Programmieren mit der BASIC-Stamp 2

Teil 8: (Ende): Der photophobische Roboter

Von Dennis Clark

Phase 3: Der photophobische Roboter und die subsumtive Programmierung

Wir wollen nun überlegen, ob es für den Roboter eine anspruchsvollere Tätigkeit gibt als nur plan- und ziellos im Raum umherzufahren. Ein Schritt in diese Richtung ist das Verhalten ähnlich einer Grille, die das Licht scheut und danach trachtet, sich in einer dunklen Ecke zu verkriechen. Dieses Verhalten bezeichnen wir als "photophobisch", was ungefähr so viel wie "lichtscheu" bedeutet. Der Fluchtreflex gehört zum Selbsterhaltungs-Repertoire vieler Lebewesen. Für die Selbsterhaltung des Roboters ist er zwar nicht unbedingt notwendig, doch wir können auch ihm zu diesem Urverhalten verhelfen. Dabei liefern die Lichtsensoren, die wir schon bei früheren Experimenten installiert haben, die vom Programm benötigten Informationen. Zum Abfragen der Sensor-Informationen verwenden wir den Basic-Stamp-2-Befehl *rctime*. Das Lesen der Sensor-Informationen dauert um so länger, je weniger Licht detektiert wird. Da das Lesen eines einzelnen Lichtsensors zu viel Zeit in Anspruch nimmt, wenden wir einen kleinen Trick an. Unterschieden werden sollen lediglich zwei Fälle: Es soll erkannt werden, ob der rechte oder der linke Sensor heller beleuchtet ist. Während seines Weges durch den Raum blickt der Roboter sozusagen ständig abwechselnd nach rechts und nach links. An Hand der gewonnenen Informationen entscheidet er sich, den Weg in die Richtung fortzusetzen, in die der schwächer beleuchtete Lichtsensor zeigt. Diese Richtung muss er eine bestimmte Zeit lang beibehalten, denn sonst kann es passieren, dass er kaum von der Stelle kommt. Ein ruckweise nur wenige Zentimeter vor- und rückwärts fahrender Roboter würde sicher nicht dem Verhalten einer Grille entsprechen. Das photophobische Verhalten ist komplexer als die einfacheren Verhaltensmuster der vorangegangenen Experimente. Hier gibt es nicht nur zwei Zustände, sondern fünf:

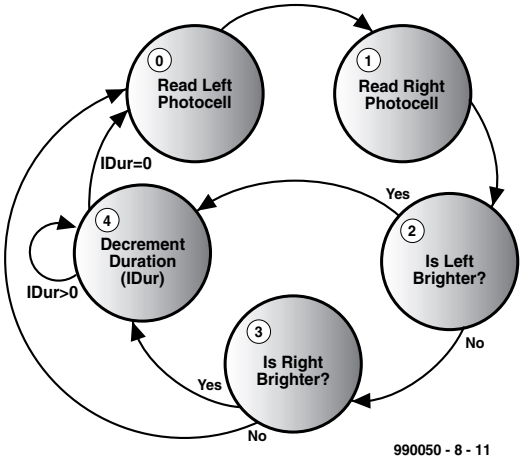

Bild 1. Vorschlag für die State Machine des photophobischen Verhaltens.

```
State 0
Lese linken Lichtsensor
Setze state = 1
State 1
Lese rechten Lichtsensor
Multipliziere den Wert mit 1,5, da
der rechte Sensor bei gleicher
Beleuchtungsstärke etwas niedrigere
Werte als der linke liefert
Setze state = 2

State 2
Addiere die Differenz zum Messwert
des linken Sensors
Wenn Links heller ist als Rechts
   Setze lDir = turn right (tr)
   Setze lDur = 30
   Setze lstate = 4 (Dekrementie-
   ren)
  Anderenfalls
   Setze lstate = 3
State 3
  Addiere die Differenz zum Mess-
  wert des rechten Sensors
  Wenn Rechts heller ist als Links
   Setze lDir = turn left (tl)
   Setze lDur = 30
   Setze lstate = 4
  Anderenfalls
   Setze lstate = 0 (keine Aktion,
   beide Seiten ungefähr  gleich)

State 4
  Dekrementiere lDur, lDur = lDur - 1
  Wenn lDur = 0
```

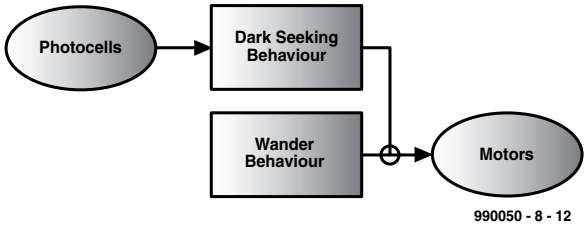

Bild 2. Wander- und Seek-Aktionen, dargestellt mit den Termen der subsumtiven Programmierung.

```
  Setze lstate = 0 (Aktion been-
  det, beginne von vorn)
Anderenfalls
  keine Aktion, Wiederholung von
  State 4 für weiteres  Dekremen-
  tieren, Aktion noch nicht been-
  det
```

Bild 1 zeigt die State Machine (vgl. Teil 6), die Grundlage des photophobischen Verhaltens unseres Roboters ist. Diese State Machine wurde erst erstellt, als alle notwendigen Aktionen definiert und in logischer Abfolge organisiert waren. Da sich Variablenwerte ändern und Daten zwischen den States ausgetauscht werden müssen, haben wir diesen Weg gewählt. In anderen Fällen mag es einfacher sein, zuerst das State-Diagramm zu erstellen und hiervon ausgehend die einzelnen State-Funktionen zu definieren und zu organisieren. Wie die Erfahrung zeigt, gibt es meistens mehr als nur einen Weg, um ein bestimmtes Verhalten zu realisieren.
Die State-Deskriptionen sollen so detailliert und ausführlich wie möglich sein, so dass bei der Programmierung möglichst nichts vergessen wird. Bei State 1 wurde der Messwert des rechten Sensors mit 1,5 multipliziert. Durch den Korrekturfaktor werden die unvermeidbaren Exemplarstreuungen der Lichtsensoren kompensiert. Der Faktor hängt natürlich von den real verwendeten Sensoren ab, er muss durch Messungen ermittelt werden. Die etwas komplexere State Machine des photophobischen Roboters deutet schon darauf hin, dass auch das Programm komplexer ist als bisher gewohnt. Die Programmierung wird dadurch jedoch nicht wesentlich schwieriger. Wenn die Einzelschritte in der Form definiert wurden, die wir oben für den photophobischen Roboter gewählt haben, können wir ein Programm unmittelbar an Hand der dort stehenden State-Deskriptionen schreiben. Die Einzelschritt-Beschreibung ist dem Programm schon recht ähnlich, sie wird deshalb auch "pseudo code" genannt. Wenn man im konkreten Fall einen solchen Pseudo-Code nicht erstellen kann, kann man auch das Programm nicht schreiben!
Das Basic-Stamp-2-Programm für das photophobische Roboter-Verhalten ist in Listing 1 zu finden. Das Programm enthält einige Shortcuts, die eventuell der Erklärung bedürfen. Der Befehl tmp = pright >> 1 teilt pright durch zwei und legt das Ergebnis in tmp ab. In der nächsten Zeile werden tmp und pright addiert. Auf diese Weise wird pright mit 1,5 multipliziert, was die Unterschiede zwischen dem rechten und dem linken Sensor kompensiert. Der Sprungbefehl hat eine indizierte Programm-Verzweigung zu einem anderen Programmteil zur Folge; hier ist der Index der betreffende State.
Übernehmen Sie die Variablen- und Konstantenliste auf der linken Seite in Ihr Programm (an den Anfang), und hängen Sie die Routine lightlook an das Ende Ihres Programms an! Die Hauptroutine main kann dann zum Beispiel wie folgt aussehen:

```
'Setup for running
wstate = 0   'Ausgangswert Wander-State
lstate = 0   'Ausgangswert Photophobie-State

main:
    gosub wander
    gosub lightlook
    gosub act
goto main
```

Nach dem Programmstart setzt die Routine wander die Variable drive auf eine zufällige Bewegungsrichtung. Anschließend ändert die von wander völlig unabhängige Routine lightlook die Variable drive, sobald sie eine fluchtträchtige Dunkelzone erkennt. Zuletzt setzt die Routine act die Variable drive auf einen Wert, der die Aktivierung und Drehrichtung der Antriebsmotoren in der gewünschten Weise bestimmt. Die genauere Betrachtung ergibt, dass lightlook eine höhere Priorität als wander besitzt, denn lightlook kann drive verändern, nachdem diese Variable von wander gesetzt wurde. An diesem Beispiel werden die erweiterten Möglichkeiten deutlich, die diese Methode der Roboter-Programmierung bietet. In Bild 2 ist das neue BoE-Bot-Verhalten in Form eines Subsumtions-Diagramms dargestellt.
Hier tut sich ein spannendes Betätigungsfeld auf: Verwandeln Sie den photophobischen Roboter in eine Grille, die sich in die Dunkelheit verzieht und dort zirpt! Am Anfang dieses Experiments können folgende Überlegungen stehen:

- Beide Lichtsensoren müssen Beleuchtungsstärken detektieren, die unterhalb der Dunkelgrenze liegen (niedrige Beleuchtungsstärke = hoher Messwert),
- sobald die Dunkelgrenze unterschritten ist, muss der Roboter stehen bleiben,
- es werden die Messwerte von lightlook übernommen, so dass keine neuen Messungen nötig sind,
- der Roboter zirpt nur dann, wenn er im Dunkeln still steht,
- dieser Zustand hat höhere Priorität als lightlook und wander,
- wenn einer der beiden Sensoren eine über dem Schwellenwert liegende Beleuchtungsstärke detektiert, wird die Roboterbewegung fortgesetzt.

Wenn wir die von lightlook gemessenen Beleuchtungswerte in ein anderes Modul übernehmen, verletzen wir die Regel, dass die Module voneinander unabhängig sein sollen. Da eine zweite Messung

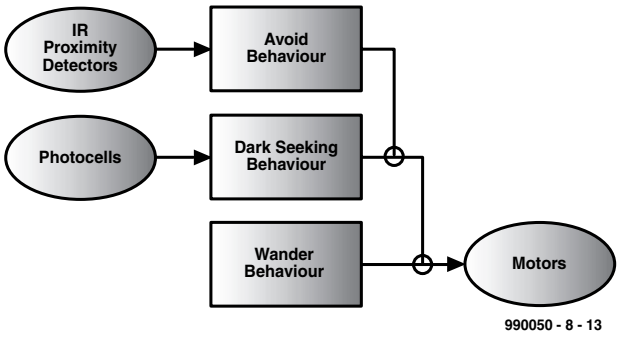

Bild 3. Subsumption Network Diagram eines photophobischen Roboters.

Listing I3

```
'light looker vars and constants
LLIGHT   con      11        'left sensor
RLIGHT   con      4         'right sensor
pleft    var      word      'left value
pright   var      word      'right value
lstate   var      byte      'FSM state
lDur     var      byte      'how long to go
lDir     var      word      'where to go
LMARG    con    15        'light margin
lightlook:
  low LLIGHT              'set up for sensors
  low RLIGHT              'branch takes 200us
  branch lstate,[lread1,lread2,lcomp1,lcomp2]
    lDur = lDur - 1       'state 4 decr duration
    drive = lDir 'correct direction
    if lDur > 0 then lDone1       'still counting
        lstate = 0                'restart FSM
lDone1:                           'done
    return
lread1:                           'state 0
    rctime LLIGHT,0,pleft         'get left value
    lstate = 1                    'go next state
    return

lread2:                           'state 1
    rctime RLIGHT,0,pright        'get right value
    tmp = pright >> 1             'compensation
    pright = pright + tmp
    lstate = 2                    'go next state
    return
lcomp1:                           'state 2
    tmp = pleft +LMARG            'set threshhold
    if tmp > pright then lDone2   'left not past
                                  'threshhold
        lDir = tr                 'bright to left
        lDur = 30                 'for a while
        lstate = 4                'go decr state
        return
lDone2:
    lstate = 3                    '2nd compare
    return
lcomp2:                           'state 3 compare
    tmp = pright +LMARG  'set threshhold
    if tmp > pleft then lDone3'not past
        lDir = tl                 'bright to right
        lDur = 30                 'for a while
        lstate = 4                'go decr state
        return
lDone3:
    lstate = 0                    'none past
    return
```

Listing I4

```
'IRPD vars and constants
ileft    var      in9      'IR LED outputs
iright   var      in0      'i=(see code)
IEN      con      5        'enable for 555
ilast    var      byte     'hit counter
avoid:'IRPD routine
  High IEN                 'enable 555
  i=0
  i = ileft * 2 + iright   'read IRPD
  low IEN                  'disable 555
  if ilast = I then ickit  'two reads agree
    goto iDone             'just first read
ickit:  'This line chooses new direction
  lookup i,[rr,tr,tl,drive],tmp
  drive = tmp
  i=0                      'clear history
iDone:
  ilast = i                'new history
return
```

praktisch das gleiche Ergebnis liefert und nur unnötig Zeit kostet, ist dieser Regelverstoß tolerierbar. Außerdem ist die Unabhängigkeit der Module kein unumstößliches Gesetz, vor allem wenn der Verstoß keine oder nur vernachlässigbare Nebeneffekte hat. Natürlich kann man auch hier die rctime-Funktion hinzufügen, falls man absolut korrekt vorgehen möchte.
Der nächste Schritt ist das Aufstellen einer detaillierten Liste der Aktionen, die innerhalb der einzelnen States der Finite State Machine stattfinden müssen. Anschließend wird das State-Diagramm erstellt. Es muss überlegt werden, wo die Verhaltens-Routine im Subsumtions-Diagramm des Roboters einzubauen ist. Danach folgt das Schreiben der Routine und das Einfügen des gosub-Befehls im Hauptprogramm main. Nicht vergessen werden darf die Initialisierung der State Machine im Setup-Teil vor dem Hauptprogramm!

Phase 4: Ausweichverhalten

Unser BoE-Bot-Roboter ist jetzt so programmiert, dass er sich auf zufälligen Wegen bewegt und sich dabei photophobisch verhält. Er ist jedoch noch nicht in der Lage, Hindernissen auszuweichen und sie zu umfahren. Ein Vermeidungsverhalten, das ihn vor Zusammenstößen mit Wänden und Möbeln schützt, würde seine Gebrauchsfähigkeit sicher beträchtlich erhöhen. Wir setzen die schon aus einem vorangegangenen Experiment bekannten Infrarot-Näherungsdetektoren (IRPD's) zum Erkennen der Hindernisse ein. Der Roboter soll seine Bewegungsrichtung um 180 Grad ändern, wenn er ein Hindernis unmittelbar vor sich hat. Wie kann man dieses Verhalten programmieren? Zum Beispiel so:

```
Lese den IRPD-Messwert
Wenn es der erst Messwert ist, dann
Speichere den Messwert
Anderenfalls
Wenn es der letzte Messwert ist,
dann
   Bestimme die Bewegungsrichtung
   und setze drive
   Lösche den Messwert-Speicher
Anderenfalls
   Speichere den Messwert
```

