DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Realisierung eines Vier-Gewinnt Roboters

ggf. Untertitel mit ergänzenden Hinweisen

Studienarbeit T3_3200

Studiengang Elektrotechnik

Studienrichtung Automation

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Simon Gschell / Patrik Peters

Abgabedatum:	7. Juli 2025
Bearbeitungszeitraum:	01.01.2025 - 31.06.2025
Matrikelnummer:	123 456
Kurs:	TEA22
Betreuerin / Betreuer:	Prof. Dr. ing Thorsten Kever

Erklärung

gemäß Ziffer 1.1.14 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 24.07.2023.

Ich versichere hiermit, dass ich meine Studienarbeit T3_3200 mit dem Thema:

Realisierung eines Vier-Gewinnt Roboters

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Friedrichshafen, den 7. Juli 2025

Simon Gschell / Patrik Peters

Kurzfassung

Diese Studienarbeit wurde im Rahmen der sechsten Akademiephase angefertigt. Im ersten Teil der Arbeit wurden verschiedene Spieltheorien miteinander verglichen. Aufbauend auf den Erkenntnissen der ersten Studienarbeit, lag der Schwerpunkt diesmal in der praktischen Umsetzung eines Vier-Gewinnt-Roboters. Das Ziel dieser Arbeit bestand darin, einen Roboter zu entwickeln, der eigenständig Spielzüge beim Spiel „Vier gewinnt“ ausführen kann und somit in der Lage ist, gegen einen anderen Roboter anzutreten.

Für die Realisierung des Projekts wurde ein LEGO Spike Prime Set verwendet. Die Konstruktion des Roboters besteht aus verschiedenen zusammengesetzten LEGO-Bauelementen. Vereinzelt wurden auch Elemente selbst entworfen und mittels 3D-Drucker angefertigt. Die Steuerung und Überwachung der Aktoren, wie zum Beispiel der Motoren und Sensoren, erfolgt mithilfe eines LEGO Spike Hub. Dieser Mikrocontroller verarbeitet die eingehenden Sensordaten und steuert die Bewegungen des Roboters entsprechend der programmierten Logik.

Die Programmierung erfolgte in der Sprache MircoPython in der LEGO Spike App. Das Kernstück des Programms ist der Alpha-Beta-Algorithmus, mit ihm wird der nächste optimale Zug berechnet. Durch die Kombination aus mechanischer Konstruktion und programmierter Software entstand ein funktionsfähiger Prototyp, der die gestellten Anforderungen erfüllt und einen Spielzug eigenständig ausführen kann.

Abstract

This student research project was carried out during the sixth academy phase. In the first part of the project, various game theories were examined and compared. Building on the insights from that initial work, the focus this time was on the practical development of a Four-in-a-Row robot. The goal was to create a robot capable of making its own moves in the game "Connect Four,älowing it to compete against another robot.

The project was implemented using a LEGO Spike Prime set. The robot itself was built from a range of LEGO components, with some parts specially designed and produced using a 3D printer. The motors and sensors are managed by a LEGO Spike Hub, which acts as the robot's microcontroller. This hub processes sensor data and directs the robot's movements according to the programmed logic.

Programming was done in MicroPython using the LEGO Spike app. At the heart of the software is the alpha-beta algorithm, which determines the best possible move at each turn. By combining mechanical design with custom software, the project resulted in a working prototype that meets the requirements and can play the game autonomously.

Inhaltsverzeichnis

1 Einleitung

2 Grundlagen

In diesem Kapitel werden die zentralen Begriffe und Methoden ausführlich vorgestellt. So wird das notwendige Grundlagenwissen vermittelt, auf dem die weitere Ausarbeitung aufbaut.

2.1 Vier-Gewinnt

Das Spiel Vier-Gewinnt wird auf einem Spielfeldraster mit sechs Zeilen und sieben Spalten gespielt. Zum Spielbeginn erhält jeder Spieler 21 Spielchips, entweder Rote oder Gelbe. Ziel des Spiels ist es, möglichst schnell vier Chips der eigenen Farbe in eine Reihe zu bringen – waagrecht, senkrecht oder diagonal. Die Spieler werfen abwechselnd ihre Chips in das Spielfeld, bis entweder ein Spieler das Ziel erreicht oder alle 42 Felder belegt sind [Has20].

2.2 Alpha-Beta-Pruning-Algorithmus

Alpha-Beta-Pruning ist ein Verfahren, das bei Spielen wie Schach, Dame oder Vier Gewinnt eingesetzt wird, um den optimalen nächsten Zug zu bestimmen. Ziel des Algorithmus ist es, die Suche im Spielbaum effizienter zu machen. Im Unterschied zum Minimax-Algorithmus werden beim Alpha-Beta-Pruning gezielt Teilbäume weggelassen, die für das Endergebnis keine Rolle spielen. Während der Tiefensuche durch

den Spielbaum arbeitet der Algorithmus mit zwei Schrankenwerten, dem Alpha (α) und dem Beta (β). Zu Beginn wird Alpha auf $-\infty$ und Beta auf $+\infty$ gesetzt. An jedem MAX-Knoten wird das Maximum aus dem bisherigen Alpha und den Werten der Nachfolgeknoten ausgewählt. Das bedeutet, Alpha wird immer dann erhöht, wenn ein nachfolgender Knoten einen höheren Wert liefert als das aktuelle Alpha. Am MIN-Knoten hingegen wird Beta jeweils auf das Minimum aus dem bisherigen Beta und den Werten der Nachfolgeknoten gesetzt. Sobald an einem Knoten die Bedingung $\alpha \geq \beta$ erfüllt ist, wird der restliche Teilbaum nicht weiter betrachtet. In diesem Fall hat der MIN bereits eine bessere Alternative gefunden, sodass MAX diesen Zweig des Baums nicht mehr wählen würde [Ado09].

