Ein Bild, das Text, Schrift, Screenshot, Grafiken enthält.

Automatisch generierte Beschreibung

**Camera-Based Line Following and Mapping for Autonomous Navigation with ROS2**

Frankfurt University of Applied Sciences

**Intelligente Sensoren und Autonomes Fahren**

Mechatronik und Robotik (M.Sc.)

vorgelegt von

|  |  |
| --- | --- |
| Mutasem Bader | 1343928 |
| Felix Fritz Biermann | 1369795 |

Im März 2025

Prüfer: Prof. Dr. Peter Nauth

# Eidesstattliche Erklärung

Hiermit versichere ich, das vorliegende Projekt selbstständig und nur unter Verwendung der von mir angegebenen Quellen und Hilfsmittel verfasst zu haben. Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit hat in dieser oder vergleichbarer Form noch keinem anderem Prüfungsgremium vorgelegen.

Datum:

Unterschrift:

Unterschrift:

# Inhaltsverzeichnis

[Eidesstattliche Erklärung II](#_Toc192503790)

[Inhaltsverzeichnis III](#_Toc192503791)

[Abbildungsverzeichnis VI](#_Toc192503792)

[Tabellenverzeichnis VII](#_Toc192503793)

[Kurzfassung & Aufgabestellung VIII](#_Toc192503794)

[1 Einleitung 1](#_Toc192503795)

[2 Hardwareaufbau 2](#_Toc192503796)

[2.1 Radencoder 3](#_Toc192503797)

[2.2 Mikrocontroller 3](#_Toc192503798)

[2.3 Kamera 4](#_Toc192503799)

[2.4 Steuerungs-PC 4](#_Toc192503800)

[3 Computer Vision 5](#_Toc192503801)

[3.1 Kamera 5](#_Toc192503802)

[3.2 Objektiv & Abbildungsfehler 8](#_Toc192503803)

[3.3 Bildauswertung mit OpenCV 9](#_Toc192503804)

[3.3.1 OpenCV-Bildformat 10](#_Toc192503805)

[3.3.2 Vorbereitung und Filterung des Bildes 10](#_Toc192503806)

[3.3.3 Detektion der Linie 12](#_Toc192503807)

[3.3.4 Auswertung der Konturen 13](#_Toc192503808)

[3.3.5 Der Kalman Filter 14](#_Toc192503809)

[3.3.6 Visualisierung der Bildverarbeitung 14](#_Toc192503810)

[3.4 Validierung der Robustheit der Linienerkennung 14](#_Toc192503811)

[3.5 Anwendung der Bildverarbeitung und Linienerkennung bzw. -verfolgung auf praktische Anwendungen 16](#_Toc192503812)

[3.6 Grenzen und Nachteile der kamerabasierten Bildverarbeitung 16](#_Toc192503813)

[4 GUI-Toolkits 18](#_Toc192503814)

[4.1 Designwerkzeug „Qt Designer“ 18](#_Toc192503815)

[4.1.1 Interface Schnittstelle zu ROS 2 19](#_Toc192503816)

[5 PID-Regler zur Linienverfolgung 21](#_Toc192503817)

[5.1 Grundlagen des PID-Reglers 21](#_Toc192503818)

[5.2 Anwendung im geschlossenen Regelkreis 22](#_Toc192503819)

[5.3 PID-Tuning zur Optimierung der Steuerung 22](#_Toc192503820)

[5.4 Kommunikationsprotokoll zwischen PC und Arduino 23](#_Toc192503821)

[5.4.1 Datentypen der übertragenen Nachrichten 23](#_Toc192503822)

[5.4.2 Datenstruktur des Motorsteuerungs-Frames 23](#_Toc192503823)

[5.4.3 Datenstruktur des Encoder-Daten-Frames 24](#_Toc192503824)

[5.4.4 Gesamtarchitektur der Kommunikation 24](#_Toc192503825)

[6 Protokollierung in ROS 2 25](#_Toc192503826)

[7 ROS2 Framework 26](#_Toc192503827)

[7.1 Definition & Architektur 26](#_Toc192503828)

[7.2 Kommunikationsmodellen 27](#_Toc192503829)

[7.3 Graphische Darstellung der Kommunikation zwischen Nodes 28](#_Toc192503830)

[8 Path Mapping: Kartierung der Roboterbewegung 29](#_Toc192503831)

[8.1 Funktionsweise des Path-Mapping-Nodes 29](#_Toc192503832)

[8.1.1 Bestimmung der Roboterbewegung 29](#_Toc192503833)

[8.2 Transformation von Koordinatenrahmen in ROS 2 30](#_Toc192503834)

[8.3 Implementierung des TF2-Nodes 30](#_Toc192503835)

[8.4 Erstellung der Karte und Visualisierung 31](#_Toc192503836)

[8.5 Visualisierung der Transformationen und Pfade in RViz2 32](#_Toc192503837)

[9 Zusammenfassung und Ausblicke 33](#_Toc192503838)

[Literaturverzeichnis VIII](#_Toc192503839)

[Anhang X](#_Toc192503840)

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kurzfassung & Aufgabestellung

# Einleitung

# Hardwareaufbau

Das für dieses Projekt entwickelte Fahrzeug ist ein motorisiertes Kettenfahrzeug mit Kameraaufbau. Das verwendete Fahrwerk stammt von einem Zumo-Roboter der Firma Pololu. Dieses Fahrwerk verfügt über zwei Kettenantriebe, die jeweils von einem DC-Getriebemotor mit einer Übersetzung von 75:1 angetrieben werden. Es besitzt außerdem eine Batteriehalterung, in der vier AA-Batterien bzw. -Akkus platziert werden können. Die in Reihe geschalteten Zellen dienen zur Versorgung der beiden DC-Motoren.

Ein Bild, das Maschine, Kamera, Autoteile, optisches Instrument enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .:Aufbau des verwendeten Roboters [Eigenaufnahme]

Zum Betrieb der Motoren befindet sich oberhalb des Fahrwerks eine Platine, das sogenannte Shield. Dieses Shield stellt die Schnittstelle zwischen den Motoren, der Spannungsversorgung und dem verwendeten Mikrocontroller dar. Zur Motoransteuerung befinden sich darauf zwei H-Brücken. Mit diesen H-Brücken ist es möglich, die Motoren sowohl rückwärts als auch vorwärts laufen zu lassen. Außerdem kann durch ein PWM-Signal die Geschwindigkeit der Motoren unabhängig voneinander geregelt werden.

Auf das Shield kann ein Mikrocontroller im Formfaktor eines Arduino UNO aufgesteckt werden.

## Radencoder

Eine Anforderung an das Projekt war, eine Rückmeldung über die gefahrene Strecke zu erhalten. Das Zumo-Fahrwerk wurde dafür mit Radencodern ausgestattet. Die Firma Pololu bietet passende Radencoder an, diese sind jedoch für den Einbau in das Nachfolgemodell vorgesehen, das über ein spezielles Shield verfügt. Mit mittlerem Aufwand konnte jedoch auch das verwendete Modell auf Radencoder umgerüstet werden.

Hierzu wird an den Motor eine Platine angelötet, auf der zwei Hall-Sensoren verbaut sind. Zusätzlich wird eine Magnetscheibe auf die Motorwelle aufgesteckt. Die Hall-Sensoren sind in der Lage, die Magnetfeldänderungen der Scheibe zu detektieren und in elektrische Signale umzuwandeln.

Da die beiden Hall-Sensoren um 90 Grad versetzt angeordnet sind, kann über eine XOR-Verknüpfung die Drehrichtung des Motors ermittelt werden. Diese Signale werden anschließend von einem Mikrocontroller ausgewertet. Zu beachten ist, dass die Motordrehzahl vor dem Getriebe erfasst wird.[16]

## Mikrocontroller

Zur Verarbeitung der Encoder Signale und zur Steuerung der Motoren kommt ein Arduino Leonardo zum Einsatz. Dieses Entwicklungsboard besitzt zwar denselben Formfaktor wie ein Arduino UNO, ist jedoch mit dem ATmega32U4 ausgestattet, einem leistungsfähigem Mikrocontroller.

Der Arduino Leonardo wurde gewählt, da er im Gegensatz zum ATmega328P nicht nur an zwei, sondern an mehreren Pins Hardware-Interrupts auslösen kann. Während der ATmega328P Interrupts nur an den Pins 2 und 3 unterstützt – die beim Zumo Shield bereits anderweitig belegt sind – kann der ATmega32U4 Interrupts an den Pins 0, 1, 2, 3 und 7 verarbeiten.

Interrupts sind für die Auswertung der Radencoder unerlässlich, da die Signalimpulse im kHz-Bereich liegen und jeder einzelne Impuls zuverlässig erkannt und gezählt werden muss.

Pololu stellt eine Arduino-Bibliothek zur Verfügung, die jedoch für das Nachfolgemodell konzipiert wurde. Diese Bibliothek wurde so angepasst, dass sie mit dem Arduino Leonardo und den verwendeten Pins kompatibel ist.

Der Mikrocontroller ist zudem über eine serielle Schnittstelle mit dem Steuerungs-PC verbunden. Über diese Schnittstelle können die Radencoder-Werte übertragen sowie Geschwindigkeitsvorgaben für die Motoren empfangen werden, die anschließend in PWM-Signale umgewandelt werden.

## Kamera

Zum Einsatz kommt eine USB-2.0-Kamera mit einem „Sony IMX317“-Sensor und einem Objektiv mit variabler Brennweite von 2,8 bis 12 mm. Näher auf Kamera und Objektiv wird im Kapitel xxxxx und xxxxx eingegangen.

