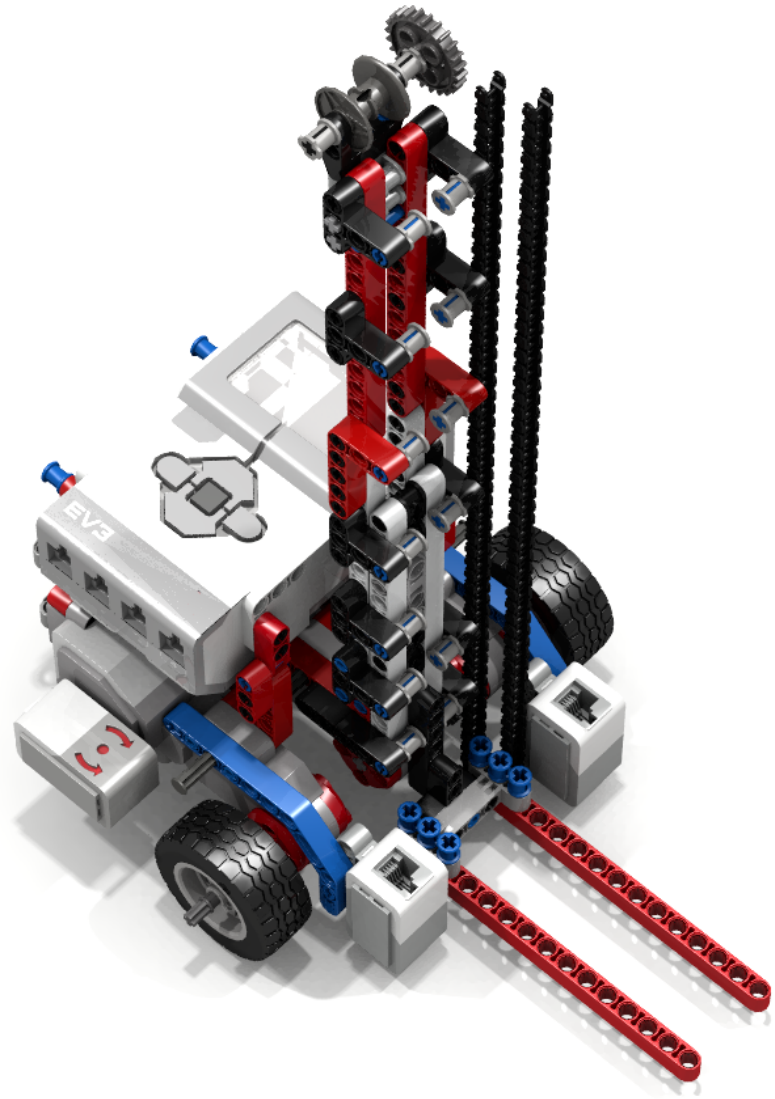
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

EIT FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK

LEGO-Praktikum. entwickeln + programmieren + optimieren

Berichte der Studierenden zum Projektseminar Elektrotechnik/Informationstechnik



Eine Schriftenreihe der Otto-von-Guericke-Universität Magdeburg, Fakultät für Elektrotechnik- und Informationstechnik, Institut für Medizintechnik sowie Institut für Elektrische Energiesysteme

Herausgeben von:
Mathias Magdowski, Thomas Schallschmidt, Jörg Petzold und Marius Klahm

Band 9 vom Wintersemester 2025/2026

Inhaltsverzeichnis

Impressum

Herausgeber:	Mathias Magdowski, Thomas Schallschmidt, Jörg Petzold und Marius Klahm Institut für Medizintechnik, Institut für Elektrische Energiesysteme Fakultät für Elektro- und Informationstechnik Otto-von-Guericke-Universität Magdeburg Postfach 4120, 39016 Magdeburg
DOI:	10.24352/UB.OVGU-2026-048
ISSN:	2629-6160
Redaktionsschluss:	April 2026
Seminarzeitraum:	02. Februar – 16. Februar 2026
Bezug:	Open Access, Digitale Hochschulbibliothek Sachsen-Anhalt `http://edoc2.bibliothek.uni-halle.de/`

1. Auflage, Magdeburg, Otto-von-Guericke-Universität, 2026

Erstellung des Sammelbandes mittels LaTeX, `hyperref` und `pdfpages`

GEOMETER – Der Landvermesser-Roboter

Yannick Christopher Kastens, Elektrotechnik/Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—In diesem Beitrag wird der „Geometer“ vorgestellt, ein differentiell angetriebener LEGO-Mindstorms-EV3-Roboter, der als linienfolgender Vermessungsroboter fungiert. Ziel der Arbeit ist es, dass der Roboter eine dreifarbige Linie selbstständig abfährt, seine Bewegung odometrisch erfasst und die gefahrene Strecke mathematisch rekonstruiert. Dadurch können unterschiedliche Strecken und Flächen genau vermessen werden und Abstände bestimmt werden.**

Die Steuerung erfolgt vorausschauend mithilfe von MATLAB über zwei Farbsensoren, aus denen die notwendigen Fahrmanöver abgeleitet werden. Die Odometrie erfolgt über die Motordrehwinkel, die nach Kalibrierung in reale Strecken- und Winkeldaten umgerechnet werden. Die Auswertung zeigt, dass der Roboter einfache geometrische Strecken präzise rekonstruieren kann. Die Arbeit demonstriert, dass auch mit einfacher LEGO-Hardware komplexe Vermessungsaufgaben umsetzbar sind.

***Schlagwörter*—Differentieller Antrieb, LEGO Mindstorms EV3, MATLAB, Odometrie, Zustandsautomat, Farbsensor, Linienfolger**

I. Grundidee

DIE autonome Vermessung von Strecken und Flächen ist eine grundlegende Aufgabe in der Robotik mit vielfältigen Anwendungen, von der Kartierung bis zur Qualitätskontrolle. Leicht zugängliche Plattformen wie LEGO Mindstorms und MATLAB ermöglichen es, zentrale Konzepte der Odometrie und Pfadrekonstruktion praktisch zu erproben. Ziel dieses Projekts ist die Entwicklung eines Roboters, der eine vorgegebene Strecke abfährt, seine Bewegung odometrisch erfasst und daraus ein mathematisches Abbild der Strecke erstellt, um geometrische Kenngrößen wie Abstände und Flächeninhalte zu berechnen.

II. Konzept

Der Roboter soll eine bestimmte Strecke abfahren. Dafür muss entschieden werden, wie diese Strecke festgelegt bzw. wie der Roboter gesteuert werden kann. Da die gefahrene Strecke auch vermessen werden soll, sollte das Bewegungsmuster des Roboters daran angepasst sein.

A. Steuerbarkeit

Es gibt zwei grundlegende Prinzipien, nach denen der Roboter eine gewisse Strecke entlang gesteuert werden kann: Entweder fährt der Roboter automatisch eine vordefinierte Strecke ab oder er wird manuell gesteuert. Wenn der Roboter automatisch gesteuert wird, hat das den Vorteil, dass das Ergebnis der Vermessungen des Roboters genauer reproduziert und überprüft werden kann. Potenzielle Probleme und Fehler können so besser erkannt und einfacher auf ihre jeweiligen Ursachen zurückgeführt werden. Eine mögliche Umsetzung ist, den Roboter einer Linie folgen zu lassen.

10.24352/UB.OVGU-2026-026

B. Fahrweise

Damit während der Fahrt brauchbare Bewegungsdaten erhoben werden, sollte der Roboter sich möglichst strukturiert bewegen. Ein geeignetes Bewegungsprofil ist geradliniges Fahren kombiniert mit einer Drehung auf der Stelle. Dadurch lassen sich die Bewegungsdaten eindeutig in Streckenlänge und Drehwinkel aufteilen. Das bedeutet, dass die Strecke aus einzelnen geradlinigen Teilen bestehen muss, die zueinander in einem bestimmten Winkel stehen.

Die abzufahrende Linie muss derart beschaffen sein, dass der Roboter erkennt, wann er sich in welche Richtung drehen soll. Dafür eignet sich eine dreifarbige Linie. Eine Farbe steht dabei für die Mitte der Linie, eine Farbe für rechts der Mitte und eine Farbe für links der Mitte.

III. Konstruktion

Das bisherige Konzept stellt folgende drei Anforderungen an die Konstruktion des Roboters: Er kann geradeaus fahren, er kann sich auf der Stelle drehen und er kann die Farbe der Linie unter ihm erkennen. Für die passende Fahrweise wird ein differentieller Kettenantrieb verwendet (siehe Abb. 1).

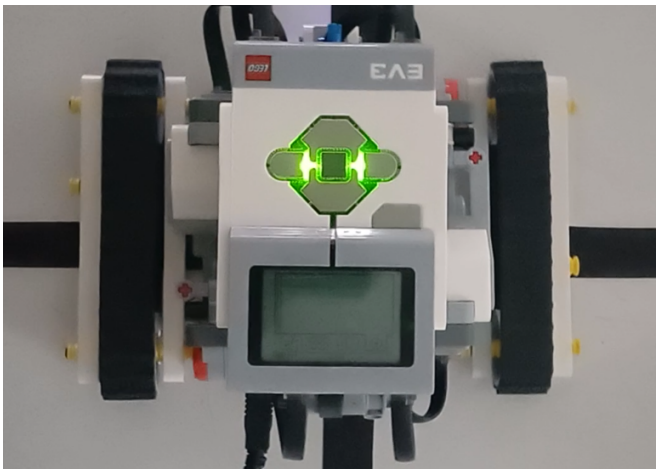


Abbildung 1. Aufbau des Geometer-Roboters mit Kettenantrieb

Damit der Roboter erkennt, welche Farbe vor ihm liegt, also eine vorausliegende Kurve vorzeitig erkennt, hat er einen Farbsensor an der Front. Um sicherzustellen, dass sich der Roboter selbst stets in der Mitte der Linie befindet, hat er einen zweiten Sensor in seiner Mitte (siehe Abb. 2). Als Mitte wird hier das Drehzentrum des Roboters bezeichnet. Damit dieses Zentrum eindeutig ist, ist der Roboter punktsymmetrisch gebaut; die zwei Motoren für den jeweils links- und rechtsseitigen Antrieb sind dafür entgegengesetzt orientiert.

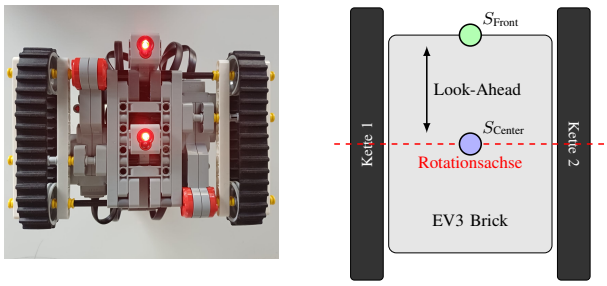


Abbildung 2. Unterseite mit Sensorplatzierung (links) und schematische Ansicht (rechts)

IV. Aufbau der Strecke

Bisher festgelegt ist, dass die Linie dreifarbig sein soll. Zu klären bleibt, ob die konkrete Farbwahl entscheidend ist und welche Breite die einzelnen Streifen haben sollten.

A. Farbwahl

Beim Test des Farbsensors zeigte sich, dass dieser nicht immer die korrekte Farbe erkannte. Am zuverlässigsten erkannte er die Farben Rot und Grün; lagen Rot und Grün aber nebeneinander, erkannte der Sensor auf der Grenze Gelb. Dieses Problem wurde gelöst, indem Rot und Grün jeweils außen positioniert wurden und für den mittleren Streifen Weiß gewählt wurde. Die Farbe Weiß erkannte der Sensor oft als Blau, deswegen wird die Farbe Blau im Programm als Weiß interpretiert. Es wurde festgelegt, dass der rechte Streifen rot und der linke Streifen grün sein soll. Um das Ende der Strecke zu markieren, wird die Farbe Gelb verwendet, da sie in der gewählten Anordnung nicht fälschlicherweise erkannt wird.

B. Streifenbreite

Im Gegensatz zur Breite der äußeren Streifen ist die genaue Breite des mittleren Streifens elementar, da sie bestimmt, wie weit der Roboter von der tatsächlichen Mitte der Linie abweichen darf. Der Farbsensor ist $2\,\text{cm}$ breit; es ist daher sinnvoll, dass der mittlere Streifen mindestens ebenso breit ist. Damit der Roboter auch in leicht schiefer Lage zur Linie eine längere gerade Strecke fahren kann und sich nicht durchgehend bei minimalen Abweichungen selbst korrigiert, sollte die Linie breiter als $2\,\text{cm}$ sein. Damit die Messergebnisse möglichst präzise sind, sollte der Streifen möglichst schmal sein. In Tests erwies sich eine Streifenbreite von $2{,}5\,\text{cm}$ als geeignet.

V. Programmierung

Das Steuerungsprogramm wird in MATLAB umgesetzt. Es muss die Farbkombinationen der beiden Sensoren interpretieren und den entsprechenden Bewegungsablauf ausführen, gleichzeitig muss es die Bewegungsdaten erfassen.

A. Interpretation der Farbkombinationen

Da beide Sensoren jeweils drei unterschiedliche Farben auf der Linie erkennen können, sind insgesamt neun Farbkombinationen möglich. Erkennt der mittlere Sensor eine Farbe und nicht Weiß, ist der Roboter falsch positioniert und muss seine Position korrigieren; das trifft auf sechs der neun Farbkombinationen zu. Ist der Roboter hingegen richtig positioniert, erfolgt die eigentliche Lenkung; das ist für die übrigen drei Farbkombinationen der Fall. Erkennen beide Sensoren Weiß, fährt der Roboter geradeaus.

Tabelle I
Sensorzustände und abgeleitete Fahrmanöver

S_{Center}	S_{Front}	**Motor-Aktion**
Weiß	Weiß	Geradeausfahrt
Weiß	Grün	Rechtskurve einleiten
Weiß	Rot	Linkskurve einleiten
Grün	Grün	Lagekorrektur nach rechts
Rot	Rot	Lagekorrektur nach links
Gelb	Gelb	Stopp (Ziel erreicht)

B. Rotation

Erkennt nur der vordere Sensor eine Farbe, der mittlere aber Weiß, soll der Roboter abbiegen – nach links bei Rot und nach rechts bei Grün. Der Roboter muss sich dafür so lange drehen, bis der vordere Sensor wieder Weiß erkennt (siehe Tabelle I). Da der Roboter auf der Stelle rotiert, darf er sich nicht unmittelbar nach Erkennen der Farbe drehen; andernfalls wäre er nach Abschluss der Drehung nicht in einer Linie mit der nächsten Teilstrecke, da das Drehzentrum über dem mittleren Sensor liegt, welcher sich zu diesem Zeitpunkt noch nicht auf dem Treffpunkt der Teilstrecken befindet. Stattdessen muss der Roboter noch so lange weiterfahren, bis das Drehzentrum genau auf der Ecke liegt, also dem Treffpunkt der Mittellinien der zusammenlaufenden Teilstrecken. Da beide Sensoren einen festen Abstand zueinander haben, ist diese zusätzliche Strecke immer gleich lang; sie kann im Programm festgelegt werden und muss nicht situativ ermittelt werden (siehe Abb. 3).

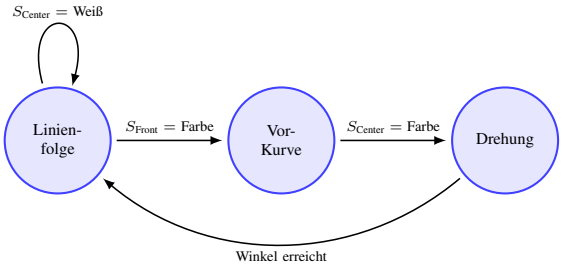


Abbildung 3. Zustandsautomat der Steuerungslogik

C. Lagekorrektur

Wird eine der anderen sechs Farbkombinationen erkannt, muss sich der Roboter neu positionieren. Erkennen beide Sensoren unterschiedliche Farben, steht der Roboter schief zur Strecke. In diesem Fall fährt der Roboter so lange geradeaus, bis der Mittelsensor den Mittelstreifen erreicht hat. Dann dreht er sich zur Mitte, bis der Frontsensor ebenfalls auf dem Mittelstreifen liegt. Erkennen beide Sensoren die gleiche Farbe, liegt der Roboter parallel zum Mittelstreifen; er muss sich daher zuerst zur Mitte drehen, bis der Frontsensor den Mittelstreifen erreicht hat.

D. Aufzeichnung der Bewegungsdaten

Um zu ermitteln, welche Distanz und welchen Winkel der Roboter zu welchem Zeitpunkt zurückgelegt hat, kann der Befehl `tachoCount` verwendet werden. Dieser Befehl gibt in seiner Grundeinstellung im Gradmaß aus, wie viel sich ein Motor seit dem letzten Zurücksetzen gedreht hat. Um daraus Bewegungsdaten zu generieren, wird bei jeder Aktion des Roboters diese Zahl für einen der beiden Motoren sowohl davor als auch danach erhoben. Aus beiden Werten wird die Differenz ermittelt, die dann in Reihenfolge des Bewegungsablaufs in einem Array gespeichert wird. Dabei wird zwischen Drehungen und Geradeausbewegung unterschieden. Da bei einer Rückwärtsdrehung des Motors zurückgezählt wird, ist eine Rechtsdrehung in der Differenz vorzeichenumgekehrt zu einer Linksdrehung.

VI. Interpretation der Bewegungsdaten

Die erhobenen Daten geben an, wie viel sich der gewählte Motor bei einer Aktion gedreht hat; für sich genommen geben diese noch keine Auskunft über die Länge der einzelnen Teilstrecken bzw. die Größe der jeweiligen Winkel der Roboterdrehungen. Hierzu müssen die Verhältnisse zwischen Motordrehung und zurückgelegter Strecke bzw. zwischen Motordrehung und Roboterdrehung ermittelt werden.

A. Motordrehung und Strecke

Um das Verhältnis zwischen Motordrehung und Strecke zu ermitteln, fährt der Roboter eine bestimmte Strecke, während die Motordrehung gemessen wird; die Länge der Strecke und die entsprechende Motordrehung werden dann zueinander ins Verhältnis gesetzt. Die Teststrecke war $6\,\mathrm{m}$ lang; es wurden mehrere Messungen auf dieser Strecke durchgeführt. Es stellte sich heraus, dass der Roboter auf längeren Strecken nicht in einer geraden Linie fuhr, sondern in einem Bogen; auf $6\,\mathrm{m}$ gerader Strecke zeigte sich eine seitliche Abweichung von $20\,\mathrm{cm}$ (siehe Abb. 4). Vermutlich geht diese Abweichung auf Verunreinigungen der Ketten zurück, die auf beiden Seiten zu unterschiedlicher Bodenreibung führen.

Die gemessenen Werte wurden jeweils zu Motordrehwinkel je gefahrenem Meter heruntergerechnet und der Durchschnitt aller Messergebnisse berechnet. Auf Zehner gerundet ergaben die Messungen einen Durchschnitt von $3300°$ je m. Dieser Wert wurde anschließend überprüft, indem der Roboter angewiesen wurde, genau $3300°$ Motordrehung auf gerader Strecke zurückzulegen. Der Roboter ist dabei $10\,\mathrm{cm}$ weiter als $1\,\mathrm{m}$ gefahren. Daher wurde ein zweiter Test mit einem entsprechend angepassten Wert von $3000°$ durchgeführt; dabei legte Roboter $1\,\mathrm{m}$ zurück. Daraus wurde abgeleitet, dass eine $30°$ Motordrehung einer Strecke von $1\,\mathrm{cm}$ entspricht.

B. Motordrehung und Roboterdrehung

Das Verhältnis zwischen Motordrehung und Roboterdrehung lässt sich bestimmen, indem die Motordrehung bei einer Volldrehung des Roboters gemessen wird. Hierzu wird der Roboter auf einem schwarzen, rechtwinkligen Kreuz so positioniert, dass das Drehzentrum auf der Mitte liegt und der Frontsensor

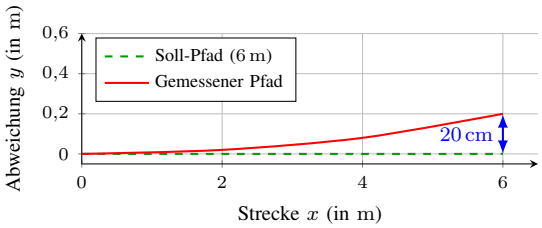


Abbildung 4. Veranschaulichung der seitlichen Abweichung auf einer geraden Strecke von $6\,\mathrm{m}$

auf einem der vier Arme. Der Roboter soll sich nach dem Start so lange drehen, bis der Frontsensor das vierte Mal schwarz erkennt; er sollte dann eine volle Drehung durchgeführt haben. Jedes Mal, wenn der Sensor schwarz erkennt, speichert er die zwischen zwei Markierungen durchgeführte Motordrehung. Das bedeutet, dass er vier Werte speichert, die jeweils einer Vierteldrehung entsprechen.

Der Test wurde drei Mal durchgeführt und es wurde der Durchschnitt aus allen gemessenen werten berechnet. Dieser wurde auf ganze Zahlen gerundet und auf volle Drehungen hochgerechnet, was ein Verhältnis von $2240°$ je Volldrehung ergab. Zur Überprüfung des Messwertes lässt man den Roboter diesen Winkel als Drehung zurücklegen; der Wert ist korrekt, wenn genau eine Volldrehung zurückgelegt wird.

Der Test bestätigte, dass $2240°$ Motordrehung einer vollständigen Rotation entspricht. Die Bewegungsdaten des Roboters können durch die beiden ermittelten Verhältnisse nun umgewandelt werden. Längendaten werden in cm umgerechnet, indem sie durch das gemessene Verhältnis von $30°$ je cm geteilt werden; Winkeldaten werden ins Bogenmaß umgewandelt, indem sie durch das ermittelte Verhältnis von $2240°$ je 2π rad geteilt werden.

VII. Datenauswertung

Nachdem der Roboter die vorgegebene Strecke abgefahren hat, werden die Bewegungsdaten mathematisch derart transformiert, dass die Strecke grafisch dargestellt werden kann. Dazu werden Länge und Richtung der geradlinigen Teilstrecken jeweils durch einen Vektor repräsentiert und anschließend zu einem Gesamtstreckengraphen zusammengefügt.

A. Algebraische Transformation

In der Ebene der gefahrenen Strecke wird ein Koordinatensystem wie folgt definiert: Der Koordinatenursprung entspreche dem Startpunkt, ab dem der Roboter seine Fahrt beginnt; die erste geradlinige Teilstrecke ab diesem Punkt liege auf der x-Achse. Zunächst werden alle Teilstrecken als Vektoren entlang der x-Achse dargestellt. Diese müssen jeweils um den Winkel rotiert werden, in dem sie zur x-Achse stehen. Dieser Winkel ergibt sich durch vorzeichenbehaftete Kumulation der einzelnen Drehwinkel. Die Vektoren werden mit der Rotationsmatrix des entsprechenden Winkels multipliziert und dadurch rotiert (vgl. [1]).

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} \begin{pmatrix} \Delta d \\ 0 \end{pmatrix}$$

B. Konstruktion des Graphen

Zur Konstruktion des Gesamtstreckengraphen werden die Eckpunkte der Strecke, an denen die Teilstrecken jeweils zusammenlaufen, in ein Koordinatensystem eingezeichnet und der Reihenfolge nach mit Linien verbunden. Dazu werden die einzelnen rotierten Vektoren kumuliert. Daraus ergeben sich die Koordinaten der Eckpunkte, die mit dem Befehl `plot` dargestellt werden können (siehe Abb. 5).

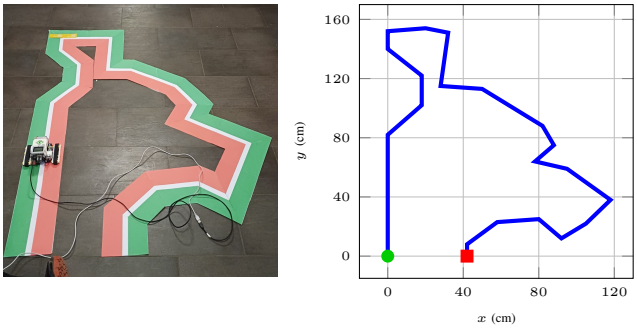


Abbildung 5. Gegenüberstellung Teststrecke vs. Odometrie-Rekonstruktion

VIII. Geometrische Berechnungen

Aus den verarbeiteten Daten lassen unterschiedliche geometrische Eigenschaften der Strecke bestimmen, beispielsweise der euklidische Abstand zwischen Start und Endpunkt sowie die durch die Strecke eingeschlossene Fläche.

A. Euklidischer Abstand

Der Abstand zwischen Start und Endpunkt lässt sich ermitteln, indem der Betrag des Verbindungsvektors bestimmt wird. Da der Startpunkt mit dem Koordinatenursprung identisch ist, entsprechen die Komponenten des Verbindungsvektors den Koordinaten des Endpunktes (siehe Abb. 6).

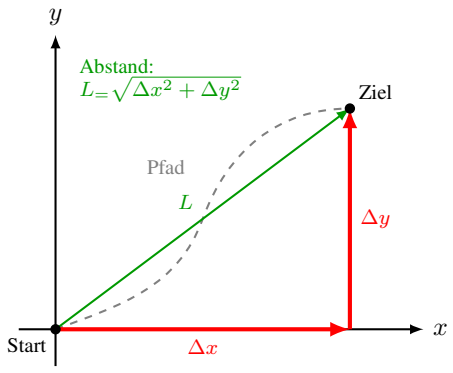


Abbildung 6. Veranschaulichung des euklidischen Abstands L

B. Flächeninhalt

Eine Fläche ist ein abgeschlossener Bereich, der durch einen geschlossenen Weg begrenzt wird. Da nicht festgelegt ist, dass der Roboter zum Startpunkt zurückkehrt, bildet die gefahrene Strecke im Allgemeinen keinen geschlossenen Weg. Daher existiert allgemein keine Begrenzung, die eine Fläche definiert. Der Weg wird geschlossen, indem er durch eine Verbindungsstrecke zwischen Start- und Endpunkt ergänzt wird. Dieser Abschluss ist nicht eindeutig, da unterschiedliche Verbindungen möglich sind. Eine naheliegende Möglichkeit ist die direkte lineare Verbindung. Nach dem Schließen des Weges, lässt sich die Größe der eingeschlossenen Fläche mithilfe der Funktion `polyarea` berechnen.

IX. Ergebnis und Diskussion

Um die Funktionsfähigkeit und Präzision des Programms zu überprüfen, wurde eine Teststrecke gebaut, die der Roboter vermessen sollte (siehe Abb. 5). Das Ergebnis ist eine präzise grafische Rekonstruktion der Strecke. Insbesondere der im Model ermittelte Abstand von 43 cm zwischen Start- und Endpunkt entspricht der Realität. Trotz der hohen Genauigkeit im Test gibt es Einschränkungen:

1) Der Roboter kann auf längeren Strecken nicht geradeaus fahren, sondern weicht leicht zur Seite ab; das Programm interpretiert diesen Bogen dennoch als gerade Strecke.
2) Kommt der Roboter von der Strecke ab, sind einerseits die erhobenen Daten verfälscht, andererseits besteht das Risiko, dass der Roboter sich unkontrolliert bewegt.
3) Erkennt der Farbsensor eine falsche Farbe, wird der Bewegungsablauf des Roboters gestört.

X. Zusammenfassung und Fazit

Im Rahmen dieser Arbeit wurde ein Messroboter auf Basis des LEGO Mindstorms EV3 entwickelt, der eine vordefinierte Strecke eigenständig abfahren und anschließend grafisch rekonstruieren kann. Durch die Verwendung eines differentiellen Kettenantriebs kombiniert mit zwei Farbsensoren konnte eine zuverlässige Pfadverfolgung und Datenerfassung realisiert werden.

Die Fahrposition wurde mittels Odometrie aus den Motordaten bestimmt. Durch die Ermittlung der Verhältnisse zwischen Motordrehung und zurückgelegter Strecke bzw. Rotationswinkel ließen sich genaue Bewegungsdaten ableiten, durch die die Strecke rekonstruiert werden konnte, woraus ihre geometrischen Eigenschaften bestimmt wurden.

Der Vergleich von der realen Teststrecke mit dem daraus rekonstruierten Pfad bestätigte die Genauigkeit des Roboters. Insgesamt zeigt das Projekt, dass sich mit vergleichsweise einfachen technischen Mitteln komplexe Aufgaben der Landvermessung und digitalen Kartierung erfolgreich umsetzen lassen.

Literaturverzeichnis

[1] WikiPedia, The Free Encyclopedia: *Drehmatrix*. https://de.wikipedia.org/wiki/Drehmatrix. Version: Dezember 2025

GEOMETER – Der Landvermesser-Roboter

Laurin Keanu Webers, Elektrotechnik/Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Dieser Beitrag beschreibt den "Geometer", einen differentiell angetriebenen Roboter auf Basis des LEGO Mindstorms EV3, der als messendes Linienfolgesystem konzipiert ist. Entwickelt wurde ein Fahrzeug, das eine dreifarbig kodierte Strecke selbsttätig abfährt, seine Bewegung per Odometrie protokolliert und daraus ein mathematisches Modell des zurückgelegten Pfades generiert. Auf Grundlage dessen lassen sich sowohl Distanzen als auch Flächeninhalte präzise ermitteln. Die Steuerung erfolgt mit MATLAB anhand der Daten zweier Farbsensoren, aus deren Signalen die erforderlichen Fahrmanöver abgeleitet werden. Die Weg- und Winkelmessung nutzt die Drehgeber der Motoren, deren Rohwerte durch eine Kalibrierung in physikalische Größen überführt werden. Die Validierung zeigt, dass der Roboter einfache geometrische Formen exakt rekonstruieren kann. Insgesamt wird deutlich, dass selbst anspruchsvolle Vermessungsaufgaben mit LEGO-Hardware lösbar sind.**

***Schlagwörter*—Differentieller Antrieb, LEGO Mindstorms EV3, MATLAB, Odometrie, Zustandsautomat, Farbsensor, Linienfolger**

I. Grundidee

DIE Fähigkeit, Strecken und Flächen autonom zu erfassen, zählt zu den grundlegenden Anforderungen an robotische Systeme und findet in zahlreichen Bereichen Anwendung, von der Kartierung bis zur industriellen Qualitätskontrolle. Plattformen wie LEGO Mindstorms und MATLAB erlauben einen niedrigschwelligen Zugang zur praktischen Erprobung von Verfahren der Odometrie und Pfadrekonstruktion. Das hier vorgestellte Projekt verfolgt das Ziel, einen Roboter zu entwickeln, der eine vorgegebene Route selbstständig befährt, seine Bewegung odometrisch dokumentiert und daraus ein mathematisches Abbild der Strecke gewinnt, um geometrische Größen wie Entfernungen und Flächeninhalte zu bestimmen.

II. Konzept

Die Aufgabe des Roboters besteht darin, eine vorgegebene Wegstrecke zu durchfahren. Hierfür muss festgelegt werden, wie diese Strecke definiert wird und wie die Steuerung des Roboters erfolgen soll. Da die zurückgelegte Strecke anschließend vermessen werden muss, ist es erforderlich, das Bewegungsverhalten des Roboters entsprechend darauf auszulegen.

A. Steuerbarkeit

Grundsätzlich lassen sich zwei methodisch verschiedene Ansätze zur Steuerung des Roboters entlang einer Strecke unterscheiden: Entweder folgt der Roboter automatisch einer vorher festgelegten Route, oder er wird durch manuelle Eingriffe gelenkt. Die automatische Steuerung bietet den Vorteil, dass die Messergebnisse besser reproduzierbar und nachvollziehbar sind. Dadurch lassen sich auftretende Probleme und Abweichungen leichter identifizieren und ihren Ursachen zuordnen. Die automatische Steuerung wird daher mittels Linienverfolgung realisiert.

B. Fahrweise

Um während der Fahrt verwertbare Messdaten zu erhalten, muss sich der Roboter nach einem definierten Bewegungsmuster verhalten. Ein geeignetes Schema besteht darin, Phasen der Geradeausfahrt mit Drehungen auf der Stelle zu kombinieren. Auf diese Weise lassen sich die erfassten Daten klar in zurückgelegte Wegstrecken und Drehwinkel unterteilen. Daraus folgt, dass die zu durchfahrende Strecke aus mehreren geradlinigen Segmenten aufgebaut sein muss, die in definierten Winkeln zueinander angeordnet sind.

Die zu verfolgende Linie muss Eigenschaften aufweisen, die es dem Roboter ermöglichen, den Zeitpunkt und die Richtung einer erforderlichen Drehung eindeutig zu identifizieren. Hierfür bietet sich der Einsatz einer dreifarbigen Linie an. Dabei steht eine Farbe für den Mittelbereich der Linie, während die beiden anderen Farben die Bereiche rechts bzw. links vom Mittelbereich kennzeichnen.

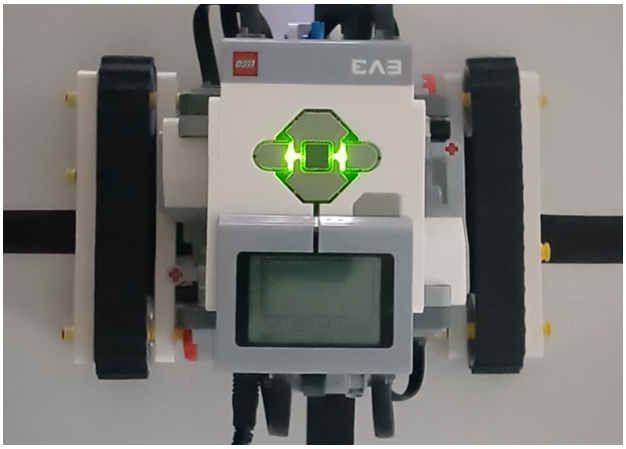


Abbildung 1. Aufbau des Geometer-Roboters mit Kettenantrieb

III. Konstruktion

Aus dem erarbeiteten Konzept ergeben sich drei grundlegende Anforderungen an die mechanische Gestaltung des Roboters. Er muss in der Lage sein, sowohl Geradeausfahrten als auch Drehungen auf der Stelle durchzuführen und gleichzeitig die Farbe der Fahrbahn unterhalb seiner Basis zu erfassen. Um das geforderte Fahrverhalten zu realisieren, kommt ein kettenbasierter Differenzialantrieb zum Einsatz (siehe Abb. 1).

10.24352/UB.OVGU-2026-027

Zur frühzeitigen Erkennung einer bevorstehenden Richtungsänderung ist ein Farbsensor an der Vorderseite des Roboters angebracht. Ein zweiter Sensor ist im Zentrum des Roboters positioniert (siehe Abb. 2). Dessen Aufgabe ist es, dauerhaft die Ausrichtung des Roboters auf den Mittelbereich der Fahrspur zu gewährleisten. Als Zentrum wird hierbei der Punkt definiert, um den der Roboter bei Drehungen auf der Stelle rotiert. Um einen eindeutigen Drehpunkt zu gewährleisten, wurde der Roboter punktsymmetrisch aufgebaut. Dies wird unter anderem durch die entgegengesetzte Ausrichtung der beiden Antriebsmotoren für die linke und rechte Kette erreicht.

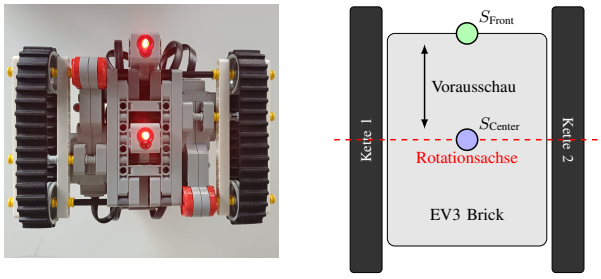


Abbildung 2. Unterseite mit Sensorplatzierung (links) und schematische Ansicht (rechts)

IV. Aufbau der Strecke

In Abschnitt II wurde die Verwendung einer dreifarbigen Linie als Grundlage für die Steuerung festgelegt. Es bleibt zu klären, ob die Auswahl der Farben von Bedeutung ist und welche Breite die einzelnen Streifen aufweisen sollten.

A. Farbwahl

Tests mit dem Farbsensor ergaben Ungenauigkeiten in der Farberkennung. Rot und Grün wurden am zuverlässigsten unterschieden; bei direkt nebeneinanderliegenden roten und grünen Flächen wurde der Übergang jedoch als Gelb interpretiert. Dieses Problem konnte umgangen werden, indem Rot und Grün an den Außenseiten platziert und Weiß für den Mittelbereich festgelegt wurden (siehe Abb. 5). Da Weiß vom Sensor überwiegend als Blau erkannt wurde, wurde im Programm eine Umdeutung implementiert: Blau wird als Weiß gewertet. Zur Markierung des Streckenendes wurde Gelb gewählt, da diese Farbe im gewählten Schema eindeutig identifiziert wird und nicht mit anderen Farbzuständen kollidiert.

B. Streifenbreite

Im Gegensatz zu den seitlichen Streifen ist die Breite des Mittelstreifens von entscheidender Bedeutung, da sie den zulässigen Toleranzbereich für die Positionierung des Roboters festlegt. Zwar besitzt das Sensorgehäuse eine Breite von $2\,\text{cm}$, die aktive Sensorfläche ist mit $0{,}75\,\text{cm}$ jedoch deutlich schmaler. Für eine zuverlässige Erkennung muss der Mittelbereich mindestens diese aktive Fläche abdecken. Um auch bei leichter Schräglage eine stabile Geradeausfahrt über längere Distanzen zu gewährleisten und ein permanentes Nachjustieren bei kleinsten Abweichungen zu vermeiden, ist eine Breite von mehr als $0{,}75\,\text{cm}$ anzustreben. Gleichzeitig erfordert eine hohe Messgenauigkeit einen möglichst schmalen Streifen. In praktischen Versuchen wurden $2{,}5\,\text{cm}$ als geeignet ermittelt.

V. Programmierung

Die Aufgabe des Steuerungsprogramms in MATLAB besteht darin, die von den Sensoren erfassten Farbkombinationen auszuwerten, daraus die erforderlichen Fahrmanöver abzuleiten und während der Fahrt die Bewegungsdaten aufzuzeichnen.

A. Interpretation der Farbkombinationen

Da beide Sensoren drei Farben (Weiß, Rot, Grün) unterscheiden können, ergeben sich neun mögliche Kombinationen. Erkennt der mittlere Sensor Rot oder Grün anstelle von Weiß, ist der Roboter dezentriert und führt eine Lagekorrektur durch. Dies trifft auf sechs Farbkombinationen zu. Bei korrekter Zentrierung (mittlerer Sensor auf Weiß) leitet der Frontsensor das eigentliche Fahrmanöver ein: Beide Sensoren auf Weiß bedeuten Geradeausfahrt; erkennt nur der Frontsensor Rot oder Grün, wird eine Kurve eingeleitet (siehe Tabelle I).

Tabelle I
Sensorzustände und abgeleitete Fahrmanöver

S_{Center}	S_{Front}	**Motor-Aktion**
Weiß	Weiß	Geradeausfahrt
Weiß	Grün	Rechtskurve einleiten
Weiß	Rot	Linkskurve einleiten
Grün	Grün	Lagekorrektur nach rechts
Rot	Rot	Lagekorrektur nach links
Gelb	Gelb	Stopp (Ziel erreicht)

B. Rotation

Beim Erkennen einer Farbe (Rot oder Grün) durch den Frontsensor, während der Mittelsensor gleichzeitig Weiß erkennt, wird eine Kurve eingeleitet – Rot signalisiert eine Linkskurve, Grün eine Rechtskurve (siehe Tabelle I). Die Drehung auf der Stelle erfolgt solange, bis der Frontsensor wieder Weiß erfasst (siehe Abb. 3). Da das Drehzentrum (Position des Mittelsensors) den Schnittpunkt der Mittellinien noch nicht erreicht hat, muss vor der Rotation eine Vorwärtsbewegung erfolgen, bis das Drehzentrum über dem Eckpunkt liegt. Aufgrund des unveränderlichen Sensorabstands ist diese Vorlaufstrecke stets identisch und wird als fester Wert im Programm eingefügt; eine dynamische Ermittlung ist obsolet.

C. Lagekorrektur

Bei sechs der neun Farbkombinationen weicht der Roboter von der Sollposition ab und die korrekte Weiterfahrt erfordert eine Lagekorrektur. Wenn der mittlere Sensor nicht Weiß erkennt, unterscheidet die Korrekturstrategie zwei Fälle:

Schräglage – Unterschiedliche Farben an den Sensoren. Der Roboter fährt zunächst geradeaus, bis der Mittelsensor den weißen Mittelstreifen erreicht. Anschließend dreht er sich, bis auch der Frontsensor auf dem Mittelstreifen liegt.

Parallelversatz – Beide Sensoren erkennen die gleiche Farbe. Der Roboter dreht sich zuerst, bis der Frontsensor den Mittelbereich erfasst. Es folgt die Lagekorrektur analog zur Schräglage.

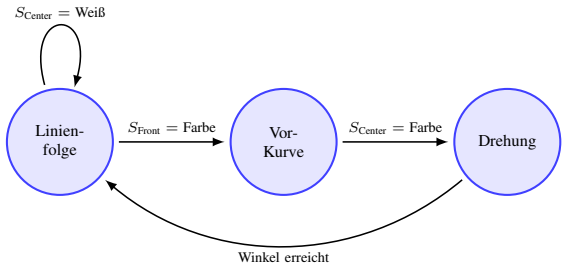


Abbildung 3. Zustandsautomat der Steuerungslogik

D. Aufzeichnung der Bewegungsdaten

Zur Erfassung der zurückgelegten Distanz und der Drehwinkel wird die Funktion `tachoCount()` verwendet, die den aktuellen Zählerstand der integrierten Motordrehgeber im Gradmaß zurückgibt. Vor und nach jeder Aktion (Geradeausfahrt oder Drehung) wird dieser Wert für einen der beiden Motoren ausgelesen. Aus der Differenz der beiden Messungen wird die inkrementelle Motordrehung ermittelt und in einem Array abgelegt, wobei zwischen den beiden Aktionstypen unterschieden wird. Da der Zähler bei Rückwärtsdrehungen dekrementiert wird, haben Rechtsdrehungen das umgekehrte Vorzeichen von Linksdrehungen. Diese Rohdaten bilden die Grundlage für die spätere Pfadrekonstruktion.

VI. Interpretation der Bewegungsdaten

Die erfassten Rohdaten repräsentieren die gemessene Motorrotation für jede durchgeführte Aktion, erlauben jedoch keine direkten Rückschlüsse auf die tatsächlich zurückgelegten Streckenlängen oder die Drehwinkel des Roboters. Hierfür müssen zwei Kalibrierungsfaktoren ermittelt werden: das Verhältnis zwischen Motordrehung und zurückgelegter Strecke sowie das Verhältnis zwischen Motordrehung und Roboterdrehung.

A. Motordrehung / Strecke

Zur Bestimmung des Übersetzungsverhältnisses zwischen Motorrotation und Fahrtstrecke wurde der Roboter wiederholt auf einer geraden Teststrecke von $6\,\text{m}$ gefahren und dabei die Motorrotation aufgezeichnet. Die Auswertung ergab eine systematische Abweichung vom idealen Geradeauskurs: Über die gesamte Distanz trat ein seitlicher Drift von $20\,\text{cm}$ auf (siehe Abb. 4). Als wahrscheinliche Ursache kommt eine ungleichmäßige Bodenreibung infolge von Kettenverunreinigungen infrage. Aus den Messreihen wurde der mittlere Drehwinkel pro Meter ermittelt und auf $3300°$ pro Meter gerundet. Bei einem Validierungsversuch legte der Roboter mit diesem Wert jedoch $1{,}10\,\text{m}$ zurück. Eine zweite Kalibrierung mit $3000°$ ergab exakt $1\,\text{m}$. Daraus wurde der Faktor $30°$ pro cm abgeleitet.

