 m	4.3.2.4	4.66

Tabelle 7.5: Strecken, Distanzen, Radien, Umfänge

Beschreibung	Parameter	Wert	Kap.	Gl.
Drehzahl Gleichstrommotor	n_{Gm}	60 U/min	4.1.3	-
Winkelgeschwindigkeit Gleichstrommotor	ω_{Gm}	6.283 rad s ⁻¹	4.1.3	4.1
Geschwindigkeit Transportband	v_T	0.112 m s ⁻¹	4.1.3	4.4
Übersetzung Getriebe	i	0.5	4.3.2.4	4.58
Anzahl möglicher Feldwechsel	n_W	5	4.3.2.4	4.74
Anzahl Motorschritte für einen Feldwechsel	n_S	12	4.3.2.4	4.77

Tabelle 7.6: Sonstige

Histogramme

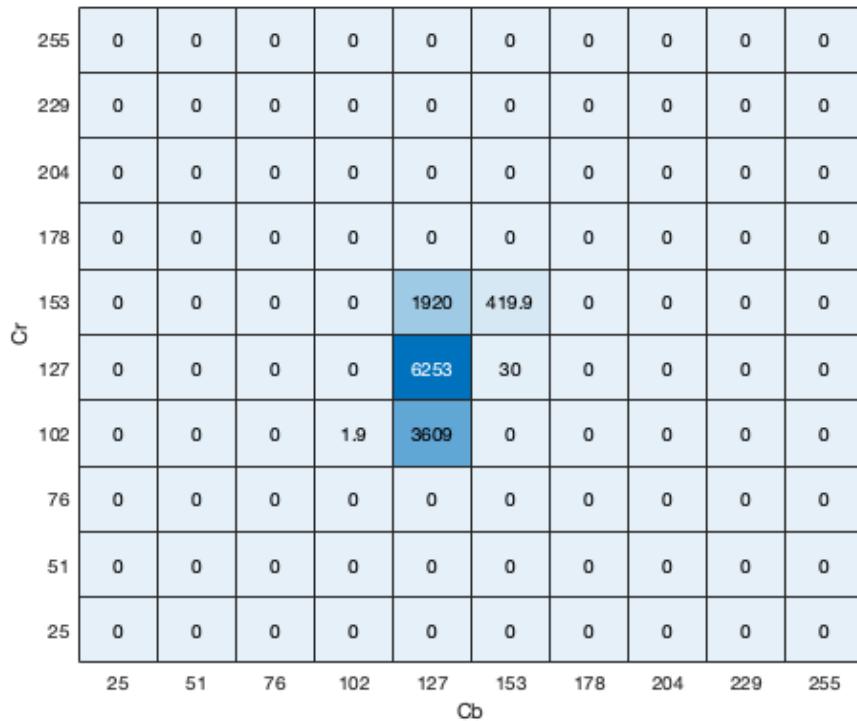


Abbildung 7.1: Grünes Objekt

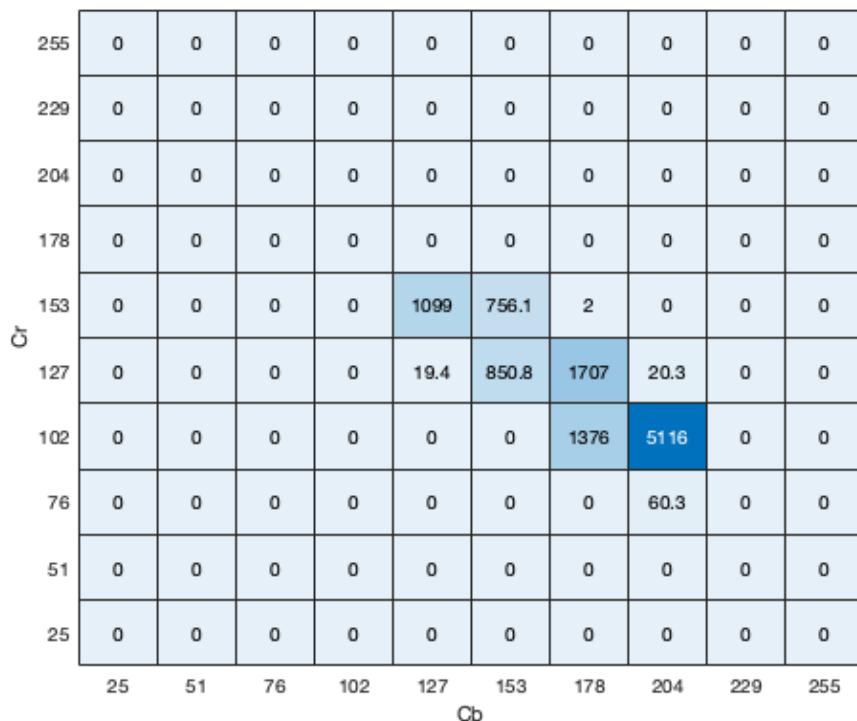


Abbildung 7.2: Blaues Objekt

255	0	0	0	0	0	0	0	0	0	0
229	0	0	0	0	0	0	0	0	0	0
204	0	0	0.4	2024	7	0	0	0	0	0
178	0	0	2.6	4700	1475	0.1	0	0	0	0
153	0	0	0	0	2516	381	0	0	0	0
127	0	0	0	0	8.3	6.5	0	0	0	0
102	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	0	0	0	0	0
51	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0
	25	51	76	102	127	153	178	204	229	255
	Cb									

