3 Entfernung

Wie weit ist das weg? Ultraschall-Entfernungsmesser gehören zu den Sensoren, die wir in unseren Kursen über eingebettete Systeme am häufigsten verwenden. Ein Roboter muss erkennen können, dass er einem Hindernis zu nahe kommt, damit er ihm ausweichen kann. Und ist es nicht auch viel bequemer, einfach nur mit der Hand zu wedeln, anstatt eine Taste zu drücken? Eine Alarmanlage kann Eindringlinge anhand von geänderten Abständen oder Wärmeverteilungsmustern erkennen. Vielleicht haben Sie bei sich zu Hause, im Büro oder in der Schule eine solche Alarmvorrichtung.

Die beiden am häufigsten eingesetzten Verfahren zur Messung von Entfernungen sind Echos von Schallsignalen und Lichtreflexion. Um niemanden durch ständiges Piepsen und Blinken zu stören, wird dabei gewöhnlich eine Schall- bzw. Lichtfrequenz gewählt, die Menschen nicht hören bzw. nicht sehen können, also hochfrequenter *Ultraschall* bzw. niederfrequentes *Infrarot (IR)*.

> Infrarot ist zwar unsichtbar, aber wir werden Ihnen zeigen, wie Sie es mit handelsüblichen Gegenständen beobachten können.

Ein Ultraschallsensor kann genaue Entfernungsangaben machen und Ihnen beispielsweise mitteilen, dass der Abstand zu einem Objekt 36 cm beträgt.

Um die *Nähe* von Menschen und anderen Lebewesen zu erkennen, gibt es auch Sensoren, die die von ihnen ausgesandte Wärmestrahlung erfassen. Dadurch können Sie die Anwesenheit von warmen Objekten im Messbereich feststellen, aber nicht die genaue Entfernung. Wärme kann auf verschiedene Weise übertragen werden: durch Wärmeleitung (Konduktion), Wärmeströmung (Konvektion) und Wärmestrahlung. Ein *passiver Infrarotsensor* misst Wärmestrahlung in Form von infrarotem Licht.

Anders als diese passiven Sensoren strahlen *aktive Infrarot-Entfernungssensoren* unsichtbares IR-Licht aus und prüfen, ob es reflektiert wird. Dadurch können sie ermitteln, ob irgendetwas einen gegebenen Abstand unterschreitet. Beispielsweise kann ein aktiver Infrarotsensor Ihnen mitteilen, ob ein Objekt näher

ist als 30 cm, aber er weiß nicht, ob es 5 cm oder 29 cm entfernt ist. Nur einige wenige Sensoren sind in der Lage, Entfernungen anhand des reflektierten Infrarotlichts abzuschätzen.

Gängige Einsatzbereiche für aktive Infrarotsensoren sind die automatischen Wasserhähne und Händetrockner in öffentlichen Toiletten. Auch manche Mülleimer öffnen ihren Deckel, wenn Sie ihnen nahe kommen. Infrarotlicht sorgt für mehr Hygiene, da sie keine Objekte berühren müssen, die schon viele andere Menschen angefasst haben.

In Langstrecken-Entfernungsmessern werden auch Laserstrahlen zur Messung von Abständen eingesetzt. Dabei wird die Entfernung anhand der Lichtgeschwindigkeit und der Zeit berechnet, die die Reflexion des Lichtstrahls gedauert hat. Da Licht sehr schnell ist, müssen solche Geräte in der Lage sein, die Zeit äußerst präzise zu messen. Das macht sie ziemlich teuer (die Preise fangen bei ca. 100 € an). Für die Prototypentwicklung mit dem Arduino oder Raspberry Pi werden sie weit seltener eingesetzt als Schall- und IR-Sensoren.

3.1 Experiment: Abstände mit Ultraschall messen (Ping)

Ping ... 1, 2, 3 ... pong! Ein Ultraschallsensor sendet einen Ton aus und misst dann, wie lange es dauert, bis der Schall reflektiert wird. Da bekannt ist, dass sich Schall mit einer Geschwindigkeit von ca. 330 m/s bewegt, kann ein Programm aus dieser Zeit die Entfernung berechnen.

Heutzutage gibt es viele preisgünstige Ultraschallsensoren, die dem Ping-Sensor von Parallax nachempfunden sind (siehe Abb. 3–1). Weiter hinten in diesem Kapitel finden Sie auch Code für einen dieser preiswerten Sensoren, den HC-SR04 (siehe Abschnitt 3.2). Um mit den ganzen Ping-ähnlichen Sensoren umgehen zu können, ist es jedoch sinnvoll, sich zunächst einmal mit dem Original vertraut zu machen, weshalb wir Ihnen im nächsten Abschnitt Code dafür zeigen. Außerdem haben viele Universitäten sowie Bastlerclubs oder HackerSpaces Ping-Sensoren in ihrer Sammlung – auch deshalb ist es gut, sich damit auszukennen.

> Wie ein Ultraschallsensor Entfernungen misst, erfahren Sie in Abschnitt 3.2.3.

Ping ist ein älterer, weitverbreiteter Sensor von Parallax. Im Vergleich mit den Alternativen ist er mit ca. 30 € recht teuer. Wenn Sie viele Abstandssensoren brauchen, können Sie auch auf die billigeren ausweichen, aber wenn Sie nur einen kaufen wollen, bietet sich der Ping an. Der ähnliche HC-SR04 kostet nur 1 bis 2 €, und der einzige Konfigurationsunterschied zwischen Ping und HC-SR04 besteht in einem Pin (der HC-SR04 verwendet einen Pin, um den Sendeimpuls auszulösen, und einen anderen, um das Echo zu lesen). Der Code für diese beiden Sensoren ist fast identisch.

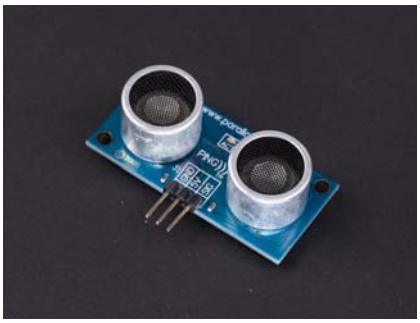

Abb. 3–1 *Der Ping-Sensor*

3.1.1 Code und Schaltung für den Ping am Arduino

Abbildung 3–2 zeigt den Anschluss des Ping-Sensors an den Arduino. Bauen Sie die Schaltung, kompilieren Sie dann in der Arduino-IDE den Code und laden Sie ihn hoch.

> Den Beispielcode können Sie von *http://www.dpunkt.de/sensoren* herunterladen.

Um die Messwerte anzuzeigen, verwenden Sie den seriellen Monitor (*Arduino IDE > Tools > Serial Monitor*). Falls statt Text nur wirres Zeug erscheint, prüfen Sie, ob Sie im Code (`Serial.begin`) und im seriellen Monitor der Arduino-IDE dieselbe Geschwindigkeit (in bit/s oder Baud) angegeben haben.

Obwohl das Beispiel sehr viel Code umfasst, lässt es sich leicht zur Entfernungsmessung in Ihren eigenen Projekten verwenden. Kopieren Sie einfach die Kernteile des Codes (die Funktion `distanceCm()` und die globalen Variablen) in Ihr eigenes Programm. Dann können Sie Abstände mit der folgenden Codezeile messen:

```
int d=distanceCm();
```

Da der Ping auf das Echo des ausgesandten Schalls lauscht, ist seine Platzierung sehr wichtig. Wenn Sie immer denselben Messwert erhalten (z. B. immer 2 cm), prüfen Sie, ob der Schallkegel nicht an unvorhergesehener Stelle reflektiert wird, etwa am Steckbrett oder am Tisch. Setzen Sie den Ping am besten am Rand des Steckbretts ein, damit Letzteres keine Echos verursacht.

Mit einem Servo-Verlängerungskabel (männlich/weiblich) können Sie den Ping auch bequem weiter weg vom Arduino platzieren. Da der Ping nur drei Pins aufweist, ist ein solches Kabel ideal geeignet.

Listing 3–1 zeigt den vollständigen Code zum Lesen der Werte von einem Ping-Ultraschallsensor.

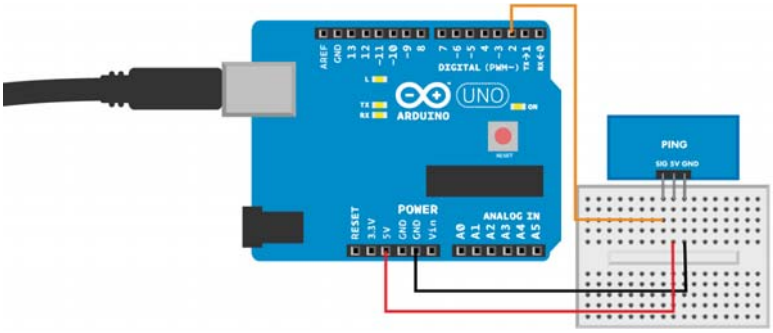

Abb. 3–2 *Ping-Schaltung für den Arduino*

```
// distance_ping.ino - Abstandsmessung mit Ping-Ultraschallsensor
// (c) BotBook.com - Karvinen, Karvinen, Valtokari

int pingPin = 2;
float v=331.5+0.6*20; // m/s                1

void setup()
{
   Serial.begin(115200);
}
float distanceCm(){
   // Sendet Schallimpuls
   pinMode(pingPin, OUTPUT);             2
   digitalWrite(pingPin, LOW);
   delayMicroseconds(3);                 3
   digitalWrite(pingPin, HIGH);
   delayMicroseconds(5);                 4
   digitalWrite(pingPin, LOW);

   // Lauscht auf Echo
   pinMode(pingPin, INPUT);
   float tUs = pulseIn(pingPin, HIGH); // microseconds        5
   float t = tUs / 1000.0 / 1000.0 / 2; // s        6
   float d = t*v; // m                   7
   return d*100; // cm
}

void loop()
{
   int d=distanceCm();                   8
   Serial.println(d, DEC);               9
   delay(200); // ms                     10
}
```

Listing 3–1 *distance_ping.ino*

1 Berechnet die Schallgeschwindigkeit v für eine Umgebungstemperatur von 20 °C. Wenn die Umgebungstemperatur erheblich davon abweicht, ersetzen Sie 20 durch den entsprechenden Celsius-Wert. Die Geschwindigkeit beläuft sich auf ca. 340 m/s oder 1200 km/h.