B. Motordrehung / Roboterdrehung

Die Bestimmung des Zusammenhangs zwischen Motorrotation und resultierendem Drehwinkel des Roboters erfolgte durch eine Kalibrierungsmessung bei einer vollständigen Eigenrotation. Hierfür wurde der Roboter derart auf einem schwarzen, rechtwinkligen Kreuz positioniert, dass sein Drehzentrum exakt mit der Kreuzmitte übereinstimmte und der Frontsensor auf einem der vier Kreuzarme lag. Anschließend rotierte der Roboter solange, bis der Frontsensor viermal die Farbe Schwarz detektiert hatte, was einer vollständigen Umdrehung entspricht. Bei jeder Detektion wurde die seit der letzten Markierung aufgelaufene Motordrehung gespeichert, sodass vier Werte für die jeweiligen Vierteldrehungen vorlagen. Dieser Prozess wurde dreimal wiederholt; aus allen Einzelwerten wurde der Mittelwert gebildet, ganzzahlig gerundet und auf eine Volldrehung hochgerechnet. Daraus ergab sich ein Verhältnis von $2240°$ Motorrotation pro vollständiger Roboterdrehung. Ein Validierungslauf, bei dem der Roboter angewiesen wurde, genau diesen Winkel zu rotieren, bestätigte die korrekte Kalibrierung: Es wurde exakt eine Umdrehung ausgeführt. Mit den beiden ermittelten Faktoren können die erfassten Rohdaten in die entsprechenden physikalischen Größen umgerechnet werden.

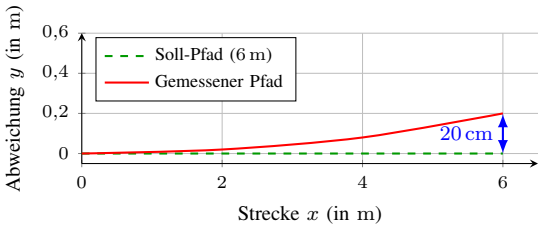


Abbildung 4. Seitliche Abweichung auf einer geraden Strecke von $6\,\text{m}$

VII. Datenauswertung

Nach Abschluss der Datenerfassung werden die aufgezeichneten Bewegungsdaten einer mathematischen Transformation unterzogen, um eine grafische Rekonstruktion der gefahrenen Strecke zu ermöglichen. Hierzu werden die einzelnen geradlinigen Segmente der Strecke als Vektoren modelliert, die anschließend zu einer Gesamtstrecke zusammengesetzt werden.

A. Algebraische Transformation

Zur Beschreibung der Bewegung in der Ebene wird ein kartesisches Koordinatensystem eingeführt, dessen Ursprung im Startpunkt des Roboters liegt und dessen x-Achse mit der ersten geradlinigen Teilstrecke zusammenfällt. Alle weiteren Streckensegmente werden zunächst ebenfalls als Vektoren in Richtung der x-Achse betrachtet. Anschließend werden diese Vektoren mittels Rotationsmatrizen um den jeweiligen kumulierten Drehwinkel rotiert, um ihre tatsächliche Orientierung in der Ebene abzubilden. Der kumulierte Drehwinkel ergibt sich aus der vorzeichenbehafteten Summe der zuvor ermittelten Einzeldrehungen. Die Transformation eines beliebigen Segments der Länge Δd mit dem zugehörigen kumulierten Winkel φ lässt sich beschreiben durch (vgl. [1]):

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} \begin{pmatrix} \Delta d \\ 0 \end{pmatrix}$$

B. Konstruktion des Graphen

Die Rekonstruktion der Gesamtstrecke als Graph erfolgt über die Bestimmung der Koordinaten sämtlicher Knotenpunkte, an

denen die geradlinigen Segmente aufeinandertreffen. Hierzu werden die transformierten Vektoren in der Reihenfolge ihrer Erfassung aufsummiert. Die resultierenden Koordinaten der Knotenpunkte können anschließend mit dem Befehl `plot()` dargestellt werden (siehe Abb. 5).

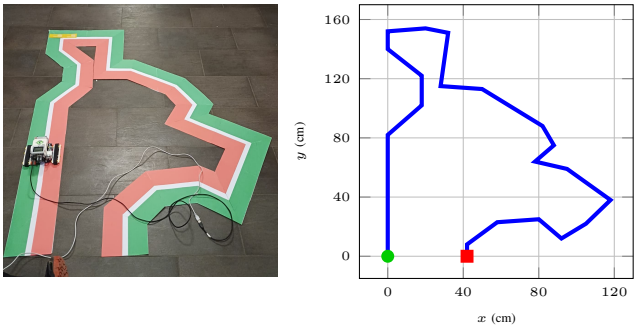


Abbildung 5. Gegenüberstellung Teststrecke vs. Odometrie-Rekonstruktion

VIII. Geometrische Berechnungen

Aus den verarbeiteten Bewegungsdaten lassen sich verschiedene geometrische Kenngrößen der befahrenen Strecke ableiten. Dazu zählen insbesondere der euklidische Abstand zwischen Start- und Endpunkt sowie der Flächeninhalt einer durch die Strecke umschlossenen Fläche.

A. Euklidischer Abstand

Die direkte Entfernung zwischen Anfangs- und Endpunkt wird durch die Länge des Verbindungsvektors bestimmt. Da der Startpunkt definitionsgemäß im Koordinatenursprung liegt, entsprechen die Komponenten dieses Vektors den kartesischen Koordinaten des Endpunktes. Die Berechnung erfolgt über den Satz des Pythagoras (siehe Abb. 6).

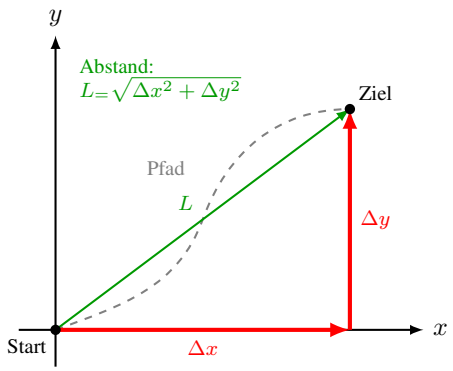


Abbildung 6. Veranschaulichung des euklidischen Abstands L

B. Flächeninhalt

Eine Fläche ist als ein abgeschlossener Bereich definiert, dessen Berandung einen geschlossenen Weg bildet. Da die gefahrene Strecke im Allgemeinen nicht zum Startpunkt zurückführt, liegt kein geschlossener Weg vor. Um dennoch eine Flächenberechnung zu ermöglichen, wird der Weg durch eine Verbindungsstrecke zwischen Start- und Endpunkt ergänzt, die jedoch nicht eindeutig ist. Dennoch ist eine naheliegende Option die direkte lineare Verbindung beider Punkte (siehe Abb. 6). Der eingeschlossene Flächeninhalt kann so mit der MATLAB-Funktion `polyarea()` berechnet werden.

IX. Ergebnis und Diskussion

Zur Validierung der Funktionsfähigkeit und Messgenauigkeit des Roboters wurde eine reale Teststrecke aufgebaut und abgefahren (siehe Abb. 5). Die aus den Odometriedaten rekonstruierte Wegstrecke zeigt eine hohe Übereinstimmung mit der tatsächlichen Geometrie der Teststrecke. Insbesondere die berechnete Distanz von $43\,\mathrm{cm}$ zwischen Start- und Endpunkt entspricht dem real gemessenen Wert.

Trotz dieser grundsätzlich guten Genauigkeit sind folgende Einschränkungen zu beachten:

1) Bei längeren Geradeausfahrten weicht der Roboter infolge asymmetrischer Bodenreibung von der idealen Linie ab und bewegt sich leicht bogenförmig. Das Programm behandelt diese Bewegung dennoch als geradliniges Segment, was zu einer systematischen Verfälschung der rekonstruierten Geometrie führt.
2) Ein Verlassen der farbigen Fahrspur hat zweierlei Konsequenzen: Zum einen werden die erfassten Bewegungsdaten fehlerhaft, zum anderen kann der Roboter die Orientierung verlieren und unkontrollierte Manöver ausführen.
3) Fehlerkennungen des Farbsensors, beispielsweise durch Umgebungslicht oder Verschmutzungen, beeinträchtigen den korrekten Ablauf der Zustandsmaschine und können Fehlmanöver auslösen.

X. Zusammenfassung und Fazit

Im Rahmen dieser Arbeit wurde ein Messroboter auf Basis des LEGO Mindstorms EV3 und MATLAB entwickelt, der in der Lage ist, eine vorgegebene, farbkodierte Strecke autonom abzufahren und aus den erfassten Bewegungsdaten eine grafische Rekonstruktion des Pfades zu erstellen. Die Realisierung einer zuverlässigen Pfadverfolgung und Datenerfassung gelang durch den Einsatz eines differentiellen Kettenantriebs in Kombination mit zwei Farbsensoren. Die Bestimmung der Fahrposition erfolgte odometrisch über die Auswertung der Motorrotationswinkel. Eine Kalibrierung der Motordrehung erlaubte die Umrechnung der Rohdaten in Streckenlängen und Drehwinkel. Ein Vergleich der rekonstruierten Strecke mit der realen Teststrecke bestätigte die hohe Genauigkeit des Gesamtsystems. Das Projekt demonstriert, dass sich mit vergleichsweise einfachen technischen Möglichkeiten komplexe Aufgabenstellungen aus den Bereichen der Landvermessung und digitalen Kartierung erfolgreich umsetzen lassen.

Literaturverzeichnis

[1] WIKIPEDIA, THE FREE ENCYCLOPEDIA: *Drehmatrix*. https://de.wikipedia.org/wiki/Drehmatrix. Version: Dezember 2025

Umblättermaschine

Andrii Vahanov, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Bericht geht es um die Entwicklung der Umblättermaschine unter Verwendung von LEGO Mindstorms im Rahmen des LEGO-Projektseminars 2026 an der Otto-von-Guericke-Universität Magdeburg. Der Roboter reagiert auf die Bewegung der Hand neben dem Ultraschallsensor, blättert das Buch mithilfe von zwei Motoren um und macht ein Foto. Das Programm wurde in MATLAB geschrieben. Im Beitrag werden der Aufbau und die Möglichkeiten der Maschine erklärt. Außerdem werden die Probleme, die während des Entwicklungsprozesses aufgetreten sind, sowie deren Lösungen erläutert.**

***Schlagwörter*—Buch, LEGO Mindstorms, Maschine, MATLAB, Umblätterrung**

I. Einleitung

DIE moderne Welt ist sehr digitalisiert, und es ist oft wichtig, Dokumente, Bücher oder Texte aus Heften zu scannen. Mit Papieren, deren Seiten getrennt sein können, ist das kein Problem, da der Scanner gut mit ihnen funktioniert. Wenn man jedoch mit Büchern arbeitet, muss man jede Seite selbst umblättern und fotografieren. Die Umblättermaschine erleichtert diesen Prozess. Der Roboter trennt eine Seite, blättert sie um und macht dann ein Foto. Deshalb läuft die Buchscannung viel schneller.

II. Bestandteile des Roboters

In diesem Abschnitt werden wichtige Informationen bezüglich der genutzten Bauteile gegeben, um die Funktionsweise des Roboters verständlicher zu machen.

A. Entwicklungsplattform

In der Arbeit wird LEGO Mindstorms EV3 als Kontrolleinheit verwendet (siehe Abbildung 1). Der EV3 wird mit der MATLAB-Sprache programmiert, um das Programm zu vereinfachen und keine weiteren Apps zu benötigen.

B. Motoren

Für diesen Roboter benötigt man zwei große Motoren (siehe Abbildung 2). Der erste Motor mit dem Rad trennt die Seiten auseinander, und dann blättert der zweite Motor sie um. Dafür sollen beide Motoren miteinander synchronisiert sein.

C. Ultraschallsensor

Der Ultraschallsensor (siehe Abbildung 3) ist der Hauptbestandteil der Konstruktion. Der Sensor reagiert auf die Bewegung und lässt die Motoren drehen. Damit kann man kontrollieren, wie viele Seiten umgeblättert werden und wie groß die Pause dazwischen ist.

DOI: 10.24352/UB.OVGU-2026-029

Abbildung 1. LEGO Mindstorms EV3

Abbildung 2. LEGO-Motor

III. Aufbau und Programmierung

In diesem Abschnitt werden das Konzept und die Realisierung erläutert.

A. Aufbau

Das ganze Projekt besteht aus der EV3-Box, zwei Motoren, einem Ultraschallsensor, einer Taste und LEGO-Bausteinen.

Abbildung 3. LEGO-Ultraschallsensor

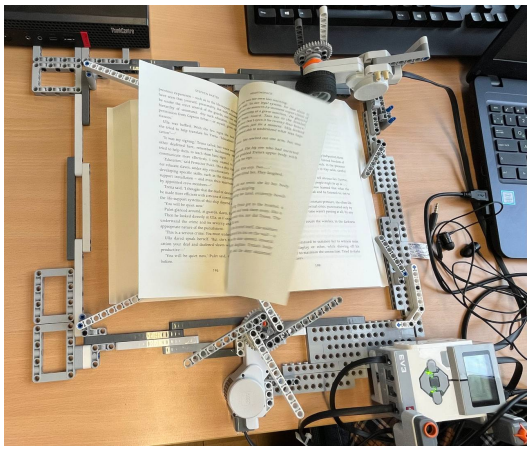


Abbildung 4. Umblättermaschine

Aus LEGO-Steinen wurden der Rahmen, der die Buchbewegung begrenzt, erstellt. An dem Rahmen sind beide Motoren und die EV3-Box befestigt (siehe Abbildung 4). Die EV3-Box ist auch mit dem Sensor und der Taste verbunden. Die Hauptidee war, dass der Ultraschallsensor reagiert, wenn der Abstand zum oberen Objekt kleiner als 20 cm ist. Falls diese Bedingung erfüllt ist, beginnt Motor A seine Bewegung. Er trennt eine Seite, damit Motor B sie umblättern kann. Die Motoren arbeiten nicht gleichzeitig, sondern nacheinander. Auch ist es unmöglich, bis zum Ende der ersten Umblätterung mit dem zweiten anzufangen.

Wichtig ist, dass der Roboter selbst keine Fotos machen kann. Dafür braucht man einen Buchscanner oder eine Kamera.

B. MATLAB-Programm

Das MATLAB-Programm ist einfach, trotzdem funktioniert es gut genug. Zuerst sollte man den EV3-Block, beide Motoren, den Sensor und die End-Taste im MATLAB-Code definieren. Dann werden die Parameter der Motoren sowie die Leistung und der Drehwinkel festgestellt. Wichtig ist, dass die Motoren nach dem Ende der Arbeit sofort die Bewegung unterbrechen sollen. Dafür kann man die Funktion *brake.mode = 'Brake'* nutzen.

Dann wurde der Hauptteil des Programms geschrieben. Den Programmablaufplan kann man in Abbildung 5 sehen.

Und der gesamte Code ist im Anhang in Listing 1 verfügbar.

C. Kamera

Beim Aufbau kam der CZUR Shine Scanner [1] (siehe Abbildung 7) zusammen mit der zugehörigen Software [2] zum Einsatz. Der Grund dafür ist die gute Bildqualität und die Möglichkeit, Fotos nach dem Umblättern automatisch zu machen.

IV. Ergebnisdiskussion

Bevor das Projekt seine Ziele erreichte, hatte unser Team viele Probleme getroffen. Es ist möglich, alle Herausforderungen in zwei große Teile zu trennen: Das sind Konstruktionsprobleme und Programmiermissverständnisse.

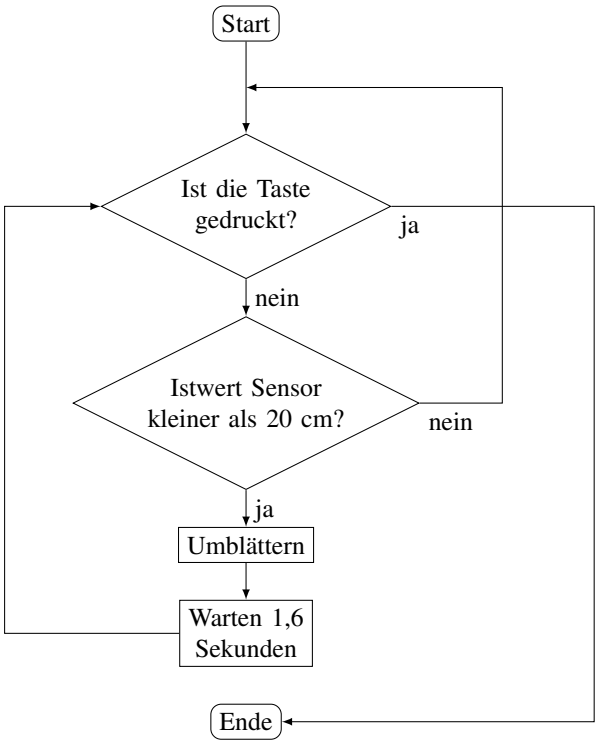


Abbildung 5. Programmablaufplan

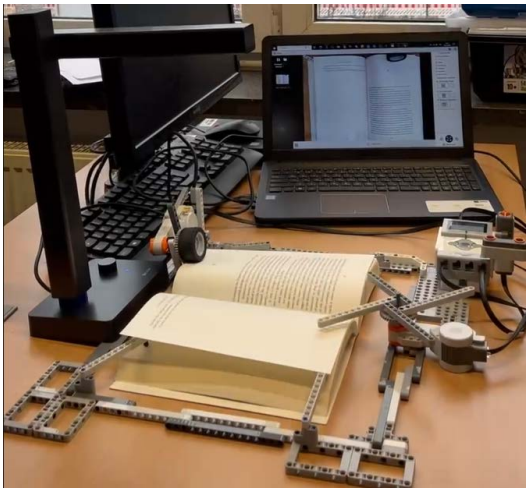


Abbildung 6. Fertiger Roboter

A. Konstruktionsherausforderungen

Es gab zwei große Dinge, wegen derer die Maschine nicht funktioniert hat. Die erste war die instabile Konstruktion. Die Rahmen konnten die Buchbewegung nicht begrenzen. Das wurde durch eine totale Änderung der Maschine gelöst.

Das zweite Problem ist schwerer zu lösen. Es liegt an verschiedenen Buchformaten und -größen. Die Lösung ist eine einfache Rahmenkonstruktion, die man für jedes Buchformat schnell verändern kann.

Abbildung 7. CZUR-Kamera

B. Probleme mit dem Programm

Diese Herausforderungen waren viel einfacher, aber gefährlicher für Bücher, weil die Motoren sie sehr leicht beschädigen könnten. Die Probleme waren die Synchronisation der Motoren sowie die richtige Leistung und der richtige Winkel. Um sichere Parameter zu finden, sollte das Team viele verschiedene Zahlen testen.

V. Zusammenfassung und Fazit

Insgesamt hat das Projekt seine erklärten Ziele erreicht (siehe Abbildung 6). Selbstverständlich kann man den Roboter noch verbessern und weiterentwickeln. Beispielsweise wird die Maschine mit allen Buchgrößen effizient arbeiten, oder werden die Rahmen das Buch automatisch begrenzen.

VI. Anhang

A. Programmcode der Umblättermaschine

Listing 1. Programmcode

```
b = EV3();
b.connect('usb');
ma = b.motorA;
ma.setProperties('Power', 20, ...
    ... 'LimitValue', 180);
ma.brakeMode = 'Brake';
mb = b.motorB;
mb.brakeMode = 'Brake';
s1 = b.sensor1;
s4 = b.sensor4;
while s1.value != 1
    if s4.value < 20
        ma.start();
        waitFor(ma);
        ma.stop();
        mb.setProperties('Power', ...
    ... -20, 'LimitValue', 180);
        mb.start();
        waitFor(mb);
        mb.stop();
        pause(1.6);
    end
end
```

Literaturverzeichnis

[1] CZUR TECH CO: *CZUR Shine Scanner*. https://www.czur.com/product/shineultra. Version: März 2026

[2] CZUR TECH CO: *CZUR Software*. https://www.czur.com/support. Version: März 2026

Konstruktion und Programmierung eines autonomen Robo-Transporters zur Lastenmanipulation

Bohdan Soldatko, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Das vorliegende Paper beschreibt die Entwicklung und Konstruktion eines autonomen Transportsystems unter Verwendung von LEGO Mindstorms EV3 und MATLAB. Im Fokus der Arbeit steht die mechanische Realisierung eines stabilen Hubmechanismus in Form eines Gabelstaplers sowie die softwareseitige Implementierung einer präzisen Steuerung. Das System ist in der Lage, Objekte mittels Ultraschallsensorik zu detektieren und autonom aufzunehmen. Die Kommunikation zwischen der Steuerungssoftware und der Hardware wird über Bluetooth realisiert. Die Ergebnisse der Erprobungsphase werden detailliert dargelegt und mögliche Optimierungspotenziale für zukünftige Weiterentwicklungen aufgezeigt.**

***Schlagwörter*—Greifarm, Transporter, Manipulator, Ultraschallsensor, Gabelstapler**

I. Einleitung

AUTONOME Transportsysteme und fahrerlose Transportfahrzeuge (FTF) stellen ein zentrales Forschungsfeld in der modernen Intralogistik und Automatisierungstechnik dar [1]. Sie werden eingesetzt, um Materialflüsse effizienter zu gestalten und menschliche Fehlerquellen zu minimieren. Entsprechende Prototypen und Machbarkeitsstudien lassen sich im Labormaßstab hervorragend mit modularer Robotik abbilden.

Im Rahmen dieses Projekts wurde ein Prototyp entwickelt, der die grundlegenden Prinzipien der automatisierten Lastenmanipulation demonstriert. Die zentrale Herausforderung bestand in der Konstruktion eines Roboters, der nicht nur eine ausreichende mechanische Stabilität für Hebevorgänge aufweist, sondern auch über eine zuverlässige Umgebungserfassung verfügt. Das System wurde so konzipiert, dass es sowohl teleoperiert über eine grafische Benutzeroberfläche als auch in teilautonomen Routinen betrieben werden kann [2].

Der folgende Bericht dokumentiert die systematische Entwicklung, von den grundlegenden Anforderungen über den mechanischen Aufbau bis hin zur Programmierung der Algorithmen.

II. Systemanforderungen und Vorüberlegungen

Dieser Abschnitt definiert die grundlegenden mechanischen und softwareseitigen Anforderungen, die das spätere Designkonzept des Roboters maßgeblich bestimmen. Dabei wurden insbesondere die physikalischen Grenzen der verwendeten LEGO-Komponenten berücksichtigt.

DOI: 10.24352/UB.OVGU-2026-031

A. Anforderungen an die Mechanik

Die grundlegenden Anforderungen an die Robotik-Hardware bestanden in der Fähigkeit, eine Nutzlast von bis zu $0{,}35\,\mathrm{kg}$ stabil anzuheben und zu transportieren, ohne dass das System das Gleichgewicht verliert. Hierfür war ein tiefer Schwerpunkt essenziell. Da die Last an der Front des Roboters aufgenommen wird, verlagert sich der Gesamtschwerpunkt bei Beladung signifikant nach vorne. Diesem Effekt musste durch ein ausreichendes Gegengewicht im Heckbereich sowie einen verlängerten Radstand entgegengewirkt werden. Zudem musste das Fahrwerk so ausgelegt werden, dass eine hohe Traktion auf verschiedenen Untergründen gewährleistet ist, während gleichzeitig eine hohe Wendigkeit für das Manövrieren auf engem Raum erhalten bleibt.

B. Anforderungen an Sensorik und Software

Um Objekte vor dem Roboter zuverlässig zu erfassen, wurde eine frontal ausgerichtete Sensorik gefordert. Das System muss Entfernungen präzise messen können, um den Greifmechanismus im exakt richtigen Moment auszulösen. Eine Fehlmessung in diesem Bereich würde dazu führen, dass die Gabeln die Last entweder rammen oder ins Leere greifen. Softwareseitig wurde eine Implementierung in MATLAB vorgegeben, welche eine benutzerfreundliche Schnittstelle zur Überwachung und Steuerung (App Designer) sowie stabile Kommunikationsprotokolle zur Datenübertragung an den Microcontroller erfordert [3]. Die Latenzzeit der Bluetooth-Verbindung durfte dabei einen kritischen Schwellwert nicht überschreiten, um Echtzeitreaktionen zu ermöglichen.

III. Technische Umsetzung und Hardware

Die konstruktive Umsetzung gliedert sich im Wesentlichen in den Aufbau des mobilen Fahrwerks, die mechanische Realisierung des Hubmechanismus sowie die Integration der notwendigen Umfeldsensorik.

A. Mechanische Konstruktion des Fahrwerks

Das Basisgestell des Roboters wurde gezielt asymmetrisch aufgebaut, um die Lastenverteilung zu optimieren. Als Antriebskomponenten kommen zwei servogesteuerte große Motoren zum Einsatz, welche die hintere Achse antreiben. Um die Traktion zu maximieren, wurde im Heckbereich ein Raupenketten-Antrieb installiert. An der Vorderachse wurden hingegen Räder mit Gummibereifung verwendet. Diese Hybrid-Lösung aus Ketten und Rädern reduziert den Reibungswiderstand

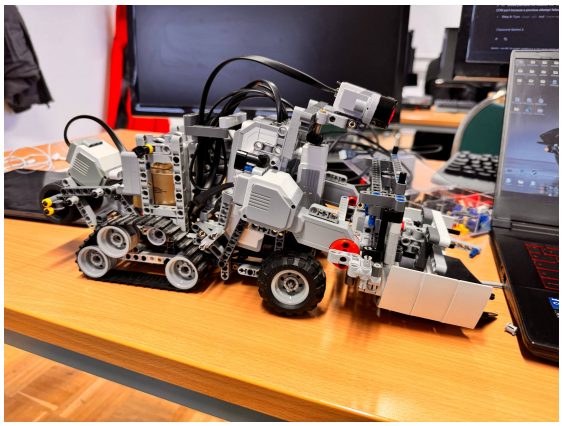


Abbildung 1. Seitenansicht des Robo-Transporters mit sichtbarem Kettenlaufwerk im Heckbereich und dem vorderen Hubwerk.

bei Kurvenfahrten, der bei reinen Kettenfahrzeugen oft zu Ungenauigkeiten in der Odometrie führt.

In Abbildung 1 ist die Anordnung der Antriebskomponenten zu erkennen. Der EV3-Stein, welcher als zentrale Steuereinheit fungiert, wurde mittig platziert, um als Gegengewicht zu den frontal aufgenommenen Lasten zu dienen.

B. Konstruktion des Hubmechanismus

An der Front des Fahrzeugs befindet sich der Gabelstapler-Mechanismus. Dieser wird von einem EV3-Medium-Motor angetrieben, der über eine Zahnradübersetzung mit einer Zahnstange verbunden ist. Durch diese Konstruktion wird die Rotationsbewegung des Motors in eine lineare Vertikalbewegung umgewandelt. Die Gabeln wurden flach und langlaufend konstruiert, um leicht unter Zielobjekte fahren zu können. Die mechanische Belastbarkeit der Zahnstange wurde so gewählt, dass auch bei maximaler Last kein Durchrutschen der Zähne auftritt.

C. Sensorik zur Umfelderfassung

Wie in Abbildung 2 ersichtlich, ist im oberen Frontbereich ein Ultraschallsensor integriert. Dieser ist leicht nach unten geneigt, um kleine Objekte direkt vor der Gabel zu detektieren. Der Sensor arbeitet nach dem Impuls-Echo-Verfahren. Hierbei wird ein Schallsignal emittiert und die Zeit bis zum Eintreffen der Reflexion gemessen. Die Distanz s zum Objekt berechnet sich wie folgt:

$$s = v \cdot \frac{t}{2} \tag{1}$$

In dieser Gleichung repräsentiert v die Ausbreitungsgeschwindigkeit des Schalls in Luft (etwa $343\,\frac{\mathrm{m}}{\mathrm{s}}$ bei Raumtemperatur) und t die gemessene Laufzeit des Schallsignals.

IV. Softwarearchitektur und Steuerung

Die Steuerung des Systems erfolgt aus der Ferne über einen Host-Computer. Hierfür wurde in MATLAB eine objektorientierte Programmstruktur entwickelt.

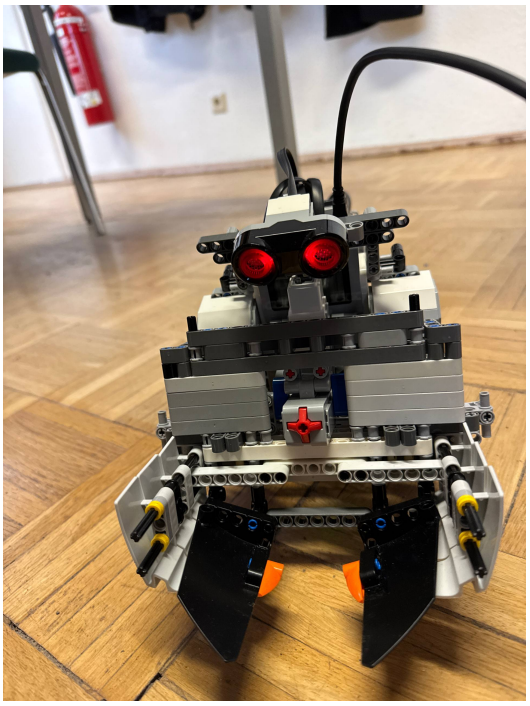


Abbildung 2. Frontansicht des Hubmechanismus und der Sensormontage.

A. Grafische Benutzeroberfläche (GUI)

Über den MATLAB App Designer wurde eine interaktive Steuerungssoftware implementiert. Diese GUI ermöglicht das Herstellen der Bluetooth-Verbindung, das Auslesen von Sensorwerten sowie die manuelle Steuerung der Motoren.

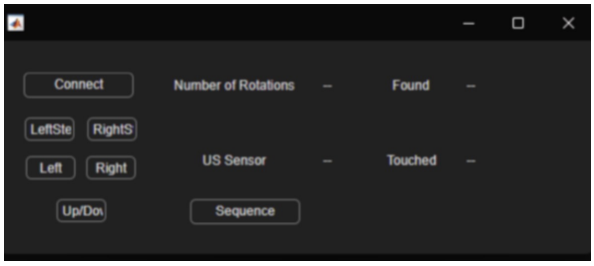


Abbildung 3. Layout der MATLAB-Benutzeroberfläche zur manuellen und teilautonomen Steuerung des Systems.

Wie in Abbildung 3 dargestellt, wurden spezifische Schaltflächen programmiert. Jeder Button löst eine definierte Callback-Funktion aus, welche die entsprechenden Steuerbefehle an den Stein sendet. Die Oberfläche bietet zudem ein Echtzeit-Display für die Distanzwerte des Ultraschallsensors.

B. Ablaufsteuerung des autonomen Modus

Neben der manuellen Steuerung wurde eine Routine für die autonome Objektaufnahme entwickelt. Der Prozess beginnt mit einer Suchrotation. Sobald ein Objekt im Erfassungsbereich des Ultraschallsensors detektiert wird, richtet sich der Roboter aus und fährt geradeaus, bis ein minimaler Schwellwert der Distanz unterschritten wird. Anschließend wird das Fahrzeug gestoppt und der Hebevorgang initiiert. Der vollständige logische Ablauf

dieser Routine ist im Programmablaufplan in Abbildung 4 visualisiert.

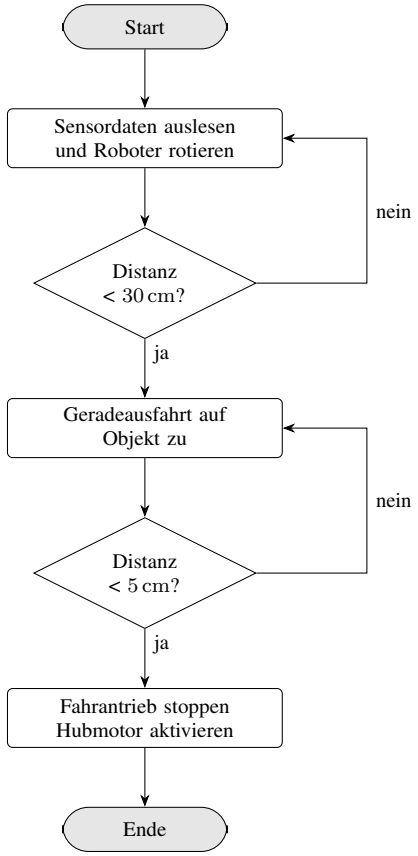


Abbildung 4. Programmablaufplan des teilautonomen Hebealgorithmus.

V. Ergebnisdiskussion

Die Funktionstüchtigkeit des Systems wurde in mehreren Testreihen evaluiert. Dabei wurde besonders auf das Zusammenspiel zwischen mechanischer Lastgrenze und Softwarestabilität geachtet.

A. Mechanische Zuverlässigkeit

Die mechanische Struktur erwies sich als äußerst robust. Der Gabelmechanismus konnte Lasten ohne signifikante Deformationen anheben. Es wurde jedoch beobachtet, dass der Roboter auf glatten Laborböden aufgrund der hohen Traktion der Gummiräder im Frontbereich ein leichtes Ruckeln bei engen Kurvenfahrten aufwies.

Die hinteren Ketten neigten bei plötzlichen Bremsmanövern zu minimalem Schlupf, was in einer leichten Abweichung der Odometriewerte resultierte. Durch die Anpassung der Beschleunigungsrampen in der Software konnte dieser Effekt jedoch weitgehend kompensiert werden. Bei einer Maximalbelastung von über $0{,}4\,\mathrm{kg}$ stieg die Stromaufnahme des Hubmotors stark an, was die thermische Belastung erhöhte.

B. Software- und Verbindungsstabilität

Ein wesentliches Problem während der Entwicklungsphase war die Stabilität der Bluetooth-Kommunikation. In den MATLAB-Konsolenausgaben traten sporadisch Fehler vom Typ `Error using brick.connect` auf. Die Ursache lag in asynchronen Aufrufen innerhalb der Callback-Funktionen, wenn mehrere Steuerbefehle zu schnell hintereinander gesendet wurden. Dies wurde gelöst, indem ein Zustandsautomat (State Machine) implementiert wurde, der sicherstellt, dass Befehle erst gesendet werden, wenn der vorherige Befehl vom Microcontroller quittiert wurde.

VI. Zusammenfassung und Fazit

Das Projekt hat gezeigt, dass mit LEGO Mindstorms und MATLAB komplexe Automatisierungsaufgaben erfolgreich modelliert werden können. Es wurde ein funktionsfähiger Robo-Transporter entwickelt, der Objekte zuverlässig erkennt, anfährt und anhebt. Die Kombination aus manuellem Teleoperationsmodus und autonomer Suchroutine macht das System äußerst flexibel für verschiedene Einsatzszenarien.

Zukünftige Erweiterungen könnten die Integration eines Farbsensors umfassen, um eine sortenreine Logistik zu ermöglichen (z. B. nur grüne Objekte transportieren). Zudem könnte durch den Einsatz eines PID-Reglers für die Radmotoren der Geradeauslauf weiter perfektioniert werden, um auch über größere Distanzen präzise Navigationsergebnisse zu erzielen.

Literaturverzeichnis

[1] L. Hering und H. Hering, *Technische Berichte: Gliedern, Gestalten, Vortragen*, 8. Aufl. Springer Vieweg, 2022.
[2] LEGO Group, *LEGO MINDSTORMS EV3 Benutzerhandbuch*, 2013.
[3] The MathWorks Inc., *MATLAB App Designer Overview*, 2024.

Wall-E: Der Spracherkennungsroboter

Abdulrahman Hamouda, Elektro- und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen dieser Arbeit wurde ein sprachgesteuerter LEGO-Roboter auf Basis einer NXT-Steuerungseinheit realisiert. Zur Erfassung von Sprachbefehlen wurde ein externes Smartphone-Mikrofon verwendet, dessen Daten über eine Bluetooh-Schnittstelle und MATLAB Drive an die Steuerungssoftware übertragen wurden. Die Distanzmessung und Hinderniserkennung erfolgten über einen Ultraschallsensor. Die gesamte Programmlogik wurde in MATLAB implementiert. Im Ergebnis konnten die Befehle „Forward“, „Back“, „Left“, „Right“, „Turn“, „Zigzag“, „Circle“ und „Stop“ erfolgreich erkannt und in entsprechende Motorbewegungen umgesetzt werden.**

***Schlagwörter*—Bluetooth, LEGO Mindstorms, MATLAB, NXT-Controller, Robotik, Spracherkennung**

I. Einleitung

DIE Steuerung von Robotersystemen durch menschliche Sprache gewinnt in Bereichen wie der Unterstützung von Menschen mit körperlichen Beeinträchtigungen sowie bei Einsätzen in gefahrvollen Arbeitsumgebungen zunehmend an Bedeutung. Ziel des vorliegenden Projekts war die Entwicklung des Robotermodells „Wall-E“, welches die Simulation einer solchen freihändigen Interaktion ermöglicht. Durch die Verarbeitung verbaler Befehle soll eine Unterstützung für Nutzer realisiert werden, die in ihrer Mobilität eingeschränkt sind oder unter erschwerten Bedingungen, wie beispielsweise auf Baustellen, tätig sind. Die technische Umsetzung erforderte die Lösung zweier zentraler Herausforderungen.

Zunächst musste ein Verfahren implementiert werden, um akustische Signale in interpretierbare Textdaten umzuwandeln. Des Weiteren stellte die Systemarchitektur, bestehend aus der Datenübertragung vom Smartphone-Mikrofon über MATLAB Drive bis hin zur Ausführung auf dem NXT-Controller [1], eine Herausforderung hinsichtlich der Signallatenz dar. Um einen sicheren Betrieb zu gewährleisten, wurde eine Priorisierungslogik integriert: Trotz aktiver Fahrbefehle erzwingt der integrierte Ultraschallsensor bei Detektion eines Hindernisses den sofortigen Stillstand des Roboters.

Das Paper ist wie folgt strukturiert: Nach den theoretischen Grundlagen werden die technische Realisierung (Hardware und Software) sowie die anschließende Ergebnisdiskussion mit Systemevaluation beschrieben. Die Arbeit schließt mit Fazit, Anhang und Literaturverzeichnis ab.

II. Vorbetrachtungen

Die erfolgreiche Realisierung eines sprachgesteuerten Robotersystems erfordert eine präzise Abstimmung zwischen Hardwarekomponenten, Softwarealgorithmen und der Kommunikationsinfrastruktur. In diesem Abschnitt werden die theoretischen Grundlagen und technischen Rahmenbedingungen erläutert, die für den Aufbau des Roboters „Wall-E“ maßgeblich sind. Dies umfasst sowohl die Definition der Systemanforderungen als auch die Analyse der verwendeten Sensortechnik und der Datenübertragungswege über MATLAB Drive.

DOI: 10.24352/UB.OVGU-2026-033

A. Systemanforderungen und theoretische Konzeption

Die erfolgreiche Realisierung des Roboters „Wall-E“ setzt die Erfüllung spezifischer funktionaler Anforderungen voraus. Im Zentrum der Konzeption steht die Implementierung eines robusten Sprachsteuerungssystems, welches in der Lage sein muss, acht distinkte verbale Befehle zu identifizieren und in präzise Bewegungsabläufe zu übersetzen. Zu diesen Befehlen zählen die Richtungsanweisungen für Vorwärts- und Rückwärtsfahrten, Drehungen nach links und rechts sowie komplexere Manöver wie Zickzack- und Kreisbewegungen.

Eine kritische Anforderung stellt hierbei die Sicherheitspriorisierung dar: Um Kollisionen zu vermeiden, muss das System bei der Detektion eines Hindernisses durch den Ultraschallsensor innerhalb einer Sicherheitsdistanz von $30\,\mathrm{mm}$ einen unmittelbaren Stopp erzwingen. Dieser hardwarenahe Sicherheitsmechanismus agiert unabhängig von der softwareseitigen Spracherkennung und besitzt im Steuerungsalgorithmus die höchste Priorität.

B. Analyse der Sensorik und Hardwarebeschränkungen

In der frühen Entwurfsphase wurde die Eignung des LEGO-NXT-Schallsensors [2] für die direkte Spracherkennung evaluiert. Die technische Analyse ergab jedoch, dass dieser Sensor ausschließlich zur Messung der relativen Schallintensität (Amplitudenwerte auf einer Skala von 0 bis 100) ausgelegt ist. Da systembedingt keine Kapazitäten für eine Frequenzanalyse oder Phonem-Erkennung vorhanden sind, ist eine Unterscheidung spezifischer Wörter allein über die NXT-Hardware nicht realisierbar. Infolgedessen wurde die Spracherkennung auf ein externes Smartphone ausgelagert. Dieser Ansatz nutzt die fortschrittlichen Speech-to-Text-Algorithmen moderner mobiler Endgeräte, um akustische Signale effizient in interpretierbare Textdaten umzuwandeln und so die Hardwarelimitierungen des NXT-Controllers zu umgehen.

C. Datenübertragung und Softwarearchitektur

Die softwareseitige Kopplung zwischen dem mobilen Endgerät und der zentralen Steuereinheit wird über eine Cloud-Synchronisation mittels MATLAB Drive realisiert. Die Architektur basiert auf einer funktionalen Trennung in zwei Kerndateien: Eine Datei `voice` fungiert als Input-Speicher für die erfassten Sprachbefehle, während die zweite Datei `main` die übergeordnete Steuerungslogik enthält. Durch ein

Abbildung 1. Hardwareaufbau von „Wall-E“: Antriebsmotoren, Ultraschallsensor (Frontmontage unter Kabelzuführung, über NXT-Baustein) und passives Stützrad. Der Ultraschallsensor ist an den Öffnungen erkennbar und liegt neben dem rechteckigen Geräuschsensor.

kontinuierliches Polling-Verfahren überwacht der MATLAB-Algorithmus die Input-Datei auf neue Einträge. Sobald eine Änderung detektiert wird, erfolgt ein Abgleich mit dem programmierten Befehlssatz und die anschließende Übertragung des Steuerbefehls via Bluetooth an den Roboter. Die daraus resultierende Latenzzeit muss im Bewegungsmodell des Roboters berücksichtigt werden, um eine flüssige Interaktion zu gewährleisten.

D. Physikalische Grundlagen des Ultraschallsensors

Die Distanzmessung zur Hindernisvermeidung basiert auf dem physikalischen Prinzip der Laufzeitmessung (Time-of-Flight). Der Sensor emittiert ein Ultraschallsignal, welches an Objekten im Umfeld reflektiert wird. Unter Berücksichtigung der Schallgeschwindigkeit $c \approx 343\,\frac{\mathrm{m}}{\mathrm{s}}$ wird aus der Zeitspanne Δt zwischen Emission und Detektion die Distanz d zum Objekt berechnet:

$$d = \frac{c \cdot \Delta t}{2} \tag{1}$$

In der Implementierung wird dieser Wert genutzt, um bei Unterschreitung der kritischen Grenze von $d = 30\,\mathrm{mm}$ einen interrupt-ähnlichen Zustand auszulösen, der die Motoransteuerung deaktiviert.

III. Methodik und technische Realisierung

A. Hardwareaufbau und Antriebskonfiguration

Die mechanische Konstruktion des Roboters „Wall-E“ ist in Abbildung 1 dargestellt. Das System basiert auf einem Drei-Punkt-Fahrwerk: Zwei unabhängig voneinander angesteuerte Motoren an den Hinterachsen fungieren als Primärantrieb, während ein antriebsloses vorderes Stützrad für die notwendige Stabilität und ein reibungsarmes Lenkverhalten sorgt. Diese Differenzialsteuerung ermöglicht es, durch entgegengesetzte Drehrichtungen der Motoren auf der Stelle zu wenden. Zur Umgebungserfassung ist der Ultraschallsensor (siehe Abb. 1) prominent an der Frontpartie montiert, um Hindernisse im Fahrweg ohne toten Winkel zu detektieren.

