Programmieren mit der BASIC-Stamp

Teil 3: BASIC-Programmierung

Dieser Teil der BASIC-stamp-Serie beschreibt Sprungbefehle und den Umgang der BASIC-stamp mit einem angeschlossenen EEPROM, um den Boe-Bot einen vorgegebenen Weg folgen zu lassen. Ein Piezolautsprecher wird als Rückkopplungs-Mechanismus eingesetzt, um die aktuelle Position innerhalb des Programms zu identifizieren. Ein mobiler Temperaturlogger als optionales Projekt soll das Verständnis für die synchrone serielle Kommunikation vertiefen.

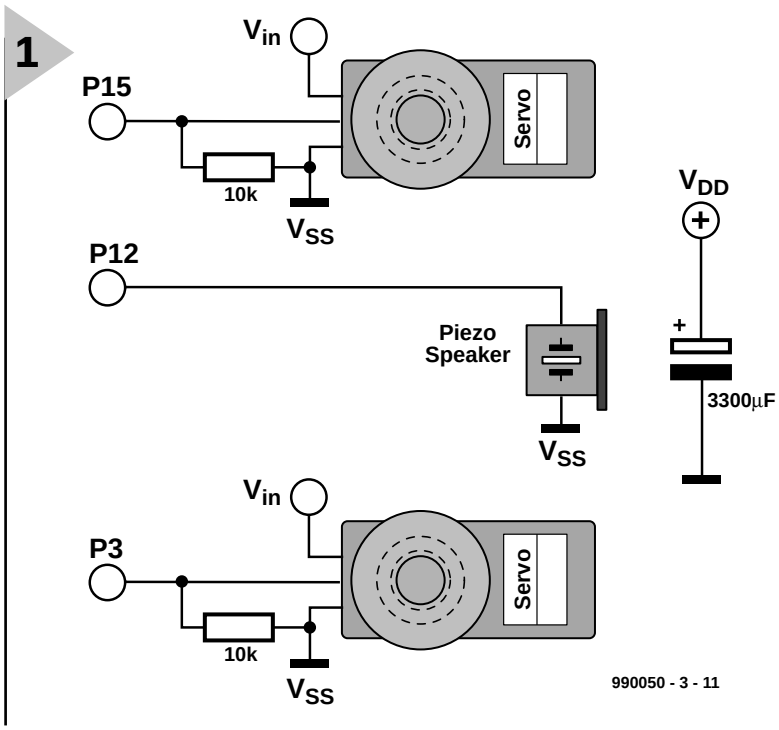


Bild 1. Grundsteuerung und akustische Rückkopplung.

Stückliste

1 BoE-Bot
1 Piezosummer
1 Elko 3300 μF
2 Widerstände 10 kΩ

Achtung
Beachten Sie die wichtige Nachlese zum BoE-Bot in Readers´ Corner!

Entwurf von Chuck Schoeffler, Ph. D. und Ken Gracey (Parallax)

Bewegung mit Subroutinen und Speicher

Bewegung ist eine der hervorragendsten Eigenschaften eines Roboters und außerdem ideal, um ein einfaches PBASIC-Programm zu strukturieren und zu schreiben. Das Experiment zeigt, wie man den BoE-Bot bewegt, ohne dazu einen Sensor-Eingang zu benötigen. Um das Programm zu strukturieren, das den BoE-Bot wie gewünscht bewegt, müssen Subroutinen aufgerufen und Bewegungsmuster im EEPROM gelesen werden. Durch FOR...NEXT-Schleifen erfährt der BoE-Bot, wie weit er fahren muß und wie er physikalisch und im Programm an seine Startposition zurückkehren kann. In der Stückliste finden Sie alle für diesen Kursabschnitt benötigten Teile. Die vollständige "Schaltung" ist in **Bild 1** zu sehen.

