HAW HAMBURG

MASTER THESIS
Shabir Rastagar

Deep Reinforcement Learning und Bildverarbeitung zur generalisierbaren Steuerung physischer Labyrinthe

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Shabir Rastagar

Deep Reinforcement Learning und Bildverarbeitung zur generalisierbaren Steuerung physischer Labyrinthe

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Automatisierung*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Marc Hensel
Zweitgutachter: Prof. Dr. Marco Grimm

Eingereicht am: 09.10.2025

Shabir Rastagar

Thema der Arbeit

Deep Reinforcement Learning und Bildverarbeitung zur generalisierbaren Steuerung physischer Labyrinthe

Stichworte

Deep Reinforcement Learning, Bildverarbeitung, Labyrinthsteuerung, Visuelle Navigation, Robotik, Echtzeitsteuerung

Kurzzusammenfassung

Diese Arbeit untersucht einen Reinforcement-Learning-Ansatz zur autonomen Steuerung eines physischen Kugellabyrinths. Ziel ist es, dass ein lernfähiger Agent eine Kugel über verschiedene Labyrinthmuster hinweg selbstständig vom Start- zum Zielpunkt navigiert. Aufbauend auf einem bestehenden Demonstrator wird ein kamerabasiertes Bildverarbeitungssystem genutzt, um den Systemzustand zu erfassen und als Eingabe für das lernbasierte Steuerungsverfahren bereitzustellen. Ein besonderer Fokus liegt auf dem Sim-to-Real-Transfer der trainierten Agenten.

Title of Thesis

Deep reinforcement learning and image processing for generalizable control of physical labyrinths

Keywords

Deep Reinforcement Learning, Image Processing, Maze Control, Visual Navigation, Robotics, Real-Time Control

Abstract

This thesis investigates a reinforcement learning approach for the autonomous control of a physical marble maze. The goal is for a learning agent to independently navigate a ball from the start to the goal across various maze patterns. Building on an existing demonstrator, a camera-based image processing system is used to capture the system state and provide it as input to the learning-based control method. A particular focus lies on the sim-to-real transfer of the trained agents.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

Tabellenverzeichnis **x**

Abkürzungsverzeichnis xii

1 Einleitung **1**

1.1 Motivation 1

1.2 Umfeld 2

1.3 Zielsetzung 2

2 Theoretische Grundlagen **3**

2.1 Kugellabyrinth 3

2.2 Kameraverzerrungskompensation 4

2.3 Bildverarbeitung 6

2.3.1 Bildvorverarbeitung 7

2.3.2 Bildsegmentierung 9

2.3.3 Merkmalsextraktion 13

2.3.4 Klassifizierung 15

2.3.5 HSV-Farbraum 17

2.4 Maschinelles Lernen 18

2.5 Reinforcement Learning 19

3 Stand der Technik **22**

3.1 Bisheriger Systemstand 22

3.1.1 Eingesetzte Hardware 23

3.1.2 Verwendete Algorithmen und Verfahren 24

3.2 Vergleich bestehender Ansätze 27

3.2.1 Deutsches Forschungszentrum für künstliche Intelligenz 27

3.2.2 Eidgenössisch Technische Hochschule Zürich 28

3.2.3 Turtlebot Navigation . . . 29
3.3 Zusammenfassung und Abgrenzung . . . 30

4 Anforderungsanalyse 32
4.1 Systemumgebung . . . 32
4.2 Stakeholder . . . 33
4.2.1 Auftraggeber . . . 33
4.2.2 Entwickler . . . 34
4.2.3 Nachfolgende Entwickler . . . 35
4.2.4 Anwender . . . 35
4.3 Physischer Demonstrator . . . 36
4.3.1 Anwendungsfälle . . . 36
4.3.2 Anforderungserhebung . . . 37

5 Konzept 41
5.1 Systemarchitektur . . . 41
5.2 Hardware . . . 43
5.2.1 Sensorische Erfassung des Labyrinthzustands . . . 43
5.2.2 Steuereinheit . . . 46
5.3 Software . . . 47
5.3.1 Entwicklungsumgebung und Programmiersprache . . . 47
5.3.2 Bildverarbeitungsstrategie . . . 48
5.3.3 Agentensystem . . . 50

6 Entwicklung 52
6.1 Kamerawahl . . . 52
6.1.1 Aufbau und Montage der Kamerahardware . . . 53
6.1.2 Softwareanbindung und Kamerasteuerung in Python . . . 55
6.1.3 Kamerakalibrierung . . . 56
6.2 Relevante Bildinformationen . . . 60
6.3 Kugelerkennung . . . 61
6.3.1 Methodenanalyse . . . 61
6.3.2 Ablauf . . . 63
6.3.3 Implementierung . . . 65
6.4 Falllöchererkennung . . . 66
6.4.1 Methodenanalyse . . . 66
6.4.2 Ablauf . . . 67

6.4.3 Implementierung 69
6.5 Spielfelderkennung 70
6.5.1 Methodenanalyse 71
6.5.2 Ablauf 71
6.5.3 Implementierung 73
6.6 Wände 73
6.6.1 Methodenauswahl 74
6.6.2 Ablauf 74
6.6.3 Implementierung 75
6.7 Ziel 76
6.7.1 Methodenanalyse 77
6.7.2 Ablauf 77
6.7.3 Implementierung 78
6.8 Programmstruktur und Systemarchitektur 80

7 Evaluierung **83**
7.1 Spielfelderkennung 83
7.1.1 Bewertungsmethodik 84
7.1.2 Testergebnisse 84
7.1.3 Einordnung der Ergebnisse 86
7.2 Kugelerkennung 87
7.2.1 Bewertungsmethodik 87
7.2.2 Testergebnisse 88
7.2.3 Einordnung der Ergebnisse 89
7.3 Locherkennung 90
7.3.1 Bewertungsmethodik 91
7.3.2 Testergebnisse 91
7.3.3 Einordnung der Ergebnisse 93
7.4 Zielerkennung 93
7.4.1 Bewertungsmethodik 94
7.4.2 Testergebnisse 94
7.4.3 Einordnung der Ergebnisse 95
7.5 Wanderkennung 95
7.5.1 Bewertungsmethodik 95
7.5.2 Testergebnisse 96
7.5.3 Einordnung der Ergebnisse 98

7.6 Anforderungsprüfung 99
7.6.1 Funktionale Anforderungen 99
7.6.2 Nicht-funktionale Anforderungen 100

8 Fazit und Ausblick **104**

9 Danksagung **109**

Literaturverzeichnis **110**

Selbstständigkeitserklärung **113**

Abbildungsverzeichnis

2.1 Die drei mitgelieferten Labyrinthmuster: (a) einfach, (b) mittel, (c) schwierig 5
2.2 Beispiel für Konvexe Hülle und Bounding Box [6] 14
2.3 HSV-Kegel zur Veranschaulichung der Farbparameter Hue, Saturation und Value 18
2.4 Grundstruktur eines Reinforcement-Learning-Systems 20

3.1 Labyrinthmuster HOLES_2 [15] 24
3.2 Kameragestell mit befestigter Webcam zur Top-Down-Erfassung [15] . . . 25
3.3 Virtuelle Umgebung mit Labyrinthmuster *HOLE_8* [15] 26
3.4 Kamerabild des zugeschnittenen Spielfelds mit blauen Referenzpunkten [3] 29

4.1 Systemumgebung mit Interaktion zwischen Agent, physischem System und virtueller Umgebung 33
4.2 Use-Case-Diagramm des physischen Demonstrators 37

5.1 Komponentendiagramm 42

6.1 Verwendete Hardware: (a) Kamera, (b) Objektiv 54
6.2 Physischer Aufbau der Kameraintegration oberhalb des Kugellabyrinths . 55
6.3 UML-Diagramm der Kamerasteuerungs-Architektur [10] 56
6.4 Aktivitätsdiagramm für CameraCalib 59
6.5 Vergleich der Kameraaufnahmen: (a) Unkorrigiertes Bild, (b) kalibriertes Bild 60
6.6 Aktivitätsdiagramm der Kugelerkennung 64
6.7 Aktivitätsdiagramm der Falllöchererkennung 69
6.8 Schrittweise Verarbeitung der Locherkennung: (a) Binarisierung, (b) Segmentierung der Lochkonturen, (c) Extraktion und Klassifizierung der Merkmale 70
6.9 Aktivitätsdiagramm der Spielfelderkennung 72
6.10 Aktivitätsdiagramm der Wanderkennung 75

6.11 Aktivitätsdiagramm der Zielerkennung 79
6.12 Beispielbild des Asterisk-Ziels, das als Template verwendet wird. 80
6.13 Klassendiagramm der Bildverarbeitungsarchitektur 81

7.1 Beispielhafte Spielfelderkennung unter künstlicher Raumbeleuchtung . . . 86
7.2 Exemplarischer Frame aus der Evaluierung der Kugelerkennung 90
7.3 Ergebnisse der Locherkennung für unterschiedliche Labyrinthmuster. . . . 92
7.4 Exemplarische Zielerkennung im Labyrinthmuster `HOLES_2` 94
7.5 Wanderkennung im Labyrinthmuster HOLES_2 97

8.1 Aktuelles Szenario 106
8.2 Optimiertes Szenario 107

Tabellenverzeichnis

3.1 Vergleich relevanter methodischer Aspekte 30

5.1 Vergleich der RGB-Kamera und Tiefenkamera anhand relevanter Bewertungskriterien 45
5.2 Vergleich der klassischen Bildverarbeitung und Deep Learning hinsichtlich relevanter Kriterien 49

6.1 Überblick relevanter Bildinformationen zur Zustandserfassung 61
6.2 Übersicht der implementierten Detektionsmodule 82

7.1 Erkennungsleistung der Spielfelderkennung bei jeweils 10 Testdurchläufen pro Lichtbedingung 85
7.2 Ergebnisse der Locherkennung bei verschiedenen Labyrinthmustern unter Raumbeleuchtung 91
7.3 Erkennungsleistung der Zielerkennung bei jeweils 10 Testdurchläufen pro Labyrinthmuster 94
7.4 Evaluierung der Wändeerkennung 96

Abkürzungsverzeichnis

AI	Artificial Intelligence
API	Application Programming Interface
ArUco	Augmented Reality University of Cordoba
BGR	Blue Green Red
ChArUco	Chessboard ArUco
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
DFKI	Deutsches Forschungszentrum für künstliche Intelligenz
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ETH	Eidgenössisch Technische Hochschule
FA	Funktionale Anforderung
FN	False Negative
FP	False Positive
GPIO	General Purpose Input/Output
HAW	Hochschule für Angewandte Wissenschaften
HSV	Hue, Saturation, Value
IA	Image Acquisition
IoU	Intersection over Union
IP	Image Processing
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
ML	Maschinelles Lernen
NFA	Nicht-funktionale Anforderung
ODE	Open Dynamics Engine
OpenCV	Open Source Computer Vision Library
RGB	Rot, Grün, Blau
RL	Reinforcement Learning
SDK	Software Development Kit
TN	True Negative
TP	True Positive
UML	Unified Modeling Language

1 Einleitung

Die Steuerung von Objekten in komplexen Umgebungen gehört zu den größten Herausforderungen in der Robotik und künstlichen Intelligenz. Herkömmliche, regelbasierte Ansätze stoßen schnell an ihre Grenzen, da sie nicht flexibel auf unvorhergesehene Situationen reagieren können. Reinforcement Learning (RL) bietet hier einen alternativen Ansatz. Anstatt festen Regeln zu folgen, lernt ein Agent durch Belohnung und Bestrafung, optimale Strategien selbst zu entwickeln. Diese Methode hat in den letzten Jahren erhebliche Fortschritte erzielt und wird in Bereichen wie Robotik, autonomem Fahren und komplexen Strategiespielen erfolgreich eingesetzt [14, 2, 15].

1.1 Motivation

Reinforcement Learning hat in virtuellen Umgebungen, etwa in Strategiespielen und Simulationen, beeindruckende Erfolge erzielt. Während sich Reinforcement-Learning-Ansätze bislang vor allem in simulierten Szenarien bewährt haben, konzentriert sich diese Arbeit auf die bildbasierte Wahrnehmung des Systems als Grundlage für zukünftige Lern- und Regelungsverfahren in einem realen physischen System. Als Testumgebung dient das Kugellabyrinth, ein Geschicklichkeitsspiel, das präzise Bewegungssteuerung und kontinuierliche Zustandsrückmeldung erfordert. Ziel ist es, visuelle Informationen über den aktuellen Systemzustand und relevante Merkmale des Spielfelds zu erfassen und daraus eine Grundlage für die spätere Regelung des Systems zu schaffen. Durch den Vergleich zwischen in der Simulation erfassten Zuständen und den aus realen Bilddaten gewonnenen Zuständen lassen sich sowohl die Grenzen als auch die Potenziale bildbasierter Ansätze aufzeigen. Die Ergebnisse liefern nicht nur einen Beitrag zur Übertragbarkeit von Reinforcement Learning auf physische Systeme, sondern zeigen auch Potenziale für Anwendungen in Robotik und Automatisierung auf.

1.2 Umfeld

Die vorliegende Arbeit ist eingebettet in ein fortlaufendes Forschungsprojekt an der HAW Hamburg unter der Betreuung von Prof. Marc Hensel. In den vorangegangenen Projektphasen wurden die Hardwareplattform entwickelt und erste Steuerungsansätze in einer virtuellen Umgebung erprobt. Darauf aufbauend liegt der Fokus dieser Arbeit auf der Implementierung einer bildverarbeitungsbasierten Wahrnehmung des Systems, die als Grundlage für zukünftige Steuerungs- und Lernverfahren auf der realen Hardware dient.

1.3 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung eines bildverarbeitungsbasierten Systems zur Erfassung und Auswertung des Systemzustands eines mechanischen Kugellabyrinths. Dabei soll ein funktionaler Demonstrator realisiert werden, der die technische Umsetzbarkeit und Zuverlässigkeit des Konzepts unter realitätsnahen Bedingungen belegt.

Im Fokus stehen die Konzeption, Implementierung und Evaluation effizienter Verfahren zur visuellen Erkennung relevanter Merkmale des Spielfelds und zur robusten Bestimmung des aktuellen Systemzustands.

Die Arbeit verfolgt zwei zentrale Zielsetzungen. Zum einen die Entwicklung und Validierung robuster Bildverarbeitungsmethoden für eine präzise und stabile Zustandserfassung, zum anderen die Integration dieser Methoden in die bestehende Hardwareplattform, um eine konsistente und reproduzierbare Datengrundlage für zukünftige Regelungs- und Lernverfahren zu schaffen.

Langfristig soll die Arbeit einen Beitrag zur Vorbereitung einer lernbasierten Steuerung leisten, indem sie die notwendigen bildbasierten Grundlagen für den Einsatz von Reinforcement Learning in realweltlichen Steuerungsszenarien schafft. Der entwickelte Demonstrator dient somit als Plattform für zukünftige Forschung und experimentelle Weiterentwicklungen im Bereich der automatisierten Steuerung und Robotik.

2 Theoretische Grundlagen

Das Verständnis der in dieser Arbeit entwickelten Methoden und eingesetzten Technologien setzt grundlegendes Wissen in mehreren Bereichen voraus. In diesem Kapitel werden daher zentrale theoretische Konzepte und Grundlagen vorgestellt, die für das Verständnis der nachfolgenden Abschnitte erforderlich sind.

Der Fokus liegt dabei ausschließlich auf Themen, die unmittelbar für die Umsetzung und Analyse der in dieser Arbeit behandelten Fragestellung relevant sind. Es wird auf ausgewähltes Fachwissen zurückgegriffen, das im Kontext dieser Arbeit notwendig ist, jedoch in seiner Tiefe auf das Wesentliche reduziert bleibt.

Behandelt werden sowohl hardwarebezogene Aspekte wie der physikalische Aufbau des Kugellabyrinths und der Kameratechnik, als auch softwareseitige Verfahren, im Bereich der Bildverarbeitung und des Reinforcement Learnings.

2.1 Kugellabyrinth

Das Kugellabyrinth ist ein für Kinder konzipiertes Geschicklichkeitsspiel, das darauf ausgelegt ist, die Feinmotorik, Konzentrationsfähigkeit sowie die Hand-Augen-Koordination zu fördern [1]. Besonders bekannt wurde das Spiel im Jahr 1964 durch die Markteinführung des BRIO-Labyrinths, das bis heute als Referenzmodell für diese Art von Spiel gilt[2].

Ziel des Spiels ist es, eine Stahlkugel entlang eines Pfads von einem definierten Startpunkt zu einem Zielpunkt zu balancieren. Dieser Pfad verläuft innerhalb eines Labyrinthmusters, das durch Wände und Hindernisse begrenzt ist. Die Wände sind als leichte Erhebungen im Spielfeld realisiert und bilden die Struktur des Labyrinths. Eine Besonderheit dieser Labyrinthform besteht darin, dass zusätzlich zu den Wänden mehrere Löcher auf

[1] https://www.thalia.de/shop/home/artikeldetails/A1008268054
[2] https://www.cyberrunner.ai/history/

der Spielfläche verteilt sind. Diese Löcher sind groß genug, dass die Kugel darin versinken kann, was einen sofortigen Neustart erzwingt und den Schwierigkeitsgrad der Aufgabe erheblich erhöht.

Zur Orientierung ist in jedes Muster eine schwarze Führungsline integriert, die den optimalen Weg vom Start- zum Zielpunkt visualisiert. Diese Linie schlängelt sich an Wänden und Löchern vorbei und dient als grobe Richtlinie zur Navigation.

Die Bewegung der Kugel erfolgt ausschließlich durch Schwerkrafteinwirkung infolge von Neigung. Hierzu ist das Labyrinthmuster in ein Gehäuse eingesetzt, das über zwei Drehregler entlang orthogonaler Achsen geneigt werden kann. Durch das manuelle Drehen dieser Räder entsteht eine kontrollierte Neigung der Spielfläche, welche die Kugel in Bewegung versetzt.

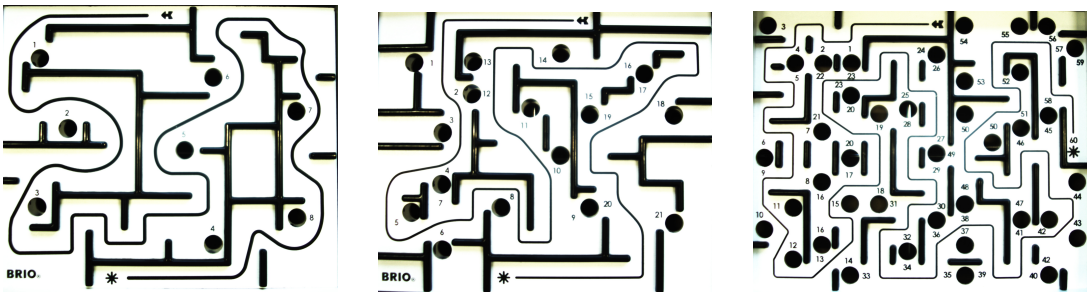
In dieser Arbeit wird das BRIO 34020 Labyrinth[3] verwendet. Das Spiel besteht hauptsächlich aus Holz, besitzt ein rotes Gehäuse und hat eine Gesamtgröße von 33,0 cm × 30,5 cm × 9,4 cm (L×B×H, eigene Messung). Die schwarzen Neigungsräder ermöglichen eine maximale Auslenkung der Spielfläche von etwa ±3,17° in horizontaler Richtung und ±2,43° in vertikaler Richtung [15].

Dem Spiel liegen drei austauschbare Labyrinthmuster mit ansteigendem Schwierigkeitsgrad bei, die in Abb. 2.1 dargestellt sind. Diese tragen die Bezeichnungen `HOLES_8`, `HOLES_21` und `HOLES_52` (von Vorgänger Arbeit übernommen). Alle drei Platten haben identische Abmessungen von 27,3 cm × 22,8 cm × 0,9 cm. Die Labyrinthe bestehen jeweils aus einer weißen Grundplatte mit schwarzen Wänden, einer schwarzen Führungsline sowie einer nummerierten Markierung aller Löcher.

2.2 Kameraverzerrungskompensation

Technische Kameras bilden reale Szenen nicht perfekt ab. Optische Bauelemente wie Linsen verursachen systematische Abweichungen von idealen Projektionsmodellen. Zur mathematischen Beschreibung der Bildentstehung dient häufig das Pinhole-Kameramodell, das die Abbildung eines räumlichen Punktes $P = [X, Y, Z]^\top$ auf ein Bildpunktpaar $p = [u, v]^\top$ idealisiert [1].

[3] https://www.brio.de/de-DE/produkte/spiele/labyrinth-mit-uebungsplatten-rot-63402000



(a) HOLES_8 (einfach) (b) HOLES_21 (mittel) (c) HOLES_52 (schwierig)

Abbildung 2.1: Die drei mitgelieferten Labyrinthmuster: (a) einfach, (b) mittel, (c) schwierig

Zentraler Bestandteil dieses Modells ist die intrinsische Kameramatrix $\mathbf{K}$, welche die inneren Parameter der Kamera zusammenfasst [23]:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Dabei stehen f_x und f_y für die Brennweite in Pixeln in horizontaler bzw. vertikaler Richtung, berechnet über f/d_x und f/d_y, während c_x und c_y die Position des Hauptpunkts angeben, also den Schnittpunkt der optischen Achse mit der Bildebene.

Die Projektion erfolgt unter Berücksichtigung eines Tiefenfaktors s nach folgender Beziehung [5]:

$$s \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.2}$$

Tatsächliche Objektive verursachen jedoch Bildverzerrungen, die durch das Pinhole-Modell nicht erfasst werden. Um diese korrigieren zu können, werden die Pixelkoordinaten zunächst normalisiert:

$$\begin{aligned} x &= \frac{u - c_x}{f_x}, \\ y &= \frac{v - c_y}{f_y} \end{aligned} \tag{2.3}$$

Die erste Verzerrungsart ist radialer Natur und entsteht durch die sphärische Form der Linsen. Sie bewirkt, dass Bildpunkte mit zunehmendem Abstand vom Zentrum stärker verzerrt werden. Diese Verzerrung wird durch die Terme zweiter, vierter und sechster Ordnung in den Koeffizienten k_1, k_2, k_3 modelliert:

$$\begin{bmatrix} x_{\text{korr}} \\ y_{\text{korr}} \end{bmatrix} = \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.4}$$

mit $r^2 = x^2 + y^2$ als quadrierter Abstand zum Hauptpunkt.

Zusätzlich tritt häufig eine tangentiale Verzerrung auf, die auf kleine Verkippungen oder Dezentrierungen innerhalb des Linsensystems zurückzuführen ist. Diese wird durch zwei weitere Parameter p_1 und p_2 beschrieben:

$$\begin{aligned} x_{\text{korr}} &= x + 2p_1xy + p_2(r^2 + 2x^2) \\ y_{\text{korr}} &= y + p_1(r^2 + 2y^2) + 2p_2xy \end{aligned} \tag{2.5}$$

Das vollständige Modell der optischen Verzerrung wird durch den Parametervektor

$$D = [k_1, k_2, p_1, p_2, k_3]$$

beschrieben. Durch Anwendung einer geeigneten inversen Transformation kann auf Basis dieser Parameter eine entzerrte und geometrisch korrekte Bilddarstellung berechnet werden.

2.3 Bildverarbeitung

Im Kontext dieser Arbeit bezieht sich der Begriff klassische Bildverarbeitung auf eine sequenzielle Folge von Verarbeitungsschritten, die von der Aufnahme eines Bildes bis

hin zur Interpretation bzw. Nutzung der extrahierten Informationen reicht. Die typische Bildverarbeitungskette lässt sich in fünf aufeinanderfolgende Hauptphasen gliedern [6]:

1. **Bildaufnahme:** Die Bildverarbeitungskette beginnt mit der Bildaufnahme, also der Erfassung der Bilddaten durch das Kamerasystem.
2. **Bildvorverarbeitung:** In diesem Schritt werden Rohdaten (z. B. RGB-Bilder) optimiert, um eine bessere Weiterverarbeitung zu ermöglichen.
3. **Bildsegmentierung:** Ziel ist die Trennung relevanter Bildbereiche zum Rest des Bildes.
4. **Merkmalsextraktion:** In diesem Schritt werden gezielt Informationen aus den segmentierten Regionen extrahiert.
5. **Klassifizierung:** Abschließend erfolgt die Interpretation der extrahierten Merkmale.

Die eigentliche Bildaufnahme bildet den vorbereitenden Schritt der Bildverarbeitungskette. Dabei ist durch eine abgestimmte Kombination aus Beleuchtung, Objektiv und Kamerasystem sicherzustellen, dass die relevanten Bildinhalte möglichst klar und kontrastreich erfasst werden.

2.3.1 Bildvorverarbeitung

Ziel der Bildvorverarbeitung ist die gezielte Verbesserung der Bildqualität, um in späteren Schritten eine zuverlässige Trennung relevanter Objekte vom Hintergrund zu ermöglichen [21].

Grundsätzlich lassen sich zwei zentrale Klassen von Operationen unterscheiden: Punktoperationen und lokale Operationen [6].

Punktoperationen Punktoperationen verändern den Wert eines Pixels ausschließlich basierend auf dessen ursprünglichem Wert. Die Nachbarpixel bleiben dabei unberücksichtigt. Man unterscheidet zwischen homogenen und inhomogenen Punktoperationen [19]:

- **Homogene Punktoperation:** Die Transformation ist unabhängig von der Pixelposition (x, y).
- **Inhomogene Punktoperation:** Die Transformation hängt explizit von der Pixelposition (x, y) ab.

Ein typischer Anwendungsfall ist die Kontrastverbesserung durch Histogrammanalyse. Das Histogramm eines Bildes beschreibt die Häufigkeit des Auftretens bestimmter Grauwerte und wird formal definiert als:

$$H(z) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (g(x, y) == z) \tag{2.6}$$

Hierbei ist $g(x, y)$ der Grauwert des Pixels an Position (x, y), z ein möglicher Grauwert, M und N sind Bildhöhe und -breite.

Zur Kontrastanpassung wird unter anderem die Gammakorrektur (Inhomogene Punktoperation) eingesetzt. Diese basiert auf einer nichtlinearen Transformation der Grauwerte mittels einer Potenzfunktion:

$$g' = g^{\gamma} \quad \text{mit } g \in [0, 1] \tag{2.7}$$

Der Exponent γ (Gammawert) bestimmt dabei die Art der Kontrastanpassung: Werte $\gamma < 1$ hellen das Bild auf, während $\gamma > 1$ zu einer Abdunkelung führen.

Lokale Operationen Im Gegensatz zu Punktoperationen beziehen lokale Operationen auch benachbarte Pixel in die Berechnung des neuen Pixelwertes ein. Eine zentrale Methode stellt dabei die Faltung mit einem Faltungskern dar [6]:

$$g'(x, y) = \frac{1}{S} \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} g\left(x + \frac{m-1}{2} - u, y + \frac{m-1}{2} - v\right) \cdot h(u, v) \tag{2.8}$$

Hierbei ist $g(x, y)$ das Eingangsbild, $g'(x, y)$ das gefilterte Bild, $h(u, v)$ der Filterkernel der Größe $m \times m$, und S ein Normierungsfaktor (in der Regel die Summe der Koeffizienten von h).

Je nach Gestaltung des Filterkerns unterscheidet man zwischen Tiefpass- und Hochpassfiltern:

- **Tiefpassfilter (Glättungsfilter):** Dienen zur Rauschunterdrückung und Glättung von Bildinhalten. Ein typisches Beispiel ist der Gauß-Filter, dessen Kernel einer zweidimensionalen Gaußverteilung folgt:

$$h(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.9}$$

- **Hochpassfilter (Kantenfilter):** Betonen Helligkeitsänderungen und somit Kanten im Bild. Ein weit verbreiteter Ansatz ist der Canny-Filter, der in mehreren Schritten arbeitet:

  1. Rauschunterdrückung mit Gauß-Filter
  2. Berechnung des Gradienten (Sobel-Operator)
  3. Non-Maxima-Suppression
  4. Doppelschwellenwertverfahren zur Kantenverfolgung

  Die Canny-Kantenerkennung basiert auf dem Gradientenbetrag G und dem Gradientenwinkel θ:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{2.10}$$

Durch den gezielten Einsatz von Bildvorverarbeitungsschritten lässt sich die Qualität der Segmentierung erheblich verbessern, insbesondere in Echtzeitanwendungen oder bei schwierigen Lichtverhältnissen.

2.3.2 Bildsegmentierung

Ziel der Bildsegmentierung ist es, ein gegebenes Bild in inhaltlich zusammenhängende Bereiche zu unterteilen. Insbesondere in die für die jeweilige Anwendung relevanten Objekte und den nicht relevanten Hintergrund.

Im engeren Sinne bezeichnet Segmentierung die vollständige Trennung dieser Regionen, sodass jedes Pixel des Bildes eindeutig einem Segment zugeordnet ist.

Im folgenden werden verschiedene Verfahren der Bildsegmentierung behandelt. Dazu zählen die Schwellwertsegmentierung, ein modellbasiertes Verfahren sowie morphologische Operationen, die zur gezielten Nachbearbeitung segmentierter Bildbereiche eingesetzt werden [6].

Schwellwertsegmentierung

Ein einfaches und weit verbreitetes Verfahren zur Bildsegmentierung ist die Schwellwertsegmentierung. Dabei wird jedes Pixel eines Bildes basierend auf seinem Grauwert einer von zwei Klassen zugeordnet. In der Regel Objekt oder Hintergrund. Voraussetzung für eine zuverlässige Segmentierung ist ein deutlicher Helligkeitsunterschied zwischen den relevanten Bildbereichen.

Im Idealfall zeigt das Histogramm des Bildes ein ausgeprägtes Minimum zwischen zwei Häufungen (bimodales Histogramm). In diesem Fall kann ein globaler Schwellwert gewählt werden, der die Grauwertverteilung in zwei sinnvolle Bereiche unterteilt. Die Entscheidung, ob ein Pixel zum Objekt oder zum Hintergrund gehört, erfolgt durch einen einfachen Vergleich mit diesem Schwellenwert.

In der Praxis treten jedoch häufig Störeinflüsse auf, die die Anwendung eines globalen Schwellwerts erschweren. Insbesondere bei wechselnden Lichtverhältnissen oder dynamischen Szenen, wie sie typischerweise in der Echtzeitverarbeitung auftreten. Dazu zählen [6]:

- Shading, ortsabhängige Helligkeitsunterschiede im Bild,
- zeitliche Grauwertveränderungen durch wechselnde Lichtverhältnisse,
- Fremdlicht oder Reflexionen,
- unterschiedliche Materialstärken bei Durchlichtaufnahmen,

Gerade bei zeitkritischen Anwendungen wie der Echtzeit-Bildanalyse, bietet die globale Schwellwertsegmentierung trotz ihrer Einfachheit weiterhin einen relevanten Vorteil. Sie ist äußerst effizient und kann mit geringem Rechenaufwand umgesetzt werden, sofern die Beleuchtungssituation kontrolliert oder stabil genug ist.

Modellbasierte Segmentierung mittels Hough-Transformation

Schwellwertbasierte Segmentierungsverfahren arbeiten typischerweise ausschließlich auf Basis von Grau- oder Farbwerten. In der Praxis sind diese Verfahren jedoch anfällig gegenüber Störeinflüssen wie Beleuchtungsänderungen, Bildrauschen oder Reflexionen, was zu unvollständigen oder fehlerhaften Segmentierungen führen kann [12].

In vielen Anwendungen liegen jedoch bereits Vorabinformationen über die geometrische Struktur der zu erkennenden Objekte vor. Die modellbasierte Segmentierung nutzt dieses Wissen, um gezielt nach bekannten Formen wie Linien, Kreisen oder Ellipsen zu suchen. Ein zentraler Ansatz in diesem Kontext ist die Hough-Transformation.

