Der Erkundungsroboter

Ben Duwanoff, Elektrotechnik und Informationstechnik
Otto-von-Guericke-Universität Magdeburg

***Abstract*—Dieses Paper enthält das Ergebnis des Projektseminars Elektrotechnik/Informationstechnik (LEGO Mindstorms), dessen Ziel es ist einen Erkundungsroboter zu bauen. Dabei wurde ein NXT-Set der LEGO Mindstorms Serie und MATLAB als Programmiergrundlage verwendet. Zudem wurden für die Programmierung Bug-Algorithmen und der A*-Algorithmus erforscht sowie eine eigene Lösung eines Zielfindungsalgorithmus aufgestellt und ein Roboter mit diesen grundlegenden Funktionen konstruiert sowie sein autonomes Fahrverhalten im Zusammenhang getestet und ausgewertet.**

***Schlagwörter*—autonomes Fahren, Erkundung, Fahrzeug, LEGO Mindstorms NTX, Suchalgorithmus**

I. Vision und Antrieb

Die Erkundung der Welt ist ein urmenschliches Bedürfnis. Jedoch kommt der Mensch immer wieder an seine physikalischen Grenzen, die ersehnten Orte zu erreichen, seien es die Tiefen der Meere, die höchsten Berge oder fremde Planeten. Viele dieser Orte konnte sich der Mensch mit Hilfe von Technik zugänglich machen. Der Erkundungsroboter soll dabei das Problem lösen, dass eine direkte Steuerung nicht immer möglich oder gewollt ist, sowie die Aufzeichnung und Übertragung der Messdaten gewährleisten. Das Projekt beschäftigt sich zunächst mit der Bewegung, Steuerung und Orientierung eines solchen Fahrzeugs. Es soll entweder über eine Benutzeroberfläche ferngesteuert oder über eine Zielkoordinate eigenständig auf den Weg geschickt werden und dabei eine Kollision mit Hindernissen vermeiden.

II. Grundlagen und Nachforschungen

A. LEGO Mindstorms NTX Serie

In diesem Projekt wurde ein NTX-Baustein der LEGO Mindstorms Serie verwendet. Im NTX-Grundset sind ein NXT-Stein, drei Elektromotoren mit Rotationssensor, zwei Tastsensoren, einen Ultraschallsensor, ein Farbsensor bzw. Lichtsensor, ein Schallsensor, sieben RJ12 Kabelverbindungen, ein USB-Kabel sowie einige LEGO-Technic-Elemente, Bauanleitungen, eine Testfläche und NXT-G als integrierte Entwicklerumgebung enthalten. Diese NTX-Steuerungseinheit besitzt Anschlüsse für vier Sensoren und drei Elektromotoren. Angesteuert wird der NXT-Stein über ein USB A zu USB B Kabel, mit dessen Hilfe man entweder den Roboter von einem Computer aus befehlen oder ein Programm auf den Roboter übertragen kann, welches dann mit Hilfe der Tasten und der Menüführung auf dem NTX-Stein direkt ausgeführt wird. Die Sensoren und Motoren werden durch RJ12-Kabelverbindung mit den NTX-Stein an den entsprechenden Anschlüssen verbunden [1]. Um das Programm zu realisieren, wurde MATLAB als Programmiersprache sowie als Compiler des Programms verwendet. Da MATLAB selbst jedoch keine Schnittstelle zur Steuerung von LEGO-Mindstorms-NTX-Bausteinen besitzt, diente die MATLAB-Bibliothek, die die RWTH Aachen bereits entwickelt hat und diese freundlicher Weise auf ihrer Seite [2] frei zur Verfügung stellt, als Solche.