SERVO-STEUERUNG

Servos sind Regelelemente, die fortwährend die durch den BASIC-Befehl PULSOUT befohlene Position mit ihrer aktuellen vergleichen, die sie durch das Widerstandsverhältnis eines Potis ermitteln, das mechanisch mit der Servo-Achse verbunden ist. Wenn eine mehr als kleine Differenz zwischen den Positionen besteht, bemüht sich die Elektronik des Servos, diesen Fehler zu eliminieren.

Die Servos wurden im letzten Kursteil so modifiziert, daß das Getriebe stoppt, wenn die BASIC-stamp Impulse in einer Länge von 1500 µs sendet. Ein PULSOUT-Wert von 750 entspricht genau dieser Zeit, da der Befehl in Einheiten von 2 µs arbeitet. Ein geringerer Wert als 750 dreht den Servo links, ein höherer Wert rechts herum. Bei einem Wert, der nahe bei 750 liegt (etwa 760), dreht der Servo ganz langsam. **Bild 2** stellt ein Timing-Diagramm der Pulsbreiten-Modulation dar.

Eine FOR-NEXT-Schleife kann genutzt werden, um zu demonstrieren, wie unterschiedliche Impulsbreiten die Servo-Geschwindigkeit beeinflussen. Stellen Sie den BoE-Bot auf die Front oder legen etwas unter, damit er nicht davonrollen kann. Laden Sie das Programm **Listing 1** in die BASIC-stamp. **Bild 3** zeigt den Zusammenhang zwischen Impulsbreite und Drehgeschwindigkeit eines Servos des Typs Futuba S-148.

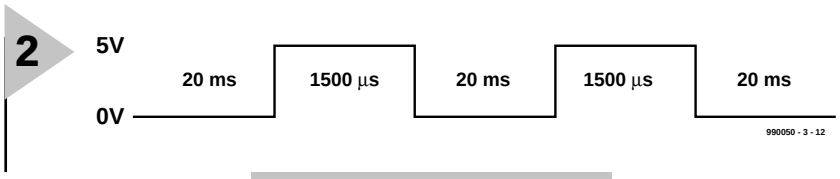


Bild 2. Servosteuerung mit einem pulsbreitenmodulierten Signal.

Programm-Listing 1

```
left_servo con 15
right_servo con 3
x var word
pause 2000
start:
for x = 650 to 850                    ´Beginn der Routine
  pulsout left_servo,x                ´Pulsbreite 1500 us
  pulsout right_servo,1500-x          ‘Pulsbreite 1500 us
  pause 20                            ‘Pause 20 ms
next
```

Programm-Listing 2

```
Hz var word
for Hz = 1 to 4000 step 1000
  freqout 12,70,Hz,4000-Hz     ‘ Generiert zwei 70-ms-Töne an P12
next
```

SOUND-RÜCKMELDUNG

Der BASIC-stamp-Befehl FREQOUT kann genutzt werden, um eine zusätzliche akustische Rückmeldung des BoE-Bots zu erhalten. Wie alle PBASIC-Befehle besitzt auch dieser eine bestimmte Syntax. Um einen Ton zu erzeugen, laden Sie folgende Programmzeile zur BASIC-stamp:

```
FREQOUT 12,750,2000    ´750 ms
  2000 Hz Ton an P12
```

Laden Sie das Programmbeispiel **Listing 2**, um mehr “Roboter-Sound” zu erzeugen.

Diese Routine beginnt mit der Deklaration einer Wort-Variablen mit der Bezeichnung Hz. Eine Wort-Variable kann einen Wert zwischen 0 und 65536 erhalten. Die Schleife wird insgesamt viermal durchlaufen [(4000-1/1000] und erzeugt pro Durchlauf zwei Frequenzen an Port P12. Die erste Frequenz wird von 1 Hz auf 4000 Hz erhöht, die zweite fällt von 4000 Hz auf 1 Hz, jeweils in 1000-Hz-Schritten. Sounds wie dieser können dem Programm an jeder Stelle hinzugefügt werden.