Die Hough-Transformation basiert auf der Idee, einen Parameterraum (auch Hough-Raum genannt) zu definieren, in dem geometrische Objekte durch ihre Parameter dargestellt werden. Jeder Punkt im Bildraum, der auf einer erkannten Kante liegt (z. B. durch einen vorhergehenden Kantendetektor wie Canny), wird dabei mit allen möglichen Parametern der gesuchten Form assoziiert. Die Suche nach einem Objekt im Bildraum wird damit zu einer Maximumsuche im Hough-Raum.

Zur Erkennung von Geraden wird typischerweise die Hesse'sche Normalform verwendet:

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta \tag{2.11}$$

Hierbei ist ρ der Abstand der Geraden vom Ursprung und θ der Winkel zwischen der Normalen der Geraden und der x-Achse. Jeder Kantenpunkt (x, y) im Bild definiert somit eine Sinuskurve im Hough-Raum (ρ, θ). Schneiden sich viele dieser Kurven in einem Punkt, so deutet dies auf das Vorhandensein einer Geraden im Bild mit den Parametern ρ und θ hin [6].

Diese Methode erlaubt eine robuste Erkennung geometrischer Strukturen, selbst bei fragmentierten Kanten oder geringem Kontrast.

Morphologische Operationen

Die Segmentierung liefert in der Praxis oftmals keine perfekten Ergebnisse. Häufig werden zusammenhängende Objektbereiche fehlerhaft getrennt, obwohl sie semantisch zusammengehören, oder es entstehen unerwünschte, irrelevante Bereiche durch Bildrauschen

oder andere Störeinflüsse. Um solche Fehler gezielt zu korrigieren, kommen morphologische Operationen zum Einsatz [20].

Der Begriff Morphologie stammt ursprünglich aus der Biologie und bezeichnet dort die Lehre von der Struktur und Form von Organismen. In der Bildverarbeitung versteht man darunter Verfahren zur strukturellen Manipulation von Bildsegmenten. Die Operationen werden typischerweise auf Binärbilder oder Grauwertbilder angewendet und dienen der gezielten Veränderung von Objektformen.

Zentrale Operationen sind dabei Erosion und Dilatation. Bei der Erosion wird ein strukturelles Element, pixelweise über das Bild bewegt. Ein Pixel im Ausgangsbild bleibt nur dann erhalten, wenn das Strukturelement vollständig innerhalb des betrachteten Objektbereichs liegt. Mathematisch wird die Erosion einer Bildmenge X mit einem Strukturelement B durch folgende Mengenoperation beschrieben:

$$\varepsilon_B(X) = \{x \mid B_x \subseteq X\} \tag{2.12}$$

Dabei bezeichnet B_x das an die Position x verschobene Strukturelement. Das Ergebnis enthält also genau diejenigen Positionen x, bei denen das Strukturelement vollständig in die Objektmenge X hineinpasst. Die Erosion eignet sich insbesondere zur Eliminierung kleiner Artefakte oder zur Trennung schwach verbundener Objektbereiche.

Das Gegenstück zur Erosion ist die Dilatation. Hier wird ein Pixel bereits dann gesetzt, wenn das Strukturelement an mindestens einer Stelle mit dem Objektbereich überlappt. Mathematisch wird die Dilatation einer Bildmenge X mit einem Strukturelement B durch folgende Mengenoperation beschrieben:

$$\delta_B(X) = \{x \mid B_x \cap X \neq \emptyset\} \tag{2.13}$$

Dabei bezeichnet B_x auch hier das an die Position x verschobene Strukturelement. Das Ergebnis umfasst alle Positionen, an denen das Strukturelement zumindest teilweise mit dem Objektbereich überlappt. Die Dilatation ist somit der zur Erosion bezüglich der Komplementbildung duale Operator.

Mehrfache Anwendungen führen zu einer schrittweisen Ausdehnung der Objektbereiche, was insbesondere zur Verbindung fragmentierter Segmente genutzt werden kann.

In der Praxis werden beide Operationen häufig kombiniert. So wird durch eine aufeinanderfolgende Anwendung von Erosion und Dilatation (Opening) die Trennung eng benachbarter Objekte erreicht. Umgekehrt lassen sich durch Closing (Dilatation gefolgt von Erosion) kleine Lücken schließen oder voneinander getrennte Objekte verbinden. Zusätzlich eignen sich morphologische Operationen auch zur Reduktion von Bildrauschen und zur Glättung von Segmentgrenzen.

2.3.3 Merkmalsextraktion

Ziel der Merkmalsextraktion ist es, aus zuvor segmentierten Bildbereichen gezielt Informationen zu gewinnen, die für folgende Verarbeitungsschritte (wie etwa die Klassifikation oder Positionsbestimmung) verwertbar sind. Dabei handelt es sich bei einem Merkmal (engl. *feature*) in der Regel um eine numerische Kennzahl, die bestimmte Eigenschaften des betrachteten Segments beschreibt [12].

Der Prozess der Merkmalsextraktion (engl. *feature extraction*) bezeichnet die gezielte Berechnung dieser Merkmale auf Basis der Bilddaten. Im Folgenden werden verschiedene Merkmalsarten betrachtet, darunter die Vermessung von Segmenten sowie geometrische Merkmale.

Vermessung von Segmenten

Nach der erfolgreichen Segmentierung eines Objekts stellt die Vermessung dessen Kontur einen wichtigen Schritt zur Beschreibung und Weiterverarbeitung dar. Die Kontur eines Segments beschreibt die Menge an Bildpunkten, die zum Objekt selbst gehören und in deren direkter Nachbarschaft sich mindestens ein Hintergrundpixel befindet [6].

Ein häufig verwendetes Maß ist die Bounding Box, also das kleinst mögliche Rechteck, das alle gesetzten Pixel eines Segments vollständig einschließt. Sie dient als kompakte geometrische Beschreibung des Objekts [6].

Ergänzend dazu kann die konvexe Hülle eines Objekts bestimmt werden. Dabei handelt es sich um das kleinste konvexe Polygon, das alle Punkte des Segments umschließt. Die Berechnung erfolgt beispielsweise mittels des Quick-Hull-Algorithmus und liefert eine abstrahierte Kontur, die in nachfolgenden Schritten wie eine gewöhnliche Objektkontur weiterverarbeitet werden kann.

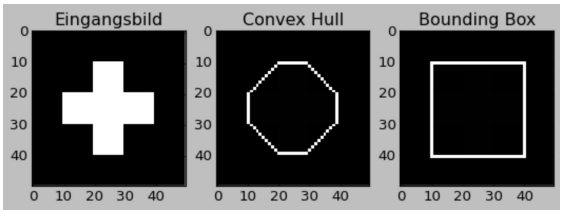


Abbildung 2.2: Beispiel für Konvexe Hülle und Bounding Box [6]

Geometrische Merkmale

Geometrische Merkmale beschreiben grundlegende strukturelle Eigenschaften eines segmentierten Objekts und liefern zentrale Informationen für die weiterführende Analyse.

Fläche: Die Fläche A eines Objekts entspricht der Anzahl aller gesetzten Pixel innerhalb des Segments. Sie berechnet sich zu:

$$A = \sum_{(x,y)\in\text{Segment}} 1 \tag{2.14}$$

Umfang: Der Umfang U eines Objekts beschreibt die Anzahl der Pixel entlang der Kontur:

$$U = \sum_{(x,y)\in\text{Kontur}} 1 \tag{2.15}$$

Kompaktheit: Ein skalierungsinvariantes Maß zur Beschreibung der Form eines Objekts ergibt sich durch das Verhältnis von Fläche zu Umfang im Quadrat:

$$K = \frac{A}{U^2} \tag{2.16}$$

Das maximale Kompaktheitsmaß tritt bei einem idealen Kreis auf. Für diesen ergibt sich:

$$K_{\text{Kreis}} = \frac{\pi r^2}{(2\pi r)^2} = \frac{1}{4\pi} \tag{2.17}$$

Rundheit: Die Rundheit R stellt ein normiertes Kompaktheitsmaß dar und wird durch Bezug auf den Kreis wie folgt definiert:

$$R = \frac{A}{U^2} \cdot \frac{1}{K_{\text{Kreis}}} = \frac{4\pi A}{U^2} \tag{2.18}$$

Ein perfekter Kreis hat einen Rundheitswert von $R = 1$. Unregelmäßigere Formen erreichen entsprechend kleinere Werte.

Dichte: Die Dichte D beschreibt das Verhältnis zwischen der Fläche des Segments und der Fläche seiner konvexen Hülle:

$$D = \frac{\text{contourArea(Segment)}}{\text{contourArea(Hull)}} \tag{2.19}$$

Ein Wert nahe 1 deutet auf ein weitgehend konvexes Objekt hin, niedrigere Werte sprechen für ausgeprägte Konkavität.

2.3.4 Klassifizierung

Unter Klassifizierung versteht man die Zuordnung von Objekten zu bestimmten vordefinierten Kategorien, den Klassen. Im engeren Sinne bezeichnet Klassifikation den systematischen Prozess des Definierens solcher abstrakten Klassen, die zur Abgrenzung und Ordnung von Objekten dienen. In der Bildverarbeitung werden die Begriffe Klassifikation und Klassifizierung häufig synonym verwendet. Im Folgenden wird ausschließlich der Begriff Klassifizierung genutzt [6].

Ziel der Klassifizierung ist es, auf Basis spezifischer Merkmale eines Objekts zu entscheiden, zu welcher Klasse es gehört. Diese Merkmale werden in Form eines Merkmalsvektors zusammengefasst. Dabei handelt es sich um eine strukturierte Sammlung numerischer Werte, die charakteristische Eigenschaften des Objekts abbilden.

Der Merkmalsvektor beschreibt die Position eines Objekts in einem Merkmalsraum. In diesem abstrakten Raum ist jede Achse einem bestimmten Merkmal zugeordnet, und jedes Objekt nimmt entsprechend seiner Merkmalsausprägungen eine eindeutige Position ein. Die Trennung der Klassen erfolgt durch Entscheidungsgrenzen. Diese definieren Flächen im Merkmalsraum, die den Raum in Regionen aufteilen, wobei jede Region einer bestimmten Klasse zugeordnet ist.

Befindet sich ein Objekt mit seinem Merkmalsvektor innerhalb einer Region, so wird es der entsprechenden Klasse zugeordnet. Die Qualität dieser Zuordnung ist jedoch abhängig von der Wahl und Qualität der Merkmale sowie von der Position und Form der Entscheidungsgrenzen [12].

Bewertung der Klassifizierung Die Qualität eines Klassifizierungsverfahrens lässt sich anhand verschiedener Metriken bewerten. Grundlage bildet dabei die Unterscheidung zwischen den tatsächlichen Klassenzugehörigkeiten und den vom System vorhergesagten Klassenzugehörigkeiten. Dies führt zu vier möglichen Ausprägungen [6]:

	Klassifizierung negativ	**Klassifizierung positiv**
Tatsächlich negativ	True Negative (TN)	False Positive (FP)
Tatsächlich positiv	False Negative (FN)	True Positive (TP)

Auf Basis dieser Werte lassen sich verschiedene Kennzahlen berechnen:

- **Accuracy (Genauigkeit)** gibt an, welcher Anteil aller Klassifikationen korrekt ist. Sie berechnet sich wie folgt:

$$\text{Accuracy} = \frac{TP + TN}{n} \tag{2.20}$$

  wobei n die Gesamtanzahl aller Fälle darstellt. Eine hohe Accuracy bedeutet, dass das System insgesamt zuverlässig klassifiziert. Sie sagt jedoch nichts darüber aus, wie gut positive oder negative Klassen jeweils erkannt werden.

- **Precision** beschreibt den Anteil der tatsächlich positiven Objekte unter allen als positiv klassifizierten Objekten:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.21}$$

  Eine hohe Precision bedeutet, dass eine als positiv erkannte Klassenzugehörigkeit mit hoher Wahrscheinlichkeit auch korrekt ist. Das allein garantiert jedoch noch keine umfassende Erkennung aller relevanten Objekte.

- **Recall** gibt an, welcher Anteil der tatsächlich positiven Objekte korrekt erkannt wurde:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.22}$$

  Ein hoher Recall zeigt, dass nahezu alle relevanten Objekte erkannt wurden. Allerdings kann dies auch durch eine großzügige Klassifikation erfolgen, bei der viele irrelevante Objekte fälschlicherweise als positiv erkannt werden.

Je nach Anwendungskontext ist ein ausgewogenes Verhältnis zwischen Precision und Recall entscheidend für eine sinnvolle Klassifizierungsleistung.

2.3.5 HSV-Farbraum

Zur digitalen Repräsentation von Farben existieren verschiedene Farbmodelle, die je nach Anwendung unterschiedliche Vorteile bieten. Während RGB (Rot, Grün, Blau) auf der physikalischen Mischung von Lichtfarben basiert und vorwiegend in technischen Systemen zur Farberzeugung verwendet wird, orientiert sich das HSV-Modell stärker an der menschlichen Farbempfindung. Daher eignet es sich besonders für die farbbasierte Bildanalyse [4].

Das HSV-Modell beschreibt jede Farbe durch drei voneinander getrennte Komponenten:

- **Farbwert (Hue)**: Definiert den Farbton einer Farbe als Winkel auf dem Farbkreis, gemessen in Grad. Typische Referenzpunkte sind: 0° für Rot, 120° für Grün und 240° für Blau. Der Bereich reicht von 0° bis 360°.
- **Sättigung (Saturation)**: Beschreibt die Intensität bzw. Reinheit der Farbe. Ein Wert von 0 % entspricht einem neutralen Grauton, während 100 % eine vollständig gesättigte Farbe ohne Weißanteil beschreibt.
- **Helligkeit (Value)**: Gibt die Leuchtkraft bzw. den Helligkeitsanteil der Farbe an. Der Wert reicht von 0 % (schwarz) bis 100 % (volle Helligkeit).

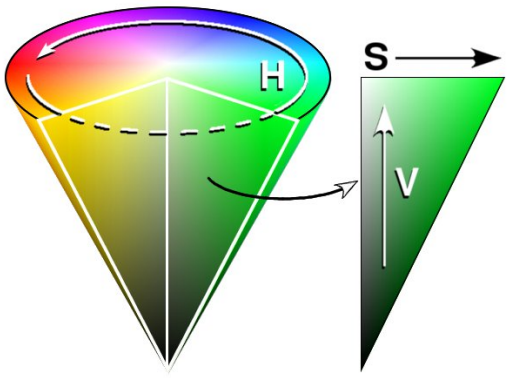


Abbildung 2.3: HSV-Kegel zur Veranschaulichung der Farbparameter Hue, Saturation und Value

Im Gegensatz zum RGB-Modell, bei dem Farbton, Sättigung und Helligkeit eng miteinander verknüpft sind, ermöglicht HSV eine getrennte Betrachtung dieser Eigenschaften. Dies vereinfacht insbesondere Aufgaben wie die Segmentierung nach Farbe, da beispielsweise nur der Farbwert zur Erkennung eines Objekts betrachtet werden kann, unabhängig von Beleuchtung oder Sättigung.

Ein häufig genutztes Darstellungsmodell ist der HSV-Kegel, wie in Abb. 2.3[4] dargestellt. In dieser Visualisierung ist der Farbwert entlang des Umfangs eines Farbkreises angeordnet. Die Sättigung nimmt vom Mittelpunkt nach außen zu, während die Helligkeit entlang der vertikalen Achse variiert. Der Kegel verjüngt sich zur schwarzen Spitze hin und bietet so eine anschauliche Darstellung aller möglichen Farbkombinationen innerhalb des Modells.

Gerade in der Bildverarbeitung ist das HSV-Modell weit verbreitet, da es eine intuitive Filterung und Analyse von Farben ermöglicht.

2.4 Maschinelles Lernen

Maschinelles Lernen (ML) ist ein Teilgebiet der künstlichen Intelligenz, das sich mit Algorithmen befasst, die aus Daten selbstständig Muster erkennen und darauf basierend Entscheidungen ableiten können. Solche Verfahren ermöglichen die Lösung komplexer

[4] https://commons.wikimedia.org/wiki/File:HSV_cone.jpg

Aufgaben, für die keine festen Regeln bekannt sind, die jedoch durch viele beispielhafte Daten beschrieben werden können. Anders als bei regelbasierten Systemen werden die Entscheidungsstrategien dabei nicht vorgegeben, sondern durch Erfahrung aus Trainingsdaten erlernt [8].

Grundlegend lassen sich drei Hauptparadigmen des maschinellen Lernens unterscheiden:

- **Überwachtes Lernen (Supervised Learning):** Modelle lernen aus gelabelten Datenpaaren, d. h. jeder Eingabe ist eine bekannte Zielgröße zugeordnet. Ziel ist es, eine Abbildung zu lernen, die auch bei neuen Eingaben verlässliche Vorhersagen ermöglicht[18].
- **Unüberwachtes Lernen (Unsupervised Learning):** Hier werden ausschließlich Eingabedaten betrachtet, ohne bekannte Zielgrößen. Ziel ist die Strukturierung, Clustering oder Reduktion der Daten durch das Erkennen innerer Zusammenhänge[17].
- **Bestärkendes Lernen (Reinforcement Learning):** Der Lernprozess erfolgt durch Interaktion mit einer Umgebung. Der Agent trifft Entscheidungen, erhält Belohnungen sowie Bestrafungen und optimiert sein Verhalten auf Basis langfristiger Zielerreichung[13].

In dieser Arbeit steht das Reinforcement Learning im Mittelpunkt. Es stellt eine eigenständige Klasse innerhalb des maschinellen Lernens dar, bei der die Datengenerierung dynamisch durch das System selbst erfolgt. Die nachfolgenden Abschnitte widmen sich daher ausschließlich der Struktur und dem Lernverhalten dieser Methode.

2.5 Reinforcement Learning

Reinforcement Learning (RL) ist ein Teilbereich des maschinellen Lernens, der sich mit dem selbstständigen Lernen durch Interaktion mit einer Umgebung beschäftigt. Im Zentrum steht ein sogenannter Agent, der wiederholt mit seiner Umgebung interagiert, um durch Rückmeldung in Form von Belohnungen oder Bestrafungen eine Strategie zu entwickeln, die langfristig optimale Entscheidungen ermöglicht. Ziel ist es, nicht direkt vorgegebene Zielwerte zu reproduzieren, wie es im überwachten Lernen der Fall ist, sondern aus Erfahrungen zu lernen, welche Handlungen in bestimmten Situationen besonders vorteilhaft sind [22].

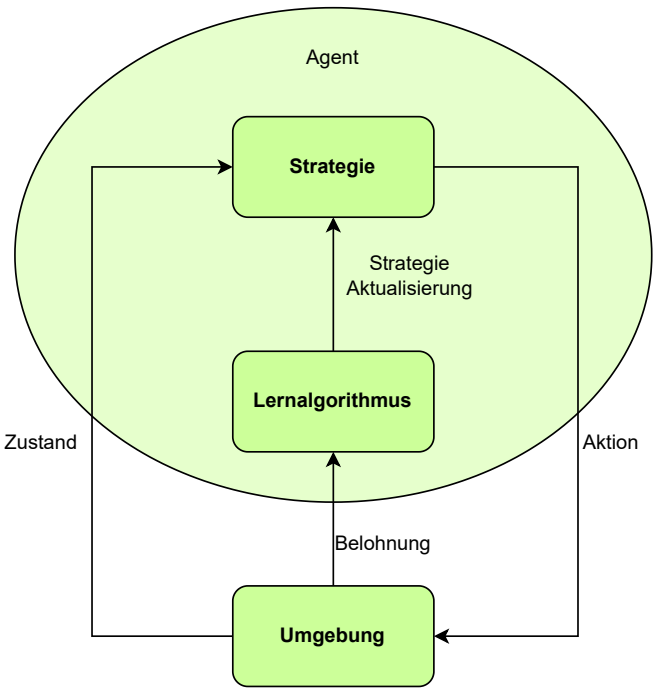


Abbildung 2.4: Grundstruktur eines Reinforcement-Learning-Systems

Die Grundstruktur eines RL-Systems ist in Abbildung 2.4 dargestellt.

Der Agent sieht zu jedem Zeitpunkt einen Zustand, der eine abstrahierte Repräsentation der aktuellen Umgebung darstellt. Auf Basis dieses Zustands wählt er eine Aktion aus, die wiederum eine Veränderung in der Umgebung bewirkt. Als Reaktion auf diese Aktion liefert die Umgebung dem Agenten zwei Rückmeldungen. Den Folgezustand, der aus der Aktion resultiert sowie eine numerische Belohnung (Reward), die ausdrückt, wie günstig oder ungünstig die getroffene Entscheidung war. Ziel des Agenten ist es, eine Policy zu erlernen, eine Strategie, die in jedem möglichen Zustand eine möglichst optimale Aktion bestimmt [22].

Der Lernprozess basiert auf einem iterativen Zyklus. Der Agent beobachtet den Zustand, führt eine Aktion aus, erhält eine Belohnung und aktualisiert daraufhin seine Strategie. Die Qualität einer Strategie bemisst sich dabei nicht ausschließlich an der unmittelbaren Belohnung, sondern berücksichtigt auch zukünftige Belohnungen, die sich aus der aktuellen Entscheidung langfristig ergeben. Eine besondere Herausforderung besteht in der

Balance zwischen Exploration und Exploitation. Einerseits muss der Agent neue, potenziell bessere Aktionen ausprobieren (Exploration), andererseits soll er bereits bekannte, erfolgreiche Handlungen wiederholen (Exploitation).

Der Zyklus aus Zustand, Aktion, Belohnung und Folgezustand wiederholt sich fortlaufend. Ziel des Lernprozesses ist es, eine Policy zu erlernen, die die Summe der erwarteten zukünftigen Belohnungen maximiert. Dabei berücksichtigt der Agent nicht nur unmittelbare Rückmeldungen, sondern auch deren langfristige Auswirkungen. Dieses Prinzip wird durch die Rückdiskontierung zukünftiger Belohnungen beschrieben [9]. Eine häufig verwendete Zielfunktion ist der erwartete Return G_t, definiert als gewichtete Summe künftiger Belohnungen ab einem Zeitpunkt t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.23}$$

Hierbei ist R_{t+k+1} die Belohnung im Zeitschritt $t + k + 1$ und $\gamma \in [0, 1]$ der Diskontfaktor, der angibt, wie stark zukünftige Belohnungen gegenüber unmittelbaren gewichtet werden.

Die in Abbildung 2.4 gezeigten Begriffe lassen sich wie folgt zusammenfassen: Der Agent bezeichnet das lernende System, das eigenständig Entscheidungen trifft. Die Umgebung (Environment) ist das dynamische System, mit dem der Agent interagiert. Zustände (State) repräsentieren die aktuelle Situation der Umgebung, Aktionen (Action) sind mögliche Handlungen des Agenten. Die Belohnung (Reward) ist ein numerisches Feedback über die Qualität der Entscheidung. Die Policy beschreibt die Strategie zur Aktionswahl, während der Lernalgorithmus dafür verantwortlich ist, diese Policy auf Basis der gesammelten Erfahrungen kontinuierlich zu verbessern.

3 Stand der Technik

Die Analyse des aktuellen Stands der Technik bildet eine wesentliche Grundlage für die wissenschaftliche Einordnung der vorliegenden Arbeit. Durch die systematische Betrachtung bestehender Ansätze und Verfahren lassen sich bewährte Lösungsstrategien identifizieren, vorhandene Forschungslücken aufdecken und die eigene Arbeit in den wissenschaftlichen Kontext positionieren. Besondere Bedeutung kommt dabei der Auseinandersetzung mit der Vorgängerarbeit zu, da sie die technische Grundlage für die in dieser Arbeit verfolgten Weiterentwicklungen bildet.

Die Analyse erfolgt in drei Abschnitten. Zunächst erfolgt eine umfassende Analyse des bisherigen Systemstands, wobei sowohl die eingesetzte Hardware als auch die verwendeten Algorithmen und Ergebnisse detailliert betrachtet werden. Anschließend werden relevante wissenschaftliche Arbeiten und bestehende Ansätze vergleichend analysiert. Abschließend werden die Erkenntnisse zusammengefasst und die spezifischen Abgrenzungen der vorliegenden Arbeit zu existierenden Lösungen herausgearbeitet.

3.1 Bisheriger Systemstand

Die vorliegende Arbeit baut auf einem bestehenden System auf, das im Rahmen einer früheren Masterarbeit entwickelt wurde. Im Zentrum dieser Vorarbeit stand die Entwicklung einer virtuellen Umgebung, die als Trainings- und Evaluationsplattform für einen lernfähigen Agenten dient. Ergänzt wurde diese durch die Umsetzung eines physischen Demonstrators, der eine Übertragung der Simulationsergebnisse in die reale Umgebung ermöglicht. Die Ergebnisse der Vorgängerarbeit zeigen bereits ein zum Teil funktionierendes System mit erfolgreichen Lernergebnissen und bilden eine Grundlage für weiterführende Entwicklungen. Zudem bestehen an mehreren Stellen Anknüpfungspunkte zur Optimierung und Erweiterung.

3.1.1 Eingesetzte Hardware

Die zentrale Grundlage für die vorliegende Arbeit bildet der physische Demonstrator, der im Rahmen der vorhergehenden Masterarbeit realisiert wurde [15]. Ein grundlegendes technisches Fundament ist damit bereits vorhanden. Dazu zählen insbesondere das Kugellabyrinth als zentrales Spielfeld, ein kameratragendes Gestell zur Erfassung des Systemzustands sowie die zugehörige elektronische Steuerungseinheit. Im Folgenden werden diese Komponenten systematisch beschrieben.

Den Ausgangspunkt bildet das Kugellabyrinth, welches die physische Umgebung für die Steuerung durch den Agenten bereitstellt. Alle relevanten technischen Details, wie Abmessungen und physikalische Eigenschaften, sind in Kapitel 2 dargestellt. Das Spielfeld selbst blieb im Vergleich zur vorherigen Arbeit unverändert, ebenso die drei vorhandenen Labyrinthmuster. Im Rahmen der vorliegenden Arbeit wurden jedoch gezielte Erweiterungen vorgenommen. So wurde statt der ursprünglich verwendeten silbernen Kugel eine kontrastreichere grüne Kugel eingesetzt. Zudem wurde die Steuerung des Spielfelds so angepasst, dass feste Neigungswinkel in diskreten Schritten angesteuert werden können.

Neben den bestehenden Mustern wurde ein weiteres Labyrinthmuster namens *HOLES_2* entworfen und umgesetzt. Dieses besitzt einen geradlinigen, einfachen Verlauf mit zwei Falllöchern und dient insbesondere dem Testen des Agenten unter reduzierten Bedingungen. Die Wände dieses Labyrinths wurden additiv gefertigt und auf einer kompatiblen Trägerplatte montiert. Das grundlegende Spielprinzip und die physikalische Interaktion mit dem Spielfeld blieben jedoch unverändert. Die Struktur und der Aufbau dieses Labyrinthmusters sind in Abbildung 3.1 dargestellt.

Zur Erfassung des Systemzustands wurde in der Vorgängerarbeit ein kamerabasiertes Verfahren realisiert. Dafür kam ein Aufbau aus Aluminiumprofilen (B-Typ, $20 \times 20\,\mathrm{mm}$, 6-Nut) zum Einsatz, welcher eine stabile Halterung der Kamera in einer Top-Down-Perspektive über dem Spielfeld ermöglicht. Dieses Gestell bildet die Grundlage für die bildbasierte Positionsbestimmung der Kugel, welche dem Agenten als Eingabeinformation dient. Eine Abbildung des verwendeten Kameragestells ist in Abbildung 3.2 dargestellt.

Die elektronische Steuerung basiert auf einem *Arduino UNO R3* in Kombination mit einem Servotreiber und zwei Servomotoren, die für die gezielte Neigung des Spielfelds

Abbildung 3.1: Labyrinthmuster HOLES_2 [15]

verantwortlich sind. Zusätzlich wird ein Tiefsetzsteller zur Spannungsregulierung eingesetzt. Zur visuellen Erfassung kommt eine *W06 1080P HD Webcam* zum Einsatz. Detaillierte technische Spezifikationen dieser Komponenten sind in der ursprünglichen Arbeit dokumentiert [15].

3.1.2 Verwendete Algorithmen und Verfahren

Der Schwerpunkt der vorhergehenden Arbeit lag auf der Entwicklung einer virtuellen Umgebung, die das reale Kugellabyrinth einschließlich aller vorhandenen Labyrinthmuster (mit Ausnahme des Musters *HOLES_52*) digital abbildet. Diese Umgebung dient als Trainings- und Testumgebung für selbstlernende Agenten.

Die entwickelte Simulationsumgebung ermöglicht eine realitätsnahe Nachbildung der physikalischen Eigenschaften des Spiels und stellt dem Agenten strukturierte Zustandsinformationen zur Verfügung. Darauf aufbauend wurden lernfähige Steuerungsverfahren auf Basis von Deep Reinforcement Learning implementiert, die durch Interaktion mit der Umgebung optimiert werden.

Virtuelle Umgebung

Die virtuelle Umgebung wurde in Python mithilfe der Bibliothek VPython umgesetzt. Ziel war es, eine physikalisch realitätsnahe Simulation des Kugellabyrinths zu erstellen,

Abbildung 3.2: Kameragestell mit befestigter Webcam zur Top-Down-Erfassung [15]

die sowohl zur Entwicklung als auch zur Evaluierung lernfähiger Agenten geeignet ist. Das Labyrinth wurde in seiner Geometrie originalgetreu nachgebildet und als dreidimensionales Modell gerendert. Die Umgebung erlaubt die interaktive Einstellung von Neigungswinkeln entlang der beiden Hauptachsen, wodurch die Steuerung der Kugel durch den Agenten oder manuell erfolgen kann.

Im Rahmen der Implementierung wurden insgesamt sechs verschiedene Labyrinthmuster digital umgesetzt. Die physikalische Simulation umfasst wesentliche Aspekte des realen Spiels, darunter die Kollision der Kugel mit den Spielfeldwänden, das Fallen in Löcher sowie die kontinuierliche Bewegung auf der Oberfläche unter Einfluss der Schwerkraft. Eine Ansicht der virtuellen Umgebung mit dem Labyrinthmuster *HOLE_8* ist in Abbildung 3.3 dargestellt.

Deep Reinforcement Learning Agent

Für die agentenbasierte Steuerung des Kugellabyrinths wird ein Deep-Q-Learning-Ansatz umgesetzt. Grundlage bildet dabei eine Vielzahl an Trainingsdurchläufen in der virtuellen Umgebung, bei denen der Agent schrittweise lernt, das Labyrinth zu bewältigen. Das

Abbildung 3.3: Virtuelle Umgebung mit Labyrinthmuster *HOLE_8* [15]

Verhalten verbessert sich durch wiederholte Episoden, in denen das System Erfahrungen sammelt und auswertet.

Zur Unterstützung des Lernprozesses kommt ein Replay Buffer zum Einsatz, der vergangene Erfahrungen speichert und für das Training nutzt. Die Auswahl von Aktionen erfolgt über eine ε-Greedy-Strategie, die im Laufe des Trainings angepasst wird. Dabei wird mit einer Wahrscheinlichkeit von ε eine zufällige Aktion zur Exploration gewählt und mit einer Wahrscheinlichkeit von $1 - \varepsilon$ die aktuell beste bekannte Aktion zur Ausnutzung.

Für das Training und die Evaluation steht eine grafische Benutzeroberfläche zur Verfügung, über die relevante Parameter wie Lernrate, Episodenzahl oder Netzwerkstruktur festgelegt werden können. Trainierte Agenten lassen sich speichern und zu einem späteren Zeitpunkt erneut laden, um bestehende Modelle weiterzuverwenden oder zu analysieren.

Für jedes Muster werden separate Agenten trainiert. Für das komplexe Muster *HOLE_21* kommen drei verschiedene Agenten zum Einsatz, die jeweils auf ein bestimmtes Drittel des Spielfelds spezialisiert sind.

3.2 Vergleich bestehender Ansätze

Der Ansatz, einen Reinforcement-Learning-Agenten zur autonomen Lösung eines Kugellabyrinths zu trainieren, wurde bereits in verschiedenen wissenschaftlichen Arbeiten verfolgt. Dabei kamen jeweils unterschiedliche Rahmenbedingungen, Simulationsumgebungen und technische Umsetzungen zum Einsatz. Auch die im Rahmen dieser Arbeit weiterentwickelte Vorgängerarbeit orientiert sich grundsätzlich an diesem Prinzip.