Die Kamera sowie der USB-Hub sind auf einer 3D-gedruckten Halterung montiert, die am Fahrgestell des Arduino verschraubt ist. Die Kamera ist etwa 12 cm über der Fahrbahn positioniert und in einem 45-Grad-Winkel zur Horizontalen nach unten gerichtet.

## Steuerungs-PC

Da die Bildauswertung einer hochauflösenden Kamera sehr rechenintensiv ist, kann sie nicht auf dem Arduino des Roboters durchgeführt werden. Das Kamerabild, die Encoder Werte und die Fahrbefehle werden daher über einen USB-Hub übertragen, während die eigentliche Auswertung und Steuerung auf einem leistungsfähigen Linux-Rechner erfolgt.

Da dieses Projekt die Integration zahlreicher Komponenten und Aufgaben erfordert, kommt hierzu das Robot Operating System (ROS 2) zum Einsatz.

*Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.*

Abbildung .: schematischen Aufbau und die Verbindungen der einzelnen Komponenten

# Computer Vision

Der Bereich Computer Vision ist ein umfangreiches Gebiet, das darauf abzielt, nützliche Informationen aus einem Bild zu generieren. Die Hauptaufgaben der Computer Vision bzw. Bildverarbeitung liegen dabei in der Bild- bzw. Objektklassifikation, also dem Zuordnen eines Bildes oder von Objekten innerhalb des Bildes in eine oder mehrere bestimmte Kategorien. Dazu gehören das Extrahieren visueller Merkmale wie geometrischer Formen und Farben bestimmter Bereiche sowie deren Positionen, die Bewegungserkennung und die 3D-Rekonstruktion einer Umgebung [1, S. 4ff].

Die Verarbeitung der Bildinformationen kann dabei durch klassische Bildverarbeitungsalgorithmen erfolgen, die auf Basis festgelegter logischer Operationen arbeiten. Sie kann jedoch auch mithilfe eines neuronalen Netzwerks und mittels Deep- bzw. Machine Learning erfolgen. Die Verarbeitung mithilfe neuronaler Netzwerke bietet sich insbesondere bei besonders komplexen Aufgaben an.

In dieser Projektarbeit soll die Linienverfolgung mithilfe klassischer Bildverarbeitungsalgorithmen realisiert werden. Dazu wird eine Kamera verwendet, deren Bild in der Programmierumgebung Python mit OpenCV ausgewertet wird. Auf die verwendete Hard- und Software wird im Folgenden eingegangen:

## Kamera

Bei diesem Projekt kommt ein sogenannter CMOS-Sensor zum Einsatz. Im Vergleich zu älteren CCD-Sensoren besitzt jeder einzelne Pixel einen Verstärker und eine Auswerteeinheit. Bei den älteren CCD-Sensoren mussten die Helligkeitswerte nacheinander über ein Schieberegister an den Rand des Sensors „schoben“ werden, um dort ausgewertet zu werden [2, S. 130]. Beide Sensortypen nutzen zur eigentlichen Detektion der Helligkeit in der Regel einen Photokondensator. Dieser Photokondensator ist bei modernen CCD-/CMOS-Sensoren zumeist als MOS-Kondensator ausgeführt [2, S. 127].

Ein Bild, das Text, Screenshot, Gerät, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: Prinzipskizze einer CCD-Zelle bzw. eines Photokondensators mit einem unter der Isolationsschicht ausgebildetem Potentialtopf (vgl.[2]),[15]

Bei dieser Art von Kondensator wird bei der Herstellung zwischen zwei Elektroden eine p-dotierte Halbleiterschicht ausgebildet, wobei eine der Elektroden durch eine Siliziumdioxid-Schicht vom p-Si isoliert ist. Trifft nun ein Photon auf die dotierte Siliziumschicht, entsteht durch den inneren Photoeffekt ein freies Elektron. Dieses wird durch seine negative Ladung zur positiv geladenen Elektrode wandern, jedoch zuvor durch die nicht elektrisch leitende Siliziumdioxid-Schicht abgefangen. Werden nun durch Photoneneinstrahlung noch mehr Elektronen frei, bildet sich unter der Isolationsschicht ein sogenannter Potentialtopf. Diese Ladungsmenge an Elektronen wird jeweils nach einer bestimmten Belichtungszeit durch eine Auswerteelektronik ausgewertet. Durch den linearen Zusammenhang zwischen Ladungsmenge und der Anzahl eingetroffener Photonen kann nun genau abgeschätzt werden, wie viel Licht an welcher Stelle auf den Sensor getroffen ist. ([2], S. 127)

Vorteile des CCD/CMOS-Sensors sind die hohe Linearität und Effizienz bei der Umsetzung von Photonen. ([3], S. 22) Weiterhin kann durch das Einbringen von optischen Filtern zwischen Licht verschiedener Wellenlängen unterschieden werden (Farbbild).

Der in diesem Projekt verwendete Kamerasensor ist ein „Sony IMX317“. Dieser CMOS-Sensor bietet eine Auflösung von 3864 x 2218 Pixeln bei einer Sensorgröße von 10,7 x 8,5 mm. Sämtliche Pixel sind mit Farbfiltern überzogen, die jeweils nur rotes, blaues und grünes Licht passieren lassen. Diese sind in einem bestimmten Mosaikmuster angeordnet und ermöglichen so die Aufnahme eines farbigen Bilds mit 12 Bit Farbtiefe. ([4]) Würden für die Bildauswertung alle zur Verfügung stehenden Pixel verwendet werden, müssten pro Sekunde mehrfach mehr als 8 Millionen 12-Bit-Farbwerte ausgewertet werden. Selbst sehr leistungsfähige Hardware kommt dabei schnell an ihre Grenzen. Der verwendete Sensor stellt deshalb schon auf dem Chip verschiedene Modi zur Verfügung, wie Pixelwerte so zusammengefasst werden, dass die Auflösung signifikant reduziert wird, zeitgleich aber möglichst wenige Bildinformationen verloren gehen. Die verschiedenen Modi ähneln dabei stark dem in der Vorlesung vorgestellten Pooling.

Ein Bild, das Text, Screenshot, Diagramm, Farbigkeit enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: Die verschiedenen Auswertemodi vgl. [4] S.55

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: Die verschiedenen Auswertemodi vgl. [4]S.54

Durch zeitgleiches herabsetzten der Farbtiefe auf 10 Bit, können so signifikant geringere Datenraten erreicht werden. Der zu verwendender Modus kann später softwareseitig vorgegeben werden und muss die jeweiligen Anforderungen angepasst werden.

Die hier verwendete Kamera verfügt neben dem eigentlichen Kamerasensor noch über einen Microcontroller, welcher eine Kommunikation über eine USB 2.0 Schnittstelle ermöglicht.

## Objektiv & Abbildungsfehler

Um ein scharfes Bild von einem Objekt aufzunehmen ist neben dem Seensorarray ein Objektiv notwendig. Dieses hat verschiedene Aufgaben. Der wichtigste Faktor bei einem Objektiv ist die Brennweite. Sie ist so zu wählen, dass von einem Objekt in Entfernung g (Gegenstandsweite) eine Scharfe Abbildung auf dem Sensor abgebildet wird, welcher sich in Entfernung b (Bildweite) hinter dem Objektiv befindet. Über geometrische Zusammenhänge kann berechnet werden, ein wie großes Objekt bei fester Sensorgröße abgebildet werden kann. Dies wird im Folgenden für die Verwendeten Komponenten berechnet:

Ein Bild, das Diagramm, Reihe, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: Bildbereich des Roboters [Eigenanfertigung]

Die allgemeine Linsengleichung bestimmt das Verhältnis zwischen Brennweite (f), der Bildweite (b) und der Gegenstandsweite (g). ([5], S.18). Sie lautet:

Die Abbildungsgleichung stellt zusätzlich den Zusammenhang zur Bildgröße (B) und Gegenstandsgröße (G) her. ([5], S.18). Sie lautet:

Wird (2) nach b umgestellt und in (1) eingesetzt kann damit die unbekannte Größe der Bildweite eliminiert werden. Es Folgt:

Mit Formel (3) kann nun berechnet werden, ein wie breites Bild mit den verwendeten Komponenten aufgenommen und verarbeitet werden kann.

Die Brennweite (f) ist in bei mittlerer Einstellung mit 5mm bekannt. Ebenso die Gegenstandsweite (g), also die Entfernung der Hauptebene des Objektivs und der Fahrbahn. Sie beträgt 120mm. Die Bildgröße ist durch die Sensorbreite vorgegeben und wird im Datenblatt mit 10,7mm angegeben.

Die Rechnung zweigt, dass mit dem verwendeten Aufbau in 120mm Entfernung ein Bereich von 246,1mm scharf erfasst werden kann. Bei einer Linienbreite von ca. 20mm erscheint dies ein guter Wert.

Weitere Aufgabe eines Objektivs ist es, Linsenfehler minimal zu halten. Bei einfachen Linsen treten in der Regel immer Öffnungsfehler bzw. sphärische Aberration auf, welche in bestimmten Bereichen eines Bildes zu unscharfen Abbildungen führen. ([6], S.165ff.).

Noch gravierender sind bei der Bildverarbeitung sogenannte Verzeichnungs-bzw. Distorsion-Fehler. Diese sorgen an den Rändern der Abbildung für einen veränderten Abbildungsmaßstab. Dadurch können geometrische Berechnungen zwischen Abbildung und der Realität stark fehlerbehaftet sein. ([6], S.171ff]).

