Selbstbalancierender Roboter auf einem Ball

Kseniia Shustikova, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Zusammenfassung*—Im Rahmen des LEGO-Mindstorms-Seminars an der Otto-von-Guericke-Universität Magdeburg wurde ein selbstbalancierender Roboter entwickelt, der auf einer Kugel statt auf Beinen oder Rädern stabilisiert wird.**

Zur Erfassung der Neigung in X- und Y-Richtung werden zwei Gyrosensoren verwendet. Ein Algorithmus verarbeitet diese Sensordaten und steuert zwei Motoren so, dass der Roboter aktiv Korrekturen vornimmt, um das Gleichgewicht auf der Kugel zu halten. Die Programmierung erfolgte mit MATLAB.

Am Ende des Projektseminars wurde der Roboter den Teilnehmern vorgestellt.

***Schlagwörter*—LEGO-Mindstorm, MATLAB, OVGU, Projektseminar, Roboter.**

I. Einleitung

DER selbstbalancierende Roboter könnte dank seiner Fähigkeit, das Gleichgewicht auf einem Ball zu halten, ein großartiger persönlicher Assistent sein. Da er sehr flexibel und stabil ist, kann er kleine oder große Gegenstände tragen (je nach Größe des Roboters) und bei einigen Aufgaben helfen. Aus denselben Gründen könnte er auch in Krankenhäusern eingesetzt werden, und wenn ein paar zusätzliche Sensoren und Kameras hinzufügen wurden, könnte er seinen Platz im Sicherheitsbereich finden.

Der Roboter hat 2 Gyrosensoren, die den Winkel an der X- und Y-Achse messen. Außerdem hat er 2 große Motoren, die zwei Räder steuern. Mit dem richtigen Algorithmus werden die, von den Gyrosensoren gelesenen, Daten mit mehreren anderen Daten multipliziert und dann als Geschwindigkeit an die Motoren gesendet. Durch das Beschleunigen und Verlangsamen der Motoren in die eine oder andere Richtung muss sich der Roboter auf der Kugel aufrecht halten und seine Stabilität bewahren [1].

In diesem Artikel wurde daher beschrieben werden, wie dies erreicht werden kann, welche Herausforderungen auf dem Weg dorthin auftreten können und welche Lösungsmöglichkeiten es gibt.

II. Vorbetrachtungen

A. Die Idee und Prinzip

Die Idee, einen Roboter zu bauen, der Menschen helfen kann, gibt es schon seit dem ersten Tag des Projektseminars. Die Fähigkeit, sich frei zu bewegen und stabil zu sein, egal, welche Hindernisse es gibt, dürfte also das erste und wesentliche Merkmal gewesen sein. Um diese freie Bewegung zu erreichen, sollte der Roboter keine Räder oder Beine haben, da er sonst nicht so leichtgängig ist, wie er sein sollte.

Das Funktionsprinzip basiert des Roboters auf einem einziegen Ball, die als bewegliche Basis dient. Der Roboter balanciert auf einem Ball und nutzt es als Antriebselement. Eingebaute Gyrosensoren messen die Neigungs- und Drehwinkel des Roboters in Echtzeit, so dass die aktuelle Position des Roboters immer bekannt ist. Diese Sensordaten werden in einen Steuerungsalgorithmus eingespeist, der mithilfe eines PID-Reglers kontinuierlich berechnet, wie die Motoren arbeiten müssen, um das Gleichgewicht zu halten.

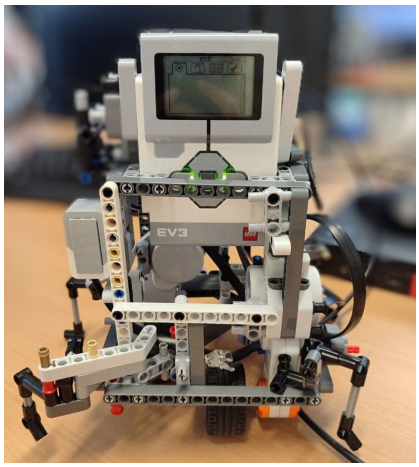

Abbildung 1. Fertige Konstruktion des Roboters

Die Motoren sind in der Lage, die Leistung dynamisch in Abhängigkeit vom Winkel und dessen Vorzeichen anzupassen. Dies ermöglicht es dem Roboter, stabil zu bleiben und sich in jede gewünschte Richtung zu bewegen.