Im Unterschied zu bestehenden Studien steht in der vorliegenden Analyse jedoch nicht die methodische Grundsatzentscheidung im Fokus, sondern die Betrachtung und Bewertung relevanter technischer und konzeptioneller Merkmale innerhalb bereits definierter Systemgrenzen. Da zentrale Elemente wie Aufbau, Sensorik und Lernverfahren bereits vorgegeben sind, richtet sich das Augenmerk auf vergleichbare Systeme unter ähnlichen Bedingungen.

Die Auswahl der betrachteten Arbeiten erfolgt gezielt entlang zweier Schwerpunkte. Zum einen werden Systeme analysiert, bei denen bildbasierte Zustandsbestimmung zur Steuerung eines physikalischen Systems eingesetzt wird. Zum anderen liegt der Fokus auf Ansätzen, bei denen ein in der Simulation trainierter Agent erfolgreich auf ein reales System übertragen wurde (Sim-to-Real-Ansätze). Im Folgenden werden drei wissenschaftliche Arbeiten im Hinblick auf diese Fragestellungen analysiert und mit der vorliegenden Systemarchitektur verglichen.

3.2.1 Deutsches Forschungszentrum für künstliche Intelligenz

Die Arbeit des Deutschen Forschungszentrums für künstliche Intelligenz (kurz: DFKI) beschreibt ein System zur agentenbasierten Steuerung eines Kugellabyrinths mittels Reinforcement Learning [16]. Zur Erfassung des Systemzustands kommt ein kamerabasiertes Verfahren zum Einsatz, welches die Kugelposition über Bildverarbeitung bestimmt. Ergänzend dazu werden Potentiometer zur Messung der Plattenwinkel sowie ein Piezo-Sensor zur Detektion von Kugelfällen eingesetzt.

Das Training der Steuerung erfolgt vollständig in einer physikalisch realitätsnahen Simulation, die mithilfe der Open Dynamics Engine (ODE) umgesetzt wurde. Das Labyrinth ist als dreidimensionales Modell abgebildet und erlaubt eine Replikation der realen Dynamik. Zur Verbesserung der Übereinstimmung zwischen Simulation und Realität wurden

zentrale Parameter wie Reibung, Ansprechverhalten der Aktuatoren oder Reglerkennwerte automatisch kalibriert. Zusätzlich wurde gezielt Sensorrauschen simuliert, um realistische Trainingsbedingungen zu schaffen.

Als Lernverfahren kommt der SARSA(λ)-Algorithmus zum Einsatz. Die Aktionen bestehen aus diskreten Änderungen der Plattenneigung. Das Belohnungssystem sieht neben einer hohen Endbelohnung für das Erreichen des Ziels auch kleinere Belohnungen für das Durchlaufen von Zwischenzielen sowie Strafwerte für Fehlbewegungen vor. Erste Experimente zeigen, dass unter geeigneten Parametern ein erfolgreicher Lernfortschritt möglich ist und die Simulation ein effektives Training erlaubt. Der Transfer auf das reale System wird aufgrund unvermeidbarer Unterschiede zwischen Simulation und Realität als herausfordernd, aber prinzipiell realisierbar beschrieben.

3.2.2 Eidgenössisch Technische Hochschule Zürich

Die Arbeit der Eidgenössisch Technischen Hochschule Zürich (kurz: ETH Zürich) stellt einen weiteren modellbasierten Reinforcement-Learning-Ansatz zur Lösung des Kugellabyrinths auf einem realen physikalischen System vor. Auch hier dient ein Kamerasystem zur Zustandserfassung des Labyrinths. Aus den Kamerabildern wird ein zentrierter Bildausschnitt der Umgebung extrahiert, der lokal strukturelle Merkmale wie Wände oder Falllöcher enthält [3].

Zur bildbasierten Zustandserfassung wird ein Top-Down-Kamerasystem eingesetzt, das die aktuelle Position der Kugel auf dem Spielfeld detektiert. Hierfür wurde ein Bildverarbeitungsansatz realisiert, der die perspektivische Verzerrung des Kamerabilds kompensiert und die relevante Spielfläche extrahiert. Zur Kalibrierung und exakten Zuschneidung des Bildausschnitts wurden in den vier Ecken des Labyrinths visuell markante blaue Punkte aufgebracht. Diese dienen als Referenzpunkte, um das Spielfeld geometrisch zu entzerren und in ein einheitliches Koordinatensystem zu überführen. Ein Beispiel eines solchen kalibrierten und zugeschnittenen Spielfeldbildes ist in Abbildung 3.4 dargestellt.

Das Training erfolgt ausschließlich auf dem realen System. Als Lernalgorithmus wird DreamerV3 verwendet. Zur Verbesserung der Generalisierungsfähigkeit kommen symmetrieerhaltende Datenaugmentierungsmethoden zum Einsatz, indem Beobachtungen während des Trainings entlang zweier Symmetrieachsen gespiegelt werden. Das Belohnungssignal basiert auf dem Fortschritt entlang des Pfads im Labyrinth.

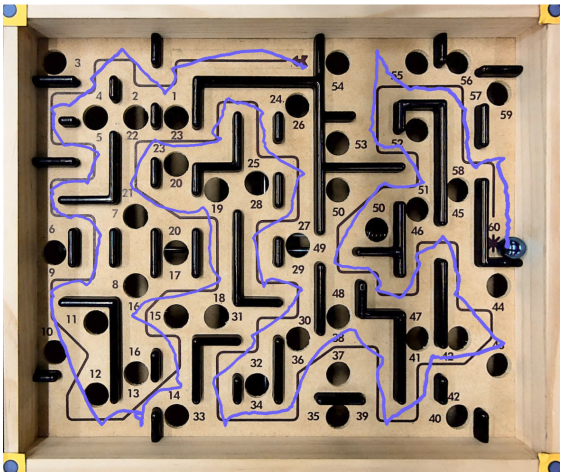


Abbildung 3.4: Kamerabild des zugeschnittenen Spielfelds mit blauen Referenzpunkten [3]

Mit fünf Stunden Trainingsdaten wurde eine robuste Steuerung erlernt, die das Spiel mit einer Erfolgsrate von 76 % abschließt.

3.2.3 Turtlebot Navigation

Ein relevanter Beitrag im Kontext des simulativen Trainings und der anschließenden Übertragung auf reale Systeme stammt aus dem Bereich der mobilen Robotik. In der zugrunde liegenden Arbeit wurde ein Deep-Q-Network-Ansatz entwickelt, um einen mobilen Roboter auf Basis visueller Beobachtungen autonom durch eine Umgebung zu navigieren, ohne zuvor eine Umgebungskarte zu generieren [7]. Aufgrund des hohen Datenbedarfs für das Training wurde das gesamte Lernverfahren zunächst in der Simulationsumgebung Gazebo durchgeführt. Im Anschluss daran konnte das trainierte Modell erfolgreich auf ein reales Robotersystem, bestehend aus einem *Turtlebot2* und einer Kamera, übertragen werden.

Zentrale Bestandteile der Methode waren die Verarbeitung von Kamerabildern zur Zustandserfassung sowie die direkte Abbildung dieser Bilder auf Steueraktionen mittels des DQN. Der Transfer vom simulierten auf das reale System gelang ohne weiteres Finetuning, wobei der Roboter in der Lage war, sich autonom fortzubewegen, Hindernissen auszuweichen und ein definiertes Ziel anzusteuern. Dies gelang auch unter realen Umgebungsbedingungen, die durch Störungen und sensorische Unsicherheiten geprägt waren.

Tabelle 3.1: Vergleich relevanter methodischer Aspekte

Vergleichskriterium	**DFKI**	**ETH Zürich**	**Turtlebot Navigation**
Systemtyp	Kugellabyrinth	Kugellabyrinth	Mobiler Roboter
Sensorik zur Zustandserfassung	RGB-Kamera (Top-Down) + Piezo-Sensor	RGB-Kamera (Top-Down)	RGB-Kamera (First-Person)
Informationsextraktion	Klassische Bildverarbeitung	Klassische Bildverarbeitung	Direkte Verarbeitung des Kamerabilds
Trainingsumgebung	Simulation + reales System	Reales System	Simulation + reales System
Sim-to-Real Transfer	nicht durchgeführt	nicht durchgeführt	Erfolgreich

Die DQN-Architektur ist prädestiniert für visuelle Eingaben. In Kombination mit einem großen Replay Buffer und einer langen Trainingsdauer von rund 500 000 Episoden konnte ein stabiler Lernfortschritt erzielt werden.

3.3 Zusammenfassung und Abgrenzung

Die ausgewählten Arbeiten zeigen unterschiedliche methodische und konzeptionelle Herangehensweisen an ähnliche Problemstellungen. Zwei der Arbeiten basierten auf einem Kugellabyrinth als physikalischer Plattform, während eine dritte auf ein mobiles Robotersystem fokussierte. Trotz variierender Anwendungsfälle lassen sich relevante Gemeinsamkeiten hinsichtlich der Sensorik, der eingesetzten Lernverfahren und der Strategien zur Datenverarbeitung erkennen.

Zur besseren Übersicht werden die zentralen Vergleichskriterien in Tabelle 3.1 zusammengefasst.

Aus den analysierten Arbeiten lassen sich verschiedene Erkenntnisse für die vorliegende Arbeit ableiten. Insbesondere die Nutzung von Landmarken zur Positionserkennung, wie blaue Markierungen an den Spielfeldrändern sowie die Verwendung einer auffälligen Kugelfarbe zur Verbesserung der Segmentierung haben sich als wirksame Strategien erwiesen. Diese Designentscheidungen wurden in der vorliegenden Arbeit aufgegriffen.

Hinsichtlich der Abgrenzung lässt sich festhalten, dass bisher keine der betrachteten Arbeiten einen vollständigen Sim-to-Real-Ansatz für ein Kugellabyrinthsystem unter Anwendung von Deep Reinforcement Learning verfolgt hat. Genau diese Lücke adressiert die vorliegende Arbeit. Die nachfolgenden Kapitel dokumentieren die hierzu entwickelten Verfahren, experimentellen Erweiterungen und Evaluierungsergebnisse im Detail.

4 Anforderungsanalyse

Die Anforderungsanalyse dient der systematischen Erfassung und Strukturierung aller relevanten Anforderungen an ein technisches System. Sie stellt sicher, dass die späteren Entwicklungs- und Implementierungsschritte auf einer klar definierten Basis erfolgen und die funktionalen sowie nicht-funktionalen Anforderungen vollständig berücksichtigt werden.

Die Analyse wird in zwei Teilbereiche untergliedert. Zunächst erfolgt die Stakeholder-Analyse, in der alle beteiligten Interessengruppen identifiziert und ihre Anforderungen sowie Wünsche an das Endprodukt dokumentiert werden. Im Anschluss wird der physische Demonstrator analysiert, wobei der Fokus auf der Erhebung von Anforderungen zur Erfassung des Systemzustandes liegt. Diese strukturierte Herangehensweise gewährleistet eine umfassende Anforderungserhebung und bildet die Grundlage für eine zielgerichtete Entwicklung des Systems.

Vor der Betrachtung dieser Teilbereiche wird zunächst auf die Zusammenhänge zwischen der virtuellen und der physischen Umgebung eingegangen, um ein einheitliches Verständnis der Systemkomponenten und ihrer Interaktionen zu schaffen.

4.1 Systemumgebung

Zur Analyse und Beschreibung der Systemumgebung werden die Hauptkomponenten des Gesamtsystems betrachtet. Diese umfassen den Agenten, das physische System und die virtuelle Umgebung. Die Interaktionen und Zusammenhänge dieser Elemente bilden die Grundlage für die Funktionsweise des Systems und sind erforderlich, um die Anforderungen zu formulieren.

Der Agent stellt die zentrale Steuerungseinheit dar. Es handelt sich dabei um eine künstliche Intelligenz, die Entscheidungen trifft, um die Kugel im Labyrinth vom Startpunkt sicher zum Ziel zu bewegen.

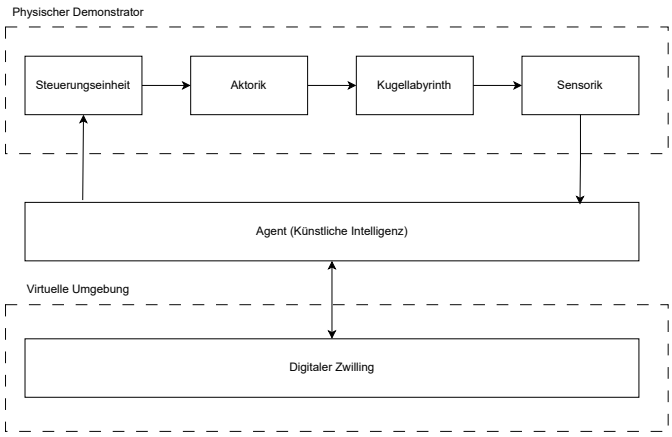


Abbildung 4.1: Systemumgebung mit Interaktion zwischen Agent, physischem System und virtueller Umgebung

Wie in Abbildung 4.1 dargestellt, befindet sich der Agent im Zentrum des Systems. Er steuert den Zustand sowohl der physischen als auch der virtuellen Umgebung. Die Sensorik liefert dem Agenten relevante Informationen aus der physischen Umgebung, während Steuerbefehle über die Aktorik umgesetzt werden. In der virtuellen Umgebung erfolgen Rückmeldungen direkt über definierte Schnittstellen, die dem Agenten den aktuellen Zustand übermitteln.

4.2 Stakeholder

Der Erfolg eines Projekts hängt nicht nur von der technischen Umsetzung und den funktionalen Anforderungen ab, sondern ebenso von der Berücksichtigung der Erwartungen aller beteiligten Interessengruppen.

Da es sich bei dieser Arbeit um ein hochschulinternes Projekt handelt, dessen beteiligte Stakeholder klar definiert und bekannt sind, wird auf eine vollumfängliche Stakeholder-Analyse verzichtet.

4.2.1 Auftraggeber

Im Rahmen dieser Masterarbeit ist der Auftraggeber, zugleich relevanter Stakeholder, Prof. Hensel. Als Erstprüfer ist es für ihn entscheidend, dass die Erwartungen an das

Projekt erfüllt werden. Dies umfasst sowohl die erzielten Ergebnisse als auch eine wissenschaftlich fundierte und sorgfältige Ausarbeitung sowie Dokumentation der Arbeit.

Es wird erwartet, dass die Masterarbeit theoretische und praktische Aspekte kombiniert und zu neuen Erkenntnissen führt. Ein wesentlicher Schwerpunkt liegt auf der Weiterentwicklung des physischen Demonstrators, der bereits in seinen Grundzügen vorhanden ist. Ziel ist es, diesen so zu optimieren, dass er mithilfe künstlicher Intelligenz die Kugel für verschiedene Labyrinth-Muster sicher vom Startpunkt zum Ziel bewegen kann.

Zu Beginn liegt der Fokus auf der präzisen Erkennung der Kugelposition im Labyrinth sowie der effektiven Bestimmung des optimalen Weges zum Ziel. Der Agent soll in der Lage sein, in unterschiedlichen Situationen die passende Entscheidung zu treffen und die Kugel sicher zu navigieren.

Darüber hinaus spielt die visuelle Demonstration der Möglichkeiten von KI eine wichtige Rolle. Der entwickelte Demonstrator soll nicht nur funktional sein, sondern auch als Plattform dienen, auf der KI, Bildverarbeitung, Mechanik und Softwareentwicklung vereint werden. Dadurch eignet er sich sowohl als Präsentationsmittel bei Messen als auch als Lehr- und Qualifikationswerkzeug für Studierende, indem er Anreize für die Auseinandersetzung mit den genannten Themen bietet. Zudem schafft der Demonstrator eine Grundlage für Folgearbeiten und fördert die Weiterentwicklung von KI-Verfahren.

4.2.2 Entwickler

Als Entwickler und Verfasser dieser Abschlussarbeit liegt das Hauptinteresse in der Erfüllung der definierten Anforderungen und im erfolgreichen Abschluss des Projekts. Dabei bestehen Überschneidungen mit den Interessen des Auftraggebers. Neben der Umsetzung der Anforderungen ist eine qualitativ hochwertige Dokumentation wichtig, die Ergebnisse und Vorgehen nachvollziehbar darstellt.

Ein weiteres Ziel ist die gezielte Anwendung des im Studium erworbenen Wissens auf die vorliegende Arbeit. Bestehende theoretische und praktische Kenntnisse sollen eingesetzt, erweitert und neue Fähigkeiten insbesondere im Bereich der Systemanalyse sowie der praktischen Umsetzung technischer Konzepte erworben werden.

4.2.3 Nachfolgende Entwickler

Nach Abschluss des Projekts wird ein Bedarf an Weiterentwicklungen bestehen. Zukünftige Entwicklerinnen und Entwickler müssen in der Lage sein, mithilfe der vorliegenden Arbeit, der Dokumentation und des aktuellen Projektstands effizient weiterzuarbeiten. Von zentraler Bedeutung ist dabei nicht nur eine saubere und detaillierte Dokumentation, sondern auch eine klare, verständliche und modular aufgebaute Software. Diese Eigenschaften gewährleisten Wartbarkeit und Erweiterbarkeit des Systems und bilden die Grundlage für künftige Anpassungen, die Entwicklung neuer Ansätze sowie ein tieferes Verständnis der bisherigen Ergebnisse. Ergänzend dazu trägt ein fundiert formulierter und detaillierter Ausblick wesentlich dazu bei, konkrete Weiterentwicklungspotenziale aufzuzeigen und zukünftigen Projektbeteiligten eine klare Orientierung für nachfolgende Arbeiten zu geben.

4.2.4 Anwender

Da diese Arbeit und das zugehörige Projekt auch der Öffentlichkeit zugänglich gemacht wird, entsteht eine weitere Gruppe von Interessenten. Diese lässt sich in verschiedene Untergruppen mit unterschiedlichen Zielen und Nutzungsmöglichkeiten gliedern.

Zum einen gibt es potenzielle Einzelpersonen, die hobbymäßig an einem Nachbau des Projekts interessiert sind. Für sie bieten die Ergebnisse sowie die Dokumentation eine wertvolle Grundlage für eigene Vorhaben.

Eine weitere Gruppe sind Personen, die ähnliche Projekte realisieren und in diesem Zusammenhang Inspiration aus dem Demonstrator ziehen möchten. Sie sind vor allem an innovativen Lösungsansätzen interessiert, die für eigene Projekte adaptiert werden können.

Schließlich gibt es Studierende oder Studieninteressierte, die sich anhand dieses Projekts ein Bild von den Ergebnissen studentischer Arbeiten machen möchten. Für diese Zielgruppe dient der Demonstrator als Beispiel für Qualität, Umfang und Anwendungsmöglichkeiten aktueller Technologien.

Für alle genannten Gruppen ist eine klare Struktur der Arbeit und eine präzise Umsetzung der Anforderungen entscheidend, um das Projekt nachvollziehen und gegebenenfalls

weiterentwickeln zu können. Damit wird das Projekt nicht nur zu einem praktischen Beispiel, sondern auch zu einer wertvollen Ressource für die technische Weiterentwicklung und den Wissenstransfer.

4.3 Physischer Demonstrator

Die folgenden Unterkapitel befassen sich detailliert mit den Anwendungsfällen und den Anforderungen des physischen Demonstrators.

Da diese Arbeit auf einem bestehenden Projekt aufbaut, knüpft sie an die in der Vorgängerarbeit definierten Systemkomponenten und Rahmenbedingungen an [15]. Der physische Demonstrator bildet dabei das zentrale reale System, anhand dessen die Funktionsweise, Struktur und Anforderungen des Gesamtsystems untersucht werden.

4.3.1 Anwendungsfälle

In diesem Unterkapitel werden die spezifischen Anwendungsfälle des physischen Demonstrators im Detail analysiert. Abbildung 4.2 zeigt ein Anwendungsfalldiagramm, das die verschiedenen Anwendungsfälle sowie deren Beziehungen zueinander übersichtlich darstellt. Im Folgenden werden die dargestellten Anwendungsfälle erläutert.

Abbildung 4.2 zeigt die wesentlichen Akteure und Interaktionen. Die gestrichelten Linien ohne Beschriftung stellen implizite «include»-Beziehungen dar.

Zu den zentralen Akteuren zählen der Anwender, die Sensorik und die Aktorik. Die Sensorik erfasst physikalische Zustände des Systems, beispielsweise das Spielfeld oder die Position der Kugel. Die Aktorik ermöglicht die aktive Beeinflussung des Systems, indem sie die Neigung des Spielfelds verändert. Der Anwender kann den Demonstrator bedienen, konfigurieren und verschiedene Betriebsabläufe initiieren.

Ein wesentlicher Anwendungsfall ist die Kalibrierung des Systems. Dabei wird das Spielfeld in eine definierte Nullstellung (0°-Neigung in beiden Achsen) gebracht, um konsistente Ausgangsbedingungen zu gewährleisten. Die Auswahl des zu verwendenden Labyrinths erfolgt im Rahmen der Systeminitialisierung. Ein Spiel kann durch das Erreichen des Zielpunkts, das Hineinfallen der Kugel in ein Loch oder durch eine manuelle Unterbrechung

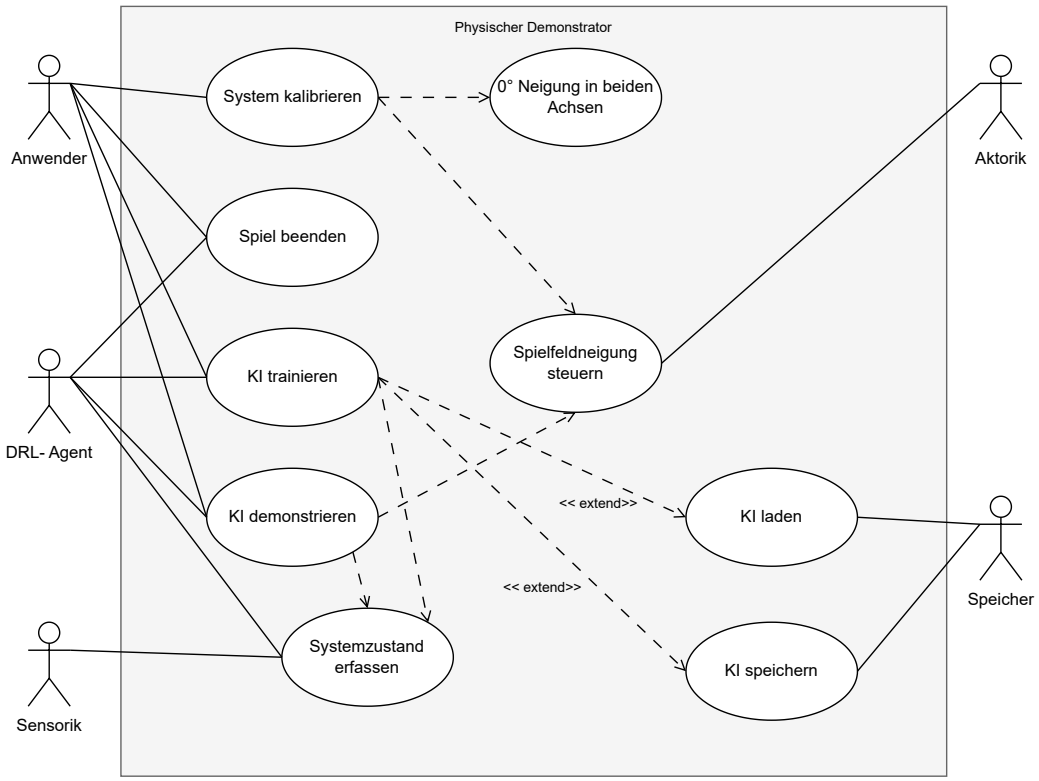


Abbildung 4.2: Use-Case-Diagramm des physischen Demonstrators

beendet werden. Dieser Vorgang wird durch den Anwendungsfall „Spiel beenden“ abgebildet.

Die Erfassung und Auswertung des Systemzustands bilden die Grundlage für zukünftige Automatisierungs- oder Regelungsansätze. Details zur genauen Umsetzung dieser Zustandsaufnahme werden in Kapitel 5 beschrieben.

4.3.2 Anforderungserhebung

Für die Definition der Anforderungen des physischen Demonstrators (PD) wird eine strukturierte, an ISO 9126 [11] orientierte Vorgehensweise angewendet. Die Anforderungen basieren auf der Analyse der Systemumgebung, der identifizierten Stakeholder sowie

den relevanten Anwendungsfällen. Sie werden in funktionale („FA“) und nicht-funktionale („NFA“) Anforderungen unterteilt.

Funktionale Anforderungen

Funktionale Anforderungen haben einen direkten Bezug zur Zweckbestimmung des Systems. Sie beschreiben die konkreten Funktionen, die der Demonstrator bereitstellen muss.

- **PD-FA1: Export der Zustandsdaten**
  Das System muss die erfassten Zustandsinformationen in einem standardisierten Format exportieren können, um weitere Analysen zu ermöglichen.
- **PD-FA2: Manueller Spielstart**
  Das System muss eine manuelle Startfunktion bereitstellen, mit der der Benutzer den Ablauf initialisieren und den Prozess aktiv starten kann.
- **PD-FA3: Erfassung der Kugelposition**
  Das System muss die Position der Kugel auf dem Spielfeld lückenlos erfassen und den vollständigen Bewegungsverlauf innerhalb des Spielfeldbereichs korrekt abbilden. Die erfassten Daten müssen für jeden Spielzyklus ohne Aussetzer bereitgestellt werden.
- **PD-FA4: Erfassung des Spielfeldrahmens**
  Das System muss den Spielfeldrahmen identisch zur virtuellen Umgebung erfassen und ausgeben können.
- **PD-FA5: Erfassung von Merkmalen verschiedener Muster**
  Das System muss charakteristische Merkmale unterschiedlicher Muster auf dem Spielfeld erfassen und unterscheiden können.
- **PD-FA6(Opt): Aktive Beleuchtungsunterstützung**
  Das System soll optional eine aktive Beleuchtung (z. B. LED-Panels) unterstützen; Aktivierung und grundlegende Parameter sind konfigurierbar.

Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen betreffen qualitative Eigenschaften des Systems, die über das reine funktionale Verhalten hinausgehen. Die Kategorisierung orientiert sich an etablierten Qualitätsmerkmalen wie Wartbarkeit, Zuverlässigkeit, Reproduzierbarkeit und Systemintegration.

Systemdesign und Integration

- **PD-NFA1: Standardisierte Spielfeldgröße**
  Die Abmessungen des Spielfelds betragen 27,4 cm × 22,8 cm und entsprechen dem Format des eingesetzten Labyrinths.
- **PD-NFA2: Spezifikation der Spielkugel**
  Für den Spielbetrieb ist eine Stahlkugel mit einem Radius von 6,35 mm und einem Gewicht von 9 g zu verwenden.
- **PD-NFA3: Nicht-invasive Motorintegration**
  Die Motoren müssen so angebracht werden, dass keine dauerhaften oder invasiven Veränderungen an der Struktur des Labyrinths erforderlich sind.
- **PD-NFA4: Kompatibilität selbstgebauter Labyrinthplatten**
  Sollten selbstgefertigte Spielplatten zum Einsatz kommen, müssen die Positionen der Löcher exakt mit denen des originalen Labyrinths übereinstimmen.

Datenqualität und Zuverlässigkeit

- **PD-NFA5: Zeitliche Stabilität der Erfassung**
  Die Erfassungsfrequenz der Sensorik muss konstant bleiben, um reproduzierbare Ergebnisse sicherzustellen.
- **PD-NFA6: Genauigkeit der Positionserfassung**
  Die Erfassung der Kugelposition muss eine ausreichende Genauigkeit aufweisen, um eine präzise Zustandsbestimmung zu ermöglichen. Details zur erreichten Genauigkeit werden im Evaluierungskapitel dargelegt.

- **PD-NFA7: Robustheit gegenüber Umgebungseinflüssen**
  Das System zur Zustandserfassung muss robust gegenüber wechselnden Lichtverhältnissen im normalen Raumbereich funktionieren.
- **PD-NFA8: Zuverlässigkeit der Ereignisdetektion**
  Die Detektion von Spielereignissen muss eine hohe Erkennungsrate aufweisen, um eine verlässliche Systemüberwachung zu gewährleisten.
- **PD-NFA9: Kalibrierbarkeit**
  Alle Sensoren, die zur Zustandserfassung dienen, müssen kalibrierbar sein, um Messfehler systematisch zu kompensieren und langfristige Genauigkeit sicherzustellen.
- **PD-NFA10: Visualisierung der Zustandsdaten**
  Das System soll eine einfache Visualisierung der gemessenen Zustände (z. B. Neigungswinkel, Kugelposition) ermöglichen, um Tests und Analysen zu erleichtern.

Technische Umsetzbarkeit und Wartung

- **PD-NFA11: Strukturierter und kommentierter Quellcode**
  Der entwickelte Quellcode muss klar strukturiert und umfassend kommentiert sein, um Wartbarkeit, Erweiterbarkeit und Nachvollziehbarkeit sicherzustellen.
- **PD-NFA12: Kompaktheit und Transportfähigkeit**
  Der Demonstrator soll kompakt, leicht transportierbar und einfach demontierbar sein, um einen flexiblen Einsatz zu ermöglichen.

Projektbegrenzungen

- **PD-NFA13: Budgetbeschränkung**
  Die Gesamtkosten für neu zu beschaffende Komponenten dürfen im Rahmen dieses Projekts 300 € nicht überschreiten.

5 Konzept

Im vorherigen Kapitel wurden die funktionalen und nicht-funktionalen Anforderungen an das zu entwickelnde System detailliert beschrieben. Diese bilden die Grundlage für das nun folgende Konzept, da die spätere Umsetzung direkt auf diesen Anforderungen aufbauen muss. Die enge Verbindung zwischen Anforderungsanalyse und Konzeption ist essenziell, um bereits in der Entwurfsphase sicherzustellen, dass alle relevanten Rahmenbedingungen berücksichtigt werden.

In diesem Kapitel wird ein technisches Konzept erarbeitet, das als Grundlage für die Realisierung des in Kapitel 1 beschriebenen Ziels dient.

Der Aufbau des Kapitels gliedert sich in zwei Hauptteile. Zunächst wird die Systemarchitektur beschrieben, welche die zentralen Komponenten sowie deren Zusammenspiel darstellt. Im Anschluss werden verschiedene methodische Ansätze analysiert, verglichen und im Hinblick auf die zuvor formulierten Anforderungen bewertet. Auf diese Weise soll ein fundiertes Gesamtkonzept entstehen, das als Ausgangspunkt für die Entwicklung des KI-Systems dient.

5.1 Systemarchitektur

Wie in Abbildung 4.1 dargestellt, setzt sich das System aus mehreren zentralen Komponenten zusammen, die sowohl in der realen als auch in der virtuellen Umgebung koordiniert miteinander arbeiten. Zur Verdeutlichung des Systemkonzepts wird in Abbildung 5.1 ein Komponentendiagramm verwendet, das die verschiedenen Systemkomponenten sowie deren jeweilige Schnittstellen zu anderen Komponenten abbildet.

