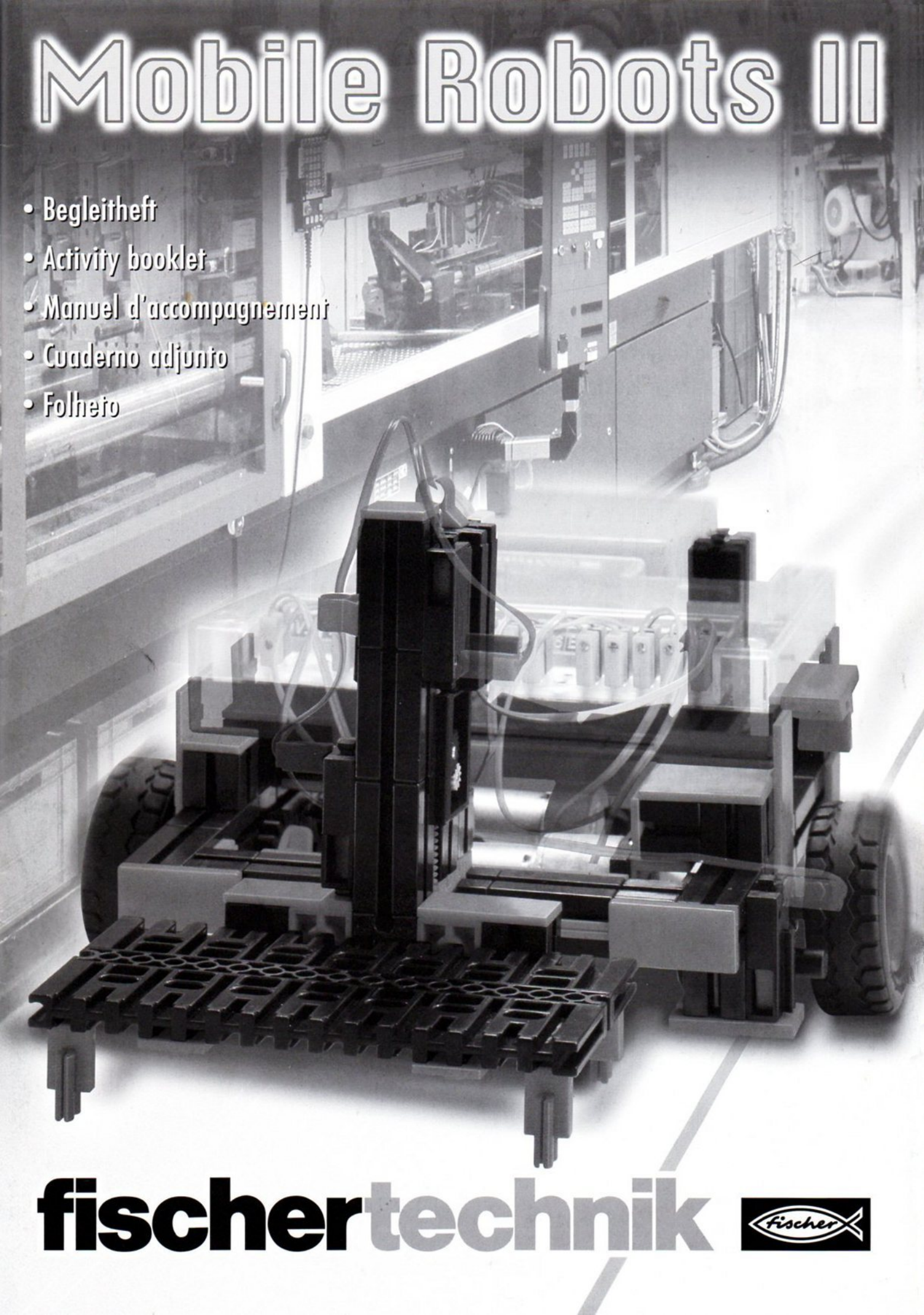
Mobile Robots II

- Begleitheft
- Activity booklet
- Manuel d'accompagnement
- Cuaderno adjunto
- Folheto



fischertechnik

(D) INHALT

D

1 Wozu brauchen wir Roboter ?

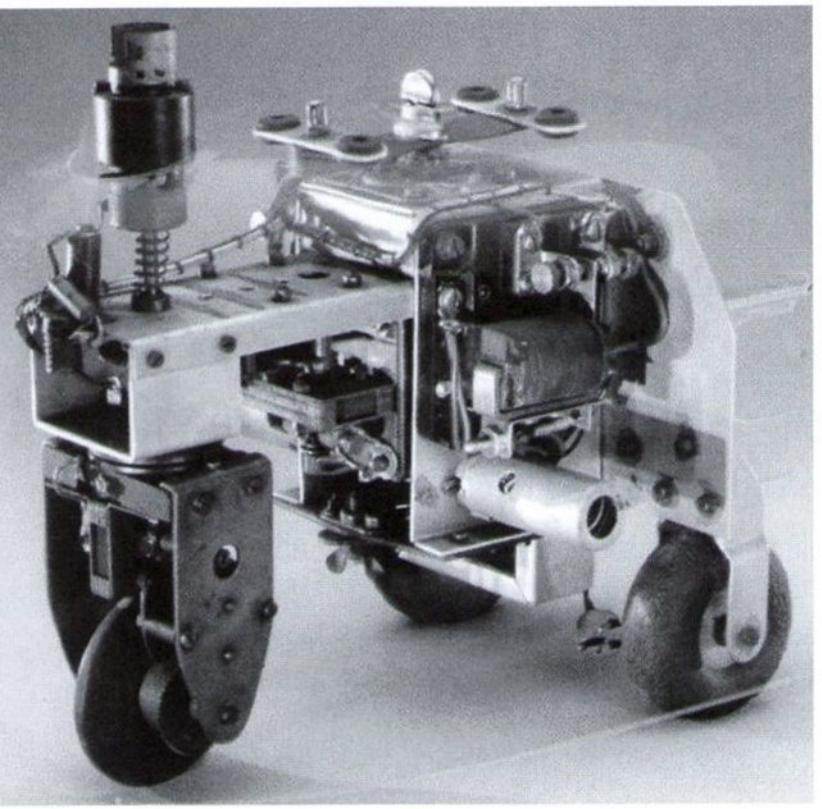
Bevor wir beginnen uns praktisch mit Robotertechnik zu befassen, wollen wir versuchen, die leicht provokativ gestellte Frage in der Überschrift zu beantworten.

Der Begriff des „Roboters" wird erstmals 1923 im Roman „Golem" von Carel Capek verwendet. Diese düstere künstlich erschaffene Figur sollte mit ihren Fähigkeiten menschliche Arbeit ersetzen.

Wie so oft bei literarischen Gestalten ist dies auch hier an Zwänge gebunden und ein gewisses Mißtrauen schlägt der Figur entgegen. In den 30er und 40er Jahren des 20. Jahrhunderts wird aus dem Roboter eher eine Art Automat. Diverse Versuche, ihn mit äußeren menschlichen Eigenschaften zu versehen, z. B. einem Kopf mit blinkenden Lampen als Augen sowie primitiver Sprachausgabe per Lautsprecher, wirken aus heutiger Sicht naiv. Offenbar sind die Befürchtungen einer potentiellen Herrschaft der Roboter über die Menschen nicht so einfach zu widerlegen.

Doch bei diesen ersten einfachen Versuchen ist von einer Mobilität oder gar Intelligenz der konstruierten Maschinen wenig zu bemerken. Erst mit dem Aufkommen elektronischer Schaltungen wurde der Aufbau von Robotern realistisch.

Eng mit der eigentlichen Robotertechnik verknüpft, ist das Problem nach den erforderlichen Steuerprinzipien. Diese Frage nach der „Intelligenz" des Roboters ist auch heute noch Forschungs- und Untersuchungsgegenstand vieler Firmen, Institute und Universitäten.



Erste Lösungsansätze versprach man sich von der Kybernetik. Die Bezeichnung „Kybernetik" ist von dem griechischen Wort Kybernetes abgeleitet. Der Kybernetes war der Navigator auf den griechischen Ruderschiffen. Er mußte den Schiffsort bestimmen und den notwendigen Kurs bis zum Ziel errechnen. Damit ist klar, die Kybernetik sollte den Roboter „intelligent" machen.

Wie kann man sich ein solches intelligentes Verhalten überhaupt vorstellen? Wir wollen versuchen, uns dies mit Hilfe eines Gedankenexperimentes zu verdeutlichen. Jeder wird schon einmal das Verhalten einer Motte im Lichtkreis einer Lampe beobachtet haben. Die Motte erkennt die Lichtquelle, fliegt darauf zu, um dann kurz vor dem Aufprall auf die Lampe auszuweichen. Es ist klar, dass die Motte für dieses Verhalten die Lichtquelle erkennen, einen Weg dahin ermitteln und dann darauf zufliegen muss. Diese Fähigkeiten basieren auf instinktiven, intelligenten Verhaltensmustern des Insekts.

Versuchen wir nun diese Fähigkeiten in ein technisches System zu übertragen. Wir müssen die Lichtquelle erkennen (optische Sensoren), eine Bewegung ausführen (Motoren steuern) und wir müssen einen sinnvollen Zusammenhang zwischen Erkennung und Bewegung aufstellen (das Programm). Unser Gedankenexperiment kombiniert nun einen optischen Sensor mit einem Motor und einer Logik, so dass dieses Fahrzeug den Motor immer in Richtung der Lichtquelle steuert. Dieses Fahrzeug würde sich demnach genau wie eine Motte verhalten, oder nicht?

Eine technische Realisierung des eben geschilderten Experimentes erfolgte in den 50er Jahren durch den Briten Walter Grey. Mit Hilfe einfacher Sensoren, Motoren und elektronischer Schaltungen wurden verschiedene „kybernetische" Tiere geschaffen, denen dann ganz spezifisches Verhalten, wie z. B. das einer Motte, eigen waren.

Diese Maschinen stellen einen wichtigen Schritt auf dem Weg zu modernen mobilen Robotern dar. Die Sensoren (Fotowiderstand, Fühler,...) der Geräte steuerten mit Hilfe ihrer Elektronik die Aktoren (Motoren, Relais, Lampen,...) so dass ein (scheinbar?) intelligentes Verhalten zustande kam. In der Abbildung ist ein Nachbau der „kybernetischen" Schildkröte, welche im Smithsonian Museum, Washington ausgestellt ist, zu sehen.

Basierend auf diesen Überlegungen erstellen wir für unsere Roboter entsprechende „Verhaltensmuster" und versuchen diese in Form von Programmen dem Roboter verständlich zu machen.

Doch wie ist mit diesen Überlegungen die eingangs gestellte Frage nach dem Nutzen von mobilen Robotern zu beantworten? Um diese Frage konkret zu beantworten, versuchen wir nun, die bislang eher abstrakten Verhalten unserer „Gedankenmotte" auf technische Belange anzuwenden. Ein einfaches Beispiel dafür ist die Lichtsuche. Wir modifizieren die Lichtquelle, indem wir einen hellen Streifen, die Leitlinie, auf dem Boden anbringen und die Sensoren nicht mehr nach vorn, sondern nach unten ausrichten. Mit Hilfe derartiger Leitlinien kann sich ein mobiler Roboter beispielsweise in einer Lagerhalle orientieren. Zusätzliche Informationen, z. B. in Form von Strichkode an bestimmten Stellen der Linie, veranlassen den Roboter an diesen Positionen zu weiteren Aktionen, wie z. B. das Aufnehmen oder Absetzen einer Palette. Solche Robotersysteme existieren bereits tatsächlich. In großen Krankenhäusern fallen zum Teil recht lange Transportwege für Verbrauchsmaterialen, wie z. B. Bettwäsche, an. Der Transport dieser Materialen durch das Pflegepersonal ist aufwändig und zum Teil mit schwerer körperlicher Arbeit verbunden. Zudem schränken solche Tätigkeiten die für die Pflege der Patienten bereitstehende Zeit ein.