B. Pfadsuchalgorithmen

Da der Roboter einen Weg zum Ziel finden soll, standen zur Wegfindung zunächst ein Bug-Algorithmus oder der A*-Algorithmus zur Auswahl. Der Bug-Algorithmus benötigt im Gegensatz zum A*-Algorithmus neben seinem Start und Ziel keine globalen Kenntnisse, da er sich an einem einfachen Verhalten orientiert. Er folgt einer geraden Linie zum Ziel und falls er auf ein Hindernis trifft, fährt er diesem nach rechts oder nach links entlang, um dann hinter dem Hindernis in einer geraden Linie Richtung Ziel weiter zu kommen. Es gibt zwei Varianten diesen zu realisieren, den Bug 1 und den Bug 2 Algorithmus. Bei dem Bug 1 Algorithmus fährt der Roboter einmal um das gesamte Hindernis herum und merkt sich zu jeder Position auf dem Weg den Abstand und die Koordinaten bis er am ersten Kontaktpunkt mit dem Hindernis wieder angekommen ist. Dann findet er den Punkt mit dem kürzesten Abstand zum Ziel und fährt noch einmal am Hindernis bis zu diesem Punkt entlang. Danach bleibt er wieder auf der Geraden aufs Ziel zu, bis er ankommt oder ein weiteres Hindernis erkennt. Der Bug2-Algorithmus verkürzt die Umfahrung des Hindernisses, da dieser nicht das ganze Objekt umrundet, sondern nur solange am Hindernis entlang fährt, bis er wieder auf die Gerade trifft, die am Anfang zum Ziel berechnet wurde. Dieser schlägt zeitlich dadurch in fast allen Fällen den Bug1-Algorithmus. Die Bug-Algorithmen benötigen sehr genaue Sensoren, da es durch Fehlermessungen zu Kollisionen oder Irrfahrten des Roboters kommt [3]. Der A*-Algorithmus benötigt im Gegensatz dazu alle globalen Informationen seines Zielgebietes, wodurch er auch ohne Sensoren den kürzesten Weg zum Ziel finden würde. Jedoch sollten zur Sicherheit einige Sensoren angebracht werden, damit der Roboter bei einer Fehlfunktion zum Start zurückkehren kann. Bei dem A*-Algorithmus berechnet man zunächst die Abstände aller möglichen Positionen im Zielgebiet zum Ziel. Danach markiert man die Positionen der Hindernisse und eröffnet mindestens zwei Listen. Die sogenannte offene Liste enthält alle noch nicht abgearbeiteten Positionen. Sie

DOI: 10.24352/UB.OVGU-2020-048

beinhaltet am Anfang nur den Startpunkt oder ist je nach spezieller Struktur leer und wird danach vom Algorithmus mit den zu bearbeitenden Positionen befüllt. Die zweite Liste ist die sogenannte geschlossene Liste. In diese werden alle Hindernisse mit ihrem Status als solche eingetragen. Man kann theoretisch auch eine eigene Liste für Hindernisse anlegen.

Nun beginnt der Algorithmus in einer Schleife mit der Position, welche in der offenen Liste steht und die geringsten Gesamtkosten zum Ziel hat. Diese setzen sich zusammen aus den Wegkosten, welche die kürzeste Strecke vom Start zu der entsprechenden Position darstellen und Restkosten, welche der direkte Abstand von der Position zum Ziel festlegt. Am Anfang beginnt der Algorithmus deshalb mit der Startposition. Danach berechnet er alle Kosten der um den fokussierten Punkt liegenden Positionen. Wobei sich die Wegkosten eines Nachbarn aus den Wegkosten des fokussierten Punktes und der Strecke zwischen diesen beiden Punkten errechnet. Zusätzlich werden die Koordinaten des fokussierten Punktes den Daten des Nachbarn angehangen, diese werden auch als Eltern bezeichnet. Danach wird geprüft, ob einer der Nachbarn das Ziel oder ein Hindernis darstellt oder bereits in einer der Listen vorhanden ist. Wenn dies nicht der Fall ist, werden die Datensätze der betroffenen Positionen der offenen Liste hinzugefügt. Falls der Nachbar bereits vorhanden ist, wird geprüft, ob die neu berechneten Wegkosten kleiner als die bereits vorhandenen sind. Sollte das der Fall sein, werden die Datensätze des Nachbarn aus allen Listen entfernt und in der offenen Liste wird der neue Datensatz dieser Position eingetragen. Ist der Nachbar ein Hindernis wird er ignoriert und sollte er das Ziel sein, wird sein Datensatz zusammen mit dem der fokussierten Position in die geschlossene Liste verschoben und aus den anderen Listen entfernt. Danach wird die Schleife abgebrochen. Sonst geht es weiter, bis alle Nachbarn abgearbeitet sind. Nun wird der Datensatz, welcher am Anfang noch der Start ist, in die geschlossene Liste verschoben und aus der offenen Liste entfernt. Danach beginnt die Schleife wieder von vorn mit der Suche eines neuen fokussierten Punktes aus der offenen Liste.

Sollte die Schleife durch das Ziel abgebrochen werden, kann man über die Koordinaten der Eltern von Punkt zu Punkt bis zum Start zurückfahren und erhält so die kürzeste Strecke zwischen Start und Ziel. Die Datensätze der Positionen der kürzesten Strecke kann man dann noch in einer eigenen Liste speichern, um den kürzesten Weg nicht zweimal berechnen zu müssen [4],[5].

III. Realisierung des Erkundungsroborters

A. *Konstuktion und technischer Aufbau*

Der Erkundungsroboter besteht aus einem LEGO Mindstorms NTX-Stein, welcher oben am Konstrukt angebracht ist. Darunter befinden sich zwei Elektromotoren aus dem LEGO-Mindstorms-Set. Jeder Motor steuert dabei die Vorderräder der jeweiligen Seite an. Die Hinterräder sind auch Antriebsräder, werden jedoch über ein Getriebe zum jeweiligen Vorderrad betrieben. Dadurch benötigt der Roboter keine Lenkung, sondern wird durch die Geschwindigkeit der beiden Motoren gesteuert. Durch die eingebauten Rotationssensoren kann so die Fahrstrecke oder der Drehwinkel für das Fahren bestimmt werden.

Um die Fahrweite für die Programmierung zu kalibrieren, wurde der Roboter testweise mit verschiedenen Motordrehweiten gefahren bis ein Drehwinkel von 360 Grad gefunden wurde, damit der Roboter 20 cm nach vorn fährt. Um vorwärts oder rückwärts zu fahren, müssen sich hier beide Motoren in die gleiche Richtung bewegen. Um im Programm leichter die Position des Roboters bestimmen zu können, kann sich er nur auf der Stelle drehen, jedoch keine Kurven fahren. Deshalb drehen sich die Motoren beim Ausrichten des Roboters gleich schnell, jedoch in unterschiedliche Richtungen. Dabei wurde auch hier die Drehweite der Motoren für die einzelnen Drehungen über Testläufe kalibriert, sodass sich zum Ausrichten nach 90 Grad der vorwärts drehende Motor um 380 Grad und der rückwärts drehende Motor um 100 Grad drehen musste. Da die Motoren sich proportional zur Ausrichtung bewegen müssen, lassen sich die Drehweiten der Motoren für andere Winkel durch das Verhältnis der Winkel zum Kalibrierungswinkel als Vorfaktor bestimmen.

Abbildung 1: Konstruktion des Roboters

Zudem besitzt der Roboter noch einen nach vorn zeigenden Ultraschallsensor, um Hindernisse auf dem Weg entdecken zu können. Bei der Abfrage des Sensorwertes gibt der Ultraschallsensor automatisch den Abstand des davor liegenden Objektes in Zentimetern aus. Da der Sensor einen 8 Bit Wert zurückgibt, können die Abstände dabei nur bis zu 255 cm betragen. Sollte 256 cm ausgegeben werden, übersteigt der Abstand den Messbereich. Wenn man von vorn auf den Sensor schaut, befinden sich dort zwei Öffnungen. In der rechten Öffnung befindet sich der Ultraschallgenerator und in der Linken der Ultraschallsensor. Dadurch kann es bei zu kleinem Abstand zu Fehlermessungen kommen, weshalb Sensor nur für Werte über 6 cm verwendet werden kann.

B. *Steuerung und Programmierung*

Die Robotersteuerung besteht aus zwei Programmabschnitten: der manuellen Steuerung und der automatischen Zielfindung. Beide werden durch die gleiche grafische Benutzeroberfläche gesteuert. Bei der manuellen Steuerung besitzt der Roboter auf der Benutzeroberfläche einen Geschwindigkeitsregler und vier Bewegungsregler, welche aus einem Vorwärtsknopf, einem Rückwärtsknopf, jeweils einem Knopf, zur Rechts- und Linksdrehung um 45 Grad bestehen, sowie einen Bremsknopf, um den Roboter anzuhalten.

Wenn man den Vorwärts- oder Rückwärtsknopf drückt, fährt der Roboter solange in die entsprechende Richtung, wie man

den Knopf gerückt hält. Zudem hat der manuelle Modus einen Auffahrschutz einprogrammiert, der automatisch die Vorwärtsbewegung abbricht, sollte der Roboter einem Objekt zu nahe kommen. Dies wird durch eine Bewegungsabbruchsbedingung realisiert, die bei einem zu geringen Sensorwert des Ultraschallsensors initialisiert wird. Dadurch werden alle Motoren des Roboters angehalten. Danach kann die manuelle Kontrolle mit der grafischen Benutzeroberfläche den Roboter wieder übernehmen.

Für das automatische Finden des Ziels wurde eine Anlehnung an den Bug-Algorithmus verwendet, der aufgrund des zeitlichen Aufwandes stark vereinfacht wurde. So erkennt der Roboter Hindernisse, fährt jedoch an diesen nicht entlang, sondern initialisiert einen vorgegeben Ausweichpfad. Deshalb muss das Hindernis standardisiert sein und die Strecke der Umgehung für jede Art von Hindernis angepasst werden. Bei der automatischen Zielfindung gibt man dem Roboter die Zielkoordinaten, wobei diese immer in Bezug auf den Roboter übergeben werden müssen, da für ihn selbst keine Koordinaten übergeben werden. Für diesen Zweck wurden auf der Benutzeroberfläche zwei Eingabefelder bereitgestellt, welche einmal die x- und die y-Koordinate des Ziels verlangen. Andere Koordinatenformen wie Polarkoordinaten sind zudem nicht möglich. Wenn man nun auf den „Automatisch“-Knopf unterhalb der Eingabefelder drückt, entscheidet sich der Roboter zunächst für einen der drei folgenden Fälle:

1. Das Ziel befindet sich ausschließlich in x-Richtung oder in ausschließlich y-Richtung oder das Ziel befindet sich zwischen der x- und der y-Richtung. Sollte sich das Ziel nur in x-Richtung befinden, fährt der Roboter gerade aus auf das Ziel zu. Alle 20 Zentimeter erhöht er dann seine eigene x-Position, die am Programmstart null ist, um eins bis sie gleich der Zielkoordinate ist. Dann wird das automatische Programm beendet und der Roboter wartet auf eine Eingabe aus der grafischen Benutzeroberfläche. Sollte sich ein Hindernis auf dem Weg befinden, so erkennt der Roboter dieses mit Hilfe des Ultraschallsensors. Dabei misst der Sensor vor und während der Bewegung den Abstand vor sich. Sollte nun der Abstand zu gering werden, also unter 20 cm sein, bedeutet das, dass der Roboter den nächsten Streckenabschnitt nicht ohne Kollision mit dem Hindernis fahren kann.