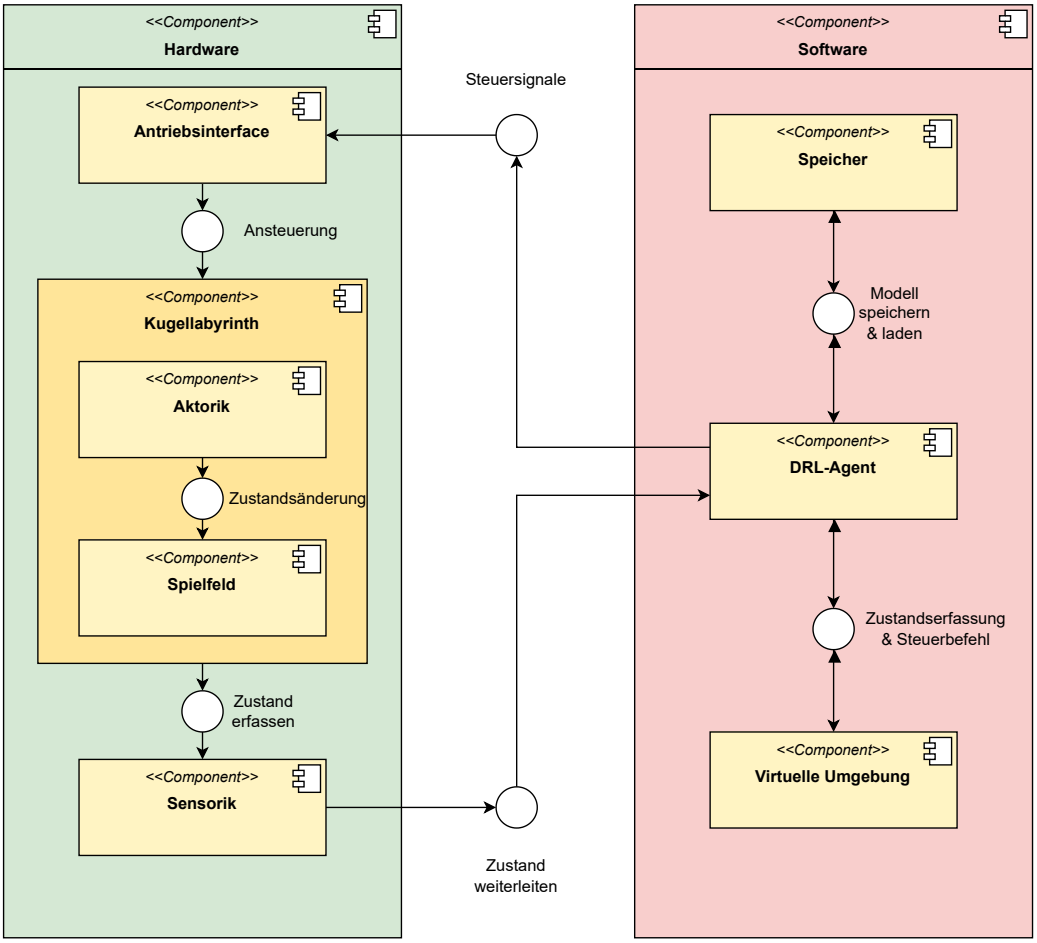


Abbildung 5.1: Komponentendiagramm

Das Komponentendiagramm unterteilt das System in Hardware- und Softwarekomponenten, die entsprechend ihres funktionalen Ablaufs von links nach rechts angeordnet sind.

Hardware: Der Ablauf beginnt bei der Steuerungseinheit, welche die Steuerbefehle entgegennimmt und über die Ansteuerung an die Aktorik weiterleitet. Die Aktorik ist für die physikalische Bewegung innerhalb des Kugellabyrinths verantwortlich.

Teil dieses Kugellabyrinths ist auch das Spielfeld, dessen Zustand sich in Abhängigkeit der ausgeführten Bewegungen verändert. Diese Zustandsänderungen werden durch die Sensorik erfasst, welche den aktuellen Zustand wiederum an den RL-Agenten zurückmeldet.

Software: Die erfassten Zustände werden in der Softwarekomponente ausgewertet. Dort verarbeitet der Agent die Informationen und gibt auf Basis seiner Bewertung neue Steuerbefehle aus. Parallel dazu existiert eine virtuelle Umgebung, in der dieselben Prozesse simuliert werden können. Ein Speicher ermöglicht das Laden und Sichern von Modellen, um Lernfortschritte zu speichern oder bestehende Strategien erneut einzusetzen.

Das Zusammenspiel dieser Komponenten ermöglicht sowohl den Betrieb eines physischen Demonstrators als auch den Einsatz einer simulierten Variante innerhalb desselben architektonischen Rahmens.

Im weiteren Verlauf wird dargelegt, wie die Bewegungssteuerung des realen Kugellabyrinths realisiert wird und auf welche Weise sensorische Informationen aus dem System extrahiert werden, um dem Agenten eine situationsabhängige Entscheidungsfindung zu ermöglichen.

5.2 Hardware

5.2.1 Sensorische Erfassung des Labyrinthzustands

Die präzise Kenntnis des aktuellen Zustands des Labyrinths stellt eine zentrale Voraussetzung für die erfolgreiche Steuerung durch das Agentensystem dar (PD-FA5). Dieser Zustand umfasst verschiedene relevante Aspekte.

Zur Erfassung dieser Informationen existieren unterschiedliche technische Ansätze. Die geeignetste Lösung stellt jedoch der Einsatz eines Kamerasystems in Kombination mit bildverarbeitenden Verfahren dar. Diese Wahl begründet sich durch die Vielzahl an zu erfassenden Merkmalen, die jeweils unterschiedliche Eigenschaften und Anforderungen mit sich bringen. Während der Einsatz spezialisierter Sensoren für einzelne Merkmale potenziell eine höhere Genauigkeit ermöglichen könnte, würde dies sowohl den technischen Aufwand als auch die Kosten sowie die Komplexität des Systemaufbaus signifikant

erhöhen, was sich negativ auf die Wartbarkeit (PD-NFA11), die Transportfähigkeit (PD-NFA12) sowie die Budgetbeschränkung (PD-NFA13) auswirken könnte.

Die Bildverarbeitung erlaubt hingegen eine simultane Erfassung aller relevanten Aspekte des Labyrinthzustands mittels eines Sensors, wodurch sowohl die Systemintegration (PD-NFA3) als auch die Kompatibilität mit verschiedenen Spielplatten (PD-NFA4) begünstigt wird. Darüber hinaus handelt es sich um eine etablierte Methode zur Umgebungswahrnehmung in vergleichbaren Anwendungsbereichen (siehe Kapitel 3). Die Positionierung der Kamera muss dabei so gewählt werden, dass sämtliche erforderlichen Informationen in einem Aufnahmebereich enthalten sind und dem Agenten zur Verfügung gestellt werden können. Auf diese Weise lässt sich der Zustand des Labyrinths vollständig und effizient erfassen, was eine präzise Entscheidungsfindung durch den Agenten gemäß PD-FA1 und PD-FA6 unterstützt.

Kameraauswahl

Für die bildbasierte Zustandserfassung des Labyrinths stehen grundsätzlich verschiedene Bildaufnahmesysteme zur Verfügung, die je nach Anwendungsfall unterschiedliche Stärken aufweisen. Im vorliegenden Kontext kommen insbesondere zwei Kameratypen in Betracht. Herkömmliche RGB-Kameras, welche farbige zweidimensionale Bilddaten liefern, oder Tiefenkameras, die zusätzlich auch Distanzdaten erfassen. Beide Systeme unterscheiden sich hinsichtlich ihrer technologischen Eigenschaften, ihrer Eignung zur Objekterkennung sowie ihres Integrationsaufwands.

RGB-Kamera: RGB-Kameras liefern hochauflösende Farbinformationen, wodurch eine direkte farbbasierte Objekterkennung ermöglicht wird. Sie zeichnen sich durch geringe Kosten, eine breite Verfügbarkeit sowie eine einfache Integration aus. Darüber hinaus existiert eine große Auswahl an etablierten Bildverarbeitungsbibliotheken, was ihre Einbindung in bestehende Systeme erleichtert. Für Anwendungen, bei denen keine Tiefeninformationen benötigt werden, stellt die RGB-Kamera eine praktikable und ressourcenschonende Lösung dar.

Zu den Nachteilen zählt, dass keine Tiefeninformationen erfasst werden können. Dies erschwert die Erkennung dreidimensionaler Strukturen im Labyrinth, insbesondere bei Objektverdeckungen. Zudem kann die Erkennungsgenauigkeit bei ungünstigen Lichtverhältnissen oder Schattenwurf beeinträchtigt sein.

Tabelle 5.1: Vergleich der RGB-Kamera und Tiefenkamera anhand relevanter Bewertungskriterien

Kriterium	**RGB-Kamera**	**Tiefenkamera**
Farbwahrnehmung	++	+
Kosten	++	--
Verfügbarkeit	++	o
Einfachheit der Integration	+	-
Auflösung	+	o
Tiefenerfassung	++	++
Robustheit gegenüber Lichtbedingungen	o	+
Echtzeitfähigkeit	+	o
Erkennung von Löchern/Wänden	-	+
Kompatibilität mit Bibliotheken	++	+

Tiefenkamera: Tiefenkameras erfassen neben der Farb- oder Intensitätsinformation auch die Distanz zwischen Kamera und Objekt, was eine dreidimensionale Analyse der Szene ermöglicht. Dadurch können Position und Form von Objekten präziser erkannt und vom Hintergrund separiert werden. Tiefenkameras sind besonders robust gegenüber wechselnden Lichtverhältnissen und ermöglichen darüber hinaus die Erkennung von Höhenunterschieden im Labyrinth, was insbesondere zur Identifikation von Löchern oder Wandstrukturen von Vorteil ist.

Demgegenüber stehen höhere Anschaffungskosten sowie erweiterte Integrations- und Parametrierungsanforderungen. Außerdem verfügen Tiefenkameras oft über geringere Auflösung und verlangen aufwendigere Vorverarbeitung mit entsprechend höherem Rechenbedarf.

Zum Vergleichen beider Optionen wurden die beiden Arten von Kameras in Tabelle 5.1 hinsichtlich verschiedener, für die Anforderungen relevanter Aspekte gegenübergestellt. Die Bewertung erfolgt mit den Symbolen: ++ (sehr gut geeignet), + (gut geeignet), **o** (neutral), **-** (weniger geeignet) und **--** (ungeeignet beziehungsweise nachteilig).

Aufgrund der Vorteile der RGB-Kamera, insbesondere in den Bereichen Farbwahrnehmung, Kosten, Verfügbarkeit und Integration, wurde sich für den Einsatz dieses Kameratyps entschieden. Die hohe Auflösung und die weit verbreiteten Bildverarbeitungsbibliotheken ermöglichen eine effiziente Implementierung, während die RGB-Kamera aufgrund ihrer geringen Kosten und der unkomplizierten Handhabung eine praktikable Lösung für die geplante Anwendung darstellt. Die Wahl steht im Einklang mit den Anforderungen

an die kosteneffiziente Umsetzung (PD-NFA13), die Wartbarkeit (PD-NFA11) sowie die Echtzeitverarbeitung des Zustands (PD-FA5).

Zwar fehlen der RGB-Kamera Tiefeninformationen, jedoch kann dieser Nachteil im vorliegenden Anwendungsfall durch alternative Methoden, wie die farbbasierte Erkennung, ausgeglichen werden. Die Verwendung farblicher Merkmale ermöglicht die zuverlässige Identifikation und Klassifikation von Kugel, Zielbereichen und Hindernissen, ohne dass die physikalischen Grenzen der Spielfeldneigung überschritten oder zusätzliche Sensorik gemäß PD-NFA3 erforderlich wäre.

Die RGB-Kamera muss in der Lage sein, sämtliche relevanten Merkmale des Spielfelds zuverlässig und in Echtzeit zu erfassen. Daraus ergeben sich folgende Anforderungen an die technischen Eigenschaften des Kamerasystems:

- **Auflösung:** Die Auflösung muss ausreichend hoch sein, um sämtliche Objekte des Labyrinths eindeutig zu erkennen und voneinander zu unterscheiden.
- **Bildrate:** Eine hohe Bildfrequenz ist erforderlich, um eine lückenlose Zustandsverfolgung in Echtzeit zu gewährleisten.
- **Sichtfeld:** Die Kamera muss das gesamte Spielfeld vollständig abbilden, sodass keine relevanten Bereiche außerhalb des Sichtfelds liegen.
- **Verzerrungskorrektur:** Bei der Nutzung eines Weitwinkelobjektivs kann es zu geometrischen Verzerrungen kommen. Diese müssen durch geeignete Kalibrierungsverfahren kompensiert werden.
- **Lichtverhältnisse und Farbkontrast:** Die Kamera soll auch unter schwierigen Lichtbedingungen verlässlich arbeiten. Eine Aufnahme im RGB-Farbraum ist erforderlich, um farbbasierte Merkmalsklassifikationen zu ermöglichen.

5.2.2 Steuereinheit

Die Steuereinheit übernimmt eine zentrale Rolle im Gesamtsystem. Sie dient als Schnittstelle zwischen Bildverarbeitung, Entscheidungslogik und Aktorik. Ihre Hauptaufgabe besteht darin, empfangene Steuerbefehle in konkrete Ansteuersignale für die Servomotoren zu überführen und dabei eine zeitnahe, zuverlässige Ausführung sicherzustellen. Der Arduino Uno ist ein kompakter Mikrocontroller mit digitalen und analogen Ein- und Ausgängen, der sich besonders für einfache Steuerungsaufgaben eignet. Aufgrund

seiner geringen Latenz, einfachen Handhabung und stabilen Echtzeiteigenschaften wurde er bereits in früheren Projektphasen erfolgreich zur Ansteuerung der Servomotoren verwendet. Die Notwendigkeit einer dedizierten Steuereinheit ergibt sich aus der Hardware-Architektur des Gesamtsystems. Obwohl die rechenintensive Bildverarbeitung und Entscheidungslogik auf einem leistungsstarken Laptop oder PC ausgeführt werden, verfügen diese Systeme nicht über die erforderlichen GPIO-Schnittstellen zur direkten Ansteuerung von Servomotoren. Der Arduino Uno schließt diese Lücke, indem er als Hardware-Schnittstelle zwischen der Software-Ebene und der physischen Aktorik fungiert.

Für dieses Projekt wird erneut auf den Arduino Uno zurückgegriffen. Die Entscheidung stützt sich vor allem auf zwei Aspekte. Einerseits wurde der Mikrocontroller bereits in den Vorarbeiten erfolgreich eingesetzt und kann somit nahtlos in die bestehende Systemarchitektur eingebunden werden. Andererseits ist der Arduino in Bezug auf Anschaffungskosten (PD-NFA13) und Energieverbrauch deutlich ressourcenschonender, was insbesondere für einfache Steuerungsaufgaben wie die Motoransteuerung vollkommen ausreicht.

5.3 Software

5.3.1 Entwicklungsumgebung und Programmiersprache

Bei der Auswahl der Programmiersprache und Entwicklungsumgebung wurde besonderer Wert auf Kompatibilität und Einheitlichkeit gelegt. Ziel war es, sowohl die Bildverarbeitung als auch die Implementierung des Reinforcement-Learning-Agenten innerhalb derselben Umgebung und Programmiersprache umzusetzen. Aus diesem Grund fiel die Entscheidung auf Python, da diese Sprache sowohl im Bereich der künstlichen Intelligenz als auch in der Bildverarbeitung umfangreiche Unterstützung bietet.

Gleichzeitig verfügt Python über eine Vielzahl an spezialisierten Bibliotheken und Werkzeugen, die sich für bildverarbeitungsbasierte Anwendungen bewährt haben. Für die Bildverarbeitung kommt in diesem Projekt primär die Bibliothek OpenCV zum Einsatz, welche eine leistungsfähige und flexible Schnittstelle zur Bilderfassung und -analyse bietet.

Die Wahl dieser Softwarebasis unterstützt explizit die in PD-NFA6 geforderte Anbindung an den Reinforcement-Learning-Agenten. Darüber hinaus trägt der Einsatz einer

etablierten Programmiersprache mit klarer Modularität und guter Dokumentationsbasis zur Wartbarkeit und Erweiterbarkeit bei (PD-NFA11).

5.3.2 Bildverarbeitungsstrategie

Auf Grundlage der zuvor getroffenen Entscheidungen hinsichtlich der Kameraauswahl sowie der verwendeten Softwarearchitektur soll nun eine geeignete Strategie zur Bildverarbeitung definiert werden. Ziel ist es, visuelle Informationen aus dem Labyrinthsystem zuverlässig zu extrahieren, um diese dem Agenten zur situationsabhängigen Entscheidungsfindung bereitzustellen.

Grundsätzlich lassen sich zwei methodische Ansätze zur Bildverarbeitung unterscheiden: die klassische Bildverarbeitung und Verfahren des maschinellen Lernens im Kontext künstlicher Intelligenz. Beide Ansätze verfolgen unterschiedliche Herangehensweisen zur Objekterkennung und bringen jeweils spezifische Vor- und Nachteile mit sich.

Klassische Bildverarbeitung Die klassische Bildverarbeitung basiert auf einer festen Abfolge von Verarbeitungsschritten: Vorverarbeitung, Segmentierung, Merkmalsextraktion und Klassifizierung. Häufig kommen dabei Funktionen aus *OpenCV* in Python zum Einsatz, die eine effiziente und transparente Umsetzung ermöglichen (siehe Abschnitt 2.3).

Stärken: Die klassische Bildverarbeitung ist durch ihren geringen Rechenaufwand und die gute Interpretierbarkeit besonders vorteilhaft. Ihre deterministischen Algorithmen erlauben eine einfache Kontrolle über Fehlerquellen und nachvollziehbare Ergebnisse.

Schwächen: In komplexen Szenen stößt sie jedoch an ihre Grenzen. Lichtveränderungen, Überlappungen oder andere Umgebungsveränderungen führen schnell zu Fehlern. Zudem ist die Generalisierungsfähigkeit begrenzt, da die Verfahren stark auf feste Parameter angewiesen sind.

Deep Learning in der Bildverarbeitung Moderne Bildverarbeitungsverfahren basieren zunehmend auf Deep-Learning-Methoden, insbesondere auf Convolutional Neural Networks (CNNs). Der typische Entwicklungsprozess beginnt mit der Festlegung eines Referenzmodells sowie der Beschaffung und Aufbereitung geeigneter Bilddaten. Dabei ist

Tabelle 5.2: Vergleich der klassischen Bildverarbeitung und Deep Learning hinsichtlich relevanter Kriterien

Kriterium	**KBV**	**DL**
Robustheit bei schwierigen Szenen	-	++
Erkennung komplexer Muster	-	++
Interpretierbarkeit	++	-
Kontrollierbarkeit	++	-
Rechenaufwand	++	–
Trainingsaufwand	++	--
Notwendigkeit großer Datensätze	++	--
Generalisierungsfähigkeit	-	+
Implementierungsaufwand	+	-

eine ausreichende Datenmenge entscheidend, da Deep-Learning-Modelle stark datengetrieben sind. Nach Auswahl einer passenden Netzwerkarchitektur wird ein Qualitätsmaß wie Accuracy, F1-Score oder IoU festgelegt, um den Lernerfolg zu bewerten.

Im Anschluss erfolgt das Training des Modells, wobei eine Vielzahl an Parametern eine Rolle spielt. Durch gezielte Hyperparameteroptimierung kann die Modellleistung weiter gesteigert werden. Typische Bibliotheken in diesem Kontext sind TensorFlow und PyTorch.

Stärken: Deep-Learning-Methoden zeigen eine hohe Robustheit gegenüber komplexen Szenen und sind in der Lage, komplexe Muster und Zusammenhänge in den Bilddaten zu erkennen, die mit klassischen Methoden schwer erfassbar wären.

Schwächen: Dem gegenüber stehen einige Herausforderungen. Das Training erfordert einen großen und qualitativ hochwertigen Datensatz sowie häufig einen erhöhten Rechen- und Hardwareaufwand. Zudem ist die Interpretierbarkeit der Ergebnisse im Vergleich zu klassischen Verfahren eingeschränkt, was die Nachvollziehbarkeit erschwert.

Zum Vergleich beider Ansätze wurden die klassische Bildverarbeitung (KBV) und Deep Learning (DL) in Tabelle 5.2 hinsichtlich verschiedener, für die Anwendung relevanter Kriterien gegenübergestellt. Die Bewertung erfolgt analog zur Kamera-Gegenüberstellung mit den Symbolen: ++ (sehr gut geeignet), + (gut geeignet), **o** (neutral beziehungsweise abhängig vom Kontext), **-** (weniger geeignet) und **--** (ungeeignet beziehungsweise nachteilig).

Für die vorliegende Anwendung wurde die klassische Bildverarbeitung gewählt. Ein wesentlicher Grund dafür ist der erhebliche Aufwand, der mit der Datenerhebung und dem Training eines Deep-Learning-Modells verbunden wäre. Da in diesem Fall nur einfache, klar erkennbare Muster auf einem Labyrinthboden detektiert werden müssen, ist der Einsatz komplexer neuronaler Netze nicht notwendig. Die klassische Bildverarbeitung bietet hier eine gut kontrollierbare, interpretierbare und ressourcenschonende Lösung, die den Anforderungen vollkommen genügt.

5.3.3 Agentensystem

Das Agentensystem übernimmt innerhalb der in Abbildung 5.1 dargestellten Systemarchitektur die zentrale Aufgabe der situationsabhängigen Entscheidungsfindung. Dabei trägt es wesentlich zur Erfüllung der funktionalen Anforderungen PD-FA3 (Erfassung der Kugelposition), PD-FA4 (Erfassung des Spielfeldrahmens) und PD-FA5 (Erfassung von Merkmalen verschiedener Muster) bei.

Ein wesentlicher Bestandteil des Konzepts ist die Wiederverwendung der in der Vorgängerarbeit entwickelten und trainierten DRL-Agenten (vgl. Abschnitt 3). Diese Modelle stellen bereits funktionstüchtige Steuerstrategien für die unterschiedlichen Labyrinthmuster bereit und bilden damit eine gute Ausgangsbasis, die den Implementierungs- und Trainingsaufwand erheblich reduziert.

Integrationsstrategie ins reale System Für die Übernahme der bestehenden Agenten in das reale System ist eine Anpassung der Daten- und Kommunikationsschnittstellen erforderlich. Anstelle der in der Simulation generierten Zustandsinformationen werden nun Echtzeitdaten der RGB-Kamera verwendet, die durch die Bildverarbeitung in das vom Agenten erwartete Zustandsformat überführt werden.

Dabei ist sicherzustellen, dass alle Zustandsinformationen, die dem Agenten in der virtuellen Umgebung zur Verfügung standen, auch im realen System in identischer Form bereitgestellt werden. Dies gewährleistet eine konsistente Schnittstelle zwischen Simulation und physischem Demonstrator und ermöglicht einen nahtlosen Transfer der trainierten Strategien. Darüber hinaus soll die Bildverarbeitung so konzipiert werden, dass für zukünftig potenziell auf dem physischen Labyrinth trainierte Agenten weitere relevante Zustandsinformationen extrahiert und bereitgestellt werden können, um erweiterte Lernstrategien zu unterstützen.

Um eine effektive Verarbeitung mit geringen Latenzen zu gewährleisten, sollen Bildaufnahme und Bildverarbeitung in separaten Threads oder Prozessen ausgeführt werden. Dies ermöglicht eine asynchrone Verarbeitung und verhindert, dass rechenintensive Operationen die Bilderfassung blockieren.

6 Entwicklung

Ein zentraler und umfangreicher Bestandteil bei der Realisierung des autonomen, physikalischen Kugellabyrinth-Systems ist die Entwicklung des Bildverarbeitungssystems. Dieses bildet die Grundlage für die präzise Erfassung des Systemzustands und somit für eine effektive Steuerung durch den RL-Agenten.

In diesem Kapitel wird zunächst die Auswahl einer geeigneten Kamera vorgestellt, einschließlich deren Integration in das physische System sowie der Implementierung der notwendigen Softwarearchitektur zur Einbindung und Steuerung der Kamera in Python. Anschließend wird die durchgeführte Kamerakalibrierung zur Korrektur von Verzerrungen beschrieben. Darauf folgt eine Analyse der relevanten Bildinformationen, die für die Steuerung des Kugellabyrinths erforderlich sind. Für jede dieser Informationen wird jeweils die gewählte Erfassungsmethode beschrieben sowie deren Genauigkeit und Robustheit anhand geeigneter Tests evaluiert.

6.1 Kamerawahl

Basierend auf den in Kapitel 5 definierten funktionalen Anforderungen wurde eine Kameralösung ausgewählt, die sowohl die technische Eignung als auch die systemseitige Kompatibilität sicherstellt. Die Kriterien umfassen unter anderem eine ausreichende Bildauflösung zur Objektunterscheidung, eine hohe Bildrate für eine kontinuierliche Zustandserfassung sowie ein vollständig abdeckendes Sichtfeld. Darüber hinaus war eine stabile und leistungsfähige Datenanbindung an das Verarbeitungssystem erforderlich, die durch eine USB-Schnittstelle gewährleistet wird. Auch die statische Positionierung in Top-Down-Perspektive wurde bei der Auswahl berücksichtigt.

Die eingesetzte Kamera ist die Alvium 1800 U-500c des Herstellers Allied Vision. Sie wurde im Rahmen der Projektarbeit durch das Unternehmen bereitgestellt. Ihre technischen Spezifikationen entsprechen vollständig den gestellten Anforderungen. Sie bietet

eine Bildauflösung von 5 Megapixel (2592 × 1944) und eine maximale Bildrate von 68 Bildern pro Sekunde. Die Kamera ist mit einem CMOS-Sensor ausgestattet und unterstützt die Bildausgabe im RGB8-Farbraum[1]. Aufgrund ihrer modularen Bauweise und Unterstützung industrieller Standards ist sie für den Einsatz in Bildverarbeitungsaufgaben unter Echtzeitbedingungen geeignet.

Das Gehäuse der Kamera ist kompakt und vollständig gekapselt (siehe Abbildung 6.1(a)). Auf der Gehäuseoberseite befinden sich Montagemöglichkeiten zur Befestigung an statischen Halterungen. Das Objektiv wird über ein CS-Mount-Gewinde eingesetzt. Der Anschluss für die Stromversorgung und die GPIOs erfolgt über einen USB Micro-B.

Für die Ansteuerung in Python kommt das Software Development Kit (SDK) von Allied Vision zum Einsatz, das Zugriff auf sämtliche relevanten Kameraeinstellungen bietet.

Das verwendete Objektiv ist das Kowa LM5NCL (Abbildung 6.1(b)), ein C-Mount Weitwinkelobjektiv, das speziell für industrielle Bildverarbeitungsanwendungen konzipiert ist. Es verfügt über eine feste Brennweite von 4,5mm und ist für Sensorformate bis zu 1/1,8 ausgelegt[2] . Das kompakte Metallgehäuse mit Raster-Verschraubungen ermöglicht eine stabile Fixierung. Sowohl der Fokus als auch die Blende sind manuell einstellbar, was eine flexible optische Anpassung an die Anwendung erlaubt. Das Objektiv wurde, wie auch die Kamera, im Rahmen der Projektarbeit durch das jeweilige Unternehmen zur Verfügung gestellt.

Im Folgenden wird zunächst der physische Aufbau sowie die Montage der Kamerahardware beschrieben. Anschließend erfolgt die Darstellung der softwareseitigen Anbindung und Steuerung der Kamera mittels Python, einschließlich der verwendeten Programmbibliotheken und Schnittstellen.

6.1.1 Aufbau und Montage der Kamerahardware

Die Kamera wird in einer festen Top-Down-Konfiguration mit einem Neigungswinkel von 90° senkrecht über dem Spielfeld montiert. Diese Anordnung minimiert perspektivische

[1] https://www.alliedvision.com/de/produktportfolio/kameraselektor/alvium-konfigurator/alvium-1800-u/500/

[2] https://www.kowa-lenses.com/de/LM5NCL-1-1.8-4.5mm-C-Mount-Weitwinkel-Objektiv/10097?srsltid=AfmBOorn9vppFdqSs42NOl3-ncNh7t8F_GIwlY6eIMReMF2YTvAJrJXe

(a) Allied Vision Alvium 1800 U-500c (b) Kowa LM5NC

Abbildung 6.1: Verwendete Hardware: (a) Kamera, (b) Objektiv

Verzerrungen und ermöglicht eine vollständige und gleichmäßige Erfassung des gesamten sichtbaren Bereichs. Die Kamera liegt dabei parallel zur Ebene des Labyrinths, was geometrische Abbildungsfehler zusätzlich reduziert.

Wie bereits in Kapitel 3 beschrieben, erfolgt die Montage über eine stabile Trägerstruktur aus Aluminiumprofilen. Diese wurde beibehalten und dient zur Befestigung der Industriekamera. Die Kamera befindet sich mittig über dem Spielfeld in einer Höhe von *[40]* cm. Dieser Aufbau hat sich bereits in der vorherigen Arbeit bewährt und bietet eine zuverlässige Grundlage für eine stabile und reproduzierbare Kameraposition.

Da die Konstruktion nicht fest mit dem Kugellabyrinth verbunden ist, sondern lediglich in Position geschoben wird, entstehen im Betrieb keine relevanten Vibrationen. Beobachtungen während der Tests zeigten keine negativen Einflüsse auf die Bildstabilität, sodass auf zusätzliche Dämpfungsmaßnahmen verzichtet werden konnte.

In Abbildung 6.2 ist der gesamte Aufbau des physischen Bildverarbeitungssystems dargestellt. Die Kamera ist zentral und orthogonal über dem Spielfeld positioniert und durch die Halterung in der Höhe auch optional justierbar.

Abbildung 6.2: Physischer Aufbau der Kameraintegration oberhalb des Kugellabyrinths

6.1.2 Softwareanbindung und Kamerasteuerung in Python

Nach dem erfolgreichen Aufbau und der Integration der Kamera in das physische Gesamtsystem folgt die softwareseitige Anbindung und Steuerung. Für diesen Zweck wurde durch Prof. Hensel ein modular aufgebautes Kameramodul bereitgestellt, das in Python implementiert wurde. Dieses ermöglicht eine einheitliche Schnittstelle zur Ansteuerung verschiedener Kameramodelle, unabhängig vom SDK.

Das vorliegende UML-Diagramm (Abbildung 6.3) zeigt eine modulare Softwarearchitektur für die einheitliche Verwaltung verschiedener Kamerasysteme. Die Architektur ermöglicht die Integration unterschiedlicher Kamera-Backends in eine gemeinsame Anwendungsumgebung.

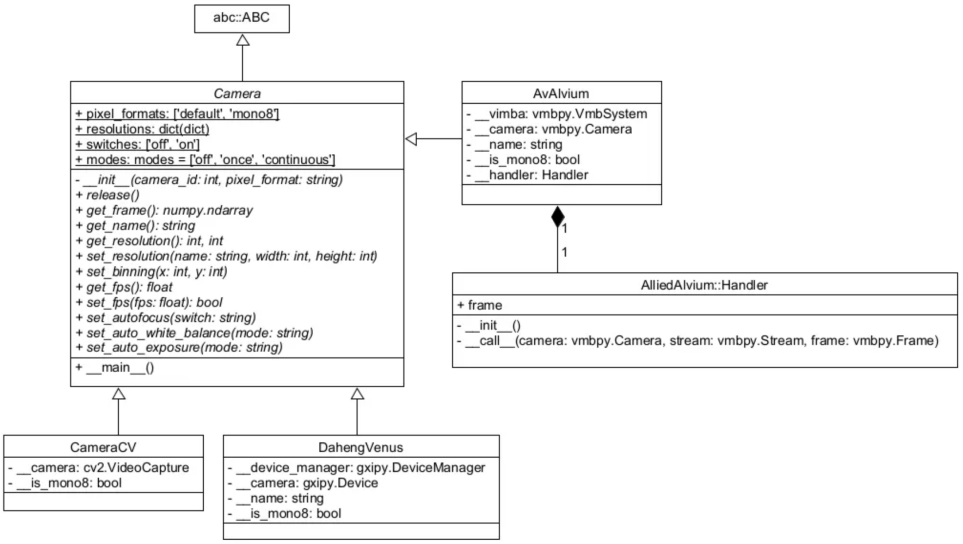


Abbildung 6.3: UML-Diagramm der Kamerasteuerungs-Architektur [10]

Die Hierarchie beginnt mit einer abstrakten Basisklasse abc:ABC, welche das grundlegende Interface-Pattern definiert. Darauf aufbauend stellt die zentrale Camera-Klasse eine einheitliche Programmierschnittstelle für alle Kameraoperationen bereit. Diese umfasst sowohl grundlegende Funktionalitäten wie die Konfiguration von Pixel-Formaten, Auflösungen und Betriebsmodi als auch erweiterte Features zur Frame-Erfassung und Parametersteuerung.