2 Ping verwendet denselben Pin sowohl als Eingang als auch als Ausgang.

3 Wartet, bis der Pin einen stabilen Zustand erreicht hat; 1 µs = 1 Millionstel Sekunde = 1×10^{-6} s = 0,000001 s.

4 Sendet einen sehr kurzen Schallimpuls von 5 µs = 5×10^{-6} s.

5 Misst (in Mikrosekunden), wie lange es dauert, bis `pingPin` (D2) den Zustand `LOW` einnimmt.

6 Rechnet die Werte in die SI-Basiseinheit Sekunde um (*Système international*, siehe *http://de.wikipedia.org/wiki/Internationales_Einheitensystem*). Beachten Sie, dass wir statt des Integers 1000 die Fließkommazahl 1000.0 verwenden, um als Ergebnis eine Fließkommazahl zu bekommen. Die hier berechnete Zeit für eine Richtung beträgt die Hälfte der Zeit für den Hin- und Rückweg des Schallsignals.

7 Entfernung ist Zeit mal Geschwindigkeit.

8 Misst den Abstand und speichert ihn in der neuen Variablen `d`. So sollten Sie es auch in Ihrem eigenen Code machen.

9 Gibt den Wert von `d` auf dem seriellen Monitor aus.

10 Bauen Sie in Schleifen immer Verzögerungszeiten ein. Wenn Sie den Sketch ohne Pause laufen lassen, strapazieren Sie damit die CPU des Arduino und verschwenden Strom (alles so schnell wie möglich zu erledigen, kann auf einem Einkernprozessor 100 % des verfügbaren Stroms in Anspruch nehmen).

3.1.2 Code und Schaltung für den Ping am Raspberry Pi

Bauen Sie die Ping-Schaltung für den Raspberry Pi wie in Abbildung 3–3 und führen Sie dann den in Listing 3–2 angegebenen Code aus.

Seien Sie bei Anschlüssen an der GPIO-Leiste sehr vorsichtig. Eine falsche Verbindung kann sehr leicht einen Pin oder (schlimmstenfalls) den gesamten Raspberry Pi zerstören. Um Probleme zu vermeiden, sollten Sie den Pi von der Stromversorgung trennen, während Sie Anschlüsse herstellen oder ändern, und alle Pinverbindungen vor dem Einschalten genau prüfen.

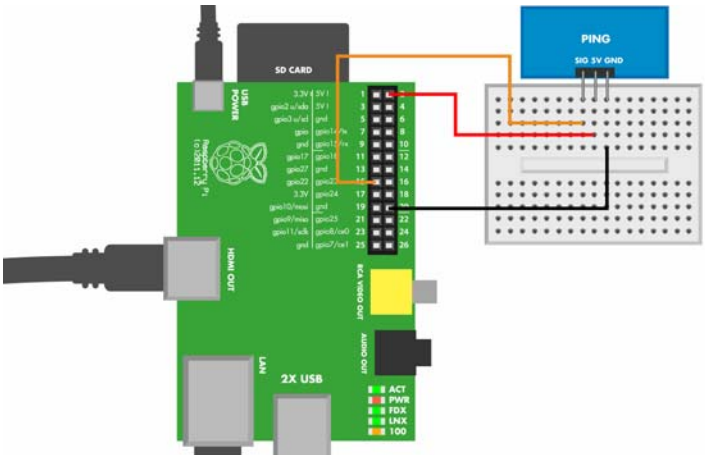

Abb. 3–3 *Ping-Schaltung für den Raspberry Pi*

```
# distance_ping.py - Gibt Entfernung aus
# (c) BotBook.com - Karvinen, Karvinen, Valtokari

import time                 1
import botbook_gpio as gpio         2

def readDistanceCm():
  sigPin=22
  v=(331.5+0.6*20)

   gpio.interruptMode(sigPin, "both")          3

  gpio.mode(sigPin, "out")              4
  gpio.write(sigPin, gpio.LOW)          5
  time.sleep(0.5) # s

  gpio.write(sigPin, gpio.HIGH)         6
  time.sleep(1/1000.0/1000.0)           7
  gpio.mode(sigPin, "in")               8

 # Liest hohe Pulsbreite
  t = gpio.pulseInHigh(sigPin) # s      9
  d = t*v
  d = d/2                               10
 return d*100 # cm

def main():
  d = readDistanceCm()                  11
  print "Distance is %.2f cm" % d       12
  time.sleep(0.5)

if __name__ == "__main__":
  main()
```

Listing 3–2 *distance_ping.py*

1 Der Import der Bibliothek `time` erstellt den gleichnamigen *Namensraum* für die Funktionen dieser Bibliothek. Diese Zeile erlaubt es Ihnen also, weiter hinten im Code `time.sleep(1)` aufzurufen.

2 Um eine Bibliothek importieren zu können, muss sie sich in demselben Verzeichnis befinden wie der Code. Achten Sie also darauf, dass *botbook_gpio.py* im selben Verzeichnis steht wie *distance_ping.py*. Diese Bibliothek finden Sie im Beispielcode auf *http://makesensors.botbook.com*. (Weitere Informationen darüber, wie Sie den Raspberry Pi für den Zugriff auf die GPIO-Pins konfigurieren, finden Sie in Abschnitt 1.4.)

3 Der Interruptmodus `both` bedeutet, dass `pulseInHigh()` einen vollständigen Impuls vom Anstieg (von 0 auf 1) bis zum Abfall misst (von 1 zurück auf 0).

4 Für den Ping-Sensor schalten wir ein und denselben Pin nach Bedarf auf `"out"` und `"in"`. Andere Sensoren, z. B. der HC-SR04, verwenden unterschiedliche Pins für die einzelnen Funktionen.

5 Schaltet den Pin aus und wartet darauf, dass er einen stabilen Zustand annimmt. Mit einer Wartezeit von einer halben Sekunde sind wir auf der sicheren Seite.

6 Startet den Impuls (Anstieg). Hier beginnt der zeitkritische Code.

7 Wartet eine Mikrosekunde (1×10^{-6} s), also eine Millionstel Sekunde.

8 Versetzt den Pin in den Modus `"in"`. Als Nebenwirkung wird der Impuls dadurch abgeschaltet, sodass die Kurve abfällt.

9 Liest die Pulsbreite in Sekunden. `gpio.pulseInHigh()` misst die Gesamtlänge des Impulses vom Anfang (Anstieg) bis zum Ende (Abfall). Da auf dem Raspberry Pi ein komplettes Betriebssystem ausgeführt wird und andere Programme die zeitliche Abstimmung stören können, ist sie nicht so genau wie auf dem Arduino.

10 Die Zeit für eine Richtung beträgt die Hälfte der Zeit für den Hin- und Rückweg des Schallsignals.

11 Diese Zeile brauchen Sie, um in Ihren eigenen Programmen Entfernungen zu messen.

12 Gibt die Entfernung auf dem Terminalfenster aus, in dem das Programm läuft. Die Angabe `%.2f` gehört zum Formatierungsstring und dient als Platzhalter für die Variable `d`. `%f` steht für eine (dezimale) Fließkommazahl, und `.2` gibt an, wie viele Dezimalstellen angezeigt werden sollen. Wenn Sie nur `print d` schreiben, erhalten Sie eine sehr lange Dezimalzahl.

3.2 Ultraschallsensor HC-SR04

Der HC-SR04 ähnelt dem Ping, ist aber für einen Bruchteil des Preises erhältlich. Auch der Code ist fast identisch mit dem für den Ping, wobei der HC-SR04 jedoch getrennte Pins zum Auslösen des Schallsignals und zum Lauschen auf das Echo verwendet. Ausführliche Erläuterungen zum Code erhalten Sie in Abschnitt 3.1.1 bzw. Abschnitt 3.1.2. Die Erklärungen in diesem Abschnitt beschränken sich auf die Unterschiede zwischen den Sensoren.

Wie ein Ultraschallsensor Entfernungen misst, erfahren Sie in Abschnitt 3.2.3.

Abb. 3–4 *Der Ultraschallsensor HC-SR04*

3.2.1 Code und Schaltung für den HC-SR04 am Arduino

Bauen Sie die Schaltung wie in Abbildung 3–5 gezeigt und laden Sie den Code hoch.

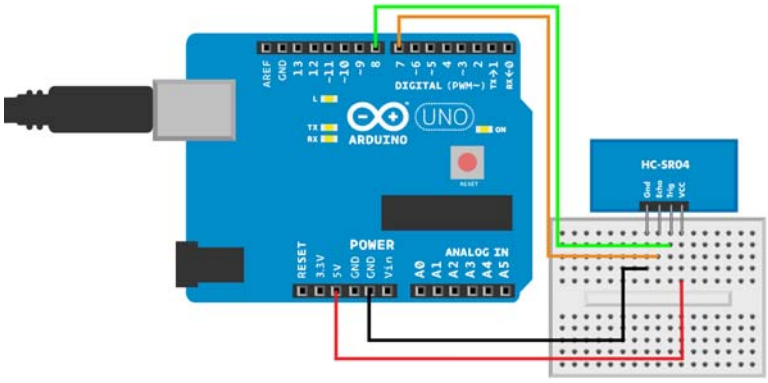

Abb. 3–5 *HC-SR04-Schaltung für den Arduino*

```
// hc_sr04.ino - Gibt Entfernung auf dem seriellen Monitor aus
// (c) BotBook.com - Karvinen, Karvinen, Valtokari

int trigPin = 8;
int echoPin = 7;
float v=331.5+0.6*20; // m/s

void setup()
{
   Serial.begin(115200);
   pinMode(trigPin, OUTPUT);            1
   pinMode(echoPin, INPUT);             2
}

float distanceM(){
   // Sendet Schallimpuls
   digitalWrite(trigPin, LOW);
   delayMicroseconds(3);
   digitalWrite(trigPin, HIGH);
   delayMicroseconds(5);
   digitalWrite(trigPin, LOW);

   // Lauscht auf Echo
   float tUs = pulseIn(echoPin, HIGH); // Mikrosekunden
   float t = tUs / 1000.0 / 1000.0 / 2; // s
   float d = t*v; // m
   return d*100; // cm
}

void loop()                             3
{
   int d=distanceM();
   Serial.println(d, DEC);
   delay(200); // ms
}
```

Listing 3–3 *hc-sr04.ino*

1 Beim Ping sind wir nicht der üblichen Vorgehensweise gefolgt, den Pinmodus in der Funktion `setup()` festzulegen, da wir ihn ständig ändern mussten, denn der Ping verwendet denselben Pin, um den Impuls auszulösen und auf das Echo zu lauschen. Beim HC-SR04 dient der Pin `Trig` dazu, das Schallsignal auszulösen.

2 Der Pin `Echo` gibt die Dauer bis zum Eintreffen des Echos als Pulsbreite zurück.

3 Abgesehen von den Änderungen in `setup()` sieht das Hauptprogramm bei der Verwendung des HC-SR04 größtenteils genauso aus wie beim Ping.

3.2.2 Code und Schaltung für den HC-SR04 am Raspberry Pi

Bauen Sie die Schaltung (siehe Abb. 3–6) und laden Sie den Code aus Listing 3–4 hoch. Beachten Sie, dass Sie neben den Jumperkabeln auch zwei 10-kΩ-Widerstände benötigen. (Um die richtigen Widerstände zu finden, können Sie die Methode aus dem Kasten »Der Trick mit dem dritten Ring« aus Abschnitt 1.3.7 anwenden.) Der Code ist dem für den Ping sehr ähnlich.

```
# hc-sr04.py - Gibt die Entfernung zu einem Ort in cm aus

# (c) BotBook.com - Karvinen, Karvinen, Valtokari

import time
import botbook_gpio as gpio

def readDistanceCm():
      triggerPin = 22                          1
      echoPin = 27

      v=(331.5+0.6*20) # m/s

      gpio.mode(triggerPin, "out")

      gpio.mode(echoPin, "in")
      gpio.interruptMode(echoPin, "both")

      gpio.write(triggerPin, gpio.LOW)
      time.sleep(0.5)

      gpio.write(triggerPin, gpio.HIGH)
      time.sleep(1/1000.0/1000.0)
      gpio.write(triggerPin, gpio.LOW)

      t = gpio.pulseInHigh(echoPin) # s
      d = t*v
      d = d/2
      return d*100 # cm


def main():
      d = readDistanceCm()                     2
      print "Distance is %.2f cm" % d
      time.sleep(0.5)


if __name__ == "__main__":
      main()
```

Listing 3–4 *hc-sr04.py*

1 Der einzige Unterschied zum Ping-Beispiel besteht darin, dass der HC-SR04 zwei Pins verwendet (`Trig` und `Echo`).

2 Das Ergebnis berechnen Sie beim HC-SR04 genauso wie beim Ping.

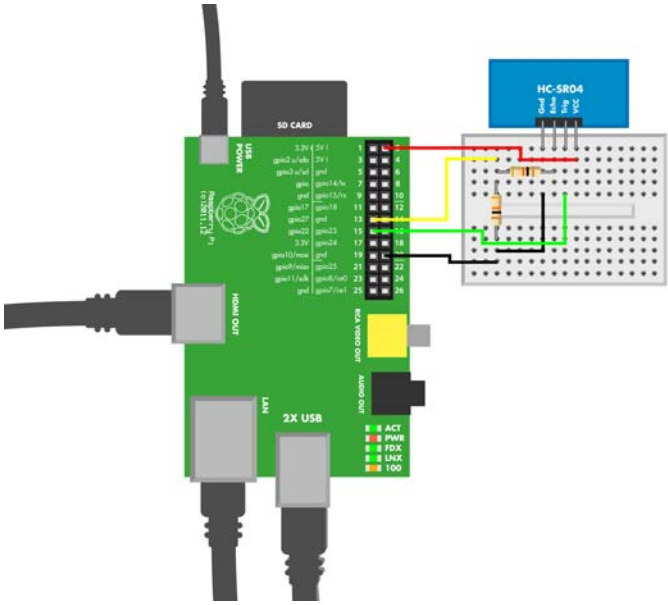

Abb. 3–6 *HC-SR04-Schaltung für den Raspberry Pi*

Warum braucht der HC-SR04 einen Schutzwiderstand, der Ping aber nicht? Laut Datenblatt hat der HC-SR04 einen Ausgang mit *TTL-Pegel,* also mit +5 V, während das Datenblatt für den Ping eine Kompatibilität mit dem Logikpegel von +3,3 V verspricht. Durch Messung der Ausgänge konnten wir beide Maximalwerte bestätigen. Die GPIO-Pins des Raspberry Pi aber sind für eine Maximalspannung von +3,3 V ausgelegt, deshalb können sie beschädigt werden, wenn eine Spannung von +5 V angelegt wird.

3.2.3 Echoberechnungen

Bei einem Gewitter können Sie die Entfernung zu der Stelle, an der der Blitz einschlägt, anhand der Zeit bestimmen, die zwischen Blitz und Donner vergeht. Zählen Sie die Sekunden nach dem Aufflammen des Blitzes. Jede Sekunde entspricht einer Entfernung von 330 m zum Blitz. (Die genaue Zahl hängt von der Lufttemperatur ab, worüber wir in Kürze noch sprechen werden.)

Jetzt müssen wir ein bisschen rechnen: 330 m/s × 3 s = 990 m. Das heißt, dass sich der Schall in drei Sekunden ungefähr einen Kilometer fortbewegt. Das Licht sehen wir praktisch sofort, aber der Schall braucht eine gewisse Zeit, bis er unser Ohr erreicht.

Ultraschallsensoren können gewöhnlich Entfernungen zwischen 3 cm und 6 m messen. Ultraschall-Entfernungsmesser aus dem Werkzeugfachhandel haben eine größere Reichweite von ca. 20 m, da sie über einen Kegel zur Schallprojektion und ein Thermometer zur Kalibrierung auf die Schallgeschwindigkeit in der aktuellen Lufttemperatur verfügen.

Um einen Zentimeter weit zu kommen, braucht der Schall nur sehr kurze Zeit, nämlich 30 Mikrosekunden, also 30 Millionstel einer Sekunde. Wie kommen wir auf diese Zahl?

Bei schwierigen Problemen, insbesondere solchen, bei denen es um sehr kleine oder sehr große Zahlen geht, ist es oft hilfreich, sie anhand eines vergleichbaren Problems mit vertrauten, alltäglichen Größenordnungen zu betrachten. Nehmen wir beispielsweise an, ich fahre zwei Stunden lang (Zeit *t*) mit einer Geschwindigkeit *(v)* von 50 km/h. Das klingt nach nervtötender Fahrerei, aber immerhin können wir mit *t* und *v* die Entfernung *d* berechnen:

```
t = 2 h

v = 50 km/h

d = t*v
  = 2 h * 50 km/h
  = 2 * 50 km * h/h = 100 km
```

Das war einfach! Wir wissen jetzt also, dass zwei Stunden in diesem System 100 km entsprechen. Nun wenden wir diese Formel auf sehr kurze Zeitdauern (3,33 Millisekunden bzw. ms) und sehr viel höhere Geschwindigkeiten an und benutzen die Basiseinheiten Meter und Sekunde. (Das Präfix *milli* bedeutet ein Tausendstel, eine Millisekunde ist also eine Tausendstelsekunde.)

```
t = 3,33 ms = 0,00333 s

v = 330 m/s

d = t * v = 0,00333 s * 330 m/s = 1,10 m
```

Auf diese Weise berechnen Sie in Ihrem Programm die Entfernung anhand der Messwerte des Ultraschallsensors: Wenn der reflektierte Schall 3,33 ms gebraucht hat, um zum Sensor zurückzukehren, hat er 1,1 m zurückgelegt.

Andere Personen verwenden in ihrem Code manchmal auch eine »Schrittdauer« für Schall. Statt mit der Geschwindigkeit in Metern pro Sekunde wird in vielen Programmen mit dem Kehrwert in Sekunden pro Meter gerechnet (oder wie im folgenden Beispiel in Millisekunden pro Meter):

```
1/v = 1/(330 m/s) = 0,00303 s/m = 3,03 ms/m
```

Der Schall braucht also ca. 3 ms, um einen Meter zurückzulegen.