B. Roboterteile

Die Teile, die zum Bau des Roboters verwendet wurden (siehe Abbildung 1):

1) Gyrosensoren 2×
2) Große LEGO-Motoren 2×
3) EV3-Brick
4) Batterie
5) Kabel
6) Verschiedene LEGO-Teile

C. Gyrosensoren

Die Gyrosensoren sind für den selbstbalancierenden Roboter auf einem Ball unerlässlich, da sie die Winkel und Neigungswinkel pro Sekunde auslesen und so präzise Informationen über den aktuellen Neigungszustand liefern. Dieser besondere Roboter ist mit zwei Gyrosensoren ausgestattet, erster befindet sich auf der linken Seite und misst kontinuierlich die Winkelveränderungen entlang der Y-Achse, während der andere Sensor auf der

DOI: 10.24352/UB.OVGU-2025-027

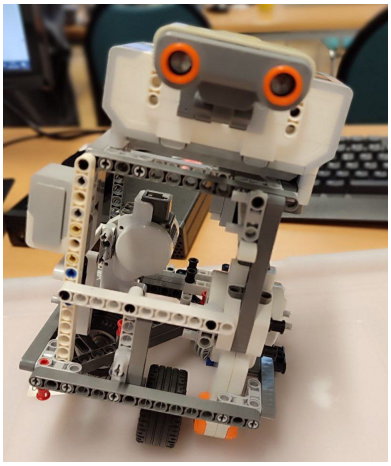

Abbildung 2. Der erste Entwurf

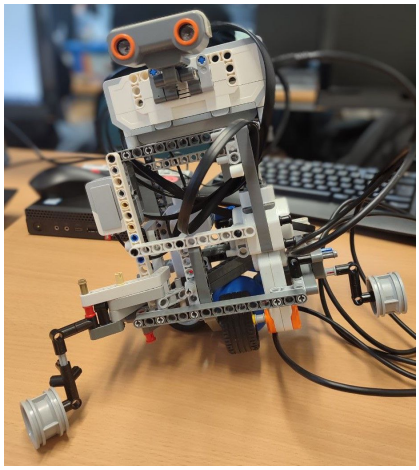

Abbildung 3. Konstruktion mit zusätzlichen Rädern

Rückseite die Winkelveränderungen entlang der X-Achse erfasst.

Diese Sensoren arbeiten mit hoher Aktualisierungsrate und Genauigkeit, was entscheidend ist, damit der Roboter schnell auf Veränderungen reagieren kann. Ihre Hauptaufgabe besteht also darin, die relevanten Daten in Echtzeit zu erfassen, die anschließend in einem Steuerungsalgorithmus – in diesem Fall PID-Algorithmus – korrekt berechnet und an die Motoren weitergeleitet werden. Die Motoren nutzen diese Informationen, um kontinuierlich kleine Korrekturen vorzunehmen, wodurch der Roboter immer wieder in die aufrechte Position zurückgeführt wird und somit stabil bleibt.

III. Hauptteil

A. Aufbau

Der Bau des Roboters wurde durch ein Handbuch mit detaillierten Anweisungen erleichtert, jedoch gab es noch einige Probleme, die eine Anpassung der Konstruktion erforderlich machten [2].

Das Hauptproblem bei der Entwicklung des robotergestützten Auswuchtsystems war die Herausforderung, das gewünschte Gewicht zu erreichen. Der EV3-Stein erwies sich als zu schwer. Bei der anfänglichen Konstruktion wurde er oben auf dem Roboter positioniert, was dazu führte, dass er bei den leichtesten Bewegungen herunterfiel (siehe Abbildung 2). Trotzdem wurde zunächst versucht, zusätzliche Räder anzubringen, um eine bessere Stabilität zu erreichen, ohne dabei auf die Verlagerung des Schwerpunkts zu achten (siehe Abbildung 3).

Nach einer Überprüfung der Konstruktion des Roboters wurde beschlossen, den Massenschwerpunkt (den EV3-Brick) so niedrig wie möglich zu platzieren, um die Stabilität zu erhalten (siehe Abbildung 1). Außerdem die zusätzlichen Räder wurden entfernt und stattdessen nur kleine Beine hinzugefügt, die dem Roboter bei Tests Halt geben.