Die konkrete Implementierung erfolgt durch spezialisierte Klassen, die verschiedene Hardware-Backends repräsentieren. Die CameraCV-Klasse nutzt OpenCV als Backend und eignet sich für Standard-Kameraanwendungen mit herkömmlicher Hardware.

Der AlliedAlvium::Handler implementiert dabei spezifische Frame-Verarbeitungslogik für die Kamera.

6.1.3 Kamerakalibrierung

Durch die gewählte Objektivkonfiguration und die feste Top-Down-Positionierung kommt es zu typischen Verzerrungseffekten (Kissenverzerrung) im aufgenommenen Bild, insbesondere an den Rändern. Diese Verzerrungen sind optischer Natur und resultieren aus den physikalischen Eigenschaften der eingesetzten Optik.

Um eine korrekte geometrische Interpretation des aufgenommenen Bildes zu ermöglichen, insbesondere im Hinblick auf eine präzise Lokalisierung von Objekten im Bild, ist eine Kalibrierung der Kamera erforderlich. Dabei wird das Kameraabbild so korrigiert, dass es einer idealen, verzerrungsfreien Abbildung möglichst nahekommt.

Abbildung 6.5(a) zeigt das aktuelle Bild der Kamera mit deutlichen Verzerrungseffekten (Kissenverzerrung) an den Bildrändern.

Zur Korrektur der im Rohbild auftretenden Objektivverzerrungen wurde eine geometrische Kamerakalibrierung durchgeführt. Zum Einsatz kam dabei das etablierte Verfahren der *Checkerboard Calibration*, das insbesondere im Zusammenspiel mit der OpenCV-Bibliothek eine robuste und effiziente Methode zur Kameramodellierung darstellt. Das für die Kalibrierung verwendete Schachbrettmuster sowie der zugehörige Kalibrierungscode wurden von Prof. Hensel zur Verfügung gestellt[3].

Das Grundprinzip der Checkerboard-Kalibrierung besteht darin, die Projektion eines bekannten, regelmäßigen Musters aus verschiedenen Perspektiven aufzunehmen. Anhand der bekannten 3D-Positionen der Ecken im Muster und ihrer 2D-Projektion im Kamerabild können anschließend die intrinsischen Parameter der Kamera sowie die extrinsische Position relativ zum Muster/Ecken ermittelt werden.

Im Rahmen dieser Arbeit wurde ein *ChArUco*-Muster (aus *Chessboard* und *ArUco* zusammengesetzt). Dieses Kalibriermuster kombiniert die regelmäßige Struktur eines Schachbretts mit den eindeutig identifizierbaren binären Markern des *ArUco*-Systems (Artificial Recognizable Markers Using OpenCV).

Die Kalibrierung des Kamerasystems wurde mit der Klasse *CameraCalib* durchgeführt. Das Programm ermöglicht eine interaktive Kalibrierung durch manuelles Hinzufügen von Kamerabildern, in denen ein ChArUco-Muster im Bild zu erkennen ist. Dieses Kalibriermuster wird dabei in mehreren Aufnahmen in unterschiedlichen Positionen und Orientierungen auf der Höhe des Spielfelds platziert. Durch diese Variation wird sichergestellt, dass die Kamera einen möglichst vollständigen Eindruck von der perspektivischen Projektion des Musters erhält. Der genaue Ablauf des Kalibrierungsprozesses ist im Aktivitätsdiagramm dargestellt (siehe Abbildung 6.4).

Die Kalibrierung wurde mit der OpenCV-Funktion `cv2.aruco.calibrateCameraCharuco()` durchgeführt. Diese Funktion analysiert die zuvor erkannten Informationen in den Bil-

[3] https://github.com/MarcOnTheMoon/imaging_learners/tree/main/src/python/cameras

dern und berechnet daraus die Kameramatrix K sowie die Verzerrungskoeffizienten in Form eines Vektors D.

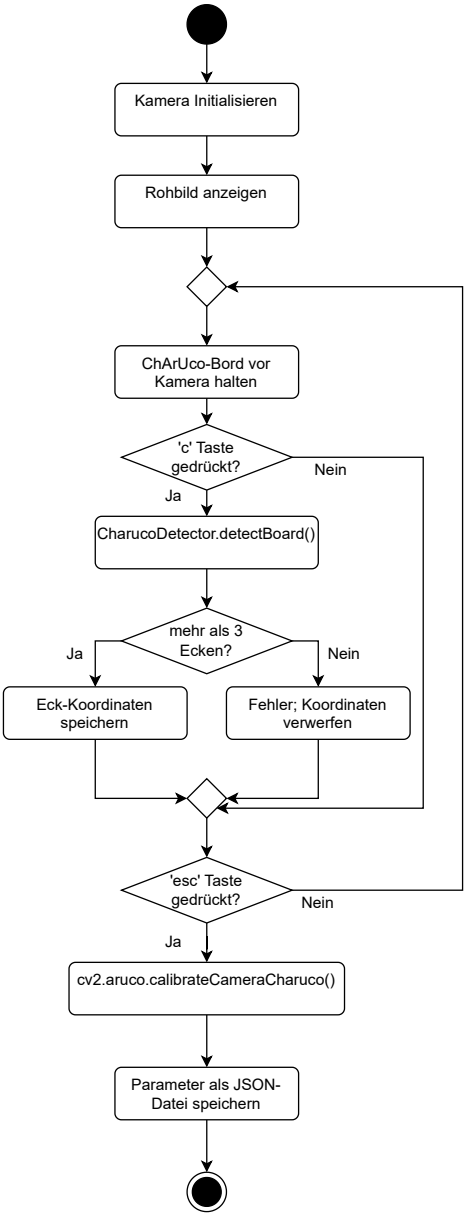
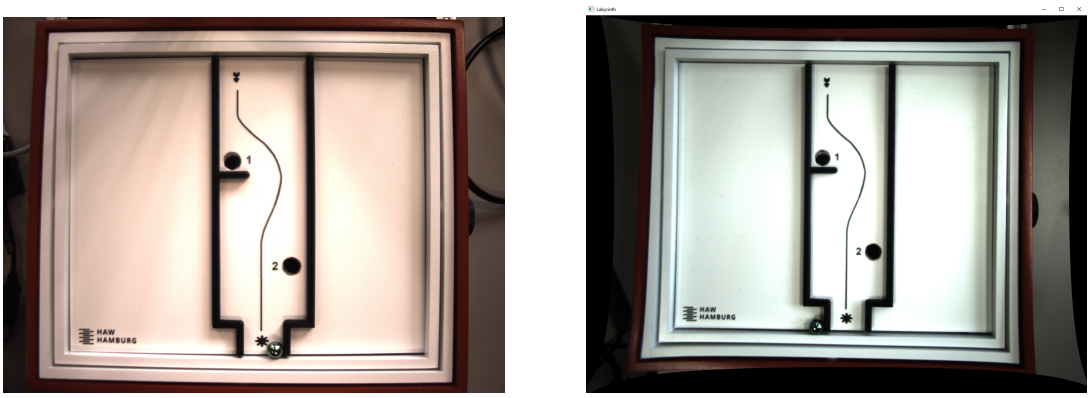


Abbildung 6.4: Aktivitätsdiagramm für CameraCalib



(a) Unkorrigiertes Kamerabild (b) Kalibriertes Kamerabild

Abbildung 6.5: Vergleich der Kameraaufnahmen: (a) Unkorrigiertes Bild, (b) kalibriertes Bild

Die Parameter werden anschließend im JSON-Format gespeichert. Dies ermöglicht eine spätere Wiederverwendung der Kalibrierdaten in der Echtzeit-Bildverarbeitung, ohne dass der Vorgang erneut durchgeführt werden muss.

Durch die Entzerrung mit den gewonnenen Parametern konnte eine signifikante Verbesserung der Bildgeometrie festgestellt werden, wie auf Abbildung 6.5(b) zu sehen. Die zuvor sichtbaren Verzerrungen an den Bildrändern wurden vollständig korrigiert. Dadurch ergibt sich ein weitgehend perspektivisch korrektes Bild des Spielfelds, das als zuverlässige Grundlage für die nachfolgende Bildanalyse dient.

6.2 Relevante Bildinformationen

Die Auswahl und Extraktion visueller Informationen basiert in diesem Projekt gezielt auf Grundlage der funktionalen Anforderungen der Aufgabenstellung. Diese Steuerungsaufgabe erfordert eine präzise Zustandserfassung, die vollständig aus dem Kamerabild abgeleitet werden muss.

Die Gesamtaufgabe wird in mehrere Teilfunktionen zerlegt. Für jede Teilfunktion ist dabei eine spezifische Bildinformation erforderlich, um sie zuverlässig erfüllen zu können. Die dafür relevanten Informationskomponenten sind in Tabelle 6.1 systematisch zusammengefasst.

Tabelle 6.1: Überblick relevanter Bildinformationen zur Zustandserfassung

Informationskomponente	**Beschreibung**	**Typ**
Kugelposition	Lokalisierung der aktuellen Lage der Kugel im Spielfeld	dynamisch
Zielbereich	Detektion der Zielregion, die vom Agenten erreicht werden soll	statisch
Falllöcher	Identifikation gefährlicher Zonen, in die die Kugel nicht gelangen darf	statisch
Spielfeldstruktur (Wände)	Lokalisierung von Barrieren zur Wegplanung und Kollisionsvermeidung	statisch
Spielfeldgrenzen	Abgrenzung des gültigen Bewegungsraums gegenüber unwichtigen Bildbereichen	statisch

Im folgenden Abschnitt werden die relevanten Objekte mithilfe klassischer Bildverarbeitungsmethoden detektiert und extrahiert. Aufbauend auf dieser visuellen Zustandserfassung wird anschließend die vollständige Programmstruktur vorgestellt, welche die entsprechenden Module und deren Zusammenspiel im Gesamtsystem beschreibt.

6.3 Kugelerkennung

Zu Beginn der Entwicklung steht eine der kritischsten Systemkomponenten, die Kugelerkennung. Für eine erfolgreiche Steuerung des Labyrinths ist es essentiell, die exakte Position der Kugel innerhalb des Spielfelds kontinuierlich und präzise zu erfassen. Die Kugelposition stellt die einzige bewegliche Komponente innerhalb des ansonsten strukturell statischen Spielfelds dar. Diese Eigenschaft macht die Kugelerkennung zu einer besonderen Herausforderung, da sie in Echtzeit und unter verschiedenen Bedingungen zuverlässig funktionieren muss.

6.3.1 Methodenanalyse

Grundsätzlich existieren verschiedene Ansätze zur Erfassung beweglicher Objekte innerhalb eines definierten Spielfelds. Die folgende Analyse konzentriert sich auf zwei vielversprechende Verfahren, die Kugelerfassung mittels Differenzenbild und die farbbasierte Erkennung durch HSV-Segmentierung.

Differenzenbild-Verfahren

Das Differenzenbild-Verfahren basiert auf der Subtraktion aufeinanderfolgender Frames, wodurch sich bewegende Objekte als Änderungen im Bild hervorgehoben werden. Diese Methode hat sich bereits in ähnlichen Anwendungen bewährt, wie im Stand der Technik dokumentiert. Der grundlegende Algorithmus berechnet die pixelweise Differenz zwischen dem aktuellen Frame und dem vorherigen Bild, wodurch statische Bildbereiche eliminiert werden und nur bewegliche Elemente sichtbar bleiben.

Das Verfahren lässt sich durch die Verwendung eines kugelfreien Referenzbildes des Labyrinths weiter verfeinern. Diese Erweiterung ermöglicht die Erkennung der Kugel auch in Ruhephasen, da ausschließlich die Abweichungen vom ursprünglichen Zustand berücksichtigt werden. Dadurch werden sowohl positive als auch negative Bildveränderungen erfasst, was eine kontinuierliche Objektverfolgung gewährleistet.

HSV-Farbsegmentierung

Die HSV-basierte Farbsegmentierung nutzt die charakteristische grüne Farbe der Kugel als primäres Erkennungsmerkmal. Im HSV-Farbraum (Hue, Saturation, Value) lassen sich Farben unabhängiger von Helligkeitsschwankungen definieren, was eine robustere Erkennung unter verschiedenen Lichtverhältnissen ermöglicht.

Besonders vorteilhaft erweist sich bei diesem Ansatz die Möglichkeit, den Farbraum gezielt auf die grüne Kugel abzustimmen. Da das Spielfeld keine anderen grünen Objekte enthält, kann der Erkennungsbereich großzügig dimensioniert werden, um Helligkeitsvariationen und Reflexionen zu kompensieren. Dies führt zu einer erhöhten Detektionsrobustheit und reduziert das Risiko von Falscherkennungen.

Methodenvergleich und -auswahl

Beide Verfahren wurden implementiert und unter verschiedenen Testbedingungen evaluiert. Während das Differenzenbild-Verfahren grundsätzlich funktionsfähig ist, zeigt die HSV-Methode deutliche Vorteile in der praktischen Anwendung. Die farbbasierte Erkennung erweist sich als weniger anfällig für Störungen durch Schatten, Beleuchtungsänderungen und minimale Kamerabewegungen. Zusätzlich ermöglicht sie eine stabilere Objektverfolgung auch bei geringen Geschwindigkeiten oder temporären Stillständen

der Kugel. Aufgrund dieser überlegenen Performance und Zuverlässigkeit wird die HSV-Farbsegmentierung als primäres Erkennungsverfahren gewählt.

6.3.2 Ablauf

Die Kugelerkennung erfolgt in mehreren aufeinanderfolgenden Verarbeitungsschritten. Der gesamte Ablauf ist deterministisch aufgebaut und zielt darauf ab, die grüne Kugel anhand ihrer charakteristischen Farbeigenschaften und geometrischen Merkmale zuverlässig zu identifizieren. Die Verarbeitung lässt sich in folgende Hauptphasen gliedern:

1. **Farbsegmentierung im HSV-Farbraum:**
   Zunächst wird das aktuelle Kamerabild vom BGR- in den HSV-Farbraum konvertiert. Dies ermöglicht eine robustere Farbsegmentierung, da der HSV-Farbraum weniger empfindlich gegenüber Helligkeitsschwankungen ist. Durch Anwendung eines definierten Schwellwertbereichs (HSV-Lower: [40, 40, 40], HSV-Upper: [90, 255, 255]) wird eine binäre Maske erzeugt, die ausschließlich Pixel im Grünbereich hervorhebt. Anschließend erfolgt eine Gauß-Filterung der Maske mit einem 3×3-Kernel, um Bildrauschen zu reduzieren und kleine Störungen zu eliminieren. Danach werden morphologische Operationen angewendet. Eine Erosion (1 iteration) zur Entfernung kleiner Artefakte, gefolgt von einer Dilatation (2 Iterationen) zur Wiederherstellung der Objektgröße und Schließung kleiner Lücken.

2. **Konturenerkennung und geometrische Validierung:**
   Im binären Maskenbild werden alle äußeren Konturen mittels der OpenCV-Funktion `cv2.findContours` mit dem Modus `RETR_EXTERNAL` extrahiert. Für jede gefundene Kontur werden folgende geometrische Merkmale berechnet:

   - Fläche (`cv2.contourArea`)
   - Umfang (`cv2.arcLength`)
   - Zirkularität

   Nur Konturen, die folgende Kriterien erfüllen, werden als potenzielle Kugelkandidaten betrachtet:

   - Fläche zwischen 50 und 10.000 Pixeln
   - Zirkularität größer als 0,4 (weist auf annähernd kreisförmige Struktur hin)

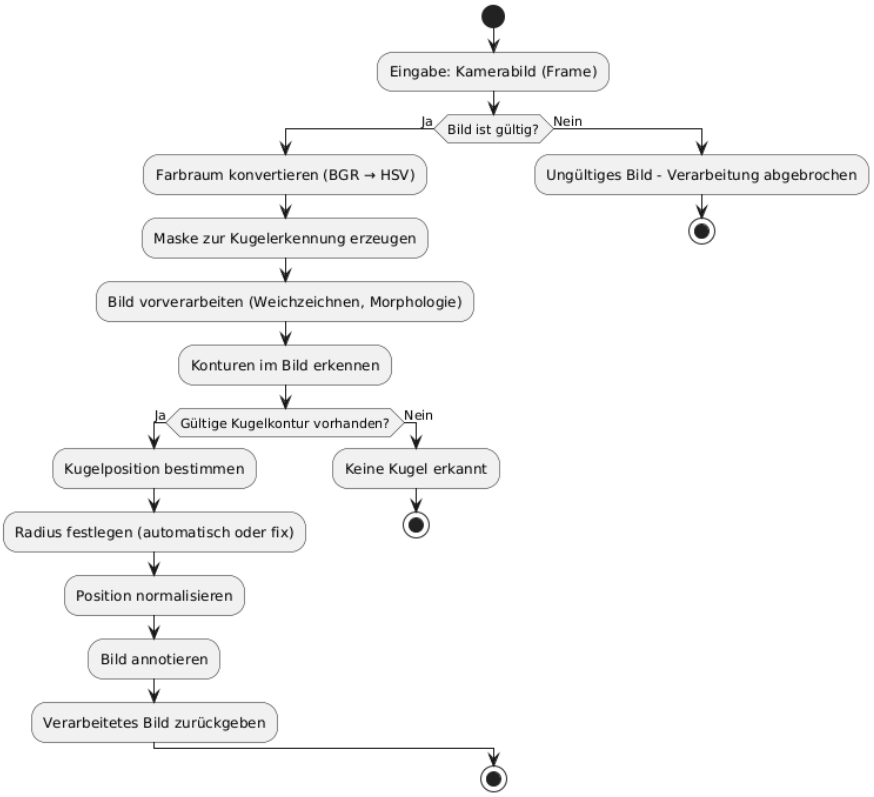


Abbildung 6.6: Aktivitätsdiagramm der Kugelerkennung

3. **Kugelauswahl und Positionsbestimmung:**
   Wurden mehrere gültige Konturen gefunden, wird die Kontur mit der größten Fläche als Kugel identifiziert. Mittels `cv2.minEnclosingCircle` werden Zentrum und Radius des kleinsten umschließenden Kreises bestimmt. Die Position wird sowohl in Pixelkoordinaten als auch normiert (relativ zur Bildgröße) gespeichert. Im Ausgabebild wird die erkannte Kugel durch einen grünen Kreis markiert, dessen Zentrum durch einen ausgefüllten Punkt hervorgehoben wird. Zusätzlich werden die normalisierten Koordinaten als Text eingeblendet.

Abbildung 6.6 visualisiert den beschriebenen Ablauf als Aktivitätsdiagramm. Nach dem Eingang eines gültigen Kamerabildes beginnt die Farbsegmentierung im HSV-Farbraum mit anschließender morphologischer Filterung. Danach werden alle Konturen extrahiert und anhand ihrer geometrischen Eigenschaften (Fläche, Zirkularität) prüft. Die größte gültige Kontur wird als Kugel klassifiziert, ihre Position bestimmt und im Ausgabebild

Listing 6.1: Initialisierung der `BallDetector`-Klasse

```
class BallDetector:
    def __init__(self):
        self.hsv_lower = np.array([40, 40, 40])
        self.hsv_upper = np.array([90, 255, 255])
        self.min_area = 50
        self.max_area = 10000
        self.fixed_radius = None
        self.gaussian_kernel = (3, 3)
        self.erosion_iters = 1
        self.dilation_iters = 2
```

visualisiert. Wird kein gültiges Bild geliefert oder keine passenden Strukturen gefunden, endet der Prozess ohne Ergebnis.

6.3.3 Implementierung

Der Ansatz wird durch eine eigene BallDetector-Klasse realisiert, die alle benötigten Methoden und Parameter in einer klaren Struktur bündelt. Dadurch ist eine Trennung zwischen Konfiguration, Verarbeitung und Ausgabe gewährleistet.

Die Grundstruktur der BallDetector-Klasse legt alle wesentlichen Parameter zur Bildverarbeitung fest, wie in Listing 6.1 dargestellt.

Die HSV-Schwellwerte (`hsv_lower`, `hsv_upper`) definieren den Farbbereich für grüne Objekte. Der gewählte Hue-Bereich von 40–90° deckt verschiedene Grüntöne ab und ist robust gegenüber unterschiedlichen Lichtverhältnissen.

Die zentrale Methode `track_ball()` führt die komplette Verarbeitung eines einzelnen Kamerabildes durch. Das Eingabebild wird zunächst in den HSV-Farbraum konvertiert. Anschließend erstellt die `inRange()`-Funktion eine Binärmaske, die nur Bildbereiche im definierten Grünbereich enthält.

Die Funktion `findContours()` extrahiert alle Umrisse der Maske. Die Fläche und Rundheit (Verhältnis zwischen Umfang und Fläche) jeder Kontur wird geprüft. Eine ideale Kugel hätte eine Zirkularität von 1. Der gewählte Schwellenwert erlaubt Abweichungen, blendet aber unregelmäßige Formen aus.

Die Position der Kugel wird relativ zur Bildgröße normiert, was spätere Auswertungen unabhängig von der Auflösung ermöglicht.

Zum Schutz vor fehlerhaften oder instabilen Ausgaben werden alle Ergebnisse zunächst zurückgesetzt und anschließend auf Gültigkeit geprüft.

Die Position wird nur dann als gültig gewertet, wenn sie innerhalb der Bildgrenzen liegt und der Radius eine Mindestgröße überschreitet. Diese Prüfung verhindert fehlerhafte Erkennungen durch Bildstörungen oder Randeffekte.

6.4 Falllöchererkennung

Die Falllöcher stellen das größte Risiko im Spielverlauf dar, da ein Hineinfallen der Kugel zum sofortigen Abbruch und damit zum Scheitern des aktuellen Spiels führt. Aus diesem Grund ist eine im Optimalfall vollständige und präzise Erkennung aller Falllöcher auf dem Spielfeld von zentraler Bedeutung. Im Gegensatz zur Kugelposition sind die Falllöcher statisch und befinden sich in jedem Labyrinthmuster an festen Positionen. Diese Eigenschaft vereinfacht die Detektion grundsätzlich, erfordert jedoch eine robuste Initialerkennung, um alle relevanten Positionen zuverlässig zu identifizieren.

6.4.1 Methodenanalyse

Zwei vielversprechende Ansätze sind die klassische Formanalyse auf Basis von Schwellenwertsegmentierung und Konturmerkmalen sowie die Hough-Kreis-Transformation in Kombination mit Kantendetektion. Im Folgenden werden beide Verfahren vorgestellt und hinsichtlich ihrer Eignung für das vorliegende System bewertet.

Schwellwertsegmentierung mit konturbasierter Formanalyse

Der gewählte Ansatz basiert auf einer Grauwert-Segmentierung mittels Schwellwertverfahren, gefolgt von einer geometrischen Analyse der erkannten Konturen. Zunächst wird das Kamerabild mit einem festen Schwellenwert binarisiert, um die dunkel erscheinenden Falllöcher hervorzuheben.

Im binären Bild werden dann alle äußeren Konturen extrahiert und anhand ihrer Fläche, ihres Radius und insbesondere ihrer Rundheit analysiert. Konturen, die diese Kriterien erfüllen, sollen als potenzielle Falllöcher klassifiziert und entsprechend markiert werden.

Dieser Ansatz nutzt rein lokale Bildinformationen und erfordert keine komplexe Modellierung oder Vorverarbeitung. Durch die gezielte Einschränkung auf bestimmte Größen- und Formbereiche kann die Erkennung sehr effizient und robust erfolgen, solange der Kontrast zwischen Löchern und Spielfeld ausreichend ist.

Kantendetektion mit Hough-Kreis-Transformation

Ein alternativer Ansatz zur Erkennung kreisförmiger Strukturen ist die Verwendung der Hough-Transformation in Kombination mit einem vorgelagerten Kantenfilter (z.B. Canny). Dabei werden zunächst alle signifikanten Kanten im Bild identifiziert, bevor in einem Parameterraum gezielt nach Kreisen mit definierbarem Radius gesucht wird.

Die Hough-Kreis-Transformation eignet sich besonders zur Detektion idealer, geschlossener Kreisformen, da sie unabhängig von Helligkeit oder Textur arbeitet.

Methodenvergleich und -auswahl

Die konturbasierte Formanalyse wurde gewählt, da sie insbesondere bei konstanten Lichtverhältnissen eine zuverlässige und reproduzierbare Detektion ermöglicht. Aufgrund ihrer geringen Komplexität, der hohen Verarbeitungsgeschwindigkeit und der guten Erkennungsgenauigkeit erwies sich dieses Verfahren als besonders geeignet für die Falllocherkennung im vorliegenden System.

6.4.2 Ablauf

Die Löchererkennung erfolgt in mehreren aufeinanderfolgenden Verarbeitungsschritten ähnlich zu den anderen Detektoren. Der gesamte Ablauf ist deterministisch aufgebaut und zielt darauf ab, kreisförmige, dunkle Strukturen mit bestimmten geometrischen Merkmalen zuverlässig zu identifizieren. Die Verarbeitung lässt sich in folgende Hauptphasen gliedern:

1. **Bildeingang und Vorverarbeitung:**
   Zunächst wird das aktuelle Kamerabild entgegengenommen und in ein Graustufenbild mittels `cv2.cvtColor` umgewandelt. Anschließend erfolgt eine globale Schwellwertbinarisierung mit invertierter Logik (`cv2.threshold` mit `THRESH_BINARY_INV`) bei einem Schwellwert von 20. Diese Invertierung bewirkt, dass dunkle Objekte wie Falllöcher als weiße Bereiche hervorgehoben werden, während hellere Bereiche schwarz erscheinen.

2. **Konturenerkennung und geometrische Validierung:**
   Im gefilterten Binärbild werden alle äußeren Konturen mittels `cv2.findContours` mit dem Modus `RETR_EXTERNAL` extrahiert. Für jede gefundene Kontur werden folgende geometrische Merkmale berechnet:

   - Fläche (`cv2.contourArea`)
   - Umfang (`cv2.arcLength`)
   - Rundheit
   - Minimaler umschließender Kreis (`cv2.minEnclosingCircle`)

   Nur Konturen, die folgende vordefinierte Kriterien erfüllen, werden als potenzielle Falllöcher betrachtet:

   - Fläche größer als 500 Pixel
   - Rundheit zwischen 0,8 und 1,05
   - Radius zwischen 10 und 30 Pixeln

Abbildung 6.7 visualisiert den beschriebenen Ablauf als Aktivitätsdiagramm. Nach dem Eingang eines gültigen Kamerabildes beginnt die Vorverarbeitung mit Grauwertumwandlung und globaler invertierter Schwellwertbildung. Das resultierende Binärbild wird mittels Gauß-Filter geglättet, bevor alle Konturen extrahiert werden. Diese werden anhand ihrer geometrischen Eigenschaften (Fläche, Rundheit, Radius) validiert. Validierte Strukturen werden als Falllöcher klassifiziert, mit dem Persistenzmechanismus abgeglichen, visualisiert und in den Datenspeicher übernommen. Wird kein gültiges Bild geliefert oder keine passenden Strukturen gefunden, endet der Prozess ohne Ergebnis.

In Abbildung 6.8 ist die schrittweise Herangehensweise der Locherkennung dargestellt. Zunächst wird das Eingangsbild durch Schwellenwertbildung binarisiert (a), um die dunklen

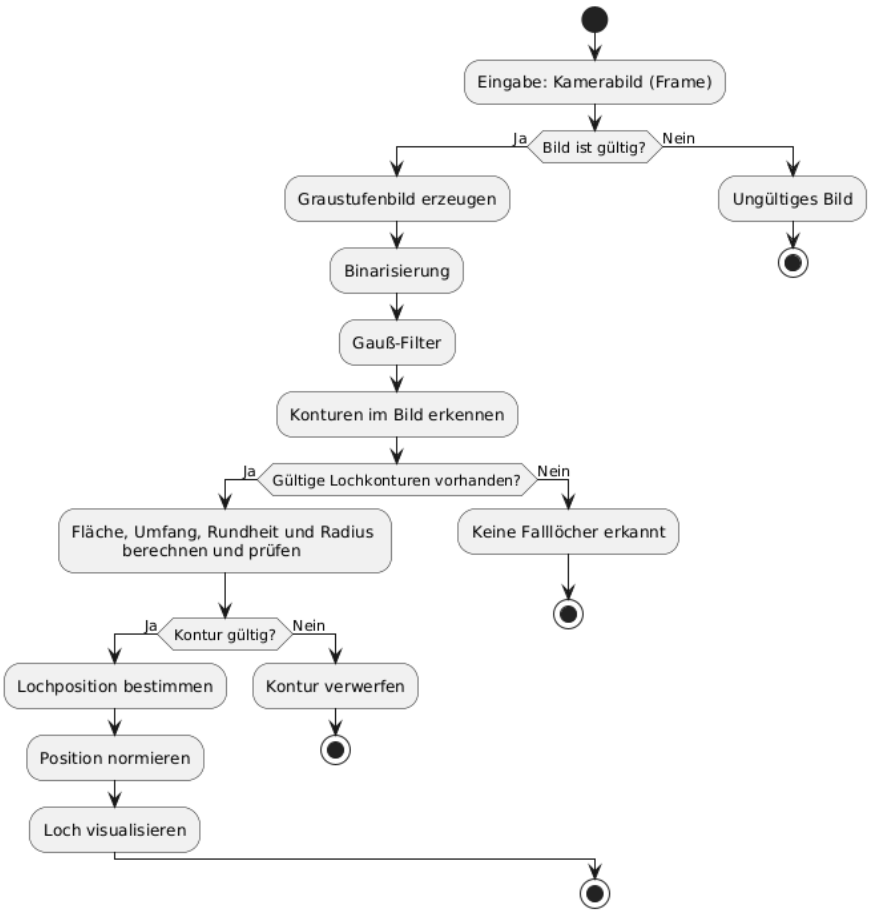


Abbildung 6.7: Aktivitätsdiagramm der Falllöchererkennung

Lochobjekte vom hellen Hintergrund zu trennen. Anschließend erfolgt die Segmentierung (b), bei der die relevanten Konturen identifiziert und isoliert werden. Im letzten Schritt werden die geometrischen Merkmale wie Rundheit und Radius extrahiert und die erkannten Löcher entsprechend klassifiziert (c).

6.4.3 Implementierung

Die Detektion der Falllöcher erfolgt in der Löcher Detector-Klasse. Im Folgenden werden die zentralen Verarbeitungsschritte anhand ausgewählter Codeabschnitte erläutert.

Die Vorverarbeitung umfasst die Konvertierung des Kamerabilds in ein Graustufenformat, eine binäre Schwellenwertbildung, wie in Abbildung 6.7 beschrieben.

(a) Binarisierung (b) Segmentierung (c) Merkmalsextraktion und Klassifizierung

Abbildung 6.8: Schrittweise Verarbeitung der Locherkennung: (a) Binarisierung, (b) Segmentierung der Lochkonturen, (c) Extraktion und Klassifizierung der Merkmale

Die zentrale Logik der Objekterkennung basiert auf einer Analyse der im binären Bild erkannten Konturen. Für jede Kontur werden die Fläche, der Umfang sowie der minimale umschließende Kreis berechnet. Als Schlüsselkriterium für die Klassifikation dient die Rundheit. Sie erlaubt eine Unterscheidung zwischen tatsächlich kreisförmigen Strukturen und unregelmäßigen Bildartefakten.

Nur Konturen, die innerhalb eines definierten Bereichs für Fläche, Radius und Rundheit liegen, werden als Falllöcher klassifiziert. Diese Schwellenwerte sind als Parameter in der Klasse definierbar und ermöglichen eine flexible Anpassung an verschiedene Labyrinthmuster und Maßstäbe. Abschließend werden die erkannten Falllöcher im Originalbild visualisiert.

