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

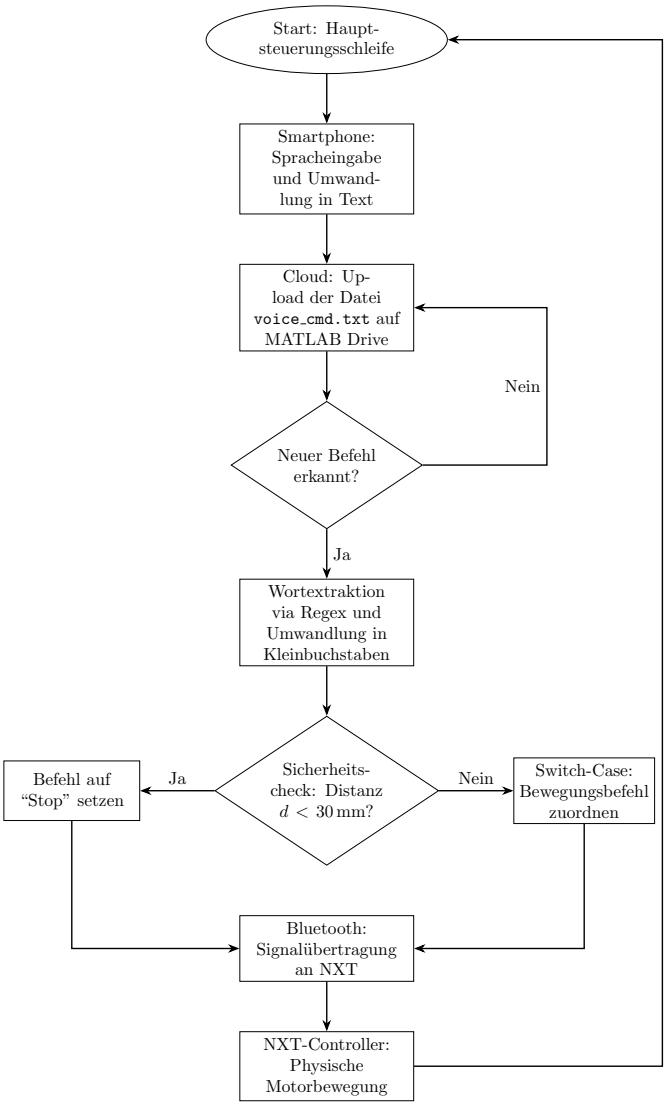

Abbildung 2. Systemsteuerung: Datenfluss vom Smartphone über MATLAB Drive zur Motorsteuerung mit Priorisierung der Hinderniserkennung (Ultraschallsensor).

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.