GOTO-BEFEHL

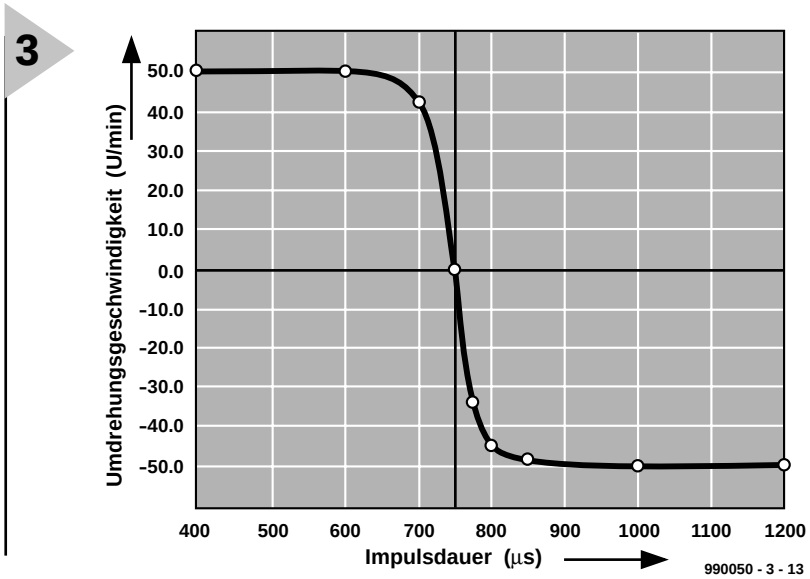
Normalerweise werden PBASIC-Programme Zeile für Zeile abgearbeitet. Der GOTO-Befehl allerdings verursacht einen Sprung zu der Programmstelle, deren Namen hinter dem GOTO-Befehl genannt wird. Ob diese Programmstelle vor oder hinter dem GOTO-Befehl liegt, ist dabei unerheblich. Die Syntax ist, wie man sieht, sehr einfach:

```
GOTO FORWARD   ´springe zur
  Routine Forward
```

MIT GOTO VERWANDT: GOSUB

Der Befehl GOSUB (Goto Subroutine) ist ein naher Verwandter von GOTO. Er bewirkt ebenfalls einen Sprung zu einer Routine irgendwo im Programm, die auf den GOSUB-Befehl folgende Zeile wird allerdings als Rücksprungadresse vorgemerkt, so daß das Programm nach Beendigung der Subroutine automatisch auf den RETURN-Befehl zu dieser Zeile zurückspringt und dort fortfährt.

Bild 3. Rotationsgeschwindigkeit in Abhängigkeit von der Impulslänge.



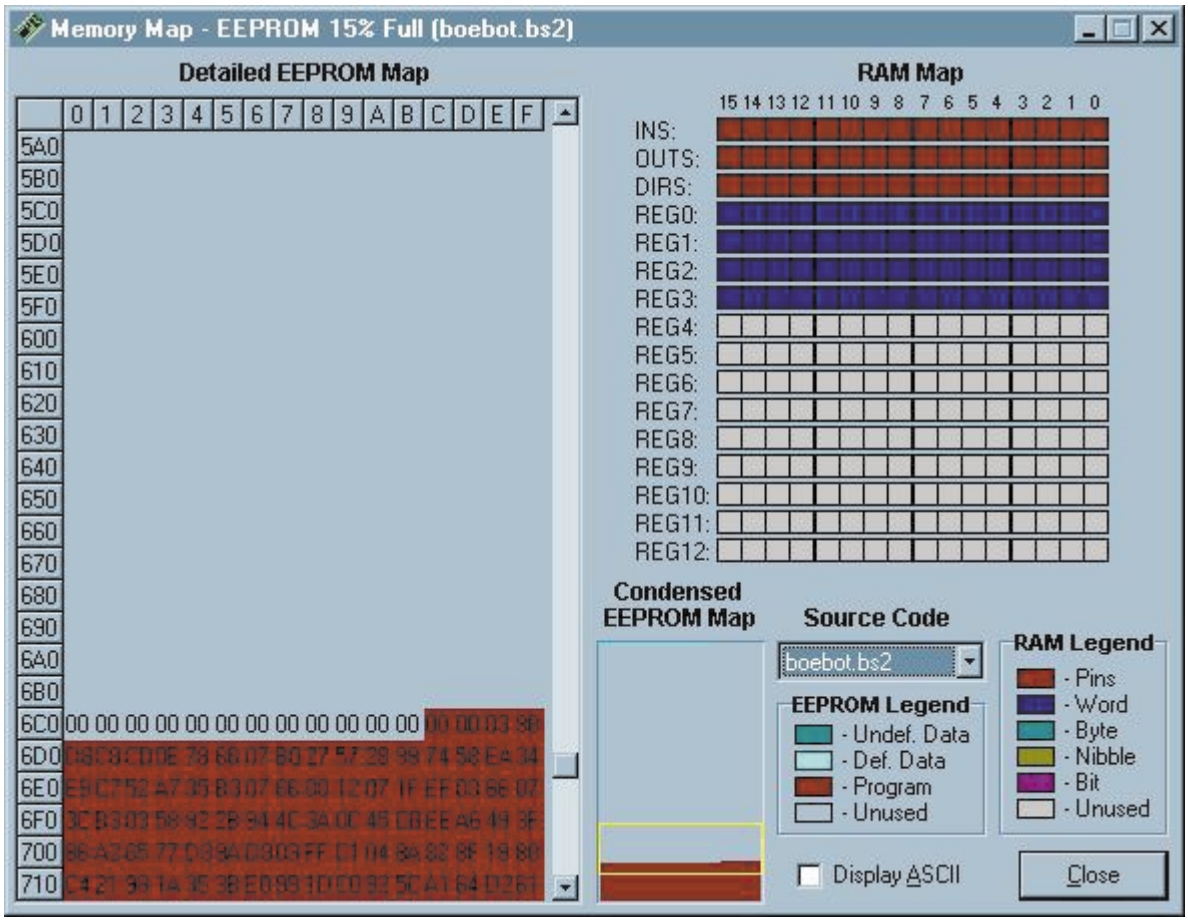


Bild 4. Memory-map des Windows-Editors.

Mit dem GOSUB-Befehl kann man oft benötigten Routinen von verschiedenen Programmeilen anspringen und zum Ursprung zurückkehren. Das folgende Beispiel illustriert die Verwendung des GOSUB-Befehls:

```
GOSUB RIGHT
PAUSE 1000
GOSUB RIGHT
END
RIGHT:
FOR X=1 TO 18
   PULSOUT LEFT_SERVO,650
   PULSOUT RIGHT_SERVO,650
PAUSE 20
RETURN
```

Dieses Beispiel zeigt die Routine RIGHT, die zweimal mit einer Pause von einer Sekunde ausgeführt wird. Der GOSUB-Befehl kann vierfach verschachtelt werden, so daß jedes RETURN einen Sprung zum Befehl hinter dem zuletzt genannten GOSUB bewirkt.

BEWEGUNGEN SPEICHERN MIT DATA UND EEPROM

Die BASIC-stamp verfügt über ein 2K großes EEPROM, das sowohl für das Anwendungsprogramm (das abwärts

Programm-Listing 3

```
‘ Program Listing 3
´BoE-Bot-Programm für Bewegung und Sound

´Definition der Wort-Variablen und Konstanten
‘_____________________________-
x var word               ´Schleifenzähler für Pulsout
position var word        ´Adreßzähler für EEPROM
direction var word       ´Wertspeicher im EEPROM
Hz var word              ´Frequenzvariable
right_servo con 3        ´Rechter Servo an P3
left_servo con 15        ´Linker Servo an P15
speed con 40             ‘Addierter oder substrahierter Wert

‘_____________________________-
´Programmierte Bewegungsmuster
_____________________________-
data ”FRFRFRBBTFE”       ´Gespeicherte Bewegungen

‘_____________________________-
´Hauptprogramm
‘_____________________________-
position=0                        ´Starten bei EEPROM-Zelle 0
move:                             ´Hauptschleife
read position,direction           ´Richtungsbefehl lesen
position=position+1               ‘Zu nächster Zelle erhöhen
  if direction=”E” then quit      ‘Entscheidung, welche Aktion
                                  ‘unternommen wird
  if direction=”F” then forward   ‘Durch Vergleich der Befehls-
                                  ‘Buchstaben
  if direction=”R” then right
  if direction=”L” then left
  if direction=”B” then backward
  if direction=”T” then turn_around
goto move                         ‘Wiederholen, bis E erscheint
```

Distanz angenähert

Es ist ganz einfach, die zurückgelegte Strecke des BoE-Bots abzuschätzen. Dazu benötigt man lediglich den Radumfang und das Verhältnis zwischen PULSOUT-Wert (Pulsbreite) und Umdrehungsgeschwindigkeit. Der Radumfang U beträgt

$$U = 2 \cdot \pi \cdot r = \pi \cdot d$$

$$U = 3{,}14159 \cdot 6{,}67\ cm = 21\ cm$$

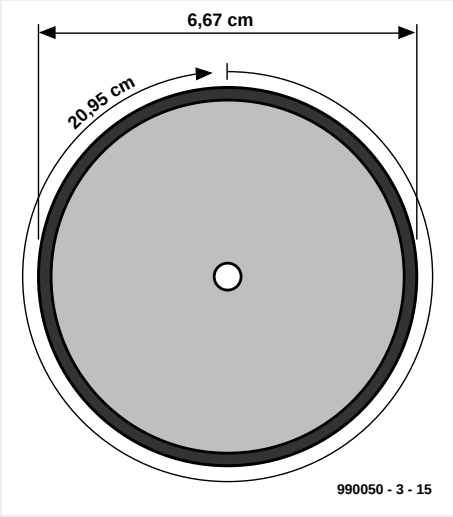
Nun kommt das Verhältnis Pulsbreite und Umdrehungsgeschwindigkeit ins Spiel. Erzeugt zum Beispiel ein PULSOUT-Befehl von 850 genau 50 Umdrehungen pro Minute (0,83 s^{-1}), ergibt sich eine Geschwindigkeit von

21 Zentimeter pro Umdrehung · 0,83 Umdrehungen pro Sekunde = 17,5 cm/s

Um 100 cm zurückzulegen, muß sich der BoE-Bot also

100 cm / 17,5 cm/s = 5,7 s

fortbewegen. Wenn jeder Impuls etwa 1,5 ms dauert und zwischen den Impulsen 20 ms Pause ist, nimmt jede Schleife eine Ausführungszeit von etwa 23 ms (1,5 ms + 1,5 ms + 20 ms) oder 0,023 s in Anspruch. Um 100 cm zurückzulegen, müssen demnach 247 dieser Schleifen (5,7 s / 0,023 s/Schleife = 247 Schleifen) durchlaufen werden. Entsprechend sind die FOR...NEXT-Schleifen im Programm einzustellen:




```
FORWARD:
FOR X=1 TO 247
       PULSOUT LEFT_SERVO,850
       PULSOUT RIGHT_SERVO,850
       PAUSE 20
NEXT
```

```
'-------------------------------
'Sound Routinen
'-------------------------------
forward_sound:
for Hz = 1 to 4000 step 1000
  freqout 12,70,Hz,4000-Hz
next
return

back_sound:
for Hz = 4000 to 6000 step 1000
  freqout 12,70,Hz,Hz-400
next
return

right_sound:
  freqout 8,800,2500
return

left_sound:
  freqout 8,800,4500
return
'-------------------------------
'Bewegungen
'-------------------------------
forward:
gosub forward_sound
for x=1 to 60
  pulsout left_servo,750-speed
  pulsout right_servo,750+speed
pause 20
next
goto move

backward:
gosub back_sound
for x=1 to 60
  pulsout left_servo,750+speed
  pulsout right_servo,750-speed
pause 20
next
goto move

right:
high 0
gosub right_sound

for x=1 to 18
  pulsout left_servo,750-speed
  pulsout right_servo,750-speed
pause 20
next
low 0
goto move

left:
high 14
gosub left_sound

for x=1 to 18
  pulsout left_servo,750+speed
  pulsout right_servo,750+speed
pause 20
next
low 14
goto move

turn_around:
for x=1 to 30
  pulsout left_servo,850
  pulsout right_servo,850
pause 20
next
goto move

quit:
end
'-------------------------------
```

von Adresse 2047 an gespeichert wird) als auch als Datenspeicher (aufwärts von Adresse 0 bis 2047) genutzt werden kann. Sollten Daten mit dem Quellkode kollidieren, wird das Programm nicht mehr ordnungsgemäß ausgeführt. Jede Speicherstelle ist ein Byte breit. Dies ist zwar nicht ausreichend Raum, um einen komplexen Datenlogger zum Beispiel für Umweltmessungen zu realisieren, aber sicherlich genug, um ein paar Bytes mit wichtigen Informationen abzulegen, die im Programmablauf benötigt werden.

Das EEPROM der BASIC-stamp unterscheidet sich in einigen Punkten von der Speicherung von RAM-Variablen:

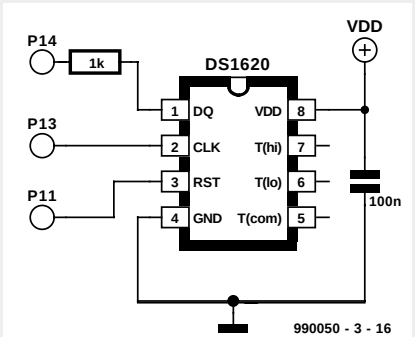
- Die Zugriffszeit auf das EEPROM ist deutlich länger, manchmal sogar einige Millisekunden.
- Es können nur etwa 10 Millionen EEPROM-Schreibzugriffen durchgeführt werden, ganz im Gegensatz zum RAM, das unbegrenzte Schreib/Lesezugriffe erlaubt.
- Die Hauptaufgabe des RAMs ist die Programmspeicherung. Die Daten sollen nur im übrigleibenden Speicherraum untergebracht werden.

Drei Befehle werden im Zusammenhang mit dem EEPROM gebraucht, nämlich DATA, READ und WRITE. Die im EEPROM gespeicherten Daten werden ab der linken oberen Ecke der

Synchrone serielle Kommunikation mit dem Digitalthermometer DS1620

Um Datenbytes mit der BASIC-stamp zu senden oder zu empfangen, stehen die Befehle SHIFTIN und SHIFTOUT zur Verfügung. Wenn Sie sich mit serieller Kommunikation beschäftigen möchten, versuchen Sie dieses Projekt mit dem Board of Education. Dazu benötigen Sie folgende Teile:

einen Temperatursensor DS1620 (Dallas Semiconductor)
einen 1-kΩ-Widerstand
einen 0,1-µF-Kondensator




```
x     var byte         ‘ Definition der Allzweckvariablen im Byte-Format
degC  var byte         ‘ Definition der Grad-Celsius-Variablen im Byte-Format
                       ‘ Beachte: DS1620 im Modus 2 vorprogrammiert
outs=%0000000000000000‘ Definition des Anfangszustandes aller Anschlüsse fedcba9876543210
dirs=%1111111111111111‘ als Low-Ausgänge
freqout 0,20,3800      ‘ Ton zur Betriebsanzeige
high 11                              ‘ DS1620 selektieren
  shiftout 14,11,lsbfirst,[238]      ‘ Befehl “start conversions” senden
low 11                               ‘ Befehl ausführen
  loop:                              ‘ Einmal pro Sekunde anzeigen
  high 11                            ‘ DS1620 selektieren
  shiftout 14,13,lsbfirst,[170]      ‘ Befehl “get data” senden
  shiftin 14,13,lsbpre,[x]           ‘ Daten erfassen
  low 11                             ‘ Ende des Befehls
  degC=x/2                           ‘ Daten in Grad Celsius umrechnen
  debug ? degC                       ‘ Ergebnis auf PC-Monitor darstellen
  pause 1000                         ‘ 1 Sekunde Pause
goto loop                            ‘ Temperatur lesen und anzeigen
```