Qualitativ hochwertige Objektive arbeiten deshalb mit mehreren Linsen hintereinander, welche den Zweck haben, dass sich die Abbildungsfehler der einzelnen Linsen gegenseitig aufheben. ([6] S.154).

## Bildauswertung mit OpenCV

OpenCV ("Open Source Computer Vision Library") ist eine von Intel entwickelte Software-Bibliothek zur Computer-Vision- und Bildverarbeitung. Sie enthält fundamentale Funktionen und Algorithmen, beispielsweise zur Kanten-, Farb- oder Konturerkennung, jedoch auch komplexe Algorithmen zur Objekt-, Gesichts- und Bewegungserkennung. Sie ist für eine Vielzahl von Programmiersprachen wie Java, C++ und dem hier verwendeten Python verfügbar ([7], S. 1ff).

### OpenCV-Bildformat

Bilder werden in OpenCV als Pixelmatrix bearbeitet ([7], S. 12). Farbige Bilder werden pro Pixel mit jeweils einem Helligkeitswert für die Farben Rot, Grün und Blau dargestellt. Die Auflösung dieses Farbwertes kann eingestellt werden, wodurch sich verschiedene sogenannte Farbräume ergeben. Bilder werden jedoch oft auch nur mit einem Helligkeitswert pro Pixel definiert, wodurch sich ein Graustufenbild ergibt. Da ein einfacher Helligkeitswert einfacher zu verarbeiten ist, kommen Graustufenbilder in der Bilderkennung oftmals zum Einsatz, wenn Farbunterschiede nicht von Bedeutung sind.

Da auch in diesem Projekt eine schwarze Linie auf weißem Hintergrund erkannt werden soll, wird das Kamerabild in ein Graustufenbild umgewandelt. In OpenCV werden Einzelbilder, also auch eine Kamera stream, Bild für Bild verarbeitet.

Geht es darum, Linien bzw. Fahrspuren zu erkennen, müssen aus dem umfangreichen Softwareportfolio von OpenCV geeignete Funktionen, Filter und Algorithmen ausgewählt werden, um ein bestmögliches Ergebnis zu erzielen. Im Folgenden werden die wichtigsten verwendeten Funktionen erläutert, um deren Aufgabe in der Gesamtsoftware aufzuzeigen.

### Vorbereitung und Filterung des Bildes

Bei vielen Anwendungsfällen in der Bildverarbeitung empfiehlt es sich, vor der weiteren Verarbeitung des Bildes ein sogenanntes Schwellwertverfahren (Threshold) durchzuführen. Dabei werden Pixel, die eine festgelegte Helligkeitsschwelle über- oder unterschreiten, entweder auf null oder den Maximalwert gesetzt. Ziel ist es oftmals, zusammenhängende Pixelbereiche zu finden, die vorher geringfügig unterschiedliche Helligkeitswerte aufgewiesen haben. So können besonders gut Linien oder andere geometrische Formen in einem Bild erkannt werden.

Ein Bild, das Screenshot, Text, Quadrat, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: Die verschiedenen Threshold-Modi vgl. [[9]

Da die zuvor gezeigten Schwellwertverfahren mit festen Grenzen arbeiten, eignen sie sich optimal für Szenarien, in denen immer gleiche Umgebungsbedingungen herrschen, z. B. in einer stets beleuchteten Halle. Handelt es sich jedoch um eine Umgebung wie z. B. den Straßenverkehr mit stets wechselnden Umgebungshelligkeiten, kommt ein Schwellwertverfahren mit festen Grenzen schnell an seine Grenzen. Für diesen Fall bietet OpenCV die Funktion adaptiveThreshold, welche die Grenzen des Verfahrens stets an die globalen Helligkeitswerte anpasst [9, Kap. Adaptive Thresholding].

Ein weiterer, für die Bildvorverarbeitung nützlicher Filter ist der Gauss-Filter. Dieser kann durch seine Beschreibung als e-Funktion mit negativen Exponenten als Tiefpass angesehen werden. Er glättet somit „hohe Frequenzen“ in einem Bild. Dieser Effekt äußert sich durch eine Weichzeichnung des Bildes. Ziel ist es, einzelne „Störpixel“, die beispielsweise durch Bildrauschen entstanden sind, so in der Helligkeit zu vermindern, dass sie durch das Schwellwertverfahren herausgefiltert werden können.

Ein Bild, das Text, Entwurf, Diagramm, Zeichnung enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: Darstellung des Adapive Thresholding und des Gauss-Filters vgl. [9]

### Detektion der Linie

Da eine Linie in einem Bild in der Regel von mehreren zusammenhängenden Pixeln gebildet wird, muss ein Algorithmus zur Anwendung kommen, der diese Ansammlungen von Pixeln erkennen kann. OpenCV bietet dazu die Klasse findContours() an [8]. Diese arbeitet nach dem Suchalgorithmus von Satoshi Suzuki [10]. Der Algorithmus geht von links nach rechts und von oben nach unten durch das Bild und kontrolliert, ob ein Pixel bereits einer Kontur angehört. Ist dies nicht der Fall, wird geprüft, ob benachbarte Pixel mit dem jeweiligen Pixel verbunden sind. Trifft dies zu, erhält das Pixel einen Zusammenhangscode. Am Ende des Durchlaufs wird überprüft, ob zusätzliche Verbindungen zwischen unterschiedlichen Zusammenhangscodes existieren. Falls ja, werden diese in einen gemeinsamen Zusammenhangscode zusammengefasst.

### Auswertung der Konturen

Sind in einem Bild alle Konturen gefunden, die für eine Liniendetektion von Interesse sein könnten, müssen diese klassifiziert und ausgewertet werden. Ein erster entscheidender Faktor ist dabei die Größe der Kontur. Diese kann mit der Funktion cv.contourArea() abgerufen werden [8]. Da in den meisten Fällen die ungefähre Länge und Breite der zu detektierenden Linie bekannt ist, können über diesen Parameter Konturen herausgefiltert werden, die für die weitere Auswertung von Interesse sind.

Zur weiteren Eingrenzung kann die Funktion cv.minAreaRect() angewendet werden. Diese Funktion rahmt die Kontur in einem Rechteck ein. Von besonderem Interesse ist dabei der Parameter angle of rotation. Dieser Wert gibt Aufschluss darüber, in welchem Winkel relativ zur Kamera die gefundene Kontur verläuft. Im Zusammenhang mit dieser Funktion kann es sich ebenfalls anbieten, das Bild in verschiedene Sektionen zu zerlegen und jede Sektion einzeln auszuwerten. Dies kann vor allem bei stark kurvigen Linien von Vorteil sein.

Neben dem Winkel der Linie sind jedoch vor allem auch die x- und y-Position der gefundenen Kontur von Bedeutung, damit das Fahrzeug weiß, ob es auf oder neben der Linie fährt und sich der Linie nähert oder sich von dieser entfernt. Besonders vorteilhaft ist dabei die Berechnung des Konturschwerpunkts mit Bild-Momenten. Dazu werden die Helligkeitswerte der Konturen mit ihrer Position verrechnet und über die gesamte Kontur der Schwerpunkt bestimmt. Ein großer Vorteil dieser Methode ist, dass auch bei einer lückenhaften oder verrauschten Kontur noch ein guter Schwerpunkt gefunden werden kann, der in der Realität oft in der Linienmitte liegt. Die Berechnung erfolgt dabei nach den folgenden Formeln [11].

Vor allem bei der Berechnung von Konturenschwerpunkten mit der Bild-Momente-Methode kann es sich anbieten, ein Bild in verschiedene Sektionen aufzuteilen. Da in diesem Fall die Linie vorwiegend vertikal durch das Bild verläuft, wird dieses in mehrere horizontalen Bildstreifen zerlegt, welche jeweils einen Teil der Linie enthalten. Mit der Momente-Methode wird in jedem Streifen der Schwerpunkt berechnet und aus diesen Koordinaten durch eine gewichtete Funktion der Verlauf der Linie rekonstruiert. Dazu werden weiter entfernte Liniensektionen stärker in die Gewichtung für den Lenkbefehl aufgenommen als näher liegende. Dies spiegelt sich in einer vorausschauen deren Fahrweise des Fahrzeugs wider.

### Der Kalman Filter

Der errechnete Lenkbefehl weist trotz komplexer Bildverarbeitung noch ein gewisses Rauschen auf. Dies ist zum einen durch Schattenwurf, Verschmutzung der Linie und Kamerarauschen zurückzuführen. Zum anderen auf Vibrationen und Schwingungen, die durch das Fahrwerk des Zumo-Roboters hervorgerufen werden. Diesem Umstand kann optimal mit dem Kalman-Filter begegnet werden. Der Kalman-Filter glättet nicht nur das unter Umständen verrauschte Positionssignal, sondern bildet eine Art Gedächtnis aus vorherigen Messwerten, um zukünftige Messwerte vorherzusagen. OpenCV bietet für die Realisierung eines Kalman-Filters die Klasse KalmanFilter() an. Zur Initialisierung müssen dazu die Dimension der Zustandsmatrix und des Messwerts angegeben werden. Zudem müssen die „measurementMatrix“ und die „transitionMatrix“ definiert werden. Während der Messung kann dann mit „kalman.predict()“ und „kalman.correct()“ der Messwert vorhergesagt bzw. korrigiert werden.[12]

Tests zeigten ein deutlich verbessertes Verhalten, vor allem bei schlechten Lichtverhältnissen und stark kurvigen Linien. Sogar lückenhafte Linien konnten mit diesem Filter nahezu zuverlässig erkannt werden.