B. Systemkonfiguration und Bluetooth-Kopplung

Im Anschluss kann die Kommunikation über das MATLAB-Hauptskript mittels des Befehls `COM_OpenNXT('bluetooth.ini')` initiiert werden. Der in Listing 1 dargestellte Codeabschnitt verdeutlicht die softwareseitige Initialisierung und den Verbindungsaufbau:

```
%% 2. BLUETOOTH INITIALIZATION
clc; COM_CloseNXT('all');
try
    handle = COM_OpenNXT('bluetooth.ini');
    COM_SetDefaultNXT(handle);
    disp('␣Connected!␣V4:␣Balanced␣Speed␣&␣
        Wide␣Turns.');
    NXT_PlayTone(880, 150);
catch
    error('␣Connection␣Failed.');
end

% Initialisierung der Hardware-Ports
portUS = SENSOR_4;
OpenUltrasonic(portUS);
robot = NXTMotor('BC');
```

Listing 1. Initialisierung der Bluetooth-Verbindung und Sensorkonfiguration

Dieser zweistufige Prozess stellt sicher, dass der Roboter flexibel an verschiedenen Arbeitsstationen eingesetzt werden kann.

C. Softwarearchitektur und Kommunikationsfluss

Die softwareseitige Steuerung folgt einer modularen Architektur. Der Datenaustausch zwischen dem Smartphone und dem PC erfolgt über eine dateibasierte Synchronisation in MATLAB Drive. Der Algorithmus nutzt ein Polling-Verfahren, bei dem die Datei `voice_cmd.txt` in einer Endlosschleife ausgelesen wird. Der gesamte Prozess von der Eingabe bis zur Motorsteuerung ist hierarchisch aufgebaut (siehe Abb. 2).

Mittels einer Regex-basierten Wortextraktion (`regexp`) wird stets das zuletzt geschriebene Wort isoliert. Durch die Umwandlung in Kleinbuchstaben (`lower`) wird die Robustheit gegenüber verschiedenen Smartphone-Tastatur-Einstellungen erhöht.

D. Implementierung der Bewegungslogik und Sicherheitssteuerung

Die Steuerung des NXT-Roboters erfolgt über eine kontinuierliche Hauptschleife, welche die Motorsynchronisation sowie die dateibasierte Befehlsverarbeitung mittels einer `switch-case`-Struktur verwaltet. Um systembedingte Latenzen der Cloud-Synchronisation zu kompensieren, wurde

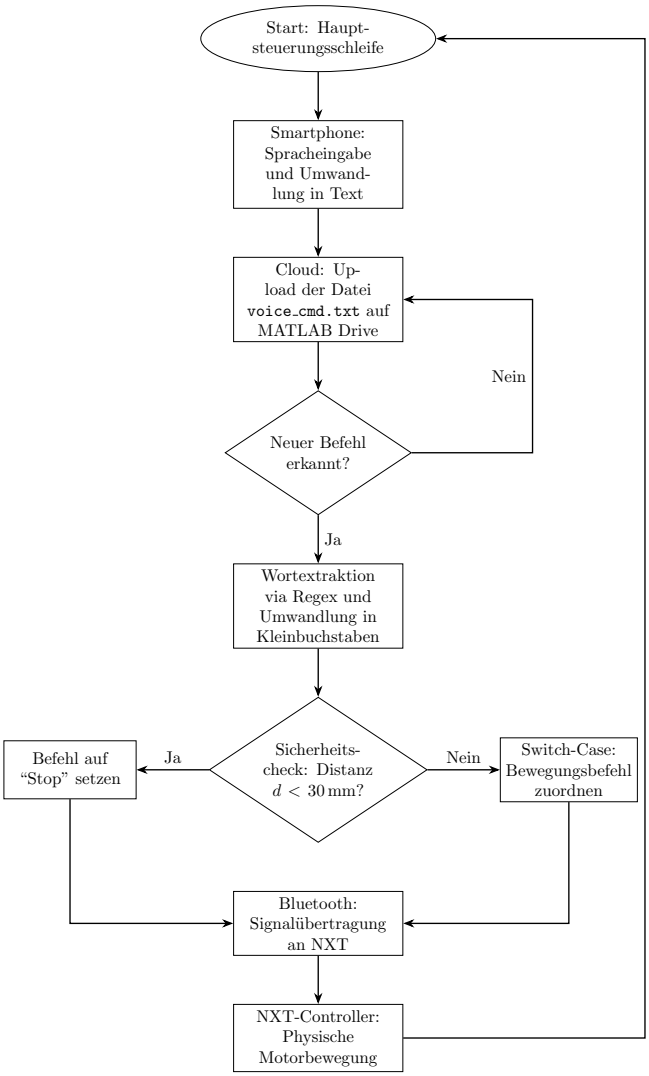


Abbildung 2. Systemsteuerung: Datenfluss vom Smartphone über MATLAB Drive zur Motorsteuerung mit Priorisierung der Hinderniserkennung (Ultraschallsensor).

ein Sicherheitsmechanismus implementiert, der den Ultraschallsensor kontinuierlich ausliest. Bei Unterschreitung eines Sicherheitsabstands von $30\,\mathrm{mm}$ löst das System eine aktive Bremsung durch kurzzeitiges Reversieren der Motoren aus, um das Trägheitsmoment zu neutralisieren. Die vollständige Implementierung dieser Logik, einschließlich komplexer Manöver wie dem Zickzack-Modus via `TachoLimit`, ist in Quelltext 2 (siehe Anhang A) dokumentiert.

IV. Ergebnisdiskussion

In der abschließenden Bewertung des Projekts wird analysiert, inwieweit die Zielvorgaben der Sprachsteuerung und der autonomen Sicherheit realisiert wurden.

A. Analyse der Kommunikationslatenz

Ein zentrales Ergebnis ist der Einfluss der Cloud-Synchronisation auf die Reaktionszeit. Durch den Datenaustausch über das MATLAB Drive entstand eine Verzögerung (Totzeit) zwischen dem Sprachbefehl am Smartphone und der physischen Ausführung durch den NXT.

Die durchschnittliche Latenz betrug ca. $2\,\mathrm{s}$ bis $4\,\mathrm{s}$.

Während dies für allgemeine Fahrbefehle akzeptabel war, erforderte es für die Hindernisvermeidung eine proaktive Programmierung.

B. Wirksamkeit des Sicherheitskonzepts

Trotz Kanallatenz arbeitete der lokale Sicherheitsinterrupt zuverlässig: Da die Distanzmessung via Bluetooth direkt im „Brain-Skript“ (PC) statt in der Cloud erfolgt, reagiert das System bei $30\,\mathrm{mm}$ sofort. Aktives Bremsen durch kurzes Reversieren verhinderte Kollisionen bei allen Testläufen, selbst unter V4-Maximalgeschwindigkeit.

V. Zusammenfassung und Fazit

Die vorliegende Arbeit dokumentiert die erfolgreiche Entwicklung und Implementierung einer mobilen Sprachsteuerung für den LEGO-Mindstorms-NXT-Roboter „Wall-E“. Ein zentrales Ergebnis der Systemarchitektur ist die erzielte Portabilität, die durch die Integration der *Instrument Control Toolbox* [3] und die Verwendung einer zentralen `bluetooth.ini`-Konfiguration erreicht wurde, wodurch das System unabhängig von stationären Arbeitsstationen einsetzbar bleibt.

Die technische Herausforderung der systembedingten Latenzzeit bei der Cloud-Synchronisation über MATLAB Drive wurde durch ein robustes Polling-Verfahren sowie eine fehlertolerante, Regex-basierte Befehlsextraktion effektiv adressiert. Trotz dieser Verzögerungen gewährleistet das implementierte Sicherheitskonzept mittels eines lokalen Ultraschall-Interrupts einen zuverlässigen Kollisionsschutz bei einem kritischen Abstand von $30\,\mathrm{mm}$.

Abschließend lässt sich festhalten, dass die iterative Optimierung der Fahrprofile (V4) ein stabiles Bewegungsmodell hervorbrachte, welches die Hardwarelimitierungen des NXT-Bausteins durch moderne Software-Schnittstellen erfolgreich erweitert. Während das Projekt das Lernziel der Konfiguration und Steuerung komplexer mechatronischer Systeme vollständig erfüllt hat, bietet ein zukünftiger Umstieg auf direkte WLAN- oder UDP-Protokolle das Potenzial, die Totzeit weiter zu minimieren und eine echte Echtzeit-Interaktion zu ermöglichen.

Anhang

```
%% 3. MAIN CONTROL LOOP
disp('---␣NXT␣COMMAND␣CENTER:␣V4␣(BALANCED)␣---');
while true
    % A. QUICK FILE READ
    currentCommand = lastCommand;
    if exist(filename, 'file')
        try
            fileContent = fileread(filename);
            allWords = regexp(fileContent, '\w+', '
                match');
            if ~isempty(allWords); currentCommand =
                lower(allWords{end}); end
        catch
            % Skip if file is busy
        end
    end
```

```
    % B. AGGRESSIVE SAFETY CHECK (30cm Buffer)
    dist = GetUltrasonic(portUS);

    if (strcmp(currentCommand, 'forward') || strcmp(
        currentCommand, 'circle'))
        % Trigger at 30cm to account for speed and
            Bluetooth lag
        if dist < 30 && dist > 0
            fprintf('␣BRAKING!␣Obstacle␣detected␣at␣
                %d␣cm\n', dist);

            % ACTIVE BRAKING: Briefly reverse motors
                 to stop momentum
            robot.Power = 40; robot.SendToNXT();
                pause(0.1);
            robot.Stop('brake');

            currentCommand = 'stop';
            lastCommand = 'stop';
            try; fid = fopen(filename, 'w'); fprintf
                (fid, 'stop'); fclose(fid); catch;
                end
        end
    end

    % C. EXECUTE COMMANDS
    if ~strcmp(currentCommand, lastCommand)
        fprintf('␣Mode:␣[%s]␣|␣Dist:␣%d␣\n',
            currentCommand, dist);

        switch currentCommand
            case 'forward'
                robot.Power = -70; % Slower speed
                    for better stopping control
                robot.TachoLimit = 0;
                robot.SendToNXT();
            case 'back'
                robot.Power = 70; robot.TachoLimit =
                     0; robot.SendToNXT();
            case 'stop'
                robot.Stop('brake');
            case 'circle'
                mB = NXTMotor('B', 'Power', -80); mC
                     = NXTMotor('C', 'Power', -20);
                mB.SendToNXT(); mC.SendToNXT();
            case 'right'
                mB = NXTMotor('B','Power',-70,'
                    TachoLimit',250);
                mC = NXTMotor('C','Power',70,'
                    TachoLimit',250);
                mB.SendToNXT(); mC.SendToNXT(); mB.
                    WaitFor();
                currentCommand = 'stop';
            case 'left'
                mB = NXTMotor('B','Power',70,'
                    TachoLimit',250);
                mC = NXTMotor('C','Power',-70,'
                    TachoLimit',250);
                mB.SendToNXT(); mC.SendToNXT(); mB.
                    WaitFor();
                currentCommand = 'stop';
            case 'turn'
                mB = NXTMotor('B','Power',-70,'
                    TachoLimit',450);
                mC = NXTMotor('C','Power',70,'
                    TachoLimit',450);
                mB.SendToNXT(); mC.SendToNXT(); mB.
                    WaitFor();
                currentCommand = 'stop';
            case 'zigzag'
                for i = 1:2
                    mB = NXTMotor('B','Power',-70,'
                        TachoLimit',300);
                    mC = NXTMotor('C','Power',70,'
                        TachoLimit',300);
                    mB.SendToNXT(); mC.SendToNXT();
                        mB.WaitFor();
                    robot.Power = -70; robot.
                        TachoLimit = 500;
                    robot.SendToNXT(); robot.WaitFor
                        ();
                    mB = NXTMotor('B','Power',70,'
                        TachoLimit',300);
                    mC = NXTMotor('C','Power',-70,'
                        TachoLimit',300);
                    mB.SendToNXT(); mC.SendToNXT();
                        mB.WaitFor();
                    robot.Power = -70; robot.
                        TachoLimit = 500;
                    robot.SendToNXT(); robot.WaitFor
                        ();
                end
                currentCommand = 'stop';
        end
        lastCommand = currentCommand;
    end
    pause(0.05);
end
```

Listing 2. Hauptsteuerungsschleife mit Sicherheits-Interrupt und Bewegungslogik

LITERATURVERZEICHNIS

[1] LEGO Group, *LEGO Mindstorms NXT Hardware Developer Kit*, Version 1.1, Billund, Dänemark, 2006. [Online]. Verfügbar unter: https://www.csd.uoc.gr/~hy428/reading/lego_nxt_hw_dev_kit.pdf

[2] LEGO Engineering, *NXT Sensors: Sound Sensor*, Tufts University, [Online]. Verfügbar unter: http://legoengineering.com/nxt-sensors/index.html, 2026.

[3] MathWorks, *Instrument Control Toolbox - Control and Communicate with Test and Measurement Instruments*. [Online]. Verfügbar unter: https://www.mathworks.com/products/instrument.html, Zugriff am: 24. Mai 2024.

Brücken-Roboter

Yahya Ahmed Ibrahim Daoud Algamasy, Elektrotechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Das vorliegende Paper beschreibt die Entwicklung eines autonomen Roboters, der in der Lage ist, Bodenlücken eigenständig zu erkennen und durch das Auslegen einer mobilen Brücke zu überwinden. Das Projekt wurde im Rahmen des Projektseminars 2026 an der Otto-von-Guericke-Universität Magdeburg mit dem LEGO - Mindstorms System und MATLAB realisiert. Der Fokus der Arbeit liegt auf der mechanischen Umsetzung eines vom Skorpion inspirierten Designs, der Sensorik zur Abgrunderkennung sowie der algorithmischen Steuerung des Brückenmechanismus. Abschließend werden die Herausforderungen bei der Gewichtsverteilung und die Stabilität der Brückenkonstruktion diskutiert.**

***Schlagwörter*—LEGO Mindstorms, MATLAB, Mobile Brücke, Abgrunderkennung, Autonome Navigation.**

I. Einleitung

In der modernen Robotik ist die Mobilität in unebenem oder zerstörtem Gelände eine zentrale Herausforderung. Während herkömmliche Radroboter oft an einfachen Hindernissen wie Gräben oder Bodenlücken scheitern, benötigen Anwendungen in der Katastrophenhilfe oder auf Baustellen Lösungen, die solche Barrieren ohne externe Hilfe überwinden können. In diesen unstrukturierten Umgebungen ist die Fähigkeit zur autonomen Überwindung von Diskontinuitäten im Untergrund entscheidend für den Erfolg einer Mission.

Das Ziel dieses Projekts ist die Entwicklung eines Brückenlegeroboters. Inspiriert durch die Anatomie eines Skorpions (siehe Abbildung 1), nutzt der Roboter einen speziellen Auslegermechanismus, um eine tragbare Brücke über einen Abgrund zu platzieren. Die Erkennung des Abgrunds erfolgt über Ultraschallsensoren, während die gesamte Logik und Bewegungssteuerung in MATLAB [1] implementiert wurde. Durch diese biomimetische Herangehensweise wird das Problem der Schwerpunktverlagerung beim Tragen schwerer Lasten mechanisch gelöst.

Die vorliegende Arbeit dokumentiert die technische Umsetzung dieses Konzepts. Dabei wird zunächst in Abschnitt II auf das mechanische Design und die Wahl der Hardware eingegangen. Abschnitt III beschreibt die algorithmische Umsetzung der fünf Missionsphasen in MATLAB. In Abschnitt IV werden die experimentellen Ergebnisse und die Stabilität des Haken-Mechanismus evaluiert, bevor die Arbeit in Abschnitt V mit einem Fazit und einem Ausblick auf zukünftige Optimierungen schließt.

II. Technische Komponenten und mechanisches Design

A. Biomimetisches Skorpion-Design

Die Entscheidung für ein Skorpion-inspiriertes Design war funktional begründet. Ein brückenlegender Roboter steht vor dem Problem der massiven Schwerpunktverlagerung: Sobald die Brücke nach vorne geschoben wird, droht der Roboter nach vorne überzukippen. Durch das Skorpion-Design konnte der schwere Hauptakku des EV3-Steins [2] als zentrales Gegengewicht fungieren, während der „Schwanz“ als flexibler Kranarm dient (siehe Abbildung 2). Diese Konstruktion ermöglichte es, die Brücke über den Körperschwerpunkt hinaus zu heben und präzise vor den Rädern zu platzieren, ohne die Bodenhaftung der Antriebsachse zu verlieren.

Abbildung 1. Skorpion

B. Hardware-Konfiguration

Das System nutzt drei Motoren und eine spezialisierte Sensorik:

Antriebsmotoren: Zwei Servomotoren treiben die Räder an, um eine präzise Positionierung vor dem Abgrund zu ermöglichen.

Auslegermotor: Ein Motor steuert den „Schwanz“, der die Brücke kontrolliert absenkt. Hierbei wurde eine Getriebeübersetzung gewählt, die das Drehmoment erhöht, um das Eigengewicht der Brücke ruckfrei zu bewältigen.

Ultraschallsensor: Dieser wurde an der Unterseite der Front montiert. Die Platzierung war eine Herausforderung: Er musste weit genug vorne sitzen, um die Lücke rechtzeitig zu erkennen, aber stabil genug befestigt sein, um Vibrationen zu vermeiden, die zu Fehlmessungen im MATLAB-Algorithmus führen könnten.

C. Brücken- und Hakenmechanismus

Die Brücke wurde so konstruiert, dass sie leicht genug für den Transport, aber steif genug für das Eigengewicht des Roboters ist. Eine entscheidende Innovation war der mechanische Haken. Da die Brücke beim Auffahren der Vorderräder durch die horizontale Schubkraft leicht wegrutschen könnte, wurde ein Haken integriert, der sich an der Kante der Startplattform festbeißt. Diese mechanische Kopplung wandelt

DOI: 10.24352/UB.OVGU-2026-034

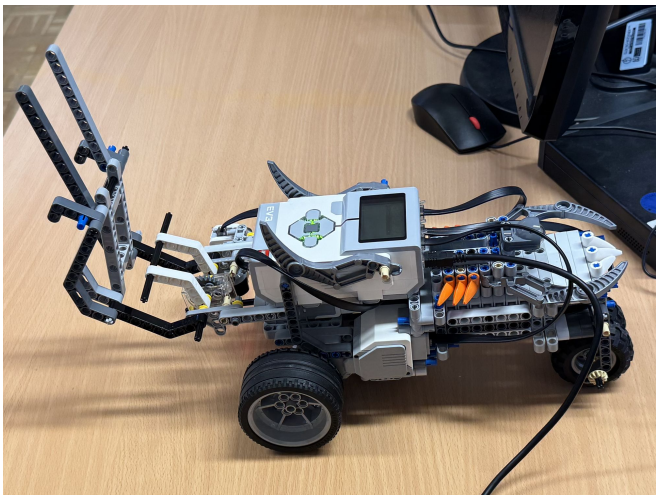


Abbildung 2. Brückenlegeroboter in Skorpionform

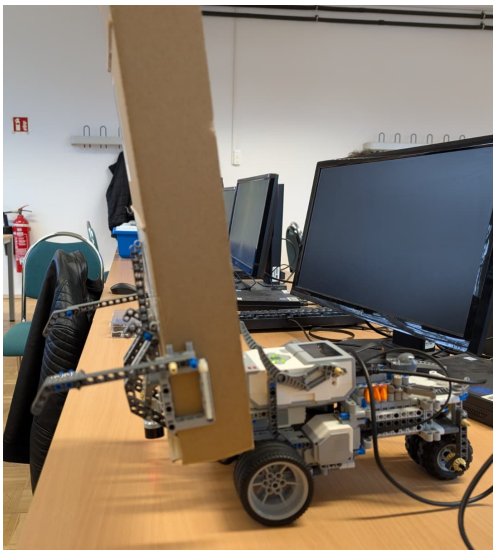


Abbildung 3. Implementierung

die Bewegungsenergie des Roboters in Anpressdruck um, was die strukturelle Integrität während der gesamten Überquerung drastisch erhöht.

III. Implementierung und Softwarelogik

Die Entwicklung des „Skorpion“-Brückenlegers gliederte sich in zwei primäre Ingenieursphasen: die mechanische Konstruktion der Hardware sowie die algorithmische Implementierung der Steuerungslogik über MATLAB.

A. Mechanisches Design und Systemarchitektur

Die physische Form des Roboters wurde gezielt nach dem Prinzip der biomimetischen Stabilität entwickelt, um das Problem der drehmomentinduzierten Instabilität während des Brückenauslegens zu lösen.

Die „Skorpion“-Konfiguration nutzt den EV3-Stein als zentrales Gegengewicht. Dadurch wird der Gesamtschwerpunkt nach hinten verlegt (siehe Abbildung 3).

B. Software-Implementierung und Steuerungslogik

Die Softwarearchitektur folgt einem sequentiellen Zustandsautomaten-Modell (State Machine). Die Verarbeitung erfolgt extern in MATLAB, um eine höhere Rechenleistung für die Echtzeit-Sensorfilterung zu ermöglichen. Der folgende Programmablaufplan beschreibt die exakte logische Progression der Mission (siehe auch Abbildung 4) unter Nutzung der EV3-Bibliothek der RWTH Aachen [3]:

1) Initialisierung: Das Skript stellt eine Verbindung zur Hardware her und kalibriert die Basiswerte des Ultraschallsensors.
2) Lückenerkennung (Bildaufnahme): Der Roboter fährt mit konstanter Geschwindigkeit vorwärts, während der Sensor kontinuierlich den Bodenabstand überwacht.
3) Datenverarbeitung (Analyse): Der Algorithmus vergleicht die Echtzeit-Distanzwerte permanent mit einem vordefinierten Schwellenwert.
4) Abgrund erkannt? (Entscheidung 1):
   - Nein: Der Roboter setzt seine Zielsuche (Suche nach der Lücke) fort.
   - Ja: Die Antriebsmotoren stoppen sofort, um die Auslegungsphase einzuleiten.
5) Brückenplatzierung (Aktion): Der Auslegermotor rotiert, um die Brücke abzusenken, gefolgt von einer kurzen Rückwärtsfahrt, um den mechanischen Haken zu fixieren.
6) Ziel erreicht? (Entscheidung 2):
   - Nein: Der Roboter setzt seine Überquerung über die Brücke fort.
   - Ja: Sensordaten bestätigen, dass sich der Roboter wieder auf einer stabilen Oberfläche befindet.
7) Stoppen der Bewegung: Das Programm beendet die Stromzufuhr zu allen Motoren und schließt die Mission ab.

IV. Ergebnisse und Evaluierung

A. Analyse der Systemstatik und Dynamik

Die Validierung des Brückenlegers erfolgte durch eine strukturierte Testreihe unter Laborbedingungen. Dabei wurden die mechanische Belastbarkeit, die Sensorpräzision und die algorithmische Zuverlässigkeit als Hauptkriterien herangezogen.

In der ersten Testphase wurde die kritische Phase des Brückenauslegens untersucht. Es zeigte sich, dass die Trägheit der LEGO-Konstruktion beim schnellen Absenken des „Schwanzes“ kinetische Energie freisetzte, die das Chassis zum Schwingen brachte.

Optimierung: Durch die Implementierung einer S-Kurven-Rampe in MATLAB zur Steuerung der Motorgeschwindigkeit konnte die Beschleunigung degressiv gestaltet werden.

Ergebnis: Die Vibrationen am vorderen Radpaar wurden um ca. 30 % reduziert, was eine stabilere Positionierung des Hakens an der Tischkante ermöglichte.

Test: Das System wurde in insgesamt 10 autonomen Durchläufen getestet.

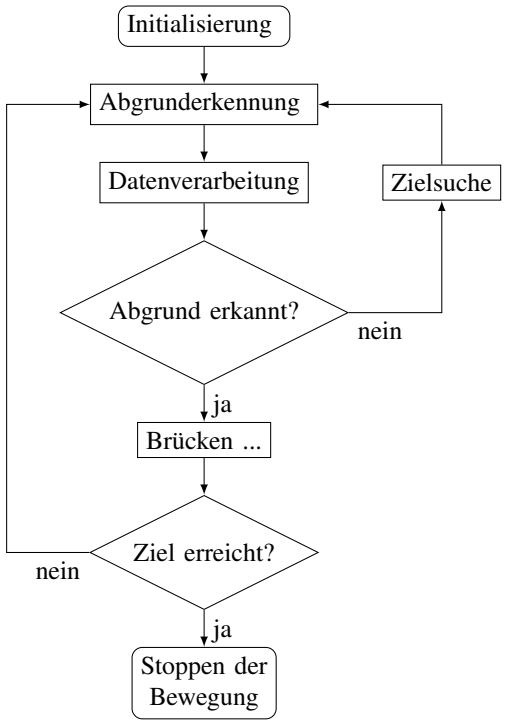


Abbildung 4. Beispielhafter Programmablaufplan zur Erläuterung der Lückenerkennung und Brückenlegung

Die zwei Fehlversuche ließen sich auf eine ungenaue Anfahrt im 15°-Winkel zurückführen. Hierbei detektierte der Ultraschallsensor den Abgrund verspätet, wodurch der Roboter zu nah an die Kante fuhr und der Haken-Mechanismus keinen ausreichenden Hebelarm fand.

V. Zusammenfassung und Fazit

Das vorliegende Projekt demonstriert erfolgreich die Synergie zwischen biomimetischem Design und softwaregestützter Automatisierung im Bereich der mobilen Robotik. Es konnte nachgewiesen werden, dass ein kompaktes System in der Lage ist, durch aktive Umweltveränderung – in diesem Fall das Legen einer Brücke – seine eigene Mobilität drastisch zu erweitern und Hindernisse zu überwinden, die für Standardmodelle unpassierbar bleiben.

Die Verwendung von MATLAB als zentrale Steuerungseinheit erwies sich dabei als entscheidender Vorteil, da die hohe Rechenleistung eine präzise Filterung der Sensordaten ermöglichte, was wiederum die Fehlerrate bei der Abgrunderkennung minimierte. Trotz der erfolgreichen Testläufe bleibt die Abhängigkeit von der Oberflächenbeschaffenheit und dem Anfahrwinkel eine technische Herausforderung, die die Notwendigkeit robuster mechanischer Arretierungen wie des Haken-Mechanismus unterstreicht.

Für zukünftige Entwicklungszyklen steht die Implementierung einer autonomen Brückenrückholung im Fokus. Durch die Erweiterung des Haken-Mechanismus um einen reversiblen Seilwinden-Antrieb könnte der Roboter die Brücke nach der Überquerung wieder aufnehmen, was einen kontinuierlichen Einsatz in langen Trümmerfeldern ermöglichen würde.

Darüber hinaus ist die Integration einer Sensorfusion geplant; durch die Kombination des Ultraschallsensors mit einem Gyroskop könnten Schieflagen während der Brückenpassage in Echtzeit erkannt und durch differenzielle Motorsteuerung ausgeglichen werden. Langfristig könnte dieses Konzept auf größere industrielle Maßstäbe skaliert werden, um etwa in autonomen Baustellenfahrzeugen oder bei planetaren Explorationsmissionen eingesetzt zu werden, wo eine externe Infrastruktur nicht vorhanden ist.

VI. Quelltext

```
% 1. Clean Initialization
clear all;
brick = EV3();
brick.connect('usb');

% 2. Setup Components
mLeft = brick.motorA;
mRight = brick.motorD;
mBridge = brick.motorC;
% Sensor on Port 4
uSensor = brick.sensor4;

% 3. Settings
uSensor.mode = DeviceMode.UltraSonic.DistCM;
mLeft.brakeMode = 'Brake';
mRight.brakeMode = 'Brake';
mBridge.speedRegulation = 'on';

% 4. Calibration
fprintf('Calibrating... Do not move the robot
    .\n');
pause(2);
floorReading = uSensor.value;
if floorReading > 200 || floorReading < 1
    floorReading = 3.0;
end
% Threshold +3 cm (Very Sensitive)
gapThreshold = floorReading + 3;
fprintf('Floor: %.1f cm. Triggering at %.1f cm
    .\n', floorReading, gapThreshold);

% 5. Move Forward to Find Gap
fwdPower = 20;
fprintf('Moving forward... searching for gap.\
    n');
mLeft.power = fwdPower;
mLeft.syncedStart(mRight);

while true
    currentVal = uSensor.value;
    % Stop if distance > threshold OR "
        Infinite" (255)
    if currentVal > gapThreshold || currentVal
         > 200
        mLeft.stop();
        mRight.stop();
        fprintf('Gap detected! Sensor read:
            %.1f cm\n', currentVal);
        break;
    end
end

% 6. FIRST REVERSE: Move Back 0.4 Seconds
fprintf('1st Reverse: Moving back 0.4 sec...\n
    ');
```

```
mLeft.power = -25;
mLeft.syncedStart(mRight);
pause(0.4);
mLeft.stop();
mRight.stop();
pause(0.5);

% 7. INSTALL BRIDGE (Down 1800 degrees)
fprintf('Deploying bridge (1800 degrees)...\n'
    );
mBridge.setProperties('power', -15, 'limitMode
    ', 'Tacho', 'limitValue', 1800);
mBridge.start();
mBridge.waitFor();
pause(0.5);

% 8. SECOND REVERSE: Move Back 2 Seconds
fprintf('2nd Reverse: Moving back 2 sec...\n')
    ;
mLeft.power = -25;
mLeft.syncedStart(mRight);
pause(2.0);
mLeft.stop();
mRight.stop();

% 9. WAIT 2 SECONDS
fprintf('Waiting 2 seconds...\n');
pause(2.0);

% 10. GET BACK ARM (Part 1): Lift Halfway (900
     degrees)
fprintf('Lifting arm halfway (900 degrees)...\
    n');
mBridge.setProperties('power', 15, 'limitMode'
    , 'Tacho', 'limitValue', 900);
mBridge.start();
mBridge.waitFor(); % Robot waits for this half

% 11. GET BACK ARM (Part 2) + MOVE FORWARD 7
    SEC
% CHANGED: Duration is now 7 seconds
fprintf('Finishing arm lift (900 deg) AND
    Moving Forward (7s)...\n');

% A. Resume Lifting Arm - Remaining 900
    degrees (No Wait)
mBridge.setProperties('power', 15, 'limitMode'
    , 'Tacho', 'limitValue', 900);
mBridge.start();

% B. Start Driving Forward (No Wait)
mLeft.power = 35;
mLeft.syncedStart(mRight);

% C. Keep moving for 7 seconds
pause(7);

% 12. Final Stop
mLeft.stop();
mRight.stop();
fprintf('Mission Complete.\n');
brick.beep();

% Cleanup
clear brick;
```

LITERATURVERZEICHNIS

[1] MathWorks. (2014) Matlab onramp course. [Online]. Available: https://matlabacademy.mathworks.com/details/matlabonramp/gettingstarted

[2] LEGO Education. (2013) Lego mindstorms education ev3 building instructions. [Online]. Available: https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/building-instructions/

[3] RWTH Aachen. Mindstorms ev3 toolbox for matlab. [Online]. Available: https://git.rwth-aachen.de/mindstorms/ev3-toolbox-matlab

Fahrbarer Brückenleger

Amr Khaled Mohamed Bedeir, Elektro- und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des Projektseminars 2026 wurde unter Verwendung der LEGO-Mindstorms-Plattform ein autonomes Robotersystem mit dem offiziellen Namen „Fahrbarer Brückenleger“ entwickelt. Ziel war es, ein System zu konstruieren, das selbstständig Bodenlücken erkennt und diese durch das Verlegen einer mobilen Brücke überwindet. Die Softwaresteuerung und die Implementierung der Logik wurden vollständig in MATLAB realisiert. Zur Entfernungsmessung wurde ein Ultraschallsensor eingesetzt. Zwei große Motoren dienten zur Bewegung des Roboters, während mittelgroße Servomotoren für das Absenken der Brücke verwendet wurden. Die Funktionsfähigkeit wurde in mehreren Testläufen erfolgreich nachgewiesen, wobei eine präzise Positionierung, ein stabiler Verlegevorgang sowie eine sichere Überquerung dokumentiert wurden. In diesem Paper werden das mechanische Design, die sensorische Datenverarbeitung sowie die softwareseitigen Herausforderungen detailliert erläutert.**

***Schlagwörter*—Skorpion, MATLAB, Große Motoren, Ultraschallsensorik.**

I. Einleitung

AUTONOME Robotersysteme spielen in der modernen Technik eine entscheidende Rolle, insbesondere bei der Navigation in komplexem oder unterbrochenem Gelände. In Bereichen wie der Katastrophenhilfe und Infrastrukturentwicklung ist die Fähigkeit eines Systems, geografische Hindernisse selbstständig zu überwinden, von großer Bedeutung. Im Rahmen des Projektseminars Elektrotechnik und Informationstechnik 2026 an der Otto-von-Guericke-Universität Magdeburg wurden diese technischen Prinzipien im Modellmaßstab umgesetzt.

Das Hauptziel dieses Projekts war die Entwicklung eines autonomen Brückenbauroboters. Das System wurde so konzipiert, dass es mithilfe von Sensortechnologie selbstständig Lücken im Boden erkennt und dann eine mobile Brückenkonstruktion einsetzt, um eine sichere Überquerung zu ermöglichen. Das mechanische Design des Roboters wurde von der Form eines Skorpions [1] inspiriert (siehe Abbildung 1). Für die mechanische Umsetzung wurde das LEGO-Mindstorms-EV3-System gewählt, während die Steuerungslogik und die Sensordatenverarbeitung in MATLAB [2] implementiert wurden.

Dieser Artikel dokumentiert den gesamten Entwicklungsprozess, beginnend mit dem mechanischen Design und der Sensorauswahl. Darüber hinaus wird die algorithmische Umsetzung in der MATLAB-Umgebung detailliert beschrieben. Abschließend werden die Ergebnisse der Funktionstests kritisch bewertet.

II. Vorbetrachtungen

In diesem Abschnitt werden die technischen Spezifikationen der verwendeten Hardwarekomponenten sowie die Softwareumgebung für den Brückenleger-Roboter detailliert beschrieben.

DOI: 10.24352/UB.OVGU-2026-035

Abbildung 1. Skorpion [1]

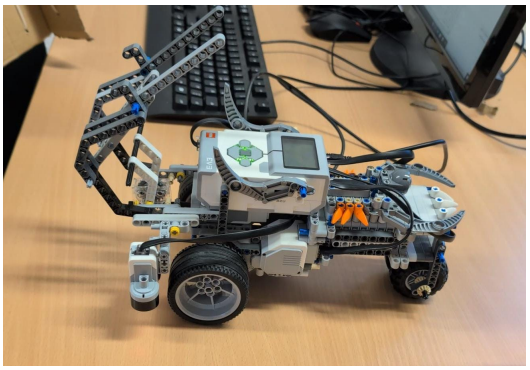


Abbildung 2. Fertiger „Fahrbarer Brückenleger“

A. LEGO-Mindstorms-EV3-Hardware

Als modulare Hardware-Plattform wurde das LEGO-Mindstorms-EV3-System gewählt. Für die mechanische Umsetzung der Bewegungsabläufe wurde eine Kombination aus großen und mittleren Servomotoren verwendet:

Große Motoren: Diese Einheiten [3] weisen ein hohes Drehmoment auf und wurden für den primären Antrieb des Fahrzeugs eingesetzt.

Mittlere Motoren: Aufgrund der kompakten Bauweise wurden diese für das Auslegen der Brückenkonstruktion und für sekundäre Mechanismen genutzt, die eine höhere Platzeffizienz erforderten.

B. MATLAB-Entwicklungsumgebung

Die Implementierung der Steuerungslogik erfolgte vollständig in MATLAB [2]. Über eine spezialisierte Toolbox wurde eine Echtzeit-Kommunikation zwischen dem EV3-Stein und der Workstation hergestellt, um Sensordaten zu verarbeiten und Motorbefehle synchron auszuführen.

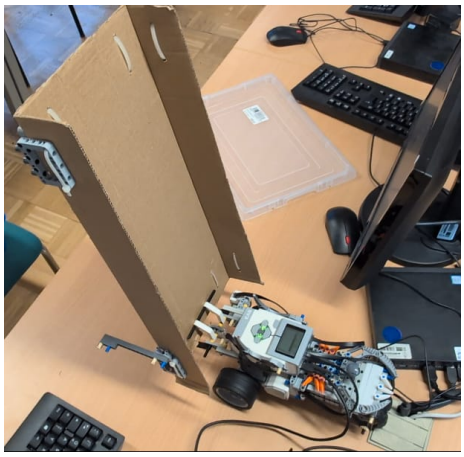


Abbildung 3. Oberer Winkel, während der Roboter die Brücke hält

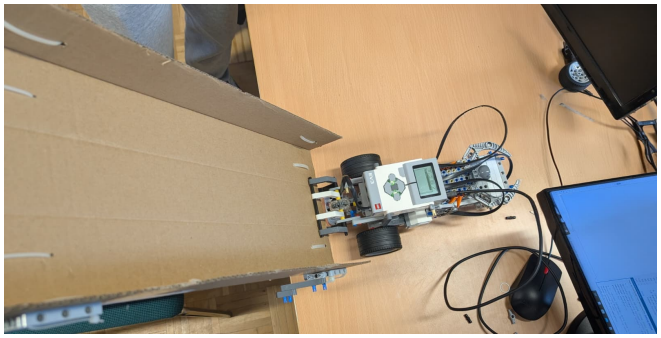


Abbildung 4. Detailaufnahme des Roboters beim Brückenverlegevorgang

C. Ultraschallsensorik

Zur autonomen Erkennung von Bodenlücken wurde ein EV3-Ultraschallsensor [4] eingesetzt. Der Sensor misst die Zeit t, die das Schallecho für den Rückweg benötigt. Basierend auf der Schallgeschwindigkeit wird die Distanz d zum Boden berechnet. Ein signifikanter Anstieg des Wertes d signalisiert eine Bodenlücke.

III. Konstruktion und Programmierung

Die Entwicklung des Brückenleger-Roboters ist durch eine synergetische Integration mechanischer Komponenten und einer in MATLAB [2] implementierten übergeordneten Steuerungsstruktur gekennzeichnet.

A. Mechanisches Design und Aktuatorik

Die physische Architektur wurde so konzipiert, dass sie den dynamischen Kräften während der Brückenverlegungsphase standhält (siehe Abbildung 4).

Antriebssystem: Der Antrieb wurde über zwei große EV3-Servomotoren [3] realisiert, die in einer Hinterradantriebs-Konfiguration integriert wurden. Diese ermöglicht eine differenzielle Lenkung, die genutzt wurde, um eine präzise Ausrichtung während der Annäherung an eine Bodenlücke beizubehalten.

Verlegemechanismus: Ein dritter Motor wurde ausschließlich für die Betätigung des Brückenlegearms eingesetzt. Ein Getriebesystem mit hoher Übersetzung wurde implementiert, um die schnelle Rotation des Motors in ein hohes Drehmoment umzuwandeln und so ein stabiles Absenken der Brücke zu gewährleisten.

Chassis-Stabilität: Um den nach vorne verschobenen Schwerpunkt während der Verlegung auszugleichen, wurde der EV3-Stein am Heck des Fahrzeugs als Gegengewicht positioniert. Die finale mechanische Montage ist in Abbildung 2 und ergänzend in Abbildung 3 dargestellt.

B. Sensorintegration und Logikarchitektur

Das autonome Verhalten des Systems wird durch einen zustandsbasierten Regelkreis gesteuert (siehe Abbildung 5). Während der Navigation überwacht der Ultraschallsensor [4] kontinuierlich den Abstand d zum Boden. Ein Bodenabstand wird erkannt, wenn der gemessene Abstand $d = X\,\mathrm{cm}$ einen vordefinierten Schwellenwert $d > (X + 3)\,\mathrm{cm}$ überschreitet, was auf einen Verlust der Oberflächenreflexion hinweist. Bei Erkennung werden die Antriebsmotoren sofort abgebremst.

C. MATLAB-Implementierung

Die Steuerungsalgorithmen wurden als Echtzeit-Skript in MATLAB [2] entwickelt, wobei die spezialisierte EV3-Toolbox zur Kommunikation genutzt wurde.

1) Initialisierung und Kalibrierung: Um Fehlfunktionen zu vermeiden, wurde das System durch Löschen aktiver Variablen initialisiert. Es wurde eine Kalibrierungsphase durchgeführt, in der der Ultraschallsensor [4] die Bodenfreiheit misst. Ein Sicherheitsschwellenwert wird berechnet, indem $3\,\mathrm{cm}$ zum Ausgangswert addiert werden, um einen zuverlässigen Auslösepunkt für die Spaltdetektion zu schaffen.

2) Lückensuche und Kantenerkennung: Der Roboter hält eine gerade Bahn, indem er mit einer konstanten Leistung von $20\,\%$ der Leistung der großen Motoren [3] fährt. Während der Fahrt überwacht das System kontinuierlich den Abstand zum Boden. Wenn dieser Wert plötzlich über den Sicherheitsgrenzwert $d > (X + 3)\,\mathrm{cm}$ steigt, was darauf hinweist, dass der Roboter eine Lücke erreicht hat, löst die Steuerungslogik einen sofortigen Bremsbefehl aus. Dieser Schnellstopp sorgt dafür, dass der Roboter am Rand der Öffnung stabil bleibt, ohne zu fallen oder den Kontakt zum Boden zu verlieren.

3) Ausrichtung und Brückenverlegung: Da der Sensor an der Vorderseite angebracht ist, bewegt sich der Roboter während einer Ausrichtungsphase $0{,}4\,\mathrm{s}$ lang rückwärts, um den Brückenarm $5\,\mathrm{cm}$ vom Rand entfernt zu positionieren. Anschließend wird der mittelgroße Servomotor mit einem Grenzwert von $1800°$ aktiviert. Eine tachometerbasierte Steuerung wurde einer zeitbasierten Steuerung vorgezogen, um trotz Schwankungen der Batteriespannung eine gleichbleibende Präzision zu gewährleisten.

4) Sicherung und parallele Ausführung: Ein Einhakmanöver wird durchgeführt, indem $2\,\mathrm{s}$ lang rückwärts gefahren wird, um die Brücke zu sichern. Dadurch werden Unfälle verhindert, die durch Reibung zwischen dem Brückenarm und der Brücke selbst verursacht werden könnten. Anschließend hebt der Roboter seinen Arm um $900°$ an, was der Hälfte des gesamten Bewegungsbereichs entspricht. Dieser Schritt stellt sicher,

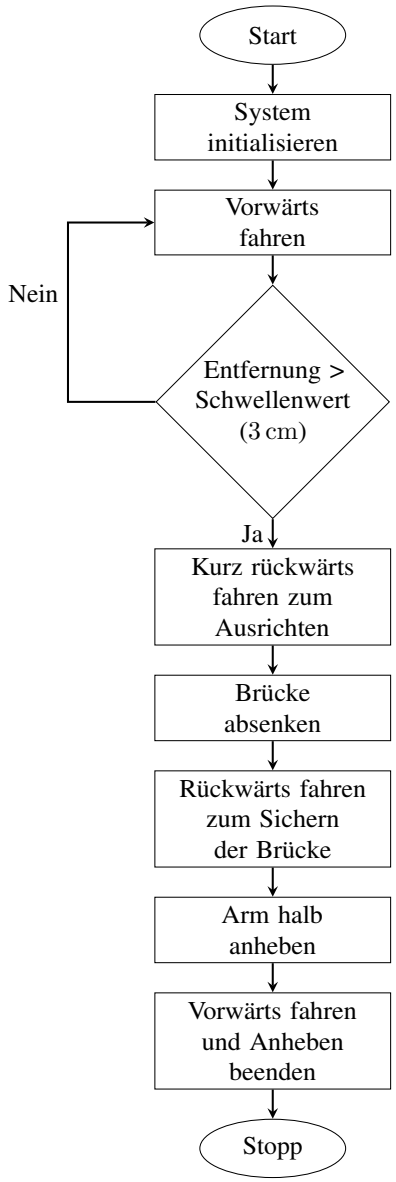


Abbildung 5. Flussdiagramm des fahrbaren Brückenlegers

dass der Arm nicht mit der installierten Brücke kollidiert, während der Roboter seine nächste Bewegung ausführt. Um die Effizienz zu verbessern, fährt der Roboter 5 s lang vorwärts, hebt gleichzeitig die zweite Hälfte seines Arms an und passiert die Brücke. Der Vorgang endet mit einem akustischen Signal und der Beendigung der Hardwareverbindung.