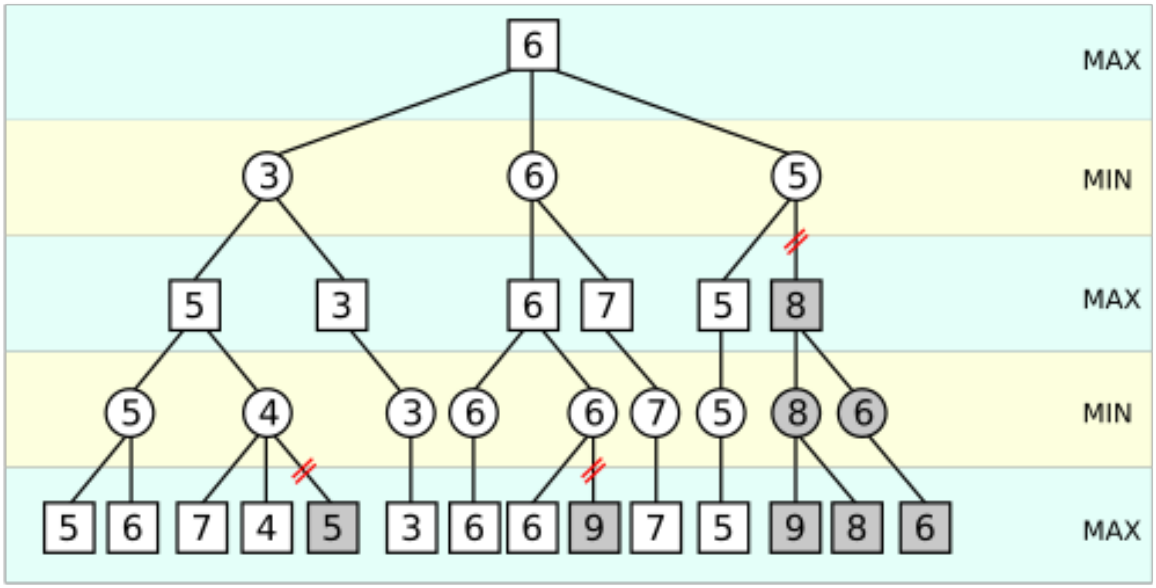

Abbildung 2.1: Alpha-Beta Spielbaum

2.3 Mikropython

MicroPython ist eine speziell für Mikrocontroller angepasste Version der Programmiersprache Python. Im Gegensatz zur Desktop-Variante lässt sich MicroPython-Code direkt auf Hardware mit begrenzten Ressourcen ausführen[SS23][PLJ23]. Im Unterschied zu Standard-Python 3 bringt MicroPython allerdings nur einen Teil der

gewohnten Python-Standardbibliotheken mit.
Wodurch es weniger Speicherplatz benötigt. Zudem besitzt MicroPython einen eigenen Interpreter, der direkt auf einem Mikrokontroller ausgeführt werden kann. Dadurch eignet sich MicroPython besonders gut für die Programmierung des LEGO Spike Hubs[Bel24].

2.4 LEGO Spike Hub:

Der LEGO Spike Hub ist das Herzstück des LEGO Spike Prime Sets. Er dient als programmierbare Steuereinheit mit sechs LPF2 input/output ports, an die alle LEGO-Sensoren und -Motoren angeschlossen werden können. Im Inneren arbeitet ein eigener Prozessor (100 MHz ARM Cortex-M4), unterstützt von 320 KB RAM und 1 MB Flash-Speicher. Die Programmierung des LEGO Spike Hubs erfolgt in der Sprache MicroPython. LEGO stellt dafür eine eigene Entwicklungsumgebung (IDE) bereit, mit dieser der Hub einfach programmiert werden kann. Hierfür kann der Hub über USB oder via Bluetooth mit dem Computer verbunden werden. Die Steuereinheit wird über einen wiederaufladbarer Lithium-Ionen-Akku mit Strom und Spannung versorgt [LEG20b].

Weitere technische Merkmale des LEGO Spike Hubs sind:

- Individuell anpassbaren Lichtmatrix (5x5)
- Lautsprecher
- Taster mit integrierter Statusleuchte
- Tasten für die Navigation und Steuerung durch Menüs am Hub
- Lautsprecher

- sechsachsiger Gyrosensor

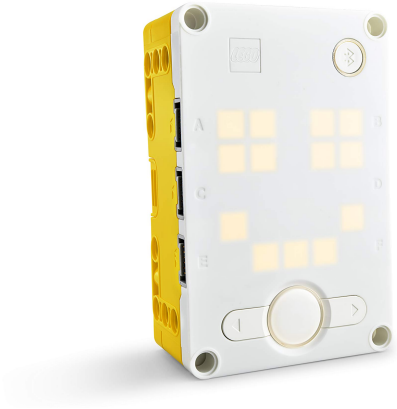

Abbildung 2.2: LEGO Spike Hub

2.5 Sensorik

2.5.1 LEGO Spike Kraft- oder Tuchsensor

Dieser Sensor erkennt, ob er gedrückt wurde, und misst dabei gleichzeitig die auf ihn ausgeübte Kraft. Mit einer Abtastrate von 100 Hz erfasst er Kräfte im Bereich von 2,5 bis 10 Newton und arbeitet dabei mit einer Genauigkeit von $\pm 0,65$ Newton. Der gemessene Wert wird als Prozentwert ausgegeben, wobei 100% einem Tastendruck von 10 Newton entsprechen. Typischerweise wird der Sensor zum Erkennen von Hindernissen oder als Start- bzw. Stopptaste eines Roboters eingesetzt. Der Kraft- oder Tuchsensor wird direkt am LEGO Spike Hub angeschlossen [LEG20c].

Abbildung 2.3: LEGO Spike Kraft- oder Tuchsensor

2.5.2 LEGO Spike Farbsensor

Dieser elektronische Farbsensor wurde speziell für LEGO Spike entwickelt. Seine Abtastrate beträgt 1 kHz und er kann direkt am Hub angeschlossen werden. Der Sensor kann bis zu acht verschiedene Farben erkennen, darunter Schwarz, Blau, Rot, Weiß, Braun, Gelb, Pink und Grün. Außerdem misst er sowohl die Intensität des reflektierten Lichts als auch die des Umgebungslichts [LEG20a][LEG20c].

Für die Farberkennung erfasst der Sensor die Farbwerte sowohl im RGB- (Rot, Grün, Blau) als auch im HSV-Farbraum (hue = Farbton, saturation = Sättigung, value = Helligkeit). Die Messergebnisse werden als Ganzzahlen ausgegeben [LEG20a].

Zur Reflexionsmessung sendet der Sensor weißes Licht auf eine Oberfläche und misst das zurückgeworfene Licht. Diese Funktion wird häufig für Linienführung eingesetzt [Bet25][LEG20a].