### Visualisierung der Bildverarbeitung

OpenCV bietet die Möglichkeit, das aktuelle Bild nach jedem Schritt der Bildverarbeitung auszugeben bzw. anzuzeigen. Dies kann von Vorteil sein, um die Qualität der Erkennung zu beurteilen und mögliche Fehler zu erkennen. Die Funktion imshow() gibt dabei das aktuelle Bild in einem von OpenCV generierten Fenster aus. Zur weiteren Analyse bietet es sich an, gefundene Konturen mit dem Befehl drawContours() farblich im Bild hervorzuheben. Weiterhin kann es sinnvoll sein, ermittelte Koordinaten mit circle() in das Bild einzufügen. Möchten andere Nodes im Robot Operating System (ROS2) das erkannte Bild erhalten, muss dieses in einem ROS2-kompatiblen Format versendet werden. Dazu kann auf die Funktion cv2\_to\_imgmsg() zurückgegriffen werden.

## Validierung der Robustheit der Linienerkennung

Um die Robustheit der Linienerkennung zu beurteilen, werden vier Umgebungsszenarien simuliert. Das erste Szenario findet in einer von Fensterlicht beleuchteten Umgebung statt. Zur Beurteilung der Umgebungshelligkeit wird der Lichtstrom pro Fläche in Lux vermessen. Im ersten Szenario wird ein Wert von 530 gemessen. Im zweiten Szenario wird ein abgedunkelter Raum mit schwacher LED-Beleuchtung erhellt, was einem Wert von 150 Lux entspricht. Im dritten Szenario wird der Raum so weit abgedunkelt, dass mit dem menschlichen Auge gerade noch helle von dunklen Flächen unterschieden werden können. Dies könnte in etwa Mondlicht bei Nacht entsprechen. Es wird ein Wert von 10 Lux gemessen. Das letzte Szenario 4 findet bei gleicher Helligkeit wie Szenario 1 statt, die Linie wird jedoch stark verschmutzt, um eine robuste Erkennung zu beurteilen.

Alle Szenarien werden mit zwei unterschiedlichen Erkennungsmethoden vermessen. Methode 1 nutzt dabei einen Threshold mit festen Grenzen und keine zusätzlichen Filteralgorithmen. Methode 2 wendet neben einem Gaussfilter auch den adaptiven Threshold an, der sich ständig an das Umgebungslicht anpasst (siehe Kap. 2.3.2).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Umgebungslicht (530 Lux) | LED-Beleuchtung(150 Lux) | Schwaches Umgebungslicht (10 Lux) | Verschmutzte Linie (530 Lux) |
| Erkennung  ohne Filter und fixen Threshold-  Grenzen |  |  |  |  |
| Erkennung mit Filtern und Adaptive-  Threshold |  |  |  |  |

Zur Auswertung wird von jeder Messmethode die Erkennung in der jeweiligen Messumgebung visualisiert und die Ergebnisse über das farbige Ursprungsbild gelegt.

Es fällt auf, dass beide Erkennungsmethoden bei gut ausgeleuchteten und sauberen Linien wenig Probleme haben. Bei schwachen Lichtverhältnissen kann der adaptive Threshold jedoch erstaunliche Ergebnisse liefern. Durch die kontinuierliche Anpassung an die globalen Helligkeitswerte konnte die Linie selbst bei minimalem Restlicht erkannt werden.

Bei der Verschmutzung der Linie konnte vor allem der Gauss-Filter gute Ergebnisse liefern. Durch sein Verhalten, das mit einem Tiefpass verglichen werden kann, wurden vor allem kleine Störungen, wie die Zuckerkörner, effektiv herausgefiltert. Größere Bereiche, die vom Algorithmus nicht erkannt wurden, fallen durch die bildmomentbasierte Berechnung (siehe 2.3.4) meist nur wenig ins Gewicht. Sollte die Linie unterbrochen sein, wird dies durch die fünf Segmente bis zu einem bestimmten Grad kompensiert.

## Anwendung der Bildverarbeitung und Linienerkennung bzw. -verfolgung auf praktische Anwendungen

Die Bildverarbeitung autonomer und teilautonomer Systeme gewinnt immer mehr an Bedeutung. Dies ist vor allem darauf zurückzuführen, dass mit Kameras eine umfassende Wahrnehmung der Umgebung realisiert werden kann. Hinzu kommt, dass die Preise für Kamerasensoren und die zur Auswertung notwendige Rechenkapazität in den letzten Jahrzehnten massiv gefallen sind. Dies macht das Verfahren vor allem im Vergleich zu deutlich teureren Sensoren sehr attraktiv. Der Autohersteller Tesla beispielsweise verfolgt seit 2021das Ziel, autonomes Fahren ausschließlich mit kamerabasierter Erkennung zu realisieren.[14] nützlich sind dabei die enormen Fortschritte im Bereich der künstlichen Intelligenz und neuronalen Netzwerke, die bei der Verarbeitung großer Datenmengen völlig neue Möglichkeiten eröffnen.

Die Linienerkennung bzw. -verfolgung wird bereits seit Längerem im Automotive-Bereich eingesetzt. Mit dieser Technik wurden bisher Spurhalteassistenten realisiert, die ein unbeabsichtigtes Verlassen der Fahrspur verhindern sollen. In Zukunft wird die Linien- bzw. Fahrspurerkennung jedoch eine weitaus größere Rolle spielen, wenn es darum geht, höhere Autonomiegrade von Fahrzeugen zu erreichen. Ein weiterer großer Vorteil der Linienerkennung besteht darin, dass Linien bzw. Markierungen mit geringstem Aufwand und minimalen Kosten in nahezu alle Umgebungen integriert werden können, ohne beispielsweise Straßen mit Sensoren oder Signalgebern auszustatten. [1]

Vor allem aufgrund der geringen Kosten stößt die Linienerkennung auch in der Industrie auf großes Interesse. Während für die Navigation autonomer Transport- und Sortierroboter früher die Umgebung oder jeder Roboter mit kostspieliger Sensorik und Aktorik ausgestattet werden musste, kann beispielsweise eine Lagerhalle durch das Aufmalen von Fahrspuren und Landmarken in kürzester Zeit für autonome Systeme hergerichtet werden. [13]

## Grenzen und Nachteile der kamerabasierten Bildverarbeitung

Bei all den Vorteilen müssen jedoch auch die Grenzen und Nachteile der kamerabasierten Bildverarbeitung berücksichtigt werden. Da es sich um eine passive Messmethode handelt, die – anders als Lidar und Radar – ihr eigenes Messmedium nicht aktiv ausstrahlt, ist sie stark von den Lichtverhältnissen der Umgebung abhängig. Obwohl diesem Umstand mit hochempfindlichen Kamerasensoren und einer entsprechenden Ausleuchtung entgegengewirkt werden kann, zeigen die Tests in Kapitel 2.4, dass die kamerabasierte Bildverarbeitung ihre Grenzen hat.

Darüber hinaus ist der Einsatz von Kamerasensoren oftmals stark auf das jeweilige Anwendungsszenario anzupassen. Dieser erhöhte Entwicklungsaufwand lohnt sich häufig nur bei hohen Stückzahlen. Hinzu kommt, dass Kamerasensoren direkt keine Entfernungswerte liefern. Diese können nur abgeschätzt oder durch ein zweites, versetztes Kamerabild berechnet werden.

Ein Bild, das Maschine, Grün, Straße, Sauberkeit enthält.

AI-generated content may be incorrect.

Abbildung .: Industrieroboter mit visueller Linienverfolgung[13]

# GUI-Toolkits

Ein GUI-Toolkit ist eine Programmbibliothek, die das Erstellen grafischer Benutzeroberflächen unterstützt. Sie bietet eine Sammlung von Steuerelementen, sogenannten Widgets, wie Buttons, Eingabefelder und Layouts. Durch diese Widgets wird die GUI-Entwicklung erheblich erleichtert. Es gibt verschiedene GUI-Toolkits und Designwerkzeuge, die je nach Projektanforderungen verwendet werden können (2).

Eine der bekanntesten und am häufigsten verwendeten Bibliotheken ist „Qt“, zusammen mit dem Qt Designer, einem grafischen Tool, das die Erstellung von Benutzeroberflächen in einer visuellen Umgebung ermöglicht.

Qt Designer 5 kann direkt durch den folgenden Befehl auf Linux Betriebssystem installiert werden:

**“sudo apt-get install qttools5-dev-tools “**

## Designwerkzeug „Qt Designer“

Qt Designer generiert .ui-Dateien, die ein spezielles XML-basiertes Format verwenden. Diese Dateien speichern die Struktur der Benutzeroberfläche in Form eines Widget-Baums. Die .ui-Dateien können entweder direkt zur Laufzeit geladen oder in Programmiersprachen wie C++ oder Python übersetzt werden. Der resultierende Code enthält die notwendigen Funktionen und Ereignisbehandlungsroutinen, um die GUI zum Laufen zu bringen und Benutzerereignisse zu verarbeiten (2).

In Qt Designer stehen viele Steuerelemente zur Verfügung. Die wichtigsten Elemente sind „Frames“, die andere Widgets gruppieren, sowie „Steuerelemente (Controls)“ wie Buttons und Checkboxen zur Benutzerinteraktion. „Inputelemente“, beispielsweise Text-Edit, ermöglichen die Eingabe und Anzeige von Text.