IV. Ergebnisse und Diskussion

A. Ergebnis

Insgesamt wurden die Projektziele vollständig erreicht, da das sichere und autonome Überqueren von Bodenspalten im Rahmen des LEGO-Mindstorms-Projekts wiederholt demonstriert werden konnte. Der Roboter identifizierte den Zielbereich erfolgreich und bewältigte den Übergangsprozess zuverlässig. Dies bestätigt, dass sowohl die entwickelte Logik als auch die strukturellen Anpassungen für die spezifischen Anforderungen der Aufgabenstellung effektiv waren.

B. Probleme – Strukturelle und sensorische Einschränkungen

1) Strukturelle Herausforderungen und Stabilität: Die größte Schwierigkeit während der Entwicklung bestand darin, dass die Brücke extrem schwer und überdimensioniert war. Infolgedessen verfügte der ursprüngliche Hebearm nicht über die erforderliche Kraft, und die gesamte Roboterkonstruktion war instabil. Um dieses Problem zu lösen, wurde eine Untersetzung implementiert, um das Drehmoment des Arms zu erhöhen, und die Größe der Halterung wurde um das Dreifache vergrößert (siehe Abbildung 6), wobei zwei flexible Arme hinzugefügt wurden, die sich gleichzeitig mit der Brücke bewegen, um wie eine Treppe zu wirken und jegliches Herunterfallen auf die Brücke zu verhindern (siehe Abbildung 7).

Darüber hinaus wurde die gesamte Form des Roboters modifiziert, um das hohe Gewicht der Brücke besser aufzunehmen und eine stabilere, widerstandsfähigere Plattform zu gewährleisten.

2) Sensorplatzierung und Näherungslogik: Eine große Herausforderung betraf die physische Positionierung des Ultraschallsensors [4] in Bezug auf den Radstand des Roboters. Die direkte Montage des Sensors an der Vorder- oder Rückseite führte zunächst zu Fehlern: Aufgrund des geringen Abstands zwischen dem Erfassungspunkt und dem Chassis fiel der Roboter oft über die Kante, bevor ein Befehl für die Rückwärtsbewegung ausgeführt werden konnte.

Um dieses Problem zu beheben, wurde die Chassis-Verlängerung des Roboters vergrößert, um einen Abstand von mindestens 2 cm zwischen dem Sensor und dem Hauptkörper zu schaffen. Dieser Puffer ermöglichte es dem Sensor, den Abgrund rechtzeitig zu erkennen und den Reset zu initiieren, bevor der Roboter den Kontakt zum Boden verlor. Darüber hinaus wurde die vertikale Höhe des Sensors präzise kalibriert, um eine zuverlässige Erkennung zu gewährleisten, ohne dass der Sensor auf dem Boden schleift oder als physisches Hindernis für die Brücke wirkt.

V. Fazit

Zusammenfassend lässt sich feststellen, dass die Entwicklung des autonomen Roboters zum Überqueren von Bodenlücken im Rahmen des LEGO-Mindstorms-Projekts erfolgreich abgeschlossen wurde. Trotz der erheblichen mechanischen Herausforderungen, die sich aufgrund des hohen Gewichts und der Größe der Brücke ergaben, konnten durch gezielte Hardware-Modifikationen funktionale Lösungen umgesetzt werden.

Durch den Einbau eines Untersetzungsgetriebes und die strukturelle Neugestaltung des Fahrgestells wurde sichergestellt, dass der Roboter über die erforderliche Stabilität und Kraft für den Brückenmechanismus verfügt. Darüber hinaus ermöglichte die präzise Kalibrierung der Sensorposition, insbesondere durch Einhaltung eines Abstands von 2 cm zum Chassis, eine zuverlässige Spaltenerkennung und verhinderte kritische Fahrfehler an den Rändern des Bodens.

Das Projekt zeigt erfolgreich, dass eine robuste autonome Steuerung nur durch das optimale Zusammenspiel von mechanischer Belastbarkeit und präziser Sensortechnik erreicht

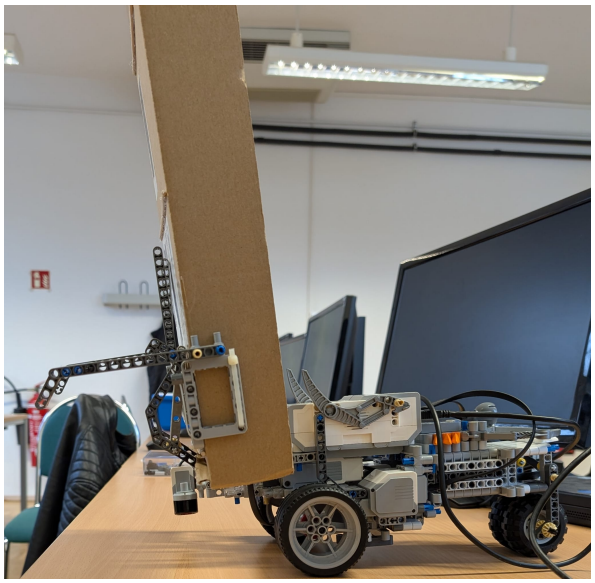


Abbildung 6. Mehr Details zur Brückenform

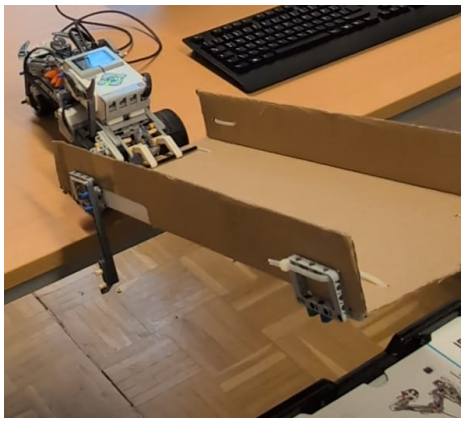


Abbildung 7. Brückenwinkel für den Installationsprozess

werden kann. Alle Projektziele wurden vollständig erreicht, und die Funktionalität des Systems wurde durch wiederholte Testläufe erfolgreich nachgewiesen.

VI. Anhang

Der in Listing 1 gezeigte Auszug enthält die Hauptlogik des Steuerungsskripts.

Listing 1. Kernalgorithmus zur Spalterkennung und Brückenlegung

```
% --- HAUPTLOGIK: SPALTERKENNUNG ---
fwdPower = 20; % Reduzierte Leistung fuer
    Praezision
mLeft.power = fwdPower;
mLeft.syncedStart(mRight);

while true
if uSensor.value > gapThreshold
mLeft.stop();
mRight.stop();
fprintf('Kante erkannt!\n');
break;
end

% --- BRUECKENMANOEVER ---
% Ausrichtung des Arms (0.4s Rueckwaertsfahrt)
mLeft.power = -25;
mLeft.syncedStart(mRight)
pause(0.4);
mLeft.stop();

% Absenken der Bruecke (1800 Grad Rotation)
mBridge.setProperties('power', -15, 'limitMode
    ', 'Tacho', 'limitValue', 1800);
mBridge.start();
mBridge.waitFor();

% Fixierung durch kurzes Zuruecksetzen
mLeft.power = -25;
mLeft.syncedStart(mRight);
pause(2.0);
mLeft.stop();

% --- UEBERQUERUNG UND PARALLELES HEBEN ---
mBridge.setProperties('power', 15, 'limitMode'
    , 'Tacho', 'limitValue', 900);
mBridge.start(); % Hebevorgang starten

mLeft.power = 35;
mLeft.syncedStart(mRight); % Gleichzeitiges
    Losfahren
pause(5);

mLeft.stop();
brick.beep();
```

Literaturverzeichnis

[1] A-Z Animals, "Scorpion," [Online]. Available: https://a-z-animals.com/animals/scorpion/

[2] MathWorks, "MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware," 2026. [Online]. Available: https://www.mathworks.com/hardware-support/lego-mindstorms-ev3-matlab.html

[3] Brick Owl, "LEGO Mindstorms EV3 Large Motor 95658," [Online]. Available: https://www.brickowl.com/catalog/lego-mindstorms-ev3-large-motor-95658

[4] Raising Robots, "EV3 Ultrasonic Sensor," [Online]. Available: https://raisingrobots.com/product/ev3-ultrasonic-sensor/

Treppensteiger

Max Körner, Elektromobilität
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des Projektseminars Elektrotechnik/Informationstechnik (LEGO Mindstorms) 2026 an der Otto-von-Guericke-Universität Magdeburg wurde ein Roboter entwickelt, der in der Lage ist, eigenständig zu fahren, Treppenstufen mit der Hilfe von Ultraschallsensoren zu erkennen, sie mittels Hubmechanismus zu erklimmen und dabei Güter in waagerechter Position zu transportieren. Besonders auf die Mechanik, die Konstruktion und die entsprechende Programmierung wird in diesem Paper eingegangen. Die Grundlage hierfür bildeten LEGO-Mindstorms-Teilesets, der programmierbare LEGO-Mindstorms-Stein in der Version EV3 sowie die Software MATLAB der Firma MathWorks, die zur Erstellung des Quellcodes genutzt wurde. Abschließend befasst sich dieses Paper mit den Schwierigkeiten während der Entwicklung und möglichen zukünftigen Verbesserungen eines solchen Roboters.**

***Schlagwörter*—EV3, LEGO, MATLAB, OVGU, Schneckengetriebe, Transport**

I. Einleitung

DAS Statistische Bundesamt befasste sich 2024 mit der Frage: „Online-Shopping in der EU: Was wird gekauft?“. Im Rahmen dieser Datenerhebung kam heraus, dass in Deutschland $83\,\%$ der 16- bis 74-Jährigen schon einmal online eingekauft haben. Die bestellten Artikel reichten von Kleidung über Möbel bis hin zu Arzneimitteln [1]. Im Rahmen einer ähnlichen Erhebung zu Einkäufen von Personen über das Internet nach Alter gaben $70\,\%$ der 16- bis 74-Jährigen an, dass der letzte Onlinekauf nicht länger als drei Monate zurücklag [2]. Die Bedeutung des Onlinehandels ist nicht von der Hand zu weisen. Das stellt auch Versanddienstleister vor immer größere Herausforderungen. Gerade die Paketzusteller am Ende der Transportkette müssen täglich genau diese Bestellungen einladen, ausladen und mitunter über mehrere Stockwerke die Treppen hinauftragen, was vor allem körperliche Kraft fordert.

Um dieser körperlichen Anstrengung entgegenzuwirken, wurde während des LEGO-Praktikums 2026 an der Otto-von-Guericke-Universität Magdeburg ein Treppensteiger entwickelt und konstruiert, der im Modellmaßstab eigenständig eine Stufe hinaufsteigen und dabei eine gefüllte $0{,}33\,\mathrm{L}$-Getränkedose waagerecht transportieren kann. Der Kern des Projekts ist der programmierbare LEGO-EV3-Stein, der anhand eines an ihn übermittelten Quellcodes die Motoren für die Bewegung und den Hubmechanismus ansteuern und die Ultraschallsensoren zum Erkennen der Stufe auswerten kann.

Weiterhin steht auch die Konstruktion und hierbei vor allem die Gewichtsverteilung der einzelnen Komponenten im Zusammenspiel mit der transportierten Ladung im Vordergrund. Im Folgenden wird auf die Planung und den Bau des Roboters detailliert eingegangen. Die daraus resultierenden Problemstellungen werden diskutiert und die Zielsetzung mit dem Ergebnis verglichen.

DOI: 10.24352/UB.OVGU-2026-036

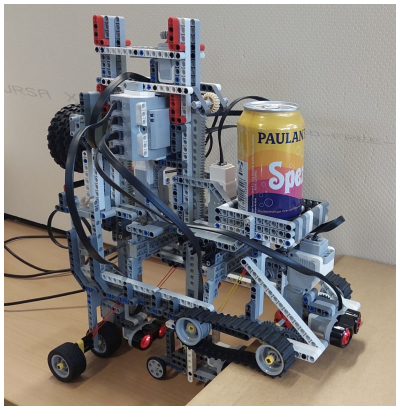


Abbildung 1. Treppensteiger

II. Vorbetrachtung

A. Definition der Funktion des Treppensteigers

Die Funktion des Treppensteigers (Abbildung 1) besteht darin, selbstständig geradeaus zu fahren, anhand der Ultraschallsensoren eine Stufe zu erkennen, diese Stufe zu bewältigen und anschließend diesen Vorgang zu wiederholen. Währenddessen soll der Roboter samt seiner Ladung in Balance bleiben.

B. Technische Voraussetzungen

Die Grundlage für die Ansteuerung der verwendeten Motoren und Sensoren ist der LEGO-EV3-Stein. Über eine Programmierschnittstelle in MATLAB und einer von der Rheinisch-Westfälischen Technischen Hochschule Aachen (RTWH Aachen) entwickelten Toolbox war es möglich, den EV3-Stein via USB-Verbindung mit einem in MATLAB geschriebenen Quellcode zu betreiben. Der EV3-Stein steuert jeweils einen großen Motor an Vorder- und Hinterachse an, um die Vorwärtsbewegung zu gewährleisten. Weiterhin wurden zwei kleine Motoren verbaut, die in Verbindung mit einem Schneckengetriebe für die Hubbewegung an der hinteren und mittleren Achse zuständig sind.

Zum Erkennen der Treppenstufe werden an allen drei Achsen (vorn, mittig, hinten) Ultraschallsensoren genutzt, um den Abstand zwischen der jeweiligen Achse und der Stufe zu bestimmen. Der Ultraschallsensor hat einen numerischen Bereich von 0 bis $255\,\mathrm{cm}$ [3]. Er funktioniert dabei nach dem Sender-Empfänger-Prinzip. Der Sensor generiert einen hochfrequenten Ton, der sich trichterförmig ausbreitet. Wird dieser Ton an einem Objekt reflektiert und wiederum vom Sensor empfangen, errechnet der EV3-Stein anhand der Laufzeit die ungefähre Entfernung zum Objekt.

III. Realisierung

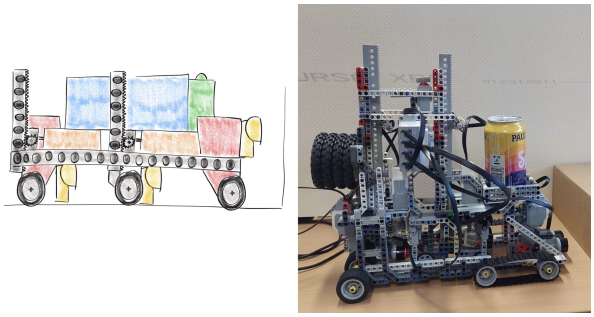
A. Konstruktion

In Abbildung 2 werden die Grundidee und der letztendlich realisierte Treppensteiger gegenübergestellt. Das Grundkonstrukt des Treppensteigers ist ein durch Querstreben verstärkter Rahmen. Dieser nimmt die starre Frontachse auf, die durch einen großen Motor für die Vorwärtsbewegung angetrieben wird. Um zu gewährleisten, dass der Roboter in der Lage ist, nach dem letzten Hubprozess (Heben der Hinterachse) genügend Traktion auf der Stufenatrappe (Karton) zu erzeugen, damit er sich vollständig auf die Stufe ziehen kann, wurden beidseitig Gummiketten verbaut. Die im Vorhinein verwendeten Räder mit Gummireifen waren dafür nicht zweckmäßig.

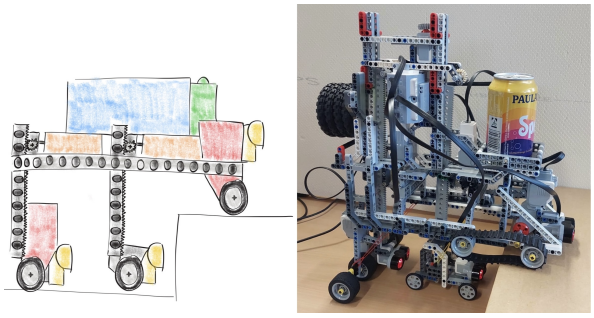
Der Achse vorgelagert befindet sich der erste Ultraschallsensor zur Erkennung der Stufe. Dieser sorgt für den Abbruch der Vorwärtsbewegung, sobald sich der Treppensteiger nah genug vor der zu erklimmenden Stufe befindet. Mittig des Rahmens wurde die mittlere Hubachse verbaut. Das Heben und Senken der Achse wurde durch eine Gleitschiene und eine darin gelagerte Zahnstange realisiert. Die Länge dieser Zahnstange ist vor allem für die Hubhöhe entscheidend. Sie ist direkt mit dem Schneckengetriebe (Abbildung 3) verbunden, das durch einen kleinen Motor angetrieben wird. Durch das verwendete Schneckengetriebe wird sichergestellt, dass ein Durchrutschen der Zahnstange bei nicht aufgeschalteter Leistung am Motor verhindert wird und somit auch die Gewichtslast nicht direkt auf den Motor einwirkt.

Da bei jedem Programmschritt mindestens die Vorder- oder Hinterachse Kontakt zum Untergrund hat, war eine bewegende Anregung in Vorwärtsrichtung nicht nötig. Insgesamt sind am unteren Ende der Zahnstange vier Räder verbaut, die durch ihre wagenartige Anordnung dafür sorgen, dass die gesamte Konstruktion auf der mittleren Achse stehen kann. Auch an dieser Achse befindet sich ein entsprechender Ultraschallsensor, der den Abstand zur Stufe bestimmt. Die Hinterachse gleicht im Aufbau der mittleren Achse. Die Unterschiede sind lediglich die Anordnung der Räder und dass diese für die Bewegung des Roboters durch einen großen Motor angetrieben werden. Die in einer Linie und die Breite des Treppensteigers überragend angeordneten Räder sorgen für die nötige Stabilität, damit der Roboter nicht schwankt. Die Notwendigkeit der breiten Hinterachse ergab sich aus den ersten Versuchen der Bewegung, wobei die Achse aus nah aneinander angeordneten zwei Rädern bestand und die Stabilität nicht gewährleistet werden konnte. An der Hinterachse kommt ebenfalls ein Ultraschallsensor zur Abstandsbestimmung zum Einsatz.

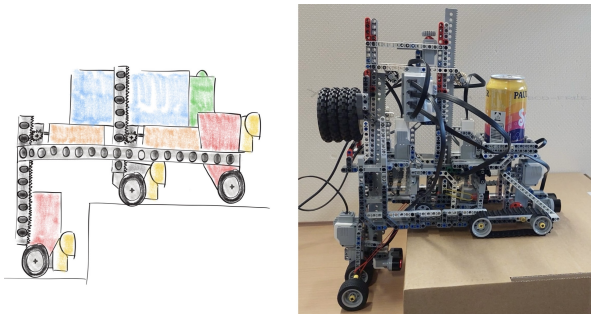
Aufgrund der schon bestehenden Konstruktion wurde die Ablage für die $0{,}33\,\mathrm{L}$-Dose zwischen der vorderen und mittleren Achse platziert. Um das Gewicht gleichmäßig zu verteilen und aus Gründen der beschränkten Möglichkeiten der Platzierung wurde der Programmierstein zwischen hinterer und mittlerer Achse installiert. Da auch dadurch noch kein Optimum der Gewichtsverteilung erreicht wurde, mussten an der hinteren Gleitschiene zusätzliche Gewichte in Form von großen LEGO-Rädern angebracht werden. Zum Starten des Programmablaufs wird ein einfacher Taster verwendet, der sich im oberen Bereich der mittleren Gleitschiene befindet.



(a) Ausgangslage



(b) Anheben des Rahmens



(c) Stufe zu 2/3 erstiegen

Abbildung 2. Schritte des Steigens

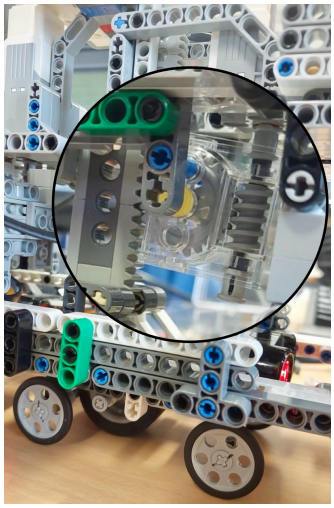


Abbildung 3. Schneckengetriebe

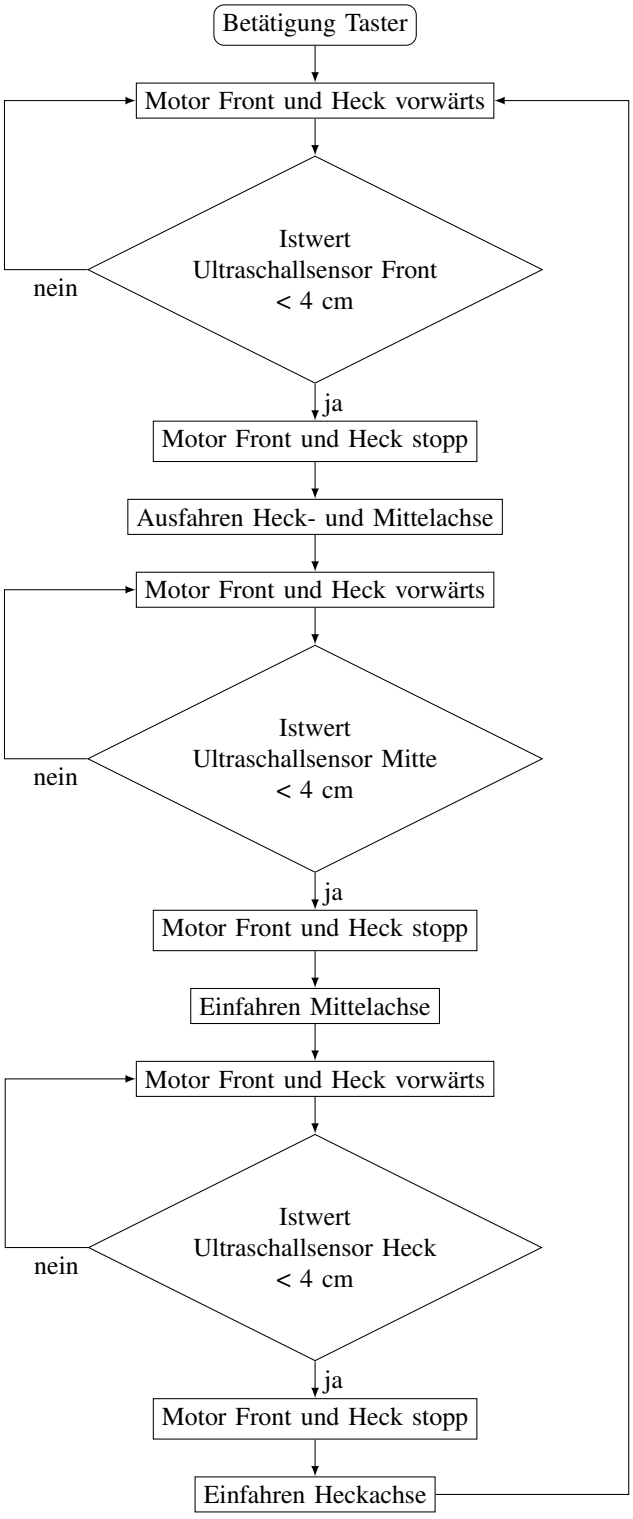


Abbildung 4. Programmablaufplan

B. Programmablauf

Zum Start des Programms muss zuvor eine Verbindung zwischen dem EV3-Stein und MATLAB via USB hergestellt werden. Nach Ausführung des Programms werden alle Voreinstellungen des Programmiersteins gelöscht und er wird neu initialisiert, um fehlerhafte Programmabläufe zu vermeiden. Die Initialisierung wird durch einen Piepton signalisiert. Um den eigentlichen Ablauf (Abbildung 4) zu starten, wird der Taster des Treppensteigers betätigt. Mit dieser Betätigung werden die Motoren an der Front und am Heck mit einer vorher definierten Leistung angeregt und der Roboter führt eine Vorwärtsbewegung aus. Je nach Leistungseinstellung bewegt er sich langsamer oder schneller. Im Zuge dieser Bewegung wird der vordere Ultraschallsensor ausgewertet.

Sobald der Abstand zur Stufe geringer als $4\,\mathrm{cm}$ wird, stoppen die Heck- und Frontmotoren. Anschließend fahren Mittel- und Heckachse aus, wodurch der Rahmen des Treppensteigers samt transportierter Ladung gleichmäßig und waagerecht angehoben wird. Für die Gleichmäßigkeit und den waagerechten Hub muss die Leistung der jeweiligen kleinen Motoren dem Gewicht angepasst werden, das auf dem angeschlossenen Schneckengetriebe lastet. Nachdem die entsprechende Hubhöhe (vorgegeben durch die maximale Zahl der Umdrehungen der Motoren in Grad) erreicht wird, startet wie zuvor die Vorwärtsbewegung. Stellt der Programmierstein mit Hilfe des Ultraschallsensors an der mittleren Achse einen geringeren Abstand als $4\,\mathrm{cm}$ zur Stufe fest, wird die Bewegung beendet.

Im Folgenden fährt die Mittelachse wieder ein. Dafür wird lediglich die Drehrichtung des Hubmotors geändert, indem der Leistungswert negiert wird. Die Zahl der Umdrehungen bleibt identisch, damit die Achse in Ihre Ausgangslage fährt. Ist dieser Abschnitt beendet, fährt der Roboter wieder nach vorn.

Im letzten Schritt wird der Abstand zur Stufe über den Sensor an der Hinterachse bestimmt. Werden die $4\,\mathrm{cm}$ wieder unterschritten, stoppen die Bewegungsmotoren und die Hinterachse wird in ihre Ausgangsposition eingefahren. Abschließend wird der Vorgang ab dem Moment des Auslösens des Tasters wiederholt, was beinhaltet, dass der Treppensteiger die Stufe vollständig befährt und auf die nächste Treppenstufe steigen kann.

IV. Ergebnisdiskussion

Grundlegend ist die anfängliche Idee in Form und Funktion bis auf ein Detail umgesetzt worden. Ursprünglich sollten die Ultraschallsensoren auch dafür genutzt werden, das obere Ende der Stufe zu detektieren und damit den Hub der Konstruktion zu begrenzen. Der Vorteil wäre gewesen, dass sich der Treppensteiger variabel an die Höhe von unterschiedlichen Stufen anpassen kann, ohne dass im Programmablauf etwas geändert werden muss. Aufgrund der trichterförmigen Ausbreitung der Schallwellen ist ein verlässliches Auswerten der Abstandsvergrößerung bei Erreichen des oberen Endes der Stufe nicht garantiert.

Ein großes Problem bei der Konstruktion des Treppensteigers war die Gewichtsverteilung. Das aktuelle System ist nur auf die gefüllte $0{,}33\,\mathrm{L}$-Dose angepasst. Ein Ändern oder sogar Weglassen der zu transportierenden Ladung würde dazu führen, dass

der Treppensteiger nach vorn oder hinten kippen würde. Gerade in den kritischen Abschnitten des Hebens des Rahmens und des Einfahrens der Hinterachse besteht Kippgefahr. Bezüglich des Gewichts war auch festzustellen, dass die Zahnstangen, die auch ein tragendes Element darstellen, in ihrer Konstruktion an ihre Grenzen geraten sind. Diese müssten verstärkt werden, was auch dazu führen würde, dass die Gleitschienen entsprechend angepasst werden müssten. Auch der Motor in Verbindung mit dem Schneckengetriebe an der Mittelachse ist ausgelastet. Der kleine Motor müsste durch einen großen Motor ersetzt werden und das Schneckengetriebe müsste stärker verankert werden, was jedoch wieder Auswirkungen auf die Konstruktion und auf die Gewichtsverteilung hätte.

V. Zusammenfassung und Fazit

Das Projektseminar Elektrotechnik/Informationstechnik 2026 ist erfolgreich abgeschlossen worden. Die zuvor erarbeitete Idee konnte so umgesetzt werden, dass der Treppensteiger in der Lage ist, autonom Treppenstufen zu erkennen und diese dann zu erklimmen.

Bei der Realisierung des Projekts konnten Kenntnisse in der Programmierung mit MATLAB erlernt und vertieft werden, was nicht nur dem Projekt zugute kam, sondern auch im weiteren Verlauf des Studiums von Nutzen ist. Bei einer eventuellen Weiterentwicklung des Treppensteigers würde der Fokus zum einen auf der Unabhängigkeit des Gewichts der Ladung und der Höhe der Stufe liegen, als auch auf der Möglichkeit, die Treppenstufe auch wieder hinabzusteigen.

Literaturverzeichnis

[1] Destatis: *Immer mehr Menschen kaufen online*. https://www.destatis.de/Europa/DE/Thema/Wissenschaft-Technologie-digitaleGesellschaft/Online_Shopping.html. Version: Januar 2025

[2] Destatis: *Einkäufe von Personen über das Internet nach Alter*. https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Einkommen-Konsum-Lebensbedingungen/IT-Nutzung/Tabellen/onlineeinkaeufe-privatezwecke-alter-mz-ikt.html. Version: November 2025

[3] Mindstorms, LEGO: *Verwendung des Ultraschallsensors*. https://ev3-help-online.api.education.lego.com/Retail/de-de/page.html?Path=editor%2FUsingSensors_Ultrasonic.html

Der Treppensteiger

Christian Schmidt, Elektromobilität
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Das folgende Paper beschreibt die Entwicklung eines Treppensteiger-Roboters im Rahmen des LEGO-Projektpraktikums 2026 der Otto-von-Guericke-Universität Magdeburg. Die Grundlage dafür sind LEGO-Mindstorms und MATLAB. Der Roboter ist in der Lage, eine Treppenstufe mittels Ultraschallsensoren autonom zu erkennen und sein Eigengewicht anzuheben, um eine Treppenstufe zu überwinden. Zentrale Schwerpunkte dieses Papers sind die mechanische Konstruktion, die Sensorik sowie die softwareseitige Umsetzung. Zudem werden Herausforderungen und Verbesserungsansätze diskutiert.**

***Schlagwörter*—Autonomes Treppensteigen, LEGO Mindstorms, MATLAB, Robotik, Ultraschallsensorik.**

I. Einleitung

DIE zunehmende Umweltverschmutzung durch Plastikflaschen ist ein zentrales Problem. Alternativen dafür sind beispielsweise Glasflaschen oder Getränkedosen. Allerdings stellt das höhere Gewicht eine starke Anstrengung für die zunehmende alternde Gesellschaft und die damit verbundene Mobilitätseinschränkung dar. Entsprechende Forschungsprojekte, wie beispielsweise ein Exoskelett der Hochschule für Technik, Wirtschaft und Kultur Leipzig, befassen sich mit der Thematik des Treppensteigens, sodass das Treppensteigen bis ins hohe Alter ermöglicht werden soll [1]. Im Rahmen des LEGO-Projektpraktikums 2026 an der Otto-von-Guericke Universität Magdeburg wurde ein autonomer Treppensteiger-Roboter entwickelt, um den Transport von Getränken über Treppenstufen zu simulieren und im Modellmaßstab umzusetzen. Die gewählte Entwicklungsplattform, LEGO Mindstorms EV3, bietet hierfür eine geeignete Kombination aus Mechanik, dem Einsatz von Ultraschallsensorik und einer kompatiblen Programmierschnittstelle in MATLAB.

Zielsetzung des Projektes war die Entwicklung eines Roboters, der mithilfe von Ultraschallsensoren eine Treppenstufe erkennt und diese überwindet. Die zentralen Herausforderungen dabei bestanden in der Gewichtsverteilung sowie das Erzeugen des nötigen Drehmomentes, um das Fahrzeug anzuheben. Das Projekt verbindet die praktischen Erfahrungen aus vorangegangenen Tätigkeiten im Kraftfahrzeugbereich mit den im Studium der Elektromobilität erworbenen Fachkenntnissen. Das vorliegende Paper dokumentiert die technischen Grundlagen, den Konstruktions- und Entwicklungsprozess sowie erarbeitete Lösungsansätze. Anschließend werden die erzielten Ergebnisse kritisch reflektiert und mit der ursprünglichen Zielsetzung verglichen.

II. Vorbetrachtungen

A. Anforderungen an den Treppensteiger

Die grundlegenden Anforderungen an den Treppensteiger-Roboter bestanden in der Fähigkeit, Treppenstufen eigenständig zu erkennen und das eigene Gewicht mit der zusätzlichen Belastung von Getränkedosen anzuheben, ohne dabei nach vorn, zur Seite oder nach hinten zu kippen. Dabei steht die stabile mechanische Konstruktion mit nahezu idealer Gewichtsverteilung und die wiederholbare Funktion im Vordergrund, siehe Abbildung 1.

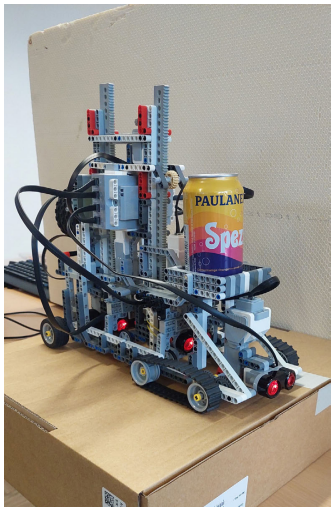


Abbildung 1. Fertiger Treppensteiger

B. Technologische Grundlagen

Die Basis lieferten der EV3-Controller und die LEGO Mindstorms-Baukästen. Der große Vorteil dieser Basis lag darin, dass eine intuitive Programmierung mit MATLAB möglich war. Es wurden insgesamt vier Motoren verbaut: je zwei Motoren für den Antrieb des Fahrzeuges und den Hubmechanismus. Des Weiteren wurden drei Ultraschallsensoren zum Erkennen der Treppenstufe und ein Tastsensor zum Starten des Vorgangs verbaut. Der EV3-Ultraschallsensor hat dabei folgende Funktionsweise: Er strahlt Ultraschallwellen aus und empfängt die vom Hindernis reflektierten Schallwellen. Die Zeit, die die Schallwellen brauchen, um nach dem Aussenden wieder empfangen zu werden, wird vom Sensor gemessen. Bei einer Schallgeschwindigkeit von $344\,\frac{\mathrm{m}}{\mathrm{s}}$ kann dadurch die Distanz zum Hindernis berechnet werden. Die Schallwellen sind nicht zu hören, da ihre Frequenz zu hoch ist und nicht im hörbaren Ultraschallbereich liegt. Die Ausgabe der berechneten Distanz erfolgt dann über den Bildschirm des EV3-Controllers in cm [2].

Das für den Hubmechanismus verbaute Schneckengetriebe besteht aus einer Welle mit mehreren Schraubengängen und ein Zahnrad, welches in die Welle eingreift, siehe Abbildung 2. Der entscheidende Vorteil dieses Getriebes ist die Selbsthemmung.

DOI: 10.24352/UB.OVGU-2026-037

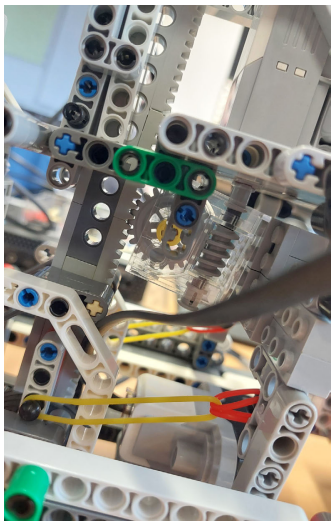


Abbildung 2. Schneckengetriebe

Sie beschreibt einen durch Reibung erzeugten Widerstand, der einem Zurückdrehen des Hubmotors entgegenwirkt, sobald dieser nicht mehr angesteuert wird [3].

III. Umsetzung

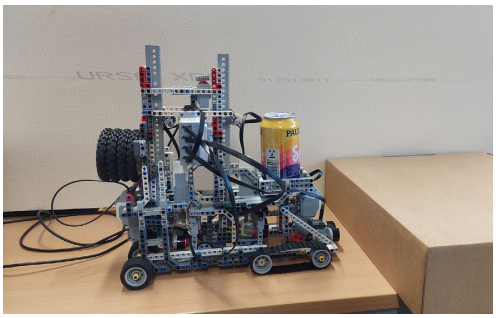
A. Mechanische Konstruktion

Der Roboter basiert auf einem stabilen Gerüst mit angetriebener Vorder- und Hinterachse. Der Antrieb der Vorderachse erfolgt über einen großen EV3-Motor und einen Kettenantrieb. Anfangs wurden Räder getestet, die jedoch nicht genug Traktion generiert haben, um das Fahrzeug fortzubewegen, sobald die Hinterachse in der Luft hängt. Die hintere Achse wird ebenfalls über einen großen EV3-Motor angetrieben. Im Gegensatz zur Vorderachse sind hier vier Räder verbaut, welche über die komplette Fahrzeugbreite verteilt sind, um die Wankstabilität zu gewährleisten.

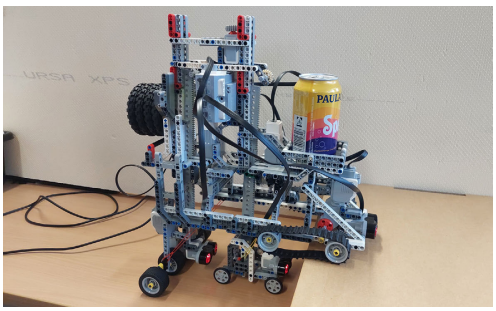
Zudem besitzt der Treppensteiger Gleitschienen an der hinteren und an der mittleren Achse, in denen die Zahnstangen für den Hubmechanismus geführt werden. Der Hubmechanismus wird pro Achse über einen kleinen EV3-Motor und ein daran angeschlossenes Schneckengetriebe realisiert, siehe Abbildung 2. Am oberen Ende des Konstruktes sind kleine Zahnräder integriert, um die Zahnstangenführung zu verbessern. Die mittlere Achse besteht aus einer Zahnstange und vier kleinen Rädern, um das Gewicht zu optimieren.

Außerdem befindet sich am vorderen Ende des Roboters ein Dosenhalter, welcher dem Transport von Getränken dienen soll. Als Gegengewicht sind am Heck des Fahrzeuges vier große Räder angebaut. Um die Gewichtsverteilung weiter zu optimieren, wurde der EV3-Stein in der Mitte des Fahrzeuges integriert.

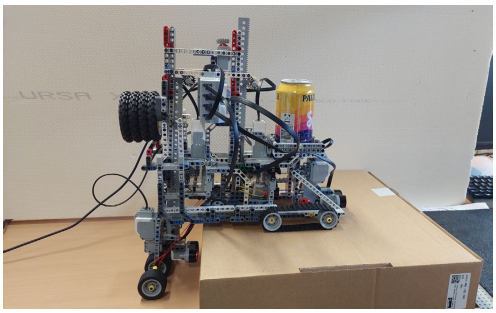
Die drei Ultraschallsensoren, welche an den Achsen sitzen, mussten so tief wie möglich verbaut werden, um den Schwerpunkt des Fahrzeuges nach unten zu verlagern. Zuletzt wurde noch ein Tastsensor am oberen Ende der mittleren Gleitschiene verbaut, um den gesamten Vorgang des Treppensteigens zu starten.



(a) Schritt 1



(b) Schritt 2



(c) Schritt 3

Abbildung 3. Programmschritte

B. Programmablauf

Die Ansteuerung und Programmierung der Sensoren und Motoren erfolgte mit MATLAB. Die Hinweise zur MATLAB-Programmierung des EV3-Controllers wurden aus [4] bezogen. Zunächst erzeugt der EV3-Controller einen Piepton, welcher signalisiert, dass alle Motoren und Sensoren aktiv und angelernt sind. Anschließend wird der Programmablauf mittels Drücken auf den Tastsensor gestartet. Im ersten Schritt fährt der Roboter mit einer festgelegten Motordrehzahl solange bis der vordere Ultraschallsensor ein Hindernis bzw. eine Treppenstufe erkennt, siehe Abbildung 3a. Der Wert, ab dem der Sensor reagiert, liegt bei $4\,\mathrm{cm}$. Im zweiten Schritt werden die beiden kleinen EV3-Motoren angesteuert, welche über die Schneckengetriebe und die Zahnstangen die mittlere und hintere Achse ausfahren und somit das gesamte Fahrzeug anheben, siehe Abbildung 3b. Der Hub ist über eine festgelegte Anzahl an Umdrehungen begrenzt.

Sobald die beiden Achsen auf die vorgegebene Höhe ausgefahren sind, werden die Antriebsmotoren wieder angesteuert und das Fahrzeug fährt nach vorn. Wenn anschließend der

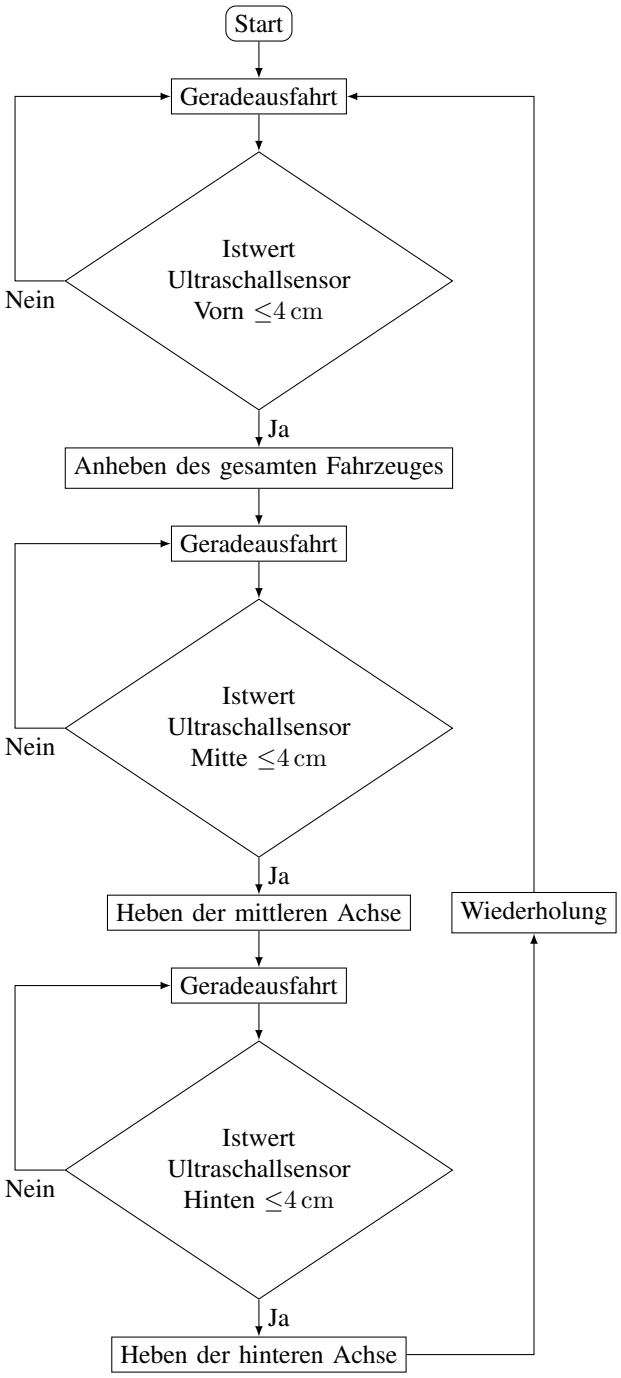


Abbildung 4. Programmablaufplan

mittlere Ultraschallsensor ein Hindernis erkennt, auch hier gilt der Abstand von $4\,\mathrm{cm}$, beginnt der dritte Schritt. Hier wird die mittlere Achse um den gleichen Hub wieder eingefahren. Das Fahrzeug fährt weiter nach vorn, bis der Ultraschallsensor an der Hinterachse ebenfalls ein Hindernis erkennt, siehe Abbildung 3c. Der Abstand wurde genauso wie bei den anderen beiden Sensoren programmiert. Die Hinterachse wird angehoben und der Roboter fährt wieder nach vorn, bis er ein weiteres Hindernis erkennen würde, siehe Abbildung 4.