B. Sensorik und Regelung

Wie bereits erwähnt, werden die Gyroskope verwendet, um die Winkel entlang der X- und Y-Achse zu überwachen. Diese Daten werden dann verarbeitet und an die Motoren

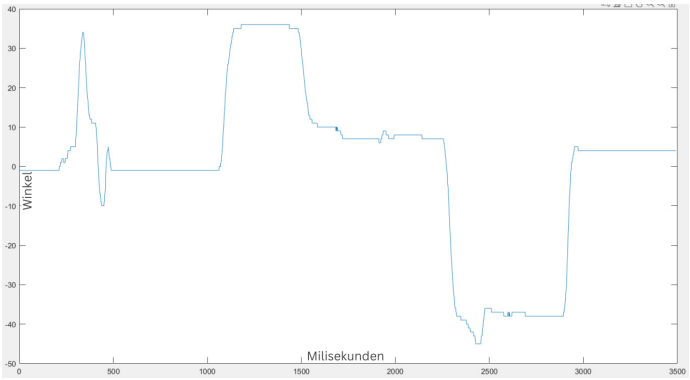

Abbildung 4. Grafik, die den Gyrosensor-Fehler anzeigt

gesendet. Der PID-Algorithmus ist das geeignete Werkzeug zur Verarbeitung dieser Daten [3].

Der PID-Algorithmus verwendet die Daten von Gyrosensoren, um das Gleichgewicht zu halten, indem er den aktuellen Winkel mit der gewünschten aufrechten Position vergleicht. Der proportionale Teil befasst sich mit dem unmittelbaren Fehler, der integrale Teil berücksichtigt die akkumulierten Fehler der Vergangenheit und der derivative Teil antizipiert plötzliche Änderungen. Die Summe dieser Terme bildet das Steuersignal, das den Ball in die entgegengesetzte Richtung jeder Neigung bewegt, wodurch eine kontinuierliche Rückkopplungsschleife entsteht, um den Roboter aufrecht zu halten.

Die folgende Formel veranschaulicht den Prozess, bei dem die PID-Werte zur Berechnung des Endergebnisses verwendet werden, das in diesem Fall die Leistung der Motoren ist.

$$\text{MotorPower} = K_p \cdot e + K_i \cdot \int e\,dt + K_d \cdot \frac{de}{dt} \qquad (1)$$

Allerdings traten bei jedem Test Fehler auf. Insbesondere die Gyroskopsensoren lieferten ungeeignete Daten. Wie in der Grafik dargestellt, ergibt die Ausgangsposition des Roboters (aufrecht) einen Winkel von 0. Dies ist die richtige Ausrichtung. Nach der Neupositionierung sollte der Winkel wieder auf Null zurückgehen. Dies ist jedoch nicht der Fall; stattdessen erhöht er (siehe Abbildung 4).