Ist das Programm erst einmal gestartet, muß der Schreibbefehl genutzt werden, um Temperaturwerte im EEPROM zu speichern, der Lesebefehl, um gespeicherte Daten zu lesen und im Terminal anzuzeigen.

Memory-map bei Position (0,0) in **Bild 4** abgelegt, der Quellkode Reihe für Reihe aufwärts ab der unteren rechten Position (16,128).
Der Windows-Editor der BASIC-stamp stellt eine solche Memory-map automatisch zusammen (Run/Memory/Map).
Die Syntaxen für den Lesebefehl READ und den Schreibbefehl WRITE sind identisch:

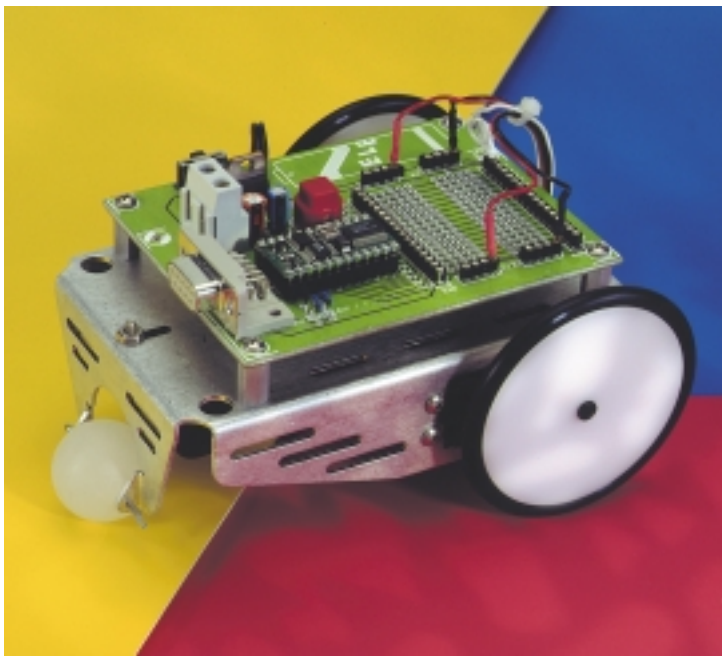
```
WRITE 0,100    ´100 in EEPROM-
  Position Byte 0 schreiben

READ 0,x       ´EEPROM-
  Position Byte 0 lesen und
  Wert in Variablen x
  speichern

DEBUG dec ? x  ´Wert von x im
  Editor anzeigen
```

KOMBINATIONEN

Listing 3 verknüpft die drei Konzepte miteinander: Sound, Bewegung und Geschwindigkeit. Um die Geschwindigkeit effektiv zu nutzen, muß man (das heißt, die BASIC-stamp) stets über die exakte Mittenposition der Servos informiert sein. Eine langsame Verschiebung des Mittenwertes von 750 (1500 µs) ist durchaus möglich. Das Programm-**Listing 1** kann genutzt werden, um die Mittenwerte exakt zu identifizieren. Bei höheren Geschwindigkeiten fallen geringe Abweichungen nicht auf, bei geringer Fahrt allerdings würde sich der BoE-Bot langsam zur Seite bewegen, wenn eine Mittenposition nicht genau stimmt. Das Programm-Listing 3 muß nicht eingetippt werden, wenn man über einen Internet-Zugang verfügt und es von der Site *www.stampsinclass.com* herunterlädt.



(990050-3)rg