Wir erkennen also, dass mobile Roboter einen wichtigen Platz in der modernen Gesellschaft einnehmen können. Doch wie hängt dies mit den fischertechnik-Baukästen zusammen?

Für einen Roboter brauchen wir außer den Sensoren und Aktoren viele mechanische Teile um ein Modell zu konstruieren. Der fischertechnik-Baukasten Mobile Robots II ist hierzu eine ideale Grundlage. Wir können die Mechanik-

teile in einer beinahe unerschöpflichen Vielfalt miteinander kombinieren und erhalten robuste Roboterfahrzeuge. Mit dem dazugehörigen „Intelligent Interface" (Art.-Nr. 30402, zusätzlich erforderlich) verfügen wir auch über genug Rechenpower um anspruchsvolle Programme zu entwerfen. Über dieses Interface erfolgt die Ankopplung und Auswertung einer Vielzahl verschiedener Sensoren und Aktoren.
Die Sensoren wandeln physikalische Messgrößen, wie Lichtmenge oder Temperatur, in elektrisch erfassbare Werte um. Dabei gibt es sowohl analoge auch digitale Messgrößen. Unter digitalen Größen verstehen wir solche, die entweder logisch wahr oder logisch falsch sein können. Diese Zustände werden mit 0 bzw. 1 gekennzeichnet. Ein Schalter ist ein gutes Beispiel für einen digitalen Sensor.

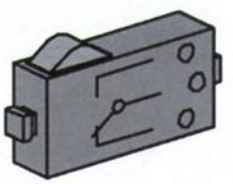
Viele Messgrößen ändern sich jedoch kontinuierlich zwischen ihren Extremwerten, diese nennt man analoge Werte. Sie können nicht einfach als 0 oder 1 dargestellt werden. Damit diese Größen von einem Computer verarbeitet werden können, müssen sie in entsprechende Zahlenwerte umgewandelt werden. Das fischertechnik-Interface stellt hierzu zwei Analogeingänge EX und EY bereit. Der an diesen Klemmen angelegte Widerstandswert wird in einem Zahlenwert gespeichert. Die Messwerte eines Temperatursensors, z. B. 0...5 kΩ, werden somit in einem Bereich von 0...1024 erfasst und stehen für eine nachfolgende Bearbeitung zur Verfügung.
Die wichtigste Funktion des Interfaces besteht in der logischen Verknüpfung der Eingangsgrößen. Dazu benötigt das Interface ein Programm. Das Programm entscheidet in welcher Weise aus Eingangsdaten, den Sensorsignalen, passende Ausgangsdaten, Motorsteuersignale etc., entstehen.
Damit wir möglichst effektiv die für das Interface notwendigen Programme erstellen können, gibt es eine grafische Programmieroberfläche. Hinter dem Begriff „Programmieroberfläche" verbirgt sich eine Software, die es uns ermöglicht, auf sehr hohem Niveau unsere Programme zu erstellen. Wir können nämlich mit Hilfe grafischer Symbole Programme bzw. Algorithmen entwerfen. Der Computer des Intelligent Interface kann eigentlich nur Befehle aus seinem sogenannten Maschinenbefehlssatz ausführen. Das sind im Wesentlichen einfache logische bzw. arithmetische Kontrollstrukturen, deren Anwendungen für Einsteiger außerordentlich schwierig sind. Die PC-Software LLWin (Art.-Nr. 30407, nicht im Baukasten enthalten) stellt deswegen Grafikelemente bereit, die anschließend in eine für das Interface ausführbare Sprache übersetzt werden.

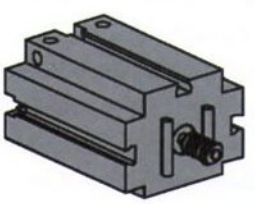
Wir werden bei unserem Eindringen in die faszinierende Welt der mobilen Roboter schrittweise vorgehen. Wir beginnen mit einem einfachen Testaufbau zur Prüfung der Grundfunktionen von Interface und Sensorik. Danach starten wir mit einfachen Modellen denen bestimmte Aufgaben zugeordnet sind und versuchen uns dann mit immer komplizierteren Systemen. Damit auftretende Fehler nicht zu permanentem Verdruss führen, werden wir uns in einem Kapitel mit Eigenschaften und Besonderheiten von Sensoren und Aktoren vertraut machen und für ganz „hartnäckige" Fälle gibt es am Ende einen Abschnitt „Fehlersuche und Behebung".

Ein sehr wichtiger Punkt ist die Sorgfalt beim Aufbau und der Inbetriebnahme unserer Roboter. Wir haben es mit komplexen Maschinen zu tun, deren einziger Unterschied zu wirklichen Robotersystemen ihre vergleichsweise geringe Größe ist. Beim Aufbau der elektrischen Komponenten halten wir uns eng an die Vorgaben und überprüfen lieber doppelt oder dreifach, ob alles stimmt. Bei den mechanischen Konstruktionen, auch bei eigenen Kreationen, achten wir sehr auf Leichtgängigkeit und Spielarmut in den Getrieben und Befestigungen. Nirgendwo müssen wir beim Zusammenbau „Gewalt" anwenden. Es ist unserer Kreativität vorbehalten, neue Programme zu schreiben und damit neue „Verhalten" zu definieren, deren Komplexität nur durch die zur Verfügung stehenden Ressourcen an Speicherkapazität und Rechenleistung begrenzt ist. Die folgenden Beispiele geben dazu einige Anregungen.

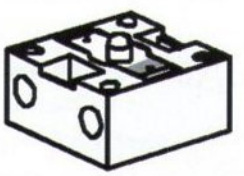
2 Erste Schritte

Nach den theoretischen Überlegungen wollen wir nun endlich beginnen eigene Experimente durchzuführen. Sicherlich möchte der eine oder andere sofort starten, vielleicht sogar mit dem komplizierten automatischen Gabelstapler. Das ist natürlich möglich und bei sorgfältiger Beachtung der Bauanleitung gelingt der Aufbau des Modells auf Anhieb.
Doch was tun, wenn es nicht funktioniert? In diesen Fällen muß systematisch nach der Fehlerursache gesucht werden. Ganz zwangsläufig tauchen dabei Fragen zur Funktionsweise und den Eigenschaften der verwendeten Komponenten auf. Offenbar ist ein gewisses Maß an Grundlagenwissen zu Sensoren und Aktoren unerlässlich. Bevor wir jedoch beginnen, uns mit diesen Dingen vertraut zu machen, prüfen wir das Zusammenspiel von Computer und Interface.
Entsprechend den Vorgaben aus dem LLWin-Handbuch wird die Steuersoftware auf dem PC installiert.
Mit Hilfe der Interface-Diagnose testen wir die unterschiedlichen Sensoren bzw. Aktoren. Wir können jetzt z. B. einen Taster mit zwei Leitungen an den Eingang E1 anschließen und sehen dann, welcher logische Schaltzustand vom Interface erkannt wird. Eine Betätigung des Tasters muss zu einer Zustandsänderung am entsprechenden Eingang führen.



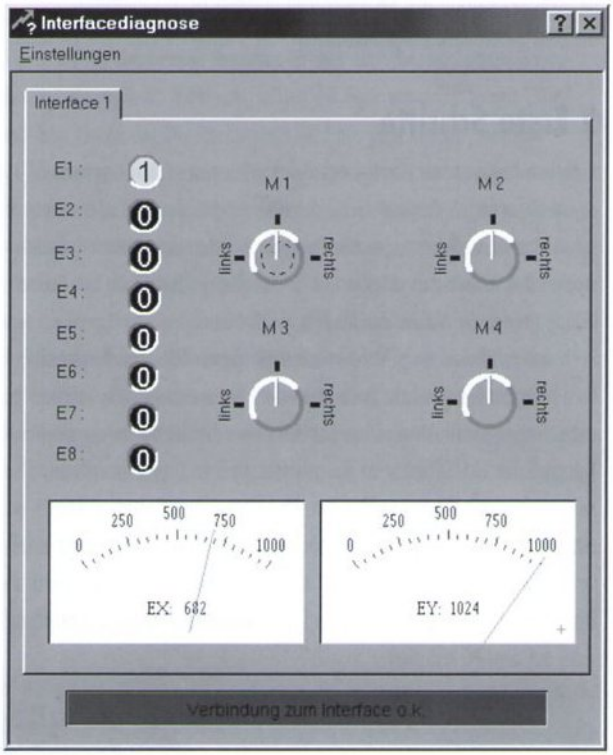


Die Ausgänge überprüfen wir, indem wir einen Motor mit einem Motorausgang, z. B. M1, verbinden. Mit der Maus gelingt es uns, den Motor in Drehung zu versetzen. Wollen wir auch den Analogeingang testen, so funktioniert dies, indem wir einen Fototransistor als Analogsensor verwenden. Während beim Motor bzw. Taster die Polarität der Anschlüsse keine Rolle spielt (der Motor dreht sich im ungünstigsten Fall verkehrt herum), ist der richtige Anschluss des Fototransistors für die korrekte Funktion zwingend notwendig.



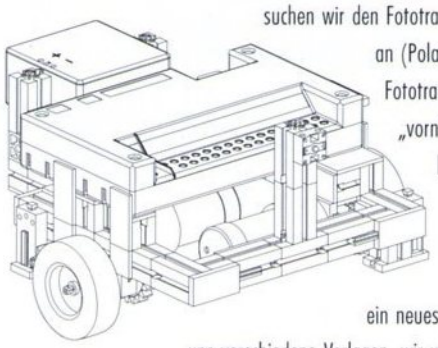
Ein Kontakt des Transistors ist mit einer roten Markierung versehen, diesen verbinden wir mit einem roten Anschlußstecker, den nicht gekennzeichneten Kontakt mit einem grünen Stecker.
Der zweite rote Stecker kommt in die näher am Rand des Interfaces liegende Buchse des Eingangs EX, der zweite grüne Stecker passt in die weiter innen liegende Buchse von EX. Nun können wir mit Hilfe einer Taschenlampe die Beleuchtungsstärke des Fototransistors variieren und damit den Zeigerausschlag verändern.

Sollte der Zeiger sich nicht von seinem Maximalausschlag wegbewegen, schauen wir nochmals nach den Anschlüssen des Fototransistors. Ist hingegen auch bei ausgeschalteter Taschenlampe der Zeiger auf Null, so kann es sein, dass die Beleuchtung im Raum, die Umgebungshelligkeit, zu groß ist. Der Zeigerausschlag ändert sich dann, wenn wir den Fototransistor abdecken.