6.5 Spielfelderkennung

Die Erkennung des Spielfeldrahmens ist eine Voraussetzung für die korrekte Verarbeitung der Bilddaten. Besonders für den Agenten, der in einer virtuellen Umgebung mit festen Spielfeldgrenzen trainiert wurde, ist es entscheidend, dass der erkannte Bildausschnitt exakt diesen Bedingungen entspricht. Da es sich beim Spielfeld um eine statische Information handelt, kann die Erkennung einmalig zu Beginn durchgeführt und für alle weiteren Schritte verwendet werden.

6.5.1 Methodenanalyse

Für die Erkennung des Spielfeldrahmens bieten sich zwei Ansätze an. Eine geometriebasierte Detektion auf Grundlage von Kanten- und Konturanalyse sowie eine farbbasierte Segmentierung im HSV-Farbraum, wie sie auch bei der Kugelerkennung verwendet wird. Beide Verfahren sind prinzipiell geeignet, um rechteckige Bildbereiche zu identifizieren.

Die konturbasierte Methode nutzt strukturelle Merkmale wie Kantenverläufe und geschlossene Konturen zur Bestimmung des Spielfelds. Sie basiert auf gängigen Verfahren der Kantenfilterung und geometrischen Approximation. Alternativ wäre auch eine Segmentierung des rot gefärbten Spielfeldrahmens im HSV-Farbraum möglich gewesen. Da der Rahmen farblich klar vom Hintergrund abgesetzt ist, hätte dieser Ansatz ebenfalls zur Lokalisierung verwendet werden können.

Die Entscheidung fiel zugunsten der konturbasierten Methode, da sie sich in der Praxis als zuverlässig und robust erwies. Erste Tests zeigten eine konsistente und exakte Rahmenerkennung, sodass auf die zusätzliche Implementierung der farbbasierten Variante verzichtet wurde.

6.5.2 Ablauf

Die Erkennung des Spielfelds erfolgt in mehreren aufeinanderfolgenden Verarbeitungsschritten. Der zugrunde liegende Algorithmus ist darauf ausgelegt, das rechteckige Spielfeld automatisch zu identifizieren und über mehrere Frames hinweg stabil zu verfolgen.

Liegt ein Bild vor, erfolgt zunächst die Überprüfung, ob das Spielfeld bereits in einem vorherigen Schritt erfolgreich erkannt wurde. Ist dies der Fall, wird das Bild direkt anhand des bekannten Rechtecks zugeschnitten. Dieser Schritt reduziert die Rechenlast erheblich, da eine erneute Analyse entfällt.

Falls noch keine stabile Rahmenerkennung vorliegt, beginnt die eigentliche Bildanalyse. Das Bild wird in Graustufen umgewandelt und anschließend erfolgt eine Kantendetektion mit dem Canny-Algorithmus.

Im nächsten Schritt werden die Konturen im Bild identifiziert. Nur solche Konturen, die bestimmte geometrische Eigenschaften erfüllen, werden als potenzielle Spielfeldkandidaten berücksichtigt. Dazu zählen eine eingeschlossene Fläche sowie die Approximation der Kontur auf ein Viereck.

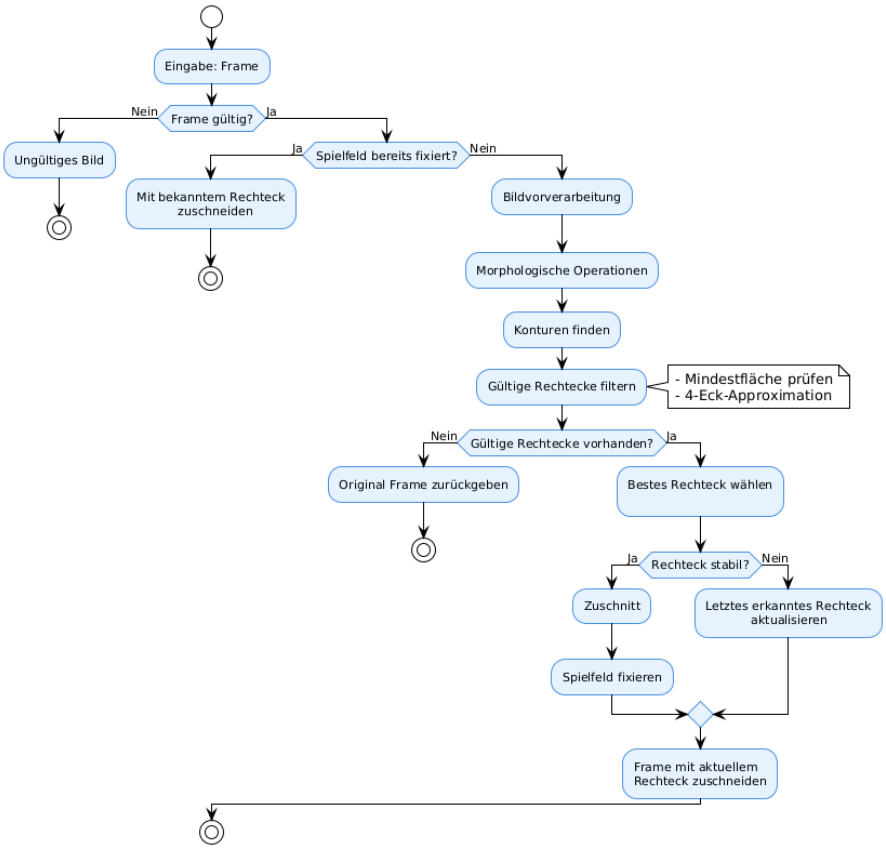


Abbildung 6.9: Aktivitätsdiagramm der Spielfelderkennung

Wird kein geeignetes Rechteck gefunden, wird das Originalbild ohne Zuschnitt zurückgegeben. Liegen valide Rechtecke vor, erfolgt die Auswahl des geeignetsten Kandidaten anhand der Nähe zum Bildzentrum, da das Spielfeld zentral im Kamerabild angeordnet ist.

Abschließend wird das Bild anhand des erkannten Rechtecks beschnitten. In den darauffolgenden Frames kann der erkannte Bildausschnitt direkt verwendet werden, ohne den Erkennungsprozess erneut durchzuführen. Abbildung 6.9 veranschaulicht den beschriebenen Ablauf als Aktivitätsdiagramm. Der Prozess zeigt die verschiedenen Verzweigungen und Erkennungsstatus.

Listing 6.2: Stabilitätsprüfung eines erkannten Rechtecks

```
def _is_stable(self, current_rect):
    if self.last_detected_rect is None:
        return False
    x1, y1, w1, h1 = self.last_detected_rect
    x2, y2, w2, h2 = current_rect
    return (abs(x1 - x2) + abs(y1 - y2)
    + abs(w1 - w2) + abs(h1 - h2)) <= self.rect_tolerance
```

6.5.3 Implementierung

Der Ansatz wird durch eine eigene FieldDetector-Klasse realisiert, die alle benötigten Methoden und Parameter zur Spielfelderkennung in strukturierter Form zusammenfasst.

Die Canny-Parameter bestimmen die Empfindlichkeit der Kantenerkennung. Besonders kritisch sind die Stabilitätsparameter: `required_stable_frames` legt fest, wie viele aufeinanderfolgende konsistente Erkennungen notwendig sind, bevor das Spielfeld als fixiert gilt.

Die zentrale Methode `detect_field()` implementiert die vollständige Verarbeitungskette. Nach einer initialen Validierung des Eingabeframes wird geprüft, ob bereits ein stabiler Spielfeldrahmen erkannt wurde. Ist dies der Fall, erfolgt direkt ein Bildzuschnitt auf Basis des gespeicherten Rechtecks, was die Verarbeitung erheblich beschleunigt.

Ein zentraler Bestandteil des Verfahrens ist die Stabilitätsprüfung über mehrere Frames hinweg (vgl. Listing 6.2).

Diese Methode vergleicht das aktuell erkannte Rechteck mit der vorherigen Erkennung und bewertet die Summe aller geometrischen Abweichungen. Nur bei ausreichender Übereinstimmung wird der interne Stabilitätszähler erhöht.

6.6 Wände

Neben der Position der Falllöcher ist auch die Erfassung der Labyrinthstruktur von Bedeutung. Die Wände innerhalb des Spielfelds stellen physische Hindernisse dar, die die Bewegung der Kugel einschränken und die Pfadplanung beeinflussen. Zwar führen Kollisionen mit den Wänden im Gegensatz zu Falllöchern nicht zum unmittelbaren Abbruch

des Spiels, sie behindern jedoch gezielte Bewegungen und können Fehlverhalten des Agenten verursachen, wenn sie im Zustandsmodell nicht korrekt berücksichtigt werden. Daher ist eine Erkennung der inneren Wandstrukturen erforderlich, um dem Agenten vollständige Umgebungsinformationen bereitzustellen.

6.6.1 Methodenauswahl

Für die Detektion der Labyrinthwände wird ein linienbasiertes Verfahren gewählt. Die Struktur der Wände ist durch klar definierte, geradlinige Segmente geprägt, die sich im Bild deutlich vom Hintergrund abheben. Aufgrund dieser Eigenschaften bietet sich die Erkennung über Kantendetektion zur Linienextraktion als effizienteste Lösung an.

6.6.2 Ablauf

Die Erkennung der Labyrinthwände erfolgt durch ein mehrstufiges Verfahren, das auf Kantendetektion und anschließender Linienextraktion mittels Hough-Transformation basiert. Ein Aspekt ist dabei die gezielte Ausblendung anderer Objekte im Spielfeld, um Fehldetektionen in überlagerten Bildbereichen zu vermeiden.

Der Algorithmus beginnt mit der Validierung des Eingabeframes sowie der übergebenen Masken für Kugel und Falllöcher. Nach erfolgreicher Prüfung wird das Bild vorverarbeitet. Anschließend erfolgt eine Schwellenwertbildung, die eine erste binäre Maske erzeugt, in der potenzielle Wandbereiche hervorgehoben sind.

Ein zentraler Verarbeitungsschritt ist die bedingte Anwendung einer Ausschlussmaske. Liegt eine Maske für Falllöcher vor, wird diese mit einer Kugelmaske kombiniert. Die resultierende Ausschlussmaske wird anschließend von der Wandmaske subtrahiert, sodass alle bekannten Objekte im Bild gezielt von der weiteren Verarbeitung ausgeschlossen werden. Ist keine Objektmaske vorhanden, wird die ursprüngliche Wandmaske unverändert verwendet.

Die eigentliche Linienerkennung erfolgt über eine Canny-Kantendetektion, gefolgt von einer Hough-Transformation. Dabei werden nur solche Linien berücksichtigt, die eine definierte Mindestlänge überschreiten und deren Segmentlücken innerhalb eines festgelegten Grenzwerts liegen. Auf diese Weise wird sichergestellt, dass nur signifikante und durchgehende Wandsegmente in das Ergebnis einfließen.

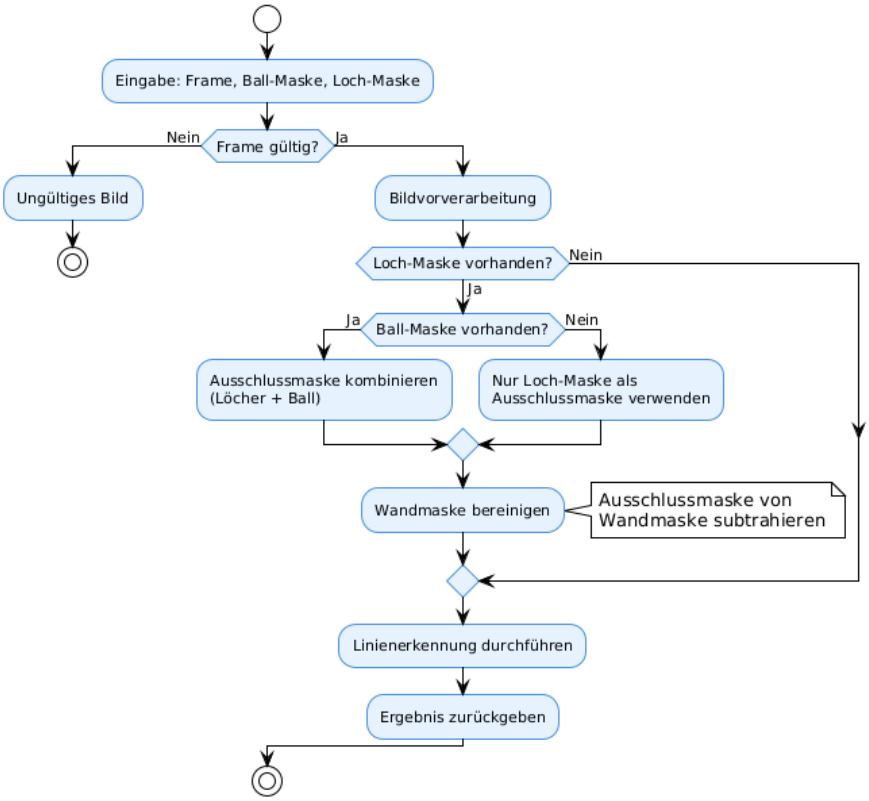


Abbildung 6.10: Aktivitätsdiagramm der Wanderkennung

Abbildung 6.10 zeigt den Ablauf der Wanderkennung als Aktivitätsdiagramm. Der Prozess veranschaulicht die bedingte Verarbeitung in Abhängigkeit von der Verfügbarkeit der Objektmasken sowie den sequentiellen Aufbau der Bildverarbeitungsschritte. Der Algorithmus ist darauf ausgelegt, stabile Wanderkennungen zu liefern und gleichzeitig Fehldetektionen durch andere Spielelemente zu minimieren.

6.6.3 Implementierung

Die zentrale Methode `detect_walls()` verarbeitet das Eingabebild gemeinsam mit den übergebenen Objektmasken. Nach der initialen Validierung erfolgt eine Vorverarbeitung bestehend aus Graustufen-Konvertierung und binärer Schwellwertbildung. Diese Schritte erzeugen eine erste Maske, in der potenzielle Wandbereiche hervorgehoben werden.

Listing 6.3: Kombination und Caching von Ausschlussmasken

```
if self.cached_dilated_hole_mask is not None:
    exclusion_mask = self.cached_dilated_hole_mask
    if ball_mask is not None:
        exclusion_mask = cv2.bitwise_or(ball_mask,
        exclusion_mask)
    wall_mask = cv2.bitwise_and(wall_mask,
    cv2.bitwise_not(exclusion_mask))
```

Ein wesentlicher Aspekt der Implementierung ist die effiziente Verwaltung von Ausschlussmasken, wie in Listing 6.3 gezeigt.

Die erweiterte Falllochmaske wird mit der Kugelmaske kombiniert, um eine gemeinsame Ausschlussmaske zu erzeugen. Diese wird anschließend von der Wandmaske subtrahiert, sodass überlagerte Bildbereiche bei der Erkennung ignoriert werden. Dadurch wird verhindert, dass Kugel oder Löcher versehentlich als Wandsegmente klassifiziert werden.

Die Linienerkennung selbst erfolgt über die Methode `_detect_lines()`, die eine Kombination aus Gauß-Filterung, Canny-Kantendetektion und Hough-Transformation verwendet. Zunächst wird das Maskenbild mit einem 3×3-Gauß-Kernel geglättet. Anschließend erfolgt die Kantendetektion mittels Canny-Filter mit den Schwellwerten 60 (unterer Schwellwert) und 180 (oberer Schwellwert). Diese Werte sind im Vergleich zu typischen Standardwerten bewusst höher angesetzt, um auch schwächere Wandkanten zuverlässig zu erfassen und gleichzeitig Rauschen zu unterdrücken. Abschließend wird die probabilistische Hough-Linientransformation (`cv2.HoughLinesP`) mit einem Akkumulator-Schwellwert von 30 angewendet, wobei nur Liniensegmente mit einer Mindestlänge von 30 Pixeln und maximal 10 Pixeln Abstand zwischen den Segmenten als gültige Wandstrukturen erkannt werden.

6.7 Ziel

Die Erkennung des Zielbereichs ist besonders wichtig, da das Erreichen dieser Position die einzige Bedingung für den erfolgreichen Abschluss des Spiels darstellt. Da es sich beim Ziel um ein statisches Objekt mit fester Position handelt, genügt eine einmalige Erkennung zu Beginn, deren Ergebnis anschließend dauerhaft gespeichert und weiterverwendet werden kann.

6.7.1 Methodenanalyse

Das Ziel sticht aufgrund seiner auffälligen geometrischen Asterisk-Form von den anderen Spielelementen ab. Mit seinen acht Spitzen bildet es ein Muster, das sich mittels klassischer Bildverarbeitungsverfahren gut aus dem Kamerabild extrahieren und klassifizieren lässt. Die komplexe Struktur des Symbols unterscheidet sich fundamental von den einfachen geometrischen Formen wie Kreisen oder geraden Linien, die andere Labyrinth-Komponenten charakterisieren.

Grundsätzlich existieren verschiedene Ansätze zur Erfassung des sternförmigen Ziels innerhalb des Spielfelds. Eine Möglichkeit besteht in der konturbasierten Erkennung, die das Asterisk anhand seiner Kanten identifiziert. Dabei werden zunächst alle Konturen im Bild extrahiert und anschließend nach geometrischen Kriterien gefiltert, um sternförmige Strukturen zu identifizieren. Die Analyse würde sich auf Merkmale wie die Anzahl der Ecken, Winkelverteilungen und das Verhältnis zwischen Innen- und Außenradius konzentrieren. Alternativ bietet sich das Template-Matching-Verfahren an, das ein vordefiniertes Musterbild des Ziels systematisch mit verschiedenen Bildregionen vergleicht und dabei sowohl Position als auch Skalierung berücksichtigt. Diese Methode nutzt Korrelationsalgorithmen, um Ähnlichkeiten zwischen dem Template und Bildbereichen zu finden.

Aufgrund der besonderen Form des Asterisk-Symbols erweist sich das Template-Matching-Verfahren als besonders geeignet für diese Anwendung. Die komplexe, aber konsistente Struktur des achtzackigen Sterns lässt sich optimal durch einen templatebasierten Ansatz erfassen, da das Muster unabhängig von geringfügigen Beleuchtungsänderungen oder Bildqualitätsschwankungen zuverlässig erkannt werden kann und über alle Labyrinthe identisch ist. Im Gegensatz zur konturbasierten Methode, die anfällig für Rauschen und unvollständige Kantenerkennung ist, bietet Template-Matching eine robustere Lösung für die Lokalisierung des statischen Ziels.

6.7.2 Ablauf

Die Erkennung des Ziels erfolgt durch einen Ansatz mit Multi-Scale-Matching und Stabilitätskontrolle. Dieser Prozess ist auf eine einmalige Lokalisierung ausgelegt.

Der Algorithmus beginnt mit einem Eingabeframe sowie der Verfügbarkeit des geladenen Templates. Falls das Ziel bereits in vorherigen Durchläufen erfolgreich erkannt und fixiert

wurde, erfolgt lediglich eine Visualisierung der gespeicherten Position, ohne dass weitere Berechnungen notwendig sind.

Für die eigentliche Zielerkennung wird das Eingabebild zunächst in ein Graustufenbild konvertiert. Anschließend erfolgt das Multi-Scale-Template-Matching, bei dem das gespeicherte Sternmuster mit fünf verschiedenen Skalierungsfaktoren (0,2; 0,25; 0,3; 0,35; 0,4) in allen Bildbereichen verglichen wird. Für jede Skalierung wird mittels `cv2.matchTemplate` mit der Methode `TM_CCOEFF_NORMED` (normalisierte Kreuzkorrelation) die beste Übereinstimmung ermittelt.

Falls eine Übereinstimmung oberhalb des definierten Schwellwerts von 0,4 gefunden wird, wird diese Detektion der internen Historie hinzugefügt. Der Algorithmus sammelt fünf aufeinanderfolgende Erkennungen und prüft deren Positionsstabilität durch Berechnung der Standardabweichung der Zentren in x- und y-Richtung. Nur wenn die Standardabweichungen unter 5 Pixeln liegen, wird das Ziel als definitiv erkannt markiert, die Position dauerhaft gespeichert und der Erkennungsprozess abgeschlossen (Status: `detection_locked`).

Abbildung 6.11 zeigt den Ablauf der Zielerkennung als Aktivitätsdiagramm.

6.7.3 Implementierung

Der Ansatz wird durch eine eigene `TargetDetector`-Klasse realisiert, die alle benötigten Parameter und Methoden für das templatebasierte Multi-Scale-Matching zusammenfasst.

Die zentrale Methode `detect_target()` implementiert die vollständige Erkennungslogik mit mehreren Zustandsebenen. Nach der initialen Überprüfung prüft der Algorithmus den aktuellen Erkennungsstatus (`detection_locked`, `detection_completed`) und entscheidet zwischen verschiedenen Verarbeitungspfaden. Für die eigentliche Zielerkennung wird das Eingabebild in ein Graustufenbild konvertiert und anschließend dem Multi-Scale-Matching unterzogen.

Abbildung 6.12 zeigt das verwendete Asterisk-Ziel, das als Template in der Erkennungslogik dient.

Das Herzstück der Implementierung ist das Multi-Scale-Matching, das systematisch fünf verschiedene Größenvarianten des Templates überprüft. Für jede Skalierung (20%, 25%,

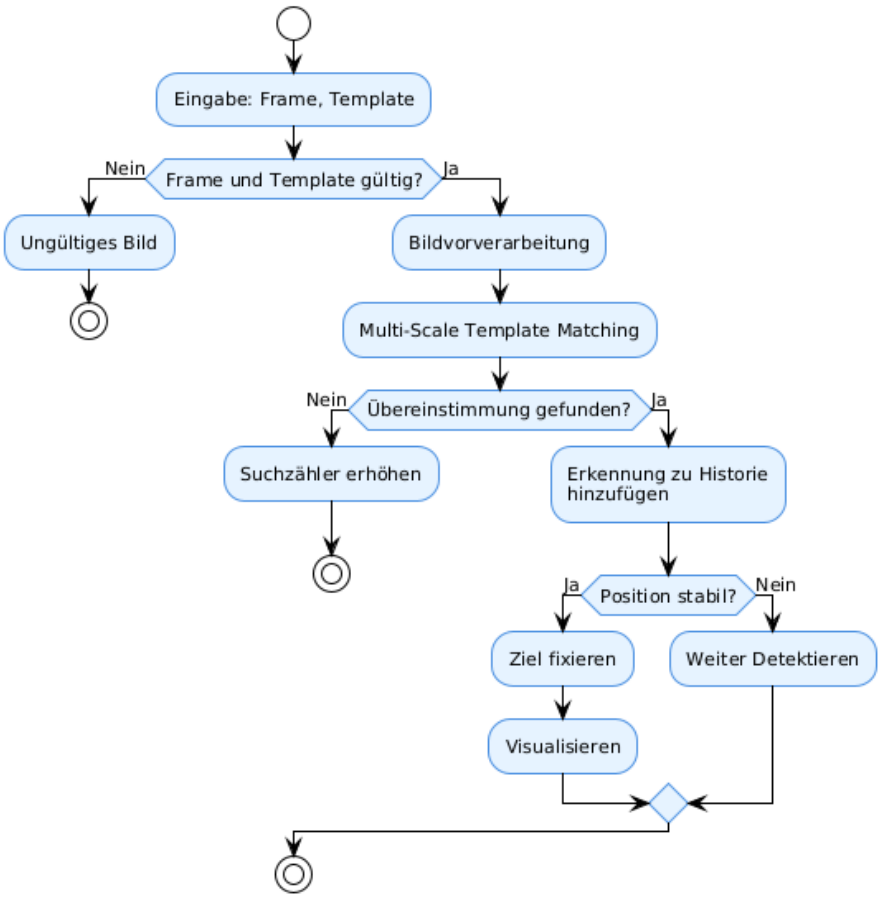


Abbildung 6.11: Aktivitätsdiagramm der Zielerkennung

30%, 35%, 40% der Originalgröße) wird das Template auf die entsprechende Größe skaliert und mittels `cv2.matchTemplate` mit der Methode `TM_CCOEFF_NORMED` über das gesamte Eingabebild verschoben. Die normalisierte Kreuzkorrelation liefert Werte zwischen -1 und 1, wobei höhere Werte eine bessere Übereinstimmung anzeigen. Durch den Vergleich aller Skalierungen wird die global beste Erkennung ermittelt, unabhängig von der tatsächlichen Größe des Ziels im Bild. Die Position mit dem höchsten Korrelationswert über alle Skalierungen hinweg wird als beste Übereinstimmung betrachtet.

Abbildung 6.12: Beispielbild des Asterisk-Ziels, das als Template verwendet wird.

6.8 Programmstruktur und Systemarchitektur

Für die Umsetzung des Bildverarbeitungssystems wurde eine objektorientierte, modular aufgebaute Softwarearchitektur entwickelt. Ziel dieser Strukturierung ist es, die Komplexität des Systems durch eine klare Trennung der funktionalen Verantwortlichkeiten zu reduzieren und gleichzeitig eine hohe Wiederverwendbarkeit und Wartbarkeit zu gewährleisten (entsprechend den Anforderungen PD-NFA11).

Die Gesamtstruktur des Bildverarbeitungssystems wird durch das Klassendiagramm in Abbildung 6.13 veranschaulicht. Dieses bietet einen systematischen Überblick über die beteiligten Komponenten und deren Beziehungen zueinander.

Das Diagramm zeigt die zentrale Rolle des `LabyrinthTracker` als Koordinator des gesamten Systems. Dieser fungiert als Schnittstelle zwischen der Kameraanbindung, der Kalibrierung und den einzelnen Detektionsmodulen. Die Klasse `CameraCalibration` wird vom `LabyrinthTracker` zur Bereitstellung der Kalibrierungsparameter verwendet. Diese liefert die Kameramatrix sowie Verzerrungskoeffizienten, anhand derer die aufgenommenen Rohbilder entzerrt werden, bevor sie der eigentlichen Detektionspipeline zugeführt werden.

Die fünf Detektorklassen (`FieldDetector`, `BallDetector`, `HoleDetector`, `TargetDetector`, `WallDetector`) sind vollständig gekapselt und implementieren jeweils eine spezifische Erkennungsaufgabe. Jeder Detektor verfügt über eine definierte Schnittstelle zur Verarbeitung von Bilddaten und gibt strukturierte Ergebnisse zurück. Diese Kapselung gewährleistet eine klare Trennung der Verantwortlichkeiten und ermöglicht

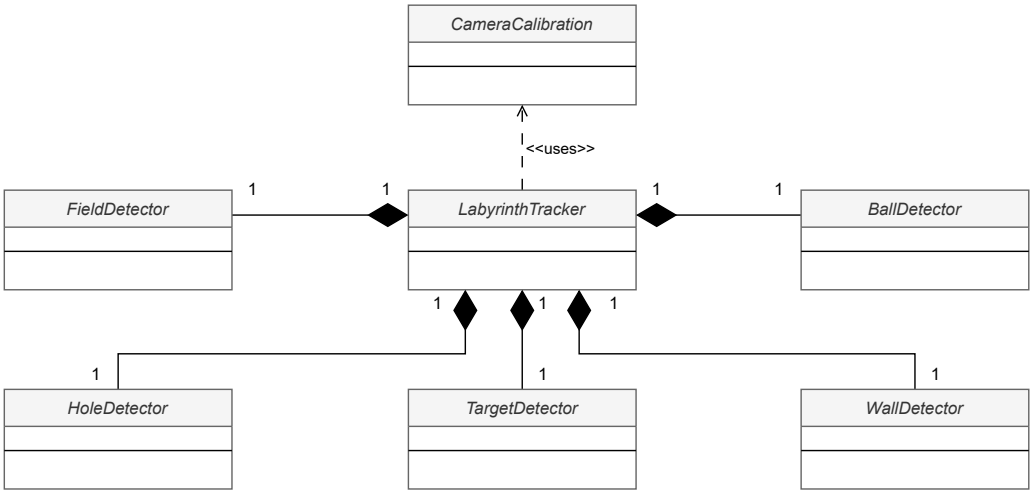


Abbildung 6.13: Klassendiagramm der Bildverarbeitungsarchitektur

es, einzelne Module unabhängig voneinander zu entwickeln, zu testen und bei Bedarf auszutauschen.

Die Architektur orientiert sich dabei an einem zentralen Steuerungselement, dem sogenannten LabyrinthTracker, welcher die Kamera initialisiert, den Bildstrom verwaltet und die verschiedenen Detektionsmodule koordiniert. Diese Module sind jeweils für die Erkennung einer spezifischen Bildinformation zuständig.

Der LabyrinthTracker implementiert zwei parallele Verarbeitungsthreads. Der Bilderfassungs-Thread übernimmt das kontinuierliche Einlesen der Kamerabilder, während der Bildverarbeitungs-Thread die Abarbeitung der Detektionspipeline ausführt. Diese Trennung verhindert Bildverluste bei hoher Auslastung.

Die Kameraeinstellungen sind in einer eigenen Klasse gekapselt. Dazu gehören unter anderem Parameter wie Belichtungszeit, Gain und Gamma-Wert. Die Klasse `CameraCalibration` implementiert die Kalibrierungsparameter und berechnet die Entzerrungsmaps einmalig. Diese werden bei jedem Frame auf das Eingangssignal angewendet.

Das System umfasst fünf Detektorklassen, die jeweils für die Extraktion einer bestimmten Bildkomponente zuständig sind. Tabelle 6.2 gibt einen Überblick über die implementierten Module sowie deren jeweilige Verantwortlichkeit, das eingesetzte Verfahren und die bereitgestellte Ausgabe.

Tabelle 6.2: Übersicht der implementierten Detektionsmodule

Modul	**Aufgabe**	**Verfahren**	**Ausgabe**
`BallDetector`	Kugelerkennung	HSV-Farbsegmentierung	Position, Radius
`HoleDetector`	Löcherdetektion	Schwellwertsegmentierung, Konturanalyse	Positionen, Masken
`WallDetector`	Wanderkennung	Kantendetektion	Liniensegmente
`TargetDetector`	Zielerkennung	Multi-Scale Template Matching	Position
`FieldDetector`	Spielfelderkennung	Kantendetektion	Bildausschnitt (Spielfeld)

Listing 6.4: Verarbeitungspipeline der Bildinformationen

```
def process_detection_pipeline(self, frame):
    # 1. Kamerakalibrierung und Entzerrung
    frame = self.apply_camera_calibration(frame)
    # 2. Spielfeldextraktion
    field_frame = self.field_detector.detect_field(frame)
    # 3. Objektdetektion
    field_frame = self.ball_detector.track_ball(field_frame)
    field_frame, hole_mask = self.hole_detector.detect_holes(field_frame)
    field_frame = self.target_detector.detect_target(field_frame)
    # 4. Wanderkennung mit Maskenausschluss
    ball_mask = self.create_ball_exclusion_mask()
    field_frame = self.wall_detector.detect_walls(
        field_frame, ball_mask, hole_mask
    )
    return field_frame
```

Die Bildverarbeitung erfolgt in einer festen Reihenfolge, die sich aus den funktionalen Abhängigkeiten der Module ergibt. Die zentrale Methode `process_detection_pipeline` verarbeitet jeden Frame wie in Listing 6.4 dargestellt.

Die modulare Struktur des Systems erlaubt eine einfache Erweiterbarkeit. Parameter lassen sich zur Laufzeit anpassen, und einzelne Bildverarbeitungsverfahren können modular ersetzt werden.

7 Evaluierung

Die Evaluierung in dieser Arbeit umfasst die Bewertung der Erkennungsleistung der entwickelten Bildverarbeitungskomponenten. Ziel ist es, die Qualität der einzelnen Detektoren im Hinblick auf deren Zuverlässigkeit und Genauigkeit zu prüfen. Dabei wird jeweils untersucht, ob das richtige Objekt erkannt wurde, ob es an der korrekten Position lokalisiert wurde und wie stabil die Erkennung über eine Vielzahl von Bildern hinweg erfolgt. Die Tests basieren auf den in Kapitel 2 und 3 vorgestellten Labyrinthmustern. Für jeden Detektor werden entsprechende Tests unter unterschiedlichen Bedingungen durchgeführt, wobei verschiedene Lichtverhältnisse berücksichtigt werden.