IV. Ergebnisdiskussion

Die ursprüngliche Zielsetzung eines autonomen Treppensteiger-Roboters konnte im Rahmen des Projektpraktikums umgesetzt werden. Während des Entwicklungs- und Konstruktionsprozesses traten verschiedene Herausforderungen auf. Aufgrund des trichterförmigen Aussendekegels der Schallwellen der Ultraschallsensoren mussten die Hübe der mittleren und hinteren Achse über die Anzahl der Umdrehungen der Hubmotoren geregelt werden, weshalb der Hub für den Karton, der die Treppenstufe simulieren soll, festgesetzt und nicht variabel ist. Dementsprechend musste das Programm modifiziert werden.

Trotzdem ist der Roboter in der Lage eine Treppenstufe bzw. den Karton autonom zu überwinden. Des Weiteren stellte die Gewichtsverteilung eine zentrale Herausforderung dar. Durch Kippen nach vorn und hinten, musste die Konstruktion mehrfach angepasst werden, um die Gewichtsverteilung zu optimieren. Zudem mussten an der Hinterachse weitere Versteifungen vorgenommen werden, da diese Achse besonders viel Gewicht getragen hat. Trotzdem durfte das hintere Gewicht nicht weiter zunehmen, da der Roboter sonst nach hinten kippen würde, sobald die hintere Achse angehoben wird. Außerdem wurden die vorderen Räder durch einen Kettenantrieb ersetzt, da diese nicht genug Traktion generiert haben, um das Fahrzeug im dritten Programmschritt fortzubewegen. Ein zusätzliches Problem stellte die Stabilität der Zahnstangen dar, weil sie den Großteil des Fahrzeuggewichtes gehalten haben. Deshalb mussten auch sie mehrfach repariert und modifiziert werden.

V. Zusammenfassung und Fazit

Zusammenfassend ist zu sagen, dass der Treppensteiger erfolgreich programmiert und sehr stabil konstruiert wurde. Die Kombination aus Mechanik, Sensorik und Programmierung mit MATLAB ermöglichte ein weitgehend autonomes Treppensteigen. Verbesserungspotenzial liegt in der Präzisierung der Ultraschallsensoren und der Stabilisierung der Zahnstangen. Dadurch wäre der Roboter nicht an eine festgelegte Treppenstufenhöhe gebunden, sondern könnte die Höhe der Treppenstufe selbst erkennen und diese überwinden.

Literaturverzeichnis

[1] Schreyer, Anika: *Für mehr Barrierefreiheit | Ein Exoskelett, das Treppensteigen bis ins hohe Alter ermöglichen soll*, März 2022. https://magazin.htwk-leipzig.de/exoskelett-treppensteigen

[2] Team, LEGO E.: *EV3 Ultraschallsensorik*, 2013. https://ev3-help-online.api.education.lego.com/Retail/de-de/page.html?Path=editor%2FUsingSensors_Ultrasonic.html

[3] Wikipedia, the free e.: *Selbsthemmung*, 2025. https://de.wikipedia.org/wiki/Selbsthemmung

[4] RWTH Aachen: *EV3 Toolbox Documentation*, 2020. https://git.rwth-aachen.de/mindstorms/ev3-toolbox-matlab/-/blob/master/MindstormsEV3Toolbox.pdf?ref_type=heads

Automatischer Flaschenöffner

Die Zukunft der Küchenautomatisierung!

Yaroslav Ivanov, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des Projektseminars an der Otto-von-Guericke-Universität Magdeburg wird die Aufgabe gestellt, mithilfe von Standard-LEGO-Bauteilen, dem LEGO Mindstorms EV3-System und der Programmierumgebung MATLAB einen funktionsfähigen Roboter-Prototypen zu entwickeln. Ziel des Seminars ist es, grundlegende Kenntnisse der Konstruktionstechnik und der algorithmischen Programmierung anzuwenden, damit das entwickelte System eigenständig in der Lage ist, nicht-standardisierte Aufgaben oder alltägliche Barrieren zu bewältigen.**

Gegenstand dieser Arbeit ist ein automatisierter Flaschenöffner. In diesem Bericht werden die Funktion des Roboters sowie sein mechanischer Aufbau beschrieben. Zudem werden die technischen Probleme beim Bau und die dafür gefundenen Lösungen in der Konstruktion aufgezeigt.

***Schlagwörter*—3D-Druck, Alltagsassistenz, Drehmomentverstärkung, Getriebe, LEGO Mindstorms, MATLAB**

I. Einleitung

DAS tägliche Öffnen von fest verschlossenen Getränkeflaschen stellt für viele Menschen, insbesondere für Senioren oder Personen mit körperlichen Einschränkungen, eine erhebliche physische Herausforderung dar. Um dieses Problem zu lösen, wurde im Rahmen des Projektseminars ein automatischer Flaschenöffner entwickelt. Die Lösung basiert auf einer stabilen Konstruktion, die die Flasche zunächst im Gehäuse fixiert, um ein Mitdrehen während des Vorgangs zu verhindern. Der eigentliche Öffnungsprozess wird durch einen Drehmechanismus realisiert, der aufgrund einer Getriebeuntersetzung über ein ausreichend hohes Drehmoment verfügt, um den Deckel sicher zu lösen. Vom Nutzer wird lediglich verlangt, die Flasche auf die Plattform zu stellen und den Startknopf zu betätigen.

Das Ziel des Projekts ist die Schaffung einer zuverlässigen Schnittstelle zwischen einfacher Sensorik und effektiver mechanischer Kraftübertragung. Dieses System ersetzt den notwendigen Kraftaufwand beider Hände vollständig und wandelt eine mühsame Alltagsaufgabe in einen einfachen, automatisierten Prozess um, was Zeit und Anstrengung im Haushalt einspart. Im folgenden Bericht werden die konstruktiven Ansätze zur Realisierung dieses Vorhabens sowie die funktionale Umsetzung im Detail dokumentiert.

II. Vorbetrachtungen

A. Ursprüngliches Konzept

In der ersten Planungsphase war vorgesehen, den Roboter für 1-Liter-Flaschen auszulegen. Da der mechanische Aufbau einer Kranmast-Konstruktion ähnelt, hätte dies einen sehr hohen Turm erfordert. Ein solch hoher Aufbau wäre durch das Gewicht des schweren Hebemechanismus instabil geworden und hätte ein massives Übergewicht nach vorne erzeugt. Um dies auszugleichen, wäre eine extrem hohe Anzahl an zusätzlichen Verstrebungen und Bauteilen notwendig gewesen, die jedoch nicht in ausreichender Menge zur Verfügung standen. Daher wurde entschieden, den Fokus auf 0,5-Liter-PET-Flaschen zu legen, um eine kompaktere und stabilere Bauweise zu ermöglichen.

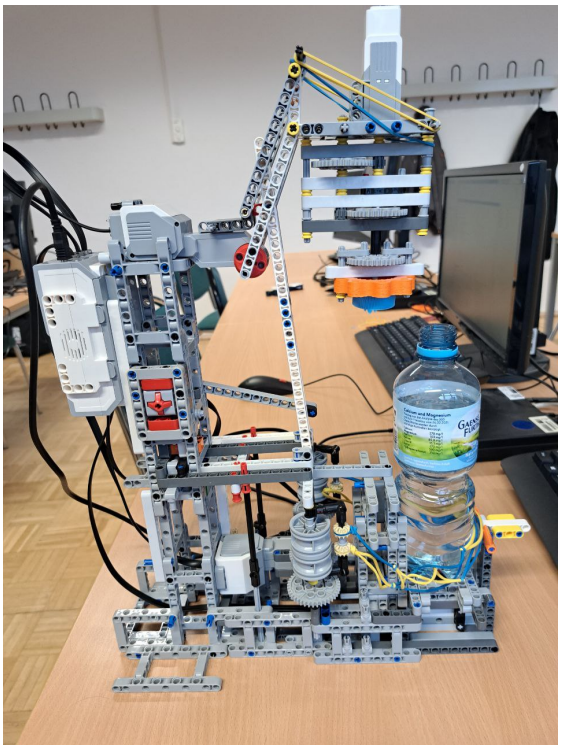


Abbildung 1. Flaschenöffner-Roboter nach der Deckelentfernung

Ein weiterer technischer Aspekt war die Wahl der Sensorik. Ursprünglich sollten Ultraschallsensoren zur Objekterkennung eingesetzt werden. Es wurde jedoch festgestellt, dass diese Sensoren keinen präzisen Punkt, sondern einen zu großen Bereich erfassen. Die Ultraschallwellen würden an der Flasche und an Teilen des eigenen Gehäuses gestreut werden, was zu Fehlmessungen führt. Dies hätte dazu führen können, dass das System den Mechanismus vorzeitig startet, noch bevor die Flasche korrekt positioniert ist. Um einen sicheren Prozessstart zu gewährleisten, wurde dieses Konzept verworfen.

DOI: 10.24352/UB.OVGU-2026-038

B. Methodik und finale Realisierung

Aufgrund der genannten mechanischen und sensorischen Herausforderungen wurde das Konzept angepasst. Anstelle der Ultraschallsensoren wurden Tastsensoren (Buttons) gewählt. Diese liefern eine präzise physische Rückmeldung, sobald die Flasche korrekt auf der Plattform steht, wodurch Fehlstarts ausgeschlossen werden. Für die Umsetzung fiel die Wahl auf das LEGO Mindstorms EV3-System, da es gegenüber dem älteren NXT-System eine stabilere Kommunikation mit MATLAB und eine höhere Rechenleistung bietet.

Ein entscheidender technischer Aspekt ist die Drehmomentverstärkung. Da die direkte Kraft der Servomotoren allein nicht ausreicht, wurde eine Getriebeuntersetzung integriert, um die Motoren zu entlasten. Zudem wurde ein beweglicher Heber entwickelt. Dieser dient primär dazu, den gesamten Drehmechanismus nach oben zu schwenken, um dem Nutzer ein bequemes Einsetzen der Flasche zu ermöglichen. Erst nach der Positionierung führt der Heber den Mechanismus sicher auf den Deckel. Die Steuerung über MATLAB stellt sicher, dass alle Schritte – vom Absenken bis zum Drehvorgang – präzise koordiniert ablaufen.

III. Mechanischer Aufbau und Funktionalität

Der mechanische Aufbau des Roboters ist als eine massive Turmkonstruktion realisiert, die optisch einem Kranmast ähnelt. Um die notwendige Stabilität für den Öffnungsvorgang zu gewährleisten, wurde die Struktur durch eine hohe Anzahl an Verstrebungen verstärkt, die jede Sektion der Konstruktion fest miteinander verbinden. Dies verhindert mechanische Verformungen unter der Last des ausgeübten Drehmoments.

Das Herzstück der Steuerung ist der EV3-Stein. Die Hardware umfasst insgesamt drei Motoren und zwei Sensoren. Zwei große LEGO-Servomotoren steuern den Heber sowie die Fixierung der Flasche, um einen hohen Anpressdruck zu erzielen. Ein kleiner LEGO-Servomotor treibt den Drehmechanismus an. Zur Objekterkennung und Prozessstart sind zwei Tastsensoren in die Basis integriert.

Die Konstruktion besteht aus diesen Hauptmechanismen: dem Heber, dem Greifmechanismus und dem Drehmechanismus. Im Folgenden wird das Funktionsprinzip des Roboters detailliert beschrieben, wobei die einzelnen Phasen des Arbeitsablaufs und das Zusammenwirken der mechanischen Komponenten erläutert werden.

A. Erkennung und Systemstart

Die erste Phase des Algorithmus befasst sich mit der Objekterkennung und der Sicherheitsfreigabe für den Start. An der Basis des Roboters befindet sich eine speziell konstruierte Plattform, auf die der Nutzer die Flasche stellt. Um einen fehlerfreien Systemstart zu gewährleisten, werden zwei Tastsensoren in einer logischen Schaltung verwendet: Der erste Tastsensor ist direkt unter der Plattform positioniert. Er wird automatisch durch das Gewicht und den physischen Kontakt der Flasche aktiviert, sobald diese auf die Basis gestellt wird. Dies dient der Bestätigung, dass das Objekt korrekt in der Halterung platziert wurde.

Der zweite Tastsensor befindet sich an einer gut erreichbaren Stelle am Gehäuse und muss vom Nutzer manuell betätigt werden. Diese doppelte Abfrage ist für den Bedienkomfort und die Betriebssicherheit entscheidend: Sie verhindert, dass der Algorithmus und die schweren mechanischen Komponenten (wie der Heber) starten, bevor der Nutzer die Flasche stabil losgelassen hat. Erst wenn beide Signale gleichzeitig am EV3-Stein anliegen, wird der Befehl zur Aktivierung der Motoren gegeben. Dies schließt Fehlstarts aus und gibt dem Anwender die volle Kontrolle über den Zeitpunkt des Prozessbeginns.

B. Absenken des Moduls (Zustellung durch den Heber)

Nach der erfolgreichen Detektion wird der Prozess durch den ersten großen Servomotor fortgesetzt, der stabil an der Oberseite der Turmkonstruktion montiert ist. Die Basis des Hebemechanismus bilden parallele Balken, die das gesamte Drehmodul tragen. Um das Modul präzise auf den Flaschendeckel zu führen, führt der Motor eine programmierte Rotation von exakt 50° aus.

Eine besondere technische Herausforderung stellte das hohe Eigengewicht des Drehmoduls dar, welches den Heber im Ruhezustand nach unten zog. Zur Lösung dieses Problems wurde die Konstruktion mechanisch mit Gummibändern verstärkt, um den Hubarm zu stabilisieren und die Last auf den Motor zu reduzieren.

Zusätzlich wurde im MATLAB-Code der sogenannte `BrakeMode` implementiert. Dieser sorgt dafür, dass der Motor den Heber nach dem Hochfahren aktiv in der oberen Position arretiert und ein ungewolltes Absacken durch die Schwerkraft verhindert wird. Durch dieses Zusammenspiel aus mechanischer Unterstützung und Software-Steuerung konnte eine präzise und sichere Positionierung auf dem Deckel erreicht werden.

C. Fixierung der Flasche

Nachdem der Heber das Drehmodul auf dem Deckel positioniert hat, aktiviert der zweite große Servomotor an der Basis der Konstruktion den Fixierungsvorgang. Ursprünglich war geplant, die Flasche lediglich durch gummierte LEGO-Räder direkt anzupressen. Erste Tests zeigten jedoch, dass diese Methode keine ausreichende Haftung bot, wodurch sich die Flasche beim Öffnungsversuch zusammen mit dem Deckel drehte.

Um dieses Problem zu lösen, wurde ein eigener Fixierungsmechanismus entwickelt, der auf einer speziellen Anordnung von vier Zahnrädern basiert, die eine symmetrische Spannung der Gummibänder erzeugen. Das erste Zahnrad ist direkt mit der Motorachse verbunden. Um auf der gegenüberliegenden Seite eine gegensinnige Rotation zu erzielen, wurde dort eine zusätzliche Zwischenzahnrad-Stufe eingebaut. Dies bewirkt, dass sich beide Seiten nicht in dieselbe Richtung, sondern nach innen zueinander drehen.

Da Standard-Gummibänder zu kurz waren, wurden mehrere Gummis miteinander verknotet, um die nötige Länge zu erreichen. Bei einer Motorrotation von 600° wickeln beide Seiten die verknotete Gummischlaufe gleichzeitig nach innen auf. Dadurch wird die Flasche gleichmäßig umschlungen und mit hohem Anpressdruck fest gegen die Rückwand

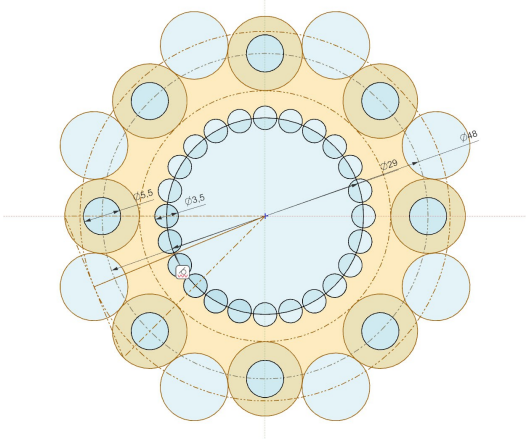


Abbildung 2. 3D-Modell des Greifaufsatzes für den Flaschendeckel

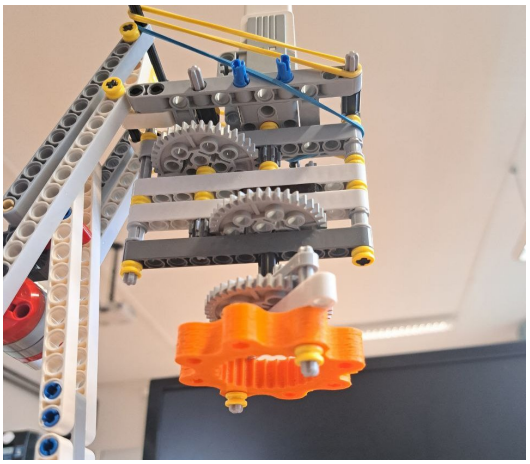


Abbildung 3. Drehmechanismus mit Getriebe und 3D-Teil

der Konstruktion gezogen. Dieser Aufbau verhindert effektiv, dass sich der Flaschenkörper während des Öffnungsvorgangs mitdreht.

D. Öffnungsvorgang mit Drehmechanismus

Nachdem die Flasche durch den Greifmechanismus fixiert wurde, startet der eigentliche Öffnungsvorgang. Eine der größten ingenieurtechnischen Herausforderungen war dabei das mangelnde Drehmoment des LEGO-Motors sowie die fehlende Griffigkeit der Standardbauteile.

Ursprünglich wurde versucht, den Deckel mit herkömmlichen LEGO-Rädern oder Greifarmen zu fassen. Es stellte sich jedoch heraus, dass diese Teile auf der glatten Kunststoffoberfläche des Verschlusses keinen ausreichenden Halt fanden und ständig durchrutschten. Um dieses Problem zu lösen, wurde ein spezialisierter 3D-gedruckter Greifaufsatz entwickelt. Wie in der Konstruktionszeichnung zu sehen ist (vgl. Abbildung 2), weist dieser Aufsatz eine präzise Innengeometrie mit einem Durchmesser von 29 mm und kleinen inneren Zähnen auf. Diese Zähne greifen direkt in die Rillen des Flaschendeckels und erzeugen einen perfekten Formschluss, der ein Durchrutschen unmöglich macht.

Zusätzlich wurde die Kraft über ein Getriebe (vgl. Abbildung 3) mit einer Untersetzung von 1:25 übertragen, um das zu geringe Motordrehmoment auszugleichen. Erst durch die Kombination aus dem passgenauen 3D-Druck-Aufsatz und der massiven mechanischen Verstärkung kann der Motor den Widerstand des Deckels überwinden und diesen zuverlässig aufdrehen.

E. Prozessabschluss

Zum Abschluss des Vorgangs werden alle Motoren in ihre Ausgangsposition zurückgeführt. Der Fixierungsmotor führt eine Rückwärtsdrehung von 600° aus, um die Gummibänder zu lösen und die Flasche freizugeben. Zeitgleich fährt der Heber durch eine entgegengesetzte Bewegung von 50° nach oben. Um ein Absinken des schweren Drehmoduls durch die Schwerkraft zu verhindern, wird im MATLAB-Code der `BrakeMode` aktiviert, welcher den Hebelarm aktiv in der oberen Position arretiert. Damit ist das System für den nächsten Einsatz bereit.

IV. Ergebnisdiskussion

Das Endergebnis zeigt, dass der Prototyp die gestellte Aufgabe erfolgreich erfüllt. Dennoch traten im Verlauf der Entwicklung technische Herausforderungen und spezifische Nachteile der aktuellen Konstruktion auf, die kritisch bewertet werden müssen. Ein Hauptproblem der aktuellen Umsetzung ist die Kinematik des Hebers. Da sich der Heber auf einer Kreisbahn bewegt, trifft der Drehmechanismus den Deckel nicht linear von oben, sondern in einem Bogen. Dies führte bei den Tests zu Schwierigkeiten, da die Flasche extrem präzise positioniert werden musste. Jede kleine Abweichung veränderte den Auftreffwinkel, was zeitintensive Berechnungen für den Senkwinkel und die Geschwindigkeit im MATLAB-Code erforderlich machte.

Für zukünftige Optimierungen sollte die aktuelle Heber-Konstruktion durch eine vertikale Linearführung ersetzt werden. Ein Mechanismus, der den Drehmechanismus exakt senkrecht von oben auf die Plattform absenkt, würde die Positionierung der Flasche erheblich vereinfachen und die Fehleranfälligkeit des Systems minimieren.

Ein weiterer limitierender Faktor ist die aktuelle 3D-gedruckte Greifkomponente. Da ihre Parameter fest definiert sind, ist die Funktionalität des Roboters stark eingeschränkt und auf nur eine Deckelgröße festgelegt. Für zukünftige Optimierungen sollte daher nicht nur der Heber durch eine vertikale Linearführung ersetzt, sondern auch das Greifmodul überarbeitet werden. Eine konusförmige Geometrie, die sich nach unten hin erweitert, würde es ermöglichen, Deckel mit unterschiedlichen Durchmessern sicher zu fassen und zu zentrieren. Dies würde die Flexibilität des Systems erheblich steigern und die Fehleranfälligkeit bei der Positionierung minimieren.

Insgesamt wurde jedoch ein funktionsfähiger Prozess realisiert, der die synergetische Nutzung von Mechanik und MATLAB-Programmierung erfolgreich demonstriert.

V. Zusammenfassung und Fazit

Im Rahmen dieses Projektseminars wurde erfolgreich ein funktionsfähiger Prototyp eines automatisierten Flaschenöffners

auf Basis des LEGO Mindstorms EV3-Systems und MATLAB entwickelt. Das Ziel, eine mechanische Unterstützung für den Alltag zu schaffen, wurde durch das Zusammenwirken von präziser Sensorik und einer kraftvollen Getriebestruktur erreicht.

Die Arbeit hat gezeigt, dass die Herausforderungen der mechanischen Instabilität und des geringen Motordrehmoments durch kreative Lösungen, wie den Einsatz einer 1:25-Untersetzung und eines spezialisierten 3D-Druck-Greifers, effektiv überwunden werden konnten. Trotz der identifizierten Optimierungspotenziale bei der Kinematik des Hebers und der Flexibilität der Greifkomponente bietet das System eine zuverlässige Hilfe beim Öffnungsvorgang.

Insgesamt demonstriert das Projekt die erfolgreiche Anwendung von Konstruktionsprinzipien und die effiziente Steuerung komplexer Bewegungsabläufe mittels algorithmischer Programmierung in MATLAB.

LITERATURVERZEICHNIS

[1] Texas Tech University STEM Center, *LEGO MINDSTORMS EV3: Motors and Sensors Technical Specifications*, [Online]. Verfügbar unter: https://www.depts.ttu.edu/coe/stem/gear/ev3/documents/EV3-Motors-Sensors.pdf (Belegt das Drehmoment von 20 Ncm für den Large Motor und 8-12 Ncm für den Medium Motor).

[2] J. Kammet, *Supplemental notes on gear ratios and torque transmission*, Brooklyn College, City University of New York, 2015. [Online]. Verfügbar unter: https://www.sci.brooklyn.cuny.edu/~kammet/gear_notes.pdf (Grundlagen der Drehmomentberechnung bei Getriebeuntersetzungen).

Roboter zum Flaschenöffnen

Oleksandr Vorontsov, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des Projektseminars 2026 an der Otto-von-Guericke-Universität wurde ein Roboter-Flaschenöffner entwickelt. Dieser Roboter erkennt, wenn eine Flasche auf ihm steht, und schraubt im halbautomatischen Modus den Deckel ab. Der Fokus der vorliegenden Arbeit liegt auf der Konstruktion des Roboters, dem Programmablauf, den während der Entwicklung aufgetretenen Problemen sowie den umgesetzten oder möglichen Lösungen.**

***Schlagwörter*—Automatisierung, Flasche, LEGO Mindstorms, MATLAB, Roboter**

I. Einleitung

Es gibt immer wieder Flaschen, deren Öffnen viel Kraft oder Zeit erfordert. Im Rahmen des Projektseminars 2026 an der Otto-von-Guericke-Universität wurde ein Roboter entwickelt, der dem Menschen bei der Öffnung von Plastikflaschen assistiert Der hier beschriebene Mechanismus kann für Menschen mit eingeschränkter Mobilität, beispielsweise für Rentnerinnen und Rentner, von Nutzen sein. Auch in Situationen, in denen eine Vielzahl an Flaschen geöffnet werden muss, beispielsweise auf verschiedenen Veranstaltungen, kann er eine nützliche Funktion erfüllen.

II. Vorbetrachtungen

A. Einschränkungen

Die Funktionalität dieses Roboters wurde auf das Öffnen von 0,5-Liter-Plastikflaschen beschränkt, um die Komplexität der Konstruktion und des Codes zu reduzieren. Im Rahmen der Testverfahren wurde eine Flasche mit einem standardmäßigen Deckel verwendet, dessen Höhe ca. $14\,\mathrm{mm}$ und Durchmesser ca. $28\,\mathrm{mm}$beträgt.

B. Technologische Grundlage

Als Basis wurde das LEGO-Mindstorm-Set mit dem EV3-Stein gewählt, das im Vergleich zu seinem Vorgänger, dem NXT-Stein, über eine erweiterte und verbesserte Funktionalität verfügt (vgl. [1]). Als Programmierumgebung wurde MATLAB gewählt, da es über eine einfache Syntax verfügt und eine Bibliothek für den EV3-Block von LEGO Mindstorms enthält (vgl. [2]). Für den Bau des Flaschenöffner-Roboters wurden LEGO-Teile aus dem LEGO-Mindstorms-Set, Tastsensoren, große und kleine Motoren sowie Gummibänder verwendet.

C. Idee und Funktionsweise

Die ursprüngliche Idee bestand darin, einen vollautomatischen Roboter zu konstruieren und die Flaschenerkennung unter Verwendung eines Ultraschallsensors zu implementieren.

DOI: 10.24352/UB.OVGU-2026-039

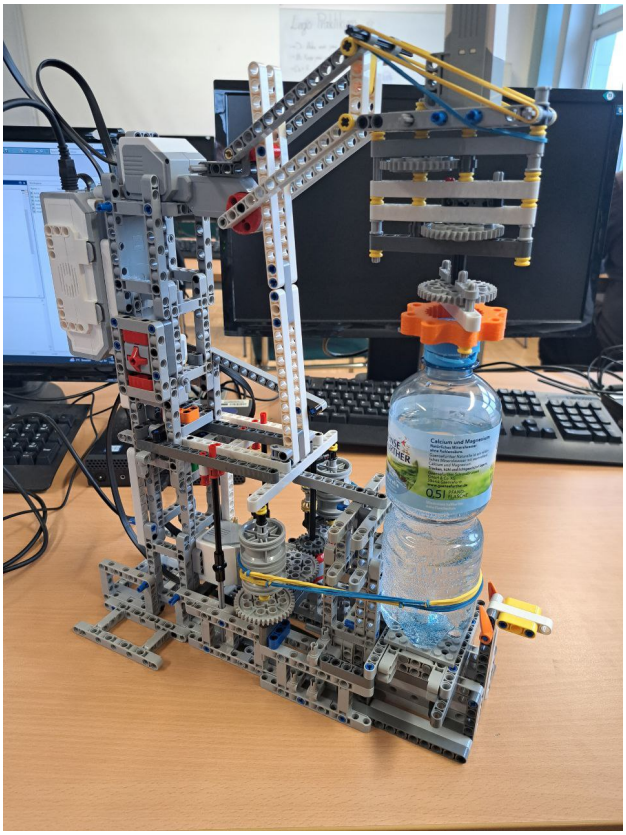


Abbildung 1. Flaschenöffner-Roboter

Aufgrund der Eigenschaften des Sensors, wie einem großen Detektionswinkel, der Unfähigkeit des Sensors, Objekte in einer Entfernung von weniger als $3\,\mathrm{cm}$ bis $5\,\mathrm{cm}$ zu erkennen [3], und dem Platzmangel in der Konstruktion, wo dieser Sensor in größerer Entfernung angebracht werden könnte, um Objekte ungehindert zu erkennen, wurde er jedoch durch eine Plattform ersetzt, die mit einem Tastsensor ausgestattet ist. Um eine versehentliche Aktivierung des Mechanismus und damit eine Verletzung des Benutzers oder eine Beschädigung des Mechanismus zu vermeiden, wurde eine Aktivierungstaste am Korpus angebracht. Das Endergebnis ist in Abbildung 1 dargestellt.

Der zugrundeliegende Mechanismus ist wie folgt: Die Plattform identifiziert das Vorhandensein einer Flasche und fixiert diese, sobald die Aktivierungstaste betätigt wird. In der Folge senkt sich ein Drehmechanismus auf den Deckel und schraubt diesen ab. Nach Abschluss des Vorgangs wird der obere Teil angehoben und die Flasche wird gelöst, sodass eine geöffnete Flasche auf der Plattform verbleibt. Es besteht die Möglichkeit, den Vorgang mehrfach zu wiederholen. Das

vollständige Blockdiagramm ist in Abbildung 2 dargestellt.

III. Konstruktion und Programmverlauf

Die Konstruktion besteht aus vier Teilen: dem Körper, dem Heber, dem Drehmechanismus und dem Greifmechanismus.

A. Körper

Der Körper ist das Hauptelement, an dem alle anderen Komponenten befestigt sind. Ein signifikantes Problem während des Entwicklungsprozesses stellte der freie Lauf der LEGO-Teile in der Konstruktion, was zu großen Ungenauigkeiten in der Arbeitsausführung führte. Zur Lösung des Problems wurden zahlreiche zusätzliche Verstärkungen und Stützen installiert.

Der Körper besteht aus einem Turm und einem Boden. Auf dem Turm sind der EV3-Stein, der Heber und der Aktivierungsknopf befestigt. Die Plattform und der Greifmechanismus sind auf dem Boden installiert.

Die Plattform ist mit einem Tastsensor ausgestattet, der zweimal pro Sekunde den Druckzustand überprüft. Wird seitens des Sensors in der Plattform eine logische 1 registriert, führt das Betätigen der Taste an der Seite des Korpus zur Aktivierung des weiteren Mechanismus. Wird seitens der Plattform kein Objekt erkannt, führt das Drücken der zweiten Taste zum Ausschalten des EV3-Stein und zum Stoppen des Programms.

B. Heber

Dieser Teil der Konstruktion ist dafür zuständig, den Drehmechanismus auf die Flasche abzusenken und nach Beendigung des Programms wieder in die ursprüngliche obere Position anzuheben.

Der Heber besteht aus einem großen Motor, an dem der Drehmechanismus mit einer Parallelogrammverbindung befestigt ist, um die Parallelität über den Boden sicherzustellen. Zur Gewährleistung einer erhöhten Stabilität wurden Gummibänder verwendet. Um unerwünschte seitliche Schwankungen zu reduzieren, wurde entlang der Bewegungsrichtung eine Führung hinzugefügt.

Ein großes Problem während der Entwicklung war, dass dieser Motor das an ihn angehängte Gewicht nicht selbstständig halten kann. Um den gesamten Mechanismus in der oberen Position im ausgeschalteten Zustand zu halten, wurde ein Safety Pin hinzugefügt. Nach dem Einschalten des EV3-Steins wird der Motorparameter `brakeMode` aktiviert und der Safety Pin kann entfernt werden.

C. Drehmechanismus

Dieser Teil des Mechanismus wird in der unteren Position des Hebers aktiviert.

Die Hauptaufgabe bestand darin, die Einschränkungen der LEGO-Teile zu überwinden, nämlich die mangelnde Motorleistung und die geringe Reibung zwischen den LEGO-Teilen und dem Flaschenverschluss.

Zur Steigerung der Ausgangsleistung und des Drehmoments wurde der Motor mit einem Reduktionsgetriebe mit einer Übersetzung von 25:1 gekoppelt.

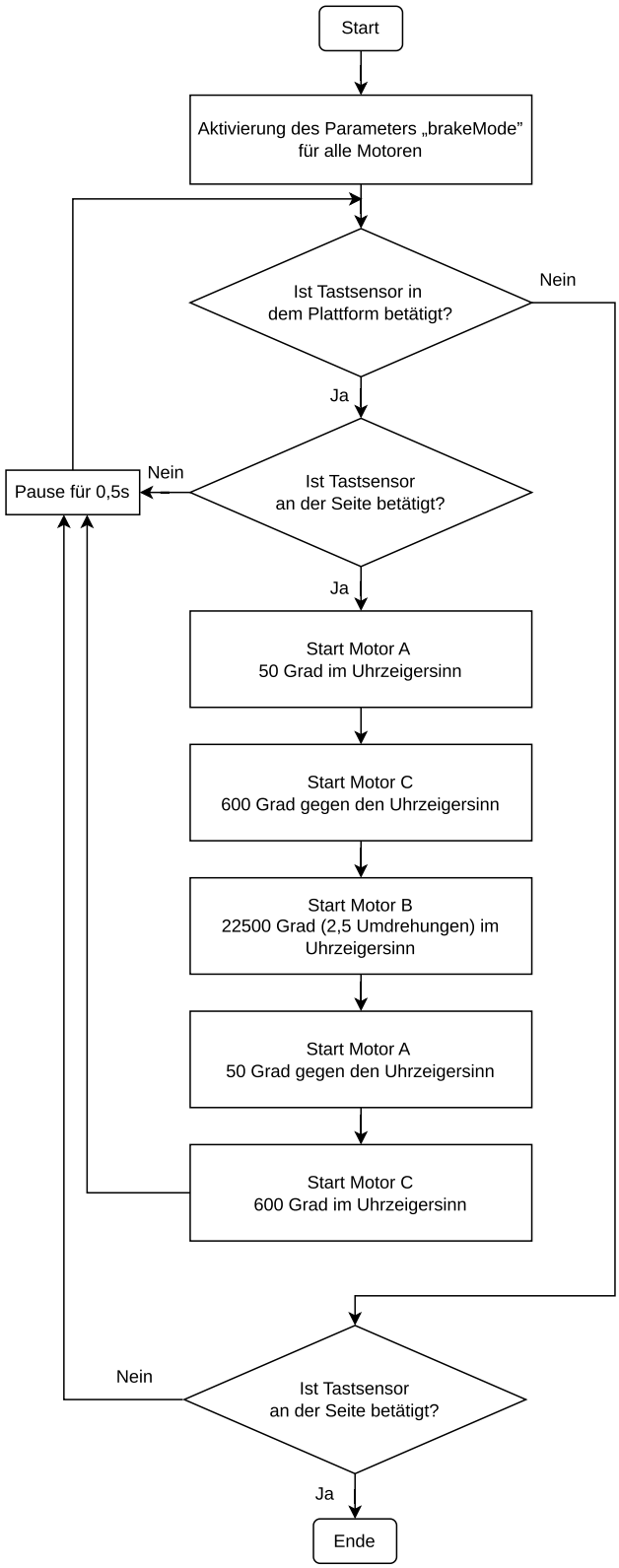


Abbildung 2. Blockdiagramm des Programmverlaufs. Motor A – Hebermotor, Motor B – Drehmechanismus, Motor C – Greifmechanismus

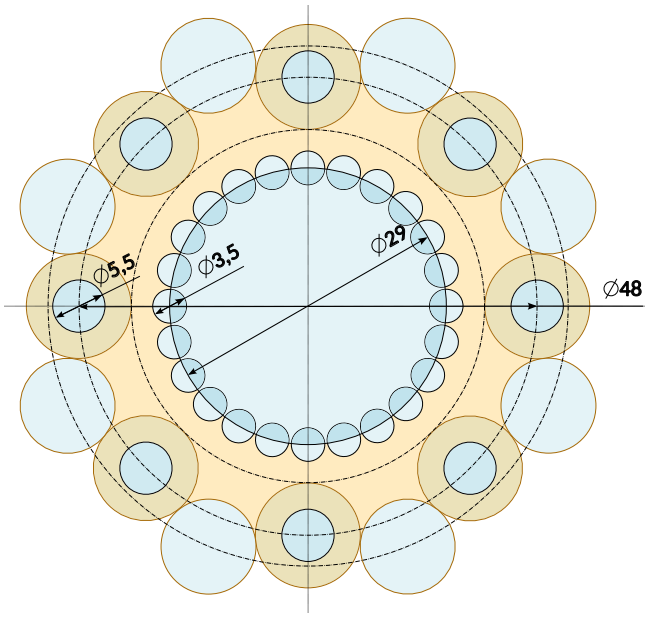


Abbildung 3. Modell des gedruckten Teils mit Abmessungen (in mm)

Zur Herstellung der Verbindung mit dem Deckel und Übertragung des Drehmoments wurde ein Teil mittels eines 3D-Druckers gefertigt (vgl. Abbildung 3). Die Abmessungen und die Form des Teils wurden speziell auf die Kompatibilität mit anderen LEGO-Teilen abgestimmt.

D. Greifmechanismus

Dieses Gerät befindet sich zwischen der Plattform und dem Turm. Die Konstruktion besteht aus einem großen Motor, der zwei Zylinder in entgegengesetzte Richtungen rotiert, sowie einer Wand, an der die Flasche anliegt. An den Zylindern sind zwei Seile aus miteinander verflochtenen Gummibändern befestigt. Im betriebsbereiten Zustand müssen sich die Gummibänder auf einer speziell dafür vorgesehenen Erhebung befinden, die sich auf der gegenüberliegenden Seite der Plattform befindet. Diese Erhebung ermöglicht die Vorbereitung der Gummibänder für einen stabileren Kontakt mit der Flasche.

Der Mechanismus wird nach dem Heber in Betrieb genommen, wobei dieser die Gummibänder um die Flasche zieht und sie gegen die Wand drückt. Nach Abschluss des Programms rotieren die Zylinder in entgegengesetzte Richtungen und befreien die Flasche.

Der Fokus zukünftiger Optimierungen könnte auf der Greifbarkeit der Flasche als initialem Schritt des Programms liegen, mit dem Ziel, die Flasche stets in die gleiche Position zu bringen und die Präzision der übrigen Programmschritte zu erhöhen. Die Lösung dieses Problems bedingt jedoch die Lösung zweier weiterer Probleme, nämlich der Ausrichtung der Flaschenmitte und des Drehmechanismus.

IV. Ergebnisdiskussion und Verbesserungsvorschläge

Die für den Roboter gesetzten Ziele wurden erreicht. Der Roboter ist nun in der Lage, eine Plastikflasche zu identifizieren und zu öffnen. Es besteht jedoch noch Optimierungspotenzial hinsichtlich der Genauigkeit und Qualität der meisten Mechanismen.

Zukünftige Optimierungen sollten sich auf den Hebermechanismus und das Festhalten der Flasche konzentrieren. Für den Heber ist es ratsam, alternative Konfigurationen in Erwägung zu ziehen, die eine präzisere und stärkere Bewegung des Drehmechanismus ermöglichen Die Handhabung der Flasche sollte dahingehend optimiert werden, dass der Vorgang des Greifens autonomer gestaltet wird.

Eine potenzielle Optimierung der Funktionalität des Roboters könnte in der Erkennung der Höhe der Flasche und der Modifikation der Form des mit einem 3D-Drucker gefertigten Teils liegen. Ziel dieser Anpassung ist die Erweiterung des Bereichs geeigneter Durchmesser der Deckel und die Erleichterung des Aufsetzens des Teils auf den Deckel. Darüber hinaus wäre es sinnvoll, während der Programmausführung eine Sensorprüfung in die Plattform zu implementieren, um im Falle der Abwesenheit einer Flasche sämtliche Mechanismen anzuhalten.

V. Zusammenfassung und Fazit

Im Rahmen des Projektseminars 2026 wurde erfolgreich ein Roboter entwickelt, der Menschen bei der Öffnung von Flaschen assistiert. Der Mechanismus wurde unter Verwendung von Standard-LEGO-Teilen, drei Motoren, zwei Drucksensoren und einem EV3-Stein konstruiert. Bei Betätigung des entsprechenden Bedienelements öffnet er eine Plastikflasche mit einem Volumen von 0,5 Litern und erfüllt somit die ihm gestellte Aufgabe.

Die primäre Herausforderung bestand in der Optimierung der Stabilität der Konstruktion und des Programms, um die autonome Ausführung aller Schritte durch den Roboter zu gewährleisten. Zukünftige Verbesserungen sollten sich auf diesen Aspekt sowie auf die Erweiterung der Funktionalität des Roboters konzentrieren.

Literaturverzeichnis

[1] LEGO Mindstorms: *HELP / FAQ*. https://web.archive.org/web/20150224035959/http://service.lego.com/en-us/helptopics/products/themes/mindstorms/mindstorms-ev3/ev3-and-nxt-differences. – Archived February 24, 2015, at the Wayback Machine

[2] Atorf, L. ; Sondermann, B. ; Stadtmann, T. ; Rossmann, J.: *RWTH - Mindstorms EV3 Toolbox*. https://git.rwth-aachen.de/mindstorms/ev3-toolbox-matlab. Version: 2018

[3] LEGO Mindstorms: *Using the Ultrasonic Sensor*. https://ev3-help-online.api.education.lego.com/Education/en-gb/page.html?Path=editor%2FUsingSensors_Ultrasonic.html. Version: February 2026

Automatischer Seifenspender mit Bilderkennung

Hadi Yahya Hadi Al-Hamed, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des Projektseminars Elektrotechnik und Informationstechnik (LEGO Mindstorms) an der Otto-von-Guericke-Universität Magdeburg wurde ein vollautomatischer, berührungsloser Seifenspender entwickelt. Ziel des Projekts war es, durch eine komplett kontaktlose Bedienung die Hygiene zu verbessern und eine smarte, effiziente Seifenausgabe zu gewährleisten. Die mechanische Umsetzung erfolgte mit dem zur Verfügung gestellten LEGO-NXT-Set. Für die Programmierung und Ansteuerung der Hardwarekomponenten wurde MATLAB genutzt. Die berührungslose Steuerung wird über eine Webcam realisiert. Das KI-gestützte Framework MediaPipe erfasst die Hand des Nutzers per Landmark-Erkennung und löst daraufhin automatisch den Motor für den Pumpstoß aus.**

***Schlagwörter*—Bilderkennung, Framework, LEGO-NXT, MATLAB, MediaPipe, Seifenspender**

I. Einleitung

In der heutigen Zeit spielt die Handhygiene im Alltag eine zentrale Rolle, um Infektionen zu vermeiden. Traditionelle Seifenspender erfordern jedoch einen direkten physischen Kontakt, wodurch sie paradoxerweise selbst zu einer potenziellen Quelle für die Übertragung von Krankheitserregern werden können. Um genau dieses Risiko zu eliminieren, wurde im Rahmen des Projektseminars „Elektrotechnik und Informationstechnik (LEGO Mindstorms)" ein automatisierter, berührungsloser Seifenspender entwickelt. Ziel des Projekts ist eine smarte, komplett kontaktlose Bedienung, die den Alltag hygienischer und effizienter macht. Anstatt sich auf übliche Infrarot- oder Ultraschallsensoren zu verlassen, wird der mechanische LEGO-Aufbau über eine handelsübliche Webcam und intelligente Bilderkennung gesteuert. Die folgende Ausarbeitung dokumentiert die Entwicklung bis zum Endresultat: vom mechanischen Aufbau über die Programmierung bis hin zu den praktischen Herausforderungen und deren Lösungen (siehe Abb. 1).

II. Vorbetrachtungen

Bevor die Konstruktion und Programmierung im Detail betrachtet werden, werden zunächst die grundlegenden Komponenten des Systems vorgestellt.

