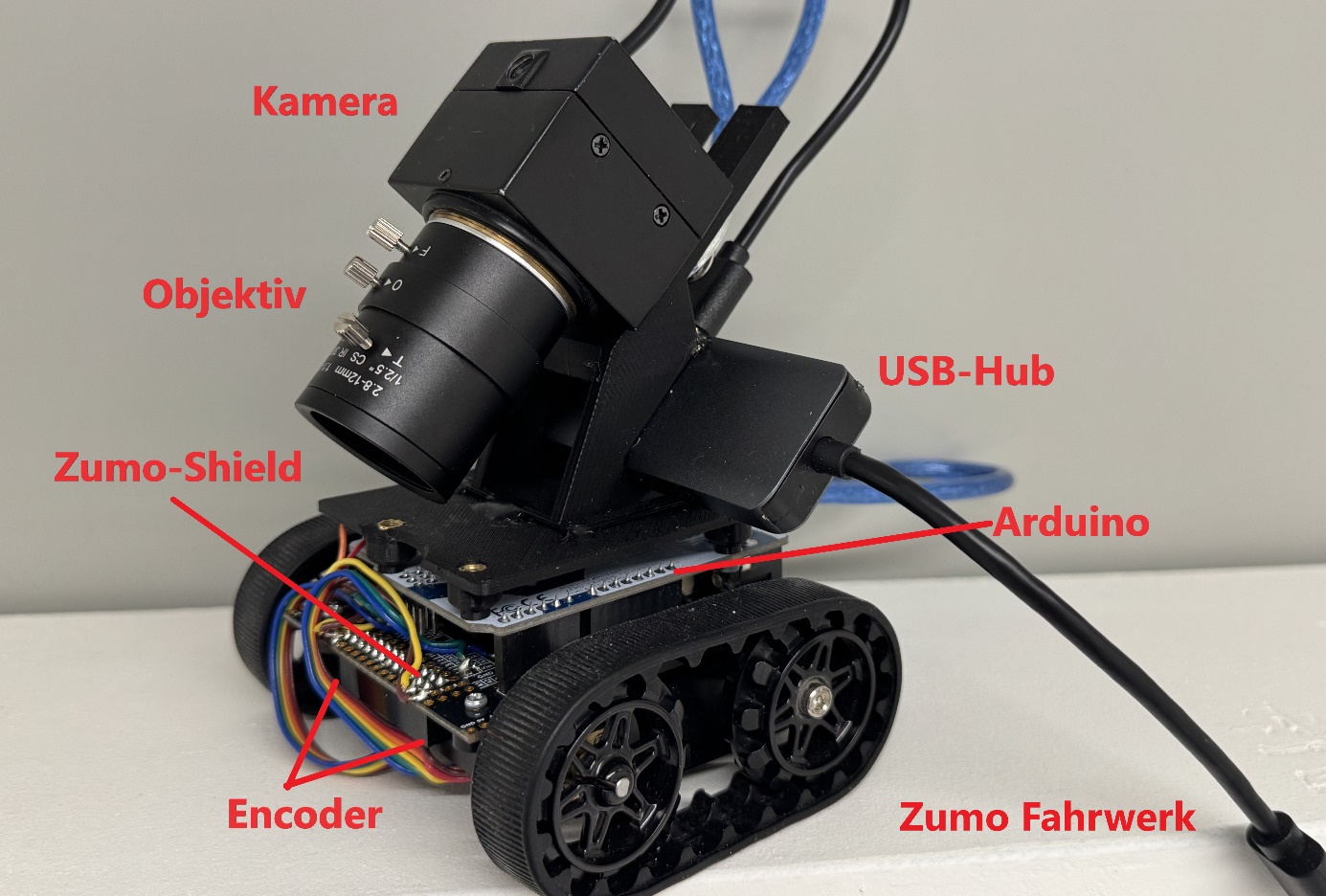
# 1 Hardware-Aufbau



*Abb.1 Aufbau des verwendeten Roboters [Eigenaufnahme]*

1.1 Zumo Fahrwerk & Shield  
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.

Zur Ansteuerung 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.

## 1.2 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.

## 1.3 Mikrocontroller

Zur Verarbeitung der Encodersignale 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 deutlich leistungsfähigeren 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.

## 1.4 Verwendete 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 2.1 und 2.2 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.