Die Evaluierung erfolgt unter definierten Rahmenbedingungen. Dazu zählen ausschließlich direkte Beleuchtungssituationen mit natürlichem Tageslicht (bei sonnigem und bewölktem Wetter) sowie Kombinationen aus Tageslicht und künstlicher Raumbeleuchtung. Diffuses Licht konnte im Aufbau nicht getestet werden, stellt jedoch einen denkbaren Aspekt für zukünftige Untersuchungen dar. Alle Testreihen werden mehrfach durchgeführt, um Aussagen über die Stabilität der Erkennung treffen zu können.

Für jeden der implementierten Detektoren werden im Vorfeld konkrete Zielkriterien definiert, anhand derer die Detektionsergebnisse bewertet werden. Diese Kriterien werden den gemessenen Resultaten gegenübergestellt.

7.1 Spielfelderkennung

Die Spielfelderkennung bildet das Fundament des Bildverarbeitungssystems, da alle weiteren Detektoren ausschließlich auf dem extrahierten Spielfeldbereich operieren. Eine fehlerhafte oder instabile Spielfelderkennung würde sich direkt auf die Leistung des gesamten Systems auswirken, da sie die Grundlage für die nachfolgenden Erkennungsschritte und letztlich für die Entscheidungsfindung des Agenten bildet. Daher liegt der Fokus der Evaluierung auf die Stabilität und Genauigkeit der Rahmenerkennung.

Im Unterschied zu den anderen Detektionsmodulen wird die Spielfelderkennung nicht mit unterschiedlichen Labyrinthmustern getestet, da sich ihre Aufgabe ausschließlich auf die Erkennung des festen Spielfeldrahmens bezieht.

7.1.1 Bewertungsmethodik

Da das Spielfeld im System nur einmalig beim Programmstart erfasst und anschließend als statische Referenz verwendet wird, erfolgen die Tests unter kontrollierten, aber realistischen Bedingungen. Es wird ausschließlich die Beleuchtung variiert, da Änderungen innerhalb des Spielfeldinhalts (z. B. Kugelposition) keinen Einfluss auf die Rahmenerkennung haben.

Ein korrekt erkanntes Spielfeld liegt vor, wenn der grüne Rahmen (vom Detektor erzeugt) exakt die Labyrinthplatte umfasst, auf der sich das Spielmuster befindet. Diese Definition orientiert sich am Spielfeldbereich der virtuellen Umgebung, da der trainierte Agent später mit identischen räumlichen Verhältnissen arbeiten muss. Das erkannte Spielfeld sollte weder die umgebenden roten Rahmenelemente einschließen noch Teile der Labyrinthplatte ausschließen. Eine erfolgreiche Detektion sollte zudem unabhängig von den Lichtverhältnissen erfolgen.

7.1.2 Testergebnisse

Alle Tests wurden mit identischen Kameraparametern durchgeführt, um Vergleichbarkeit sicherzustellen. Die Parameter waren dabei wie folgt festgelegt:

- Belichtungszeit: 10 000 µs
- Verstärkung (Gain): 1
- Weißabgleichmodus: *Continuous*
- Gamma-Wert: 1,0

Jedes Beleuchtungsszenario wurde unter konstanten Umgebungsbedingungen jeweils zehnmal getestet. Als erfolgreicher Durchlauf wurde eine Detektion gewertet, bei der das Spielfeld vollständig erkannt wurde.

Die Ergebnisse der Tests sind in Tabelle 7.1 zusammengefasst wobei folgendes gilt:

Tabelle 7.1: Erkennungsleistung der Spielfelderkennung bei jeweils 10 Testdurchläufen pro Lichtbedingung

Lichtbedingung	**TP (korrekt)**	**FP (inkorrekt)**	**FN (nicht erkannt)**	**TN**
Tageslicht	10	0	0	–
Bewölkt	3	0	7	–
Raumbeleuchtung	10	0	0	–

- **True Positive (TP)**: Das Spielfeld wurde korrekt erkannt.
- **False Positive (FP)**: Das System erkennt ein Spielfeld, das tatsächlich nicht oder nur fehlerhaft vorhanden ist.
- **False Negative (FN)**: Das Spielfeld ist im Bild vorhanden, wurde aber vom System nicht erkannt.
- **True Negative (TN)**: Dieser Fall ist im gegebenen Kontext nicht anwendbar, da das Spielfeld in allen Testbildern physisch vorhanden war. Daher bleibt die Spalte in der Tabelle unberücksichtigt.

Insgesamt wurden 30 Testdurchläufe durchgeführt (3 Szenarien mal 10 Versuche). Davon konnten 23 Fälle korrekt als *True Positives* gewertet werden, während in 7 Fällen eine Erkennung fehlschlug (*False Negatives*).

Es ist jedoch anzumerken, dass dieses Ergebnis mit Vorsicht zu betrachten ist. Obwohl das Spielfeld in den als *True Positive* gewerteten Fällen erkannt und grundsätzlich umfasst wurde, treten systematisch Abweichungen an den einzelnen Ecken des Spielfeldrahmens auf. Dies bedeutet, dass zwar eine Spielfelderkennung erfolgt, die extrahierten Rahmenkoordinaten jedoch nicht exakt mit den tatsächlichen Spielfeldgrenzen übereinstimmen.

Daraus ergibt sich eine Accuracy von:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{23 + 0}{30} \approx 76\,\%$$

Die geringere Gesamtgenauigkeit ist hauptsächlich auf die schlechte Erkennungsleistung unter dunklen, bewölkten Lichtverhältnissen zurückzuführen. In den anderen Szenarien wurde das Spielfeld jeweils in allen Fällen korrekt erkannt.

Abbildung 7.1 zeigt ein beispielhaftes Ergebnis der Spielfelderkennung. In der Aufnahme herrscht künstliche Raumbeleuchtung. Der rote Rahmen des physischen Spielfelds ist

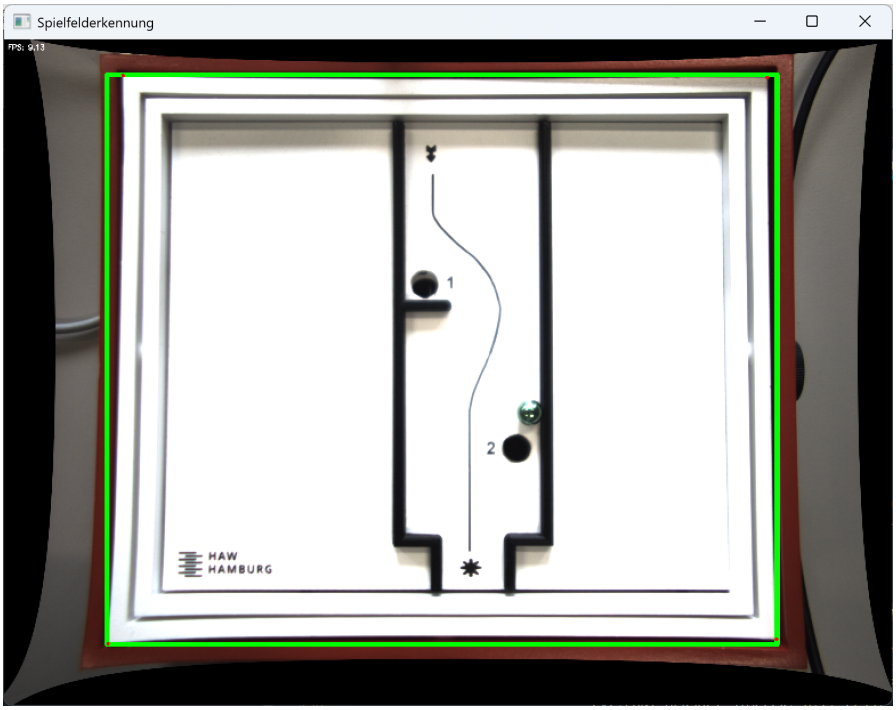


Abbildung 7.1: Beispielhafte Spielfelderkennung unter künstlicher Raumbeleuchtung

deutlich sichtbar, während der erkannte Bereich durch den überlagerten grünen Detektionsrahmen markiert ist. Die Abbildung veranschaulicht, dass die Detektion unter diesen Bedingungen grundsätzlich erfolgreich ist. Allerdings sind leichte Abweichungen an den Ecken erkennbar, bei denen der grüne Detektionsrahmen nicht exakt mit den Grenzen der Labyrinthplatte übereinstimmt. Diese Ungenauigkeiten könnten bei der präzisen Koordinatentransformation für die Agentenintegration zu Abweichungen führen.

7.1.3 Einordnung der Ergebnisse

Die Ergebnisse zeigen, dass die Spielfelderkennung unter normalen Lichtverhältnissen (sowohl bei Tageslicht als auch bei künstlicher Raumbeleuchtung) grundsätzlich funktioniert. Das System identifiziert den Spielfeldbereich stabil als Grundlage für alle nachfolgenden Verarbeitungsschritte.

Allerdings ist anzumerken, dass die erkannten Spielfeldrahmen systematische Abweichungen an den Ecken aufweisen und nicht exakt mit den in der virtuellen Umgebung definierten Spielfeldgrenzen übereinstimmen. Für eine nahtlose Übertragung des trainierten

Agenten auf das physische System wäre jedoch eine identische Rahmenkoordinatendefinition erforderlich, da der Agent mit denselben räumlichen Verhältnissen arbeiten muss wie in der Simulation. Diese Diskrepanz könnte bei der Agentenintegration zu Kompatibilitätsproblemen führen und erfordert gegebenenfalls eine Anpassung der Koordinatentransformation oder eine Nachkalibrierung der Spielfeldgrenzen.

Lediglich unter sehr schwachen Lichtbedingungen (bewölktes Tageslicht ohne zusätzliche Beleuchtung) kam es zu einer signifikant niedrigeren Erkennungsrate. Für zukünftige Arbeiten könnte hier eine Verbesserung der Robustheit durch angepasste Bildverarbeitungsverfahren sinnvoll sein.

Insgesamt erfüllt die Spielfelderkennung die grundlegende Anforderung der stabilen Detektion nur teils bis eventuell gar nicht in praxisrelevanten Lichtszenarien. Für zukünftige Arbeiten ist hier eine Nachbesserung erforderlich, um eine präzise und konsistente Rahmenextraktion zu gewährleisten, die mit den Definitionen der virtuellen Umgebung übereinstimmt und somit eine nahtlose Agentenintegration ermöglicht. Eine mögliche Lösung wäre die Implementierung eines perspektivischen Mappings über definierte Eckpunkte des Spielfelds. Durch die manuelle oder automatische Erfassung der vier Spielfeldecken könnte eine exakte Zuordnung zwischen dem Kamerabild und dem rechteckigen Spielfeld der virtuellen Umgebung hergestellt werden, wodurch die Koordinatengenauigkeit deutlich verbessert würde.

7.2 Kugelerkennung

Die Kugelerkennung ist für die Erfassung des einzigen dynamischen Objekts im System verantwortlich. Für den Agenten stellt sie die zentrale Grundlage zur Zustandserfassung dar. Eine präzise und kontinuierliche Lokalisierung ist daher zwingend erforderlich. Fehlerhafte oder instabile Kugelpositionen wirken sich unmittelbar auf das Entscheidungsverhalten des Agenten aus und können zu fehlerhaften Aktionen führen.

7.2.1 Bewertungsmethodik

Die Evaluierung erfolgt unter stabilen Lichtverhältnissen bei Raumbeleuchtung. Die gewählten Bedingungen gewährleisten reproduzierbare Ergebnisse und die sichere Spielfelderfassung.

Im Gegensatz zur statischen Objekterkennung ist die Bewertung der Kugelerkennung komplexer, da die Kugel sich in einem kontinuierlichen Bewegungszustand befindet. Eine manuelle Erfassung und Verifikation jeder Position über ein gesamtes Video hinweg ist unter Echtzeitbedingungen nicht praktikabel.

Zur objektiven Bewertung wird daher ein stichprobenbasierter Ansatz gewählt. Aus einem Testvideo mit bewegter Kugel werden 50 repräsentative Frames ausgewählt und die tatsächliche Kugelposition in jedem Frame manuell gelabelt. Anschließend wird überprüft, ob die vom System erkannte Position mit der gelabelten Ground-Truth-Position übereinstimmt.

Für die Bewertung werden folgende Metriken verwendet:

Erkennungsrate Dieser Kennwert beschreibt den Anteil der Frames, in denen das System die Kugel korrekt detektiert. Die Erkennungsrate berechnet sich wie folgt:

$$\text{Erkennungsrate} = \frac{\#\text{ korrekt erkannter Frames}}{\#\text{ gelabelter Frames}}$$

Positionsgenauigkeit Für alle korrekt erkannten Frames wird der euklidische Abstand zwischen der erkannten Position und der gelabelten Ground-Truth-Position berechnet. Der mittlere Abstand sowie die Standardabweichung dienen als Maß für die Positionsgenauigkeit:

$$\text{Fehler} = \sqrt{(x_{\text{erkannt}} - x_{\text{gelabelt}})^2 + (y_{\text{erkannt}} - y_{\text{gelabelt}})^2}$$

7.2.2 Testergebnisse

Die Tests wurden auf dem Labyrinthmuster `HOLES_2` durchgeführt. Für die Evaluierung wurde ein Video aufgenommen, in dem die Kugel von einer erhöhten Position nach unten rollt. Dabei durchläuft sie zunächst einen schattigeren Bereich, der durch die Raumbeleuchtung entsteht, und fällt schließlich in ein Loch. Aus diesem Video wurden 50 repräsentative Frames ausgewählt und die tatsächliche Kugelposition manuell gelabelt. In drei Frames war die Kugel bereits vollständig im Loch verschwunden und somit nicht mehr sichtbar. Diese wurden entsprechend als nicht vorhanden markiert.

Erkennungsrate Von den 50 gelabelten Frames wurde die Kugel in allen Fällen, in denen sie tatsächlich sichtbar war, korrekt vom System erkannt. In den drei Frames, in denen die Kugel bereits im Loch verschwunden war, wurde korrekterweise keine Kugel detektiert. Daraus ergibt sich:

$$\text{Erkennungsrate} = \frac{47}{47} = 1{,}0$$

Das System erreicht somit eine perfekte Erkennungsrate von 100 %.

Positionsgenauigkeit Für alle 47 korrekt erkannten Frames wurde der euklidische Abstand zwischen erkannter und gelabelter Position berechnet. Die Analyse zeigt eine perfekte Übereinstimmung:

- **Mittlerer Fehler:** 0 Pixel
- **Maximaler Fehler:** 0 Pixel
- **Standardabweichung:** 0 Pixel

Die erkannten Positionen stimmen in allen Frames exakt mit den manuell gelabelten Ground-Truth-Positionen überein. Dies bestätigt sowohl die hohe Zuverlässigkeit der Detektion als auch die präzise Positionsbestimmung, selbst unter erschwerten Bedingungen wie Schattenwurf.

Abbildung 7.2 zeigt einen repräsentativen Ausschnitt aus der Evaluierung. Die Visualisierung verdeutlicht die Übereinstimmung zwischen der manuell gelabelten Ground-Truth-Region (roter Kreis) und der vom System detektierten Position (grüner Mittelpunkt im Zentrum der Kugel).

7.2.3 Einordnung der Ergebnisse

Die Ergebnisse zeigen, dass die Kugelerfassung unter realistischen Bedingungen hervorragend funktioniert. Die Erkennungsrate von 100 % bestätigt, dass die Kugel in jedem Frame zuverlässig detektiert wird, auch bei Bewegung durch Bereiche mit erschwerter Beleuchtung. Die Positionsgenauigkeit mit 0 Pixel Abweichung belegt zudem, dass die ermittelten Koordinaten exakt mit den tatsächlichen Positionen übereinstimmen.

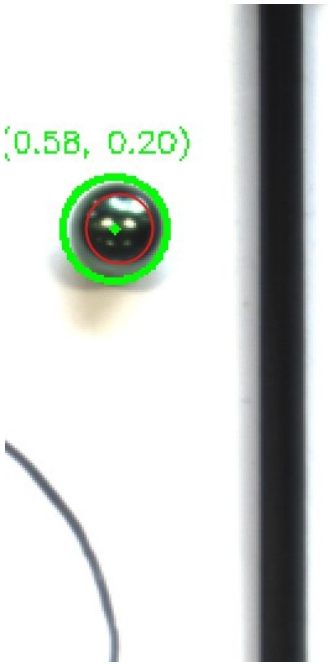


Abbildung 7.2: Exemplarischer Frame aus der Evaluierung der Kugelerkennung

Besonders bemerkenswert ist die Robustheit des Systems gegenüber Schattenwurf, wie er durch die Raumbeleuchtung im oberen Spielfeldbereich entsteht. Selbst unter diesen erschwerten Bedingungen bleibt die Detektion stabil und präzise. Die korrekte Erkennung des Verschwindens der Kugel im Loch (keine Falsch-Positiv-Detektion in den drei entsprechenden Frames) zeigt zudem, dass das System zuverlässig zwischen Anwesenheit und Abwesenheit der Kugel unterscheiden kann.

7.3 Locherkennung

Die Locherkennung erfasst eine bekannte, konstante Anzahl gleichartiger Objekte innerhalb des Spielfelds. In diesem Kontext ist insbesondere die Kennzahl des Recalls von zentraler Bedeutung, da das Übersehen der Löcher zu Problemen beim Training des Agenten führen kann. Ein hoher Recall gewährleistet, dass möglichst alle tatsächlich vorhandenen Objekte erkannt werden.

Tabelle 7.2: Ergebnisse der Locherkennung bei verschiedenen Labyrinthmustern unter Raumbeleuchtung

Labyrinth	**Löcher (gesamt)**	**TP**	**FP**	**FN**
HOLES_2	2	2	0	0
HOLES_8	8	8	0	0
HOLES_21	21	21	0	0
HOLES_59	59	57	3	2

7.3.1 Bewertungsmethodik

Die Evaluierung der Locherkennung erfolgt unter kontrollierten Bedingungen. Dabei werden ausschließlich Tests mit stabilen, guten Lichtverhältnissen durchgeführt, unter denen auch die Spielfelderkennung zuverlässig funktioniert. Zur Sicherstellung einer umfassenden Aussagekraft werden alle verfügbaren Labyrinthmuster in die Bewertung einbezogen. Ein erkanntes Loch wird dann als korrekt gewertet, wenn die zugehörige Detektion mindestens 70 % der tatsächlichen Lochfläche überdeckt.

7.3.2 Testergebnisse

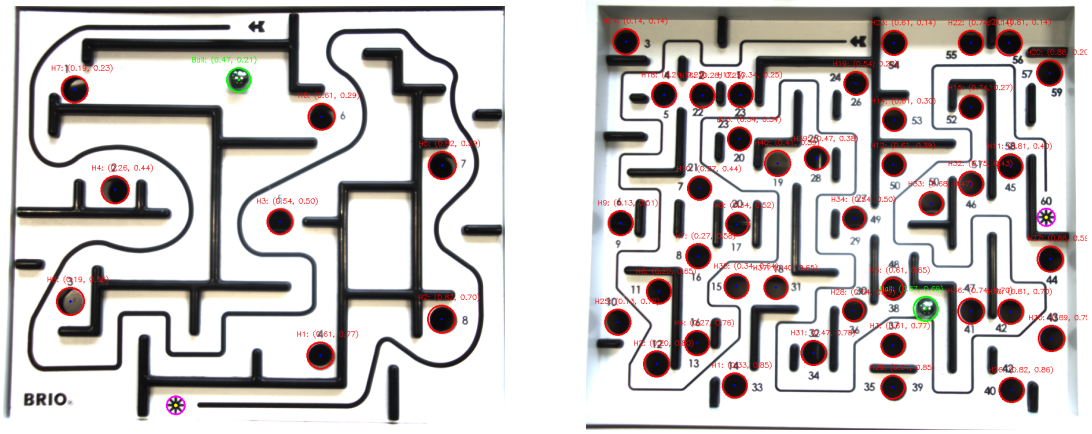
Für die nachfolgende Auswertung werden dieselben Kameraparameter wie bei der Spielfelderkennung verwendet. Dadurch wird sichergestellt, dass die Bedingungen zwischen den Detektionsmodulen konsistent bleiben und etwaige Unterschiede ausschließlich auf die jeweiligen Erkennungsverfahren zurückzuführen sind.

Im Rahmen der Evaluation wird zwischen den folgenden Kategorien unterschieden: TP bezeichnen korrekt erkannte Löcher, FP stehen für Regionen, die fälschlicherweise als Loch detektiert wurden, und FN beschreiben tatsächlich vorhandene, aber nicht erkannte Löcher.

Die Detektionsergebnisse für verschiedene Labyrinthmuster sind in Tabelle 7.2 dargestellt:

Die Ergebnisse zeigen, dass insbesondere bei Labyrinthmustern mit geringer bis mittlerer Anzahl an Löchern eine sehr zuverlässige Erkennung erreicht wird. Der insgesamt erzielte Recall über alle Testfälle hinweg beträgt:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{88}{88 + 2} = \frac{88}{90} \approx 0{,}978 \tag{7.1}$$



(a) Labyrinthmuster HOLES_8 (b) Labyrinthmuster HOLES_59

Abbildung 7.3: Ergebnisse der Locherkennung für unterschiedliche Labyrinthmuster.

Dies entspricht einem Gesamtrecall von 97,8 %.

Die wenigen falsch positiven Detektionen traten hauptsächlich bei komplexeren Labyrinthmustern auf. Dies lässt sich darauf zurückführen, dass diese Muster eine deutlich höhere Anzahl an kleineren Wandsegmenten aufweisen, die in ihrer Form und Helligkeit den Eigenschaften von Löchern ähneln. Je nach gewähltem Schwellwert für die Rundheit (*roundness*) werden solche Wandsegmente als Löcher interpretiert.

Eine Erhöhung des Rundheitsschwellwerts würde zwar die Anzahl falsch positiver Detektionen reduzieren, führt jedoch zu einem anderen Problem. Insbesondere bei komplexen Labyrinthmustern wie HOLES_59 befinden sich einige Löcher in unmittelbarer Nähe zu Wandsegmenten und werden zusätzlich durch Schattenwurf beeinflusst. Diese Löcher weisen dann eine geringere gemessene Rundheit auf und würden bei einem höheren Schwellwert nicht mehr erkannt werden (höhere FN-Rate).

Daher muss bei der Wahl des Rundheitsschwellwerts ein Kompromiss eingegangen werden. Die Entscheidung fiel zugunsten eines niedrigeren Schwellwerts, da es als wichtiger erachtet wird, dass alle tatsächlich vorhandenen Löcher erkannt werden, auch wenn dies mit einer geringfügig höheren Anzahl an Falsch-Positiv-Detektionen einhergeht. Für die Anwendung im Kontext der Agenten-Steuerung ist es kritischer, wenn ein Loch übersehen wird, als wenn ein Wandsegment fälschlicherweise als Loch klassifiziert wird.

Die Abbildungen 7.3(a) und 7.3(b) zeigen zwei beispielhafte Ergebnisse der durchgeführten Tests. Die erkannten Löcher sind jeweils rot markiert. Dargestellt sind die Labyrinth-

muster *HOLES_8* und *HOLES_59*, bei denen jeweils eine Erkennungsgenauigkeit von 100 % erzielt wurde. Im Fall von *HOLES_59* konnte dieses Ergebnis durch einen höheren Rundheitsgrenzwert und durch mehrmaliges leichtes Ausschwenken des Spielfelds erreicht werden. Letzteres führte dazu, dass Schatteneffekte und Reflexionen reduziert und einzelne Löcher klarer abgegrenzt wurden. Eine derartige Vorgehensweise könnte künftig gezielt genutzt werden, um die Erkennungsgenauigkeit zu verbessern, etwa indem zu Beginn des Spiels eine kurze Kalibrierungsphase vorgesehen wird, in der das Spielfeld aus verschiedenen Neigungswinkeln betrachtet wird, um potenzielle Störeinflüsse zu minimieren.

7.3.3 Einordnung der Ergebnisse

Der erzielte Recall-Wert von 97,8 % ist als sehr hoch einzustufen und belegt die Effektivität der Locherkennung unter den getesteten Bedingungen. Berücksichtigt man das anspruchsvollste Labyrinthmuster (HOLES_59) nicht, so ergibt sich sogar ein Recall von 100 %.

Bemerkenswert ist zudem, dass die wenigen falsch positiven Detektionen ausschließlich in Form von kleineren Wänden auftraten. Diese befinden sich nicht willkürlich auf freien Flächen, sondern stellen für sich genommen ebenfalls potenzielle Hindernisse dar. Insofern beeinträchtigen sie die Navigation des Agenten nicht negativ.

Die beiden nicht erkannten Löcher im Muster HOLES_59 befanden sich in unmittelbarer Nähe zu Wandsegmenten und waren von Schattenwurf betroffen, was ihre Rundheitswerte negativ beeinflusste. Für zukünftige Arbeiten könnte hier eine Verbesserung durch den Einsatz einer festen, an der Kamera montierten Beleuchtung erzielt werden. Eine solche aktive Beleuchtung würde Schattenwurf minimieren und damit konstantere Erkennungsbedingungen schaffen, was insbesondere bei komplexen Labyrinthmustern die Erkennungsrate weiter steigern könnte.

7.4 Zielerkennung

Die Zielerkennung muss bei allen vorhandenen Labyrinthmustern zuverlässig funktionieren, da sie die zentrale Bedingung für den erfolgreichen Abschluss des Spiels darstellt.

Tabelle 7.3: Erkennungsleistung der Zielerkennung bei jeweils 10 Testdurchläufen pro Labyrinthmuster

Labyrinthmuster	**TP**	**FP**	**FN**
HOLES_2	10	0	0
HOLES_8	10	0	0
HOLES_21	10	0	0
HOLES_59	10	0	0

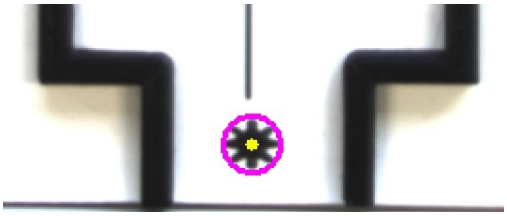


Abbildung 7.4: Exemplarische Zielerkennung im Labyrinthmuster `HOLES_2`

7.4.1 Bewertungsmethodik

Die Tests erfolgen unter gleichmäßiger Raumbeleuchtung. Alle verfügbaren Labyrinthmuster werden einbezogen. Eine erfolgreiche Erkennung liegt vor, wenn das Ziel korrekt lokalisiert wird. Als TP gilt ein korrekt erkanntes Ziel. Ein FP liegt vor, wenn ein Ziel fälschlicherweise erkannt wird. FN beschreibt Fälle, in denen das Ziel trotz Vorhandensein nicht erkannt wird.

7.4.2 Testergebnisse

Für jedes Labyrinthmuster wurden 10 Testdurchläufe durchgeführt. Eine Erkennung wurde als erfolgreich gewertet, wenn das Ziel im Detektionsbild mindestens zu 50 % überdeckt wurde. Die Ergebnisse sind in Tabelle 7.3 dargestellt.

Abbildung 7.4 veranschaulicht die Zielerkennung am Beispiel des Labyrinthmusters HOLES_2. Die Visualisierung zeigt, dass der Asterisk vollständig von der Detektionsmarkierung umschlossen wird und der Zielmittelpunkt exakt bestimmt werden kann. Diese präzise Lokalisierung ist entscheidend für die zuverlässige Bewertung, ob die Kugel das Ziel erreicht hat.

7.4.3 Einordnung der Ergebnisse

Die Ergebnisse sind optimal. Bei allen Labyrinthmustern konnte das Ziel mittels Template Matching in allen Versuchen erfolgreich und präzise erkannt werden.

7.5 Wanderkennung

Die Wändeerkennung stellt die abschließende Komponente des Bildverarbeitungssystems dar und vervollständigt die Zustandserfassung des Spielfelds. Wände bilden im Labyrinth die strukturgebenden Hindernisse, welche den Bewegungsraum der Kugel begrenzen und maßgeblich die Komplexität der Navigationsaufgabe bestimmen. Eine zuverlässige Erkennung dieser Strukturen ist daher hilfreich, damit der Agent sichere Pfade planen und Kollisionen vermeiden kann.

Im Gegensatz zu den übrigen Detektionsmodulen unterscheiden sich Anzahl, Form und Anordnung der Wände deutlich zwischen den verschiedenen Labyrinthmustern. Während Löcher und Ziel als punktuelle Objekte mit weitgehend konstanter Geometrie auftreten, handelt es sich bei Wänden um längliche, teils verzweigte Strukturen variabler Länge und Orientierung. Diese Heterogenität stellt besondere Anforderungen an das Erkennungsverfahren und erfordert eine robuste, anpassungsfähige Auswertung.

7.5.1 Bewertungsmethodik

Die Evaluierung der Wändeerkennung erfolgt analog zu den vorherigen Detektionsmodulen unter kontrollierten Lichtverhältnissen bei stabiler Raumbeleuchtung. Da die Wandstrukturen integraler Bestandteil der Labyrinthmuster sind, werden alle verfügbaren Muster (HOLES_2, HOLES_8, HOLES_21, HOLES_59) in die Bewertung einbezogen, um die Robustheit des Verfahrens über verschiedene Komplexitätsstufen hinweg zu prüfen.

Im Gegensatz zur Locherkennung, bei der diskrete Objekte mit bekannter Gesamtanzahl vorliegen, stellt die quantitative Bewertung der Wändeerkennung eine methodische Herausforderung dar. Wände bilden zusammenhängende, teilweise verzweigte Strukturen ohne eindeutig definierbare Einzelobjekte.

Daher wird zur Bewertung ein qualitatives Verfahren gewählt, das sich an den funktionalen Anforderungen des Systems orientiert. Für jedes Labyrinthmuster wird visuell

Tabelle 7.4: Evaluierung der Wändeerkennung

Labyrinthmuster	**Erfolgreich**	**Fehlgeschlagen**
HOLES_2	10	0
HOLES_8	9	1
HOLES_21	9	1
HOLES_59	8	2

geprüft, ob die wesentlichen Wandstrukturen erkannt wurden. Eine erfolgreiche Detektion liegt vor, wenn:

- alle bedeutenden Wandsegmente, die den Bewegungsraum der Kugel begrenzen, erfasst werden
- die erkannten Wände eine zusammenhängende und nachvollziehbare Struktur bilden
- keine größeren freien Flächen fälschlicherweise als Wände klassifiziert werden

Kleinere Lücken in der Wandkontur oder geringfügige Abweichungen an den Rändern sind tolerierbar, solange die grundlegende Topologie des Labyrinths korrekt abgebildet wird. Entscheidend ist, dass der Agent auf Basis der erkannten Wandinformationen eine realistische Repräsentation der Spielumgebung erhält. Jedes Labyrinthmuster wird in 10 aufeinanderfolgenden Frames analysiert. Die Bewertung erfolgt binär: Ein Testdurchlauf gilt als erfolgreich, wenn die oben genannten Kriterien erfüllt sind, andernfalls als fehlgeschlagen.

7.5.2 Testergebnisse

Für die Wändeerkennung werden dieselben Kameraparameter wie bei den vorherigen Detektionsmodulen verwendet, um eine konsistente Vergleichbarkeit zu gewährleisten. Jedes Labyrinthmuster wurde über 10 aufeinanderfolgende Frames hinweg analysiert und gemäß der definierten qualitativen Kriterien bewertet.

Die Ergebnisse der visuellen Auswertung sind in Tabelle 7.4 zusammengefasst:

Die Wändeerkennung erfasst die Wandstrukturen als Ansammlung einzelner Fragmente, die in ihrer Gesamtheit die zusammenhängenden Wandverläufe des Labyrinths abbilden.