Eine besondere Rolle spielen die „Layouts (Sizer)“, die dafür sorgen, dass die Anordnung der Widgets dynamisch auf die Fenstergröße reagiert. Um eine flexible und anpassungsfähige Benutzeroberfläche zu gestalten, müssen Layouts häufig miteinander verschachtelt werden. Dies stellt eine Herausforderung dar und erfordert eine durchdachte Planung, um die Platzierung der Elemente optimal zu gestalten. Dabei stehen verschiedene Layout-Typen zur Verfügung, darunter „Vertical Layout“ für vertikale Anordnungen, „Horizontal Layout“ für horizontale Ausrichtungen und das „Grid Layout“, das eine flexible Positionierung der Widgets in einem Gitter ermöglicht (2).

### Interface Schnittstelle zu ROS 2

Der Code des Projekts implementiert eine ROS 2 Node-basierte Qt-Anwendung „Interface\_node“ zur Steuerung eines Zumo-Roboters und zur Anzeige des Kamerafeeds in einer grafischen Benutzeroberfläche. Die Anwendung wurde mit PyQt5 und Qt Designer entwickelt. Um das Interface zum Laufen zu bringen, sind zwei Klassen erforderlich: die „ROS2Thread-Klasse“ und die „MainWindow-Klasse“.

Die „ROS2Thread-Klasse“ startet die ROS 2 Node in einem separaten Thread. Dies ist notwendig, um zu verhindern, dass der GUI-Event-Loop durch die ROS-Kommunikation blockiert wird. Ohne Threading würde die GUI einfrieren, wenn ROS-Nachrichten empfangen werden. Die Methode „run()“ wird im Hintergrund ausgeführt, während „start()“ den Thread tatsächlich startet und die Verarbeitung der ROS 2-Nachrichten übernimmt.

Die „MainWindow-Klasse“ ist für die Benutzeroberfläche verantwortlich. Sie verbindet die GUI-Elemente mit der ROS 2-Kommunikation und sorgt für die Anzeige der Sensordaten. Wichtige Methoden in dieser Klasse sind:

* „show\_encoder\_data()“: zeigt die die Encoder-Daten des Roboters an.
* „show\_pose\_data()“: stellt die aktuelle Position und Orientierung des Roboters dar.
* „show\_display\_video()“: zeigt den Kamerastream des Roboters in der GUI an.

Die Umsetzung dieser Klasse basiert auf vier grundlegenden Prinzipien. Ein zentrales Prinzip ist die Erfassung und Darstellung der Sensordaten. Dazu werden verschiedene ROS-2-Topics abonniert, wobei je nach Art der Sensordaten unterschiedliche Nachrichtentypen zum Einsatz kommen. Beispielsweise wird der Nachrichtentyp Int32MultiArray für Encoder-Daten verwendet, während Image für die Übertragung von Kamerabildern genutzt wird.

Ein weiteres wichtiges Prinzip in der Umsetzung ist das sogenannte Rate Limiting, das über die Methode „rate\_limit()“ umgesetzt wird. Diese Methode sorgt dafür, dass die Daten nur in bestimmten Zeitintervallen – beispielsweise alle 0,5 Sekunden – aktualisiert werden. Dies schont Ressourcen und gewährleistet eine flüssige Darstellung der Daten in der Benutzeroberfläche.

Eine besondere Herausforderung war die Integration von OpenCV und CvBridge. OpenCV wird verwendet, um Bilder aus ROS 2 zu verarbeiten, während CvBridge das Konvertieren zwischen ROS-Image-Nachrichten und OpenCV-Bildern übernimmt. Die Bilder werden zunächst im BGR-Format empfangen, mit OpenCV in das RGB-Format konvertiert und anschließend als QImage in der Qt-Oberfläche angezeigt.

Der Letzte Prinzip dies Klassen sind Motoren zu steuern, daher Benutzeroberfläche bietet zudem drei Hauptfunktionen:

* „send\_start\_command()“: aktiviert den Roboter.
* „send\_stop\_command()“: hält ihn an.
* „send\_reset\_command()“: setzt alle Werte wie Encoder-Daten, Geschwindigkeiten und Positionsdaten zurück.

Diese Befehle werden über drei Schaltflächen ausgelöst (s. Abbildung ‎4.1). Die Methoden senden jeweils die entsprechenden Befehle über das ROS 2-Topic „robot\_command“. Die Oberfläche ist so gestaltet, dass sie zu einem späteren Zeitpunkt um Parameter erweitert werden kann.

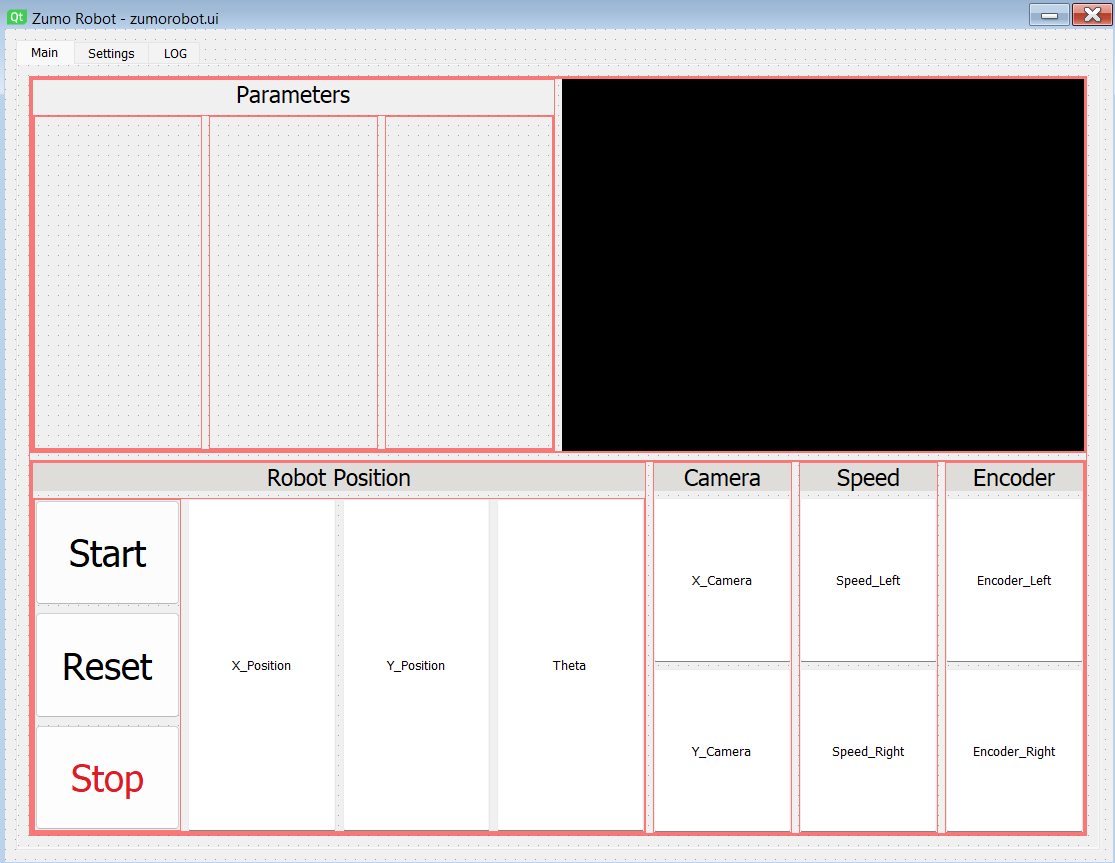


Abbildung .:GUI-Interface

# PID-Regler zur Linienverfolgung

Um eine optimale Linienverfolgung zur Steuerung des Systems zu gewährleisten, ist der Einsatz eines PID-Reglers (Proportional-Integral-Derivative Controller) unerlässlich. Dieser Regler dient dazu, den Roboter so zu steuern, dass er einer definierten Linie folgt, indem er kontinuierlich die Abweichung von der Sollposition misst und entsprechende Korrekturen an den Motoren vornimmt.

## Grundlagen des PID-Reglers

Der PID-Regler basiert auf der kontinuierlichen Berechnung eines Fehlers e(t), der als Differenz zwischen der gewünschten Zielposition „target position“ und die aktuelle gemessene Position „measured position“ definiert. Das Ausgangssignal des Reglers u(t) wird anhand folgender Gleichung bestimmt:

Hierbei stehen:

* **kp** für den Proportionalfaktor, der die unmittelbare Reaktion des Systems auf den Fehler beschreibt. Eine hohe Verstärkung kp führt zu einer schnellen Reaktion des Systems, kann jedoch auch Schwingungen verursachen, wenn der Wert zu groß gewählt wird (3).
* **ki** für den Integralanteil, der vergangene Fehler summiert und somit stationäre Abweichungen eliminiert. Ein hoher ki-Wert reduziert systematische Abweichungen, kann jedoch zu einem verzögerten Ansprechverhalten und stärkeren Überschwingern führen(3).
* **kd** für den Differenzialanteil, der die Änderungsrate des Fehlers betrachtet, um unerwünschtes Überschwingen zu reduzieren. Ein hoher kd-Wert kann dazu beitragen, Schwankungen und Überschwinger zu reduzieren, macht das System jedoch anfälliger für Signalrauschen(3).