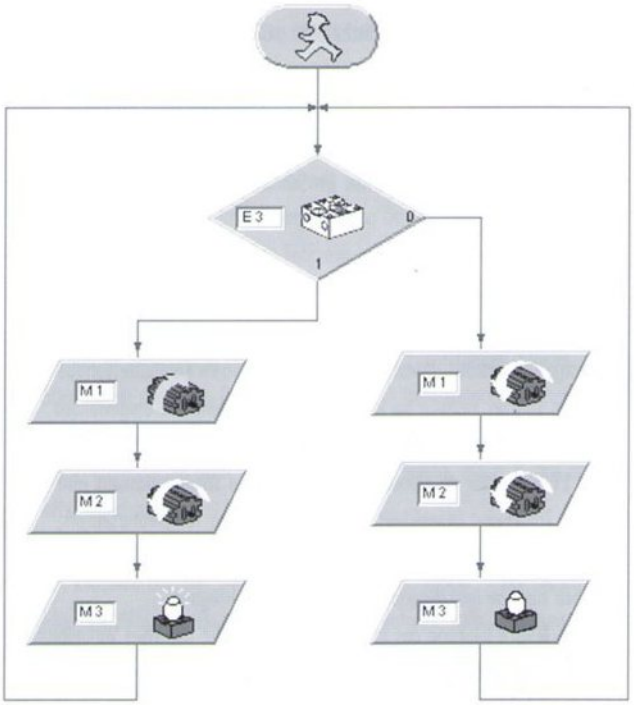


Um nochmals kurz auf die Farbzuordnung der Stecker zu kommen: Wir achten streng darauf beim Zusammenbau immer einen roten Stecker an die rote Leitung anzuschließen und einen grünen Stecker mit der grünen Leitung zu verbinden. Wenn wir in einem Schaltungsaufbau polarisierte Signale verwenden, nehmen wir immer eine rote Leitung für den Pluspol und eine grüne Leitung für den Minuspol. Das mag ein wenig pedantisch erscheinen (und es ist dem Strom auch egal welche Farbe eine Leitung hat), aber für eine systematische Fehlersuche ist eine eindeutige Farbzuordnung eine erhebliche Erleichterung.

Mit einem einfachen Programm wollen wir unsere ersten Schritte auf dem Gebiet der Robotik abschließen. Wir bauen das Basismodell mit den beiden Antriebsmotoren und dem Stützrad auf entsprechend der Bauanleitung auf. Wir schließen nur die Motoren an die Ausgänge M1 und M2 an. Außerdem suchen wir den Fototransistor und schließen ihn an den Eingang E3 an (Polarität beachten). Zuvor befestigen wir den Fototransistor so am Grundmodell, dass er nach „vorn" schaut. Wer will, kann noch die Linsenlampe an M3 anschließen und sie z. B. neben dem Akkupack befestigen, aber das ist nicht unbedingt notwendig. Wir öffnen das Programm LLWin und legen ein neues Projekt an (PROJEKT – NEU). LLWin bietet uns verschiedene Vorlagen, wir wählen „leeres Projekt" und geben ihm einen Namen, z. B. „Step1". Nachdem wir den [OK]-Button gedrückt haben, erscheint ein leeres Arbeitsblatt mit einem Ampelmännchen und dem Bausteinfenster. Das grüne Ampelmännchen symbolisiert den Programmstart.



Von diesem Startpunkt aus beginnen alle unsere Programme. Die verschiedenen Programmteile holen wir mit der Maus aus dem Bausteinfenster. Die dort vorhandenen Symbole stehen für die Ein-/Ausgänge am Interface. Mit der linken Maustaste können wir das gewünschte Symbol platzieren und mit der rechten Maustaste ändern wir die Eigenschaften.



Das Tastersymbol kennzeichnet einen Eingang. Für unser Programm platzieren wir mit der Maus den Taster unter das Startsymbol. Wenn wir das Symbol loslassen, erscheint ein Auswahldialog. Wir wählen den Fototransistor aus. Wenn wir später weitere Änderungen wünschen, können wir diesen Dialog mit der rechten Maustaste aktivieren. Bei den Motoren ordnen wir die Ausgänge zu und weisen die gewünschte Drehrichtung an. Wir wollen, dass die Motoren gleichsinnig drehen, wenn kein Licht auf den Fototransistor fällt und gegenläufig, wenn Licht erkannt wird. Dann werden die Elemente mit der Zeichenfunktion verbunden. Die Lampe an M3 signalisiert den Zustand des Fototransistors.

Die Abbildung zeigt die genaue Verbindung der Programmzweige. Wer sich nicht sicher ist, ob alles richtig ist, vergleicht sein Programm mit dem Programm Step1.mdl. Dazu wird das eigene Programm vorher gespeichert und die Datei Step1.mdl von der CD-ROM geladen, die im Baukasten enthalten ist. Ist alles in Ordnung, wird das Programm per Download in das Interface geladen und gleich gestartet (RUN - DOWNLOAD).

Unser erster Roboter dreht sich jetzt auf der Stelle. Er tut dies solange, bis wir ihn mit einer Lichtquelle anlocken. Sobald der Fototransistor das Licht erkennt, fährt der Roboter geradeaus auf die Lichtquelle zu. Falls er sich von der Lichtquelle entfernt, müssen wir beide Motoren umpolen. Wahrscheinlich wird seine Bahn nicht exakt geradeaus zeigen, so dass nach einiger Wegstrecke der Fototransistor den Kontakt zur Lichtquelle verliert. Daraufhin wird seine Bewegung von Fahrt voraus auf Drehung umgeschaltet und die Lichtsuche beginnt erneut. Damit wir erfolgreich experimentieren, haben wir vorher entsprechend Platz geschaffen, denn leider kann unser Roboter Hindernisse auf seinem Weg (noch) nicht wahrnehmen.

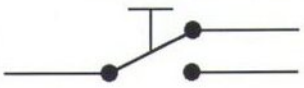
3 Sensoren und Aktoren

Wir wissen nun, dass unsere wichtigste Baugruppe, das Interface, zusammen mit dem PC „spielt". Jetzt geht es darum, wie die Signale aus der Umwelt von unserem Interface erkannt werden können.

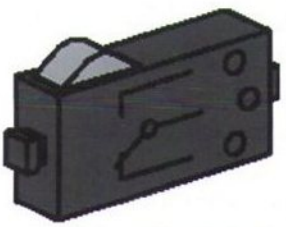
Beginnen wir mit den Eingängen. In der Fachsprache werden Eingänge bzw. Eingangssignale oft als Input bezeichnet. Für einen Computer, und um einen solchen handelt es sich beim Intelligent Interface, gibt es nur die Möglichkeit elektrische Signale zu erkennen und zu verarbeiten. Wir müssen die Umweltreize also „computertauglich" machen. Alle Sensoren sind deshalb Umsetzer, die den gewünschten „Sinn" in ein elektrisches Signal wandeln. Da wir nicht ausschließlich Bauanleitungen „blind" nachbauen wollen, ist es sinnvoll, sich mit den grundlegenden Eigenschaften der vorhandenen Sensoren zu befassen.

Dies ist um so wichtiger, da das Interface später durchaus um neue, selbst definierte Anwendungen ergänzt werden kann.

3.1 Der Schalter als digitaler Sensor

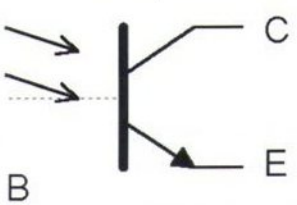


Die einfachen logischen Pegel „0" und „1" sind mit Hilfe eines Schalters darstellbar. Im System der fischertechnik-Baukästen werden Präzisionssprungschalter mit Wechselkontakten verwendet.



Die Besonderheit der Sprungschalter liegt in ihrem Schaltverhalten. Bei vorsichtigem langsamen Betätigen des roten Schaltknopfs spüren wir einen deutlichen Druckpunkt, bei dessen Überschreiten der Schaltkontakt mit einem leisen Knackgeräusch umschaltet. Lassen wir den Schalthebel nun wieder langsam los, müssen wir den Hebel deutlich weiter über den ursprünglichen Einschaltpunkt „herauslassen", um ein Zurückschalten zu erreichen. Diesen Unterschied zwischen mechanischer Einschalt- und Ausschaltposition wird als Hysterese bezeichnet. Die Schalthysterese von Kontakten oder anderen elektronischen Schaltungen ist eine wichtige Eigenschaft. Gäbe es sie nicht, d.h. Einschaltpunkt wäre gleich Ausschaltpunkt, wären große Probleme in der Signalauswertung die Folge. Winzige Störungen, wie ein ganz leichtes Zittern im Umschaltzeitpunkt, würde dem Interface mehrere Kontaktbetätigungen „vorgaukeln", ein exaktes Zählen von Ereignissen wäre nicht möglich. Der Schalter ist als Wechselschalter ausgeführt. Wir können somit beide denkbaren Ausgangslagen, d.h. im Ruhezustand geschlossen bzw. im Ruhezustand geöffnet, bei unseren Experimenten auswerten.

3.2 Lichterkennung mit dem Fototransistor



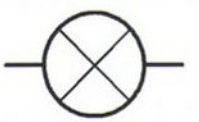
Beim Fototransistor handelt es sich um ein Halbleiterbauelement, dessen elektrische Eigenschaften lichtstärkeabhängig sind. Ein gewöhnlicher Transistor ist ein Bauelement mit drei Anschlüssen. Diese Anschlüsse werden als Emitter, Basis und Collektor bezeichnet. Seine Hauptaufgabe ist die Verstärkung schwacher Signale. Ein schwacher Strom, der von einem Signal in die Basis des Transistors fließt, hat einen viel stärkeren Strom am Collektor des Transistors zur Folge. Die Stromverstärkung kann Faktoren von über 1000 erreichen. Der Fototransistor aus dem Baukasten besitzt jedoch nur zwei Anschlüsse. Wo ist der dritte Anschluß geblieben?

Wir wollen mit unserem Transistor Licht erkennen. Jeder kennt Solarzellen, mit denen aus Sonnenlicht Strom gewonnen wird. Der Fototransistor ist als Kombination von Minisolarzelle und Transistor zu verstehen. Der Basisanschluss wird nicht nach außen geführt (deshalb in der Abbildung gestrichelt). An seiner Stelle erzeugen auftreffende Lichtimpule (Photonen) einen sehr kleinen Fotostrom, der dann vom Transistor verstärkt am Collektor zur Auswertung bereitsteht. Damit dies so wie beschrieben funktioniert, benötigt der Fototransistor eine zusätzliche Außenbeschaltung. Da diese im Interface enthalten ist, braucht sie uns nicht weiter zu interessieren.

Der Fototransistor kann sowohl als digitaler, als auch als analoger Sensor verwendet werden. Im ersten Fall dient er zur Erkennung deutlicher Hell-Dunkel-Übergänge, z. B. einer markierten Linie. Es können jedoch auch Lichtmengen in ihrer Stärke unterschieden werden, dann arbeitet der Fototransistor als analoger Sensor.

3.3 Signalausgabe mit der Glühlampe

Zur Ausgabe einfacher Lichtsignale dient die Glühlampe. Um bei unseren Fachbezeichnungen zu bleiben; die Glühlampe wird zum optischen Aktor. Der Aufbau einer Glühlampe ist recht simpel. In einem Glaskolben, in dem sich ein Vakuum befindet, ist eine Wendel aus dünnem Wolframdraht zwischen zwei Anschlußstiften aufgespannt. Fließt Strom durch die Wendel, erhitzt sich der Draht bis zur Weißglut. Da kein Sauerstoff im Glaskolben vorhanden ist, verbrennt der Draht nicht und die Lampe erreicht somit eine hohe Lebensdauer. Infolge der starken thermischen Beanspruchung der Glühwendel dehnt sich die Drahtwendel bei jedem Einschalten aus und zieht sich beim Ausschalten wieder zusammen. Diese minimalen Bewegungen führen dann infolge Materialermüdung irgendwann zum „Durchbrennen" der Glühlampe.



Eine Einsatzmöglichkeit der Glühlampe ist das Anzeigen von Schaltzuständen. Durch das Programmieren einer blinkenden Lampe können auch Warnmeldungen erzeugt werden.

Wir brauchen die Lampe noch in einem weiteren Fall. Zusammen mit zwei Fototransistoren entsteht ein Spezialsensor, mit dessen Eigenschaften Linien erkannt werden. Die Lampe arbeitet als Lichtquelle, damit der Fototransistor anhand des unterscheidlich stark reflektierten Lichtes eine Farbmarkierung erkennt.

Eine Besonderheit der im fischertechnik-Baukasten zu Einsatz kommenden Glühlampe ist die im Glaskolben enthaltene optische Linse. Damit wird der Lichtstrahl besser gebündelt und wir erkennen z. B. Markierungen zuverlässiger.

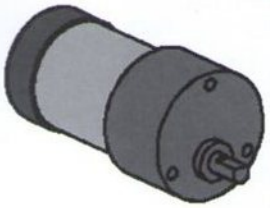
3.4 Gleichstrommotoren als Kraftquelle

Gleichstrommotoren sind wichtige Aktoren für mobile Systeme. Im Baukasten „Mobile Robots II" sind zwei verschiedene Typen von Motoren enthalten. Mechanisch recht unterschiedlich, ist ihr elektrischer Aufbau jedoch identisch.

D

Gleichstrommotoren bestehen aus einem sich drehenden „Rotor" und einem fest stehenden „Stator". Der Rotor ist vom Prinzip her als eine Leiterschleife zu verstehen, die sich im magnetischen Feld des Stators befindet. Fließt nun

ein Strom durch die Leiterschleife entsteht eine Kraft, die zur Auslenkung des Leiters im Magnetfeld führt; der Rotor bewegt sich. Für praktische Anwendungen ist die einfache Leiterschleife als Spule (mit oder ohne Eisenkern zur Verstärkung des Magnetfeldes) ausgeführt. Sehr viele Gleichstrommotoren erzeugen das notwendige Magnetfeld mit Hilfe von Dauermagneten, die in den Metallmantel des Stators eingeklebt sind. Die Stromzufuhr zum drehenden Rotor erfolgt mit Hilfe von Schleifkontakten. Diese Kontakte sorgen gleichzeitig für die Stromrichtungssumkehr in der Leiterschleife, die für eine ununterbrochene Drehbewegung notwendig ist.

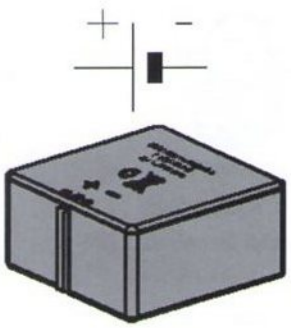


Die Drehzahl üblicher Motoren bewegt sich im Bereich von einigen tausend Umdrehungen pro Minute. Ein Getriebe sorgt für niedrigere Drehzahlen bei größerem Drehmoment.

Der Baukasten enthält zwei verschiedene Motortypen, den Minimotor und den Powermotor. Der kleine kompakte Minimotor mit Schnecke ist für Hilfsantriebe oder Sonderzwecke mit geringen Leistungsanforderungen gedacht. Er benötigt immer ein Getriebe zur Drehzahlreduzierung.

Der Power-Motor stellt viel größere Drehmomente zur Verfügung. Er hat ein fest angeflanschtes Getriebe mit einer Untersetzung von 50:1. Damit ist er ideal für unsere Antriebsanforderungen beim Roboter geeignet. Es gibt ihn auch in einer Variante mit Untersetzung 8:1 (nicht in diesem Baukasten enthalten). Hier wäre die Drehzahl auf der Abtriebswelle jedoch für den Roboterantrieb zu groß.

3.5 Stromversorgung



Mobile Systeme benötigen eine autonome Stromversorgung. Von dieser Energiequelle werden alle Verbraucher gespeist. Die Anforderungen an die Stromversorgung sind unterschiedlich. Während Antriebsmotoren sich mit unstabilisierten Spannungen zufrieden geben, benötigen viele Sensoren stabile Spannungen um exakte Ergebnisse liefern zu können.

Aus wirtschaftlichen Gründen ist der Einsatz von Batterien oder Akkus der einzig sinnvolle Weg, mobile Roboter mit Strom zu versorgen. Solarzellen oder Brennstoffzellen sind leider noch nicht leistungsfähig genug, um bei angemessenem Aufwand praktikable Ergebnisse zu liefern.

Akkus sind Batterien vorzuziehen, weil sie vielfach wieder aufgeladen werden können. Einen guten Kompromiss zwischen Energieinhalt und Größe stellt der fischertechnik-Akku-Pack dar. Der Akku-Pack ist nicht Bestandteil des Baukastens und kann zusammen mit einem speziellen Ladegerät als „Accu Set" unter der Art.-Nr. 34969 erworben werden. In der Abbildung ist das Schaltzeichen und der Akku dargestellt. Im Normalfall wird im Schaltzeichen die Polarität nicht angegeben. Mit Hilfe einer Eselsbrücke kann man sich leichter merken, welcher Anschluss Plus ist: „Den langen Strich kann man durchschneiden und zu einem Plus zusammensetzen."

Es ist sehr wichtig, beim Anschluss der Spannungsquelle an das Interface immer auf die richtige Polarität zu achten.

3.6 Zusätzliche Sensoren

Das fischertechnik-System ist relativ einfach um weitere Sensoren zu ergänzen. Im einfachsten Fall verwenden wir Sensoren aus anderen Kästen, z. B. den Wärmesensor oder den Magnetsensor aus dem Baukasten „Profi Sensoric" Art.-Nr. 30491.

Wir können jedoch auch völlig andere Sensoren verwenden. Im Fachhandel werden die unterschiedlichsten Bausätze und Komponenten angeboten. Selbst solch exotische Sensoren wie Gasmelder oder Radarsensoren sind verwendbar. Da wir das Intelligent Interface keinesfalls mit zu hohen Eingangspannungen oder falschen Lasten zerstören wollen, sollten jedoch nur erfahrene Bastler eigene Lösungen realisieren. Der sicherste Weg zum Anschluss weiterer Sensoren ist die galvanische Trennung zwischen Sensor und Interface. Eine Reihe von Sensoren besitzen ein Relais, das sich hierzu gut eignet. Die Schaltkontakte des Relais werden wie ein gewöhnlicher fischertechnik-Schalter angeschlossen und signalisieren nun das Auftreten neuer Umweltreize.

Ein Tipp: im Internet werden viele derartige Erweiterungen von begeisterten „Fischertechnikern" veröffentlicht.

4 Die Robotermodelle

Mit den folgenden Aufbauvorschlägen werden einige Varianten mobiler autonomer Roboter vorgestellt. Wir beginnen mit einem einfachen Modell. Darauf aufbauend erkennst und erprobst du den Einsatz unterschiedlicher Sensoren. Dabei kommt es darauf an, sowohl interne Zustände des Roboters, z. B. Wegstreckenmessung durch Impulsräder, als auch externe Umweltsignale, wie Licht- oder Spurensuche zu verknüpfen. Zu jedem Modell werden dazu bestimmte Aufgaben gestellt. Sie sollen als Anregung dienen und dich mit der Materie vertraut machen. Die LLWin-Programme zu den einzelnen Aufgaben befinden sich auf der CD-ROM, die im Baukasten enthalten ist. Lass dir zu den Modellen aber auch eigene Aufgaben einfallen.

Das einfachste Modell ist das Basismodell. Hier werden die Antriebsmotoren mit dem Interface zu einer kompakten Einheit zusammengebaut. Zwei Motoren sorgen für die Antriebskraft des Roboters. Sie sind so gegenüber angeordnet, dass jeder Motor auf ein Antriebsrad wirkt. Damit dieser Roboter nicht umkippt, sorgt ein Stützrad für Stabilität. Eine derartige Anordnung der Motoren wird als „differential drive" bezeichnet. Sie gewährt die höchste Beweglichkeit mit minimal notwendigem Bewegungsraum. Es sind sogar Drehungen auf der Stelle möglich. Der Mittelpunkt beider Motoren ist dabei der Drehpunkt, um den sich der Roboter bewegt. Auf diese Weise gelingt es ihm unter schwierigsten Verhältnissen mit wenig Rechenaufwand zu navigieren.

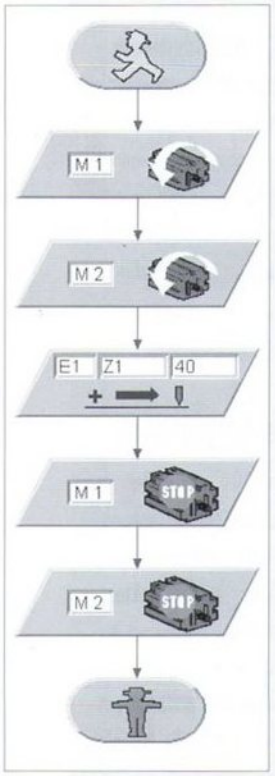
Die Motoren können über zwei unterschiedliche Getriebeuntersetzungen die Räder antreiben (langsam 100 : 1 bzw. schnell 50 : 1). Für die langsamere Variante wird der Antrieb zusätzlich mit fischertechnik-Zahnrädern im Verhältnis 2:1 untersetzt. Bei den Modellen ist jeweils angegeben, welche Untersetzung verwendet wird.

4.1 Das Basismodell

Zuerst bauen wir das Basismodell (Getriebeuntersetzung 100:1) entsprechend der Bauanleitung auf. Da uns dieses Modell bei vielen Experimenten als Grundlage dient, gehen wir beim Aufbau besonders sorgfältig vor. Wenn mechanisch alles fertig ist, prüfen wir die Leichtgängigkeit der Motoren. Dazu wird jeder Motor kurz ohne das Interface direkt mit dem Akku verbunden.

Aufgabe 1:

Programmiere das Interface so, dass das Modell 40 Impulse geradeaus fährt. Zur Messung der Impulse verwende den Zähltaster E1, als Resetschalter den Taster an E8.



Modifiziere danach das Programm so, dass das Modell unterschiedlich lange Strecken, z. B. 80 cm lang, abfährt. Wie groß ist dabei die Wiederholgenauigkeit?

Aufgabe 2:

Das Modell soll sich nach den 80 Impulsen Geradeausfahrt um 180° drehen. Beachte die unterschiedlichen Drehrichtungen der Antriebsmotoren bei Geradeausbewegung bzw. Drehung.

Lösung:

Im Mittel legt das Modell pro Zählimpuls 1 cm Wegstrecke zurück. Die Wiederholgenauigkeit liegt in der gleichen Größenordnung, etwa 1cm bei 80 Impulsen.
Sie schwankt jedoch in Abhängigkeit vom Untergrund, auf dem sich der Roboter bewegt. Besonders ungünstig sind dicke oder flauschige textile Bodenbeläge.

Bevor wir uns dieser Aufgabe zuwenden, wollen wir zwei Dinge klären.
Zum einen haben wir einen neuen Funktionsbaustein in unserem Programm verwendet, den Baustein POSITION. Hier handelt es sich um einen Baustein, der genauso lange aktiv bleibt, bis die eingestellte Anzahl von Impulsen am festgelegten Eingang (hier E1) erkannt wurde. Aus Sicht des Programms bedeutet dies, dass wir hier eine definierte Wartebedingung einsetzen.
Im ersten Versuch haben wir diese Funktion als Wegstreckenmessung zur Geradeausfahrt verwendet.
Soll sich der Roboter drehen, ist dies praktisch derselbe Ansatz, wir müssen nur die Drehrichtung der Motoren ändern. Nun brauchen wir nur noch die Anzahl der Impulse einzutragen und unser Roboter dreht sich auf der Stelle.
Und nun unser zweiter Punkt. Wir wollen nicht einfach solange probieren, bis der Roboter sich um 180° dreht; wir wollen vorher diesen Wert ausrechnen.
Die Antriebsmotoren sind als differential drive konfiguriert, d.h. die Räder des Roboters bewegen sich bei der Drehung auf dem Umfang eines Kreises, dessen Durchmesser vom Abstand der Räder bestimmt wird. Für eine Drehung um 180° muss jedes Rad deshalb eine Strecke von genau der Hälfte dieses Kreisumfangs zurücklegen.
Wir berechnen zuerst den Kreisumfang u:

$$u = \pi \cdot d = 630mm$$

d : Durchmesser (Radabstand ca. 200 mm)

Vorhin haben wir eine Wegstrecke von ca. 1 cm/Impuls ermittelt, damit brauchen wir für die 314 mm Wegstrecke (halber Umfang) 30,5 Impulse. Da wir nur ganzzahlige Werte berechnen können, müssen wir uns für 30 oder 31 Impulse entscheiden. Wir testen welcher tatsächliche Wert die größere Genauigkeit liefert.

Fazit:

Im Ergebnis unserer Messungen mit dem Impulsrad stellen wir fest, dass die erzielbare Genauigkeit unserer Messungen nicht allzuhoch ist. Insbesondere wenn mehrere Stecken hintereinander bzw. wiederholend abgefahren werden, summiert sich der absolute Messfehler auf. Genauso problematisch ist der Fehler, der durch noch nicht vollständig erfasste Takte zustande kommt.
Die Möglichkeiten diese Fehler zu minimieren sind begrenzt. Zum Einen kann man die Wegimpulse pro Streckeneinheit erhöhen. Idealerweise würde der Zähler direkt auf der Motorwelle sitzen. Neben der Tatsache, dass wir nicht an diese Welle herankommen, tritt hier das Problem der begrenzten Abtastrate des Interface auf. Wenn innerhalb einer Zeiteinheit zuviele Impule kommen, „vergisst" das Interface eventuell einige. Damit wird eine genauere Wegberechnung illusorisch.
Andere Fehler, wie z. B. den Schlupf der Räder auf verschiedenen Untergründen oder abweichende Raddurchmesser, können wir überhaupt nicht erfassen. Wir trösten uns mit dem Gedanken, dass diese Probleme teilweise auch von wesentlich komplexeren und teureren kommerziellen Systemen nur ungenügend gelöst sind.

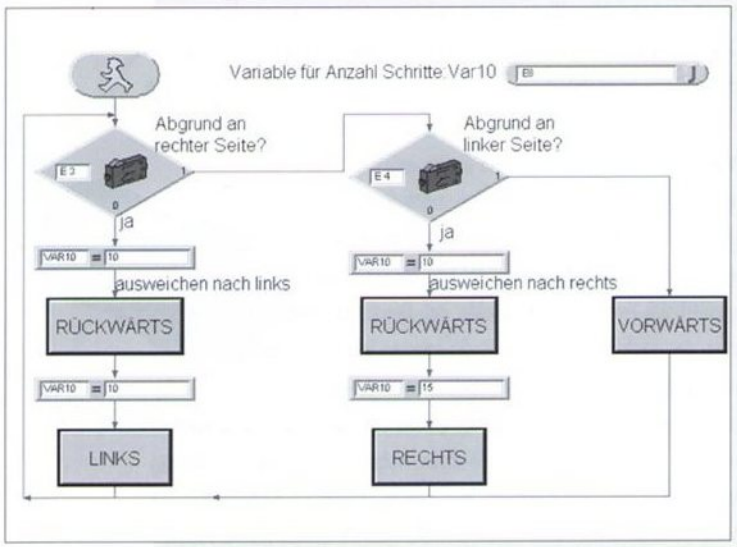
4.2 Roboter mit Kantenerkennung

Nachdem wir nun unser Basismodell entsprechend ausgiebig untersucht haben, wollen wir nun versuchen, dem Roboter die „Angst vorm Abgrund" beizubringen. Bisher ist der Roboter auf der Tischplatte umhergewuselt, von uns mit Argusaugen beobachtet, damit er ja nicht von der Platte stürzt. Dies scheint nun wahrlich kein besonders intelligentes Verhalten. Wir wollen es deswegen ändern.
Damit der Roboter eine Kante erkennt, benötigt er dazu einen Kantendetektor. Ein ebenso einfaches wie brauchbares Verfahren bedient sich zweier Hilfsräder. Diese werden ähnlich einem Fühler in Fahrtrichtung vor dem Roboter mit einem Schalter versehen. Die Räder sind so konstruiert, dass sie sich vertikal bewegen können. Eine Kante lässt das Hilfsrad nach unten fallen und löst somit den Sensor aus.

Aufgabe 3:

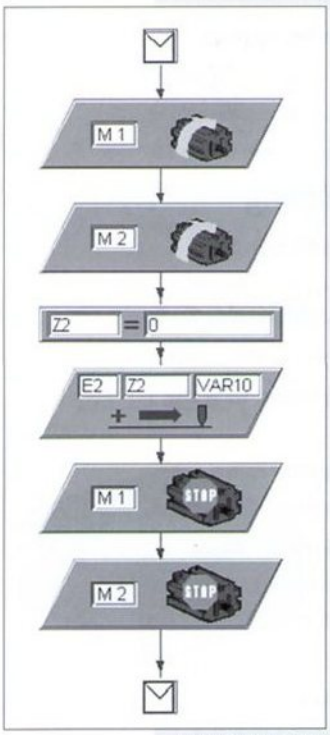
Baue das Modell „Roboter mit Kantenerkennung" entsprechend der Bauanleitung auf (Getriebeuntersetzung 50:1). Das Modell soll geradeaus fahren. Sobald es links an einen Abgrund kommt, soll es nach rechts ausweichen, ist rechts ein Abgrund, soll es nach links ausweichen. Zur besseren Übersicht werden bestimmte Bewegungen als Unterprogramme (Vorwärts, Links und Rechts) eingesetzt.
Die Schrittzahl wird über Zähltaster erkannt. Die Anzahl der Schritte wird in einer Variablen VAR10 vorgegeben. Diese Vorgabe ist für die Unterprogramme Links und Rechts verschieden. Die unterschiedlichen

Vorgabenwerte vermindern das Risiko in einer Ecke steckenzubleiben.



Lösung:

Aus der Aufgabenstellung erkennen wir, dass wir den Roboter in Abhängigkeit von den Kantendetektoren steuern müssen. Wir zerlegen die Aufgabe deshalb in kleinere Teile. Zuerst fragen wir die Taster (Kantendetektoren) ab. Ist kein Taster aktiv, fährt der Roboter vorwärts. In der Abbildung ist dies als Block „Vorwärts" zu sehen. Hinter einem solchen für uns neuen Baustein verbirgt sich ein Unterprogramm. Unterprogramme verbessern die Übersichtlichkeit komplexer Systeme und zudem können sie durch mehrfache Verwendung die Leistung von Computersystemen besser ausnutzen. Unser erstes Unterprogramm ist sehr einfach, es setzt lediglich die Motoren M1 und M2 in Bewegung.



Wir brauchen jedoch weitere Unterprogramme. Der Roboter soll ausweichen, je nach Lage entweder nach links, rechts oder zurück. In diesem Fall genügt es nicht mehr, nur einfach die Motoren zu schalten. Es muss ein bestimmter Bewegungsablauf programmiert werden.

Schauen wir uns ein weiteres Unterprogramm an. Damit soll der Roboter beim Erkennen einer Kante zurückfahren.

Wir schalten dazu die Motoren auf Rückwärtsfahrt um. Dann soll unser Impulsrad eine definierte Strecke erkennen. Die Streckenlänge legen wir mit der Variablen VAR10 fest. Neu ist der Zuweisungsblock Z2 = 0. Nach einigem Überlegen erkennen wir den Sinn. Setzen wir die Zählvariable nicht auf Null, funktioniert der Mechanismus nur einmal, denn wenn Z2 einmal den Wert der Variablen VAR10 erreicht hat, würde bei jedem folgenden Durchlauf unser Wegstreckenzähler versagen. Aus der Sicht professioneller Programmierer haben wir es an dieser Stelle mit lokalen und globalen Variablen zu tun. Die lokale Variable Z2 wird vor jeder Benutzung im Unterprogramm initialisiert.

Fazit:

Unterprogrammaufrufe erhöhen die Übersichtlichkeit von Programmen.
Wir benutzen Variablen um verschiedene Werte, hier Weglängen, zu messen. Die unterschiedlichen Weglängen brauchen wir, damit sich unser Roboter auch aus Ecken „befreien" kann. Wären die Wege absolut identisch, könnte es passieren, dass der Roboter in der Ecke immer hin und her fährt.
Bei der Benutzung von Variablen achten wir auf ihren Gültigkeitsbereich. Lokale Variablen, d.h. Variablen, die wir nur innerhalb eines Unterprogramms verwenden, initialisieren wir vor ihrem ersten Einsatz.
Wir stellen außerdem fest, dass unser Roboter einen gewissen Bewegungsspielraum braucht, damit er richtig funktioniert. Trifft er innerhalb einer Ausweichbewegung erneut auf eine Kante, kann der Roboter darauf nicht reagieren. Die ganz großen Tüftler können versuchen, ein Lösung dafür zu finden.

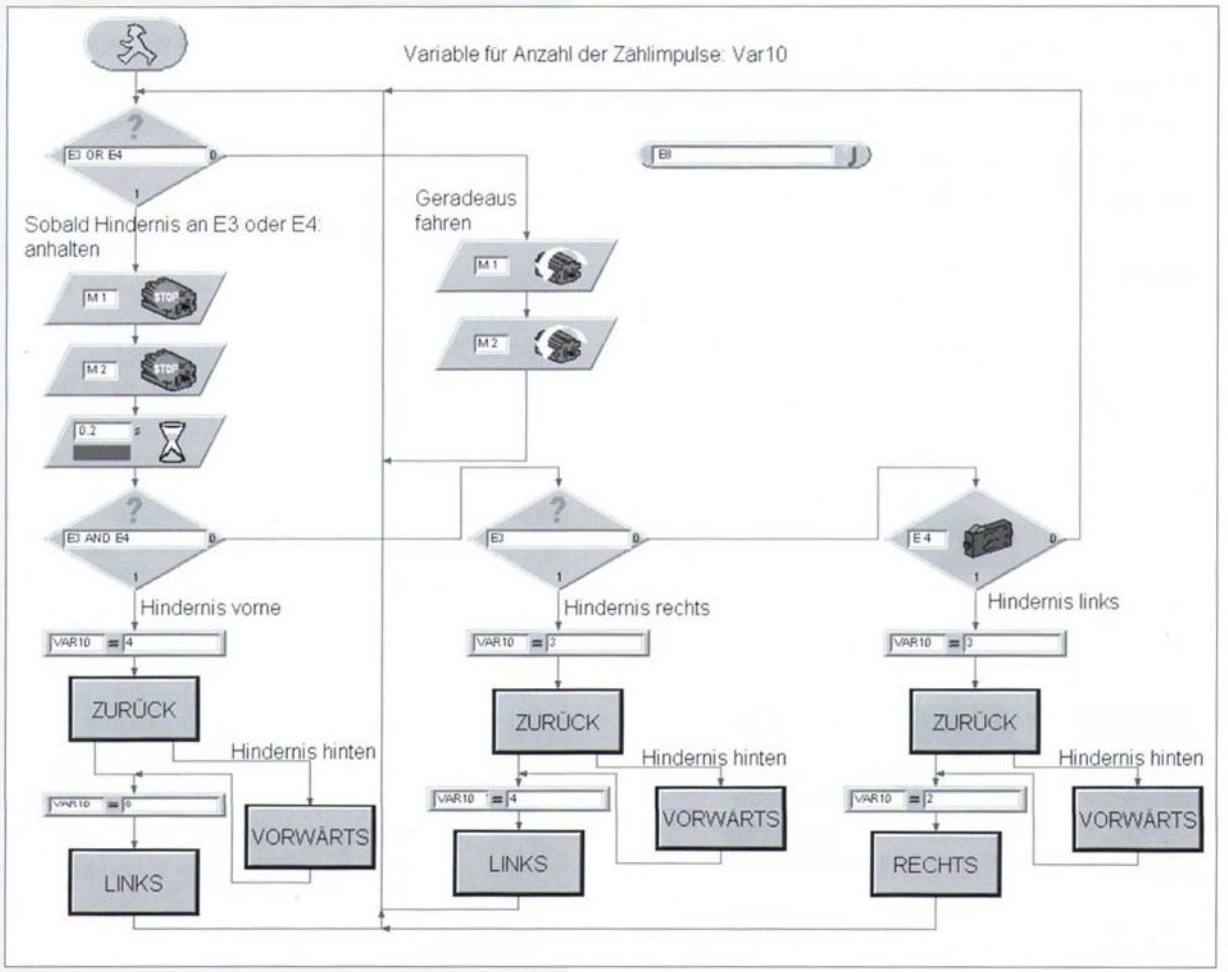
4.3 Roboter mit Hinderniserkennung

Kanten können wir nun recht gut erkennen. Problematisch wird es bei gewöhnlichen Hindernissen. Wir müssen das Prinzip der Kantenerkennung modifizieren. Ein entsprechender Aufprallkontakt ersetzt die Hilfsräder. Bei der Gelegenheit denken wir auch über einen Mangel des Kantendetektors nach; es gibt gefährliche Situationen, bei denen der Roboter während der Rückwärtsfahrt ohne Sensor hinten „blind" in der Abgrund stürzt.
Ein dritter Taster vermeidet diesen Mangel. Mittlerweile sind wir recht ausgefuchste Programmierer geworden. Ein Glück, denn nun wird die Aufgabe komplizierter, wie ein Blick auf das Programm zeigt.
Das Programm beinhaltet weitere neue Funktionsblöcke. Wir fügen an bestimmten Programmpositionen WARTE-Bausteine ein. Das ist einfach verständlich, hier wird die eingetragene Zeit abgewartet, bevor die nächste Programmfunktion abgearbeitet wird.
Richtig neu sind logische Vergleiche. Bislang haben wir immer gleich bei der Tasterabfrage einen Vergleich ausgeführt und unser Programm entsprechend verzweigt. Einen ähnlichen Vergleich gibt es auch bei der Impulszählung, den Vergleich mit einer bestimmten Anzahl von Impulsen. Neu ist auch der logische Vergleich mit mehreren Ausdrücken in einem Baustein VERGLEICH.

Aufgabe 4:

Der Roboter wird, wie in der Bauanleitung beschrieben, mit der schnelleren Getriebeuntersetzung 50:1 aufgebaut. Das Modell soll geradeaus fahren. Sobald ein Hindernis an einem der vorderen Taster (E3 oder E4) erkannt wird, stoppt der Roboter. Falls ein Hindernis rechts erkannt wurde, fährt der Roboter zurück und weicht dann nach links aus (ca 30°). Bei einem Hindernis von links weicht der Roboter nach dem Zurückstoßen nach rechts aus (ca 45°). Die ungleiche Gradzahl ist notwendig damit er auch aus einer Ecke heraus findet. Falls das Hindernis direkt von vorn erkannt wird, soll der Roboter zurück und um 90° ausweichen. Falls beim Rückwärtsfahren ein Hindernis von hinten kommt, soll er kurz nach vorne und dann wie geplant ausweichen.

Lösung:



Fazit:

Die Komplexität der Programme wird größer. Wir versuchen dieser Schwierigkeitssteigerung durch bessere Programmiertechniken zu begegnen. Unterprogramme erweisen sich dabei als gutes Mittel. Als neue Kontrollstruktur erkennen wir logische Vergleiche mit anschließender Programmverzweigung. Wir erkennen auch, dass für neue Eigenschaften bzw. eine bessere Realisierung bereits bekannter Versuche immer mehr Sensoren nötig werden.

4.4 Der Lichtsucher

Bislang haben wir den Roboter eher passiv auf Umweltsignale reagieren lassen. Nun wollen wir den Roboter aktiv auf die Suche schicken. Der Baukasten enthält zwei Fototransistoren, die wir als Lichtdetektor einsetzen. Jeder Sensor wirkt dabei auf einen Motor, damit ist eine „Leitstrahl-Verfolgung" möglich.

Das Programm besteht aus zwei Teilen. Der eine Teil beinhaltet die Suche nach einer Lichtquelle und im anderen Teil wird die Verfolgung bzw. das Ansteuern der Lichtquelle realisiert.

Wir machen natürlich wieder von den Möglichkeiten der Unterprogrammtechnik Gebrauch. Nach dem Einschalten wird das Unterprogramm LICHTSUCHE aktiviert. Dieses Unterprogramm wird erst verlassen, nachdem eine Lichtquelle gefunden wurde. Das Hauptprogramm versucht den Roboter auf die Lichtquelle zu zu steuern. Immer wenn die Richtung des Roboters stark von der Ideallinie abweicht, wird einer der Lichtsensoren nicht mehr von der Lichtquelle bestrahlt. Daraufhin wird der entsprechende Motor kurz gestoppt, so dass beide Sensoren wieder die Lichtquelle erkennen können. Daraus resultiert die Aufgabenstellung.

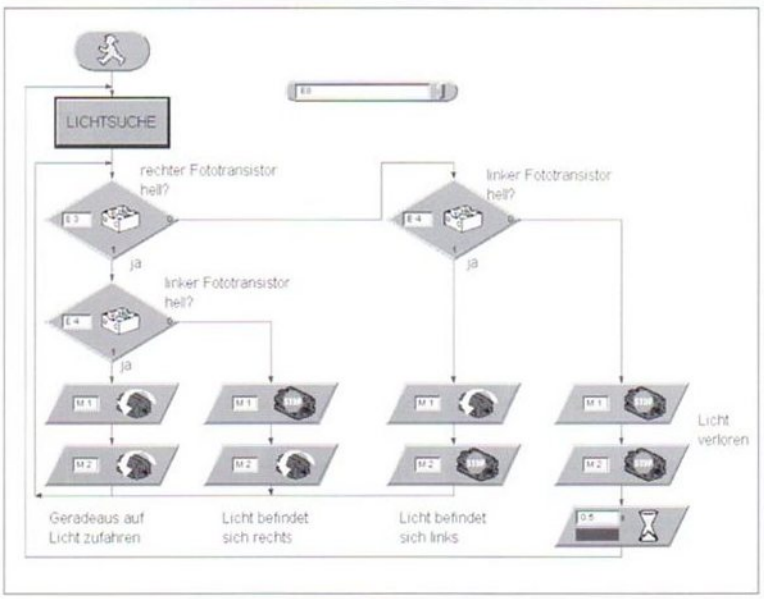
Aufgabe 5:

Wir bauen das Modell Lichtsucher mit der langsameren Untersetzung 100 : 1. Programmiere zuerst die Funktion „Licht suchen" (am Besten gleich als Unterprogramm). Der Roboter dreht sich dabei um mindestens 360° in einer Richtung. Anschließend dreht er um mindestens 360° in die andere Richtung. Wenn während der Suche ein Licht gefunden wird, stoppt der Roboter. Wird nach beiden Umdrehungen kein Licht gefunden, wird die Suche abgebrochen und das Programm beendet.

Ist die Lichtsuche erfolgreich, soll das Modell die Lichtquelle ansteuern. Bewegt sich die Lichtquelle nach links oder rechts, folgt der Roboter den Bewegungen des Lichtes. Verliert er den Kontakt zur Lichtquelle, beginnt das Programm mit der erneuten Lichtsuche. Probiere aus, ob du den Roboter mit der Taschenlampe anlocken und durch einen Hindernisparcours führen kannst.

Tipp:
Benutze als Lichtquelle eine Taschenlampe. Versuche dabei den Lichtstrahl nicht zu klein zu fokussieren, damit beide Fotosensoren von der Lichtquelle bestrahlt werden. Beachte, dass in sehr hellen Räumen deine Taschenlampe von anderen Lichtquellen, z. B. Sonnenlicht von einem großen Fenster, überstrahlt wird. Der Roboter fährt dann unter Umständen an deiner Lampe vorbei auf das hellere Licht zu.

Lösung:



Fazit:
Wir haben nun einen Roboter konstruiert, der aktiv von seiner Umgebung Notiz nimmt. Seine Sensoren sind darauf programmiert, eine Lichtquelle zu orten und falls er sie lokalisiert hat, diese Quelle anzusteuern bzw. ihr zu folgen.
Wir stellen fest, dass das Programm den Roboter stilllegt, wenn keine Lichtquelle gefunden wurde. Wenn zum Beispiel nur ein kleines Hindernis den direkten Sichtkontakt zum Roboter unterbricht, würde dieser, obwohl eine Lichtquelle da ist, diese nicht erkennen und finden können. Offenbar wäre es sinnvoll, nach einer erfolglosen Lichtsuche den Roboter mehr oder weniger zufällig auf eine andere Stelle zu bewegen und dort neu nach Licht zu suchen.
Doch was passiert, wenn das Hindernis, welches das Licht nicht zum Roboter lässt, genau in der Fahrtrichtung des Roboters liegt? Nun, diese Fragestellung scheint interessant genug, sie noch genauer zu untersuchen.

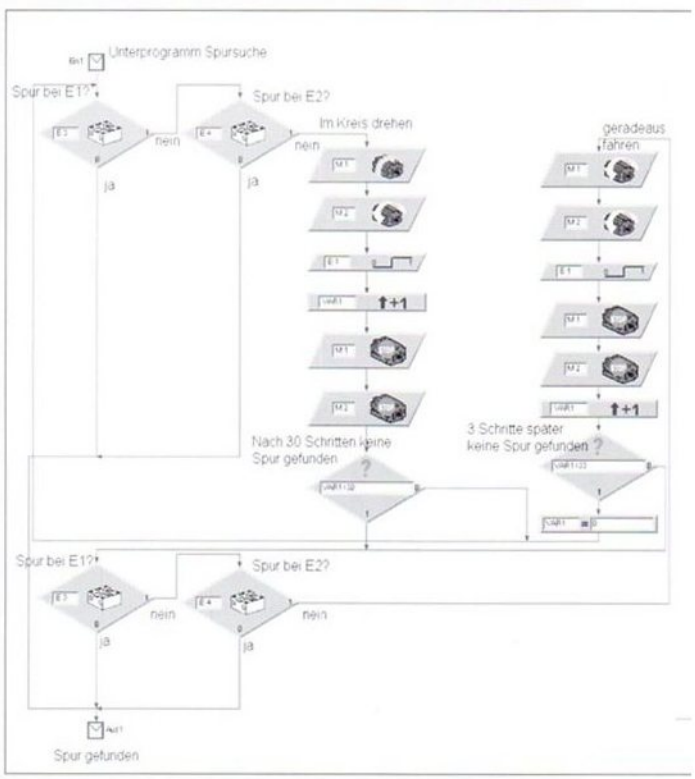
4.5 Der Spurensucher

Suche und Verfolgung sind wesentliche Eigenschaften, die intelligente Wesen besitzen. Wir haben einen Roboter gebaut und programmiert, der auf direkte Signale von seinem Ziel oder potentiellen Opfer reagiert hat.
Mit dem Spurensucher wenden wir ein anderes Suchprinzip an. Anstelle der zielgenauen Fahrt zur Lichtquelle markieren wir eine Linie, der der Roboter folgen soll. Mit den optischen Sensoren ist diese Aufgabe relativ leicht zu lösen. Wir messen das reflektierte Licht der Markierung und korrigieren danach die Motoren. Damit das auch exakt funktioniert, beleuchten wir die Linie mit unserer Lampe. Dabei achten wir darauf, dass durch ungünstige Anordnung von Lampe und Fotosensoren letztere nicht von Streulicht geblendet werden. Besonders günstig wirkt sich in diesem Zusammenhang die Lichtbündelung der optischen Linse unserer Glühlampe aus.