## 1.5 Steuerungs-PC

Da die Bildauswertung einer hochauflösenden Kamera sehr rechenintensiv ist, kann sie nicht auf dem Arduino des Roboters durchgeführt werden. Das Kamerabild, die Encoderwerte 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 das Robot Operating System (ROS 2) zum Einsatz.

# 2 Computer Vision

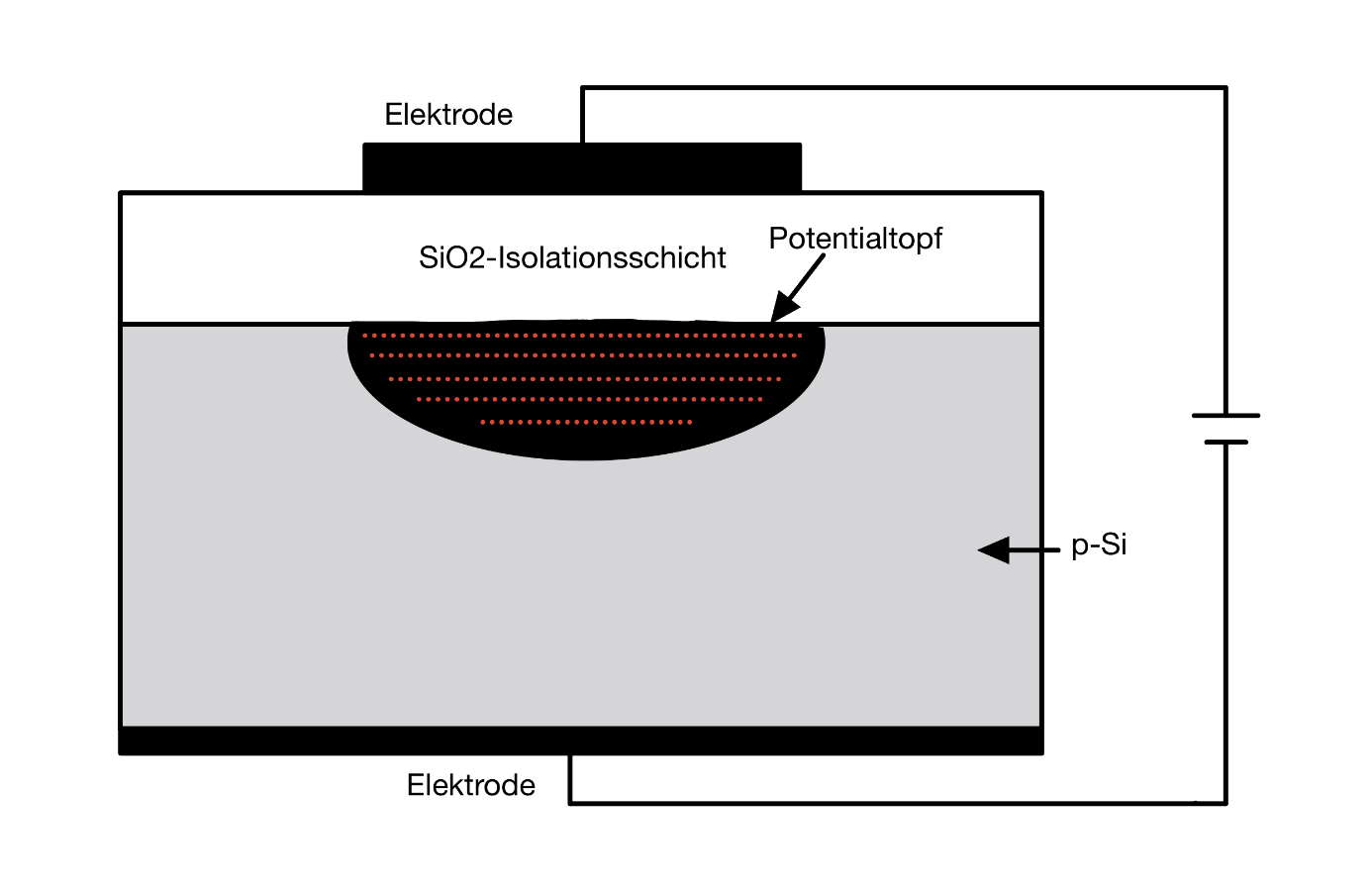
Der Bereich *Computer Vision* ist ein umfangreiches Gebiet, das darauf abzielt, nützliche Informationen aus einem Bild zu generieren. Die Hauptaufgaben der Computer Vision bzw. Bildverarbeitung liegen dabei in der Bild- bzw. Objektklassifikation, also dem Zuordnen eines Bildes oder von Objekten innerhalb 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:

## 2.1 Kamera

Bei diesem Projekt kommt ein sogenannter CMOS-Sensor zum Einsatz. Im Vergleich zu älteren CCD-Sensoren besitzt jedes 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].