Dieses Verhaltensmuster ist weniger komplex als lightlook, denn das Umfahren eines Hindernisses lässt sich einfacher realisieren als die Flucht in eine dunkle Ecke. Es ist sogar so unkompliziert, dass eine State Machine nicht erstellt werden muss. Eigentlich besteht das Ausweichverhalten aus zwei States: Der erste nimmt lediglich eine IRPD-Messung vor, der zweite nimmt eine zweite Messung vor und vergleicht sie mit der ersten. Da die beiden Aktionen sehr eng miteinander verbunden sind, ist es sowohl schwierig als auch unnötig, sie aufzuspalten.
Das Programm von Ausweich-Verhaltensmuster avoid ist aus **Listing 2** ersichtlich.
Die Basic-Stamp 2 führt mathematische Operationen stets von "links nach rechts" aus. Die Operationen müssen entweder in der richtigen Reihenfolge stehen, oder es sind Klammer-Ausdrücke zu benutzen. Die verwendeten Infrarot-Demodulatoren legen die zugehörigen I/O-Leitungen auf Low, wenn sie ein Signal detektieren. Folglich hat "3" die Bedeutung "kein Signal", "2" bedeutet "Signal rechts", "1" steht für "Signal links", und "0" ist das Zeichen für "Signale auf beiden Leitungen". Die lookup-Instruktion enthält in ihrer Tabelle den alten Wert von drive. Der Grund ist, dass keine Änderungen vorgenommen werden können, solange keine Erkennung stattfindet; der Wert von drive muss folglich unverändert bleiben. Wenn ein Hindernis erkannt wird, ändert das Modul avoid die Bewegungsrichtung des Roboters. Die Richtung wird so geändert, dass das Hindernis nicht mehr im Erfassungsbereich liegt.
Um dieses Verhalten zu implementieren, übernehmen Sie die Variablen- und Konstantenliste an den Anfang Ihres Programms und fügen Sie die avoid-Routine zu den übrigen Subroutinen hinzu. Da avoid höhere Priorität als alle anderen bisher implementierten Verhaltens-Subroutinen haben soll, wird avoid im Hauptprogramm main wie folgt aufgerufen:

```
'Setup for running
wstate = 0     'Ausgangswert Wan-
               der-State
lstate = 0     'Ausgangswert Photo-
               phobie-State
ilast = 0      'Ausgangswert der
               Avoid-History

main:
    gosub wander
    gosub lightlook
    gosub avoid
    gosub act
goto main
```

Das Subsumption Network Diagram, das alle besprochenen Verhaltensmuster enthält, ist in **Bild 3** dargestellt.
Subsumtions-Diagramme sind insbesondere dann nützlich, wenn man Außenstehenden die potentiellen Roboter-Aktivitäten erklären möchte. An Hand solcher Diagramme kann man Vorhersagen über das Roboter-Verhalten treffen und auch das Verhalten unterschiedlich programmierter Roboter miteinander vergleichen. Das Diagramm schafft meistens wesentlich schneller Klarheit über das Roboter-Verhalten als das Nachvollziehen des zugehörigen Programms.
Dem weiteren Ausbau Ihres BoE-Bot-Projekts sind kaum Grenzen gesetzt. Sie können zum Beispiel ein Verhaltensmuster implementieren, das den Roboter bei einem Zusammenstoß mit einem Objekt veranlasst, sich rückwärts zu bewegen und um das Objekt herumzufahren. Zuerst muss eine sensorische Stoßstange montiert werden, die der Basic-Stamp 2 die Berührungen signalisiert. Dann ist eine Liste der notwendigen Aktionen zu erstellen. Die Aktionen werden in States umgesetzt, und es wird festgelegt, welche Variablen eingeführt werden müssen. Überlegen Sie auch, welche Priorität die Reaktion auf ein Stoßstangen-Signal im Subsumtions-Diagramm haben muss. Die Programmierung dieses Verhaltensmusters (und aller weiteren) ist nun nicht mehr schwierig. Wir wünschen Ihnen viel Spaß!

(990050-8)gd

Internet:

http://www.parallaxinc.com
BASIC Stamp Manual Version 1.9, BASIC Stamp DOS- und Windows-Editor, Programmbeispiele, internationale Distributoren.

http://www.stampsinclass.com
BoE Dokumentation, Robotics Curriculum, BoE-Bot *.dxf und *.dwg Grafikformate, Diskussionen zum Einsatz der BASIC Stamp im Ausbildungsbereich.

chucks@turbonet.com
Initiator des BoE-Bot-Projekts und Co-Autor dieser Artikelserie. Technische Unterstützung.

kgracey@parallaxinc.com
Co-Autor dieser Artikelserie. Technische Unterstützung und Beantwortung von Fragen zum BoE-Bot-Projekt.

http://www.elektronikladen.de
Deutscher Parallax-Distributor