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.**

Abbildung 1. Skorpion

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.

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

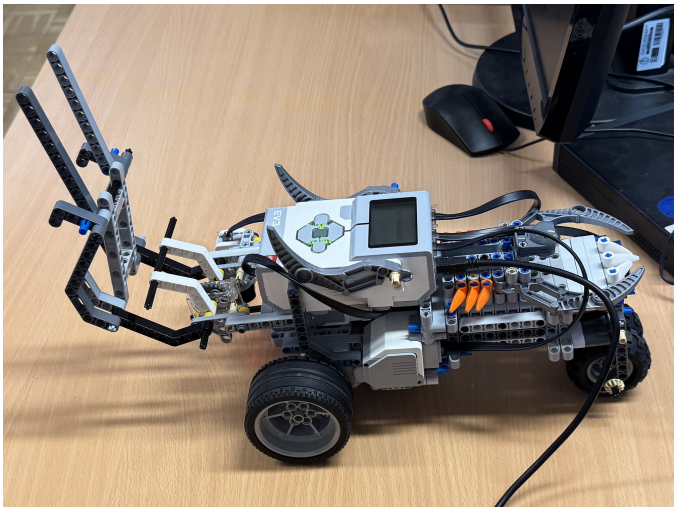

Abbildung 2. Brückenlegeroboter in Skorpionform

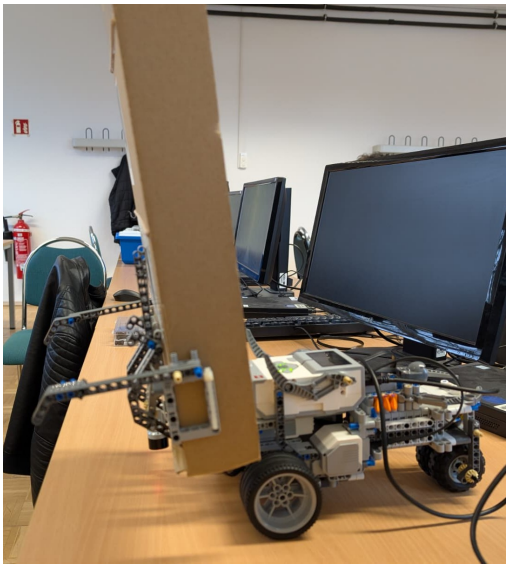

Abbildung 3. Implementierung

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.

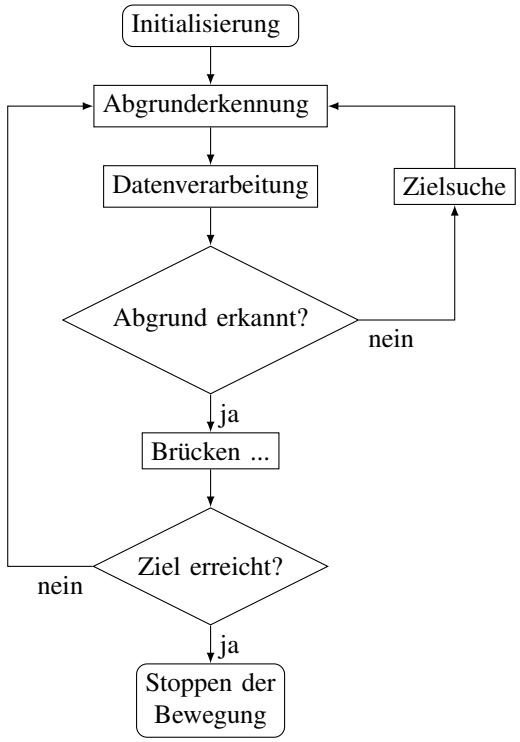

Abbildung 4. Beispielhafter Programmablaufplan zur Erläuterung der Lückenerkennung und Brückenlegung

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