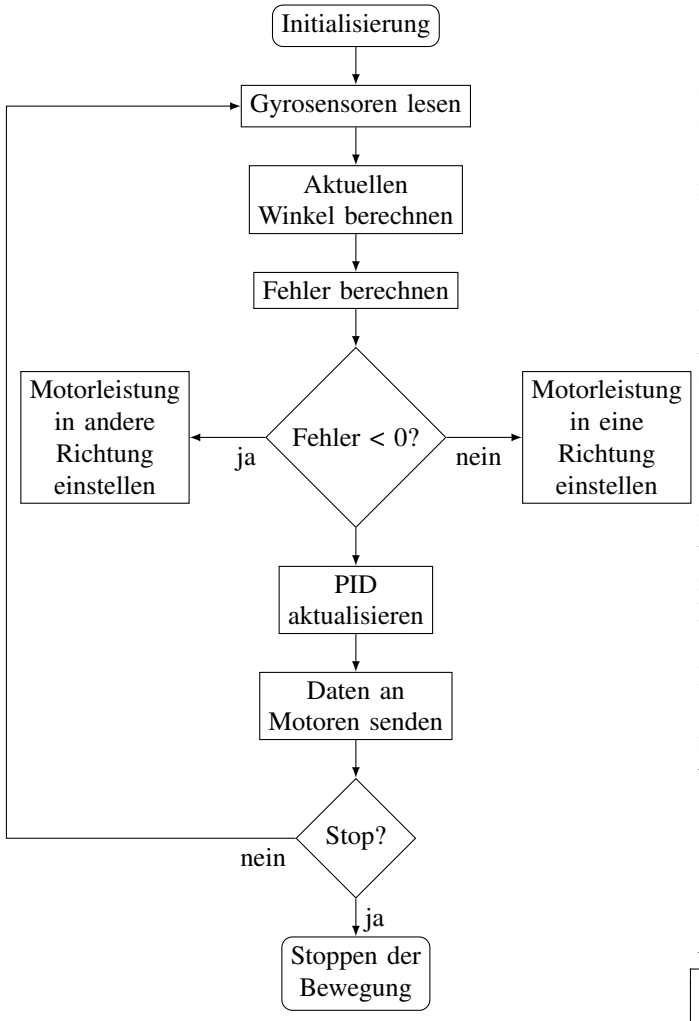

Abbildung 5. Flussdiagramm, das den Code darstellt

C. Funktionsweise und Stabilisierung

Zuerst initialisiert der Roboter alle notwendigen Komponenten, darunter seinen EV3-Brick, die Gyrosensoren und die Motoren. Dann werden die Daten der Gyrosensoren ausgelesen, die Informationen über die aktuelle Neigung liefern. Die Differenz zwischen dem gewünschten aufrechten Winkel und dem empfangenen Winkel wird als Fehler bezeichnet. Nach der Bestimmung des Vorzeichens des Fehlers aktualisiert das System den PID-Regler mit diesen neuen Informationen. Der PID-Algorithmus berechnet die erforderliche Korrektur auf der Grundlage von Proportional-, Integral- und Derivativtermen und erzeugt eine Steuerausgabe, die dann an die Motoren gesendet wird. Dieser aktualisierte Befehl veranlasst die Motoren, ihre Leistung so zu ändern, dass der Roboter wieder in eine aufrechte Position gebracht wird, und dieser

Prozess wird kontinuierlich fortgesetzt, solange der Roboter aktiv ist. In jeder Phase liest das System Sensordaten aus und berechnet den Winkel neu. Wenn eine Stopp-Bedingung erreicht wird - der Roboter hält an und es werden keine Daten mehr ausgelesen - kann kein Gleichgewicht mehr erreicht werden.

IV. Ergebnisdiskussion

Am Ende des Baus und der Programmierung wurde also dieses Ergebnis erhalten: die Räder in der Lage, ihre Geschwindigkeit und Drehrichtung auf der Grundlage der Daten von Gyrosensoren zu ändern, und genau - in einem positiven oder negativen Winkel, auf der x- oder y-Achse.

Der Roboter kann sich immer noch nicht auf einem Ball aufrecht halten, obwohl er durchaus versucht, seine Position zu ändern, um näher an die Oberseite des Balls heranzukommen und auf ihm zu bleiben. Dies liegt sowohl an der begrenzten Zeit für die Entwicklung als auch an ungenauen Sensordaten der Gyrosensoren - ein Problem, das bislang ungelöst blieb.

V. Zusammenfassung und Fazit

In zwei Wochen wurde viel Arbeit geleistet, auch wenn noch nicht alle Probleme gelöst sind. Durch diese Praxis konnte ich viele neue Kenntnisse und Fähigkeiten erwerben, wie die Arbeit mit Matlab und verschiedenen Sensoren, Motoren, Teamarbeit, Problemlösung und kritisches Denken. Auch wenn nicht alles so reibungslos verlief wie erwartet, sind die gewonnenen Erfahrungen von unschätzbarem Wert.

Und mit mehr Zeit wäre es möglich gewesen, die Ursache der fehlerhaften Sensordaten zu identifizieren und zu optimieren, wie häufig die Daten aktualisiert werden müssen, um eine stabile Balance des Roboters zu gewährleisten.

Anhang

Ein Beispiel für die Anordnung der Motorleistung in Abhängigkeit von einem positiven oder negativen Winkel.

```
if rate1 > rate2 && rate1 > 0 && rate2 < 0
    motorPower = -(Kp * error + Ki * integral
        + Kd * derivative);
    motorA.power = max(min(motorPower, 100),
        -100);
    motorB.power = 0;
end
if rate1 < rate2 && rate1 < 0 && rate2 > 0
    motorPower = (Kp * error + Ki * integral +
         Kd * derivative);
    motorB.power = max(min(motorPower, 100),
        -100);
    motorA.power = 1;
end
```

Literaturverzeichnis

[1] Wikipedia: *Ballbot*. : Wikipedia, Februar 2025. https://en.wikipedia.org/wiki/Ballbot

[2] Yamamoto, Yorihisa: *NXT Ballbot (Self-Balancing Robot On A Ball) Controller Design*. Version 1.0.0.0. : MATLAB, Februar 2025. https://ch.mathworks.com/matlabcentral/fileexchange/23931-nxt-ballbot-self-balancing-robot-on-a-ball-controller-design

[3] Wikipedia: *Proportional–integral–derivative controller*, Februar 2025. https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller