Wir messen digital

Viele Fußgängerampeln schalten nur bei Bedarf um. Will ein Fußgänger die Straße überqueren, muss er zunächst einen Knopf drücken. Erst dann bekommen die Autos auf der Straße ein Rot-Signal und die Fußgängerampel springt von Rot auf Grün, um nach einer Weile wieder auf Rot zurückzukehren.

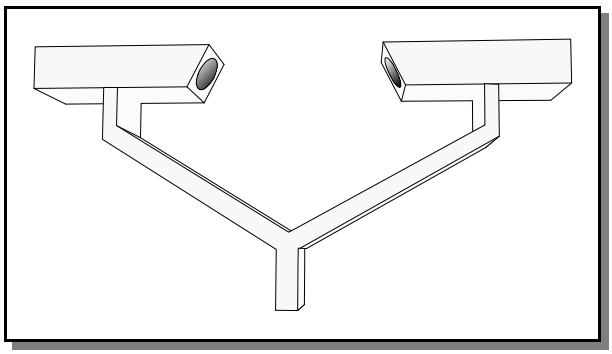

Abb. 1: Nebelmessgerät

An Rändern von Autobahnen sieht man zuweilen Nebelmessgeräte wie in Abb. 1. Je nach Stärke des Nebels werden mit ihrer Hilfe über Computer entsprechende Geschwindigkeitsbegrenzungen aktiviert.

Bei diesen beiden Anlagen ist entscheidend, dass sie auf äußere Einflüsse reagieren, indem sie Signale von Sensoren auswerten. Wenn diese Sensoren nur zwischen zwei Zuständen – wie im Fall eines Schalters oder Tasters – unterscheiden können, sprechen wir von **digitalem Messen**. Im Fall des Nebelmessgerätes liefert der Sensor viele unterschiedliche Messwerte, welche in ihrer Größe der Nebeldichte entsprechen. In solchen Fällen spricht man von **analogem Messen**.

In diesem Kapitel widmen wir uns dem digitalen Messen. Wie man mit dem Rechner analoge Messungen durchführen kann, erfährst du erst in dem nächsten Kapitel.

Die Eingangsfunktionen von COMX - Schalterzustände abfragen

Die serielle Schnittstelle besitzt mehrere Eingänge. Unbeschaltet haben sie eine Spannung von 0 V. Werden sie über ein Kabel an eine elektrische Quelle angeschlossen, so übernehmen sie deren Spannungswert. Steigt die Spannung am Eingang über den Schwellenwert +1,25 V, dann geht der Eingang in den High-Zustand (1), sinkt er unter +1,0 V, dann geht er in den Low-Zustand (0).

In Abb. 2 ist der Eingang RI über einen Schalter mit dem Ausgang RTS verbunden. Ist der Schalter geöffnet, dann ist der Eingang RI unbeschaltet, also im Low-Zustand. Wenn er geschlossen ist, dann ist er leitend mit dem Ausgang RTS verbunden. Ist dieser im High-Zustand, ist seine Spannung bei + 12 V; das ist bei weitem größer als 1,25 V, also ist dann auch der Eingang RI im High-Zustand. Wenn aber der Ausgang RTS im Low-Zustand ist, so ist seine Spannung bei −12 V, das liegt weiter unter der Grenze von 1 V, der Eingang RI ist dann also im Low-Zustand.

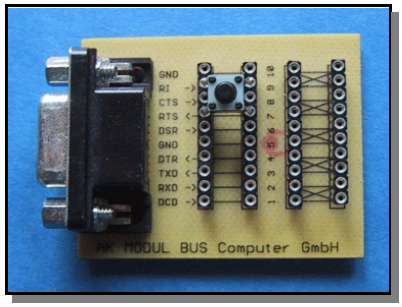

Abb. 2: Platine mit Taster

Die Eingänge RI, CTS und DSR können mithilfe der Komponente COMX auf ihren Zustand abgefragt werden. Dazu stehen die folgenden Eigenschaften zur Verfügung:

Eigenschaften	Bedeutung	Beispiel
CTS	gibt den Zustand des CTS-Eingangs an (nur Lesen)	COMX.CTS hat den Wert 1, wenn CTS auf High ist.
DSR	gibt den Zustand des DSR-Eingangs an (nur Lesen)	COMX.DSR hat den Wert 0, wenn DSR auf High ist.
RI	gibt den Zustand des RI-Eingangs an (nur Lesen)	COMX.RI hat den Wert 1, wenn RI auf High ist.

Um zu überprüfen, ob ein Schalter wie in Abb. 2 offen oder geschlossen ist, verfährt man deswegen folgendermaßen: Der Schalter wird zwischen einen Eingang und einen Ausgang gelegt; der Ausgang wird mit `COMX.Ausgang=1` auf High gelegt. Nun wird der Eingang abgefragt:

Wert von COMX.Eingang	Schalter
1	geschlossen
0	offen

Das Dokument in Abb. 3 soll nach Betätigen der Schaltfläche "Anzeigen" den Schalterzustand anzeigen: Wenn der Schalter geschlossen ist, soll der Text "geschlossen" angezeigt werden, ansonsten der Text "offen". Zunächst setzen wir den Schalter zwischen den Eingang RI und den Ausgang RTS; diese Wahl ist günstig, weil die Anschlüsse des Schalters so direkt in die entsprechenden Buchsen auf der Platine gesteckt werden können.

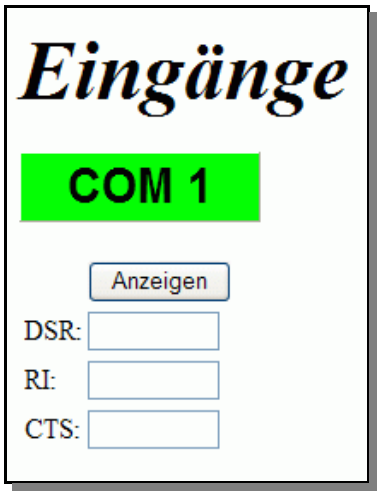

Abb. 3: Abfragen der Eingänge

Beim Laden des Programms wird die Funktion `init()` aufgerufen:

```
function init()
{
      COMX.open(1);
      COMX.RTS=1; // RTS auf High
}
```

Beim Verlassen des Dokuments wird die Funktion `schluss()` aufgerufen:

```
function schluss()
{
      COMX.close();
}
```

Wenn die Schaltfläche “Anzeigen” betätigt wird, wird die Funktion `anzeigen()` ausgeführt:

```
function anzeigen()
{
      if (COMX.RI == 1)
      {
            pform.ri.value = "geschlossen"
      }
      else
      {
            pform.ri.value = "offen"
      }
}
```

Statt des Schalters kann man auch einen LDR zwischen Ein- und Ausgang schalten. Dieses elektronische Bauteil verändert seine Leitfähigkeit je nach Lichteinfall. Fällt viel Licht auf den LDR, so wirkt er wie ein geschlossener Schalter, fällt wenig Licht auf den LDR, verhält er sich wie ein geöffneter Schalter. Mithilfe eines LDRs können wir also messen, ob viel oder wenig Helligkeit vorliegt.

Aufgaben

1. Ergänze die Funktion `anzeigen()` so, dass auch die restlichen Eingänge abgefragt werden. Teste dein Programm, indem du ein Kabel mit einem Ende in die Buchse RTS und das andere Ende in die Buchsen der verschiedenen Eingänge steckst.

2. Ersetze das Kabel aus Aufgabe 1 durch den LDR. Teste die Anordnung. Wozu könnte sie benutzt werden?

Wir warten auf den Knopfdruck (Die while-Schleife)

Für eine Fußgängerampel kann unser Dokument aus Abb. 3 noch nicht benutzt werden. Es zeigt den momentanen Zustand des Schalters nur an, wenn wir die Schaltfläche „Anzeigen“ betätigen. Bei einer Fußgängerampel muss das Programm aber selbstständig herausfinden, wann der Schalter geschlossen wird und dann entsprechend reagieren. Dazu muss das Programm solange warten, bis der Schalter geschlossen wird. Dies kann man mit einer **while-Schleife** realisieren:

Die while-Schleife hat folgende Form:

```
while (Bedingung) { <Anweisungen> }
```

Der Anweisungsblock wird auch Schleifenkörper genannt; er wird solange immer wieder ausgeführt, wie die Bedingung erfüllt ist.

Das Warten auf einen Knopfdruck lässt sich damit nun durch folgende Funktion realisieren:

```
function wartenBisKnopfdruck()
{
      while (COMX.RI == 0) { }
}
```

Dabei gehen wir davon aus, dass der Schalter zwischen RI und einem Ausgang liegt und dieser im High-Zustand ist. Solange der Schalter geöffnet ist, wird die Schleife immer wieder durchlaufen; dabei werden aber keinerlei Anweisungen ausgeführt, denn der Schleifenkörper ist leer! Vor jedem Schleifendurchlauf wird aber der Eingang RI abgefragt. Wenn nun irgendwann der Taster betätigt wird, geht der Eingang RI vom Low- in den High-Zustand über, die Bedingung `COMX.RI == 0` ist dann nicht mehr erfüllt und die Schleife wird abgebrochen. Der Browser ist also solange mit Nichts-Tun beschäftigt, bis der Taster betätigt wird.

Das wollen wir jetzt ausnutzen, um eine einfache Fußgängerampel zu realisieren: Zuerst ergänzen wir unseren Aufbau aus Abb. 2 um zwei Leuchtdioden, eine rote und eine grüne, die wir entgegengesetzt gepolt zwischen DTR und GND einbauen (Abb. 4). Bei dieser Schaltung der Leuchtdioden ist garantiert, dass die rote LED immer dann leuchtet, wenn die grüne gerade nicht leuchtet und umgekehrt. Das Ein- und Ausschalten geschieht mit den beiden Funktionen:

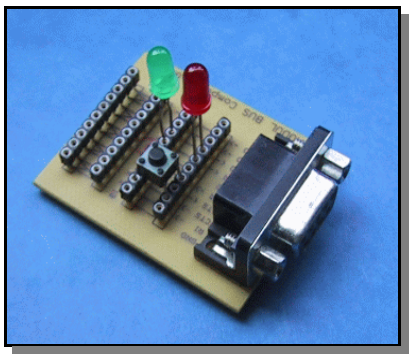

Abb. 4: Fußgängerampel

```
function rotAn_gruenAus()
{
      COMX.DTR = 1;
}

function rotAus_gruenAn()
{
      COMX.DTR = 0;
}
```

Unsere Fußgängerampel soll beim Laden des Programms dem Fußgänger Rot anzeigen und in diesem Zustand bleiben, bis der Taster betätigt wird. In diesem Fall soll sie für eine bestimmte Zeit, z. B. 8 Sekunden, auf Grün springen, um dann wieder auf Rot zu wechseln. Wir stellen uns vor, dass die Signale für den Autofahrer gerade entgegengesetzt sind. Das ist natürlich eine grobe Vereinfachung, da in Wirklichkeit die Ampel für den Autofahrer einige Sekunden lang auf Rot gesetzt sein muss, bevor der Fußgänger das Grün-Signal bekommt.

Die Funktion `init` öffnet die Schnittstelle und setzt den Ausgang RTS (für den Taster) auf High. Die Funktion soll wie bislang üblich durch das onLoad-Ereignis aufgerufen werden.

```
function init()
{
      COMX.open(1);
      COMX.RTS=1;
}
```

Wenn die Schaltfläche “Start” angeklickt wird, soll unsere Ampelanlage ihre Arbeit beginnen: Die Fußgängerampel wird auf Rot gesetzt und dann soll sie auf den ersten Fußgänger warten. Wenn der auf den Taster drückt, soll er für kurze Zeit ein Grün-Signal bekommen. Die zugehörige Funktion lautet:

```
function ampel()
{
      rotAn_gruenAus();
      wartenBisKnopfdruck(); // siehe oben
      rotAus_gruenAn();
      COMX.delay(8000); // 8 Sekunden Zeit zum Überqueren
      rotAn_gruenAus();
}
```

Beim Austesten unseres Programm müssen wir aber eine Enttäuschung erleben. Zunächst sieht noch alles gut aus. Beim ersten Fußgänger reagiert die Ampel wie geplant; aber schon beim zweiten Knopfdruck bleibt die Ampel auf Rot stehen. Wenn wir uns die Funktion `ampel` anschauen, finden wir den Grund dafür sehr schnell. Nachdem die Ampel wieder auf Rot gesetzt wurde, ist die letzte Anweisung der Funktion abgearbeitet, das Programm ist zu Ende. In Wirklichkeit sollte jetzt aber wieder auf den nächsten Fußgänger gewartet werden. Deswegen müssen wir unsere Ampelfunktion in eine Schleife stecken, die immer wieder durchlaufen wird:

```
function ampelschleife()
{
      while (COMX.CTS == 0)
      {
            ampel();
      }
}
```

Wenn nun die Funktion `ampelschleife` durch die Start-Schaltfläche gestartet wird, reagiert die Anlage immer wieder auf den Knopfdruck mit einer Grünphase, so wie es auch sein soll. Wir haben hier keine Endlosschleife vorliegen; die Schleife kann abgebrochen werden, wenn man den CTS-Eingang über ein Drahtstück mit dem Ausgang RTS verbindet.