A. Mechanismus eines traditionellen Seifenspenders

Ein traditioneller Seifenspender besteht im Wesentlichen aus einem Behälter für die Seifenflüssigkeit und einem mechanischen Pumpkopf. Durch das Herunterdrücken des Pumpenkopfes wird die Seife durch ein inneres Rohr nach oben transportiert und ausgegeben. Um diesen Fördervorgang zu starten, muss der Pumpenkopf manuell nach unten gedrückt

DOI: 10.24352/UB.OVGU-2026-040

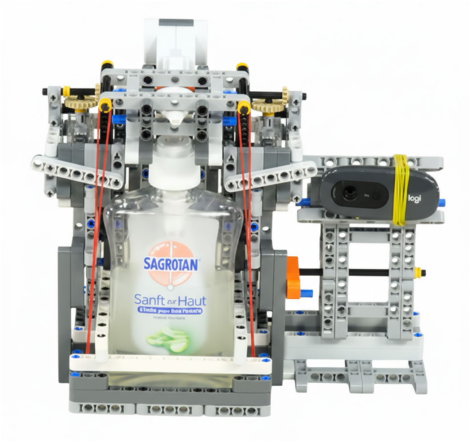


Abbildung 1. Finaler vollautomatischer, berührungsloser Seifenspender

werden, wobei er sich dem Widerstand einer eingebauten Feder entgegenstellen muss.

Um die hierfür erforderliche Betätigungskraft zu ermitteln, wurde der Seifenspender auf einer kalibrierten Küchenwaage allmählich nach unten gedrückt. Das dabei ermittelte Höchstgewicht von $2{,}1\,\mathrm{kg}$ entspricht einer Druckkraft von etwa $20{,}6\,\mathrm{N}$. Dieser Kraftaufwand ist technisch notwendig und üblich, da die eingebaute Feder kräftig genug sein muss, um nach der Betätigung die zähflüssige Seife aus dem Behälter zu saugen und das System in seine Ausgangsposition zurückzuführen. Für die Automatisierung stellt diese mechanische Gegenkraft eine Hürde dar, da der zu wählende elektrische Antrieb ausreichend Kraft aufbringen muss, um die Feder problemlos zu stauchen.

B. Antrieb und Kraftübertragung

Um die zuvor ermittelte Druckkraft von $20{,}6\,\mathrm{N}$ aufzubringen, reicht das Drehmoment eines standardmäßigen LEGO-Elektromotors nicht aus. Dieser erzeugt zunächst nur die primäre Rotationsbewegung. Daher ist ein mehrstufiger Antriebsstrang zur Kraftübersetzung erforderlich.

Im ersten Schritt wird die horizontale Drehung des Motors über zwei Kegelräder um $90°$ auf eine vertikale Achse umgelenkt. Diese Achse treibt anschließend ein in einem transparenten Gehäuse gelagertes Schneckengetriebe an (siehe Abb. 2 aus der Quelle [3]), das als zentrales Element der Kraftübertragung fungiert. Ein solches Getriebe setzt sich grundlegend aus einer schraubenförmigen Welle, der sogenannten Schnecke, und einem zylindrischen Rad, dem Schneckenrad, zusammen [1]. Dabei greift das Gewinde der Schneckenwelle in die Zahnlücken des Schneckenrades ein.

Der entscheidende mechanische Vorteil dieses Getriebes besteht darin, dass sich die Drehgeschwindigkeit stark reduziert,

Abbildung 2. LEGO-Schneckengetriebe [3]

während sich das resultierende Drehmoment enorm erhöht. Dieses Untersetzungsverhältnis beträgt 24:1, da die Schnecke bei einer vollen Umdrehung das Zahnrad mit 24 Zähnen um genau einen Zahn weiterbewegt [3]. Zudem ermöglicht es eine gute Belastbarkeit und eine hohe Kraftübertragung auf kleinstem Raum. Erst durch diese enorme Drehmomentsteigerung wird die nötige Kraft generiert, um den Pumpwiderstand der Flasche zuverlässig zu überwinden. Ein weiterer konstruktiver Vorzug des Schneckengetriebes ist seine Selbsthemmung. Diese verhindert, dass der Gegendruck der Spenderfeder das System unbeabsichtigt rückwärts antreibt.

Im letzten Schritt muss die verstärkte Rotationsbewegung schließlich in eine Linearbewegung umgewandelt werden. Hierzu treibt das Schneckengetriebe ein weiteres Zahnrad an, das direkt in eine vertikale Zahnstange greift. Durch dieses Zahnstangengetriebe wird die Drehbewegung in eine geradlinige Abwärtsbewegung umgewandelt. Diese drückt den Hebelarm und somit den Spenderkopf mit den erforderlichen $20{,}6\,\mathrm{N}$ kontinuierlich hinab und löst so den Fördervorgang der Seife aus (siehe Abb. 3).

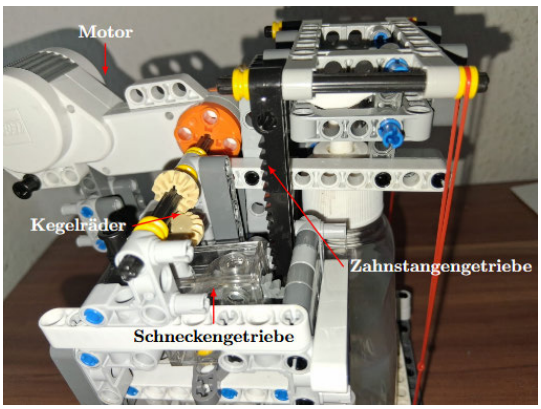


Abbildung 3. Mechanischer Aufbau der Kraftübertragungseinheit

C. Funktionsprinzip der Bilderkennung

Zur berührungslosen Steuerung des Seifenspenders wird eine handelsübliche Webcam eingesetzt (siehe Abb. 4). Diese erfasst kontinuierlich den Bereich unter dem Pumpkopf. Die bloßen Videodaten müssen anschließend von einer Software ausgewertet werden, um eine im Bild befindliche Hand zuverlässig zu identifizieren.

Abbildung 4. Webcam (Logitech C270) [4]

Hierfür kommt das KI-basierte Framework MediaPipe zum Einsatz. Anstatt fehleranfällige Merkmale wie Hautfarben zu suchen, analysiert dieses System die Geometrie im Bild. Mithilfe der sogenannten Landmark-Erkennung sucht die Software gezielt nach den typischen Gelenkpunkten und der Struktur einer menschlichen Hand (siehe Abb. 5 aus der Quelle [2]). Sobald das „Skelett“ einer Hand im Videostream eindeutig erkannt wird, wird ein Signal an die Steuereinheit gesendet, um den Motor zu starten. Dieser Ansatz ist sehr robust gegenüber veränderten Lichtverhältnissen und störenden Objekten im Hintergrund.

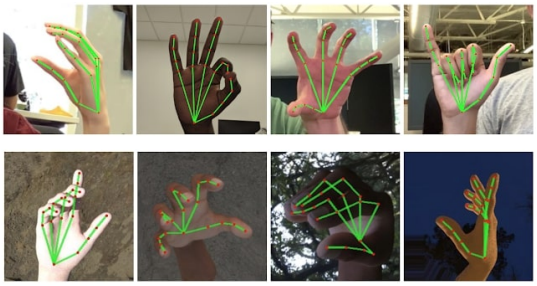


Abbildung 5. Eingabedaten des Tracking-Netzwerks: Reale (oben) und synthetische Handbilder (unten) mit markierten Gelenkpunkten [2]

III. Konstruktion & Programmierung

Hier wird detailliert beschrieben, wie der Roboter gebaut und programmiert wurde.

A. Aufbau

Der mechanische Aufbau dient dazu, den traditionellen Seifenspender sicher aufzunehmen und die Rotationsbewegung des Motors in eine lineare Abwärtsbewegung umzuwandeln. Wie bei der Kraftmessung festgestellt wurde, tritt beim Herunterdrücken des Pumpkopfs ein sehr hoher mechanischer Widerstand auf. Um zu verhindern, dass die LEGO-Konstruktion unter dieser Belastung nachgibt oder sich verformt, wurde ein fester, stabilisierender Rahmen, ein sogenannter Käfig, konstruiert (siehe Abb. 6). Dieser umschließt den Seifenspender formschlüssig und fixiert ihn gegen seitliches Verrutschen.

Oberhalb des Pumpkopfs ist die Kraftübertragungseinheit montiert. Dabei ist der Elektromotor über das Schneckengetriebe mit einer Mechanik verbunden, die exakt auf den Kopf des Spenders drückt. Der stabilisierende Käfig gewährleistet, dass die erzeugte Druckkraft zuverlässig und zielgenau auf den Pumpmechanismus übertragen wird.

Abbildung 6. stabilisierender Rahmen (Käfig)

B. Ablaufplan

Der kontinuierliche Betriebsablauf des Seifenspenders lässt sich in mehrere klar definierte Phasen unterteilen (siehe Abb. 7). Nach der erfolgreichen Initialisierung der Hardwarekomponenten und Softwareschnittstellen (NXT-Controller, Webcam und MediaPipe-Umgebung) wechselt das System in einen dauerhaften Überwachungsmodus. In dieser Phase wird das Kamerabild zyklisch erfasst und auf die vordefinierten Hand-Landmarks geprüft. Die effiziente Architektur von MediaPipe ermöglicht dabei eine Verarbeitung in Echtzeit (typischerweise unter $50\,\mathrm{ms}$ pro Frame [2]), wodurch das System nahezu verzögerungsfrei auf den Nutzer reagiert. In Bezug auf die Zuverlässigkeit erweist sich das System als äußerst robust gegenüber Fehlauslösungen durch andere Objekte. Da der Algorithmus gezielt die geometrische Skelettstruktur einer menschlichen Hand verifizieren muss, führen unbelebte Gegenstände oder Kleidung nicht zu einer Aktivierung des Motors. Wird keine Hand detektiert, verbleibt der Motor im Ruhezustand. Sobald jedoch eine Hand im Erfassungsbereich verifiziert wird, startet der Ausgabeprozess. Der Motor dreht sich zunächst vorwärts, um den Pumpkopf zu betätigen, und fährt den Hebelarm anschließend wieder in die Ausgangsposition zurück. Um eine Mehrfachausgabe zu verhindern, geht das System anschließend in eine blockierende Warteschleife über. Erst

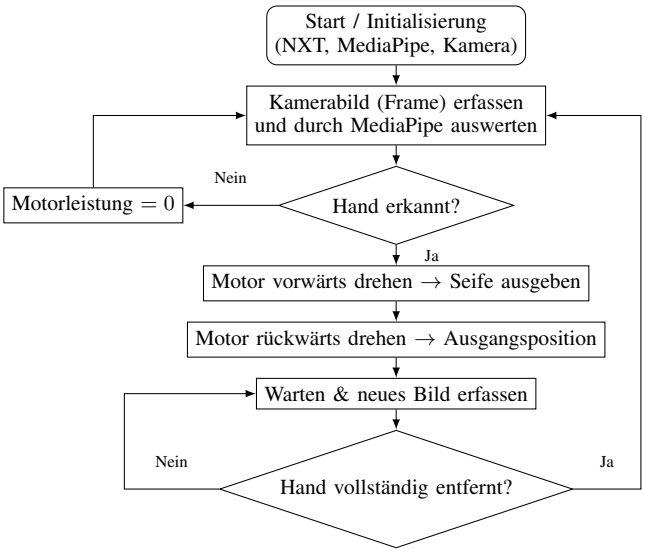


Abbildung 7. Programmablaufplan der Seifenspender-Steuerung

wenn die Hand vollständig aus dem Bildbereich entfernt wurde, wird der Zyklus abgeschlossen und der ursprüngliche Überwachungsmodus wieder aufgenommen.

C. Algorithmus

Die programmiertechnische Realisierung erfolgt in der Entwicklungsumgebung MATLAB. Dabei wird das Python-basierte Framework MediaPipe über die Schnittstelle `py.importlib` direkt eingebunden. Der Kernalgorithmus erfasst in einer Endlosschleife kontinuierlich Kamerabilder und übergibt sie an das Hand-Erkennungsmodell. Sobald eine Hand verifiziert wurde, erfolgt die Ansteuerung des Elektromotors mit einem Rotationslimit von $5000°$. Um eine ungewollte Mehrfachausgabe zu verhindern, blockiert eine Kontrollschleife eine erneute Motorauslösung, bis in drei aufeinanderfolgenden Kamerabildern keine Hand mehr detektiert wird (`confirmFrames` = 3).

D. Herausforderungen

Die zuverlässige Verarbeitung der Webcam-Daten stellte eine wesentliche Herausforderung dar. In einer ersten Entwicklungsphase wurde eine grundlegende Farbraumerkennung erprobt, bei der die Software auf bestimmte Hauttöne im Bild reagieren sollte. Dieser Ansatz erwies sich in der Praxis jedoch als sehr fehleranfällig: Wechselnde Lichtverhältnisse, Schatten oder hautfarbene Objekte im Hintergrund führten zu unkontrollierbaren Fehlauslösungen.

Deshalb wurde das System grundlegend überarbeitet und auf das KI-gestützte Framework MediaPipe umgestellt, um eine störungsfreie Funktion zu gewährleisten. Das Problem der Lichtabhängigkeit konnte vollständig beseitigt werden, indem von einer rein farbbasierten zu einer strukturbasierten Erkennung (Landmark-Erfassung) gewechselt wurde. Die Auslösung erfolgt nun ausschließlich, wenn die geometrische Skelettstruktur einer menschlichen Hand zweifelsfrei vom Algorithmus verifiziert wird. Dadurch wird die Fehlerquote auf ein Minimum reduziert.

IV. Ergebnisdiskussion

Die praktischen Tests ergaben, dass der Prototyp die Projektziele erfolgreich erfüllt. Die KI-gestützte Steuerung mittels MediaPipe erwies sich gegenüber konventionellen, ultraschallbasierten Automatikspendern als deutlich besser: Da das System ausschließlich auf die geometrischen Strukturmerkmale einer Hand reagiert, konnten Fehlauslösungen, beispielsweise durch vorbeistreifende Kleidung, vollständig eliminiert werden.

Auf mechanischer Ebene zeigte sich, dass der stabile Rahmen in Kombination mit dem Schneckengetriebe die erforderliche Druckkraft von $20{,}6\,\mathrm{N}$ einwandfrei auf den Pumpkopf überträgt. Dabei erwiesen sich die festgelegten 14 Motorumdrehungen als optimal für eine konstante und alltagstaugliche Seifenabgabe.

Ein einschränkender Faktor bleibt jedoch die Hardware-Abhängigkeit. Die rechenintensive Bildauswertung über MATLAB erfordert zwingend einen angeschlossenen Computer. Dies schränkt die Mobilität des Spenders ein und verursacht eine minimale Zeitverzögerung bei der Motorauslösung. Dennoch zeigen die Ergebnisse eindrucksvoll, dass dieses KI-gestützte, mechatronische System machbar und hoch zuverlässig ist.

V. Zusammenfassung und Fazit

Das Ziel des Projekts war die Automatisierung eines herkömmlichen Seifenspenders, um eine hygienische und kontaktlose Bedienung zu ermöglichen.

Dies wurde durch eine stabile LEGO-Käfigkonstruktion mit Schneckengetriebe sowie eine kamerabasierte KI-Steuerung (MediaPipe) in MATLAB erfolgreich umgesetzt. Das System überwindet den hohen mechanischen Widerstand von $20{,}6\,\mathrm{N}$ zuverlässig und erkennt Hände unabhängig von den Lichtverhältnissen robust. Für zukünftige Entwicklungen bestehen jedoch noch Optimierungsmöglichkeiten. So muss der Prototyp zur Bildauswertung derzeit an einen Computer angeschlossen sein, weshalb ein komplett eigenständiges System ohne externen PC der nächste sinnvolle Schritt wäre. Zudem fehlt aktuell eine automatische Füllstandskontrolle. Durch den Einbau eines weiteren Sensors könnte der Nutzer künftig rechtzeitig an das Nachfüllen der Seife erinnert werden.

LiteraturLiteraturverzeichnis

Literatur

[1] Wikipedia, die freie Enzyklopädie: *Schneckengetriebe*. https://de.wikipedia.org/wiki/Schneckengetriebe, abgerufen am 26. Februar 2026.

[2] Valentin Bazarevsky, Fan Zhang: *On-Device, Real-Time Hand Tracking with MediaPipe*. Google Research Blog, 2019. https://research.google/blog/on-device-real-time-hand-tracking-with-mediapipe/

[3] Steinpalast: *Lego Technic Getriebe Halter mit Schnecke und Zahnrad Z24*. Produktdatenblatt, 2026. https://www.steinpalast.eu/1-x-lego-technic-getriebe-halter-transparent-weiss-2x4x3-1/3-gewinde-schnecke-lang-achs-loch-stange-zahnrad-z24-neu-hell-grau-3648-24505-4716-6588-32239

[4] Bürklin Elektronik: *Logitech C920 HD Pro Webcam*. Produktspezifikationen, 2026. https://www.buerklin.com/de/p/logitech/kameras-zubehoer-kameras/960-001063/12M4243/

Tic-Tac-Toe-Roboter

Marko Gaube, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—In dieser Arbeit wird die Entwicklung und Realisierung eines LEGO-Roboters vorgestellt, der in der Lage ist, das Spiel Tic-Tac-Toe gegen einen menschlichen Gegner auf einem Blatt Papier zu spielen. Das Spielfeld wird mittels einer Kamera erfasst und durch einen Bildverarbeitungsalgorithmus ausgewertet, wobei die Erkennung der Spielerzüge auf einer Farbanalyse basiert. Zur Bestimmung des optimalen Spielzugs des Roboters kommt ein rekursiver Minimax-Algorithmus zum Einsatz. Die physische Umsetzung der Spielzüge erfolgt durch einen dreiachsigen Plotter auf Basis eines kartesischen Koordinatensystems. Grundlage bildet das LEGO-Mindstorms-System. Die Programmierung erfolgt in MATLAB.**

***Schlagwörter*—Bilderkennung, Minimax-Algorithmus, Plotter, Roboter, Tic-Tac-Toe**

I. Einleitung

ZIEL dieses Projekts ist die Entwicklung eines Roboters, der in der Lage ist, das Spiel Tic-Tac-Toe gegen einen menschlichen Gegner zu spielen. Das Spiel wird dabei konventionell auf einem Blatt Papier durchgeführt. Daraus ergibt sich die Anforderung, dass der Roboter einen menschlichen Spielzug visuell erfassen, diesen verarbeiten und auf Basis des aktuellen Spielzustands einen eigenen Zug generieren muss. Weiterhin soll der Roboter in der Lage sein, das Spielfeld vor dem eigentlichen Spielbeginn selbstständig zu zeichnen sowie den Spielverlauf zu analysieren, um das Spielende und einen möglichen Gewinner korrekt zu erkennen.

Zur Umsetzung dieser Anforderungen wird das Spielfeld mithilfe einer Kamera erfasst. Der aktuelle Zustand des Spielfelds wird anschließend durch einen in MATLAB implementierten Algorithmus ausgewertet, welcher den optimalen nächsten Zug berechnet. Der physische Eintrag des berechneten Zuges erfolgt mittels eines Plotters auf dem Papier. Der fertige Roboter ist in Abbildung 1 dargestellt. Im Folgenden wird die konkrete Umsetzung erläutert.

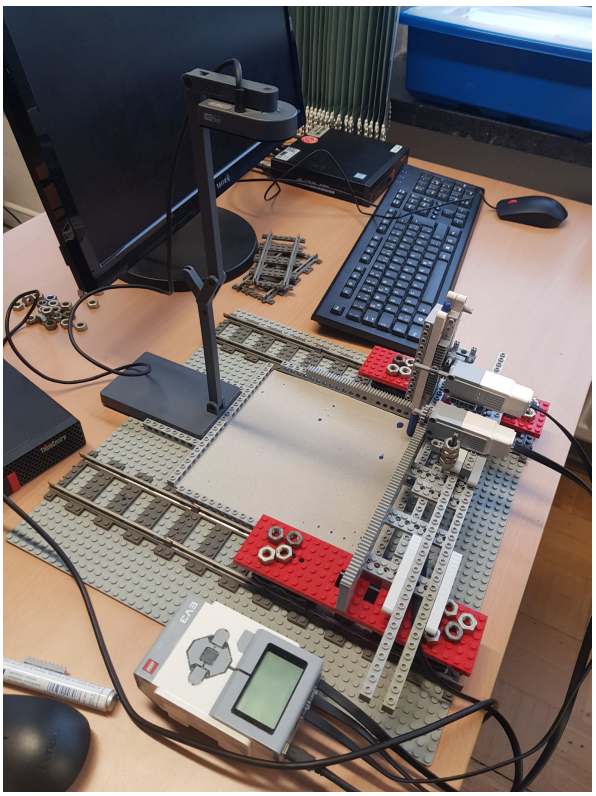


Abbildung 1. Fertiger Tic-Tac-Toe-Roboter

II. Vorbetrachtungen

Das Spiel Tic-Tac-Toe ist allgemein bekannt und zeichnet sich durch einfache Regeln sowie eine überschaubare Spielstruktur aus. Trotz dieser Einfachheit ergeben sich bei der technischen Umsetzung als Tic-Tac-Toe-Roboter vielfältige Herausforderungen.

Im Rahmen des Projektseminars Elektrotechnik/Informationstechnik an der Otto-von-Guericke-Universität Magdeburg (OVGU) wurden bereits zahlreiche Realisierungskonzepte für Tic-Tac-Toe-Roboter sowie für Schreibroboter entwickelt. Ziel dieses Projekts ist es, beide Ansätze in einem integrierten System zu vereinen.

Der softwareseitige Schwerpunkt liegt auf der Implementierung des vollständigen Spielablaufs, einschließlich einer optimalen Zugstrategie für den Roboter wie in [1]. Die zentrale mechanische Herausforderung besteht hingegen in der Konstruktion eines zuverlässigen Plotters. Darüber hinaus soll durch die Einbindung einer Kamera ein erster Einblick in die Bildverarbeitung mit MATLAB ermöglicht werden.

DOI: 10.24352/UB.OVGU-2026-042

III. Umsetzung

A. Konstruktion des Plotters

Wie Abbildung 2 zeigt, basiert der Plotter auf einem dreidimensionalen kartesischen Koordinatensystem, wobei das Spielfeld in der xy-Ebene liegt. Dieses befindet sich direkt auf einer LEGO-Basisplatte, die die Grundlage der gesamten Konstruktion bildet. Zur Positionierung des Stiftes ist neben der Navigation in x- und y-Richtung zusätzlich eine Bewegung entlang der z-Achse erforderlich, um den Stift anzuheben und abzusetzen.

Für jede Raumrichtung wird ein separater Motor eingesetzt. Durch die Verwendung der kleinen LEGO-Motoren kann der Plotter kompakter gebaut und das Gewicht reduziert werden. Die Kraftübertragung erfolgt jeweils über Zahnräder, die

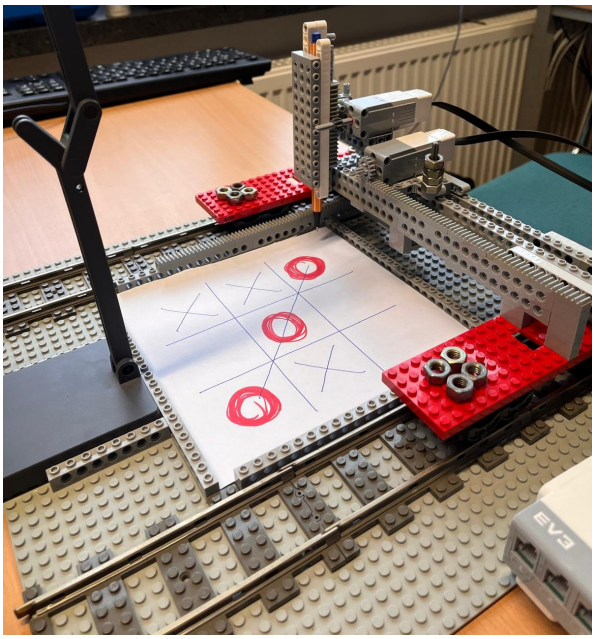


Abbildung 2. Der Plotter basiert auf einem kartesischen Koordinatensystem. Die Spitze des Stiftes befindet sich im Koordinatenursprung $(0, 0, 0)$.

in Zahnstangen eingreifen und so eine lineare Bewegung erzeugen. Dabei bewegt `motorX` einen Schlitten entlang der an beiden Seiten des Spielfeldes montierten LEGO-Schienen, um eine Auslenkung in x-Richtung zu ermöglichen. Auf diesem Schlitten ist eine weitere Führungsschiene aus LEGO-Steinen angebracht, auf der der `motorY` läuft. Der `motorZ` ist wiederum direkt am `motorY` befestigt und ermöglicht das Aufsetzen und Anheben des Filzstifts mittels einer Führung parallel zur z-Achse.

Die Positionssteuerung erfolgt durch die Übergabe von Drehwinkeln an die jeweiligen Motoren. Aufgrund der mechanischen Übersetzung zwischen Zahnrad und Zahnstange ergibt sich daraus eine lineare Verschiebung entlang der entsprechenden Achse, die proportional zum Drehwinkel ist. Die maximal zulässige Auslenkung in jede Richtung wird softwareseitig durch in Variablen gespeicherte Grenzwerte festgelegt. An die implementierte Funktion `movePen()` werden Bruchteile dieser Grenzwerte übergeben, welche in einer Auslenkung um eine entsprechende Teilstrecke innerhalb des definierten Spielfeldes resultieren.

B. Spielablauf

Der Programmablauf in der Hauptdatei `main` wird in Abbildung 3 verdeutlicht. Die `main`-Datei fasst den gesamten Spielprozess zusammen und koordiniert sämtliche Funktionsaufrufe.

Zu Beginn erfolgt die Initialisierung des Systems. In diesem Schritt werden der LEGO-EV3-Stein mit den drei Motoren sowie die Kamera in das Programm eingebunden. Zusätzlich wird eine 3×3-Matrix `playingField` als zentrale Datenstruktur initialisiert, die den aktuellen Zustand des Spielfeldes repräsentiert. Die Matrixeinträge sind wie folgt codiert: Der Wert 0 kennzeichnet ein freies Feld, 1 ein vom Roboter gesetztes Kreuz und 2 einen vom menschlichen Spieler gesetzten Kreis. Das Array `freeFields` speichert alle freien Felder. Außerdem wird die Größe des Spielfelds definiert und das Raster durch den Plotter auf das Papier aufgebracht.

Der eigentliche Spielablauf ist als Endlosschleife implementiert, das heißt, die Bedingung im Kopf der while-Schleife wurde auf `true` gesetzt. Diese wird nur beim Eintreten einer Abbruchbedingung innerhalb der Schleife verlassen. Das Spiel beginnt mit dem Zug des Roboters. Hierzu wird mit der Matrix `playingField` der aktuelle Spielfeldzustand an den Minimax-Algorithmus übergeben, der den optimalen Zug bestimmt. Die berechnete Position wird in der Matrix mit dem Wert 1 aktualisiert und physisch als Kreuz auf das Spielfeld geplottet. Anschließend erfolgt eine Überprüfung auf die Abbruchbedingungen. Diese gelten als erfüllt, wenn

1) eine Gewinnkonstellation für den Roboter vorliegt (wenn der Wert 1 dreimal in einer Zeile, Spalte oder Diagonale der Matrix auftritt), oder
2) keine freien Felder mehr verfügbar sind (`freeFields` keine Elemente enthält).

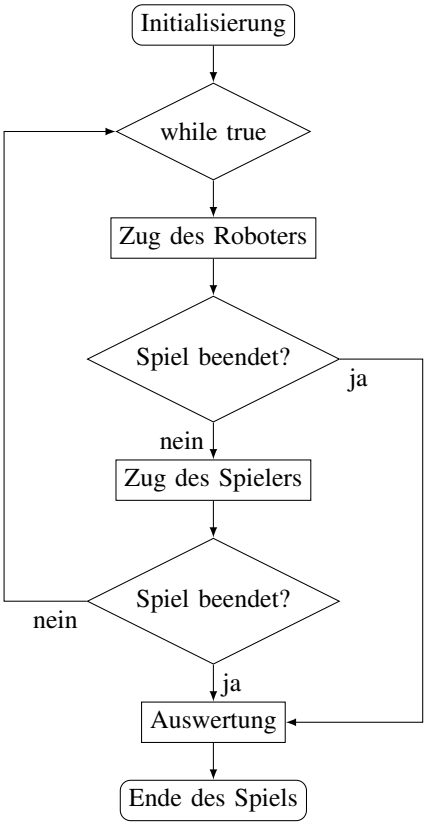


Abbildung 3. Programmablaufplan für den Spielablauf in der `main`-Datei

Ist das Spiel noch nicht beendet, folgt der Zug des menschlichen Spielers. Dieser markiert sein Feld durch das Zeichnen eines roten Kreises auf dem Papier. Die Position des Kreises wird mittels Kamerabild und eines Algorithmus zur Bilderkennung erfasst. Der erkannte Zug wird in der Matrix mit dem Wert 2 gespeichert. Anschließend werden analog zum vorherigen Schritt die Abbruchbedingungen geprüft. Allerdings

wird die Matrix `playingField` auf den Wert 2 hin überprüft, um einen eventuellen Gewinn des Spielers festzustellen.

Beim Eintreten einer Abbruchbedingung wird die Schleife durch eine `break`-Anweisung verlassen und eine abschließende Auswertung durchgeführt. Im Falle eines Gewinns („drei in einer Reihe“) markiert der Plotter diesen durch das Ziehen eines Strichs und gibt den Gewinner im MATLAB-Befehlsfenster aus; andernfalls wird das Spiel als Unentschieden beendet.

C. Bilderkennung

Zu Beginn des Spielerzugs wird der Programmablauf für 15 Sekunden pausiert, um dem menschlichen Spieler die Möglichkeit zu geben, seinen Zug durch das Einzeichnen eines roten Kreises auf dem Spielfeld zu kennzeichnen. Anschließend wird mithilfe der oberhalb des Spielfelds angebrachten Kamera eine Aufnahme des gesamten Spielfelds erstellt. Die Kamera kann durch die „Image Acquisition Toolbox“ in das Programm integriert werden [2].

Weiterhin stellt MATLAB Funktionen zur rudimentären Bilduntersuchung bereit [3]. Jedem Pixel eines Bildes kann eine eindeutige x- und y-Koordinate zugeordnet werden, welche seine Position innerhalb der Bildebene beschreibt. Dadurch ist eine feldweise Bildauswertung möglich, indem Bildausschnitte an vordefinierten Positionen, die mit den einzelnen Spielfeldern übereinstimmen, ausgewertet werden. Die Positionsbestimmung anhand der x- und y-Koordinaten wird in Abbildung 4 verdeutlicht.

Für die Detektion eines Spielerzugs wird nicht die geometrische Form des Kreises analysiert, sondern ausschließlich dessen Farbwert. In jedem definierten Bildausschnitt wird die Anzahl roter Pixel ermittelt. Ein Feld wird als gültiger Spielerzug gewertet, wenn darin mehr als 200 rote Pixel gezählt werden. Ein Pixel wird dabei als „rot“ klassifiziert, wenn es einen RGB-Farbcode von $(R > 150,\ G < 100,\ B < 100)$ aufweist. Das entsprechend identifizierte Feld wird im Array `playingField` mit dem Wert 2 markiert.

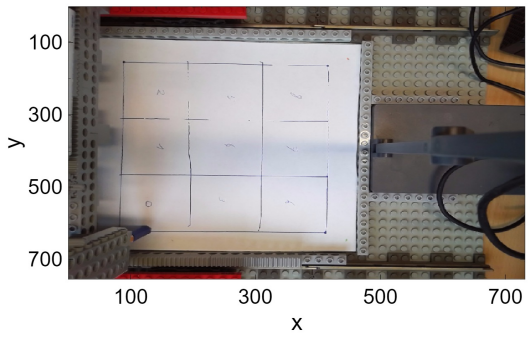


Abbildung 4. Kamerabild des Spielfelds mit geplottetem Raster. Die Positionsbestimmung der Bildausschnitte erfolgt mithilfe der Funktion `impixelinfo()`.

D. Minimax-Algorithmus

Tic-Tac-Toe zählt zu den diskreten Zwei-Spieler-Spielen. Diese Klasse von Spielen kann durch einen Minimax-Algorithmus optimal gelöst werden [4]. Die Implementierung des Algorithmus orientiert sich an dem in [5] diskutierten Programmierbeispiel und wird im Weiteren erläutert:

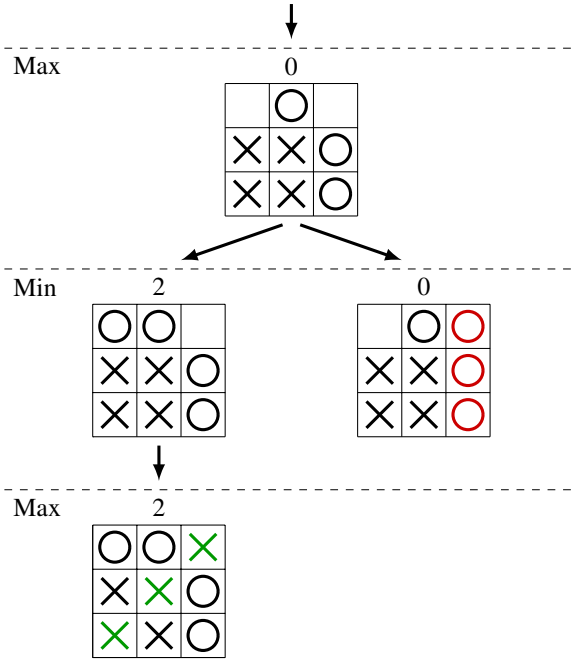


Abbildung 5. Ausschnitt aus einem Spielbaum mit Visualisierung des Minimax-Algorithmus, grün: Gewinn des Computers, rot: Gewinn des Gegenspielers

Der aktuelle Spielzustand wird dem Minimax-Algorithmus als Matrix `playingField` übergeben. Dieser Zustand wird bewertet. Dabei wird ein Gewinn des Roboters mit dem Wert 2, ein Unentschieden mit dem Wert 1 und ein Gewinn des menschlichen Spielers mit dem Wert 0 gleichgesetzt.

Liegt kein terminaler Spielzustand vor, erfolgt keine unmittelbare Bewertung. Stattdessen generiert der Algorithmus rekursiv alle möglichen Folgezustände, indem er sämtliche zulässigen Züge simuliert. Für jeden dieser Zustände wird der Minimax-Algorithmus erneut aufgerufen, sodass virtuell ein vollständiger Spielbaum erstellt wird.

Die Auswahl des optimalen Zugs erfolgt gemäß der Minimax-Strategie: Befindet sich der Roboter am Zug (Max-Knoten), wird der maximale Bewertungswert der Folgezustände gewählt. Ist hingegen der menschliche Spieler am Zug (Min-Knoten), wird der minimale Bewertungswert ausgewählt. Dieses Vorgehen basiert auf der Annahme, dass beide Spieler optimal spielen.

Wie in Abbildung 5 verdeutlicht, werden dadurch Spielzüge vermieden, die zwar schließlich zu einem Gewinn führen würden, jedoch dem Gegner einen früheren Gewinn ermöglichen.

IV. Ergebnisdiskussion

Zum Abschluss des Projekts war der Roboter in der Lage, mehrere Spielabläufe zuverlässig nacheinander auszuführen. Dessen grundsätzliche Funktionsfähigkeit konnte damit bestätigt werden.

Die softwareseitige Umsetzung des Plotters gestaltet sich vergleichsweise einfach. Da dieser auf einem kartesischen

Koordinatensystem basiert und ausschließlich Geraden zeichnet, ist die Ansteuerung der Motoren mit geringem mathematischen Aufwand umsetzbar. Demgegenüber weist die mechanische Konstruktion eine höhere Fehleranfälligkeit auf. Insbesondere auftretende Reibungskräfte an den Führungsschienen sowie zwischen Stift und Papier können die Positionsgenauigkeit beeinträchtigen. Durch die Verwendung von LEGO-Schienen ließ sich diese Problematik reduzieren. Zusätzlich führte das Anbringen von Gewichten zu einer erhöhten Bahnstabilität und damit zu einer verbesserten Präzision des Plotters.

MATLAB stellt mit der „Image Acquisition Toolbox“ umfangreiche Funktionen zur Verfügung, die die Bildverarbeitung ermöglichen. Bei der Implementierung der Zugerkennung zeigte sich jedoch, dass ein geeigneter RGB-Schwellenwert und Grenzwert für die Anzahl roter Pixel stark von der Wahl der Kamera, der Stiftfarbe und der Beleuchtungssituation abhängen. Daher mussten diese Parameter empirisch bestimmt werden, um eine zuverlässige Spielerzugerkennung zu gewährleisten.

V. Zusammenfassung und Fazit

Das zu Beginn formulierte Ziel, einen Roboter zu bauen, der ein komplettes Tic-Tac-Toe-Spiel auf Papier gegen einen Menschen spielt, wurde erreicht. Dazu wurden hauptsächlich folgende Komponenten verwendet:

1) LEGO-EV3-Stein
2) drei kleine LEGO-Motoren
3) Dokumentenkamera
4) MATLAB-Software
5) LEGO-Steine
6) LEGO-Schienen

Da der Roboter durch den Minimax-Algorithmus optimal spielt, kann dieser nicht besiegt werden. Allenfalls ist ein Unentschieden bei einer ebenfalls optimalen Strategie des menschlichen Gegenspielers erreichbar.

Darüber hinaus wäre z. B. zusätzlich eine Auswahl des beginnenden Spielers und eine Einstellung für den Schwierigkeitsgrad des Roboters denkbar. Außerdem wäre eine Erweiterung auf ein „Vier Gewinnt“-Spiel realisierbar, da sowohl die Ansteuerung des Plotters als auch der Minimax-Algorithmus entsprechend leicht adaptiert werden könnten.

Literaturverzeichnis

[1] Schulz, Karsten: *Der Tic-Tac-Toe-Robo*. https://journals.ub.ovgu.de/index.php/LEGO/article/view/2262/2242. Version: Februar 2026

[2] The MathWorks, Inc.: *Image Acquisition Toolbox — Functions*. https://de.mathworks.com/help/releases/R2025b/imaq/referencelist.html?type=function&s_tid=CRUX_topnav. Version: Februar 2026

[3] The MathWorks, Inc.: *Build Interactive Tools — Functions*. https://de.mathworks.com/help/images/referencelist.html?type=function&s_tid=CRUX_topnav&category=building-guis-with-modular-interactive-tools. Version: Februar 2026

[4] Nopper, Tobias: *Prinzip MinMax-Algorithmus am Beispiel von Tic Tac Toe | Wie spielen Computer?* https://youtu.be/3ufcmCpKb6w. Version: Februar 2026

[5] Nopper, Tobias: *Implementierung des MinMax-Algorithmus für Tic Tac Toe | Wie spielen Computer?* https://youtu.be/ODgPsXssUvk. Version: Februar 2026

Automatisierung des Spiels Tic-Tac-Toe

Finn Löffler, Elektromobilität
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Dieses Paper beschreibt die Entwicklung eines Tic-Tac-Toe-Roboters während eines zweiwöchigen Erstsemester-Projektseminars. Das Ziel war ein System, das selbstständig gegen einen Menschen spielen kann. Das Paper zeigt den Verlauf von einem einfachen Modell mit Tastaturbedienung und Zufallszügen hin zu einem System mit automatischer Farberkennung mittels Kamera und der Implementierung eines Minimax-Algorithmus zur Bestimmung des optimalen Spielzugs des Roboters. Im vorgegebenen Zeitrahmen konnte ein funktionsfähiger Roboter realisiert werden, der Spielzüge erkennt und eigenständig auf Papier ausführt.**

***Schlagwörter*— Bilderkennung, Minimax-Algorithmus, Plotter, Roboter, Tic-Tac-Toe**

I. Einleitung

Im Rahmen des zweiwöchigen Projektseminars wurde die Aufgabe gestellt, einen Roboter zu entwickeln. Die Wahl fiel auf einen Tic-Tac-Toe-Roboter, da dieses Projekt eine spannende Kombination aus Mechanik und Programmierung darstellt. Das Ziel bestand darin, einen Roboter zu konstruieren, der gegen einen menschlichen Gegner antreten kann. Das fertige System ist in Abbildung 1 dargestellt. In diesem Bericht wird beschrieben, wie der Roboter geplant, umgesetzt und schrittweise verbessert wurde, um eine funktionierende Spielumgebung zu realisieren.

II. Vorbetrachtungen

Als Hardware-Plattform wurde das LEGO-Mindstorms-Bausystem vorgegeben. Die mechanische Umsetzung erfolgte in Form eines Plotters. Da das Thema Tic-Tac-Toe bereits in früheren Seminaren behandelt wurde (siehe [1]), lag die Herausforderung in der Integration von Bilderkennung und Spielalgorithmus in ein stabiles mechanisches System. Im Prozess wurde die Steuerung von einer manuellen Tastatureingabe auf eine automatisierte Erkennung mittels Dokumentenkamera umgestellt und die Spiel-Logik sukzessive von Zufallszügen auf eine strategische Spielweise optimiert.

III. Umsetzung

A. Konstruktion

Bei der Konstruktion des Tic-Tac-Toe-Roboters wurde sich für ein Plotter-Design entschieden. Dies hat wesentliche Vorteile: Einerseits wurde hierdurch eine gleichbleibend hohe Qualität der Spielabläufe gewährleistet, da sich ein identischer Startpunkt für jeden Spieldurchlauf festlegen ließ. Andererseits konnte durch das gewählte Design die Reibung auf ein Minimum reduziert werden.
Der Plotter verfügte über drei Achsen, die jeweils von einem kleinen Motor angetrieben wurden, wie in Abbildung 2 zu sehen ist. Die Kraftübertragung erfolgte jeweils über Zahnräder, die in Zahnstangen eingriffen und so eine lineare Bewegung erzeugten.

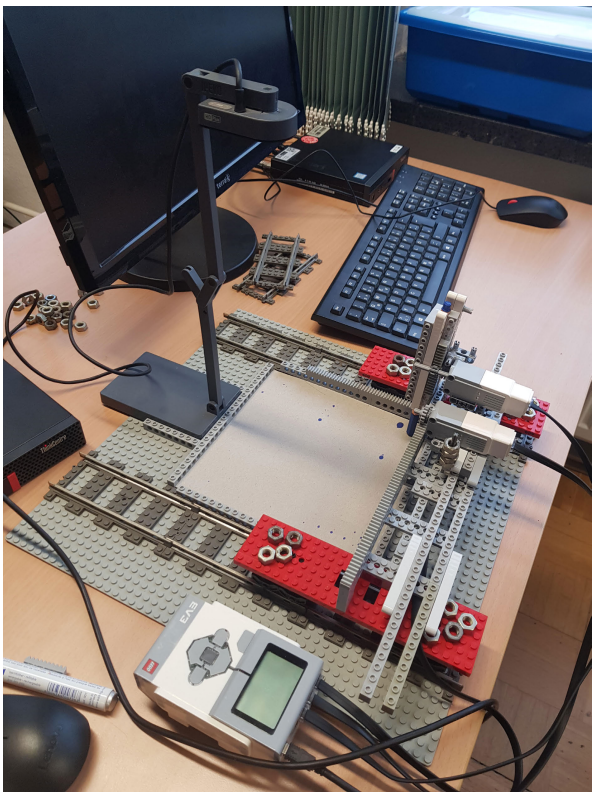


Abbildung 1. Fertiger Tic-Tac-Toe-Roboter

Der Roboter navigierte auf der x- und y-Achse über das Spielfeld, während der Stift auf der z-Achse angehoben und gesenkt wurde. Mit dem Stift wurde das Spielfeld sowie die Spielzüge des Computers aufgezeichnet. Aufgrund der niedrigen Reibung des gewählten Designs konnten kleine LEGO-Mindstorms-Motoren verwendet werden. Diese ließen sich aufgrund ihrer Größe und Form besonders leicht in das Design integrieren. Zur weiteren Reduzierung der Reibung wurden Schienen für die Bewegung auf der x-Achse verbaut und auf einer quadratischen LEGO-Grundplatte befestigt, was die Stabilität des Roboters zusätzlich erhöhte.
Der Stift wurde in einer Vorrichtung fixiert, welche die vertikale Bewegung präzise ausführte. Im weiteren Projektverlauf wurde eine Kamera ergänzt; um den Bildausschnitt konstant zu halten, wurde auch diese mit flachen LEGO-Steinen auf der Grundplatte fixiert. Da eine maßgeschneiderte Lösung zeitlich

DOI: 10.24352/UB.OVGU-2026-043

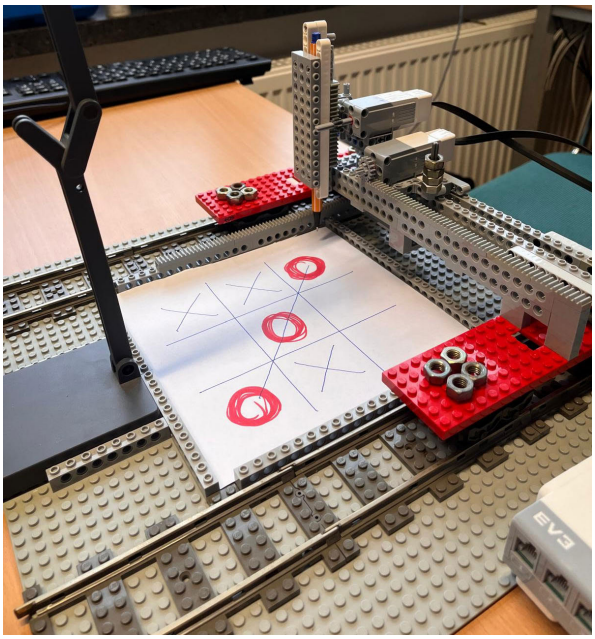


Abbildung 2. Konstruktion des Plotters

nicht mehr umsetzbar war, wurden Muttern als unfixierte Gewichte verwendet, um die Qualität der Linienzüge zu gewährleisten. Hätte mehr Zeit zur Verfügung gestanden, hätten die Muttern durch eine technisch ausgereiftere Lösung ersetzt werden können.