Die folgende Tabelle zeigt die Auswirkungen der PID-Parameter auf das Verhalten des Systems:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Anstiegszeit** | **Überschwingen** | **Einschwingzeit** | **Stationärer Fehler** |
| **kp** | Verringert | Erhöht | Geringe  Änderung | Verringert |
| **ki** | Verringert | Erhöht | Erhöht | Verringert |
| **kd** | Kaum Veränderung | Verringert | Verringert | Keine Änderung |

Tabelle .: Parameterwirkung auf den Regler(3)

## ****Anwendung im geschlossenen Regelkreis****

Das Steuerungssystem des Zumo-Roboters basiert auf einem geschlossenen Regelkreis (s. Abbildung ‎5.1), bei den Sensoren kontinuierlich Daten erfassen und dem Regler Rückmeldungen zur Systemabweichung liefern. In diesem Fall übernimmt eine Kamera die Aufgabe der Sensorik, indem sie die Position des Roboters relativ zur schwarzen Linie erfasst. Der Zielpunkt entspricht der Mitte der Linie, sodass der Regler darauf abzielt, Abweichungen vom Pfad zu minimieren.

Zu jedem Zeitpunkt wird die Differenz zwischen der aktuellen gemessenen Position und der Zielposition berechnet. Diese Differenz dient als Eingangsgröße für den Regler, der daraufhin die Motorsteuerung anpasst, um den Roboter auf die Linie zurückzuführen. Ist die gemessene Position beispielsweise 400 und die Zielposition 334, so berechnet der PID-Regler anhand dieses Fehlers eine entsprechende Anpassung der Motorleistung, wodurch sich der Roboter in Richtung der korrekten Spur bewegt.

Dieses Verfahren läuft kontinuierlich ab, wobei die Regelung mit einer festen Frequenz aktualisiert wird. Dadurch wird eine dynamische Anpassung des Antriebssystems gewährleistet, was eine stabile und präzise Linienverfolgung ermöglicht (4).

Ein Bild, das Text, Reihe, Quittung, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: PID-Regelkreis

## PID-Tuning zur Optimierung der Steuerung

Um eine optimale Steuerung zu gewährleisten, müssen die Verstärkungsparameter (kp, ki, und kd) abgestimmt werden. Dies kann durch verschiedene Methoden erfolgen. Eins davon, die hier verwendet:

**Trial-and-Error (Versuch-und-Irrtum-Methode)** (4)**:**

* + Zunächst werden **ki** und **kd** auf **0** gesetzt.
  + Der Wert von **kp** wird erhöht, bis das System zu schwingen beginnt.
  + Anschließend wird **ki** schrittweise erhöht, um stationäre Fehler zu reduzieren.
  + Zum Schluss wird **kd** angepasst, um überschießende Reaktionen zu dämpfen.

Nach dem Tuning wurden die folgenden optimalen Parameterwerte ermittelt:

Zudem wurde zusätzlich die Ableitungskomponente (kd) mit einer Low-Pass-Filterung stabilisiert, um eine stabile und glatte Steuerung zu gewährleisten. Darüber hinaus wurde die Motorsteuerung auf 125 [rpm] begrenzt, sodass die Geschwindigkeit stets innerhalb des zulässigen Bereichs bleibt.

## Kommunikationsprotokoll zwischen PC und Arduino

Zur Steuerung der Motoren des Zumo-Roboters sowie zur Erfassung der Encoder-Daten wurde ein serielles Frame-Protokoll implementiert. Die Kommunikation zwischen dem ROS-2-System und dem Arduino erfolgt über eine serielle Schnittstelle mit einer Baudrate von 115200, wobei das Little-Endian-Format zur Übertragung genutzt wird.

### Datentypen der übertragenen Nachrichten

Die über das Protokoll übertragenen Daten weisen die in Tabelle ‎5.2 dargestellten Formate auf:

|  |  |
| --- | --- |
| **Nachricht** | **Datentyp** |
| Motorsteuerung (left, right) | Int16 |
| Encoder-Werte (left, right) | Int32 |
| Kamera-Koordinaten (x,y) | float |
| Log-Nachrichten | string |
| Kamera-Frame | Image |
| Roboterposition | Pose |

Tabelle .: Datentyp der Nachrichten

### Datenstruktur des Motorsteuerungs-Frames

Der Steuerbefehl zur Ansteuerung der Motoren folgt einer fest definierten Struktur, die in Tabelle ‎5.3 dargestellt ist:

|  |  |  |  |
| --- | --- | --- | --- |
| **Start of Frame** | **Motor Data** | **Control Byte** | **End of Frame** |
| 1 Byte | 4 Byte | 1 Byte | 1Byte |

Tabelle .: Motorsteuerungs-Frames

Die Nachricht wird von ROS 2 an den Arduino gesendet und besteht insgesamt aus 7 Byte. Die genaue Bedeutung der Felder ist wie folgt:

* Start-of-Frame (1 Byte): Definiert den Beginn des Datenpakets, um eine korrekte Synchronisation sicherzustellen.
* Motor-Daten (4 Byte): Enthält die Zielgeschwindigkeit der beiden Motoren als 16-Bit-Ganzzahlen (Int16) im Little-Endian-Format.
* Steuer-Byte (1 Byte): Enthält Befehle zur Steuerung des Roboters, beispielsweise Start, Stopp oder Reset.
* End-of-Frame (1 Byte): Markiert das Ende der Nachricht.

Der Motors-Node in ROS 2 berechnet die benötigten Geschwindigkeiten der Motoren basierend auf den empfangenen Kameradaten (z. B. Linienkoordinaten) und sendet diese über die serielle Schnittstelle an den Arduino (vgl. motors\_node.py). Die Implementierung nutzt dabei das PID-Regelungsverfahren, um eine stabile Nachführung der gewünschten Trajektorie zu gewährleisten.

### Datenstruktur des Encoder-Daten-Frames

Der Arduino überträgt die erfassten Encoder-Werte in einem separaten Frame an ROS 2. Die Struktur dieses Frames ist in Tabelle ‎5.4 dargestellt:

|  |  |  |
| --- | --- | --- |
| **Start of Frame** | **Encoder Data** | **End of Frame** |
| 1 Byte | 8 Byte | 1 Byte |

Tabelle .:Encoder-Frames

Die gesendeten 10 Byte enthalten folgende Informationen:

* Start-of-Frame (1 Byte): Markiert den Beginn des Datenpakets.
* Encoder-Daten (8 Byte): Enthält die aktuellen Messwerte der linken und rechten Encoder als 32-Bit-Ganzzahlen (Int32).
* End-of-Frame (1 Byte): Signalisiert das Ende des Pakets.

Der Encoder-Node in ROS 2 verarbeitet diese Werte und veröffentlicht sie im encoder\_data-Topic, wodurch sie für andere ROS-2-Module verfügbar sind (vgl. encoder\_node.py). Diese Werte werden zur Bewegungsverfolgung und Kartenerstellung des Roboters genutzt.

### Gesamtarchitektur der Kommunikation

Die serielle Kommunikation erfolgt bidirektional: Der Motor-Node verwendet die PID-Regler-Library und sendet nach dem Drücken der Start-Schaltfläche auf der Benutzeroberfläche Befehle zur Motorsteuerung an den Arduino. Zuvor sollten alle Nodes mit der Launch-Datei aktiviert werden. Gleichzeitig empfängt der Encoder-Node die erfassten Bewegungsdaten vom Arduino und veröffentlicht sie zur Erstellung des Pfads für den Roboter. Durch dieses Kommunikationsschema wird eine Echtzeit-Regelung des Roboters innerhalb des ROS-2-Frameworks ermöglicht.

# Protokollierung in ROS 2

Das Protokollierungssystem in ROS 2 ermöglicht die Erfassung und Übermittlung von Log-Nachrichten an verschiedene Ziele, darunter die Konsole, lokale Protokolldateien, sofern ausreichend Speicher vorhanden ist und das „/rosout“-Topic im ROS-2-Netzwerk. Jedes dieser Ziele kann für jeden Knoten individuell aktiviert oder deaktiviert werden (5).

Jede Log-Nachricht ist einem bestimmten Schweregrad zugeordnet, der ihre Bedeutung und Dringlichkeit angibt. Die Schweregrade sind in aufsteigender Reihenfolge: DEBUG, INFO, WARN, ERROR und FATAL. Ein Logger verarbeitet nur Nachrichten, deren Schweregrad mindestens dem für diesen Logger festgelegten Wert entspricht.

Jedem Knoten wird automatisch ein Logger zugewiesen, dessen Name und Namensraum mit denen des Knotens übereinstimmen. Falls der Name eines Knotens extern geändert wird, spiegelt sich dies auch im Namen des zugehörigen Loggers wider. Neben den standardmäßigen Knoten-Loggern können auch unabhängige Logger erstellt werden, die einen expliziten Namen erhalten (5).

Die Loggerstruktur in ROS 2 ist hierarchisch aufgebaut. Falls für einen Logger mit einem bestimmten Namen, beispielsweise „abc.def“, kein eigener Schweregrad definiert wurde, übernimmt er automatisch den Schweregrad seines übergeordneten Loggers „abc“. Falls auch dieser keinen Schweregrad zugewiesen hat, wird der voreingestellte Standardwert genutzt. Änderungen am Schweregrad eines übergeordneten Loggers wirken sich auf alle nachgeordneten Logger aus, sofern diese nicht explizit eine eigene Stufe definiert haben (5).

Das Format der Log-Nachrichten kann angepasst werden. Standardmäßig für dieser Arbeit wird das folgende Format verwendet: [{severity}] [{time}] [{name}]: {message}, wobei {severity} den Schweregrad, {time} den Zeitstempel, {name} den Loggernamen und {message} die eigentliche Nachricht darstellt.