Technische Prozesse

Unter einem **Prozess** versteht man die Umformung oder den Transport von Materie, Energie oder Information. Man unterscheidet zwischen technischen und nicht-technischen Prozessen. Bei den technischen Prozessen werden die Veränderungen durch technische Hilfsmittel hervorgerufen. Technische Prozesse sind z. B. das Waschen eines Autos in der Waschstraße, die Steuerung einer Ampelanlage, der Spülvorgang in einer Spülmaschine oder die Wiedergabe eines Videofilmes. Nicht-technische Prozesse sind das Aufräumen eines Zimmers, das Laufen, das Erlernen eines Gedichtes. Auch der Prozess in einem Gerichtssaal ist ein nicht-technischer Prozess; hier werden Informationen zusammen getragen und so lange umgeformt, bis sich ein Beweis für die Schuld oder Unschuld eines Angeklagten ergibt.

Häufig werden bei einem Prozess Daten benutzt: Sensoren liefern Messwerte an den Computer, dieser verarbeitet sie und gibt seinerseits Daten an Aktoren, welche den Prozess beeinflussen. Deswegen spricht man in diesem Fall von **Prozessdatenverarbeitung**.

Auch im Falle nicht-technischer Prozesse spielen Informationen eine wesentliche Rolle: Beim Gehen z. B. liefern verschiedene Sinneszellen Informationen über den Gleichgewichtszustand des Körpers an das Gehirn; dieses wertet sie aus und sendet entsprechende Signale an die Muskeln des Bewegungsapparates. Dass es sich hier um einen recht komplexen Prozess handelt, wird deutlich, wenn man bedenkt, dass es auch heute nur wenige Roboter gibt, welche auf zwei Beinen gehen, Treppen steigen oder aus dem Liegen aufstehen können.

Aufgaben

1. Programmiere unsere Fußgängerampel und teste sie aus. Warum kann man die letzte Anweisung der `ampel`-Funktion jetzt entfernen?

2. Der Abbruch der Ampelschleifen-Funktion in Aufgabe 1 kann auch mithilfe eines Stopp-Knopfes realisiert werden. Warum muss dann die doEvent-Methode zum Einsatz kommen? Ändere das Ampel-Programm entsprechend ab.

3. Der Wechsel der beiden Ampelphasen soll jeweils durch eine kurze Blinkphase begleitet werden. Ergänze das Programm aus Aufgabe 1 entsprechend. Programmiere modular!

4. Programmiere eine Stoppuhr, die mit dem Taster gestartet und gestoppt wird. Warum muss man dabei auch darauf warten, dass der Taster geöffnet wird?

5. Programmiere einen Impulszähler; dieser soll zählen, wie häufig der Taster in einer vorgegebenen Zeit betätigt wurde.

6. Programmiere einen Reaktionstest.

7. Facharbeit: Baue und programmiere einen Drehzahlmesser.

8. Facharbeit: Baue und programmiere einen Barcode-Scanner (Code: dicker Strich = 1; dünner Strich = 0)

9. Nenne drei weitere technische und drei nicht-technische Prozesse. Erläutere jeweils, inwiefern es sich um einen Prozess handelt.

10. Einer der Väter der Robotik ist Isaac Asimov, Chemiker und Autor populärwissenschaftlicher Bücher sowie von Science-Fiction-Romanen. Von ihm stammen die berühmten drei Roboter-Gesetze. Versuche deren Wortlaut in Erfahrung zu bringen. Welche Ängste lassen sich hinter diesen Gesetzen erkennen?

11. Schreibe eine kurze Entwicklungsgeschichte der Roboter. Kläre dabei auch: Von wem stammt die Bezeichnung Roboter? Welche Bedeutung hatte sie ursprünglich?