Abbildung 7.3: Orangenes Objekt

255	0	0	0	0	0	0	0	0	0	0
229	0	0	0	0	0	0	0	0	0	0
204	0	0	0	0	7030	0.7	0	0	0	0
178	0	0	0	0	2414	39.6	0	0	0	0
153	0	0	0	0	2445	349	0	0	0	0
127	0	0	0	0	5	0.3	0	0	0	0
102	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	0	0	0	0	0
51	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0
	25	51	76	102	127	153	178	204	229	255
	Cb									

Abbildung 7.4: Rotes Objekt

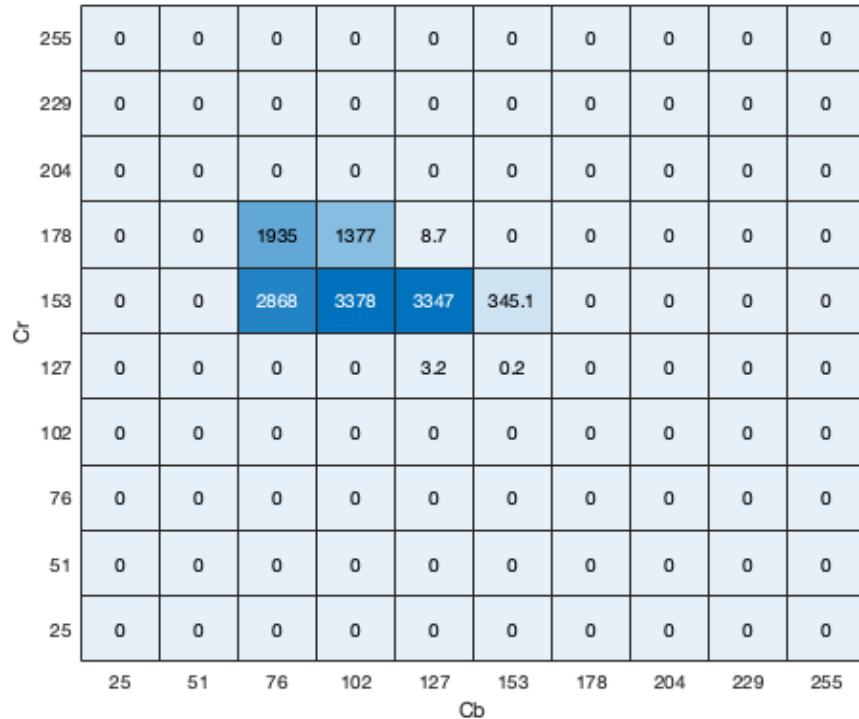


Abbildung 7.5: Gelbes Objekt

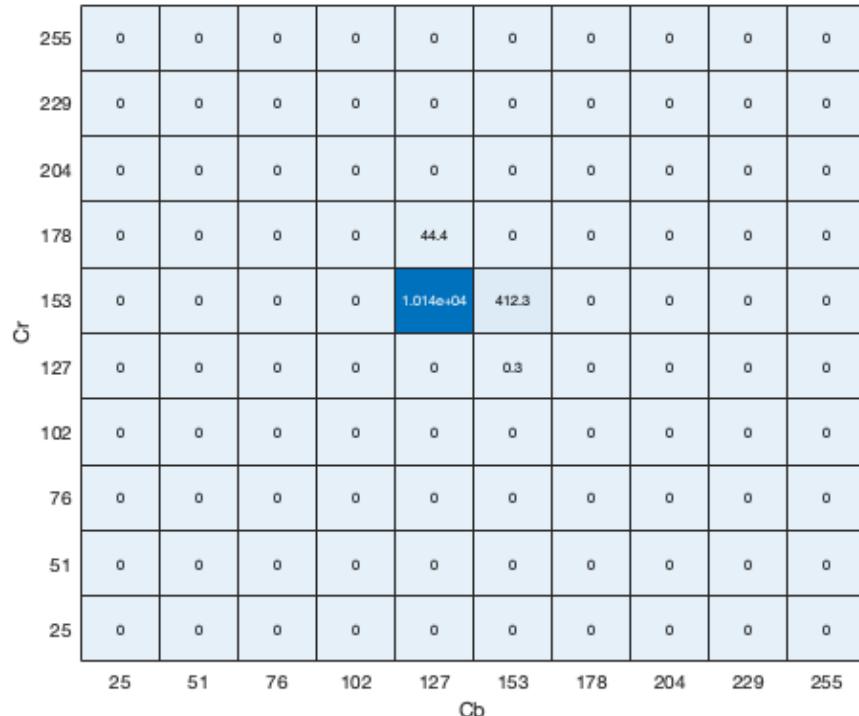


Abbildung 7.6: Braunes Objekt