Um eine übermäßige Flut an wiederkehrenden Log-Nachrichten in der Benutzeroberfläche zu vermeiden, wurde eine spezielle Log-Klasse „Log\_node“ implementiert. Sie speichert nur einmalig auftretende, wichtige Nachrichten und zeigt sie in einem speziellen Log-Feld an. Wiederholte Meldungen, etwa zu Kamerabildern oder Motorgeschwindigkeiten, werden bewusst ausgeschlossen, um die Benutzeroberfläche übersichtlich zu halten.

# ROS2 Framework

## Definition & Architektur

Das Robot Operating System 2 (ROS 2) ist eine Sammlung von Softwarebibliotheken und Tools zur Entwicklung von Roboteranwendungen. Es stellt eine Open-Source-Umgebung bereit, die von Hardware-Treibern über hochentwickelte Algorithmen bis hin zu leistungsstarken Entwicklungswerkzeugen reicht (6).

ROS 2 baut auf dem Erfolg seines Vorgängers ROS 1 auf, wurde jedoch in mehreren entscheidenden Aspekten verbessert (s. Abbildung ‎7.1). Eine der bedeutendsten Änderungen ist die Einführung des Data Distribution Service (DDS)-Kommunikationsprotokolls, das eine zuverlässige und flexible Kommunikation zwischen verteilten Systemen ermöglicht. Darüber hinaus wurde das System so gestaltet, dass es echtzeitfähig, sicherer und plattformübergreifend einsetzbar ist (unterstützt Linux, Windows, macOS und Echtzeitbetriebssysteme). Eine weitere zentrale Neuerung ist der Verzicht auf das Master-Node-Prinzip, wodurch ROS 2 als dezentrales System konzipiert wurde (7).

ROS 2-Anwendungen greifen über die Client-Bibliothek ROS Client Library (RCL) auf ROS 2-Funktionen zu. Die Rcl-Bibliothek ist in der Sprache C geschrieben und enthält Rclcpp-Client-Bibliotheken für die Sprache C++ und Rclpy für die Sprache Python. Die Client-Bibliothek ist in erster Linie mit der Standardschnittstelle ausgestattet, die zum Datenaustausch mit Topic- und Service-Ansätzen über ROS 2 erforderlich ist. Darüber hinaus verfügt die ROS2-Bibliothek über Funktionen zur Bereitstellung von Betriebssystemabstraktion (8).

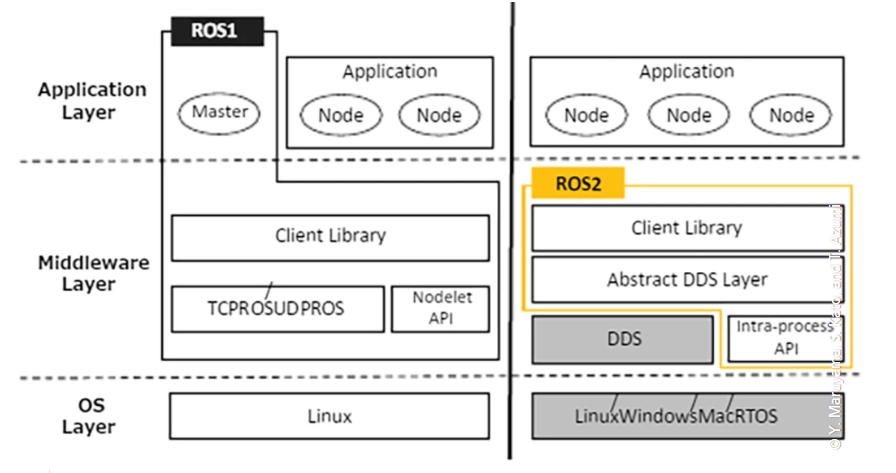


Abbildung .: ROS-Architektur (7)

## Kommunikationsmodellen

Die Architektur von ROS 2 basiert auf zwei wesentlichen Kommunikationsmodellen: Publisher-Subscriber-Modell und Service-Client-Modell (s. Abbildung ‎7.2). Die für dieses Projekt relevanten Hauptkomponenten von ROS 2 sind:

Ein Node (Knoten) ist eine unabhängige Einheit innerhalb eines ROS-2-Systems, die spezifische Aufgaben übernimmt, wie z. B. das Erfassen von Sensordaten oder die Steuerung von Motoren. Jeder Node verwendet eine ROS-2-Client-Bibliothek, um mit anderen Nodes zu kommunizieren. Nodes interagieren über Topics (bei der Publisher-Subscriber-Kommunikation) oder Services (bei der synchronen Anfrage-Antwort-Kommunikation).

ROS 2 nutzt das Publisher-Subscriber-Modell für die asynchrone Datenübertragung. Dabei sendet ein Node (Publisher) Daten über ein Topic, während andere Nodes (Subscribers) diese Daten empfangen. Eine Kommunikation zwischen zwei oder mehreren Nodes ist nur möglich, wenn sie sich auf dasselbe Topic beziehen. Dadurch wird eine lose Kopplung zwischen den Komponenten erreicht, was die Skalierbarkeit des Systems erleichtert.

Neben der asynchronen Kommunikation über Topics bietet ROS 2 auch synchrone Service-Client-Modell. Ein Node kann als Client eine Anfrage an einen anderen Node (den Service-Provider) stellen und eine direkte Antwort erhalten. Dieses Modell eignet sich für Vorgänge, die eine einmalige oder ereignisgesteuerte Kommunikation erfordern, wie z. B. das Anfordern einer aktuellen Sensorablesung oder das Aktivieren eines bestimmten Robotermodus.

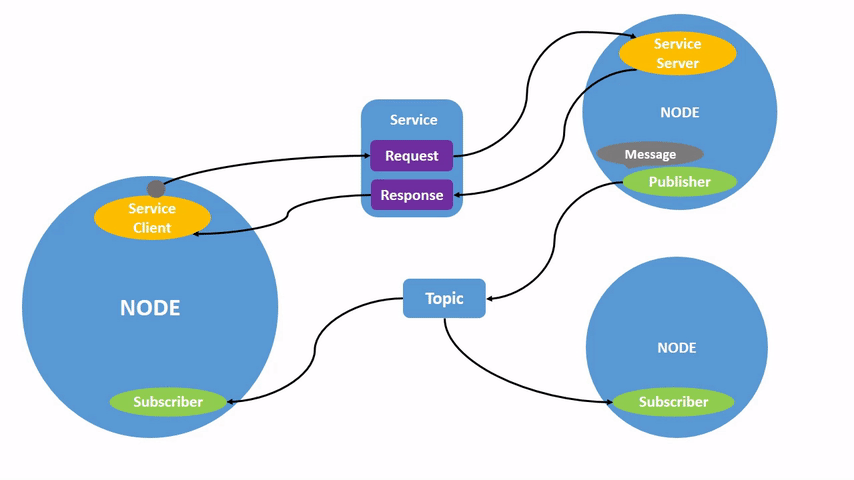


Abbildung .: Node-Modell(9)

## Graphische Darstellung der Kommunikation zwischen Nodes

Einer der Beispiele für die Anwendung dieses Modells (Publisher-Subscriber-Modell) innerhalb des Projekts ist die Kommunikation zwischen der Kamera-Node und der Motors-Node über das Topic "line\_coordinates". Die Kamera-Node erkennt die Position der Linie im Bild (Koordinaten X, Y) und sendet diese über das Topic an die Motors-Node. Diese interpretiert die Daten und passt die Motorsteuerung entsprechend an, um dem Linienverlauf zu folgen.

Die folgende schematische Darstellung veranschaulicht die Kommunikation zwischen den einzelnen Nodes. Jeder Node ist als eine Python-Klasse implementiert und wird über eine zentrale Launch-Datei gestartet, um eine koordinierte Ausführung aller Komponenten zu gewährleisten.

Xxxxxxxxxxxxxxxxx

# Path Mapping: Kartierung der Roboterbewegung

Ein mobiles Robotersystem benötigt nicht nur die aktuelle Position, sondern auch eine Aufzeichnung des zurückgelegten Weges. Dies wird durch die Pfadkartierung (Path Mapping) realisiert, die in der Implementierung des „path\_node“ erfolgt.

## Funktionsweise des Path-Mapping-Nodes

Der „path\_node“ verfolgt die Position des Roboters anhand von Encoderwerten, die von den Rädern stammen. Diese Encoder-Daten werden über das ROS 2-Topic „encoder\_data“ empfangen und in Bewegungsinformationen umgerechnet. Dabei werden die Differenzen der Encoderticks genutzt, um die zurückgelegte Strecke und die Orientierung des Roboters zu berechnen.

Die Bewegung des Roboters ergibt sich aus den Encoderwerten durch die Formel:

Wobei

* r für den Radradius des Zumo-Roboters beträgt r = 0,019.
* 75,81 das Übersetzungsverhältnis des Getriebes.
* 12 die Impulse pro Umdrehung.
* CPR für die Counts per Revolution steht. Diese wird berechnet als:

### Bestimmung der Roboterbewegung

Die Vorwärtsbewegung des Roboters ergibt sich als Mittelwert der Radbewegungen:

Die Drehbewegung des Roboters wird aus der Differenz der Radbewegungen berechnet:

Hierbei bezeichnet „wheeltrack“ den Abstand zwischen den Rädern, welcher die Drehung des Roboters beeinflusst und beträgt, 0.09 m (s. Abbildung ‎9.1).