B. Spielverlauf

Der Aufbau der Software, deren Logik durch den Programmablaufplan in Abbildung 3 verdeutlicht wird, basierte auf einem modularen Konzept. Dabei wurden sämtliche Abläufe, wie etwa die Erstellung des Spielfelds, in eigenständige Funktionen ausgelagert und erst innerhalb des Hauptprogramms miteinander verknüpft. Zu Programmbeginn wurde die EV3-Steuereinheit initialisiert, um potenziellen Fehlfunktionen der Winkelbegrenzung vorzubeugen. Zur Steuerung der Motoren wurden zudem sämtliche Parameter für Winkel und Geschwindigkeiten zentral in der Hauptfunktion als Variablen definiert. Diese Methode steigerte die Effizienz bei Anpassungen und minimierte das Fehlerrisiko, da Änderungen nicht in jeder Funktion einzeln vorgenommen werden mussten. Abschließend wurde eine 3×3-Matrix generiert, die als digitales Abbild des Spielfelds diente und den Wert 0 als Platzhalter für unbesetzte Felder nutzte.
Im darauffolgenden Schritt erfolgte die grafische Ausgabe des Spielfelds. Der Computer initiierte das Spiel, indem er seinen Zug vollzog und ein Kreuz in eines der neun unbesetzten Felder zeichnete. Dieser Vorgang wurde digital in der Matrix durch den Wert 1 sowie eine entsprechende Ausgabe in der Kommandozeile gekennzeichnet.
Im Anschluss setzte der Spieler seinen Zug, indem er einen roten Kreis in eines der verbliebenen freien Felder zeichnete, was systemseitig als 2 registriert wurde. Sobald der Spieler seinen Zug beendet hatte, identifizierte der Roboter das entsprechende

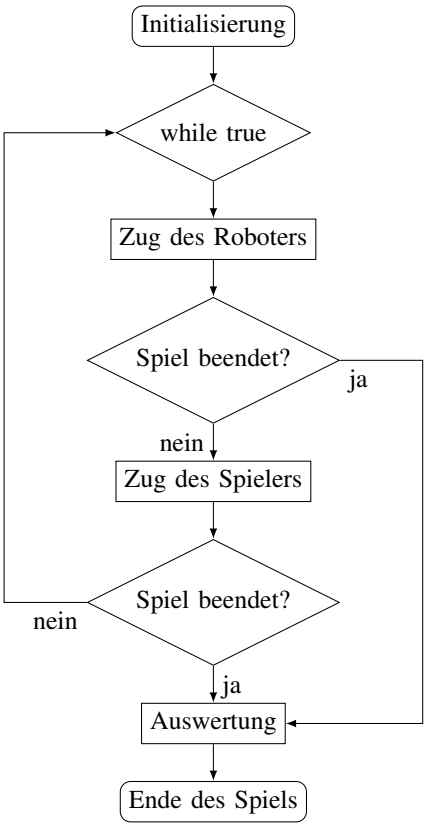


Abbildung 3. Programmablaufplan für den Spielverlauf im Hauptprogramm

Feld und reagierte mithilfe des Minimax-Algorithmus optimal auf die Eingabe. Die Spielroutine wurde so lange fortgesetzt, bis entweder eine der beiden Parteien den Sieg errang oder durch die Belegung sämtlicher Felder ein Unentschieden eintrat. Im Falle eines Sieges zeichnete der Roboter eine Linie durch die drei gewinnbringenden Felder und gab den Gewinner in der Kommandozeile aus. Standen keine freien Felder mehr zur Verfügung, deklarierte das System die Partie als Unentschieden.

C. Bilderkennung

Durch den Einsatz einer Dokumentenkamera wurde die eigenständige Erkennung sowie die Reaktion auf den Spielzug des menschlichen Spielers ermöglicht. Die Kamera konnte mithilfe der „Image Acquisition Toolbox“ in das Programm integriert werden [2]. Darüber hinaus stellt MATLAB Funktionen zur Bildanalyse bereit [3].
Der Bildausschnitt wurde zunächst digital unter Verwendung der Bildkoordinaten in neun gleich große Felder unterteilt. Diese Felder stellten in ihrer Gesamtheit das Spielfeld dar. Es wurde dabei auf die gleiche Ausrichtung der Kamera über mehrere Partien hinweg geachtet, damit die über die Koordinaten festgelegte Lage der Felder stets mit der realen Lage der Felder auf dem Spielfeld übereinstimmte. Der durch die Dokumentenkamera erfasste Ausschnitt ist in Abbildung 4 dargestellt.
Nachdem der Roboter seinen Zug ausgeführt hatte, wurde dem Spieler ein Zeitraum von 15 Sekunden eingeräumt. Innerhalb dieses Zeitraums markierte der Spieler seinen Zug in Form

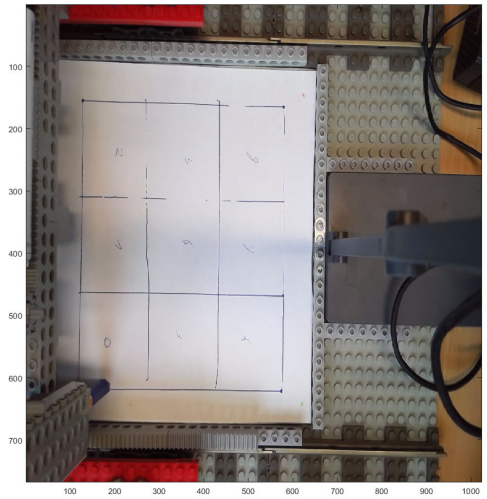


Abbildung 4. Kamerabild des Spielfelds mit aufgetragenem Spielfeldraster zur Positionsbestimmung der einzelnen Felder

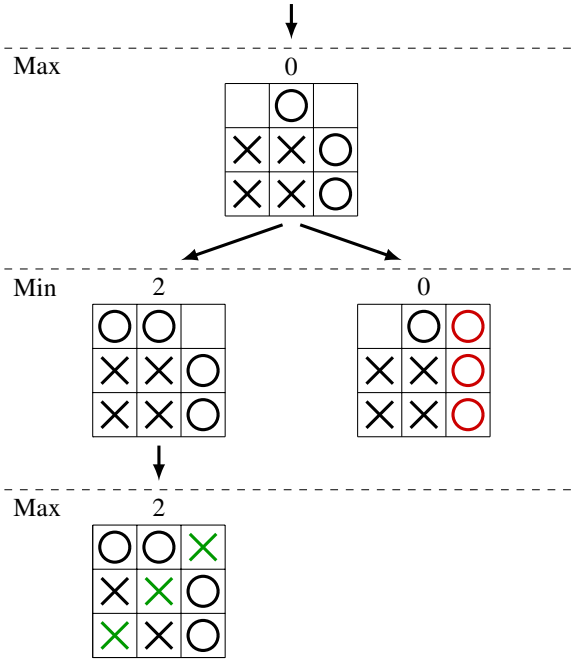


Abbildung 5. Ausschnitt aus einem Spielbaum mit Visualisierung des Minimax-Algorithmus, grün: Gewinn des Computers, rot: Gewinn des Gegenspielers

eines roten Kreises in einem der freien Felder. Nach Ablauf der vorgegebenen Zeit wurde ein Bild des Spielfelds durch die Dokumentenkamera aufgenommen und anschließend vom Computer ausgewertet. Dabei wurde nicht die Form des Kreises, sondern dessen Farbe analysiert. Ein Pixel wurde dabei als rot gewertet, sofern er den RGB-Farbcode $> 150, < 100, < 100$ aufwies. Die Voraussetzung, dass ein Zug einem Feld eindeutig zugeordnet werden konnte, beruhte auf der Messung der Anzahl von roten Pixeln innerhalb eines Feldbereichs. Wurde der Schwellenwert von 200 Pixeln in einem Feld überschritten, wurde das Feld als gültiger Spielzug gewertet.

D. Minimax- Algorithmus

Tic-Tac-Toe zählt zu den diskreten Zwei-Spieler-Spielen. Diese Klasse von Spielen lässt sich optimal durch einen Minimax-Algorithmus lösen [4].
Die Implementierung des Algorithmus orientierte sich an dem im Youtube-Video [5] diskutierten Programmierbeispiel. Dies wird im Folgenden erläutert:
Zur Bestimmung des idealen Spielzugs wurde dem Verfahren der aktuelle Status des Spielfelds in Form einer 3×3-Matrix übermittelt, woraufhin eine systematische Bewertung dieses Zustands erfolgte. Da die Komplexität des Spielbaums mit ca. $9!$ ($362\,880$) möglichen Pfaden vergleichsweise gering ausfällt, konnte eine vollständige Exploration des Zustandsraums in Echtzeit realisiert werden.
Ein Sieg des Roboters wurde dabei mit dem Wert 2, ein Unentschieden mit 1 und ein Erfolg des menschlichen Kontrahenten mit 0 definiert.
Sofern noch kein Endzustand vorlag, wurde zunächst keine unmittelbare Gewichtung vorgenommen. Stattdessen generierte der Algorithmus rekursiv sämtliche denkbaren Folgeszenarien bis zum jeweiligen Spielende, indem alle zulässigen Züge simuliert wurden. Für jeden dieser Zweige wurde der Ablauf erneut angestoßen, um den gesamten Entscheidungsbaum zu erfassen.
Die Festlegung auf den günstigsten Pfad folgte einer dualen Strategie: War der Roboter am Zug, wurde stets der höchstmögliche Ergebniswert aus den verfügbaren Optionen favorisiert, während bei den simulierten Zügen des Menschen das geringste Resultat berücksichtigt wurde. Dieses Vorgehen basierte auf der Prämisse, dass beide Parteien fehlerfrei agieren. Der Algorithmus fungierte hierbei als Werkzeug zur Lösung des Spiels, indem er bewies, dass bei beidseitig perfekter Strategie jedes Spiel zwangsläufig in einem Unentschieden (Wert 1) endet.
Dadurch ließen sich Pfade ausschließen, die zwar theoretisch zum Erfolg führen könnten, jedoch ein suboptimales Spielverhalten des Gegners voraussetzen würden.
Abbildung 5 illustriert die Funktionsweise des Minimax-Algorithmus anhand eines Spielbaums. Ausgehend vom Status Quo an der Wurzel (oben) war der menschliche Spieler am Zug. Der Algorithmus berechnete vorausschauend, dass der Gegner bei optimaler Spielweise den rechten Pfad wählen würde, um sofort zu gewinnen (rote Markierung, Wert 0). Ein Sieg des Computers war nur möglich, falls der Mensch suboptimal agierte und den linken Pfad wählte; in diesem Fall gewann der Roboter im darauffolgenden Zug (grüne Markierung auf der 3. Ebene, Wert 2). Da der Minimax-Algorithmus von einem rationalen Gegner ausging, wurde der minimale Wert (0) an den Wurzelknoten (oben) zurückgegeben.

IV. Ergebnisdiskussion

Bis zum Ende des 14-tägigen Seminars konnte der Roboter mehrere Spielabläufe zuverlässig nacheinander absolvieren. Die Bilderkennung erwies sich jedoch als ausbaufähig, da Züge des Spielers am Rand eines Feldes gelegentlich nicht zuverlässig erkannt wurden. Der Minimax-Algorithmus funktionierte hingegen einwandfrei. Ein wiederkehrendes Problem stellten die auftretenden Reibungskräfte dar: Die Reibung zwischen Stift und Papier führte vereinzelt dazu, dass gezeichnete Linien einen Knick aufwiesen oder unvollständig blieben.

V. Zusammenfassung und Fazit

Im Rahmen des Projekts ist es gelungen, einen zuverlässig funktionierenden Tic-Tac-Toe-Roboter in Form eines Plotters zu konstruieren. Dabei wurden Reibung und Balance des Roboters sukzessive optimiert. Im Projektverlauf gelang der Übergang von einer manuellen Tastatureingabe und zufallsbasierten Spielzügen hin zu einer automatisierten Bilderkennung sowie einer strategischen Spielweise mittels Minimax-Algorithmus.

Trotz der Optimierung boten Reibung und Balance weiteren Verbesserungsbedarf, um zukünftig auf die Gewichte in Form von Muttern verzichten zu können. Auch bei der Software gibt es Möglichkeiten zur Weiterentwicklung. Zudem könnten die Spielstärke des Roboters sowie der Startspieler konfigurierbar gestaltet werden.

Literaturverzeichnis

[1] Schulz, Karsten: *Der Tic-Tac-Toe-Robo*. https://journals.ub.ovgu.de/index.php/LEGO/article/view/2262/2242. Version: Februar 2026

[2] The MathWorks, Inc.: *Image Acquisition Toolbox — Functions*. https://de.mathworks.com/help/releases/R2025b/imaq/referencelist.html?type=function&s_tid=CRUX_topnav. Version: Februar 2026

[3] The MathWorks, Inc.: *Build Interactive Tools — Functions*. https://de.mathworks.com/help/images/referencelist.html?type=function&s_tid=CRUX_topnav&category=building-guis-with-modular-interactive-tools. Version: Februar 2026

[4] Nopper, Tobias: *Prinzip MinMax-Algorithmus am Beispiel von Tic Tac Toe | Wie spielen Computer?* https://youtu.be/3ufcmCpKb6w. Version: Februar 2026

[5] Nopper, Tobias: *Implementierung des MinMax-Algorithmus für Tic Tac Toe | Wie spielen Computer?* https://youtu.be/ODgPsXssUvk. Version: Februar 2026

Kehrbert – Wenn niemand kehrt, kehrt Kehrbert

Jannis Prüfer, Elektro- und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des LEGO-Praktikums wurde der selbstständige Kehrroboter „Kehrbert“ entwickelt. Ziel des Projekts war es, ein System zu entwerfen, das eigenständig Legosteine vom Boden einsammelt. Der Roboter basiert auf dem LEGO EV3 und wird mithilfe von MATLAB programmiert. Er bewegt sich selbstständig durch einen Raum, erkennt Hindernisse mittels Ultraschallsensor und transportiert eingesammelte Objekte über eine Bürste und ein Förderband in einen Sammelbehälter. Trotz technischer Herausforderungen konnte ein funktionsfähiger Roboter realisiert werden. Die Arbeit zeigt exemplarisch, wie mit begrenzten Mitteln ein autonom agierendes System entwickelt werden kann und welche praktischen Herausforderungen bei der Umsetzung auftreten.**

***Schlagwörter*—Autonomer Roboter, EV3, Hinderniserkennung, MATLAB, Reaktive Steuerung, Robotik**

I. Einleitung

LEGOSTEINE stellen häufig unterschätzte Gefahrenquellen im Haushalt dar. Ein einzelner Stein auf dem Boden kann beim Darauftreten starke Schmerzen verursachen. Um diesem alltäglichen Problem technisch zu begegnen, wurde im Rahmen des Projekts der Kehrroboter „Kehrbert“ entwickelt. Ziel war es, ein System zu realisieren, das Legosteine selbstständig aufnimmt und in einem Behälter sammelt, ohne dass manuell eingegriffen werden muss.

Der Schwerpunkt lag dabei auf dem Zusammenspiel von mechanischer Konstruktion, Sensorik und softwarebasierter Steuerung. Darüber hinaus sollte untersucht werden, inwieweit sich mit einfachen Mitteln ein autonomes Verhalten realisieren lässt und welche Grenzen sich dabei ergeben. Insbesondere stand die Frage im Mittelpunkt, wie effizient ein solches System ohne komplexe Navigationsverfahren arbeiten kann.

Autonome Robotersysteme gewinnen zunehmend an Bedeutung, sowohl im industriellen Umfeld als auch im privaten Bereich. Bereits einfache Systeme können repetitive Aufgaben übernehmen und so den Menschen entlasten. Das vorliegende Projekt ordnet sich in diesen Kontext ein und dient als praktisches Beispiel für die Umsetzung eines autonomen Systems mit begrenzten Ressourcen.

II. Konzept und Realisierung

Der Roboter wurde auf Grundlage des batteriebetriebenen LEGO EV3 aufgebaut und besteht aus vier Motoren. Zwei Motoren treiben jeweils die linke und rechte Kette an und ermöglichen Vorwärts-, Rückwärts- sowie Drehbewegungen durch unterschiedliche Ansteuerung. Durch diese Differentialallenkung ist es dem System möglich, sich auf der Stelle zu drehen und flexibel auf Hindernisse zu reagieren.

Zusätzlich wird eine Frontbürste durch einen eigenen Motor betrieben. Diese befördert die Legosteine auf ein Förderband, welches die Steine anschließend in einen herausnehmbaren Sammelbehälter transportiert. Die Drehgeschwindigkeit der Bürste musste so gewählt werden, dass die Steine zuverlässig erfasst werden, ohne seitlich weggeschleudert zu werden. Durch mehrere Tests wurde ein praktikabler Kompromiss gefunden, der sowohl eine ausreichende Aufnahmeleistung als auch eine stabile Funktion gewährleistet.

Für die Steuerung und Hinderniserkennung kommen zwei Sensoren zum Einsatz. Ein Ultraschallsensor misst kontinuierlich den Abstand zu Hindernissen vor dem Roboter. Unterschreitet der Abstand einen bestimmten Grenzwert, wird automatisch ein Drehmanöver eingeleitet. Ein zusätzlicher

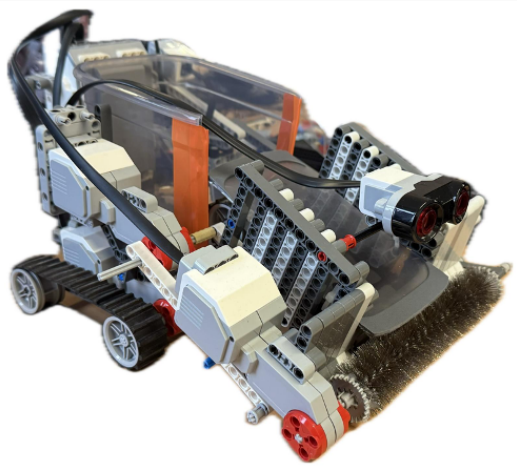


Abbildung 1. Gesamtansicht des Kehrroboters Kehrbert

Abbildung 2. Draufsicht auf den aufgebauten Kehrroboter

DOI: 10.24352/UB.OVGU-2026-044

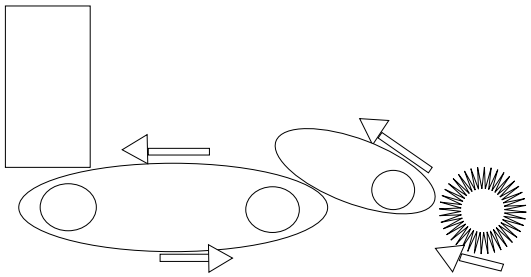


Abbildung 3. Schematische Darstellung der Bürste und des Fördermechanismus

Tastsensor dient als Start- und Stop-Schalter zur Aktivierung beziehungsweise Deaktivierung des Systems.

Die Programmierung erfolgte in MATLAB unter Verwendung der EV3-Bibliothek. Die Steuerung basiert auf einem reaktiven Prinzip. Das bedeutet, dass keine Positionsbestimmung oder Kartierung der Umgebung erfolgt. Stattdessen reagiert der Roboter ausschließlich auf aktuelle Sensordaten. Die Logik folgt dem einfachen Prinzip: Fahren, Messen, Reagieren.

Diese reduzierte Struktur erhöht die Stabilität und vereinfacht die Implementierung erheblich. Gleichzeitig führt sie jedoch dazu, dass keine optimierte Flächenabdeckung erreicht wird und der Roboter bereits gereinigte Bereiche erneut befährt. Der grundsätzliche Programmablaufplan ist in Abbildung dargestellt.

III. Vergleich mit bestehenden Arbeiten

Ein ähnliches Projekt wurde bereits im Rahmen eines früheren LEGO-Praktikums durchgeführt [1]. Beide Systeme basieren auf einem reaktiven Steuerungsansatz und weisen somit grundlegende Gemeinsamkeiten auf.

Der Verzicht auf eine Kartierung der Umgebung führt zu einer einfachen Umsetzung, bringt jedoch Einschränkungen hinsichtlich der Effizienz mit sich. Unterschiede zeigen sich vor allem in der mechanischen Umsetzung. Während im Vergleichsprojekt ein einfaches Kehrsystem verwendet wurde, verfügt Kehrbert über ein Förderband, welches die Effizienz erhöht.

Insgesamt kann Kehrbert als Weiterentwicklung betrachtet werden, wobei die grundlegenden Einschränkungen reaktiver Systeme bestehen bleiben.

IV. Ergebnisdiskussion

Während der Entwicklung traten mehrere Herausforderungen auf, die sowohl mechanischer als auch softwaretechnischer Natur waren. Mechanisch wurden Legosteine teilweise seitlich aus dem System geschleudert. Auch beim Ultraschallsensor kam es zu schwankenden Messwerten.

Ein weiterer Aspekt ist die begrenzte Effizienz der reaktiven Steuerung. Der Roboter bewegt sich nicht zielgerichtet, sondern folgt einem zufälligen Bewegungsmuster. Dies führt zu einer ineffizienten Flächenabdeckung.

Darüber hinaus zeigt sich, dass die Systemarchitektur zwar robust, jedoch nur begrenzt skalierbar ist. Eine Erweiterung um Navigationsverfahren könnte hier deutliche Verbesserungen bringen.

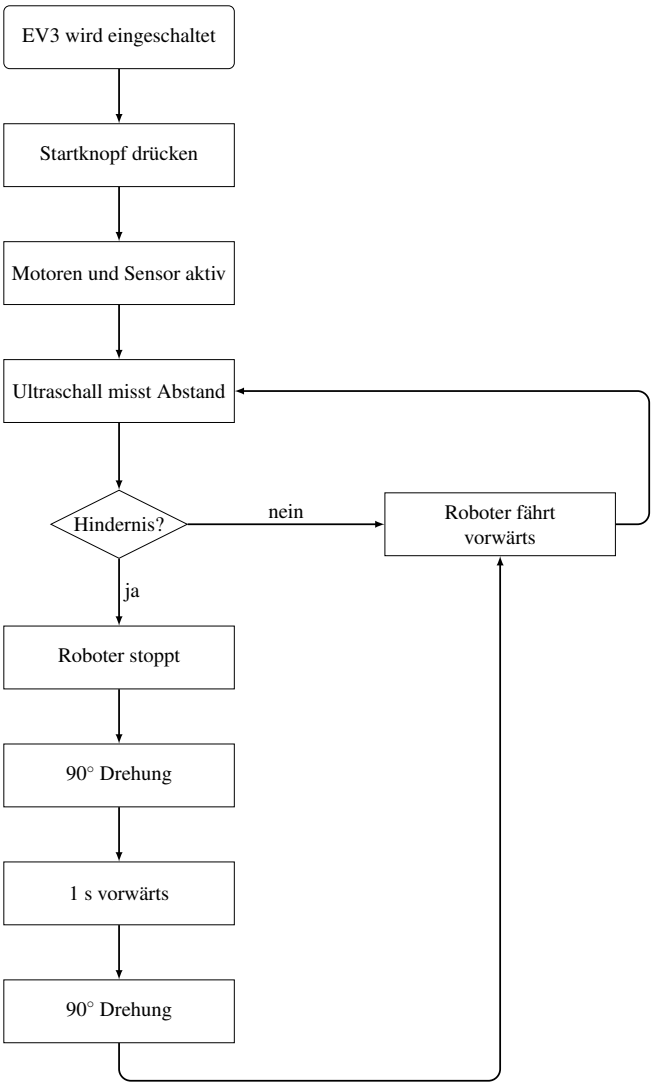


Abbildung 4. Programmablaufplan des Kehrroboters

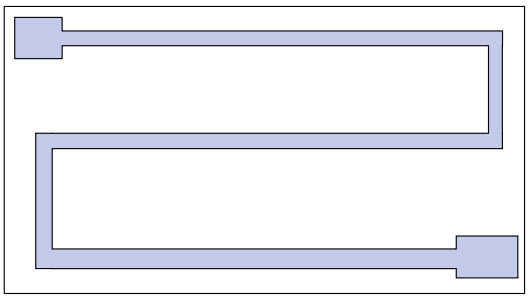


Abbildung 5. Fahrplan im Raum von Kehrbert

Die grundlegenden Anforderungen konnten dennoch erfüllt werden.

V. Ausblick und weiterführende Überlegungen

Die im Rahmen dieses Projekts gewonnenen Erkenntnisse bieten zahlreiche Ansatzpunkte für weiterführende Untersuchungen und mögliche Optimierungen. Obwohl das entwickelte

System bereits grundlegende Funktionen eines autonomen Kehrroboters erfüllt, zeigt sich bei näherer Betrachtung, dass insbesondere im Hinblick auf Effizienz, Robustheit und Skalierbarkeit weiteres Potenzial besteht.

Ein zentraler Aspekt zukünftiger Arbeiten könnte in der Erweiterung der Steuerungslogik liegen. Während im vorliegenden Projekt bewusst auf komplexe Navigationsverfahren verzichtet wurde, eröffnet der Einsatz solcher Methoden neue Möglichkeiten zur Verbesserung der Flächenabdeckung.

Darüber hinaus stellt auch die Sensorik einen wichtigen Ansatzpunkt für zukünftige Optimierungen dar. Die verwendeten Sensoren erfüllen zwar die grundlegenden Anforderungen, zeigen jedoch Schwächen in Bezug auf Messgenauigkeit und Reaktionsverhalten.

Neben softwareseitigen Anpassungen spielen auch mechanische Weiterentwicklungen eine entscheidende Rolle. Die Konstruktion des Aufnahme- und Transportsystems hat sich grundsätzlich als funktionsfähig erwiesen, lässt jedoch Raum für Verbesserungen.

Ein weiterer Gesichtspunkt betrifft die Übertragbarkeit des Systems auf andere Anwendungsbereiche. Die zugrunde liegenden Prinzipien lassen sich grundsätzlich auch auf andere Szenarien übertragen.

VI. Zusammenfassung und Fazit

Im Rahmen des Projekts wurde ein funktionsfähiger Kehrroboter konstruiert und programmiert. Das Zusammenspiel von Mechanik, Sensorik und Software stellte eine zentrale Herausforderung dar.

Das Projekt verdeutlicht, dass auch mit einfachen Mitteln autonome Systeme realisiert werden können. Gleichzeitig zeigt es die Grenzen reaktiver Steuerungen auf.

Kehrbert stellt somit ein gelungenes Beispiel dar und bietet eine Grundlage für weiterführende Entwicklungen im Bereich autonomer Robotik.

Literatur

[1] T. Schreiber, „Anfertigung eines Kehrroboters mithilfe von Lego Mindstorms und MatLab“, 2018.

Automatischer Reinigungsroboter

Vincent Weiß, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen dieser Arbeit wurde auf Basis der LEGO®-Mindstorms®-EV3-Plattform ein autonomer Kehrroboter mit dem Namen „Kehrbert“ entwickelt. Ziel war es, einen funktionsfähigen Prototyp zu erschaffen, der frei im Raum liegende LEGO-Steine selbstständig aufnehmen und Hindernissen erkennen kann.**

Die Arbeit beschreibt zunächst bestehende studentische Umsetzungen ähnlicher Systeme und grenzt das eigene Konzept davon ab. Anschließend werden die mechanische Konstruktion, der Aufbau des Aufnahme- und Fördermechanismus sowie die verwendete Sensorik erläutert. Ein weiterer Schwerpunkt liegt auf der Programmierung in MATLAB und der strukturierten Umsetzung der Navigationslogik. Abschließend werden die praktischen Ergebnisse bewertet und bestehende Einschränkungen sowie mögliche Verbesserungen diskutiert.

***Schlagwörter*—Schlagwörter—Aufsammelmechanismus, Kehrroboter, EV3, Förderband, Navigation, Ultraschallsensor**

I. Einleitung

AUTONOME Reinigungsroboter sind in Haushalten, Industrie und Logistik verbreitet und übernehmen Aufgaben wie systematisches Abfahren von Flächen, Hinderniserkennung und gezieltes Aufnehmen von Objekten. Die Entwicklung eines funktionierenden autonomen Roboters ist technisch anspruchsvoll, da Mechanik, Sensorik und Software eng zusammenarbeiten müssen.

Ziel des Projektseminars an der Otto-von-Guericke-Universität Magdeburg war es, mit der LEGO®-Mindstorms®-EV3-Plattform einen autonomen Kehrroboter zu entwickeln, der lose LEGO-Steine selbstständig aufnimmt und sie dann in einen Behälter befördert.

Um das Projekt innerhalb des Zeitrahmens realisieren zu können, wurden folgende Vereinfachungen festgelegt:

1) Keine Raumkartierung: der Roboter arbeitet ohne interne Karte
2) Aufnahme beschränkt auf Steine bestimmter Größe
3) Programmierung mit MATLAB und der RWTH-Mindstorms-EV3-Toolbox

Die Arbeit dokumentiert die mechanische Konstruktion, das Aufnahmesystem und die Softwareentwicklung und zeigt exemplarisch, wie ein funktionaler autonomer Kehrroboter im Kleinstmaßstab umgesetzt werden kann.

II. Vorbetrachtungen

Die Idee, autonome Reinigungsroboter zu entwickeln, ist seit Jahren Gegenstand technischer Weiterentwicklungen. Besonders im Haushaltsbereich existieren zahlreiche kommerzielle Lösungen, die selbstständig Räume abfahren und Verschmutzungen aufnehmen. Ziel dieses Projekts war es, ein vereinfachtes, funktionales System auf Basis der LEGO®-Mindstorms®-EV3-Plattform zu realisieren.

DOI: 10.24352/UB.OVGU-2026-045

Dabei wurde bewusst darauf verzichtet, ein bestehendes System lediglich nachzubauen. Stattdessen lag der Fokus darauf, mit den gegebenen Mitteln eine eigenständige mechanische Lösung zum Aufnehmen von LEGO-Steinen [1] zu entwickeln (siehe Abbildung 1) und diese mit einer einfachen, aber funktionalen Navigation zu kombinieren.

Abbildung 1. LEGO®-Bausteine – Ausgangsmaterial für „Kehrbert“ [1]

A. Bereits existierende Umsetzungen

Im Rahmen des Projektseminars Elektrotechnik/Informationstechnik an der Otto-von-Guericke-Universität Magdeburg wurden in den vergangenen Jahren mehrere autonome Reinigungsroboter auf Basis von LEGO® Mindstorms entwickelt.

2018 entstand der sogenannte „Cleaning Bot“ [2], ein Kettenfahrzeug mit drei Motoren: zwei für den Fahrantrieb und einer für einen rotierenden Staubwedel, der die LEGO-Teile auf ein angebrachtes Kehrblech beförderte. Die Navigation erfolgte über einen Ultraschallsensor; bei Unterschreiten eines Abstandswertes drehte der Roboter und setzte seine Fahrt in neuer Richtung fort.

2021 wurde ein ähnlicher Ansatz verfolgt [3]. Auch hier kam ein Kettenantrieb mit Ultraschallsensor zum Einsatz, die Reinigung erfolgte jedoch über ein Wischsystem mit Staubtüchern. Das Bewegungsmuster basierte auf Abstandsmessung und anschließenden 90°-Drehungen, um eine bahnähnliche Fahrt zu erzeugen.

Beide Projekte zeigen, dass mit LEGO® Mindstorms grundlegende autonome Reinigungssysteme realisierbar sind. „Kehrbert“ knüpft daran an, setzt jedoch auf einen richtigen Aufnahme- und Fördermechanismus mit Auffangbehälter.

B. Zusätzliche Herausforderungen

Im Vergleich zu kommerziellen Reinigungssystemen sind die technischen Möglichkeiten der LEGO®-Mindstorms®-EV3-Plattform deutlich eingeschränkt. Es stehen nur wenige Sensoren zur Verfügung, und auch die Rechenleistung des EV3-Steins ist begrenzt. Funktionen wie eine präzise Raumkartierung, eine kontinuierliche Positionsbestimmung oder komplexe Navigationsalgorithmen ließen sich daher im zeitlichen Rahmen dieses Projekts nicht umsetzen. Die Navigation musste bewusst einfach gehalten werden und ausschließlich auf direkten Sensorwerten basieren.

Eine weitere Herausforderung ergab sich aus der mechanischen Umsetzung des Kehrmechanismus. Die selbst konstruierte Bürste sollte LEGO-Steine zuverlässig erfassen und gezielt auf das Förderband weiterleiten, das die Teile anschließend in den Auffangbehälter transportiert. In der Praxis zeigte sich jedoch, dass kleine Ungenauigkeiten im Aufbau schnell Auswirkungen auf die Funktion hatten. Bereits geringes Spiel in den Achsen oder Verbindungen konnte dazu führen, dass Steine verklemmten oder nicht sauber weitertransportiert wurden. Auch die Abstimmung zwischen Bürstendrehzahl und Förderbandgeschwindigkeit erforderte mehrere Anpassungen, um einen möglichst gleichmäßigen Ablauf zu erreichen.

Zusätzlich stellte der kettenbasierte Fahrantrieb besondere Anforderungen. Obwohl Ketten eine gute Traktion bieten, ist ein exakter Geradeauslauf schwer zu realisieren. Das musste bei der Programmierung berücksichtigt werden. Darüber hinaus war eine saubere Koordination mehrerer Motoren notwendig, damit Fahrbewegung, Bürstenrotation und Förderbandbetrieb zuverlässig und synchron zusammenarbeiten. Erst durch wiederholtes Testen und Anpassen konnte ein stabiler und reproduzierbarer Betriebszustand erreicht werden

III. Umsetzung

A. Mechanische Basis

Der Kehrroboter fährt auf einem kettengetriebenen Fahrwerk, das von zwei Motoren unabhängig gesteuert wird. So kann er geradeaus fahren und sich auf der Stelle drehen. Die Ketten sorgen für gute Traktion, erschweren aber exaktes Geradeausfahren, da schnell kleine Richtungsabweichungen entstehen.

Das Fahrwerk liegt hinten, Bürste und Förderband vorne. Der EV3-Stein sitzt hinten, wodurch das Gewicht gleichmäßig verteilt und die Stabilität verbessert wird. Der Ultraschallsensor ist vorne für frühzeitige Hinderniserkennung angebracht, der Taster weiter oben für einfachen Zugriff.

Das Grundgerüst ist stabil und modular, sodass Bürste, Förderband, Auffangbehälter, Sensoren und Brick zuverlässig getragen werden und bei Bedarf leicht angepasst oder ausgetauscht werden können.

B. Aufsammelmechanismus

Der vordere Teil des Roboters besteht aus einer rotierenden Bürste (siehe Abbildung 2), die von einem eigenen Motor angetrieben wird. Sie erfasst die LEGO-Steine zuverlässig und bewegt sie zum Förderband, das ebenfalls von einem eigenen Motor betrieben wird und die Steine in den Auffangbehälter transportiert. Bürste und Förderband arbeiten parallel, sodass eine zuverlässige Aufnahme und Weiterleitung gewährleistet sind.

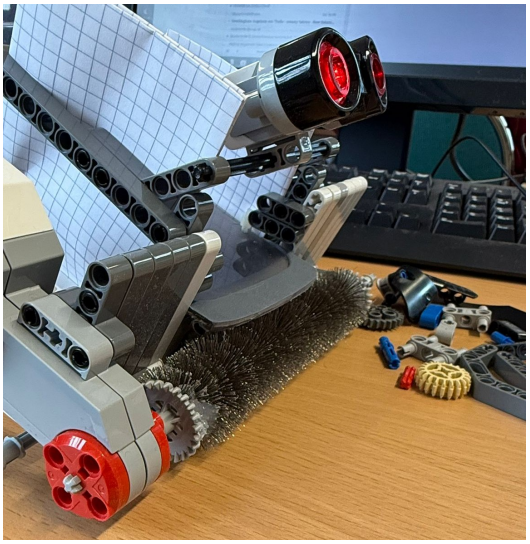


Abbildung 2. Detailansicht der Bürstenkonstruktion von „Kehrbert“

Das Förderband besitzt seitliche Wände und einen Deckel, um ein Herausfallen der Steine zu verhindern. Größe von Bürste und Förderband ist auf einen bestimmten LEGO-Bereich abgestimmt; sehr kleine oder große Teile werden ggf. nicht aufgenommen.

Der Auffangbehälter am Ende des Förderbands ist herausnehmbar, sodass die LEGO-Steine nach dem Einsammeln einfach entleert werden können. Die gesamte Einheit ist modular gestaltet, sodass Wartung, Austausch oder Anpassung von Bürste, Förderband oder Behälter unkompliziert möglich ist.

C. Sensorik

Zur Hinderniserkennung ist der Roboter mit einem Ultraschallsensor (siehe Abbildung 3) ausgestattet, der kontinuierlich den Abstand zu Wänden oder anderen Objekten vor dem Roboter misst. Die gemessenen Werte werden in regelmäßigen Intervallen innerhalb der Hauptprogrammschleife abgefragt. Liegt der Abstand unter einem definierten Grenzwert, wird ein Hindernis erkannt, wodurch die Navigationsroutine ausgelöst wird, um eine Kollision zu vermeiden.

Abbildung 3. Ultraschallsensor des Roboters [4]

Zusätzlich ist ein Tastsensor (siehe Abbildung 4) als Ein- und Ausschalter des Systems integriert. Durch Betätigung

des Sensors wird der Betriebszustand des Roboters softwareseitig umgeschaltet: Wird der Taster gedrückt, wechselt der Roboter zwischen aktiviertem und deaktiviertem Zustand. Eine Zustandsvariable speichert dabei den aktuellen Modus. Die Software erkennt nur Tasterflanken, d. h. die Umschaltung erfolgt nur beim Übergang von nicht gedrückt auf gedrückt, um Mehrfachauslösungen durch längeres Drücken zu vermeiden.

Abbildung 4. LEGO®EV3-Tastsensor – zur Bedienung von „Kehrbert“ [5]

Durch diese Kombination von Ultraschallsensor und Tastsensor kann der Roboter autonom auf seine Umgebung reagieren und gleichzeitig vom Benutzer einfach gestartet oder gestoppt werden. Die regelmäßige Abfrage der Sensoren erfolgt in einem kurzen Intervall von $20\,\mathrm{ms}$, um eine schnelle Reaktion auf Hindernisse zu gewährleisten. Gleichzeitig verhindert die Software, dass wiederholte Auslösungen während eines Drehvorgangs auftreten, indem eine Sperrvariable implementiert wurde, die erst zurückgesetzt wird, wenn der Abstand wieder über einem definierten Schwellenwert liegt.

D. Navigationsstrategie

Der Kehrroboter fährt grundsätzlich geradeaus, solange der Ultraschallsensor keinen Abstand unter einem definierten Grenzwert misst. Sobald ein Hindernis erkannt wird, startet die Hindernisvermeidung:

1) Die Fahrmotoren stoppen kurz, um Kollisionen zu vermeiden
2) Abhängig von einer internen Variablen (`turnLeftNext`) dreht der Roboter 90° nach links oder rechts. Nach jeder Drehung wechselt die Richtung, sodass die Fläche systematisch abgefahren wird
3) Danach fährt der Roboter ein kurzes Stück geradeaus und dreht ein zweites Mal in die gleiche Richtung. So entsteht ein einfaches, bahnähnliches Muster, (siehe Abbildung 5)
4) Anschließend setzen die Fahrmotoren die Geradeausfahrt fort

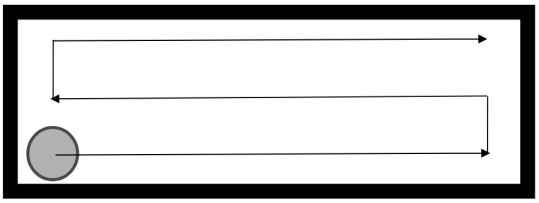


Abbildung 5. Schematische Darstellung der Navigationsstrategie von „Kehrbert“

Eine Sperrvariable (`avoidLock`) sorgt dafür, dass die Hindernisroutine während der Drehungen nicht mehrfach ausgelöst wird. Erst wenn der Abstand wieder über dem Schwellenwert liegt, wird die Sperre aufgehoben.

Das System ist simpel, funktioniert aber zuverlässig: der Roboter umgeht Hindernisse ohne komplexe Kartierung. Parameter wie Drehgeschwindigkeit, Drehzeit oder Abstandsschwelle lassen sich anpassen, um verschiedene Bodenflächen oder LEGO-Steine optimal zu bearbeiten.

E. Softwarestruktur in MATLAB

Der Kehrroboter wurde in MATLAB mit der RWTH-Mindstorms-EV3-Toolbox programmiert. Zunächst wird über USB eine Verbindung zum EV3-Stein aufgebaut, und alle Motoren (A–D) sowie Sensoren (Ultraschall- und Tastsensor) werden initialisiert.

Die Steuerung läuft in einer Endlosschleife, die alle $20\,\mathrm{ms}$ die Sensorwerte abfragt. Der Tastsensor dient als Ein-/Ausschalter und wird nur beim Übergang von nicht gedrückt auf gedrückt ausgewertet, sodass ein versehentliches wiederholtes Umschalten verhindert wird.

Die Fahrmotoren (A und D) und die Motoren für Bürste (B) und Förderband (C) werden koordiniert:

- Im aktivierten Zustand fahren die Motoren synchron, sodass der Roboter geradeaus fährt und gleichzeitig LEGO-Steine aufnimmt und transportiert.
- Wenn der Ultraschallsensor ein Hindernis näher als den Grenzwert erkennt, stoppen die Fahrmotoren kurz und der Roboter dreht sich 90° nach links oder rechts, abhängig von der Variablen `turnLeftNext`, die nach jeder Drehung umgeschaltet wird.
- Nach der Drehung fahren die Motoren wieder vorwärts. Die Sperrvariable `avoidLock` sorgt dafür, dass während der Drehbewegung keine erneute Hindernisreaktion ausgelöst wird.

Bürste und Förderband werden jeweils von einem eigenen Motor angetrieben und arbeiten parallel zur Fahrt, sodass LEGO-Steine kontinuierlich in den Auffangbehälter befördert werden. Beim Ausschalten stoppen alle Motoren sofort, was die Handhabung sicher macht.

Die klare Trennung von Sensorabfrage, Motorsteuerung und Drehlogik macht den Code übersichtlich und erlaubt einfache Anpassungen von Parametern wie Geschwindigkeit, Abstandsschwelle oder Drehzeit, ohne die gesamte Software ändern zu müssen.

F. Programmlogik

Die Steuerung von „Kehrbert“ läuft in einer fortlaufenden Hauptschleife (siehe Abbildung 6). Zunächst wird die Verbindung zum EV3-Stein aufgebaut und zentrale Parameter wie Geschwindigkeit, Drehzeit und Abstand zum Hindernis festgelegt. Der Tastsensor dient als Ein-/Ausschalter, wobei ein Zustandswechsel nur beim Übergang von „nicht gedrückt“ auf „gedrückt“ erfolgt. Im aktiven Zustand fahren die Antriebsmotoren synchron geradeaus, während Bürste und Förderband LEGO-Steine aufnehmen. Erkennt der Ultraschallsensor ein

Hindernis, wird ein Ausweichmanöver durchgeführt; andernfalls fährt der Roboter geradeaus weiter. Nach Abschluss des Manövers setzt er die Geradeausfahrt fort, wobei die Sperrvariable `avoidLock` Mehrfachreaktionen während Drehungen verhindert. Durch diese klare Trennung von Sensorabfrage, Motorsteuerung und Ausweichlogik lassen sich Parameter wie Geschwindigkeit oder Abstandsschwelle einfach anpassen.

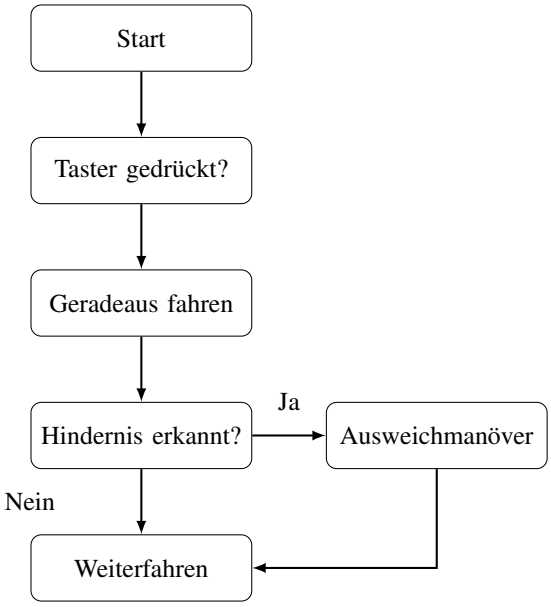


Abbildung 6. Ablaufdiagramm der Programmlogik von „Kehrbert“

IV. Ergebnisdiskussion

Der entwickelte Kehrroboter „Kehrbert“ (siehe Abbildung 7) erfüllt die zentralen Zielsetzungen des Projekts. Er sammelt lose LEGO-Steine zuverlässig auf, transportiert sie über das Förderband in den Auffangbehälter und reagiert wenn er sich einer Wand nähert mit einer Drehung. Durch den parallelen Betrieb von Bürste und Förderband wird ein kontinuierlicher Aufnahmeprozess gewährleistet, sodass der Roboter auch bei längeren Fahrtstrecken effizient arbeitet. Die einfache Navigationsstrategie erlaubt ein systematisches Abfahren der Fläche, selbst ohne interne Kartierung.

Im praktischen Einsatz zeigten sich jedoch einige Einschränkungen. Das kettengetriebene Fahrwerk verursacht auf glatten Böden leichte Richtungsabweichungen, was gelegentlich zu Korrekturfahrten führt. Die Aufnahme funktioniert zuverlässig nur innerhalb eines bestimmten LEGO-Steinegrößenbereichs. Sehr kleine oder ungewöhnlich geformte Teile werden teilweise nicht erfasst. Der Ultraschallsensor erkennt größere Hindernisse zuverlässig, stößt jedoch bei schmalen Objekten oder komplexen Ecken an seine Grenzen. Zudem führt das Fehlen einer Raumkartierung dazu, dass bereits gereinigte Bereiche mehrfach befahren werden, was die Effizienz etwas mindert.

Trotz dieser Einschränkungen zeigt das Projekt, dass die LEGO®-Mindstorms®-EV3-Plattform eine geeignete Basis für die prototypische Umsetzung autonomer Reinigungsroboter bietet. Das Zusammenspiel von mechanischer Konstruktion, Sensorik und einfacher Softwarelogik funktioniert insgesamt zuverlässig. Die gewonnenen Erfahrungen geben zudem wertvolle Einblicke in die Herausforderungen kleiner autonomer Systeme und bieten eine solide Grundlage für Optimierungen in zukünftigen Projekten.

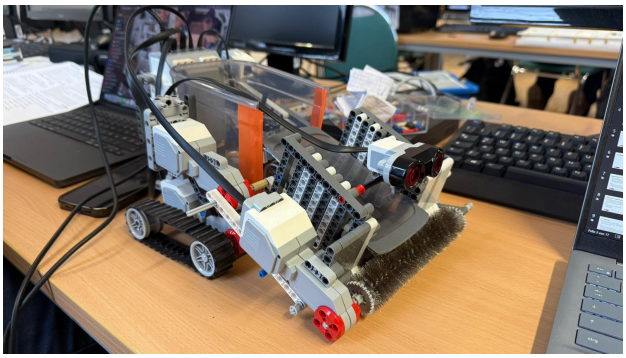


Abbildung 7. Kehrbert – Finaler Aufbau

V. Zusammenfassung und Fazit

Ziel dieser Arbeit war die Entwicklung eines autonomen Kehrroboters auf Basis der LEGO®-Mindstorms®-EV3-Plattform, der frei liegende LEGO-Steine selbstständig aufnehmen kann. Dabei stand nicht die Nachbildung eines bestehenden Systems im Vordergrund, sondern die eigenständige Konstruktion eines funktionalen Prototyps unter realistischen technischen Einschränkungen.

Im Verlauf des Projekts wurden sowohl mechanische als auch softwareseitige Lösungen erarbeitet. Dazu zählten der Aufbau eines stabilen Fahrwerks, die Entwicklung eines kombinierten Bürsten- und Fördermechanismus sowie die Umsetzung einer einfachen, sensorbasierten Navigationsstrategie in MATLAB. Die praktische Umsetzung machte deutlich, wie eng mechanische Konstruktion, Sensorik und Programmierung miteinander verknüpft sind und wie stark sich kleine konstruktive Änderungen auf das Gesamtverhalten des Systems auswirken.

Das Projekt zeigt, dass auch mit begrenzten Mitteln ein funktionsfähiges autonomes System realisiert werden kann. Gleichzeitig wurde deutlich, wo technische Grenzen der gewählten Plattform liegen. Die Arbeit bietet damit eine solide Grundlage für weiterführende Verbesserungen, etwa im Bereich der Bewegungsgenauigkeit oder der Erweiterung der Sensorik, und liefert wertvolle praktische Erfahrungen im Umgang mit autonomen Robotersystemen.

Literaturverzeichnis

[1] *LEGO®-Bausteine*. https://www.ebay.de/itm/352612246803?srsltid=AfmBOopByW_0FMhSsmYfmLKGOGRpO5ljuqZT85BryHFG3MqT4C10DWzd, 2026. – Abgerufen am 26.02.2026

[2] Schwank, Malte: *Bau eines Cleaning Bot – Bericht zum LEGO Mindstorms Seminar*. 2018. – Projektseminar Elektrotechnik/Informationstechnik https://doi.org/10.24352/UB.OVGU-2018-045

[3] Bursian, Kevin: *Reinigungsroboter im Modellversuch*. 2021. – Projektseminar Elektrotechnik/Informationstechnik https://doi.org/10.24352/UB.OVGU-2021-042

[4] LEGO Education: *LEGO MINDSTORMS Education Ultraschallsensor 45504*. https://www.amazon.de/LEGO-MINDSTORMS-Education-Ultraschallsensor-45504/dp/B00E1PTRAE. Version: o. J.. – Produktseite auf Amazon

[5] *LEGO®EV3Tastsensor(45507)*. https://www.brickwatch.net/de-DE/set/45507/EV3-Touch-Sensor.html, . – Zugriff 2026

Zeichenroboter

Vincent Gratz, Elektro- und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des LEGO-Mindstorms-Projektseminars 2026 wurde ein Zeichenroboter entwickelt, der selbstständig vorgegebene Formen oder Muster auf Papier zeichnen kann. In diesem Paper liegt der Fokus auf der Konstruktion des Roboters aus LEGO-Technic-Bauteilen sowie der Funktionsweise der Programmierung in MATLAB. Anschließend werden die erzielten Ergebnisse ausgewertet und Verbesserungsansätze ausformuliert.**

***Schlagwörter*—Automatisierung, Klemmbausteine, LEGO-Roboter, Zeichenroboter, Zeichnen.**

I. Einleitung

IN unserer heutigen Welt gehört es zur Normalität, dass täglich neue Robotersysteme, zur Erfüllung von Aufgaben sowie zur Lösung von Problemen, gebaut werden.

Im Rahmen des zweiwöchigen LEGO-Seminars bestand die Möglichkeit, selbstständig einen solchen Roboter zu konzipieren, zu konstruieren und zu programmieren. Dabei wurde das Ziel verfolgt einen Roboter zu entwickeln, welcher schreiben beziehungsweise zeichnen kann. Der entwickelte Zeichenroboter soll insbesondere der Unterstützung von motorisch eingeschränkten Personen dienen.

II. Vorbetrachtungen

In diesem Abschnitt werden die ursprüngliche Konzeptidee sowie die essenziellen Bauteile für die Realisierung des Zeichenroboters dargestellt.

A. Ursprungsidee

Die erste Konzeption sah die Entwicklung eines Roboters vor, welcher in der Lage ist Druckschrift zu schreiben. Zu diesem Zweck wurde eine erste Entwurfsskizze angefertigt, siehe Abbildung 1 . Vorgesehen war dabei ein frei beweglicher Roboterarm, der sich im dreidimensionalen Raum bewegen kann, um so entsprechende Schreibbewegungen auszuführen.

Im Verlauf der Planungsphase zeigten sich jedoch zwei Probleme. Erstens erwies sich die Konstruktion eines frei beweglichen Roboterarms unter Verwendung von Klemmbausteinen als deutlich komplexer als ursprünglich angenommen. Zweitens stellte die inverse Kinematik im dreidimensionalen Raum eine erhebliche Schwierigkeit dar, die den vorgesehenen zeitlichen Rahmen des Projekts überschritten hätte. Aus diesen Gründen wurde das ursprüngliche Konzept eines frei beweglichen Roboterarms vereinfacht und an die gegebenen Rahmenbedingungen angepasst.

DOI: 10.24352/UB.OVGU-2026-046

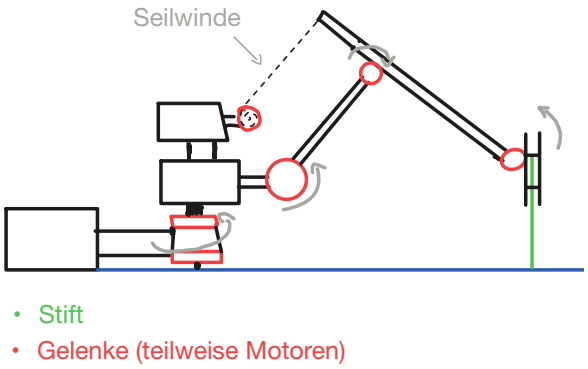


Abbildung 1. Erste Skizze des Zeichenroboters

B. Inspiration

Der zweite Konzeptionsansatz des Zeichenroboters basiert auf dem Konferenzpapier von Bryan Vásquez-Pineda und Manuel Cardona [1]. Dieser Lösungsansatz verwendet einen 5R-Parallellink-Mechanismus. Dieses Prinzip diente als Inspiration für die Weiterentwicklung des Zeichenroboters. Eine detaillierte Beschreibung des Aufbaus des Zeichenroboters erfolgt in Abschnitt III-A1 des Papers.

C. Bausteine

Für dieses Projekt wurden drei EV3-Motoren und ein Tastsensor verwendet, welche über den EV3-Stein miteinander verbunden sind. Außerdem wurden zahlreiche LEGO-Bauteile wie beispielsweise Liftarme, Zahnräder und Pins sowie zwei Gummibänder verwendet.

III. Realisierung

In diesem Abschnitt liegt der Fokus auf der detailreichen Beschreibung des Aufbaus sowie auf der Programmierung der einzelnen Funktionen des Zeichenroboters.

A. Konstruktion

1) Zeichenmechanismus: Der Zeichenmechanismus des Roboters besteht im Wesentlichen aus fünf Teilarmen: jeweils zwei Ober- und Unterarme pro Seite sowie einem Verbindungstück zwischen den beiden Motoren, siehe Abbildung 2. Die zwei Zeichenarme, bestehend aus Ober- und Unterarm, sind aus Liftarmen und Pins aufgebaut. Dabei ist jeweils der Oberarm über ein Gelenk mit dem Unterarm verbunden. Da hier zusätzliche Stabilität erforderlich ist, wird parallel zu dem Gelenk ein Gummiband gespannt.

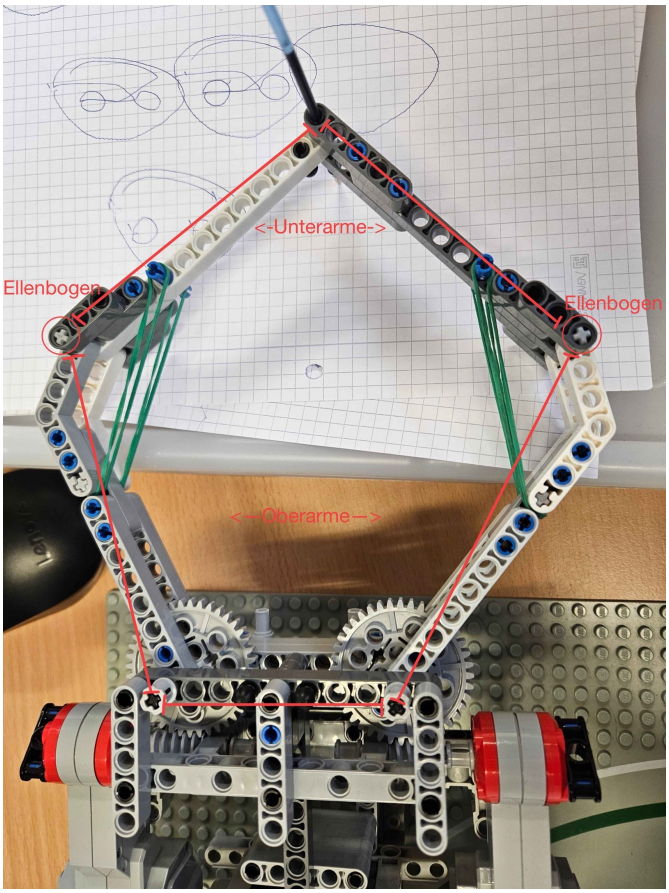


Abbildung 2. Konstruktion des Zeichenmechanismuses

An der vordersten Stelle, wo beide Zeichenarme zusammenlaufen, ist eine Kugelschreibermine befestigt, welche als Stift des Zeichenroboters agiert. Die beiden Zeichenarme sind jeweils an einem großen Zahnrad befestigt, mit dem sich der Winkel der Zeichenarme einstellen lässt.

Der Antrieb dieser Zahnräder erfolgt mithilfe der EV3-Motoren. Da für präzise Zeichnungen eine Vielzahl an sehr kleinen Bewegungen erforderlich ist und sich die EV3-Motoren nicht um beliebig kleine Gradzahlen drehen können, wurde ein Schneckengetriebe zwischen den Motoren und den großen Zahnrädern integriert. Dies bewirkt eine Untersetzung von 40:1, wodurch sowohl präzisere Bewegungen der Zahnräder als auch ein höheres Drehmoment der Motoren ermöglicht wird.

2) Hebemechanismus: Zur Darstellung von Wörtern oder komplexeren Zeichnungen ist es erforderlich, dass der Zeichenroboter den Stift vom Untergrund kontrolliert anheben und wieder absetzen kann. Um dies zu ermöglichen, wurde ein Hebemechanismus eingebaut, siehe Abbildung 3. Der im Roboter implementierte Hebemechanismus arbeitet mit einem Gegendruck am hinteren Ende der Konstruktion. Dadurch werden die Zeichenarme sowie der an der Front befestigte Stift angehoben. Dieses Prinzip ist möglich, da der gesamte Zeichenroboter lediglich über ein zentrales Gelenk mit dem Untergrund verbunden ist und sich folglich um diesen Drehpunkt kippen kann.

Die Erzeugung des Gegendrucks erfolgt durch einen dritten

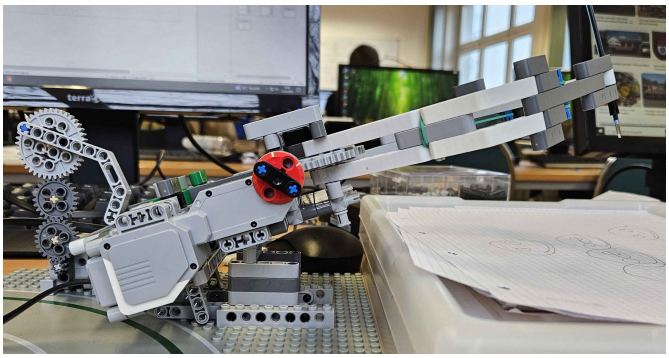


Abbildung 3. Hebemechanismus

Motor, der hinter dem Roboter fest an der Bodenplatte angebracht ist. Dieser steuert über Zahnräder einen Liftarm an und verändert dessen Winkel, wodurch der Liftarm auf den Zeichenroboter Druck ausübt. Auf diese Weise ist es dem Zeichenroboter möglich, mehrere voneinander unabhängige Linien zu zeichnen.

B. Programmierung

Ziel ist es, kartesische Zielpunkte, an denen der Stift sein soll, vorzugeben und daraufhin die Winkel der Zeichenarme zu bestimmen. Für die Realisierung wurde inverse Kinematik in MATLAB implementiert. Die Bestimmung der Position des jeweiligen Ellenbogengelenks erfolgt über die Berechnung der Schnittpunkte zweier Kreise. Einer der beiden Kreise hat seinen Mittelpunkt bei der aktuellen Position des Stiftes, wobei der Radius die Länge des Unterarms $r2$ hat. Der andere Kreis hat seinen Mittelpunkt beim Drehpunkt des Zahnrads, wobei hier der Radius der Länge des Oberarms $r1$ entspricht. Für jeden Zeichenarm ergeben sich somit zwei mögliche Schnittpunkte, welche in Abbildung 4 für den linken Zeichenarm exemplarisch veranschaulicht werden. Die Berechnung dieser Schnittpunkte erfolgt mithilfe der MATLAB-Funktion `circcirc`.

Aufgrund der mechanischen Gegebenheiten des Zeichenroboters ist jedoch nur einer der beiden Punkte tatsächlich vom Ellenbogen erreichbar. Um zu vermeiden, dass unerreichbare Positionen angesteuert werden, wurde für Motor 1 mittels einer min-Funktion der kleinere Winkel und für Motor 2 mittels einer max-Funktion der größere Winkel festgelegt. Sobald der Punkt des Ellenbogens bestimmt ist, kann die erforderliche Winkeländerung des Zahnrads, um an diese Position zu gelangen, berechnet werden.

Die vom Stift zu zeichnende Spur besteht aus einer Menge gleichmäßig verteilter Punkte. Die Dichte der Punkte ist in MATLAB manuell einstellbar und beeinflusst die Auflösung der Zeichnung. Die Abarbeitung dieser Punkte durch den Roboter ist in Abbildung 5 dargestellt.

IV. Ergebnisdiskussion

Im Rahmen der praktischen Umsetzung traten verschiedene Differenzen zwischen dem theoretischen Ansatz und dem realen System auf. Ursachen hierfür sind unter anderem mechanisches Spiel, die elastischen Nachgiebigkeiten der Bauteile sowie

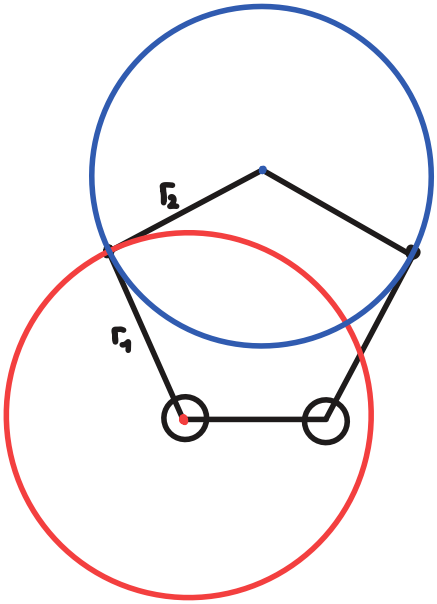


Abbildung 4. geometrische Berechnung der Ellenbogenposition

die begrenzte Winkelauflösung der eingesetzten EV3-Motoren. Diese Effekte führten zu sichtbaren Abweichungen in der Zeichnung.

Darüber hinaus konnten die beiden Motoren nicht synchron angesteuert werden. Dies hatte zur Folge, dass gerade Fahrbewegungen nicht exakt linear, sondern wellenförmig ausgeführt wurden.

Des Weiteren zeigte sich, je nach Position des Stiftes, eine Stauchung oder Streckung der Spur in y-Richtung. Dieses Phänomen ist auf die nichtlineare Änderung der y-Koordinate in Abhängigkeit des Winkels zurückzuführen.

Der Einbau eines Schneckengetriebes zur Verbesserung des Drehmoments stellte eine erforderliche Anpassung dar. Erst durch diese Maßnahme ließ sich die notwendige Spannung der Gummielemente zuverlässig einstellen und aufrechterhalten.

V. Zusammenfassung und Fazit

Im Zuge des zweiwöchigen LEGO-Mindstorm-Seminars wurde ein Zeichenroboter konzipiert, konstruiert und programmiert. Ziel war die Entwicklung eines Systems auf der Basis eines 5R-Parallelink-Mechanismus, das schreiben beziehungsweise zeichnen kann.

Der ursprüngliche Konzeptionsansatz mit einem frei beweglichen Roboterarm wurde aufgrund der hohen Konstruktiven und kinematischen Komplexität verworfen und durch einen 5R-Parallellink-Mechanismus ersetzt.

Die mechanische Umsetzung erfolgte mit Bauteilen des Systems LEGO Mindstorms. Zur Verbesserung von Drehmoment und Winkelauflösung wurde ein Schneckengetriebe integriert, während zeitgleich ein zusätzlicher motorisierter Hebemechanismus das kontrollierte Anheben des Stiftes ermöglicht. Die Berechnung der erforderlichen Zahnradwinkel erfolgte mittels

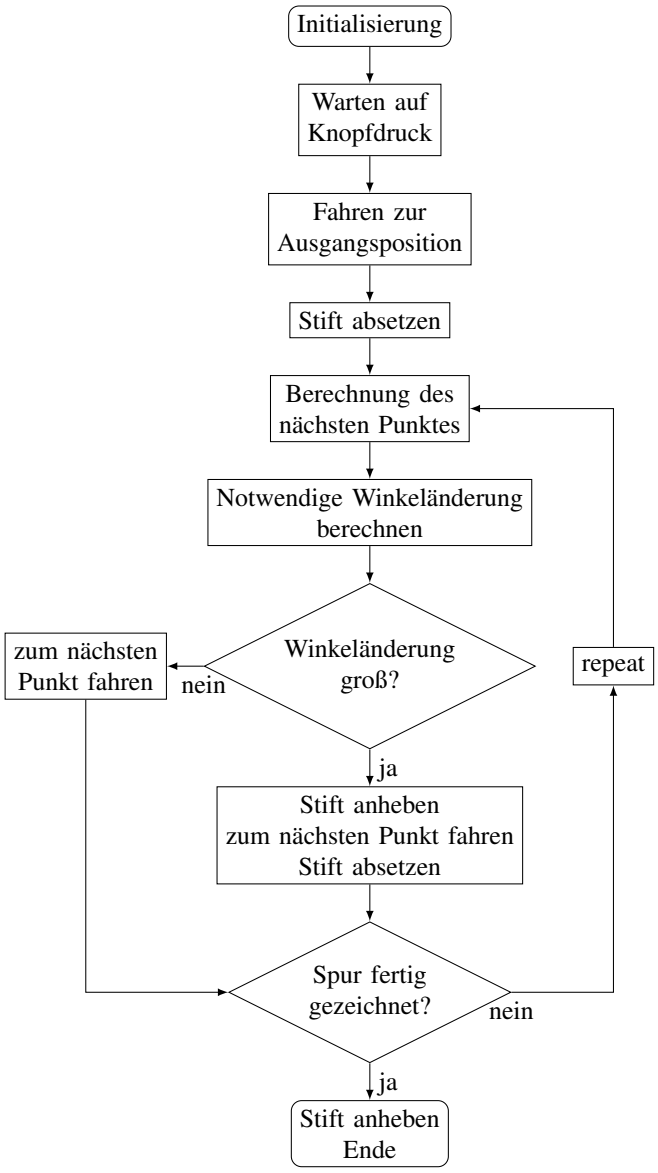


Abbildung 5. Programmablaufplan des Zeichenroboters

inverser Kinematik in MATLAB, wobei die Ellenbogenpositionen über Kreisschnittpunkte bestimmt wurden.

Bei der praktischen Umsetzung zeigten sich Differenzen zwischen dem theoretischen Ansatz und dem realen System, insbesondere durch mechanisches Spiel, elastische Nachgiebigkeit und begrenzte Winkelauflösung. Dennoch konnten einfachere Formen erfolgreich realisiert werden.

Literaturverzeichnis

[1] Vásquez-Pineda Bryan, Cardona M.: *Forward and Inverse Kinematics of a 5R Parallel Planar Robot*. In : Technology and Engineering Management Society Conference (TEMSCON LATAM), (2025), http://dx.doi.org/10.1109/TEMSCONLATAM65810.2025.11238935, DOI: 10.1109/TEMSCONLATAM65810.2025.11238935

Zeichenroboter

Vincent Schwalm, B. Sc. Elektrotechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des hier beschriebenen Projektes des Seminars Elektro- und Informationstechnik der Otto-von-Guericke-Universität 2026 wurde ein Zeichenroboter mittels theoretischer und praktischer Lösungen realisiert. Nicht nur mathematische Konzepte wie inverse Kinematik werden angerissen, sondern auch mechanische, wie die Limitationen der verwendeten LEGO-Mindstorms-Aspekte. Das Ganze wird in einer Auswertung diskutiert und mit einem Ausblick auf zukünftige Arbeiten vollendet.**

***Schlagwörter*—Legoroboter, Robotik, Mensch-Maschine Interaktion, Zeichnen.**

I. Einleitung

ZEICHENROBOTER stellen eine anschauliche Anwendung robotischer Kinematik im zweidimensionalen Raum dar. Sie ermöglichen die automatisierte Übertragung digital definierter Spuren auf eine physische Zeichenfläche und verbinden dabei mechanische Konstruktion, mathematische Modellierung sowie softwarebasierte Ansteuerung, und auch ein bisschen Kunst.

Ziel dieser Arbeit ist die Entwicklung eines planaren Zeichenroboters auf Basis eines geschlossenen 5R-Mechanismus, realisiert mit Komponenten des Systems „LEGO Mindstorms“. Ein 5R-Mechanismus ist ein kinematisches System mit zwei Freiheitsgraden, das aus fünf Gliedern besteht, die durch fünf Drehgelenke in einer geschlossenen Schleife verbunden sind. Das zu lösende Problem bestand in der Positionierung eines Stiftes innerhalb eines definierten Arbeitsraumes bei Übersetzung von Motordrehungen in X-Y-Bewegungen.

Besondere Anforderungen ergaben sich aus der mechanischen Struktur, welche eine Berechnung der inversen Kinematik erforderlich machte. Darüber hinaus mussten mechanische Maßnahmen zur Drehmomentverstärkung implementiert werden, um elastische Spannelemente im System nutzen zu können.

Die zentrale Idee des Projekts bestand darin, die inverse Kinematik geometrisch über Kreisschnittpunkte zu bestimmen und diese Berechnung in MATLAB zu implementieren. Dadurch konnte eine flexible Spurgenerierung realisiert werden, mit der geometrische Formen wie Kreise sowie komplexere Figuren, beispielsweise ein Smiley, gezeichnet werden konnten.

II. Vorbetrachtungen

A. Bestehende Lösungsansätze

Zeichenroboter werden häufig entweder als kartesische Plottersysteme oder als serielle Manipulatoren ausgeführt. Kartesische Systeme zeichnen sich durch eine einfache kinematische Beschreibung aus, benötigen jedoch lineare Führungssysteme mit entsprechendem Bauraum. Serielle Roboterarme besitzen einen größeren Arbeitsraum, weisen jedoch eine geringere strukturelle Robustheit auf.

Eine alternative Bauform stellt der planare 5R-Parallellink-Mechanismus dar. Dieser bietet eine höhere Steifigkeit bei gleichzeitig kompakter Bauweise. Als Literaturquelle für dieses Projekt fand sich der Artikel von Bryan Vasquez-Pineda [1].

B. Genutzte Verfahren

Für die eigene Umsetzung wurde ein geometrischer Ansatz zur Lösung der inversen Kinematik gewählt. Anstatt eine geschlossene analytische Lösung herzuleiten, wird der Schnittpunkt zweier Kreise bestimmt, wodurch sich die Gelenkpositionen berechnen lassen.

Die Implementierung der kinematischen Berechnungen sowie der Spurplanung erfolgte vollständig in MATLAB. Dieser Ansatz ermöglicht eine flexible Generierung von Zielpunkten sowie eine variable Auflösung der zu zeichnenden Konturen.

III. Umsetzung

A. Konzept und Kinematik

Zur mathematischen Beschreibung wird ein kartesisches Koordinatensystem verwendet, dessen Ursprung im linken Basismotor (Motor 1) liegt. Der zweite Motor befindet sich auf der x-Achse im Abstand d bei $(d, 0)$. Eine Skizze zum Aufbau ist in Abb. 1 zu sehen.

Der Mechanismus entspricht einem planaren 5R-System mit zwei Armen der Länge l_1 und zwei Koppelgliedern der Länge l_2. Für eine gewünschte Endeffektorposition (x, y) wird die inverse Kinematik geometrisch gelöst. Eine Visualisierung des entstehenden Programms ist im Programmablaufplan in Abb. 2 aufgeführt.

B. Inverse Kinematik mit Kreisschnitt

Die Berechnung erfolgt durch Bestimmung des Schnittpunktes zweier Kreise:

- Kreis um den jeweiligen Motor mit Radius l_1
- Kreis um den Endeffektor mit Radius l_2

Ein anderer Lösungsansatz für die inverse Kinematik wäre das Errechnen der Winkel über Winkel-Seitenbeziehungen. Dieser braucht jedoch eine wesentlich kompliziertere Mathematik, die in der Kürze dieses Projektes von zu großem Umfang war. Zur Bestimmung der Schnittpunkte wird in MATLAB die Funktion `circcirc` verwendet, Ausschnitte des Quellcodes für die Implementierung lassen sich in Abb. 3 finden. Im Quellcode wird zuerst die Winkelverschiebung der genutzten Arme auf 14 Grad gesetzt. Dies war nötig, da die Arme nicht direkt am Gelenk verbunden waren. Danach werden Koordinaten für die Mittelpunkte der zwei zu erstellenden Kreise

DOI: 10.24352/UB.OVGU-2026-047

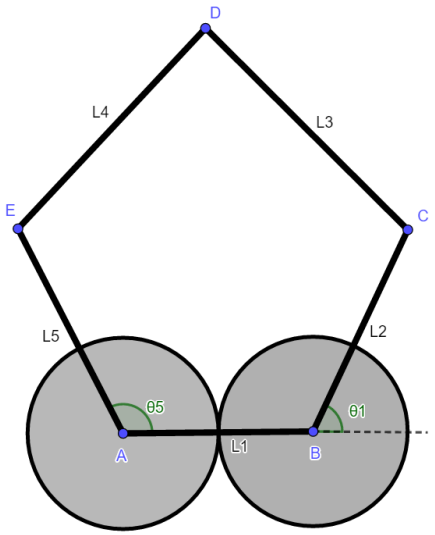


Abbildung 1. Skizze zum Aufbau

gesetzt, und über Winkelfunktionen die nötigen Motorwinkel, um die durch die Kreisfunktion berechneten Schnittpunkte zu erreichen, berechnet. Zuletzt wird die präferierte Winkellösung ausgewählt. Wie oben erwähnt, wurde die `circcirc`-Funktion verwendet, diese liefert zwei mögliche Lösungen. Aufgrund mechanischer Einschränkungen unseres Roboters war ausschließlich die sogenannte „Elbow-Up"-Konfiguration realisierbar, welche ein „Einknicken"- der Arme verhindert. Mit der sogenannten „Elbow-Down"- Konfiguration wäre ein anderer bzw. größerer Arbeitsradius möglich, diese ließ sich allerdings der Gelenksteifigkeit wegen nicht umsetzen.

Die Auswahl der korrekten Lösung erfolgte pragmatisch über eine einfache Entscheidungsstrategie:

- Für Motor 1 wurde der kleinere Winkel mittels einer `min`-Funktion gewählt.
- Für Motor 2 wurde der größere Winkel mittels einer `max`-Funktion gewählt.

Durch diese Auswahl wird konstant die Elbow-Up-Konfiguration erzwungen. Mit mehr Zeit wäre ein Algorithmisches finden der optimalen Lösung für jeden Motorwinkel mit mehreren Konfigurationen denkbar.

C. Spurgenerierung

Die Spur des Endeffektors wurde direkt im MATLAB-Code definiert. Hierbei wurden diskrete Zielpunkte mit variabler Auflösung generiert. Eine höhere Punktdichte führte zu einer feineren Approximation der gewünschten Kurve, erhöhte jedoch die Anzahl der notwendigen Motorbewegungen.

Zunächst wurde ein Kreis parametrisch erzeugt. Anschließend wurde das Verfahren erweitert, um eine komplexere Figur in Form eines Smileys zu realisieren. Die berechneten Winkelwerte wurden sequenziell an die Motoren übertragen.

D. Mechanische Realisierung

Der mechanische Aufbau wurde mit Komponenten aus dem System „LEGO Mindstorms" realisiert. Zur Verbesserung der

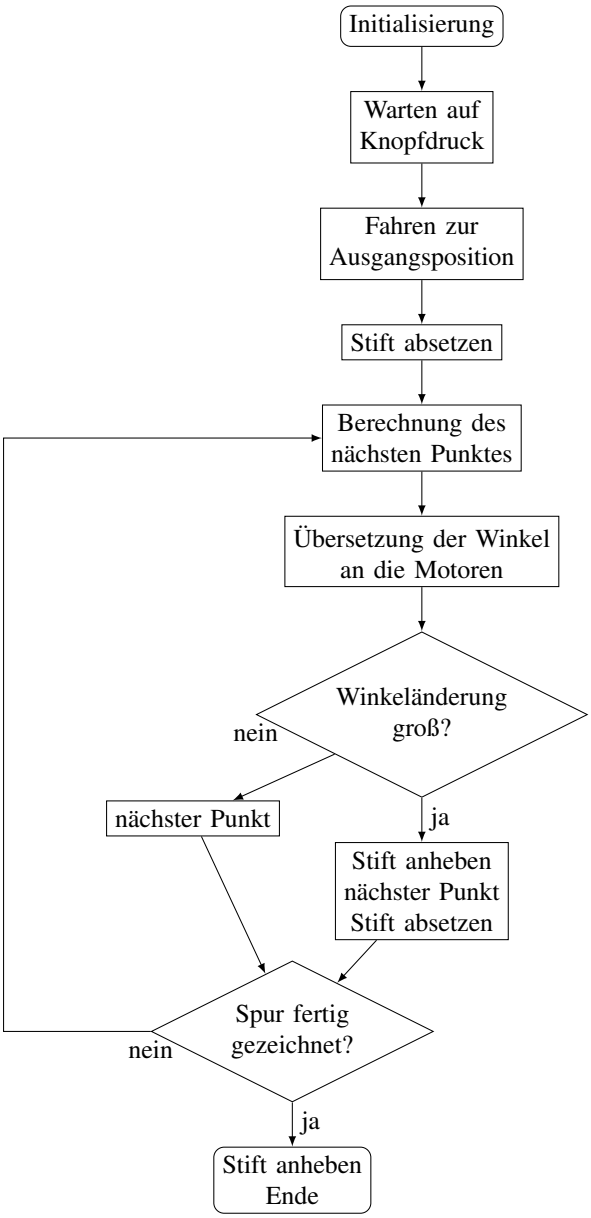


Abbildung 2. Programmablaufplan des Zeichenroboters

Bewegungsstabilität wurden Gummielemente integriert, die eine elastische Vorspannung erzeugen und insbesondere Abwärtsbewegungen des Endeffektors unterstützen. Desweiteren sollte es dem Roboter möglich sein, den Stift abzusetzen und anzuheben, um auch komplexere Formen zu zeichnen. Implementiert wurde das Ganze durch einen Kipp-Mechanismus, der über einen dritten Motor betätigt wurde, wie in Abb. 4 zu sehen.

In der initialen Konstruktion zeigte sich jedoch, dass das verfügbare Motordrehmoment nicht ausreichte, um diese Gummibänder ausreichend zu spannen. Daher wurde ein Schneckengetriebe implementiert, welches das Drehmoment deutlich erhöhte und zusätzlich eine Selbsthemmung bewirkte. Die fertige Konstruktion lässt sich in Abb. 5 einsehen.

```
function [phi1,phi2]=inverse_kinematics(x_pen,y_pen,d,arm1,arm2,hand1,hand2)
  % Positionen des linken Ellenbogen als Schnittpunkt zweier Kreise in m -> Skalar
  % circles are given with format: [XC YC R]
  offset = 14; %motorarm degree offset
  circle1=[0,0,arm1];
  circle2=[x_pen,y_pen,hand1];
  points=intersectCircles(circle1,circle2);
  % zugehörigen Winkel ausrechnen in ° -> Skalar
  phi1_a=atan2d(points(1,2),points(1,1));
  phi1_b=atan2d(points(2,2),points(2,1));
  % größeren der beiden Winkel nehmen in ° -> Skalar
  phi1=max(phi1_a,phi1_b)+offset;
  % Positionen des rechten Ellenbogen als Schnittpunkt zweier Kreise in m -> Skalar
  % circles are given with format: [XC YC R]
  circle1=[d,0,arm2];
  circle2=[x_pen,y_pen,hand2];
  points=intersectCircles(circle1,circle2);
  % zugehörigen Winkel ausrechnen in ° -> Skalar
  phi2_a=atan2d(points(1,2),points(1,1)-d);
  phi2_b=atan2d(points(2,2),points(2,1)-d);
  % kleineren der beiden Winkel nehmen in ° -> Skalar
  phi2=min(phi2_a,phi2_b)-offset;
end
```

Abbildung 3. Inverse-Kinematik Funktion

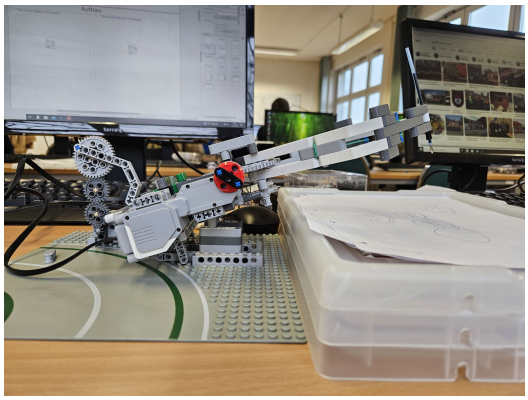


Abbildung 4. Kipp-Mechanismus in Aktion

E. Praktische Einschränkungen und Beobachtungen

Trotz funktionierender kinematischer Berechnung traten Abweichungen in der tatsächlichen Zeichnung auf. Winkelfehler, mechanisches Spiel sowie Bauraumlimitationen führten insbesondere bei Kreisbewegungen zu sichtbaren Verzerrungen.

Ein weiteres wesentliches Problem bestand darin, dass beide Motoren nicht exakt gleichzeitig angesteuert werden konnten. Die sequentielle Ansteuerung führte dazu, dass lineare Bewegungen nicht als ideale Geraden ausgeführt wurden, sondern eine leicht wellige Struktur aufwiesen.

Diese Effekte verdeutlichen die Diskrepanz zwischen Modell und realem mechanischem System. Auch die Nutzung der Formeln aus der wissenschaftlichen Arbeit von Abdel Gahny Mohammed [2] erwies sich im Nachhinein als fehlerhaft, sodass letztendlich auf die Kreisschnittpunkt-Lösung ausgewichen wurde.

IV. Ergebnisdiskussion

In der praktischen Umsetzung traten ebenfalls mehrere Abweichungen zwischen theoretischem Modell und realem System auf. Mechanisches Spiel, elastische Verformungen sowie begrenzte Winkelauflösung führten zu sichtbaren Verzerrungen, insbesondere bei Kreisbewegungen. Zusätzlich konnten die beiden Motoren nicht simultan angesteuert werden, wodurch lineare Bewegungen nicht ideal geradlinig, sondern leicht wellig ausgeführt wurden.

Die Integration eines Schneckengetriebes zur Drehmomentverstärkung stellte eine notwendige konstruktive Anpassung dar. Erst durch diese Maßnahme konnte die erforderliche Vorspannung der Gummielemente zuverlässig realisiert werden. Desweiteren gelang uns die Programmierung einer Simulation in MATLAB, mithilfe derer es uns möglich war, Spuren des Roboters im Voraus auszuwerten und mit der realen Umsetzung zu vergleichen. Insgesamt zeigt das Projekt deutlich die Wechselwirkung zwischen Theorie und praktischer mechanischer Umsetzung.

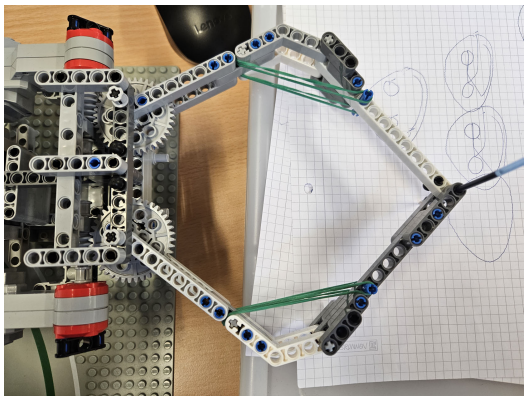


Abbildung 5. Fertiger Zeichenroboter

V. Zusammenfassung und Fazit

Im Rahmen des Projekts konnte ein funktionsfähiger planarer Zeichenroboter auf Basis eines 5R-Parallellink-Mechanismus realisiert werden. Die kinematische Modellierung sowie die Implementierung der inversen Kinematik in MATLAB erwiesen sich als geeignet, um definierte Spuren reproduzierbar abzufahren.

Die praktische Umsetzung mit LEGO-Komponenten erforderte konstruktive Anpassungen, insbesondere die Integration eines Schneckengetriebes zur Erhöhung der Belastbarkeit der Motoren. Die experimentellen Ergebnisse zeigen, dass das System in der Lage ist, definierte Spuren wie Kreise oder komplexere Figuren nachzufahren.

Für zukünftige Arbeiten könnten eine verbesserte Präzision, eine synchrone Motoransteuerung sowie eine geschlossene Positionsregelung des Endeffektors implementiert werden. Ebenso wäre eine Arbeitsraumanalyse mit Optimierung der Armlängen denkbar, um die Genauigkeit weiter zu erhöhen, oder ein Algorithmus, um die Ellenbogenlösungen konstant ideal auszuwählen.

Literaturverzeichnis

[1] Vasquez-Pineda, B. ; Cardona, M. N.: Forward and Inverse Kinematics of a 5R Parallel Planar Robot. In: *Technology and Engineering Management Society Conference (TEMSCON LATAM)* (2025). http://dx.doi.org/10.1109/TEMSCONLATAM65810.2025.11238935. – DOI 10.1109/TEMSCONLATAM65810.2025.11238935

[2] International Journal of Business Engineering and applied Sciences (Hrsg.): *Mathematical Model of two-DOF five links closed-chain mechanism (pantograph)*. 1. Future University in Egypt: International Journal of Business Engineering and applied Sciences, 2021. https://www.researchgate.net/publication/351809143_MATHEMATICAL_MODEL_OF_TWO-DOF_FIVE_LINKS_CLOSED-CHAIN_MECHANISM_PANTOGRAPH