   Deshalb stoppen zunächst alle Motoren, wodurch der Roboter anhält, um dann einen Ausweichpfad zu initialisieren, mit dem das Hindernis umfahren wird. Zuerst dreht sich der Roboter dazu um 90 Grad nach links, fährt dann 20 cm nach vorn und dreht sich um 90 Grad nach rechts. Danach fährt der Roboter 40 cm nach vorn, um das Hindernis mit Sicherheit umfahren zu haben und dreht sich erneut um 90 Grad nach rechts. Nun fährt der Roboter wieder 20 cm vor und dreht sich um 90 Grad nach links. Anschließend wird noch die Position des Roboters korrigiert, die durch den Ausweichpfad verursacht wurde und der ursprüngliche Algorithmus übernimmt wieder die Steuerung.
2. Bei der Zielführung in y-Richtung richtet sich der Roboter aus und fährt mit einem ähnlichen Algorithmus wie dem für die x-Richtung zum Ziel. Dabei wird der y-Wert statt dem x-Wert alle 20 cm Fahrstrecke erhöht. Sollte hier der Roboter auf ein Hindernis stoßen, so wird der Ausweichpfad gespiegelt zu dem bei der x-Richtung ausgeführt, damit der Roboter nicht aus dem Kartenbereich herausfährt. Er dreht sich also statt nach links nach rechts und umgekehrt.
3. Falls beide Werte der Zielkoordinate ungleich null sind, errechnet der Roboter zunächst den Winkel in dem das Ziel zu ihm liegt und dreht sich dann auf der Stelle Richtung Ziel, um anschließend dorthin zu fahren. Dabei wird immer alle 20 cm Fahrstrecke der x- und y-Werte des Roboters um Eins erhöht. Wenn er sich am Ziel befindet, hält er an und beendet den Algorithmus. Sollte der Roboter auf dem Weg dorthin mit dem Ultraschallsensor ein Hindernis entdecken, so initialisiert er einen Ausweich-pfad, welcher den Roboter zunächst um 45 Grad nach links drehen und dann um 20 cm nach vorn fahren lässt. Danach fährt der Roboter weiter und erhöht seine x- und y-Position alle 20 cm. Der x-Wert der Zielposition wurde um Eins verringert, damit der Roboter nicht am Ziel vorbeifährt. Jedoch endete der Roboter bei den Versuchsläufen hinter dem Ziel, daher dreht er sich nun als Abschlusssequenz um 135 Grad und fährt noch einmal um 20 cm nach vorne, hält an und beendet den Algorithmus.

Nun kann man das gesamte Programm beenden oder dem Roboter neue Koordinaten übergeben oder ihn manuell zur Startposition zurückfahren. Diese Programmstruktur kann auch durch den Programmablaufplan in Abb. 2 im Anhang, nachvollzogen werden.

IV. Endresultate

Nach einigen Tests zeigt sich, dass der Roboter unter Laborbedingungen sein Ziel findet und das Programm danach wieder auf eine Eingabe aus der grafischen Benutzeroberfläche wartet. Dabei kam es jedoch häufig zur Abweichung zwischen der realen und der im Programm ausgeführten Bewegung. Bei Fehlfunktionen der Getriebe, die manchmal blockierten, wenn man sie nicht vor jeder Durchführung kontrollierte oder weil der Untergrund dem Roboter unterschiedliche Haftungen auf dem abgefahrenen Weg verlieh, wodurch die Drehungen und Fahrweiten ungenau wurden, kam es zu Zielverfehlungen bis zu einer ganzen Einheitslänge von 20 cm. Zudem kann die Zielabbruchsbedingung umgangen werden, wenn sich genau auf dem Ziel ein Hindernis befindet, weil so ein Ausweichen eingeleitet wird, welches den Roboter hinter dem Ziel herauskommen lässt. Da er nun nach seinem Algorithmus in der ursprünglichen Zielrichtung weiterfährt, kann er das Ziel sowohl im Programm als auch im Realen nie erreichen, wodurch auch der Algorithmus zum automatischen Fahren nie sein Ende findet.

V. Zusammenfassung und Fazit

In diesem Projekt sollte ein Roboter konstruiert werden, dem ein Ziel übergeben wird, das er dann eigenständig anfährt, ohne dabei mit Hindernissen zu kollidieren oder dass er über eine einfache Steuerung manuell bedient werden kann. Das Grundziel wurde dabei zwar erreicht, jedoch nur bei sehr einfachen und vorbestimmten Hindernisstrukturen. Um den Roboter zu optimieren, kann man einen Bug2-Algorithmus anstelle des Ausweichpfades setzen, um auch individuelle Hindernisse um-

fahren zu können und die Positionen der Hindernisse speichern, um sie späterer Verarbeitung zugänglich zu machen.