Die neue Position des Roboters wird auf Basis der Bewegung und der Rotation aktualisiert:

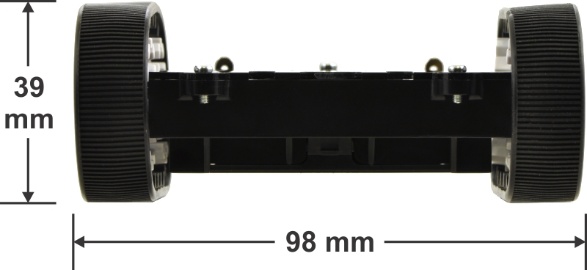


Abbildung .: Roboter Dimensionen

Diese berechneten Werte werden gespeichert und über das ROS 2-Topic „robot\_position“ veröffentlicht.

## Transformation von Koordinatenrahmen in ROS 2

In einem mobilen Robotersystem existieren zahlreiche dreidimensionale Koordinatenrahmen (Frames), die sich dynamisch über die Zeit hinweg verändern. Zu den wichtigsten gehören der Welt-Frame (map), der Basis-Frame, der Greifer-Frame und der Kopf-Frame. Die zentrale Herausforderung besteht darin, die Relationen zwischen diesen Koordinatenrahmen exakt zu bestimmen, um eine präzise Navigation und Steuerung des Roboters zu ermöglichen.

Hierfür wird das Konzept des TF2-Broadcastings genutzt, das in ROS 2 die Transformationen zwischen verschiedenen Frames verwaltet. Ohne Transformationen müssten diese Berechnungen ausschließlich mithilfe von Trigonometrie durchgeführt werden, was insbesondere in 3D sehr komplex wird. TF2 ist eine leistungsfähige Bibliothek, die eine gepufferte Baumstruktur bereitstellt, um Positions- und Orientierungsinformationen effizient zu speichern und über die Zeit zu aktualisieren. Diese Struktur erlaubt es, Punkte, Vektoren und andere Objekte in Echtzeit zwischen beliebigen Koordinatensystemen zu transformieren (s. Abbildung xxxx). Dies ist von zentraler Bedeutung für mobile Roboter, da ihre Position in der Umgebung kontinuierlich angepasst werden muss.

xxxxxxxxx

Bei einer komplexen Baumstruktur kann jeder ROS-2-Node die TF2-Bibliothek verwenden, um Transformationen von einem Frame zu einem anderen zu übertragen. Jeder Frame ist durch genau eine Transformation aus einem anderen Frame definiert, kann jedoch als Referenz für eine beliebige Anzahl weiterer Frames dienen (10).

## Implementierung des TF2-Nodes

Der tf2\_node übernimmt die Aufgabe, Transformationen zwischen den Frames zu veröffentlichen. In diesem Fall wird insbesondere die Transformation zwischen *map* und *base\_link* bereitgestellt, wodurch die Position und Orientierung des Roboters im globalen Weltkoordinatensystem erfasst werden. Eine gesonderte Kamera-Frame-Transformation wird nicht berücksichtigt, da die Kamera direkt auf dem Zumo-Roboter montiert ist und die Modellierung somit vereinfacht wird.

Der Node empfängt Positionsdaten über das ROS 2-Topic „robot\_position“, das als Pose-Nachricht bereitgestellt wird. Diese Nachricht enthält die kartesischen Positionswerte (x,y,z) wobei z gleich null ist, da es sich um eine 2D-Transformation handelt, sowie die Orientierungswerte (w,x,y,z) in Form einer Quaternion. Da der Roboter nur um die z-Achse rotiert, sind die Werte x und y gleich null, während z und w die Drehung beschreiben. Dabei entspricht z der Sinuskomponente und w der Kosinuskomponente der Rotation. Diese Transformation ermöglicht es, die Lage des Roboters relativ zur Umgebung konsistent zu bestimmen und innerhalb des ROS 2-Frameworks für Navigation und Steuerung zu nutzen

## Erstellung der Karte und Visualisierung

Eine zentrale Herausforderung in mobilen Robotersystemen ist die Erfassung und Speicherung des zurückgelegten Weges. Dies geschieht über eine 2D-Rasterkarte, die als OccupancyGrid implementiert wird. Hierbei wird die Umgebung in diskrete Zellen unterteilt, die jeweils unterschiedliche Belegungswerte aufweisen (s. Abbildung ‎9.2) (11):

* -1 = unbekannt
* 0 = freier Platz
* 1 = befahrener Pfad

Ein Bild, das Quadrat, Rechteck, Reihe, Text enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .: Occupancy Grid map

Die Karte 400 x 400 mit einer Auflösung von 0,01 (1 cm per Cell) wird kontinuierlich aktualisiert, indem die aktuelle Position des Roboters mit einer Markierung versehen wird. Auf diese Weise entsteht eine vollständige Darstellung der bereits befahrenen Bereiche durch Veröffentlichen von OccupancyGrid-Nachricht, sodass das befahrene Gebiet als Karte sichtbar gemacht wird. Die Speicherung und Visualisierung dieser Daten erfolgt in RViz2, einer ROS 2-Visualisierungssoftware.

## Visualisierung der Transformationen und Pfade in RViz2

Die in diesem Projekt ermittelten Pfaddaten werden mithilfe von RViz2 visualisiert. Dazu nutzt der implementierte visualize\_path()-Code die ROS 2 Marker-Nachrichten, um den vom Roboter zurückgelegten Pfad darzustellen.

Die Funktion erstellt einen Marker vom Typ LINE\_STRIP, der eine zusammenhängende Linie aus Punkten generiert, die den Bewegungsverlauf des Roboters repräsentieren. Alle Positionen, die im OccupancyGrid als Pfadpunkte (1) gespeichert wurden, werden in das Marker-Format umgewandelt und in Schwarz visualisiert (s. Abbildung ‎8.3).

Die Berechnung der Punktkoordinaten erfolgt durch die Umwandlung der Rasterpositionen aus self.map\_data in metrische Werte, wobei der Ursprung der Karte in die Mitte verschoben wird. Dies stellt sicher, dass die Darstellung in RViz2 mit der tatsächlichen Umgebung übereinstimmt.

Schließlich wird der Marker über self.marker\_pub.publish(marker) an ROS 2 gesendet, sodass er in RViz2 in Echtzeit angezeigt werden kann.

Ein Bild, das Screenshot, Reihe, Lenkdrachen enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung .:Pfadpunkte

## Validerung

# Zusammenfassung und Ausblicke

# Literaturverzeichnis

**Im aktuellen Dokument sind keine Quellen vorhanden.**

[1] [B. Vinoth Kumar](https://link.springer.com/book/10.1007/978-3-031-20541-5#author-1-0), [P. Sivakumar](https://link.springer.com/book/10.1007/978-3-031-20541-5#author-1-1), [B. Surendiran](https://link.springer.com/book/10.1007/978-3-031-20541-5#author-1-2), [Junhua Ding](https://link.springer.com/book/10.1007/978-3-031-20541-5#author-1-3) (2023) **Smart Computer Vision** [https://link.springer.com/book/10.1007/978-3-031-20541-5](https://link.springer.com/book/10.1007/978-3-031-20541-5%20)

[2] [Martin Löffler-Mang](https://link.springer.com/book/10.1007/978-3-8348-8308-7#author-0-0) (2012): **Optische Sensorik Lasertechnik, Experimente, Light Barriers** <https://link.springer.com/book/10.1007/978-3-8348-8308-7>

**[3] Patrick Kienle (2022) Hochauflösende optische Abstandsmessung mittels kompensierter Lasertriangulation** <https://mediatum.ub.tum.de/1624883>

**[4]Sony IMX 317** <https://www.sunnywale.com/uploadfile/2023/0605/IMX317CQC-C_TS_Full%20DS(0.1.0)_Awin.pdf>

**[5]** [Martin Löffler-Mang](https://link.springer.com/book/10.1007/978-3-8348-8308-7#author-0-0) **(2012): Optische Sensorik Lasertechnik, Experimente, Light Barriers** <https://link.springer.com/book/10.1007/978-3-8348-8308-7>

**[6] Stefan Roth und Achim Stahl (2019) Geometrische Optik** <https://link.springer.com/book/10.1007/978-3-662-59337-0>

**[7] Gary Bradski and Adrian Kaehler(2008) Learning OpenCV** <https://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf>

**[8] OpenCV (2024): Documentation,Image Filtering** <https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gac05a120c1ae92a6060dd0db190a61afa>

**[9]** **OpenCV (2025): Image Thresholding https://docs.opencv.org/4.x/d7/d4d/tutorial\_py\_thresholding.html**

[10] Satoshi Suzuki and others (1985) *Computer Vision, Graphics, and Image Processing* <https://web.archive.org/web/20231213161741/https://www.nevis.columbia.edu/~vgenty/public/suzuki_et_al.pdf>

[11] **OpenCV (2025): Moments** <https://docs.opencv.org/4.x/d8/d23/classcv_1_1Moments.html>

[12] **OpenCV (2025): KalmanFilter** https://docs.opencv.org/3.4/dd/d6a/classcv\_1\_1KalmanFilter.html

[13] **Bildquelle (2025) Turck.de** <https://www.turck.de/de/rfid-leitet-agv-in-der-federsystemproduktion-6870.php>

[14]Tesla (2024) **Tesla Vision Update: Replacing Ultrasonic Sensors with Tesla Vision** <https://www.tesla.com/support/transitioning-tesla-vision>

**[15] Felix Biermann (2024) Konzeption und Aufbau zur Laser-Entfernungsmessung per Triangulation**

**[16] Pololu (2025) Quadrature encoders** <https://www.pololu.com/docs/0J63/3.4>

# Anhang