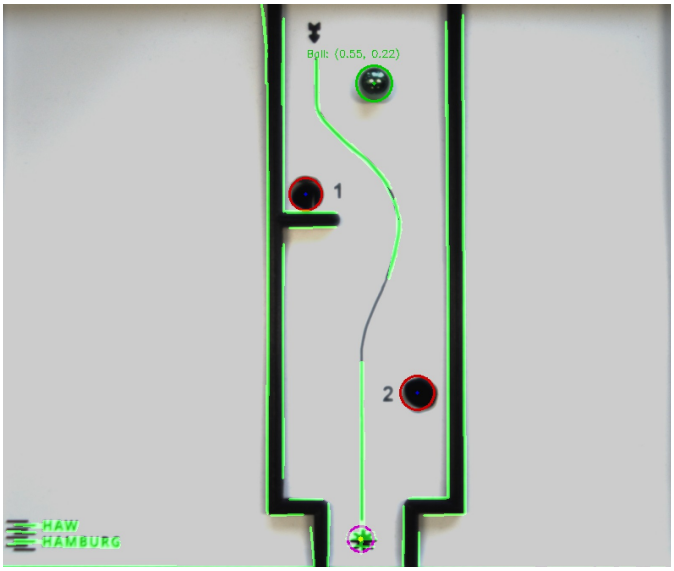


Abbildung 7.5: Wanderkennung im Labyrinthmuster HOLES_2

Bei allen getesteten Labyrinthmustern wurden die wesentlichen Wandsegmente durchgängig erkannt. Die wenigen als fehlgeschlagen gewerteten Frames traten ausschließlich bei komplexeren Mustern auf und betrafen kleine Wandvorsprünge oder Ecken, deren äußerste Spitzen nicht vollständig erfasst wurden. Diese Unvollständigkeiten sind jedoch marginal und beeinträchtigen die funktionale Nutzbarkeit der Wandinformationen nicht signifikant.

Es zeigt sich jedoch eine wiederkehrende Schwäche des Verfahrens. Neben den tatsächlichen Wänden wird auch die im Spielfeld integrierte Führungslinie, die von Start zu Ziel verläuft, teilweise als Wandstruktur detektiert. Verschiedene Ansätze zur Filterung dieser Falsch-Positiven wurden erprobt, führten jedoch zu einer Verschlechterung der eigentlichen Wanderkennungsleistung. Die Führungslinie weist in ihrer visuellen Erscheinung (Farbe, Kontrast, Geometrie) teils ähnliche Merkmale wie schmale Wandsegmente auf, was eine rein bildbasierte Unterscheidung erschwert.

Abbildung 7.5 zeigt ein beispielhaftes Detektionsergebnis auf dem Labyrinthmuster HOLES_2. Die erkannten Wandfragmente sind durch farbige Markierungen hervorgehoben. Es ist zu erkennen, dass die Hauptwandstrukturen vollständig erfasst wurden. Darüber hinaus werden auch Teile der Führungslinie als Wandsegmente klassifiziert. Dieses Verhalten ist auf die strukturelle Ähnlichkeit zwischen der Linie und schmalen Wandelementen zurückzuführen.

7.5.3 Einordnung der Ergebnisse

Die Evaluierung der Wändeerkennung zeigt ein gemischtes Ergebnis. Einerseits werden nahezu alle relevanten Wandstrukturen über alle Labyrinthmuster hinweg zuverlässig erkannt, was grundsätzlich auf ein funktionsfähiges Detektionsverfahren hindeutet. Die hohe Vollständigkeit der Erkennung, auch bei komplexen Mustern mit zahlreichen verzweigten Wandsegmenten, erfüllt die primäre funktionale Anforderung.

Andererseits stellt die systematische Fehlklassifikation der Führungslinie als Wandstruktur eine signifikante Einschränkung dar. Diese Falsch-Positiven sind nicht zufällig verteilt, sondern treten konsistent auf und würden einem Agenten eine verfälschte Repräsentation der navigierbaren Umgebung vermitteln.

Im Kontext dieser Arbeit ist diese Einschränkung jedoch von geringer praktischer Bedeutung. Der im Rahmen der Arbeit trainierte Agent lernte ausschließlich in einer virtuellen Simulationsumgebung, in der keine explizite Information über die Wände enthalten war. Die Zustandsrepräsentation, auf deren Basis das Agentennetzwerk trainiert wurde, umfasste somit keine Angaben zur Position oder Erkennung der Wände. Bei der Übertragung des trainierten Modells auf den physischen Demonstrator stellt die fälschlicherweise erkannte Führungslinie daher kein unmittelbares Hindernis für die erfolgreiche Sim-to-Real-Übertragung dar, da der Agent nicht gelernt hat, auf diese Struktur zu reagieren.

Anders verhält es sich für zukünftige Anwendungsszenarien, in denen Agenten direkt am physischen Demonstrator trainiert werden sollen. In solchen Fällen würde die Fehlklassifikation der Führungslinie das Lernverhalten beeinflussen und möglicherweise zu suboptimalen Strategien führen. Für derartige Einsatzszenarien wäre eine Vorverarbeitung erforderlich, beispielsweise durch physisches Abdecken der Führungslinie während des Trainings oder durch eine erweiterte Nachbearbeitung der Detektionsergebnisse.

Zusammenfassend lässt sich festhalten, dass die Wändeerkennung die grundlegende Anforderung der vollständigen Strukturerfassung weitgehend erfüllt, jedoch mit der Einschränkung systematischer Falsch-Positiven behaftet ist. Die praktische Eignung ist stark vom konkreten Anwendungskontext abhängig und erfordert gegebenenfalls zusätzliche Maßnahmen zur Filterung nicht relevanter visueller Elemente.

7.6 Anforderungsprüfung

Im Rahmen der Evaluierung ist es von zentraler Bedeutung zu überprüfen, inwieweit das entwickelte System die im Kapitel 4 beschriebenen Anforderungen erfüllt. Diese Anforderungsprüfung dient dazu, die Zielerreichung der Arbeit objektiv zu bewerten und festzustellen, ob die im Konzeptions- und Implementierungsprozess definierten Kriterien erfolgreich umgesetzt wurden.

Im Folgenden werden die einzelnen Anforderungen systematisch betrachtet und hinsichtlich ihrer Erfüllung analysiert. Dabei wird zwischen funktionalen und nicht-funktionalen Anforderungen unterschieden. Jede Anforderung wird einzeln geprüft und ihr Erfüllungsgrad bewertet.

7.6.1 Funktionale Anforderungen

Im Folgenden werden die funktionalen Anforderungen des physischen Demonstrators überprüft. Für jede Anforderung wird dargelegt, inwieweit diese durch das entwickelte System erfüllt wurde.

- **PD-FA1: Export der Zustandsdaten**
  Erfüllungsgrad: Erfüllt
  Das System erfasst die Zustandsinformationen und gibt diese in einem standardisierten Format aus. Nach der Erkennung des Spielfelds werden die Koordinaten normiert auf das Seitenverhältnis des Spielfelds bereitgestellt, wodurch eine einheitliche Basis für weitere Analysen geschaffen wird.

- **PD-FA2: Manueller Spielstart**
  Erfüllungsgrad: Erfüllt
  Das System ermöglicht die manuelle Initialisierung des Ablaufs. Der Benutzer kann den Prozess aktiv starten und das Labyrinth bei Bedarf manuell bedienen, wodurch flexible Testszenarien ermöglicht werden.

- **PD-FA3: Erfassung der Kugelposition**
  Erfüllungsgrad: Erfüllt
  Die Position der Kugel wird lückenlos während des gesamten Spielzyklus erfasst. Durch den Einsatz von Farbsegmentierung konnte eine zuverlässige Detektion erreicht werden, die den vollständigen Bewegungsverlauf der Kugel innerhalb des

Spielfeldbereichs korrekt abbildet. Die Evaluierung hat gezeigt, dass die Daten kontinuierlich und ohne Aussetzer bereitgestellt werden, wodurch eine vollständige Zustandsverfolgung gewährleistet ist.

- **PD-FA4: Erfassung des Spielfeldrahmens**
  Erfüllungsgrad: Teilweise erfüllt
  Der Spielfeldrahmen wird zuverlässig erkannt und die Grenzen des Spielfelds können präzise bestimmt werden. Allerdings entsprechen die erfassten Rahmenkoordinaten nicht exakt denen der virtuellen Umgebung. Diese Diskrepanz könnte bei der Übertragung trainierter Agenten aus der Simulation auf das reale System zu Kompatibilitätsproblemen führen und erfordert gegebenenfalls eine Anpassung oder eine Nachkalibrierung.

- **PD-FA5: Erfassung von Merkmalen verschiedener Muster**
  Erfüllungsgrad: Erfüllt
  Das System erfasst charakteristische Merkmale unterschiedlicher Muster auf dem Spielfeld zuverlässig. Ziel, Löcher und Wände werden korrekt erkannt und voneinander unterschieden, wodurch eine vollständige Zustandserfassung des Labyrinths ermöglicht wird.

- **PD-FA6(Opt): Aktive Beleuchtungsunterstützung**
  Erfüllungsgrad: Nicht umgesetzt
  Diese optionale Anforderung wurde nicht implementiert. Die Evaluierung hat gezeigt, dass zur Erfüllung aller anderen funktionalen Anforderungen keine aktive Beleuchtungsunterstützung erforderlich ist. Das System arbeitet unter normalen Raumbedingungen zuverlässig.

7.6.2 Nicht-funktionale Anforderungen

Im Folgenden werden die nicht-funktionalen Anforderungen des physischen Demonstrators überprüft. Diese betreffen qualitative Eigenschaften des Systems, die über das reine funktionale Verhalten hinausgehen.

Systemdesign und Integration

- **PD-NFA1: Standardisierte Spielfeldgröße**
  Erfüllungsgrad: Erfüllt
  Die Anforderung ist gewährleistet, da sowohl mit den originalen Spielfeldern als auch mit selbst erstellten Spielplatten gearbeitet wurde, die die geforderten Abmessungen von 27,4 cm × 22,8 cm erfüllen.

- **PD-NFA2: Spezifikation der Spielkugel**
  Erfüllungsgrad: Erfüllt
  Statt der ursprünglich verwendeten silbernen Kugel wurde eine grüne Kugel eingesetzt. Es handelt sich jedoch weiterhin um eine Stahlkugel mit identischem Radius von 6,35 mm und Gewicht von 9 g, sodass die physikalischen Eigenschaften der Anforderung entsprechen.

- **PD-NFA3: Nicht-invasive Motorintegration**
  Erfüllungsgrad: Erfüllt
  Die Motoren sind so angebracht, dass keine dauerhaften oder invasiven Veränderungen an der Struktur des Labyrinths vorgenommen wurden. Die Anforderung ist vollständig erfüllt.

- **PD-NFA4: Kompatibilität selbstgebauter Labyrinthplatten**
  Erfüllungsgrad: Erfüllt
  Die selbstgefertigte Spielplatte HOLES_2 weist beide Löcher exakt an den Positionen auf, die dem originalen Labyrinth entsprechen, wodurch die geforderte Kompatibilität gewährleistet ist.

Datenqualität und Zuverlässigkeit

- **PD-NFA5: Zeitliche Stabilität der Erfassung**
  Erfüllungsgrad: Erfüllt
  Durch die Verteilung der Aufgaben auf separate Threads sowie den Einsatz leistungsstarker Hardware werden konstant Bildfrequenzen von über 30 FPS erreicht. Dies gewährleistet eine zeitlich stabile Erfassung und reproduzierbare Ergebnisse.

- **PD-NFA6: Genauigkeit der Positionserfassung**
  Erfüllungsgrad: Erfüllt
  Die Evaluierung zeigt, dass die geforderte Genauigkeit der Positionserfassung erreicht wurde. Die detaillierten Messergebnisse sind in den vorangegangenen Abschnitten dieses Kapitels dargelegt.

- **PD-NFA7: Robustheit gegenüber Umgebungseinflüssen**
  Erfüllungsgrad: Weitgehend erfüllt
  Das System funktioniert einwandfrei unter normalen Raumbeleuchtungsbedingungen. Die Evaluierung zeigt zudem, dass die entwickelten Methoden auch bei teilweise abweichenden Lichtverhältnissen noch funktionsfähig bleiben, wenn auch mit leicht reduzierter Zuverlässigkeit.

- **PD-NFA8: Zuverlässigkeit der Ereignisdetektion**
  Erfüllungsgrad: Erfüllt
  Die Evaluierung hat für alle relevanten Spielereignisse gute bis sehr gute Erkennungsraten nachgewiesen, wodurch eine verlässliche Systemüberwachung gewährleistet ist.

- **PD-NFA9: Kalibrierbarkeit**
  Erfüllungsgrad: Erfüllt
  Der eingesetzte Sensor, die Kamera, kann über mehrere Wege kalibriert werden: durch Anpassung des Objektivs, interne Softwareeinstellungen sowie etablierte Kalibrierungsverfahren. Dies ermöglicht eine systematische Kompensation von Messfehlern.

- **PD-NFA10: Visualisierung der Zustandsdaten**
  Erfüllungsgrad: Erfüllt
  Alle entwickelten Methoden werden per Video-Stream dargestellt. Die erfassten Objekte werden direkt im Bild visualisiert und ihre Position innerhalb des Spielfelds angezeigt, wodurch Tests und Analysen erheblich erleichtert werden.

Technische Umsetzbarkeit und Wartung

- **PD-NFA11: Strukturierter und kommentierter Quellcode**
  Erfüllungsgrad: Erfüllt
  Der entwickelte Quellcode wurde systematisch in Module aufgeteilt und umfassend kommentiert. Dies stellt Übersichtlichkeit, Wartbarkeit, Erweiterbarkeit und Nachvollziehbarkeit sicher.

- **PD-NFA12: Kompaktheit und Transportfähigkeit**
  Erfüllungsgrad: Erfüllt
  Der Demonstrator kann durch die Trennung von Kameragehäuse und Spielfeld

in separate Komponenten zerlegt und einzeln transportiert werden, wodurch ein flexibler Einsatz ermöglicht wird.

Projektbegrenzungen

- **PD-NFA13: Budgetbeschränkung**
  Erfüllungsgrad: Erfüllt
  Da Kamera und Objektiv zur Verfügung gestellt wurden und für die restliche Implementierung auf bereits vorhandene Komponenten zurückgegriffen wurde, wurde das vorgegebene Budget von 300 € nicht überschritten.

8 Fazit und Ausblick

In dieser Arbeit stand die Weiterentwicklung eines bestehenden Projekts im Mittelpunkt, dessen Ziel es war, eine experimentelle Plattform für Deep Reinforcement Learning im Kontext eines Kugellabyrinths zu erweitern. Aufbauend auf den bisherigen Arbeiten wurde das System um eine Bildverarbeitungskomponente ergänzt, die es ermöglicht, den Übergang von der virtuellen Trainingsumgebung auf den physischen Demonstrator zu realisieren. Dabei wurde besonderer Wert darauf gelegt, dass die Bildverarbeitung allgemein gültig arbeitet, also unabhängig vom jeweiligen Labyrinthmuster in der Lage ist, die in allen Varianten bestehenden relevanten Informationen zuverlässig zu erfassen. Durch diese Integration sollte untersucht werden, ob die in der Simulation trainierten Agenten mithilfe der kamerabasierten Wahrnehmung auch auf dem realen Aufbau anwendbar sind. Gleichzeitig wurde die Grundlage geschaffen, um zukünftig auch direkt auf dem Demonstrator weitere Reinforcement Learning Agenten trainieren zu können.

Die dabei ausgewählten und verarbeiteten Informationen stellten eine Kombination aus bereits für den trainierten Agenten relevanten Merkmalen wie der Spielfelderkennung, der Kugelposition und der Zielerfassung sowie zusätzlichen Informationen für potenziell neu zu trainierende Agenten dar, beispielsweise der Wanderkennung. Durch die entwickelten Verfahren konnten unter geeigneten Beleuchtungsbedingungen, insbesondere bei konstanter Raumbeleuchtung, Ergebnisse erzielt werden, die eine zuverlässige und wiederholbare Erkennung der relevanten Objekte ermöglichen. Damit wurde ein funktionales Fundament geschaffen, das sowohl die Stabilität des bestehenden Systems gewährleistet als auch eine Weiterentwicklung der Lern- und Steuerungsmechanismen unterstützt.

Für das Erreichen der jeweiligen Ergebnisse wurden für jedes relevante Objekt verschiedene Methoden der klassischen Bildverarbeitung miteinander verglichen, bewertet und anschließend eine geeignete Vorgehensweise ausgewählt und optimiert. Dabei wurde stets berücksichtigt, dass es sich um ein dynamisches Spielfeld handelt, auf dem die Objekte während der gesamten Aufnahmedauer zuverlässig erkannt werden müssen. Durch mehrere Evaluierungsschritte wurde jede angewandte Methode systematisch geprüft, um die

Erkennungsleistung zu bewerten und mögliche Verbesserungen gezielt umzusetzen oder zumindest zu erkennen und zu erfassen.

Insgesamt wurde somit ein System entwickelt, das in der Lage ist, wesentliche Objekte und Informationen aus dem Labyrinth zu erfassen und zu extrahieren, wobei die Verfahren auf unterschiedliche Labyrinthmuster anwendbar sind. Auch wenn die erzielten Ergebnisse eine solide Grundlage darstellen, ist das System noch nicht in allen Bereichen vollständig ausgereift. Sowohl in der Bildverarbeitung als auch in weiteren Aspekten, die für einen erfolgreichen Transfer der trainierten Agenten auf den physischen Demonstrator entscheidend sind, besteht noch Optimierungsbedarf.

Ein zentraler Aspekt zukünftiger Arbeiten betrifft die Weiterentwicklung der Spielfelderkennung. Obwohl die aktuelle Erkennung bereits solide Ergebnisse liefert, gelingt es bislang nicht, das exakte, in der virtuellen Umgebung definierte Spielfeld eindeutig zu identifizieren. Für einen erfolgreichen Transfer der trainierten Agenten ist jedoch entscheidend, dass die in der realen Umgebung erfassten Informationen möglichst exakt den Bedingungen der virtuellen Umgebung entsprechen. Erste Ansätze zur Erweiterung dieser Erkennung wurden erprobt, erwiesen sich jedoch als anspruchsvoll, da die Kanten des Spielfeldes im Kamerabild nur schwach ausgeprägt sind und sich kaum vom Hintergrund unterscheiden. Potenzielle Verbesserungsansätze bestehen darin, die Ecken des Spielfelds gezielt über definierte Regionen von Interesse vorab zu erfassen oder durch physische Markierungen an den jeweiligen Eckpunkten eine robustere Detektion zu ermöglichen.

Darüber hinaus sollte künftig auch die Neigung des Spielfelds berücksichtigt werden, da sie einen dynamischen Einflussfaktor darstellt und somit ebenfalls Teil der kontinuierlichen Erkennung sein muss. Eine präzise Erfassung der Neigung könnte beispielsweise durch ein initiales Kalibrierungsverfahren erfolgen, bei dem das Spielfeld zu Beginn in verschiedene Richtungen ausgelenkt wird, um Referenzdaten zu gewinnen. Während des Spielbetriebs ließe sich die aktuelle Neigung dann anhand der Abweichungen von diesen Referenzwerten approximieren.

Ein weiterer wesentlicher Aspekt für zukünftige Arbeiten betrifft das Zeitverhalten des Systems und dessen Abbildung in der Simulation. In der physischen Umsetzung erfolgt die Ausführung einer Aktion stets mit zeitlicher Verzögerung gegenüber dem zuvor erfassten Zustand. Diese Verzögerung ergibt sich aus der Abfolge von Bildaufnahme (Image Acquisition, IA), Bildverarbeitung (Image Processing, IP), Aktionsbestimmung durch den Agenten und der anschließenden physischen Ausführung der Aktion über die Motoransteuerung.

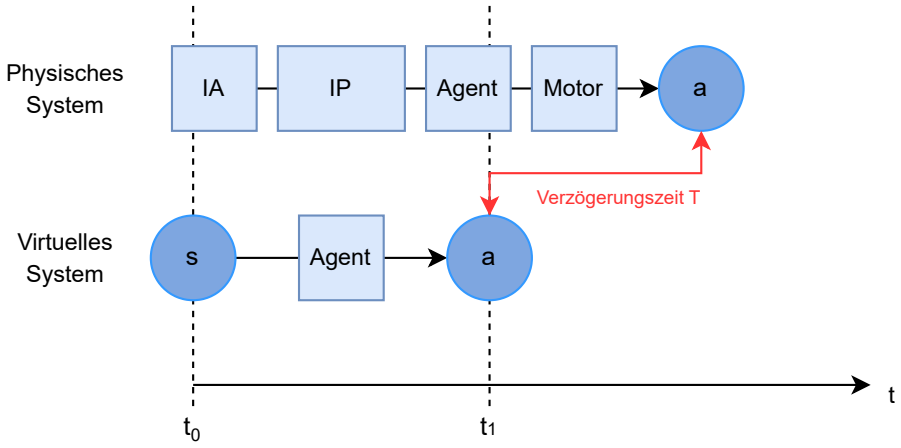


Abbildung 8.1: Aktuelles Szenario

Abbildung 8.1 veranschaulicht diesen Ablauf. Im oberen Teil der Darstellung ist das physische System gezeigt und umfasst dabei die gesamte Verarbeitungsdauer von der Bildaufnahme bis zur Motoransteuerung. Im unteren Teil ist die virtuelle Umgebung dargestellt, in der Zustand s und Aktion a theoretisch ohne realistische Zeitverzögerung ablaufen können. Diese Diskrepanz zwischen Simulation und Realität (Verzögerungszeit T) kann zu einer mangelnden Übertragbarkeit der trainierten Agenten auf das physische System führen.

Um ein realistischeres Verhalten zu modellieren und die Übertragbarkeit der trainierten Agenten zu verbessern, sollte daher in der Simulation ebenfalls eine zeitliche Verzögerung zwischen Zustand und Aktion implementiert werden. Abbildung 8.2 zeigt zwei mögliche Optimierungsansätze. Zum einen kann die Verarbeitungszeit im physischen System durch Optimierung der Bildaufnahme und -verarbeitung reduziert werden (Optimierung 1). Beispielsweise könnte durch eine Verringerung der Belichtungszeit (Exposure Time) die Bildaufnahmegeschwindigkeit erhöht werden, oder durch Reduktion der Bildauflösung die Verarbeitungslast verringert werden. Zum anderen sollte in der virtuellen Umgebung eine konfigurierbare Verzögerung T implementiert werden (Optimierung 2), die an die tatsächlich gemessenen Bedingungen des physischen Systems angepasst werden kann. Durch diese Synchronisation wird sichergestellt, dass beide Systeme Aktionen zum selben relativen Zeitpunkt ausführen, was die Sim-to-Real-Übertragung deutlich verbessern sollte.

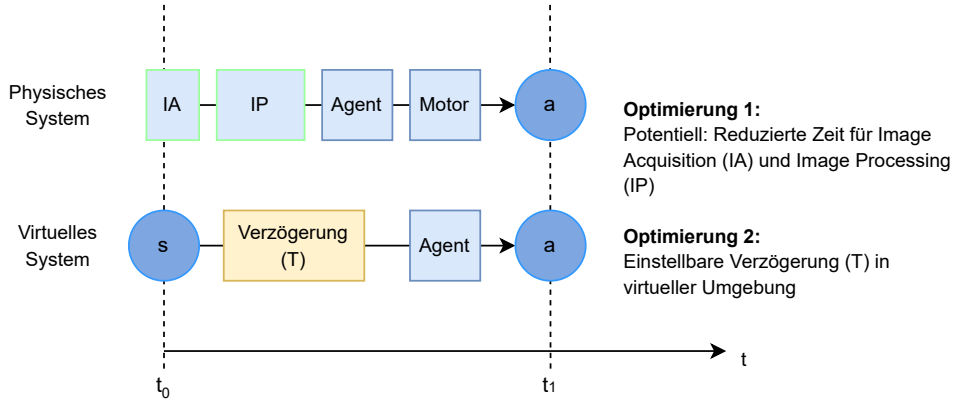


Abbildung 8.2: Optimiertes Szenario

Darüber hinaus eröffnet sich die Möglichkeit, das Zeitverhalten des physischen Systems durch den Einsatz von Multithreading weiter zu optimieren. In diesem Ansatz würden Bildaufnahme, Bildverarbeitung und Aktionsberechnung in getrennten Threads ausgeführt, wodurch sich die Gesamtlatenz des Systems reduzieren ließe. So könnte beispielsweise bereits die nächste Bildaufnahme erfolgen, während die vorherige Aufnahme noch verarbeitet und die zugehörige Aktion ausgeführt wird. Entscheidend ist dabei, dass die Bildaufnahmen in einem festen Takt erfolgen und die zugehörigen Aktionen zeitlich konsistent ausgeführt werden, um ein stabiles und reproduzierbares Systemverhalten sicherzustellen.

Ein weiterer vielversprechender Schritt in der zukünftigen Weiterentwicklung des Systems besteht darin, das Training der Agenten direkt auf dem physischen Demonstrator durchzuführen. Dabei stellt insbesondere die physische Handhabung der Kugel eine Herausforderung dar, da sie nach jedem Fehlversuch, Feststecken oder Erreichen des Ziels erneut in die Startposition gebracht werden muss. Zusätzlich erfordert ein solches Training eine ausreichend leistungsfähige Recheneinheit, um die umfangreichen Berechnungen in Echtzeit auszuführen.

Anstelle eines stationären PCs oder Laptops könnte hierfür der Einsatz eines kompakten, energieeffizienten Computers in Betracht gezogen werden, der speziell für KI-Anwendungen im Echtzeitbetrieb konzipiert ist. Systeme aus der NVIDIA-Jetson-Reihe bieten hierfür eine geeignete Grundlage, da sie eine kontinuierliche Ausführung des Lernprozesses ermöglichen und gleichzeitig für den dauerhaften Betrieb, etwa über längere

Trainingsphasen hinweg, ausgelegt sind. Ein solches Setup würde den Demonstrator zu einer vollständig autonomen Lernplattform erweitern.

9 Danksagung

Am Ende dieser Arbeit möchte ich die Gelegenheit nutzen, einigen Personen meinen Dank auszusprechen, die mich während der Zeit meiner Abschlussarbeit auf unterschiedliche Weise unterstützt haben.

Mein besonderer Dank gilt meinem betreuenden Professor, Prof. Dr. Marc Hensel. Er hat mich auf vielfältige Weise unterstützt, sei es durch die Bereitstellung und Hilfestellung bei verschiedenen Punkten oder durch die sehr gute Betreuung, die ich während der gesamten Arbeit erfahren durfte. Nicht zuletzt ist es ihm zu verdanken, dass ich ein so spannendes und motivierendes Thema bearbeiten konnte.

Ebenso möchte ich mich bei meinem Kommilitonen Arbin Medi bedanken. Wir haben während der Masterarbeit, aber auch im gesamten Studienverlauf, stets zusammengearbeitet, uns gegenseitig unterstützt und von der gemeinsamen Arbeit profitiert.

Abschließend danke ich meiner Familie, die mir in dieser Zeit Rückhalt und Motivation gegeben hat. Besonders meiner Frau gilt mein Dank für ihre Unterstützung, die mir in allen Phasen der Arbeit sehr geholfen hat.

Literaturverzeichnis

[1] An, Gwon H. ; Lee, Siyeong ; Seo, Min-Woo ; Yun, Kugjin ; Cheong, Won-Sik ; Kang, Suk-Ju: Charuco board-based omnidirectional camera calibration method. In: *Electronics* 7 (2018), Nr. 12, S. 421

[2] Aradi, Szilárd: Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles. In: *IEEE Transactions on Intelligent Transportation Systems* 23 (2022), Nr. 2, S. 740–759

[3] Bi, Thomas ; D'Andrea, Raffaello: Sample-efficient learning to solve a real-world labyrinth game using data-augmented model-based reinforcement learning. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)* IEEE (Veranst.), 2024, S. 7455–7460

[4] Bredies, Kristian ; Lorenz, Dirk: *Mathematische Bildverarbeitung*. Bd. 1. Springer, 2011

[5] Contributors, OpenCV: *Camera Calibration and 3D Reconstruction*. https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html. 2025. – URL https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html. – Zugriff am 9. Oktober 2025

[6] Dahlkemper, Prof. Dr.-Ing.: *Angewandte Industrielle Bildverarbeitung*. 2024. – Version 7.0, Stand: 26. März 2024

[7] Escobar-Naranjo, Juan ; Caiza, Gustavo ; Ayala, Paulina ; Jordan, Edisson ; Garcia, Carlos A. ; Garcia, Marcelo V.: Autonomous Navigation of Robots: Optimization with DQN. In: *Applied Sciences* 13 (2023), Nr. 12. – URL https://www.mdpi.com/2076-3417/13/12/7202. – ISSN 2076-3417

[8] Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme (IAIS): *Maschinelles Lernen – Ein Überblick*. September 2018. – URL https://www.iais.fraunhofer.de/content/dam/iais/iais-dokumen

te/studien-und-whitepaper/Maschinelles_Lernen_Whitepaper_Fraunhofer_IAIS_2018.pdf. – Zugriff am 22.08.2025

[9] Frochte, Jörg: *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. 3., überarbeitete und erweiterte Auflage. München : Carl Hanser Verlag, 2021 (Hanser eLibrary). – ISBN 9783446461444

[10] Hensel, Marc: *Klassendiagramm zur Kamerasteuerung*. persönliche Mitteilung per E-Mail. 2024. – Erhalten am 12. Dezember 2024

[11] International Organization for Standardization: *ISO/IEC 9126-1:2001 – Software engineering – Product quality – Part 1: Quality model*. 2001. – https://www.iso.org/standard/22749.html

[12] Jähne, Bernd: *Digitale Bildverarbeitung*. 5. Berlin : Springer, 2002. – ISBN 3-540-41260-3

[13] Kaelbling, Leslie P. ; Littman, Michael L. ; Moore, Andrew W.: Reinforcement Learning: A Survey. In: *Journal of Artificial Intelligence Research* (1996)

[14] Kober, Jens ; Bagnell, J A. ; Peters, Jan: Reinforcement learning in robotics: A survey. In: *The International Journal of Robotics Research* 32 (2013), Nr. 11, S. 1238–1274

[15] Lassahn, Sandra V.: *3D-Simulation und prototypischer Aufbau eines durch Reinforcement Learning gesteuerten Labyrinths*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2024

[16] Metzen, Jan H. ; Kirchner, Elsa ; Abdenebaoui, Larbi ; Kirchner, Frank: The Brio Labyrinth Game – A Testbed for Reinforcement Learning and for Studies on Sensorimotor Learning. In: *Proceedings of the Conference of the German Research Center for Artificial Intelligence (DFKI)*, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 2009. – Conference Paper

[17] Patel, Ankur A.: *Hands-On Unsupervised Learning Using Python*. O'Reilly Media, Inc., 2019

[18] Reitmaier, Tobias: *Aktives Lernen für Klassifikationsprobleme unter der Nutzung von Strukturinformationen*. Kassel : kassel university press, 2015

[19] Sablatnig, Robert ; Zambanini, Sebastian: *Einführung in Visual Computing – Kapitel 7: Point Operations*. Technische Universität Wien, 2014. – Version 14, Kapitel 7

[20] Sablatnig, Robert ; Zambanini, Sebastian: CV11: Morphologische Operationen und Bildsegmentierung / Technische Universität Wien. 2021. – Vorlesungsskript. Einführung in Visual Computing – CV-11, Vs.1.9

[21] Salvi, Massimo ; Acharya, U R. ; Molinari, Filippo ; Meiburger, Kristen M.: The impact of pre-and post-image processing techniques on deep learning frameworks: A comprehensive review for digital pathology image analysis. In: *Computers in Biology and Medicine* 128 (2021), S. 104129

[22] Zai, Alex ; Brown, Brandon: *Einstieg in Deep Reinforcement Learning: KI-Agenten mit Python und PyTorch programmieren*. Carl Hanser Verlag, 2020. – ISBN 9783446466081

[23] Zhang, Zhengyou: A flexible new technique for camera calibration. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), Nr. 11, S. 1330–1334

Erklärung zur selbständigen Bearbeitung

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

Ort	Datum	Unterschrift im Original