Aufgabe 6:
Baue das Modell Spurensucher (Getriebe 100 : 1). Als Erstes schreibe ein Unterprogramm, mit dem die Spur gesucht wird. Der Roboter dreht sich dazu 360° im Kreis herum. Wenn keine Spur gefunden wird, fährt der Roboter ein Stück geradeaus und sucht erneut. Zur Spurerkennung werden die Fotosensoren abgefragt. Hat der Roboter die Spur entdeckt, folgt er ihr. Ist die Spur zu Ende, oder verliert der Roboter diese, z. B. wegen einer starken Richtungsänderung der Spur, beginnt die Suche von neuem.

Tipp:
Nach dem Einschalten der Lampe muß eine kurze Wartezeit, ca. eine Sekunde verstreichen, bevor die Fototransistoren abgefragt werden, sonst erkennt der Fototransistor „dunkel", also eine Spur, wo keine ist. Als Spur verwenden wir ca. 20 mm breites schwarzes Isolierband oder malen uns mit Filzstift eine schwarze Spur in dieser Breite auf ein weißes Blatt Papier.

Lösung:

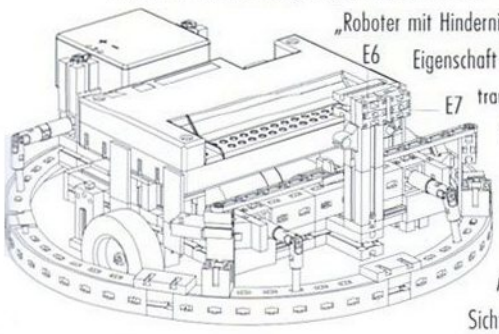


Fazit:

Hat unser Roboter die Spur gefunden, folgt er ihr unermüdlich. Wenn wir die Spur als geschlossene Linie anlegen, dreht der Roboter darauf seine Runden. Wir müssen nur zu enge Kurven vermeiden, in denen die Fotorsensoren den Kontakt zur Linie verlieren könnten. Zwar versucht der Roboter dann erneut die Spur zu lokalisieren, doch dies ist, so geben wir fairerweise zu, nur in praktisch hindernisfreier Zone möglich.

4.6 Die elektronische Motte

Das Problem der Kopplung verschiedener Verhaltensweisen, wie Suche, Verfolgen und Ausweichen ist nun bereits mehrfach aufgetaucht. Wir wollen deshalb in einem weiteren Experiment das Zusammenspiel solcher Verhalten untersuchen. Fassen wir unsere bisherigen Ergebnisse kurz zusammen. Der einfache Lichtsucher folgt einer vorgehaltenen Taschenlampe mehr oder minder blindlings. Der Roboter geht stillschweigend davon aus, dass, wo vorher die Lampe war, kein Hindernis auftauchen kann. Genauso nimmt er an, dass er die Lampe nicht wirklich erreicht. Diese Annahmen entsprechen jedoch nur in Ausnahmefällen der Realität. Wir sind gezwungen, den Roboter mit der Lampe um jedes Hindernis herum zu leiten. Wir bräuchten eigentlich einen Roboter, der Hindernissen selbstständig ausweichen kann. Dazu ergänzen wir, wie in der Abbildung dargestellt, das Modell



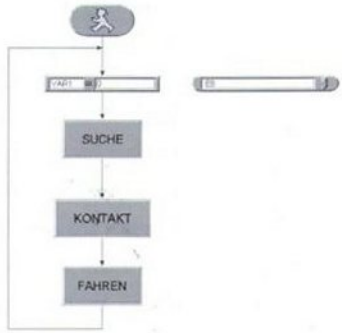
„Roboter mit Hinderniserkennung" um die Eigenschaft der Lichtsuche. Die Fototransistoren schließen wir an E6 und E7 an, alles Andere übernehmen wir von dem Modell Hinderniserkennung. Aus wissenschaftlicher Sicht haben wir den Roboter dann mit zwei Verhaltensweisen ausgestattet. Da jedoch nicht beide Verhaltensmuster gleichzeitig aktiv sein können, erhalten Sie unterschiedliche Prioritäten.

Für uns stellt sich dies so dar, dass der Roboter im Normalfall auf Lichtsuche ist. Wird ein Hindernis erkannt, quasi eine Gefahr für den Roboter, wird das Verhalten Hindernisvermeidung aktiv. Ist alles wieder im grünen Bereich, kann der Roboter weiter nach der Lichtquelle forschen.

Damit haben wir dem Roboter wesentliche Fähigkeiten vermittelt, die es ihm erlauben, selbstständig zu navigieren und Gefahren auszuweichen. Wer einen Freund hat, der ebenfalls im Besitz eines Fischertechnik-Kastens ist, kann dieses Experiment noch weiter treiben. An jeden der beiden Roboter wird einfach eine Lichtquelle montiert und dann suchen sich die Roboter gegenseitig.

Bislang haben wir bei der Programmierung meist nach dem Prinzip „Versuch und Irrtum" gearbeitet. Du hast wahrscheinlich bei deinen Programmen „einfach" angefangen und im Laufe deiner Experimente erkannt, wie (oder wie besser nicht) der Roboter dann das getan hat, was er tun sollte. Professionelle Softwareentwickler können natürlich bei ihren Produkten keinesfalls auf eine solche Entwurfsmethode zurückgreifen. In solchen Fällen muss, bevor die erste Programmzeile erstellt wird, eine genaue Entwurfsstrategie definiert werden. Das macht man nicht einfach so, sondern benutzt feste Richtlinien. Durchgesetzt haben sich Verfahren, die solche seltsamen Bezeichnungen wie z. B. „Top-Down-Entwurf" tragen. Man versucht hier das Gesamtsystem von oben herab zu definieren, ohne sich am Anfang mit allen Details zu befassen. Wir versuchen unsere Motte nach dem Top-Down-Entwurf zu programmieren. Dazu beginnen wir mit dem Hauptprogramm, der sogenannten „Main-Loop".

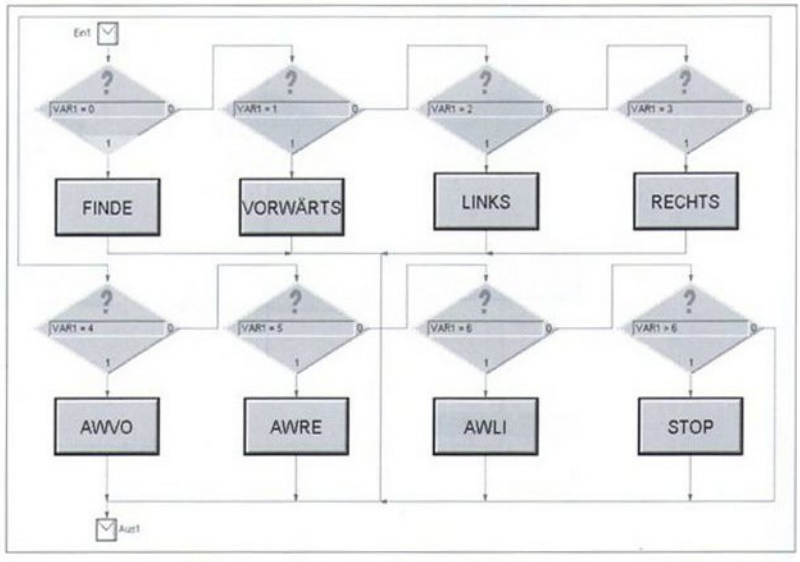
Die Abbildung zeigt den einfachen Entwurf. Die Main-Loop besteht lediglich aus den für den Roboter notwendigen Verhaltensweisen, SUCHE, KONTAKT und FAHREN. Seltsam erscheint uns die Variable Var1 und wie priorisieren wir mit einer solch einfachen Schleife die Verhaltensweisen des Roboters?



Nun, die Priorisierung erfolgt durch die Reihenfolge der Unterprogramme und zur Zuweisung der Fahraufträge dient die Variable Var1. Wenn wir genau überlegen, können wir die notwendigen Fahrbewegungen des Roboters auf eine klar definierte Anzahl Manöver reduzieren. Der Roboter benötigt eine Suchbewegung, eine Ausweichbewegung und eine Korrekturbewegung. Ausweich- und Korrekturbewegungen müssen noch nach Richtungen, rechts oder links, unterschieden werden. Damit wird deutlich, dass das Unterprogramm SUCHE eine von n (n = Gesamtzahl der Bewegungen) möglichen Bewegungen errechnet, genauso wie das Unterprogramm KONTAKT. Da KONTAKT nach SUCHE ausgeführt wird, überschreibt es die Anweisungen von SUCHE. Das Unterprogramm FAHREN führt nur noch aus, was ihm befohlen wird. Das Top-Down-Entwurfsverfahren scheint uns wirklich zu nützen.

Doch noch wissen wir nicht, wie die Unterprogramme aussehen. Beginnen wir mit dem ersten Unterprogramm SUCHE. Dieses Unterprogramm ist für die Abfrage der Fotosensoren zuständig. In Abhängigkeit von den beiden Sensoren gibt es vier mögliche Varianten, linker Sensor, rechter Sensor, linker und rechter Sensor, kein Sensor. Damit werden exakt vier mögliche Fahrmanöver definiert, FINDE, VORWÄRTS, LINKS, RECHTS. Auch diese Fahrmanöver werden Unterprogrammen zugeordnet; wir gehen streng nach Top-Down-Entwurfskriterien vor. Jetzt ist klar, wie das erste Unterprogramm SUCHE aussieht. SUCHE liefert einen Parameter der in FAHREN umgesetzt wird.

Das eigentliche Unterprogramm ist sehr einfach. Über mehrfache Vergleiche wird der Ausgangsparameter eingestellt. Im fertigen Programm wird danach im Unterprogramm KONTAKT geprüft, ob Hindernisse die Taster am Roboter ausgelöst haben. Der Einfachheit halber übergehen wir dieses Programm und sehen uns an, wie in FAHREN die Motorbewegungen aktiviert werden.