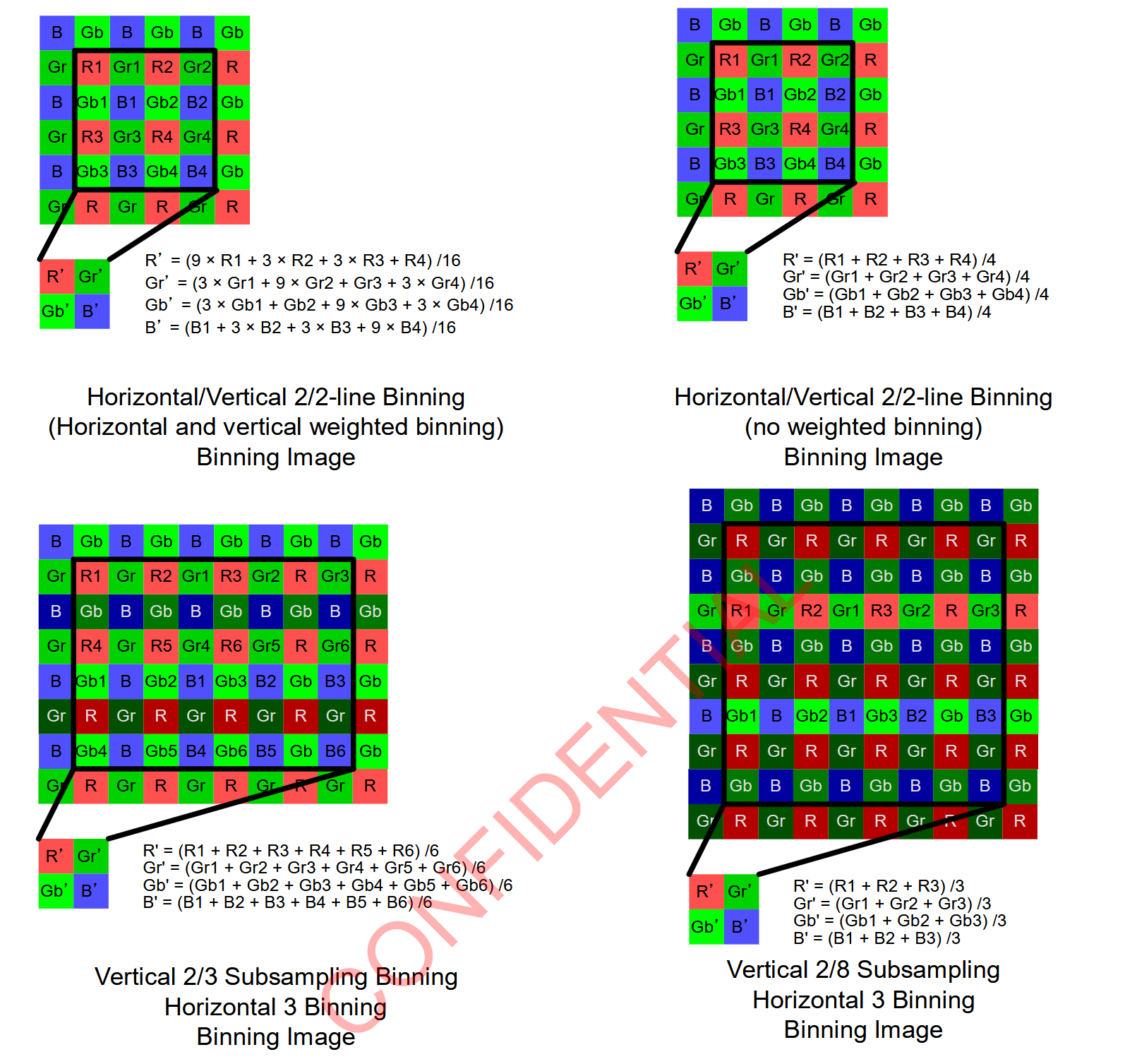


*Abb.2 Prinzipskizze einer CCD-Zelle bzw. eines Photokondensators mit einem unter der Isolationsschicht ausgebildetem Potentialtopf (vgl.[2]),[15]*

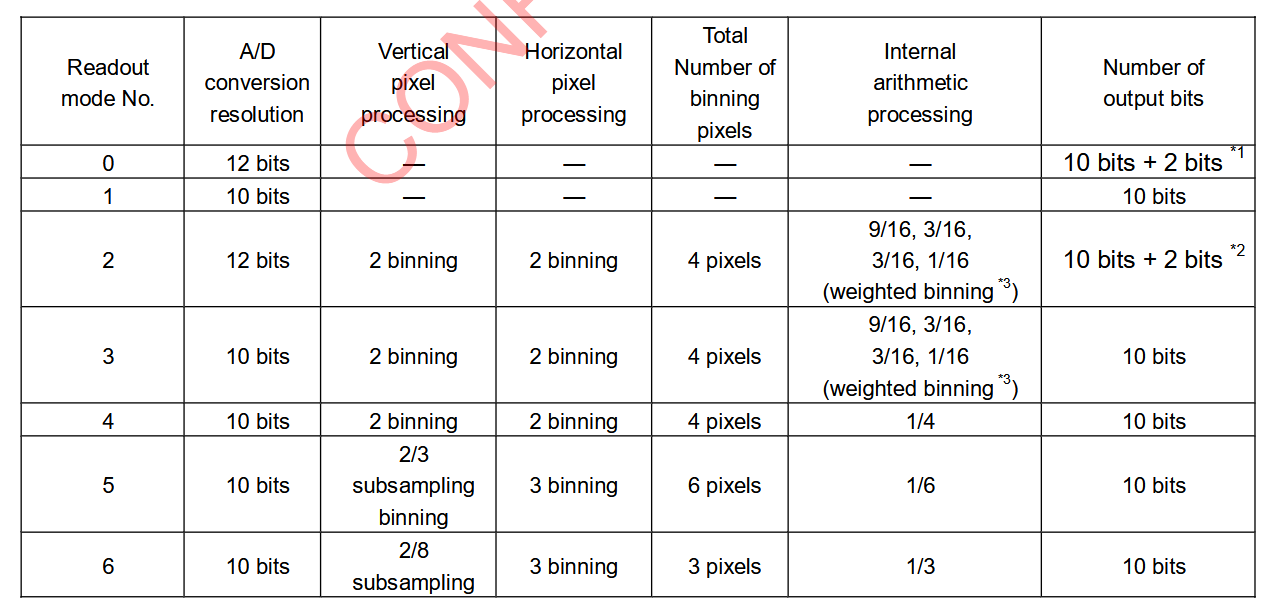
Bei dieser Art von Kondensator wird bei der Herstellung zwischen zwei Elektroden eine p-dotierte Halbleiterschicht ausgebildet, wobei eine der Elektroden durch eine Siliziumdioxid-Schicht vom p-Si isoliert ist. Trifft nun ein Photon auf die dotierte Siliziumschicht, entsteht durch den inneren Photoeffekt 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.



*Abb.4 Die verschiedenen Auswertemodi vgl.* [4]S.55



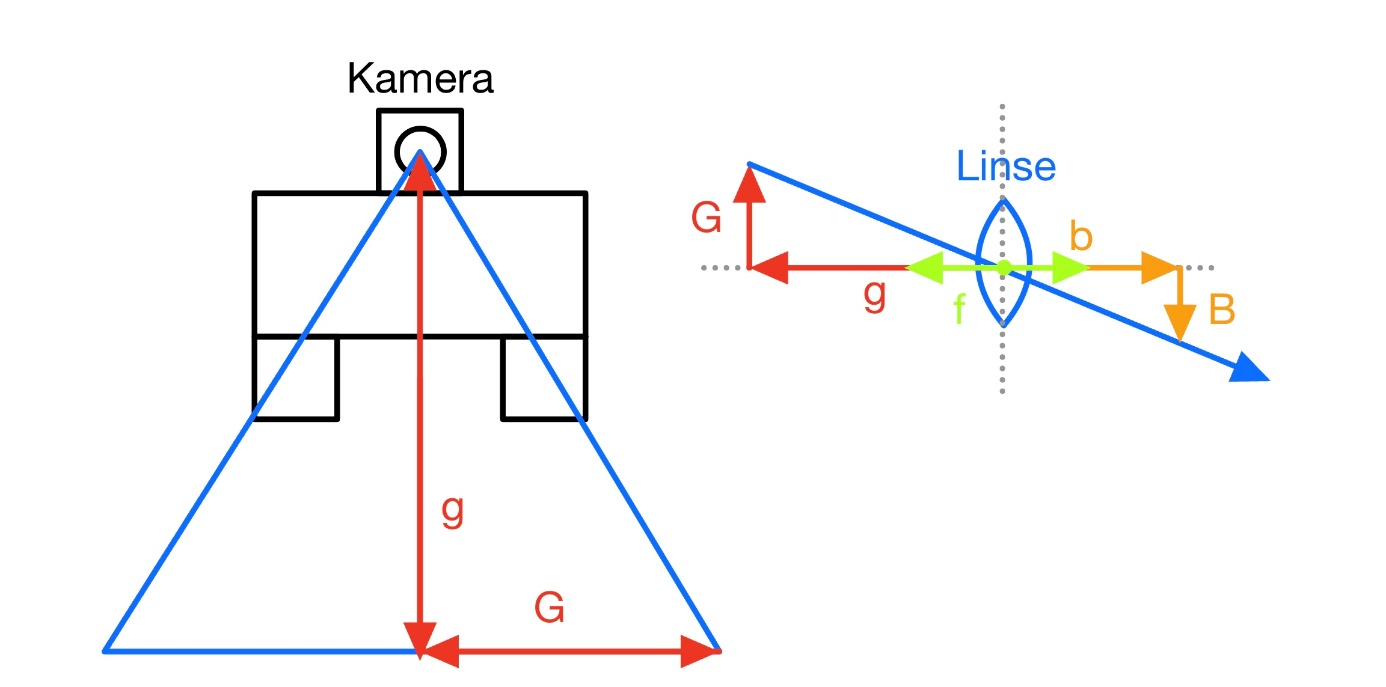
*Abb.5 Die verschiedenen Auswertemodi vgl.* [4]S.54

Durch zeitgleiches herabsetzten der Farbtiefe auf 10 Bit, können so signifikant geringere Datenraten erreicht werden. Der zu verwendende Modus kann später softwareseitig vorgegeben werden und muss dem 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.

## 2.2 Objektiv &Abbildungsfehler

Um ein scharfes Bild von eines Objekts 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 ein 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:



*Abb.6 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).

## 2.3 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).

### 2.3.1 OpenCV-Bildformat

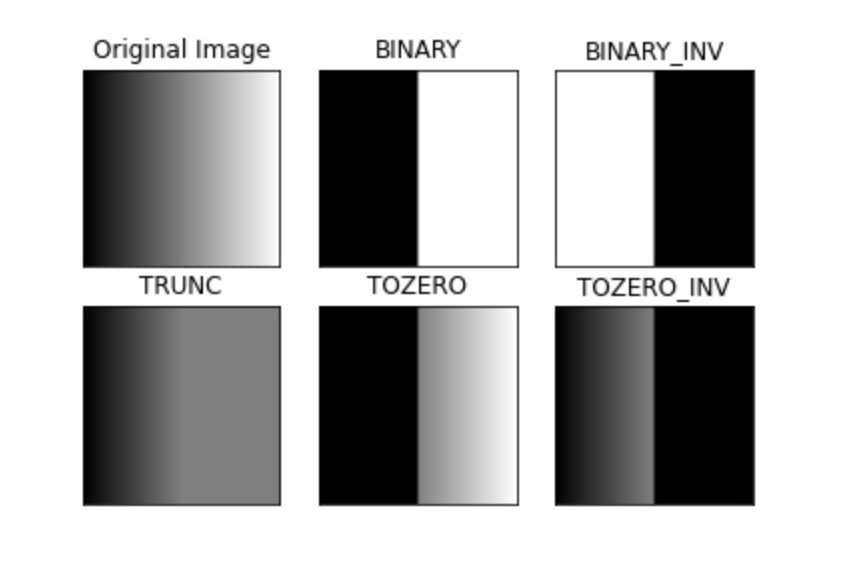
Bilder werden in OpenCV als Pixelmatrix bearbeitet ([7], S. 12). Farbige Bilder werden pro Pixel mit jeweils einem Helligkeitswert für die Farben Rot, Grün und Blau dargestellt. Die Auflösung dieses Farbwertes kann eingestellt werden, wodurch sich verschiedene sogenannte Farbräume ergeben. Bilder werden 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 ein Kamerastream, 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.