Abbildung 2.4: LEGO Technic Farbsensor

2.6 Aktorik

2.6.1 LEGO Spike Winkelmotor

Der LEGO Spike Winkelmotor nicht nur ein einfacher Elektromotor, aufgrund eines integrierten Drehsensors kann er nicht nur die Drehrichtung, sondern auch die relative und absolute Position (in Grad) sowie die Drehgeschwindigkeit erfassen. Eine vollständige Umdrehung wird dabei in 360 einzelne Zählimpulse unterteilt, wobei die Genauigkeit des Motors bei ±3 Grad liegt. Muss der Motor ein Drehmoment von mehr als 5 Ncm aufbringen, blockiert er. Mit einer Abtastrate von 100 Hz erfasst der Motor die Position sowohl beim automatischen als auch im manuellen Betrieb. Der LEGO Spike Winkelmotor eignet sich somit nicht nur für Bewegungsaufgaben, sondern auch zur Positionsbestimmung [LEG20c].

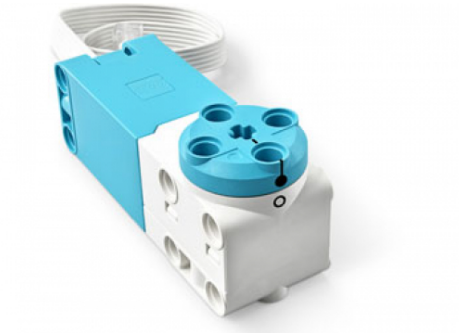

Abbildung 2.5: LEGO Spike Winkelmotor

3 Vorgehen

Im folgendem Kapitel wird auf die Planung des Vier-Gewinnt-Roboters eingegangen. Zunächst werden die Anforderungen definiert. Anschließend werden verschiedene Konzepte genauer betrachtet und miteinander verglichen.

3.1 Aufgabenpräzisierung

3.2 Anforderungen an den Roboter

In der Tabelle 3.1 sind die Anforderungen an den Vier-Gewinnt-Roboter in einer Anforderungsliste zusammengetragen. Dabei wird zwischen Forderungen und Wünschen unterschieden. Anforderungen, die mit ***F*** gekennzeichnet sind, müssen unbedingt umgesetzt werden. Während hingegen Anforderungen, die mit ***W*** markiert sind, als Wünsche zu verstehen sind und nicht zwingend im System realisiert werden müssen.

Tabelle 3.1: Anforderungstabelle für einen Vier-Gewinnt-Roboter

Nr	**Anforderung an das System**	**F/W**
-	Komplettes Spielfeld scannen	F
-	Das Ende des Spiels erkennen	F
-	Autonom fahren	F
-	Begrezungen des Spielfeld erkennen	F
-	Steine selber platzieren	F
-	Immer nur ein Stein pro Spielzug	F
-	Abwarten bis der Gegner sein Zug beendet hat	F
-	Nach jeden Zug rechts vom Spielfeld wegfahren	F
-	maximal 90 Sekunden pro Spielzug	F
-	optimalen Spielzug berechnen	F
-	Scannen bis ein neuer gegnerischer Stein erkannt wurde	W
-	Volle Spalten überspringen	W
-	Wenn eine leere Zeile erkannt wurde zur nächsten Spalte wechseln	W
-		F

Legende: F = Forderung, **W** = Wunsch,

4 Umsetzung und Ergebnisse

4.1 Mechanischer Aufbau des LEGO-Spike-4-Gewinnt-Roboters

Für die Umsetzung des 4-Gewinnt-Roboters wurde eine mechanische Konstruktion gewählt, die es erlaubt, das Spielfeld zu scannen sowie Chips gezielt in eine Spalte einzuwerfen. Der Aufbau umfasst drei LEGO Spike-Motoren, einen Farbsensor und einen Drucktaster. Im Folgenden werden die einzelnen Komponenten detailliert beschrieben.

- **Horizontalantrieb – Motor D**
  Der horizontale Antrieb des Farbsensors erfolgt über Motor D. Dieser ist dafür zuständig, die Spielfeldspalten nacheinander anzufahren. Der Motor ist mit einer Achse verbunden, welche zwei Räder antreiben. Die Bewegung erfolgt in gleichmäßigen Schritten: Eine Drehung um exakt 72 Grad bewegt den Schlitten um eine Spalte weiter. Diese Schrittweite wurde so gewählt, dass sie der Breite einer Spalte im Spielfeld entspricht. Dadurch ist eine exakte Positionierung des Sensors über jeder Spalte möglich, ohne dass zusätzliche Sensoren zur Positionsbestimmung notwendig sind.

- **Vertikalantrieb – Motor E, Kette und Farbsensor**
  Um das Spielfeld auch in vertikaler Richtung abfahren zu können, ist der Farbsensor an einer Kette montiert. Diese Kette wird durch Motor E angetrieben. Der Sensor ist an einem mittleren Segment der Kette befestigt und fährt beim

Drehen der Kette entsprechend auf und ab. Ein Schritt des Motors um etwa 95 Grad bewegt den Sensor um genau eine Spielfeldhöhe weiter. Auf diese Weise können sämtliche sechs Reihen der aktuellen Spalte nacheinander abgescannt werden. Der Sensor wurde dabei so montiert, dass er exakt über der Mitte jedes Feldes positioniert ist, um eine zuverlässige Farberkennung zu ermöglichen. Die Rückwärtsbewegung der Kette erlaubt es, den Sensor wieder nach unten zu fahren.

- **Chipauswerfer – Motor A**
  Das Einwerfen des eigenen Spielsteins erfolgt über Motor A. An diesem Motor ist eine Stange montiert, die bei einer vollständigen Umdrehung einen Spielchip aus dem Vorratsmagazin in die gewünschte Spalte stößt. Nach der Auslösung kann ein neuer Chip in die Abschussposition nachrutschen. In der Software ist eine Wartezeit nach dem Auslösen eingebaut, damit der Chip sicher im Spielfeld ankommt, bevor die nächste Aktion beginnt.

- **Startsignal – Drucksensor (Force Sensor)**
  Um dem Roboter mitzuteilen, dass der menschliche Spieler seinen Zug abgeschlossen hat, wurde ein Drucksensor verwendet. Dieser befindet sich an der Vorderseite des Roboters. Sobald der Spieler den Sensor leicht berührt, wird ein Signal ausgelöst, das Prozess startet.

- **Spielfeldscan – Farbsensor an Kette**
  Für die Farberkennung des Spielfeldes wurde ein LEGO-Farbsensor verwendet, der über die oben beschriebene Kettenkonstruktion vertikal verfahrbar ist. Die Farbmessung erfolgt jeweils in der Mitte eines Spielfeldes. Der Sensor erkennt RGB-Werte, die per Software verschiedenen Spielsteinfarben (in diesem Projekt benutzt: Rot, Gelb oder Leer) zugeordnet werden. Während des Spiels vergleicht der Algorithmus die gemessenen Werte mit diesen Referenzwerten, um die tatsächliche Belegung jedes Feldes möglichst robust zu bestimmen. Der Abstand zwischen Sensor und Spielfeld beträgt etwa 7 mm – dieser Wert hat sich als optimal für zuverlässige Farbmessung erwiesen.