Wenn es warm ist, bewegt sich der Schall schneller vorwärts. Schließlich ist Schall nichts anderes als eine Luftschwingung, und solche Schwingungen können sich besser fortpflanzen, wenn die Luftmoleküle bereits durch die Wärme in

Schwingungen versetzt sind. Wenn Sie in einer warmen Gegend leben, dann beneiden wir Sie, weil Sie weniger Kalibrieraufwand für die Messung betreiben müssen. Hier im Norden von Finnland kann es drinnen +22 °C sein und draußen -40 °C. Eine so gewaltige Temperaturdifferenz von über 60 °C wirkt sich deutlich auf die Messgenauigkeit aus. Die folgende Formel gibt den Zusammenhang zwischen der Temperatur T und der Schallgeschwindigkeit v an:

```
v = (331,3 + 0,606 * T) m/s
```

Mit dieser Formel können Sie einen für praktische Anwendungen ausreichend genauen Wert für die Schallgeschwindigkeit berechnen (343 m/s bei 20 °C). Wenn Sie Spaß daran haben, können Sie auch noch eine Kalibrierung für viele weitere Faktoren vornehmen. Beispielsweise können Sie auch noch den Luftdruck berücksichtigen, wenn Sie auf einen Berg steigen oder sich in einem U-Boot aufhalten. Wenn Sie mitten in der Sahara eine Wäscherei betreten, bietet sich auch eine Kalibrierung nach der Luftfeuchtigkeit an. Allerdings nehmen handelsübliche Ultraschall-Entfernungsmesser nur eine Kalibrierung nach der Temperatur vor.

In einem Programm sollten Sie solche Berechnungen an den Anfang des Codes stellen. Der Arduino und der Raspberry Pi können die Werte augenblicklich ausrechnen, und Berechnungen außerhalb von `loop()` werden ohnehin nur ein einziges Mal durchgeführt. Versehen Sie solche Berechnungen auf jeden Fall mit Kommentaren. Wenn Sie einige Wochen später versuchen, aus Ihrem Code schlau zu werden, werden Sie dankbar dafür sein.

3.2.4 Praxisexperiment: Unsichtbare Objekte

Einen Ultraschallsensor können Sie ganz einfach täuschen, sodass er glaubt, es befände sich nichts vor ihm. Montieren Sie den Sensor an einer »dritten Hand« und richten Sie ihn auf ein massives, glattes Objekt. Laden Sie den Code hoch und öffnen Sie den seriellen Monitor, wie Sie es schon bei den früheren Versuchen in diesem Kapitel getan haben. Sie sollten jetzt normale Entfernungsangaben lesen können.

Stellen Sie nun ein weiches Kissen, ein Plüschtier oder Ähnliches zwischen den Sensor und das massive Objekt (siehe Abb. 3–7). Zeigt der serielle Monitor das massive Objekt immer noch an?

Eine weitere Achillesferse für Ultraschallsensoren sind schiefe Ebenen. Nehmen Sie das weiche Objekt wieder weg und beginnen Sie das glatte Objekt vor dem Sensor zu kippen. Beobachten Sie dabei den seriellen Monitor.

Warum treten diese Phänomene auf? Weiche Objekte (wie unser Monty-Python-Killerkaninchen in Abbildung 3–7) schlucken so viel Schall, dass es kein ausreichendes Echo mehr gibt. Schiefe Ebenen reflektieren zwar den Schall, aber

in die falsche Richtung (nicht zurück zum Sensor). Auf eine ähnliche Weise narren Stealth-Flugzeuge das gegnerische Radar.

Abb. 3–7 *Test des Ping-Sensors mit einem weichen Objekt*

3.3 Experiment: Hindernisse mit Infrarot erkennen (IR-Abstandssensor)

Ein Infrarotsensor (siehe Abb. 3–8) ist zuverlässiger als ein Ultraschallsensor, aber weniger vielseitig. Sie können ihn zwar nicht so leicht in die Irre führen, aber dafür kann er Ihnen auch nur mitteilen, dass etwas da ist, aber nicht, wie weit es entfernt ist. Da die Sonne sehr viel Infrarotlicht abgibt, kann ein Infrarotsensor auch leicht von ihr geblendet werden.

Abb. 3–8 *Ein Infrarotsensor*

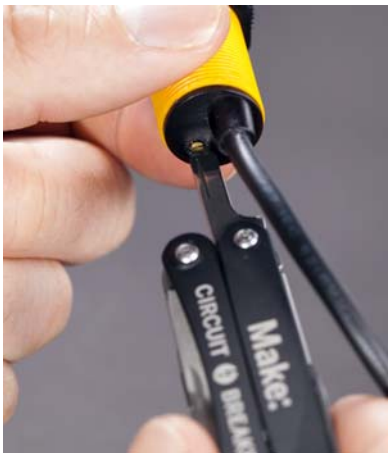

Abb. 3–9 *Sie können die Entfernung einstellen, bis zu der der Sensor Hindernisse erkennt.*

3.3.1 Code und Schaltung für den IR-Sensor am Arduino

Abbildung 3–10 zeigt, wie Sie den Infrarotsensor an den Arduino anschließen. Den Sketch sehen Sie in Listing 3–5.

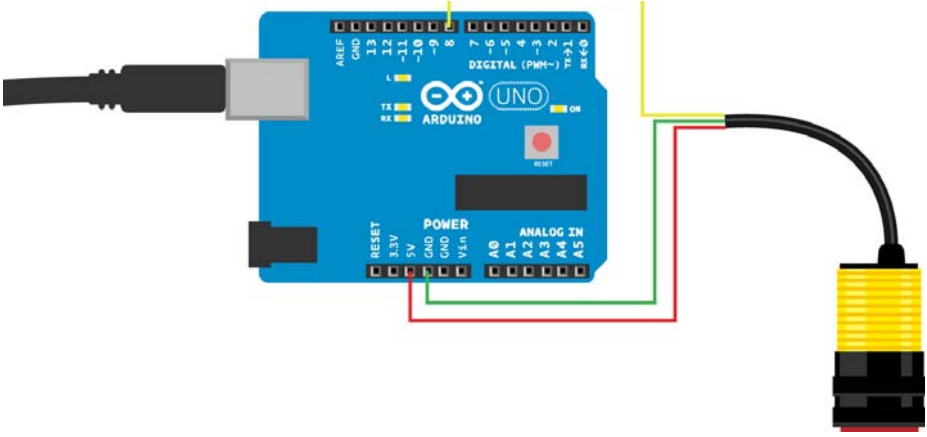

Abb. 3–10 *Anschluss des Infrarotsensors an den Arduino*

```
// adjustable_infrared_sensor_switch.ino - Gibt bei einem erkannten Objekt
// ein Ergebnis an den seriellen Monitor aus und lässt die LED aufleuchten.
// (c) BotBook.com - Karvinen, Karvinen, Valtokari


const int sensorPin = 8;
const int ledPin = 13;

//Sensorwert
int switchState = 0;

void setup() {
   Serial.begin(115200);                    1
   pinMode(sensorPin, INPUT);
   pinMode(ledPin, OUTPUT);
}

void loop() {
   switchState = digitalRead(sensorPin);          2
   Serial.println(switchState);             3
   if(switchState == 0) {
      digitalWrite(ledPin, HIGH);
      Serial.println("Object detected!");         4
   } else {
      digitalWrite(ledPin, LOW);
   }
   delay(10); // ms                         5
}
```

Listing 3–5 *adjustable_infrared_sensor_switch.ino*

1 Öffnen Sie den seriellen Monitor der Arduino-IDE (*Tools > Serial Monitor*). Sie müssen dort dieselbe Geschwindigkeit eingeben wie im Code. Der größte mögliche Wert beträgt 115.200 bit/s. Bei einem unzuverlässigen (oder sehr langen) USB-Kabel sollten Sie auf 9600 bit/s heruntergehen.

2 Ein Infrarotsensor verhält sich ähnlich wie eine Drucktaste. Mit dieser Zeile wird der Sensorwert gelesen.

3 Gibt den Zustand des Sensorpins für Debuggingzwecke aus.

4 Der Zustand `0` bedeutet, dass ein Objekt erkannt wurde. In diesem Fall wird die eingebaute LED des Arduino eingeschaltet.

5 Bauen Sie in `loop()` immer Verzögerungszeiten ein, wenn auch nur sehr kurze. Dadurch verhindern Sie, dass der Sketch den Prozessor des Arduino permanent zu 100 % auslastet.

3.3.2 Code und Schaltung für den IR-Sensor am Raspberry Pi

Abbildung 3–11 zeigt das Schaltdiagramm für den Anschluss des Infrarotsensors an den Arduino. Den zugehörigen Python-Code sehen Sie in Listing 3–6.

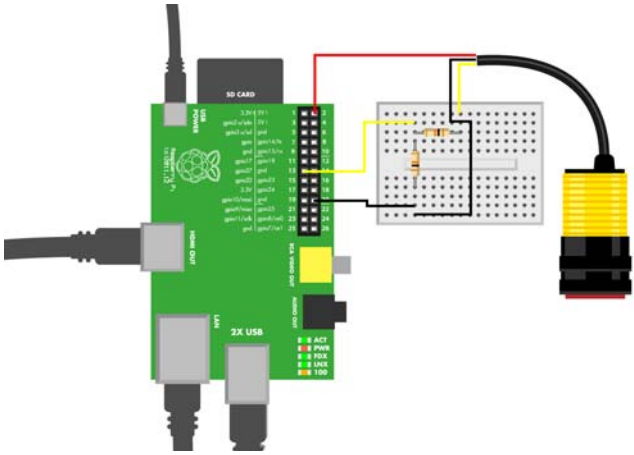

Abb. 3–11 *Anschluss des Infrarotsensors an den Raspberry Pi*

```
# adjustable-infrared-sensor-switch.py - Liest IR-Sensor
# (c) BotBook.com - Karvinen, Karvinen, Valtokari

import time
import botbook_gpio as gpio            1

def main():
   switchPin = 27
   gpio.mode(switchPin, "in")          2
   x = gpio.read(switchPin)            3
   if( x == gpio.LOW ):                4
      print "Something is inside detection range"
   else:
      print "There is nothing inside detection range"
   time.sleep(0.1)

if __name__ == "__main__":
   main()
```

Listing 3–6 *adjustable-infrared-sensor-switch.py*

1 Importiert die Bibliothek `botbook_gpio`. Dazu muss sich *botbook_gpio.py* im selben Verzeichnis befinden wie *adjustable-infrared-sensor-switch.py*. Diese Bibliothek finden Sie im Beispielcode auf *http://www.dpunkt.de/sensoren*. (Weitere Informationen darüber, wie Sie den Raspberry Pi für den Zugriff auf die GPIO-Pins konfigurieren, finden Sie in Abschnitt 1.4.)

2 Konfiguriert den Pin, mit dem der Sensor verbunden ist, als Eingang.

3 Liest den Zustand des Pins und speichert ihn in der Variablen `x`.

4 Wenn der Pin `LOW` ist, bedeutet das, dass sich ein Objekt in der Reichweite aufhält.

3.4 Praxisexperiment: Infrarotlicht sichtbar machen

Wie bereits erwähnt, liegt Infrarot nicht im Bereich des sichtbaren Lichts. Wenn Sie wollen, können Sie es jedoch sichtbar machen, und zwar entweder mit einem Nachtsichtgerät oder auch ganz einfach mit einer billigen Digitalkamera.

Versuchen Sie, durch die Kamera eines Smartphones auf den IR-Sensor zu blicken. Am IR-Sender sollten Sie ein violettes Glühen erkennen können (siehe Abb. 3–12). Dämpfen Sie die Beleuchtung und schließen Sie die Vorhänge, um den Effekt noch zu verstärken. Das Ergebnis ist zwar nicht so aufsehenerregend wie bei Nachtsichtgeräten, aber immerhin können Sie damit auf schnelle und einfache Weise überprüfen, ob der Sender funktioniert.

Wenn Sie das Gleiche mit einer teuren digitalen Spiegelreflexkamera versuchen, haben Sie allerdings das Problem, dass solche Kameras über starke Infrarotfilter verfügen, die unerwünschte Wellenlängen aus Ihren Fotos fernhalten. (Bei billigen Kameras funktioniert es, da deren Infrarotfilter nicht so gut sind, weshalb genügend Infrarotlicht durchkommt, um die Sensoren anzuregen.) Allerdings können Sie dann in einem abgedunkelten Raum mit einem Stativ und einer langen Belichtungszeit (am besten mehrere Sekunden) ein Foto von dem Sensor aufnehmen. Da die Infrarotstrahlung die einzige Beleuchtung im Raum ist, wird sie schließlich auf dem Bild erscheinen.

Falls Sie über ein Nachtsichtgerät verfügen, können Sie damit ganz hervorragend beobachten, wie IR-Sensoren funktionieren. Solche Geräte verstärken nicht nur das sichtbare Licht, sondern sind auch besonders gierig nach Strahlung im infraroten Bereich des Spektrums. Die billigsten Modelle verlassen sich ganz auf die IR-Strahlung, die sie selbst aussenden. Wenn Sie ein Modell mit IR-Sender haben, müssen Sie ihn ausschalten (oder abkleben), um den schwächeren Schein des IR-Abstandssensors erkennen zu können. Besonders interessant bei Nachtsichtgeräten ist es, dass Sie damit nicht nur feststellen können, dass Ihr Sensor funktioniert, sondern auch die Reflexionen des IR-Lichts an anderen Objekten erkennen können (siehe Abb. 3–13).

Abb. 3–12 *Ein IR-Sensor aus der Sicht einer Handykamera*

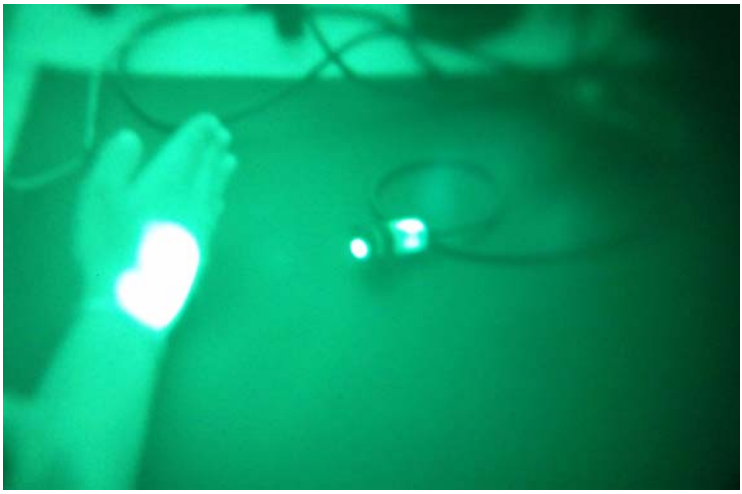

Abb. 3–13 *Ein IR-Sensor, betrachtet durch ein Nachtsichtgerät*

3.5 Experiment: Bewegungen mit Infrarot verfolgen (IR-Facettenauge)

Das als Facettenauge bezeichnete Bauteil verfügt über viele infrarotempfindliche Transistoren und LEDs. Es kann Bewegungen bis zu einem Abstand von 20 cm verfolgen. Zwar wird es insgesamt als ein einziger Sensor betrachtet, doch können die einzelnen IR-empfindlichen Transistoren getrennt voneinander abgelesen

werden. Die Korrektur für das Umgebungslicht erfolgt dadurch, dass die IR-LEDs abgeschaltet und die Werte verglichen werden.

Abb. 3–14 *Das IR-Facettenauge*

Der vollständige Name dieses Sensors lautet »IR-Facettenauge« (IR Compound Eye). Wenn Sie eigene Sensoren entwickeln und verkaufen wollen, sollten Sie ihnen neben einer solchen allgemeinen Bezeichnung auch einen einprägsamen Namen geben. Dadurch wird es viel einfacher, nach dem Bauteil zu suchen.

Abb. 3–15 *Ein Facettenauge in der Natur*

Wenn Sie die Messung mit dem Facettenauge verbessern wollen, müssen Sie den Sensor kalibrieren. Warten Sie, bis es Nacht wird und es kein IR-Umgebungslicht mehr gibt. Wenn Sie nicht so lange warten wollen, sollten Sie in den Keller oder in einen fensterlosen Raum gehen, da IR-Licht auch durch geschlossene Rollos dringt. Bauen Sie die Schaltung aus Abbildung 3–16. Halten Sie ein Blatt Papier vor den Sensor (etwa 20 cm entfernt) und prüfen Sie, wie stark die Werte an den einzelnen Pins voneinander abweichen. Mit dem Papier sollten die Werte fast identisch sein (+/- 100). Ist einer der Werte zu hoch, dämpfen Sie den IR-Lichteinfall mit undurchsichtigem Klebeband oder einem Schrumpfschlauch. Sollte ein Wert zu niedrig sein, schirmen Sie die anderen Sensoren ein wenig vor dem IR-Licht ab.

Dieser Sensor führt eine analoge Widerstandsmessung durch. Das einfachste Beispiel für eine solche Messung finden Sie in Abschnitt 5.3.

3.5.1 Code und Schaltung für das Facettenauge am Arduino

Abbildung 3–16 zeigt, wie Sie das Facettenauge an den Arduino anschließen. Den Arduino-Sketch finden Sie in Listing 3–7.

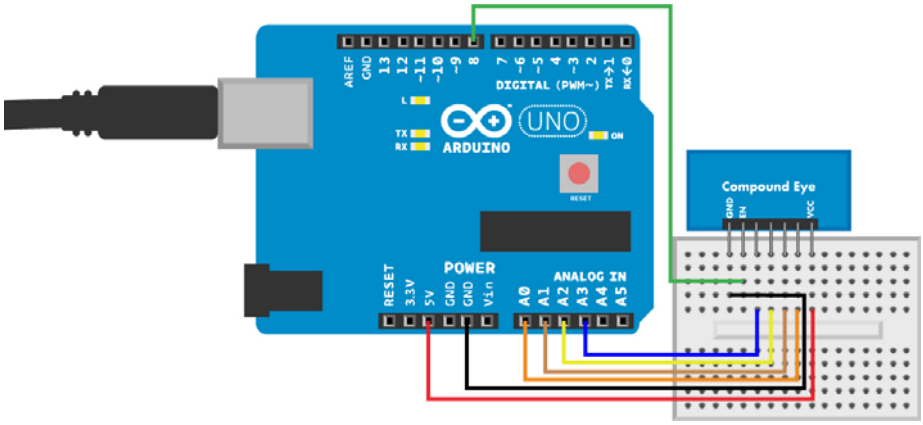

Abb. 3–16 *Anschluss des Facettenauges an den Arduino*

```
//compound_eye.ino - Gibt Abstands- und Richtungswerte auf dem seriellen
// Monitor aus
// (c) BotBook.com - Karvinen, Karvinen, Valtokari

const int irEnablePin = 8;                    1
const int irUpPin = 0;
const int irDownPin = 2;
const int irLeftPin = 1;
```

```
const int irRightPin = 3;
int distance = 0;                                   2
int irUpValue = 0;
int irDownValue = 0;
int irLeftValue = 0;
int irRightValue = 0;

void setup() {
   Serial.begin(115200);
   pinMode(irEnablePin, OUTPUT);
}

void loop() {
   readSensor();                                    3
   Serial.print("Values: ");                        4
   Serial.print("irUpValue"); Serial.print(irUpValue); Serial.print(",");
   Serial.print("irDownValue"); Serial.print(irDownValue); Serial.print(",");
   Serial.print("irLeftValue"); Serial.print(irLeftValue); Serial.print(",");
   Serial.print("irRightValue"); Serial.print(irRightValue);
   Serial.print(",");
   Serial.print("distance"); Serial.println(distance);
delay(100);
}
void readSensor() {
   digitalWrite(irEnablePin, HIGH);                 5
   delay(5); // ms                                  6
   irUpValue = analogRead(irUpPin);
   irDownValue = analogRead(irDownPin);
   irLeftValue = analogRead(irLeftPin);
   irRightValue = analogRead(irRightPin);

   int ambientLight = 0;                            7
   digitalWrite(irEnablePin, LOW);                  8
   delay(5);
   ambientLight = analogRead(irUpPin);              9
   irUpValue = irUpValue - ambientLight;           10

   ambientLight = analogRead(irDownPin);
   irDownValue = irDownValue - ambientLight;

   ambientLight = analogRead(irLeftPin);
   irLeftValue = irLeftValue - ambientLight;

   ambientLight = analogRead(irRightPin);
   irRightValue = irRightValue - ambientLight;

   distance = (irUpValue+irDownValue+irLeftValue+irRightValue) / 4;       11
}
```

Listing 3–7 *compound_eye.ino*

1 Die Pinnummern werden als Konstanten deklariert (`const`), sodass Sie sie an anderer Stelle im Code nicht mehr ändern können. (Wenn Sie – versehentlich – Code schreiben sollten, der sie neu zuzuweisen versucht, erhalten Sie beim Prüfen oder schon beim Hochladen des Sketches eine Fehlermeldung.)

2 Die Sensorwerte werden in globalen Variablen gespeichert, sodass sie in allen Funktionen zur Verfügung stehen. In C und C++ (den Programmiersprachen, auf denen die Arduino-IDE beruht) gilt es als empfohlene Vorgehensweise, Variablen gleich bei der Deklaration zu initialisieren (z. B. `int foo = 0`).

3 `readSensor()` gibt keine Werte zurück, sondern bearbeitet globale Variablen. Dadurch können Sie bei der Ausführung dieser Funktion mehrere Werte ändern.

4 Gibt das Ergebnis aus. `Serial.print()` fügt keinen Zeilenumbruch hinzu (im Gegensatz zu `Serial.println()`).

5 Schaltet die IR-LED ein, um das Ziel für die Messung zu beleuchten.

6 Wartet darauf, dass die Eingänge einen stabilen Zustand annehmen.

7 Beginnt mit der Messung des Umgebungslichts (z. B. der unsichtbaren IR-Strahlung von der Sonne).

8 Schaltet die IR-LED aus. Alles jetzt empfangene Licht stammt aus Quellen in der Umgebung.

9 Misst mithilfe der einzelnen IR-empfindlichen Transistoren das Umgebungslicht.

10 Subtrahiert den Wert für das Umgebungslicht von den Messwerten der einzelnen Sensoren.

11 Berechnet den Mittelwert des Abstands über alle vier IR-empfindlichen Transistoren.

3.5.2 Code und Schaltung für das Facettenauge am Raspberry Pi

Das IR-Facettenauge enthält acht infrarotempfindliche Sensoren, die paarweise verbunden sind, sodass Sie unter dem Strich mit vier Sensoren arbeiten. Jeder davon wird als analoger Widerstandssensor abgelesen.

Zur Nutzung der IR-Sensoren benötigt der Raspberry Pi einen externen Analog-Digital-Wandler (Analog-to-Digital Converter, ADC). Da ein MCP3002-Chip zwei analoge Eingänge lesen kann, wir es aber mit vier Sensoren zu tun haben, brauchen wir zwei MCP3002.

Die Schaltung (die Sie in Abb. 3–17 sehen) weist viele kleine Einzelheiten auf, doch das Grundprinzip ist ganz einfach: Es gibt vier analoge Widerstandssensoren, die Sie nacheinander ablesen. Bauen Sie die Schaltung wie gezeigt auf und führen Sie dann den Code aus Listing 3–8 aus.

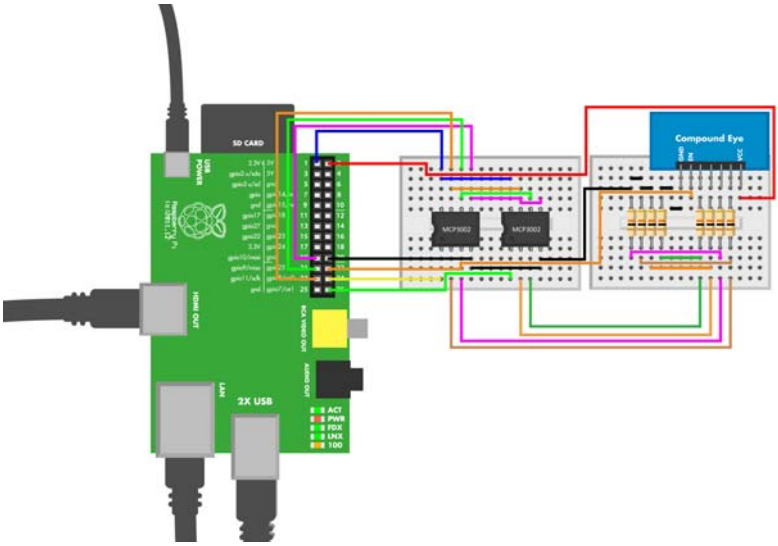

Abb. 3–17 *Anschluss des Facettenauges an den Raspberry Pi*

```
# compound_eye.py - Misst Abstand und Richtung
# (c) BotBook.com - Karvinen, Karvinen, Valtokari

import time
import botbook_gpio as gpio                    1
import botbook_mcp3002 as mcp                  2

irUpValue = 0                                  3
irDownValue = 0
irLeftValue = 0
irRightValue = 0
distance = 0

def readCompoundEye():
   global irUpValue,irDownValue,irLeftValue,irRightValue,distance          4
   ledPin = 25
   gpio.mode(ledPin, "out")                    5
   gpio.write(ledPin, gpio.HIGH)
   #Wartet auf Bereitschaft der Sensoren
   time.sleep(0.05)                            6

   irUpValue = mcp.readAnalog(0, 0)            7
   irDownValue = mcp.readAnalog(1, 0)
   irLeftValue = mcp.readAnalog(0, 1)
   irRightValue = mcp.readAnalog(1, 1)

   ambientLight = 0
   gpio.write(ledPin, gpio.LOW)                8
   time.sleep(0.05)
   ambientLight = mcp.readAnalog(0, 0)         9
```

```
    irUpValue = irUpValue - ambientLight                                    10
    ambientLight = mcp.readAnalog(1, 0)                                     11
    irDownValue = irDownValue - ambientLight
    ambientLight = mcp.readAnalog(0, 1)
    irLeftValue = irLeftValue - ambientLight
    ambientLight = mcp.readAnalog(1, 1)
    irRightValue = irRightValue - ambientLight

    distance = (irUpValue+irDownValue+irLeftValue+irRightValue)/4            12

def main():
    global irUpValue,irDownValue,irLeftValue,irRightValue,distance
    while True:                                                             13
        readCompoundEye()                                                   14
        print "Values:"
        print "Up: %f" % irUpValue
        print "Down: %f" % irDownValue
        print "Left: %f" % irLeftValue
        print "Right: %f" % irRightValue
        print "Distance: %f" % distance
        time.sleep(0.5) # s                                                 15

if __name__ == "__main__":
    main()
```

Listing 3–8 *compound_eye.py*

1 Importiert `gpio`, um die Digitalpins (hier `gpio25`) ein- und ausschalten zu können. Die Datei *botbook_gpio.py* aus dem Beispielcode zu diesem Buch muss sich im selben Verzeichnis befinden wie das Programm *compound_eye.py*.

2 Importiert die Bibliothek `mcp3002`, um mithilfe der MCP3002-ADC-Chips analoge Sensorwerte von den IR-empfindlichen Transistoren zu lesen. Die Bibliothek *botbook_mcp3002.py* muss sich im selben Verzeichnis befinden wie das Programm *compound_eye.py*. Außerdem müssen Sie die Bibliothek `spidev` installieren, die von *botbook_mcp3002.py* importiert wird. Mehr darüber erfahren Sie in den Kommentaren zu Beginn von *botbook_mcp3002/botbook_mcp3002.py* und in Abschnitt 3.5.3.

3 Deklariert globale Variablen.

4 Um die globalen Variablen innerhalb einer Funktion verwenden zu können, müssen sie zu Beginn der Funktion aufgelistet werden.

5 Schaltet den Pin `gpio25` ein, der mit den IR-LEDs verbunden ist. Dadurch wird das zu messende Gebiet beleuchtet.

6 Wartet darauf, dass die Pins einen stabilen Zustand annehmen.

7 Liest die Werte der einzelnen IR-empfindlichen Transistoren. Da der Raspberry Pi keinen eingebauten Analog-Digital-Wandler mitbringt, verwenden wir den externen MCP3002-Chip.

8 Um die Auswirkungen des Umgebungslichts zu messen, schalten wir die IR-LED aus.

9 Der Wert jedes einzelnen IR-empfindlichen Transistors wird erneut gemessen.

10 Der Wert für das Umgebungslicht wird vom eigentlichen Messwert abgezogen (der mit IR-Beleuchtung bestimmt wurde).

11 Der Wert jedes einzelnen IR-empfindlichen Transistors wird erneut gemessen.

12 Der Abstand ist der Durchschnitt der Messungen von allen vier IR-empfindlichen Transistoren.

13 In Anwendungen für eingebettete Systeme ist `while(True)` eine übliche Methode, um eine Aktion dauerhaft auszuführen. Die meisten eingebetteten Geräte sollen ihre Aufgabe ständig erfüllen. Es ist nicht vorgesehen, dass das Programm nach einer Weile aufhört und das Gerät seine Funktion einstellt. Um ein Programm abzubrechen, das in einer `while(True)`-Schleife läuft, drücken Sie in der Terminalsitzung, in der Sie es gestartet haben, (**Strg**) + (**C**).

14 Die Funktion `readSensor()` muss keine Werte zurückgeben, da sie globale Variablen bearbeitet. In Python können Sie alternativ auch mehrere Werte zurückgeben lassen, z. B. `a,b,c=foo()`.

15 Die kleine Verzögerung verhindert, dass diese triviale Schleife 100 % der CPU-Zeit verschlingt.

Das Facettenauge an sich hat schon eine ganze Reihe von Verbindungen, und zusammen mit den Anschlüssen für die ADCs benötigen Sie eine Menge Jumperkabel.

3.5.3 Bibliothek spidev installieren

Der Analog-Digital-Wandler MCP3002 greift auf das Protokoll SPI zurück. Es ist zwar ziemlich kompliziert, aber Sie können die Bibliothek `spidev` installieren, die sich um die Einzelheiten kümmert.

Für jeglichen Code, der `import spidev` verwendet, ist die Bibliothek `spidev` erforderlich. Das gilt unter anderem für den Potenziometercode für den Raspberry Pi sowie für alle in diesem Buch verwendeten analogen Widerstandssensoren, da `spidev` von *botbook_mcp3002* importiert wird.

Öffnen Sie auf dem Raspberry Pi ein Terminalfenster. Als Erstes müssen Sie die erforderlichen Komponenten installieren:

```
$ sudo apt-get update
$ sudo apt-get -y install git python-dev
```

Laden Sie nun die neueste Version von `spidev` von der entsprechenden Website herunter:

```
$ git clone https://github.com/doceme/py-spidev.git
$ cd py-spidev/
```

Installieren Sie die Bibliothek auf Ihrem System:

```
$ sudo python setup.py install
```

Als Nächstes müssen Sie das SPI-Modul auf dem Raspberry Pi aktivieren. Öffnen Sie dazu die Datei */etc/modprobe.d/rspi-blacklist.conf* mit dem Befehl `sudoedit /etc/modprobe.d/raspi-blacklist.conf` und *löschen* Sie die folgende Zeile:

```
blacklist spi-bcm2708
```

Speichern Sie die Datei, indem Sie (**Strg**) + (**X**) drücken, mit (**Y**) bestätigen und dann (**Enter**) oder (**Return**) drücken.

Um den Zugriff auf SPI ohne Root-Berechtigungen zu erlauben, kopieren Sie die Udev-Datei aus Listing 3–9 (oder aus dem Beispielcode im Verzeichnis *botbook_mcp3002*) an die richtige Stelle:

```
$ sudo cp 99-spi.rules /etc/udev/rules.d/99-spi.rules
```

```
# /etc/udev/rules.d/99-spi.rules - SPI ohne Root-Berechtigung
# auf dem Raspberry Pi
# Copyright 2013 http://BotBook.com
SUBSYSTEM=="spidev", MODE="0666"
```

Listing 3–9 *99-spi.rules*

Starten Sie den Raspberry Pi neu, öffnen Sie LXTerminal und vergewissern Sie sich, dass die SPI-Geräte angezeigt werden und die Besitzverhältnisse korrekt eingerichtet sind:

```
$ ls -l /dev/spi*
```

Im Ergebnis sollten zwei Dateien mit den Berechtigungen `crw-rw-rw` aufgeführt werden. Wenn nicht, gehen Sie die vorherigen Schritte noch einmal durch.

Jetzt können Sie den MCP3002-Chip und andere SPI-Geräte am Raspberry Pi verwenden.

3.5.4 Alternative Schaltungen für den Raspberry Pi

Die Schaltung für die Verwendung des Facettenauges am Raspberry Pi ist ziemlich kompliziert. Sie ist zwar nicht schwer verständlich, es sind aber sehr viele Kabel anzuschließen. Um ein einfacheres System zu bauen, können Sie entweder einen anderen ADC oder einen Arduino verwenden (siehe den nachfolgenden Kasten »Pi + Arduino«).

Wenn Sie nur einen einzigen ADC-Chip verwenden möchten, können Sie zum MCP3008 greifen, der über acht Eingänge verfügt. Dazu müssen Sie jedoch die Bibliothek *botbook_mcp3002* ändern. Sie können sich auch den Code von Ada-

fruit für den MCP3008 ansehen (siehe *https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code*).

Pi + Arduino

Wenn die Menge der Jumperkabel für Ihren Geschmack zu groß ist, können Sie die Sensordaten alternativ auch mit dem Arduino auslesen und das Ergebnis dann über eine serielle USB-Verbindung an den Raspberry Pi senden. Eine weitere Möglichkeit bietet die Arduino-kompatible Platine AlaMode, die huckepack auf den Raspberry Pi aufgesetzt wird, um den ultimativen Hybriden zu bauen.

Weitere Informationen erhalten Sie in Abschnitt 12.3.4.

3.6 Testprojekt: Haltungswarner (Arduino)

Jeder Computerfreak kennt dieses Problem: Je intensiver Sie arbeiten, umso näher rückt Ihr Kopf an den Bildschirm heran. Das ist nicht gerade die Haltung, die Mutter Natur für uns vorgesehen hat. Mit einem IR-Abstandsmesser und einem Piezo-Summer können Sie ganz leicht eine technische Spielerei bauen, die Sie warnt, wenn Sie dem Monitor zu nahe kommen (siehe Abb. 3–18).

Abb. 3–18 *Der fertige Haltungswarner*

3.6.1 Lernziele

Im Projekt »Haltungswarner« lernen Sie Folgendes:

- Eingabe, Verarbeitung und Ausgabe kombiniert einsetzen
- Töne mit einem Piezo-Summer abspielen
- Das Projekt in ein Gehäuse einbauen

3.6.2 Piezo-Summer

Wenn Sie an einen piezoelektrischen Kristall eine Spannung anlegen, ändert er seine Form. Mit Wechselstrom oder einem einfachen Ein/Aus-Rechteckimpuls können Sie den Kristall in Schwingungen versetzen. Dadurch beginnt die Luft zu schwingen, und Luftschwingungen äußern sich als Schall – in diesem Fall als nervtötender Ton.

Abb. 3–19 *Ein Piezo-Summer*

Die gängigen Piezoelemente (siehe Abb. 3–19) piepsen, wenn Sie ihnen eine Rechteckschwingung senden. Eine solche Schwingung wechselt ständig zwischen den Zuständen `HIGH` (was beim Arduino 5 V sind) und `LOW` (0 V).

Eine Rechteckschwingung können Sie ganz einfach dadurch erzeugen, dass Sie einen Datenpin mit `digitalWrite` wiederholt ein- und ausschalten. Alternativ können Sie auch die integrierte Funktion `tone()` verwenden, die eine solche Schwingung auf anspruchsvollere und kompliziertere Weise erzeugt.

Der Piezoeffekt funktioniert auch andersherum: Wenn Sie einen Piezokristall zusammendrücken, können Sie dadurch Elektrizität gewinnen. In Elektrofeuerzeugen wird dieser Effekt ausgenutzt, um einen Funken zu erzeugen.

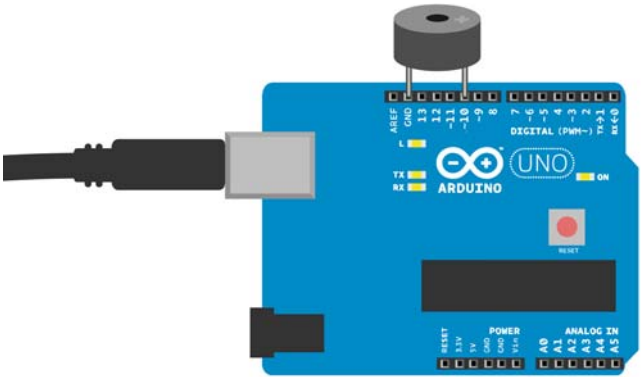

Abb. 3–20 *Anschluss des Piezo-Summers*

```
// piezo_beep.ino - Lässt den Piezo-Summer mit gegebener Frequenz piepsen
// (c) BotBook.com - Karvinen, Karvinen, Valtokari

int speakerPin = 10;

void wave(int pin, float frequency, int duration)          1
{
   float period=1/frequency*1000*1000; // Mikrosekunden    2
   long int startTime=millis();                            3
   while(millis()-startTime < duration) {                  4
      digitalWrite(pin, HIGH);                             5
      delayMicroseconds(period/2);
      digitalWrite(pin, LOW);
      delayMicroseconds(period/2);
   }
}

void setup()
{
   pinMode(speakerPin, OUTPUT);
}

void loop()
{
   wave(speakerPin, 440, 500);                             6
   delay(500);
}
```

Listing 3–10 *piezo_beep.ino*

1 Wenn Sie die Funktion `wave` in Ihren Projekten verwenden, müssen Sie den Pin, die Frequenz und die Dauer in Millisekunden angeben. Der Inhalt der Funktion stellt lediglich die Implementierung dar. Diese Einzelheiten sind zwar wissenswert, genaue Kenntnisse sind für die Verwendung der Funktion aber nicht nötig.

2 Berechnet die Schwingungsdauer oder Periodendauer T, also die Dauer einer kompletten Schwingung (die sich aus einem `HIGH`- und einem `LOW`-Abschnitt zusammensetzt). Die Schwingungsdauer ist der Kehrwert der Frequenz f: $T = 1/f$. Beispielsweise läuft bei zwei Schwingungen pro Sekunde (2 Hertz, also $2 \times 1/s$) eine Schwingung in einer halben Sekunde ab ($1/2$ Hz = $1/(2 \times 1/s)$ = 1/2 s). Die Einheit Hertz (Anzahl der Schwingungen pro Sekunde) wird mit Hz abgekürzt.

3 Dies ist ein übliches Codemuster, um einen Vorgang eine bestimmte Zeit lang auszuführen: Als Erstes wird dabei der Startzeitpunkt in einer Variablen gespeichert. (`millis` gibt die Anzahl der Millisekunden an, die seit dem Hochfahren des Arduino verstrichen sind.)

4 Danach wird gewartet, bis die festgelegte Dauer seit dem Startzeitpunkt überschritten ist.

5 Erstellt eine komplette Schwingung, zunächst den `HIGH`-Abschnitt, dann den `LOW`-Abschnitt.

6 Mehr als dieser Aufruf ist nicht erforderlich, um den Summer in Ihrem Hauptprogramm piepsen zu lassen.

3.6.3 Alarm!

Als Nächstes ändern wir den einzelnen Piepser um in einen Alarm aus einer Folge von Tönen.

```
// piezo_alarmtone.ino - Gibt Alarmtöne über den Piezo-Summer aus
// (c) BotBook.com - Karvinen, Karvinen, Valtokari

int speakerPin = 10;

void wave(int pin, float frequency, int duration)          1
{
   float period=1 / frequency * 1000 * 1000; // Mikrosekunden
   long int startTime=millis();
   while(millis()-startTime < duration) {
      digitalWrite(pin, HIGH);
      delayMicroseconds(period/2);
      digitalWrite(pin, LOW);
      delayMicroseconds(period/2);
   }
}

void setup()
{
   pinMode(speakerPin, OUTPUT);
}
```

```
void loop()
{
   wave(speakerPin, 440, 40);          2
   delay(25);
   wave(speakerPin, 300, 20);
   wave(speakerPin, 540, 40);
   delay(25);
   wave(speakerPin, 440, 20);
   wave(speakerPin, 640, 40);
   delay(25);
   wave(speakerPin, 540, 20);
}
```

Listing 3–11 *piezo_alarmtone.ino*

1 Verwendet die Funktion `wave()`, die Sie bereits zur Erzeugung eines einzelnen Tons genutzt haben.

2 Gibt einen Ton aus, wartet sehr kurze Zeit, gibt den nächsten Ton aus ... und wiederholt diesen Vorgang endlos.

3.6.4 Piezo-Summer und IR-Sensor kombinieren

Bauen Sie die Schaltung für den Haltungswarner nach Abbildung 3–21. Laden Sie den Code hoch und freuen Sie sich auf ein Leben ohne Haltungsschäden ... oder auf einen Daueralarm.

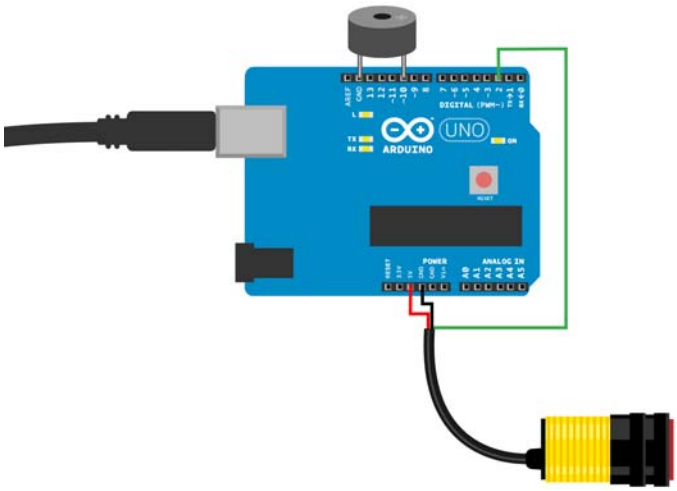

Abb. 3–21 *Der Aufbau des Haltungswarners*

```
// posture_alarm.ino - Lässt einen Alarmton erklingen, wenn der IR-Sensor
// eine schlechte Haltung erkennt
// (c) BotBook.com - Karvinen, Karvinen, Valtokari

int speakerPin = 10;
const int sensorPin = 2;
int switchState = 0;

void wave(int pin, float frequency, int duration)          1
{
   float period=1/frequency*1000*1000; // Mikrosekunden
   long int startTime=millis();
   while(millis()-startTime < duration) {
      digitalWrite(pin, HIGH);
      delayMicroseconds(period/2);
      digitalWrite(pin, LOW);
      delayMicroseconds(period/2);
   }
}

void alarm()       2
{
   wave(speakerPin, 440, 40);
   delay(25);
   wave(speakerPin, 300, 20);
   wave(speakerPin, 540, 40);
   delay(25);
   wave(speakerPin, 440, 20);
   wave(speakerPin, 640, 40);
   delay(25);
   wave(speakerPin, 540, 20);
}

void setup()
{
   pinMode(speakerPin, OUTPUT);
   Serial.begin(115200);
   pinMode(sensorPin, INPUT);
}

void loop()
{
   switchState = digitalRead(sensorPin);
   Serial.println(switchState,BIN);
   if (switchState==0) {            3
      alarm();                      4
   }
   delay(10);
}
```

Listing 3–12 *posture_alarm.ino*

1. Verwendet die Funktion `wave()`, die Sie zuvor für den Einzelton und den Alarm genutzt haben.
2. Verwendet die bereits bekannte Funktion `alarm()`.
3. Ruft wie vorher mithilfe von `digitalRead()` einen Wert vom IR-Abstandssensor ab.
4. Spielt die Alarmtonfolge ab, wenn der Sensor etwas im Bereich des Monitors erkennt, also wenn Ihr Kopf dem Bildschirm zu nahe kommt.

3.6.5 Eine elegante Verpackung für das Projekt

Prototypen wirken eindrucksvoller und sind stabiler, wenn Sie sie in ein Gehäuse einbauen. Wir haben hierfür die Arduino-Verkleidung von SmartProjects verwendet, da sie zufällig genau die richtige Größe für dieses Projekt hat. Als Erstes haben wir den Kasten mit schwarzer Farbe eingesprüht und dann mit einem 19-mm-Bohrer ein Loch für den Infrarotsensor gebohrt (siehe Abb. 3–22).

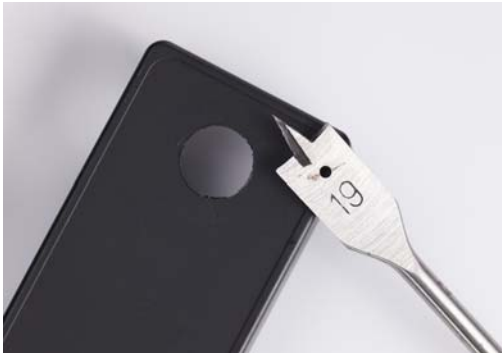

Abb. 3–22 *Das Loch für den Infrarotsensor*

Damit der Sensor besser hineinpasst und wir die Abstandsschraube später leichter einstellen können, haben wir den kleinen Deckel an der Rückseite der Unterschale entfernt (siehe Abb. 3–23).

Anschließend haben wir den Sensor mithilfe der Nuten an seiner Kunststoffumhüllung eingepasst. Den Arduino können Sie über die Kunststoffstifte in die Unterschale schieben, wo er dann fest verankert ist (siehe Abb. 3–24).

Abb. 3–23 *Die Stellschraube für den Abstand können Sie von der Rückseite der Verkleidung aus drehen.*

Abb. 3–24 *Das Innenleben des Haltungswarners*

Schließen Sie das Gehäuse, indem Sie Ober- und Unterschale zusammendrücken (siehe Abb. 3–18), und schon können Sie Ihre Haltung verbessern! Diese Verschalung verfügt über eine Öffnung für ein USB-Kabel, sodass Sie jetzt sogar ein elegantes USB-Gerät zur Verzierung Ihres Arbeitsplatzes haben.

In diesem Kapitel haben Sie verschiedene Methoden kennengelernt, um Entfernungen zu messen. Sie können Geräte bauen, die erkennen, ob sich etwas in der Nähe befindet, und sogar feststellen, wie weit es entfernt ist. Mit mehr als einem Sensor können Sie auch anspruchsvollere Verhaltensweisen erzeugen. Beispielsweise können Sie zwei IR-Sensoren mit einem Servomotor kombinieren, sodass er sich in die Richtung des nächstliegenden Objekts dreht. Das ist eine einfache Möglichkeit, um ein Gerät zu bauen, das Ihrer Hand folgt. Zwei IR-Empfänger in einem fahrbaren Roboter können dafür sorgen, dass er einer Flamme folgt. Wie werden Sie in Ihren eigenen Projekten Abstandssensoren nutzen?