### 2.3.2 Vorbereitung und Filterung des Bildes

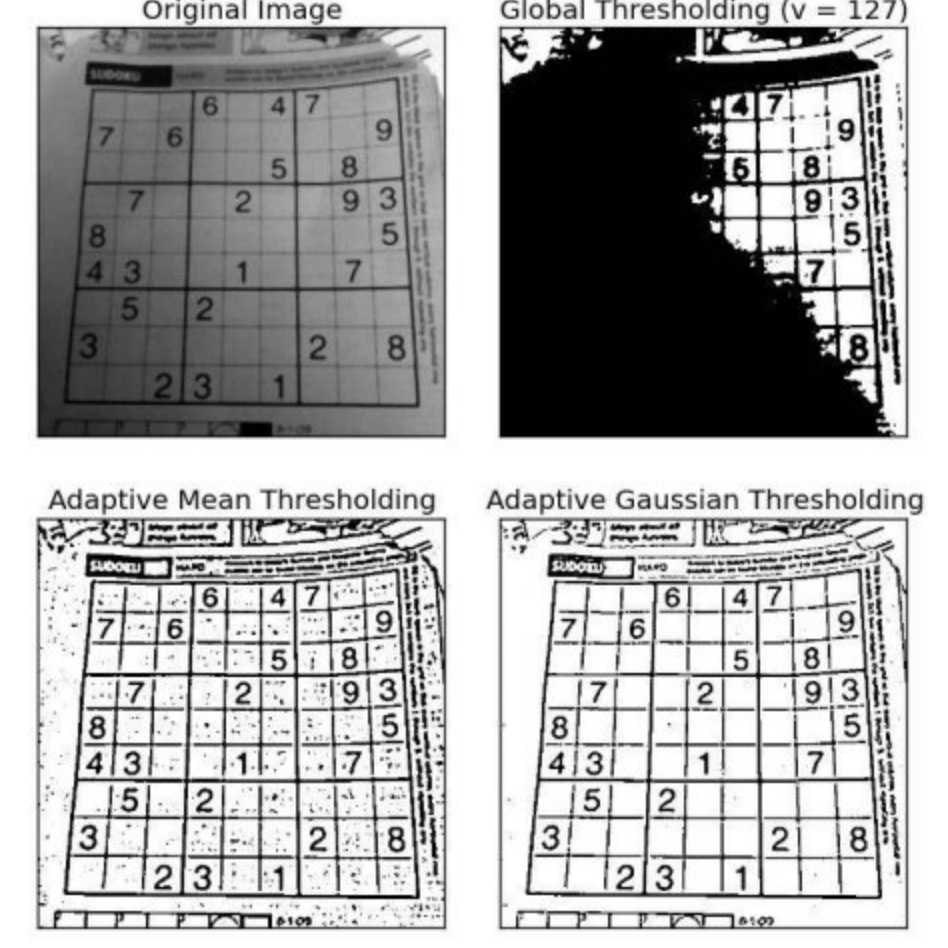
Bei vielen Anwendungsfällen in der Bildverarbeitung empfiehlt es sich, vor der weiteren Verarbeitung des Bildes ein sogenanntes Schwellwertverfahren (Threshold) durchzuführen. Dabei werden Pixel, die eine festgelegte Helligkeitsschwelle über- oder unterschreiten, entweder auf Null oder den Maximalwert 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.



*Abb.7 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.

 *Abb.8 Darstellung des Adapive Thresholding und des Gauss-Filters vgl.* [9]

### 2.3.3 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.

### 2.3.4 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 horizontale 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 vorausschauenderen Fahrweise des Fahrzeugs wider.

### 2.3.5 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.

### 2.3.6 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.

## 2.4 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.

## 2.5 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]

## 2.6 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.

*Abb.9 Industrieroboter mit visueller Linienverfolgung*[13]

[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**](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**](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**](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**](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**](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**](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**