Zusammenfassung

Die mechanische Konstruktion basiert auf einem kartesischen Koordinatensystem, bei dem der Sensor durch die Kombination aus horizontaler (Motor D) und vertikaler Bewegung (Motor E + Kette) jede Spielfeldposition präzise anfahren kann. Das System erlaubt eine vollautomatische Spielweise: Der Roboter erkennt die aktuelle Spielsituation, berechnet den optimalen Zug und setzt diesen physisch um.

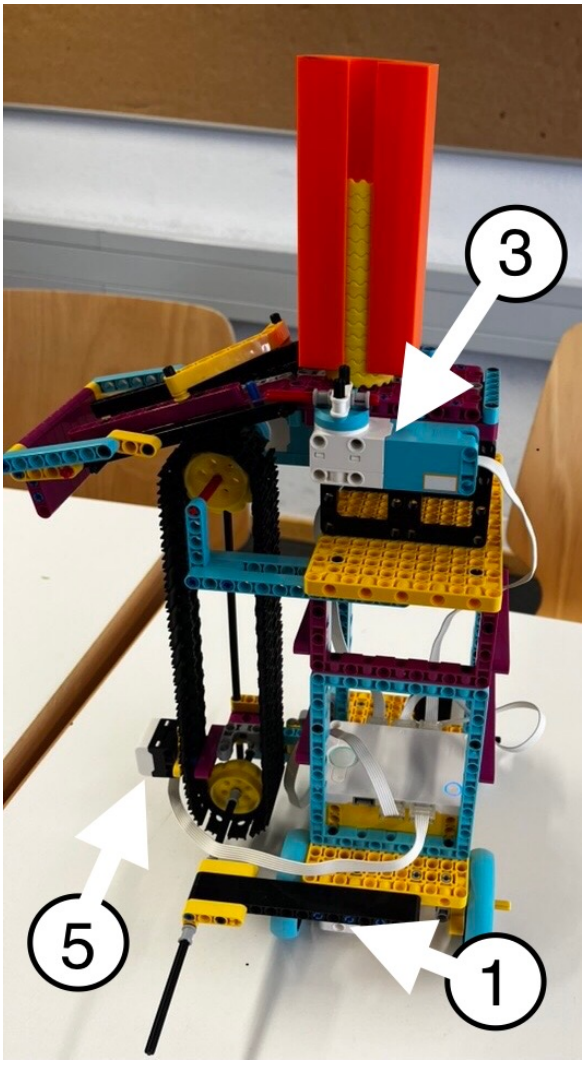

Abbildung 4.1: Seitenansicht links

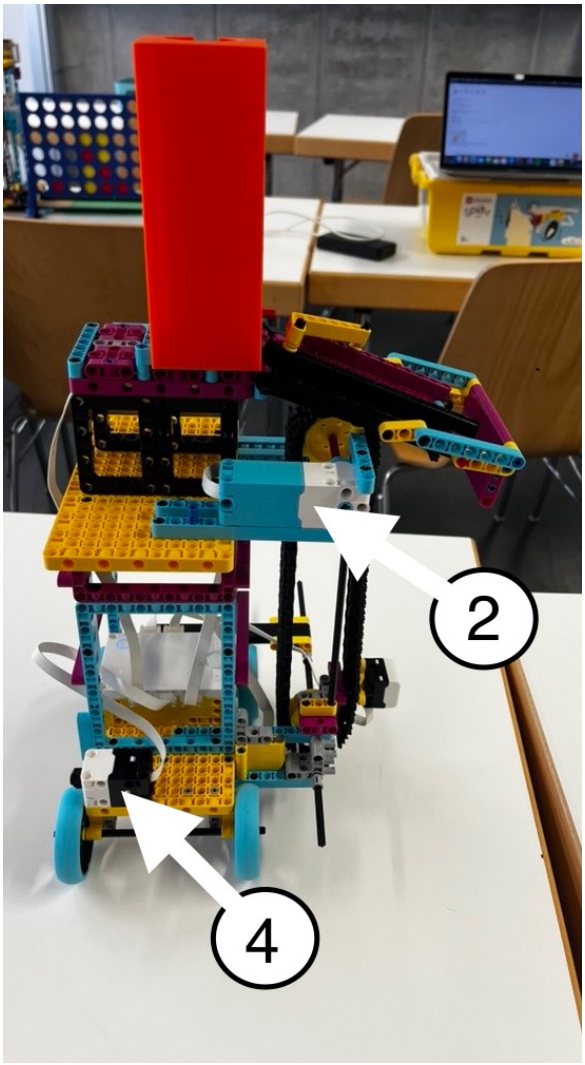

Abbildung 4.2: Seitenansicht rechts

4.2 Software

Im Kapitel Software wird genauer beschrieben, wie die Programmierung des Vier-Gewinnt-Roboters umgesetzt ist. Die Software bildet das Kernstück des Roboters und ist entscheidend dafür, dass dieser eigenständig am Spiel teilnehmen kann.

```
1 import motor
```

```
import color_sensor
import color
import time
import force_sensor
from hub import port, sound
```

```
# ── CONFIGURATION ──
my_piece = 1 # -1 = RED, 1 = YELLOW (AI player)
opponent_piece = -my_piece

speed_D = 230
speed_A = 500
speed_E = 600
move_distance_e = 97 # Step motor E per row, calibrate if necessary!
move_distance_d = 73
break_motor_e = 1
break_motor_e_back = 2
break_motor_d = 1
field_width = 7
field_height = 6
waiting_line = 1 # Waiting time after reaching the line (in seconds)
```

```
# ── HELPER FUNCTIONS ──
def sensor_activated():
    try:
        return force_sensor.force(port.C) > 30
    except:
        return False

def update_board(row, col, detected_color):
    if detected_color in (color.RED, color.PURPLE, color.MAGENTA):
        board[row][col] = -1
    elif detected_color in (color.YELLOW, color.WHITE, color.GREEN):
        board[row][col] = 1
    else:
        board[row][col] = 0

def print_board(board):
```

```
17      symbol_map = {-1: "R", 1: "Y", 0: "B", None: "B"}
18      print("\nCurrent board (bottom right = [0][0]):")
19      for row_idx in reversed(range(field_height)):
20        row = board[row_idx]
21        print(" ".join(symbol_map.get(cell, "?") for cell in row))
22
23    def is_valid_location(board, col):
24      return board[field_height-1][col] == 0
25
26    def get_next_open_row(board, col):
27      for r in range(field_height):
28        if board[r][col] == 0:
29          return r
30        return None
31
32    def winning_move(board, piece):
33      for c in range(field_width-3):
34        for r in range(field_height):
35          if board[r][c] == piece and board[r][c+1] == piece and board[r][
       c+2] == piece and board[r][c+3] == piece:
36            return True
37      for c in range(field_width):
38        for r in range(field_height-3):
39          if board[r][c] == piece and board[r+1][c] == piece and board[r
       +2][c] == piece and board[r+3][c] == piece:
40            return True
41      for c in range(field_width-3):
42        for r in range(field_height-3):
43          if board[r][c] == piece and board[r+1][c+1] == piece and board[r
       +2][c+2] == piece and board[r+3][c+3] == piece:
44          return True
45        for r in range(3, field_height):
46          if board[r][c] == piece and board[r-1][c+1] == piece and board[r
       -2][c+2] == piece and board[r-3][c+3] == piece:
47        return True
48    return False
49
50    def evaluate_window(window, player):
51      opp_player = opponent_piece if player == my_piece else my_piece
52      score = 0
```

```
53      if window.count(player) == 4:
54          score += 100
55      elif window.count(player) == 3 and window.count(0) == 1:
56          score += 5
57      elif window.count(player) == 2 and window.count(0) == 2:
58          score += 2
59      if window.count(opp_player) == 3 and window.count(0) == 1:
60          score -= 4
61      return score
62
63  def evaluate(board):
64      score = 0
65      center_array = [board[r][field_width//2] for r in range(field_height
        )]
66      center_count = center_array.count(my_piece)
67      score += center_count * 3
68      for r in range(field_height):
69          row_array = [board[r][c] for c in range(field_width)]
70          for c in range(field_width - 3):
71              window = row_array[c:c+4]
72              score += evaluate_window(window, my_piece)
73              score -= evaluate_window(window, opponent_piece)
74      for c in range(field_width):
75          col_array = [board[r][c] for r in range(field_height)]
76          for r in range(field_height - 3):
77              window = col_array[r:r+4]
78              score += evaluate_window(window, my_piece)
79              score -= evaluate_window(window, opponent_piece)
80      for r in range(field_height - 3):
81          for c in range(field_width - 3):
82              window = [board[r+i][c+i] for i in range(4)]
83              score += evaluate_window(window, my_piece)
84              score -= evaluate_window(window, opponent_piece)
85      for r in range(3, field_height):
86          for c in range(field_width - 3):
87              window = [board[r-i][c+i] for i in range(4)]
88              score += evaluate_window(window, my_piece)
89              score -= evaluate_window(window, opponent_piece)
90      return score
91
```

```
92  transposition_table = {}
93
94  def board_hash(board, maximizing_player):
95    return (tuple([item for row in board for item in row]),
    maximizing_player)
96
97  def get_dynamic_depth(board):
98    # Count empty fields
99    empty = sum(row.count(0) for row in board)
100   if empty > 30:
101     return 3 # Beginning: fast, low depth
102   else:
103     return 4 # End: high depth for best moves
104
105 def alpha_beta(board, depth, alpha, beta, maximizing_player):
106   key = (board_hash(board, maximizing_player), depth)
107   if key in transposition_table:
108     return transposition_table[key]
109   valid_locations = [col for col in range(field_width) if
    is_valid_location(board, col)]
110   valid_locations.sort(key=lambda c: abs(c - field_width // 2))
111   terminal = winning_move(board, my_piece) or winning_move(board,
    opponent_piece) or len(valid_locations) == 0
112   if depth == 0 or terminal:
113     if terminal:
114       if winning_move(board, my_piece): return (None, 1000000)
115       elif winning_move(board, opponent_piece): return (None,
    -1000000)
116       else: return (None, 0)
117     else:
118     return (None, evaluate(board))
119   if maximizing_player:
120     value = -float('inf')
121     best_col = valid_locations[0]
122     for col in valid_locations:
123       row = get_next_open_row(board, col)
124       if row is None:
125         continue
126       board_copy = [r[:] for r in board]
127       board_copy[row][col] = my_piece
```

```
128         new_score = alpha_beta(board_copy, depth-1, alpha, beta, False)
    [1]
129         if new_score > value:
130           value = new_score
131           best_col = col
132         alpha = max(alpha, value)
133         if alpha >= beta:
134           break
135       result = (best_col, value)
136     else:
137       value = float('inf')
138       best_col = valid_locations[0]
139       for col in valid_locations:
140         row = get_next_open_row(board, col)
141         if row is None:
142           continue
143         board_copy = [r[:] for r in board]
144         board_copy[row][col] = opponent_piece
145         new_score = alpha_beta(board_copy, depth-1, alpha, beta, True)
    [1]
146         if new_score < value:
147           value = new_score
148           best_col = col
149         beta = min(beta, value)
150         if beta <= alpha:
151           break
152       result = (best_col, value)
153     transposition_table[key] = result
154     return result
```

```
1  # ---- MAIN PROGRAM ----
2  board = [[0 for _ in range(field_width)] for _ in range(field_height)]
3  last_board = [row[:] for row in board]
4
5  def move_motor_e_to_zero():
6    pass
7
8  while True:
9    print("Waiting for sensor at port C...")
```

```
10      while not sensor_activated():
11          time.sleep(0.1)
12      print("Sensor detected! Starting move...")
13      time.sleep(1)
14
15      # Move to field
16      motor.run_for_degrees(port.D, 198, 170)
17      print("Field reached...")
18      time.sleep(1.5)
19
20      transposition_table.clear()
21
22      opponent_piece_found = False
23      opponent_col = None
24
25      move_motor_e_to_zero()
26
27      for col in range(field_width - 1, -1, -1):
28          matrix_col = field_width - 1 - col
29          if last_board[field_height-1][matrix_col] != 0:
30              if col > 0:
31                  motor.run_for_degrees(port.D, move_distance_d, 170)
32                  time.sleep(break_motor_d)
33              continue
34
35      free_row = None
36      for row in range(field_height):
37          if last_board[row][matrix_col] == 0:
38              free_row = row
39              break
40      if free_row is None:
41          continue
42
43      motor.run_for_degrees(port.E, move_distance_e * (free_row), speed_E)
44      time.sleep(break_motor_e)
45      time.sleep(waiting_line)
46
47      detected_color = color_sensor.color(port.B)
48      update_board(free_row, matrix_col, detected_color)
49      print("Matrix entry: Row {}, Col {}: {}".format(
```

```
50          free_row, matrix_col,
51          "RED" if board[free_row][matrix_col] == -1
52          else "YELLOW" if board[free_row][matrix_col] == 1
53          else "NONE"))
54
55      if (last_board[free_row][matrix_col] == 0 and
56          board[free_row][matrix_col] == opponent_piece):
57          print("New opponent piece in column {}, row {}".format(matrix_col,
        free_row))
58          opponent_piece_found = True
59          opponent_col = col
60          time.sleep(1)
61
62      motor.run_for_degrees(port.E, -move_distance_e * free_row, speed_E)
63      time.sleep(break_motor_e_back)
64
65      if opponent_piece_found:
66          break
67
68      if col > 0:
69          motor.run_for_degrees(port.D, move_distance_d, speed_D)
70          time.sleep(break_motor_d)
71
72      if opponent_piece_found and opponent_col is not None and
        opponent_col != 0:
73          steps_right = opponent_col
74          motor.run_for_degrees(port.D, move_distance_d * steps_right,
        speed_D)
75          time.sleep(4)
76
77      board_numeric = [[0 if x is None else x for x in row] for row in
        board]
78      dynamic_depth = get_dynamic_depth(board_numeric)
79      best_col, _ = alpha_beta(
80      board_numeric,
81      depth=dynamic_depth,
82      alpha=-float('inf'),
83      beta=float('inf'),
84      maximizing_player=(my_piece == 1)
85      )
```

```
86
87      best_row = get_next_open_row(board_numeric, best_col)
88      color_str = "RED" if my_piece == -1 else "YELLOW"
89      print("\nOptimal position for {}:".format(color_str))
90      print("Column: {}, Row: {}".format(best_col, best_row))
91
92      if best_col is not None and best_row is not None:
93          physical_target_col = field_width - 1 - best_col
94          motor.run_for_degrees(port.D, -move_distance_d *
    physical_target_col, speed_D)
95          time.sleep(break_motor_d * 4)
96          motor.run_for_degrees(port.A, -360, speed_A)
97          time.sleep(3)
98          board[best_row][best_col] = my_piece
99          last_board = [row[:] for row in board]
100
101         if winning_move(board, my_piece):
102             print(" Congratulations! The robot has WON the game!")
103             print_board(board)
104             sound.beep(440, 1000000, 100)
105             motor.run_for_degrees(port.D, -move_distance_d * best_col,
    speed_D)
106             time.sleep_ms(2500)
107             motor.run_for_degrees(port.D, -move_distance_d * 3, speed_D)
108             sound.beep(0, 1000000, 100)
109             break
110
111         motor.run_for_degrees(port.D, -move_distance_d * best_col, speed_D
    )
112
113     print_board(board)
114     time.sleep_ms(2500)
115     motor.run_for_degrees(port.D, -199, speed_D)
116
117     time.sleep(break_motor_d)
118
119     while sensor_activated():
120         time.sleep(0.1)
121     print("Move completed. Waiting for next activation...")
```

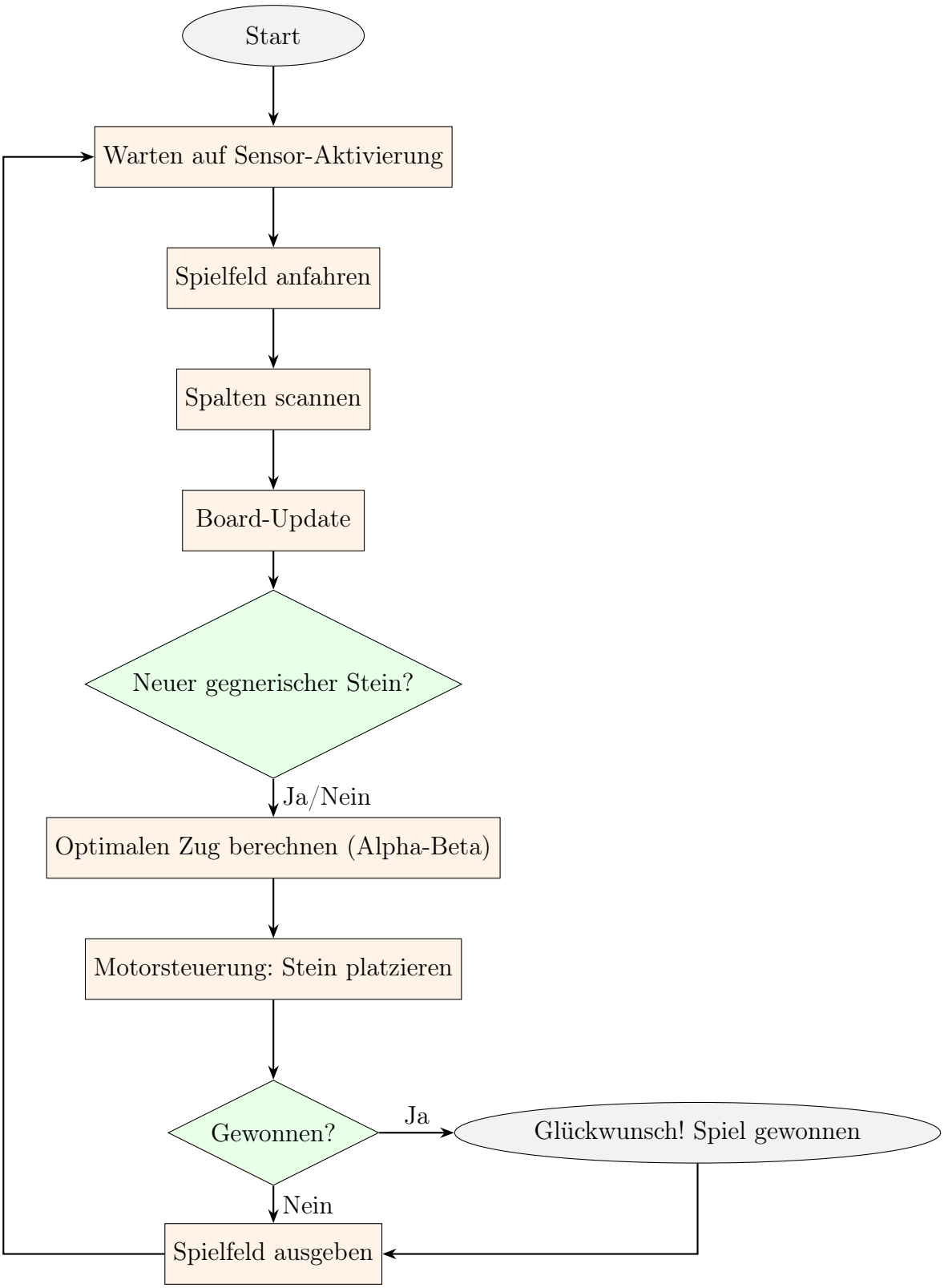

Abbildung 4.3: Flussdiagramm der Software

5 Zusammenfassung

Algorithmus zur Entscheidungsfindung

Ein zentraler Bestandteil des 4-Gewinnt-Roboters ist die Entscheidungsfindung durch einen algorithmischen Spielbaum. Dieser wird mit dem bekannten Minimax-Algorithmus unter Verwendung von Alpha-Beta-Pruning realisiert. Ziel ist es, basierend auf dem aktuellen Spielfeldzustand den optimalen Zug für die KI zu berechnen. Der Algorithmus bewertet mögliche Züge bis zu einer bestimmten Tiefe im Spielbaum und trifft Entscheidungen, die langfristig zum Sieg führen können oder gegnerische Gewinnzüge verhindern.

Spielfeld und Spielerdefinition

Im Vorfeld des Algorithmus ist festgelegt, welches Symbol der Algorithmus (die KI) spielt:

```
1         my_piece = 1  # 1 = YELLOW (AI player), -1 = RED
2         opponent_piece = -my_piece
3
```

Dabei entspricht 1 dem gelben Spielstein (der KI), -1 dem roten Spielstein (dem Gegner). Diese numerische Darstellung vereinfacht die Bewertung und das Vergleichen

der Felder im Spielfeld.

Bewertungsfunktion (evaluate)

Der Algorithmus benötigt eine Bewertungsfunktion, die die Qualität eines Spielzustands abschätzt. Dies geschieht durch eine Heuristik, die mögliche Gewinnlinien zählt und bewertet.

Die Bewertungsfunktion basiert auf der Idee, sogenannte „Fenster“ (Ausschnitte aus 4 Feldern) im Spiel zu analysieren und zu beurteilen, wie viele Steine der KI bzw. des Gegners in diesen Fenstern enthalten sind:

```
1         def evaluate_window(window, player):
2         opp_player = opponent_piece if player == my_piece else
      my_piece
3         score = 0
4         if window.count(player) == 4:
5         score += 100
6         elif window.count(player) == 3 and window.count(0) == 1:
7         score += 5
8         elif window.count(player) == 2 and window.count(0) == 2:
9         score += 2
10        if window.count(opp_player) == 3 and window.count(0) ==
      1:
11        score -= 4
12        return score
13
```

Diese Funktion bewertet sowohl offensive als auch defensive Situationen. Ein Fenster mit drei eigenen Steinen und einem leeren Feld wird positiv bewertet, ein Fenster mit drei gegnerischen Steinen und einem leeren Feld hingegen negativ, um Bedrohungen abzuwehren.

Die Hauptfunktion zur Bewertung des gesamten Spielfeldes aggregiert alle horizontalen, vertikalen und diagonalen Fenster:

```
        def evaluate(board):
        score = 0
        center_array = [board[r][field_width//2] for r in range(
    field_height)]
        center_count = center_array.count(my_piece)
        score += center_count * 3

```

Zunächst werden die mittleren Spalten stärker gewichtet, da sie strategisch wichtiger sind (von dort aus können mehr Gewinnlinien entstehen).

Anschließend werden alle Zeilen, Spalten und Diagonalen analysiert:

```
        for r in range(field_height):
        row_array = [board[r][c] for c in range(field_width)]
        for c in range(field_width - 3):
        window = row_array[c:c+4]
        score += evaluate_window(window, my_piece)
        score -= evaluate_window(window, opponent_piece)

```

Diese Schleifen bilden das heuristische Fundament für die spätere Entscheidungsfindung.

Minimax mit Alpha-Beta-Pruning

Die Hauptentscheidung trifft der Minimax-Algorithmus. Dabei wird rekursiv der Spielbaum aufgebaut, wobei sich der Algorithmus abwechselnd in die Rolle der KI („maxi-

mizing player“) und des Gegners („minimizing player“) versetzt. Um die Effizienz zu steigern, wird Alpha-Beta-Pruning genutzt. Dabei werden Äste im Spielbaum verworfen, wenn sie nachweislich zu schlechteren Ergebnissen führen.

Der Einstiegspunkt ist:

```
1         def alpha_beta(board, depth, alpha, beta,
  maximizing_player):
2
```

Zuerst wird geprüft, ob der aktuelle Zustand bereits im Transposition Table gespeichert ist – einem Cache zur Vermeidung redundanter Berechnungen:

```
1         key = (board_hash(board, maximizing_player), depth)
2         if key in transposition_table:
3         return transposition_table[key]
4
```

Anschließend erfolgt eine Terminalprüfung: Ist der Zug eine Gewinnsituation, oder wurde die maximale Tiefe erreicht?

```
1         valid_locations = [col for col in range(field_width) if
  is_valid_location(board, col)]
2         terminal = winning_move(board, my_piece) or winning_move
  (board, opponent_piece) or len(valid_locations) == 0
3         if depth == 0 or terminal:
4         ...
5
```

Falls ja, gibt die Funktion eine Bewertung zurück. Andernfalls wird der Spielbaum weiter durchlaufen.

Maximierender Spieler (KI):

```
        if maximizing_player:
        value = -float('inf')
        for col in valid_locations:
        ...
        new_score = alpha_beta(..., False)[1]
        if new_score > value:
        value = new_score
        best_col = col
        alpha = max(alpha, value)
        if alpha >= beta:
        break

```

Hier versucht der Algorithmus, die maximal erreichbare Bewertung zu finden und prüft regelmäßig, ob das aktuelle Ergebnis besser ist als die bisherige beste Option. Wenn `alpha >= beta`, wird der restliche Baum abgeschnitten (Pruning).

Minimierender Spieler (Gegner): Analog erfolgt das Vorgehen für den Gegner:

```
        else:
        value = float('inf')
        for col in valid_locations:
        ...
        new_score = alpha_beta(..., True)[1]
        if new_score < value:
        value = new_score
        best_col = col
        beta = min(beta, value)
        if beta <= alpha:
        break

```

Am Ende wird das Ergebnis in der Transpositionstabelle gespeichert und zurückgegeben:

```
        transposition_table[key] = result
        return result

```

Dynamische Suchtiefe

Je nach Spielphase kann es sinnvoll sein, tiefer oder flacher zu suchen. Zu Beginn reicht eine niedrige Tiefe, da viele Züge möglich sind. In späteren Phasen erhöht sich die Tiefe:

```
        def get_dynamic_depth(board):
        empty = sum(row.count(0) for row in board)
        if empty > 30:
        return 3
        else:
        return 4

```

Diese dynamische Anpassung balanciert Spielstärke und Rechenzeit optimal.

Fazit

Der eingesetzte Minimax-Algorithmus mit Alpha-Beta-Pruning stellt das strategische Herzstück des 4-Gewinnt-Roboters dar. Durch gezielte Bewertung von Spielpositionen, Berücksichtigung gegnerischer Drohungen und dynamische Tiefenanpassung kann der Roboter selbstständig Züge planen, Gefahren abwehren und letztlich siegreich agieren.

Die Verwendung eines Transpositionstables beschleunigt dabei die Entscheidungsfindung, indem bereits analysierte Spielsituationen nicht erneut bewertet werden müssen. Das Ergebnis ist ein hochgradig effektives Entscheidungsverfahren für ein strategisches Spiel wie 4-Gewinnt.

Literaturverzeichnis

[Ado09] Julius Adorf. *Adversariale Suche für optimales Spiel: Der Minimax-Algorithmus und die Alpha-Beta-Suche*. Proseminararbeit. Betreuer: Lars Kunze, Dominik Jain. Abgabetermin: 2. Dezember 2009. München: Technische Universität München, Fakultät für Informatik, Forschungs- und Lehreinheit Informatik IX, 2009. URL: https://www.juliusadorf.com/pub/alphabeta-seminar-paper.pdf.

[Bel24] Charles A. Bell. *MicroPython for the Internet of Things: A Beginner's Guide to Programming with Python on Microcontrollers*. English. 2nd. Berkeley, CA: Apress, 2024. ISBN: 9781484298619.

[Bet25] Betzold GmbH. *LEGO Education SPIKE Technic Farbsensor*. Zugriff am 04.07.2025. 2025. URL: https://www.betzold.de/prod/E_761144/.

[Has20] SA Hasbro. *Das Originale 4Gewinnt Anleitung*. Hasbro Gaming, 2020.

[LEG20a] LEGO Education. *Technical Specifications: Technic Color Sensor*. https://assets.education.lego.com/v3/assets/blt293eea581807678a/blt62a78c227edef070/5f8801b9a302dc0d859a732b/techspecs_technic colorsensor.pdf?locale=en-us. Accessed: 2025-07-04. 2020.

[LEG20b] LEGO Education. *Technical Specifications: Technic Large Hub*. https://assets.education.lego.com/v3/assets/blt293eea581807678a/bltf512a371e82f6420/5f8801baf4f4cf0fa39d2feb/techspecs_technic largehub.pdf?locale=en-us. Zugriff am 03.07.2025. 2020.

[LEG20c] LEGO Education Community. *Exploring SPIKE™ Prime Sensors*. Accessed: 2025-07-04. 2020. URL: https://community.legoeducation.com/blogs/31/220.

[PLJ23] Ignas Plauska, Agnius Liutkevičius und Audronė Janavičiūtė. „Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller“. In: *Electronics* 12.1 (2023). Open Access Article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license, S. 143. DOI: 10.3390/electronics12010143. URL: https://www.mdpi.com/2079-9292/12/1/143.

[SS23] Albin Samefors und Felix Sundman. *Investiating Energy Consumption and Responsiveness of low power modes in MicroPython for STM32WB55*. Bachelor's thesis. 15 hp (first-cycle education). Jönköping, Sweden, Juni 2023. URL: https://www.diva-portal.org/smash/get/diva2:1778426/FULLTEXT01.pdf.

Abbildungsverzeichnis

Tabellenverzeichnis

A Nutzung von Künstliche Intelligenz basierten Werkzeugen

Im Rahmen dieser Arbeit wurden Künstliche Intelligenz (KI) basierte Werkzeuge benutzt. Tabelle 1.1 gibt eine Übersicht über die verwendeten Werkzeuge und den jeweiligen Einsatzzweck.

Tabelle 1.1: Liste der verwendeten KI basierten Werkzeuge

Werkzeug	**Beschreibung der Nutzung**
ChatGPT	• Grundlagenrecherche zu bekannten Prinzipien optischer Sensorik zur Abstandsmessung (siehe Abschnitt ...) • Suche nach Herstellern von Lidar-Sensoren (siehe Abschnitt ...) • ...
ChatPDF	• Recherche und Zusammenfassung von wissenschaftlichen Studien im Themenfeld ... • ...
DeepL	• Übersetzung des Papers von [...]
Tabnine AI coding assistant	• Aktiviertes Plugin in MS Visual Studio zum Programmieren des ... • ...
...	• ...

B Ergänzungen

2.1 Details zu bestimmten theoretischen Grundlagen

2.2 Weitere Details, welche im Hauptteil den Lesefluss behindern

C Details zu Laboraufbauten und Messergebnissen

3.1 Versuchsanordnung

3.2 Liste der verwendeten Messgeräte

3.3 Übersicht der Messergebnisse

3.4 Schaltplan und Bild der Prototypenplatine

D Zusatzinformationen zu verwendeter Software

4.1 Struktogramm des Programmentwurfs

4.2 Wichtige Teile des Quellcodes

E Datenblätter

Auf den folgenden Seiten wird eine Möglichkeit gezeigt, wie aus einem anderen PDF-Dokument komplette Seiten übernommen werden können, z. B. zum Einbindungen von Datenblättern. Der Nachteil dieser Methode besteht darin, dass sämtliche Formateinstellungen (Kopfzeilen, Seitenzahlen, Ränder, etc.) auf diesen Seiten nicht angezeigt werden. Die Methode wird deshalb eher selten gewählt. Immerhin sorgt das Package *„pdfpages“* für eine korrekte Seitenzahleinstellung auf den im Anschluss folgenden „nativen“ LaTeX-Seiten.

Eine bessere Alternative ist, einzelne Seiten mit *„\includegraphics“* einzubinden.

2

3

4