ANHANG

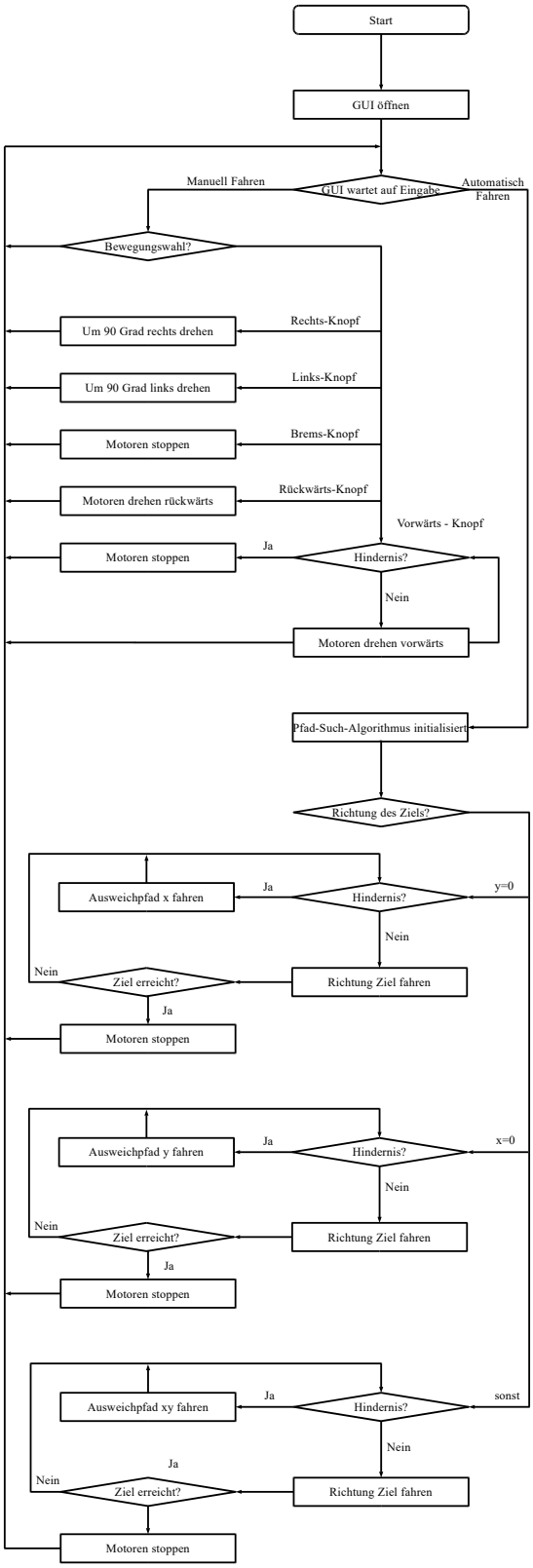

Abbildung 2: Programmablaufplan des Erkundungsroboters

LITERATURVERZEICHNIS

[1] WIKIPEDIA, DIE FREIE ENZYKLOPEDIE: *Lego Mindstorms NTX*. https://de.wikipedia.org/wiki/LEGO_Mindstorms_NXT, Version: 03.03.2020

[2] RWTH AACHEN: *RWTH - Mindstorms Toolbox*. https://www.mindstorms.rwth-aachen.de/trac/wiki/Download4.07, Version: 04.03.2020

[3] HOWIE CHOSET, CARNEGIE MELLON SCHOOL OF COMPUTER SCIENCE*: Robotic Motion Planning Bug Algorithms*. http://www.cs.cmu.edu/~motionplanning/lecture/Chap2-Bug-Alg_howie.pdf, Version: 03.03.2020

[4] WIKIPEDIA, DIE FREIE ENZYKLOPEDIE: *A*-Algorithmus*. https://de.wikipedia.org/wiki/A*-Algorithmus, Version: 05.03.2020

[5] SEBASTIAN LAGUE, YOUTUBE: *A* Pathfinding (E01: algorithm explanation)*. https://www.youtube.com/watch?v=-L-WgKMFuhE, Version: 05.03.2020