Mit einigem Erstaunen stellen wir fest, dass in diesem Unterprogramm weitere Unterprogramme aufgerufen werden. Hört das denn niemals auf? Wir sind immer noch in der Entwurfsphase (Top-Down nächste Stufe). FAHREN legt fest, wann welches Fahrmanöver gestartet wird. Erst hinter den verschiedenen Unterprogrammen FINDE, VORWÄRTS,... verbirgt sich „richtiger" Programmcode. Wir sind endlich ganz unten angekommen. Jetzt werden die Motoren an- bzw. ausgeschaltet. Jedes dieser Unterprogramme hat eine klar begrenzte Aufgabe, die wir übersichtlich programmieren können.

Fazit:

Komplexe Programme erfordern völlig neuartige Lösungsansätze. Es ist sinnvoll, einen durchgehenden Lösungsansatz anzustreben. Wir erstellen ein Programm nach dem Top-Down-Entwurfsansatz. Die notwendigen Lösungsansätze werden in (zuerst) formalen Strukturen formuliert, z. B. SUCHE oder FAHREN. Erst im weiteren Verlauf werden diese Ansätze verfeinert und zuletzt mit dem eigentlichen Programmcode gefüllt. Damit trennen wir die Programmstruktur vom eigentlichen Programmcode. Beides können wir getrennt betrachten und, das ist sehr wichtig, auch getrennt nach Fehlern analysieren.

4.7 FTS – Fahrerloses Transportsystem

Verlassen wir das Gebiet der wissenschaftlichen Experimente, böse Zungen sprechen oft von Spielerei, und begeben wir uns in den Bereich der praktischen Anwendungen.
Wir wollen nun einen Roboter nicht nur um seiner selbst willen bauen und programmieren, sondern er soll eine Aufgabe erledigen. Zu diesem Zweck wird der Spurensucher mit einer beweglichen Transportgabel ausgerüstet. Damit entsteht ein Gabelstabler-Roboter, der in der Lage ist auf einer Palette gelagertes Material zu transportieren.
So einfach dieser Versuch erscheinen mag, enthält er doch alle Komponenten einer industriellen Anwendung. Und, derartige Transportsysteme sind zum Teil bereits heute im Einsatz.
Mittlerweile sind wir, was die Programmiererfahrung betrifft, gut geschult, so dass wir uns frohen Mutes an dieses doch sehr komplexe Programm wagen.

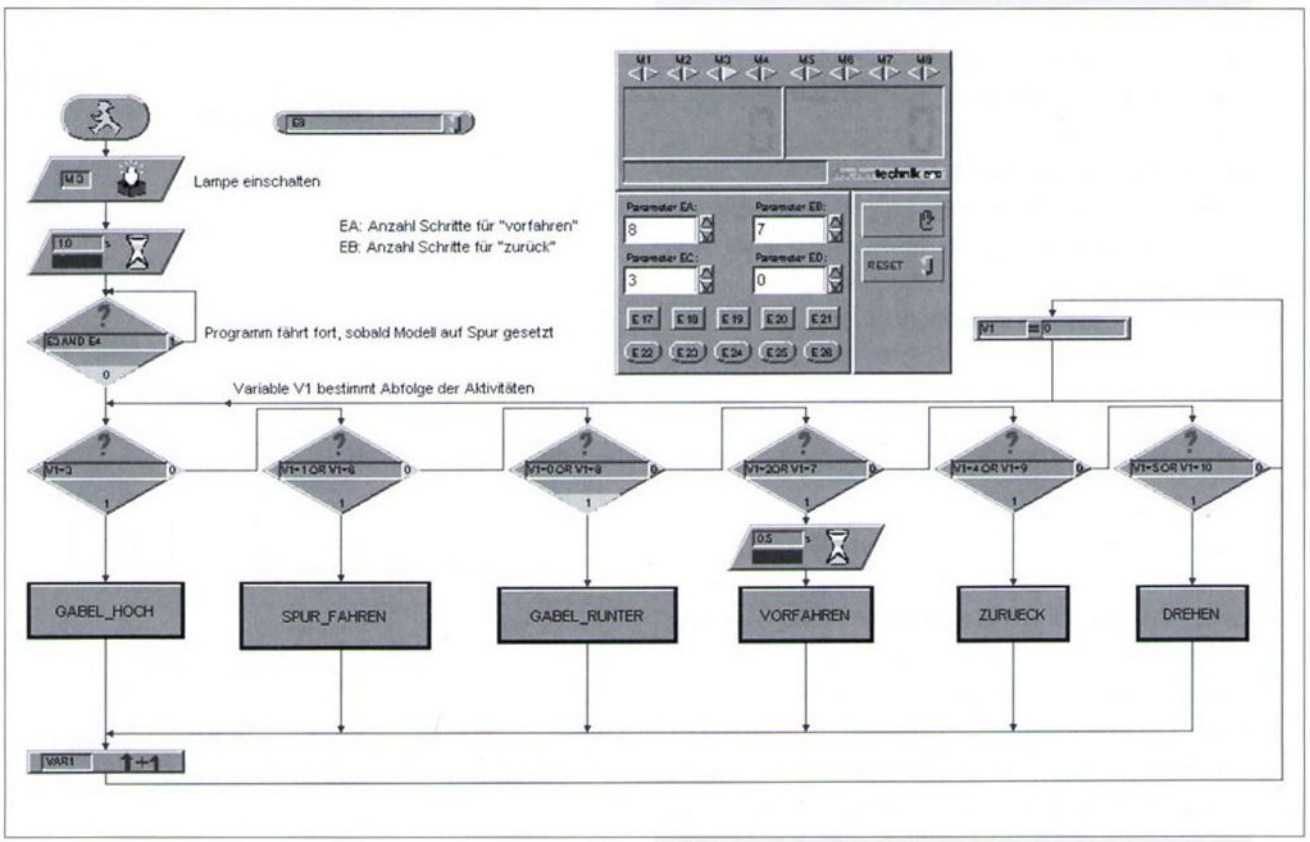
Aufgabe 7:

Das fahrerlose Transportsystem (FTS) soll entlang einer schwarzen Spur fahren. Am Ende der Spur nimmt es eine Palette auf, dreht sich um 180° und fährt auf der Spur zurück. Wird das Spurende detektiert, setzt der Roboter die Palette dort ab, wendet erneut und holt die nächste Palette.

Aufgabe 8:

Als Erweiterung der Aufgabe 7 ergänzen wir das Programm so, dass die Palette an den Endpunkten der Spur nur jeweils kurz abgesetzt wird. Der Roboter sucht also die Palette, fährt zum Spurende, setzt die Palette kurz ab, nimmt sie wieder auf, fährt zum anderen Ende,...usw.

Lösung:



Fazit:

Sobald wir einen Blick auf das Programm werfen, erkennen wir sofort den bereits bekannten Programmansatz. Eine Zustandsvariable steuert das Verhalten unseres fahrerlosen Transporters.

Mit Hilfe unserer mittlerweile recht ausgefeilten Programmiertechniken gelingt es uns, vielfältige Steuerungen zu realisieren. Mit dem fahrerlosen Transportsystem demonstrieren wir die Möglichkeiten, die die Kombination der fischertechnik-Baukästen mit dem Intelligent Interface bietet. Beginnend mit einfachen mobilen Robotern haben wir nun einen Stand erreicht, der sich kaum noch von der industriellen Steuer- und Regeltechnik unterscheidet.

5 Fehlersuche

Experimentieren macht Spaß. Doch nur solange alles funktioniert. Am besten wäre es, wenn alles immer auf Anhieb gelänge. Doch leider ist dies nicht so. Erst wenn ein Modell nicht arbeitet, stellt sich heraus, ob man den Mechanismus genau verstanden hat und den Fehler sofort findet.

Bei mechanischen Fehlern kann man immerhin noch etwas sehen (falsch zusammengebaut) oder fühlen (Schwergängigkeit). Kommen elektrische Probleme hinzu, wird es schwieriger.

Die Profis nutzen zur Fehlersuche eine Reihe sehr unterschiedlicher Messinstrumente, wie z. B. Spannungsmesser oder Oszillograf. Solche Geräte hat nicht jeder greifbar. Wir wollen deswegen versuchen, mit einfachen Mitteln einen Fehler einzukreisen und zu beheben.

Bevor wir mit unseren Experimenten beginnen, müssen wir einige Komponenten aus dem Fischertechnik-Baukasten erst fertigstellen. Im Wesentlichen handelt es sich um die Kabelverbindungen. Hier werden die mitgelieferten Stecker an die einzelnen Kabelabschnitte angeklemmt.

Zuerst wird das Kabel zugeschnitten. Wir messen dazu die vorgegebenen Längen ab und schneiden die Abschnitte zu. Bevor wir schneiden, prüfen wir genau, ob die Längen stimmen. Jedes Kabel wird nach Fertigstellung durchgemessen. Dazu brauchen wir den Akku und die Lampe. Leuchtet die Lampe nachdem sie mit dem Akku verbunden wurde, ist das Kabel in Ordnung. Stimmt auch die Farbzuordnung, roter Stecker rotes Kabel, grüner Stecker grünes Kabel, legen wir das Kabel zur Seite und prüfen das nächste.

Arbeitet das Programm (auch das mitgelieferte) nicht mit unserem Modell zusammen, starten wir die Interfacediagnose. Dieses Hilfsprogramm gestattet uns, die Ein- und Ausgänge separat zu testen. Hier muss jeder Sensor die entsprechende Aktion am Interface auslösen.

Wenn wir das geprüft haben, wissen wir, dass elektrisch alles in Ordnung sein sollte. Über die Motorsteuerknöpfe schalten wir die Aktoren einzeln ein bzw. aus. Ist hier ebenfalls alles in Ordnung, suchen wir die mechanische Ursache.

Ein übler Fehler sind Wackelkontakte. Zum einen können die Anschlussstecker lose in den Buchsen sitzen. Ist dies der Fall, werden mit einem kleinen Uhrmacherschraubenzieher die Kontaktfedern der Stecker etwas aufgeweitet. Vorsicht, zu starkes Aufweiten führt zum Bruch der Kontakte oder zu starker Schwergängigkeit beim Einstecken.

Eine andere Ursache für Wackelkontakte sind gelockerte Klemmverbindungen an den Anschraubstellen der Stecker. Bitte vorsichtig festschrauben! Bei der Gelegenheit wird gleich geprüft, ob keine der dünnen Kupferdrähtchen abgebrochen sind. Gibt es unerklärliche Aussetzer während des Betriebes, kann ein beinahe leerer Akku die Ursache sein. Die Spannung sinkt beim Zuschalten einer Last (Motor ein) kurz ab und damit wird ein Reset für den Computer auf dem Interface ausgelöst. Ein solcher Fehler ist zum Teil sehr schwer zu finden, da das Programm ja immer erst einmal funktioniert.

Treten Fehler bei selbst geschriebenen Programmen auf, die man sich nicht erklären kann, sollte sicherheitshalber ein möglichst ähnliches der mitgelieferten Programme eingespielt werden, damit elektrische oder mechanische Defekte ausgeschlossen werden.Führt dies alles nicht zum Erfolg, bleibt noch der Kontakt zum fischertechnik-Service.

Mobile Robots II

- Begleitheft
- Activity booklet
- Manuel d'accompagnement
- Cuaderno adjunto
- Folheto

fischerwerke
Artur Fischer GmbH & Co. KG
Weinhalde 14–18
D-72178 Waldachtal
Telefon: 0 74 43/12-43 69
Fax: 0 74 43/12-45 91
email: info@fischertechnik.de
http://www.fischertechnik.de

78 897 • ZE • 10/02 • Printed in Germany • Technische Änderungen vorbehalten • Subject to technical modifications

fischertechnik