Abb. 5: Ein Speisentransporter im Test

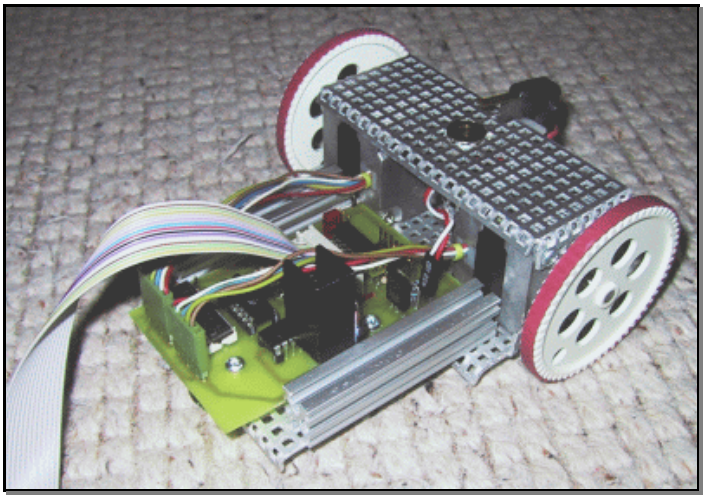

Abb. 6: Ein Eigenbau-Spurverfolger

Wir regeln einen fahrerlosen Transporter

Wie von Geisterhand gesteuert bewegen sich Transporter mit Speisen durch die Gänge eines Krankenhauses (Abb. 5). Das ist keine Zukunftsmusik und hier wirkt auch keine Fernsteuerung im Hintergrund. Vielmehr folgen die Transporter selbstständig den auf dem Boden angebrachten Spuren. Ähnliche Transportsysteme findet man auch in Fabrikhallen oder auf Umschlagplätzen von großen Häfen.

Derartige Roboter kann man auch selbst bauen: In Abb. 6 können die beiden Motoren das Fahrzeug vorwärts und rückwärts fahren sowie nach links und rechts drehen lassen. Zur Steuerung benutzen wir – wie schon beim Igel – die Methoden `vw` und `dreh`:

Methode	Bedeutung
`robot.vw(s)`	bewegt den Roboter um s Einheiten vorwärts (wenn s positiv) bzw. rückwärts (wenn s negativ)
`robot.dreh(w)`	dreht den Roboter je nach Vorzeichen von w auf der Stelle um w Grad im bzw. gegen den Uhrzeigersinn

Unter der Platine sind zwei optische Sensoren S1 und S2 angebracht; diese können über die robot-Eigenschaften E1 und E2 abgefragt werden:

Eigenschaft	Bedeutung
`robot.E1`	E1 hat den Wert 0, wenn der Sensor S1 über der schwarzen Spur steht; sonst ist der Wert von E1 gleich 1.
`robot.E2`	wie bei E1

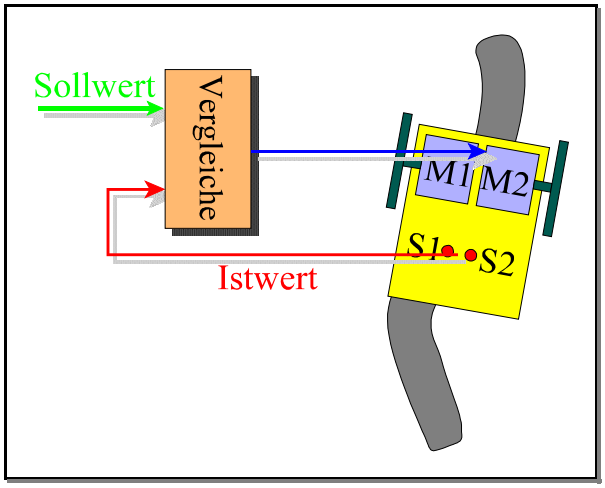

Abb. 7: Sollwert und Istwert werden verglichen.

Liegt unser Roboter korrekt über der schwarzen Spur, so haben beide Sensoreigenschaften E1 und E2 den Wert 0; in diesem Fall soll sich der Roboter einen Schritt vorwärts bewegen. Liegt einer der Sensoren aber außerhalb der Spur, so muss sich der Roboter entsprechend drehen. In der Abb. 7 liefert der Sensor S2 den Wert E2 = 1; in diesem Fall muss der Roboter sich im Uhrzeigersinn drehen. Wenn der Sensor S1 außerhalb der Spur liegt, muss der Roboter sich im Gegenuhrzeigersinn drehen; und wenn beide Sensoren außerhalb der Spur liegen, soll der Roboter stoppen. Dies leistet das folgende Programm:

```
function schritt()
{
  if (i.E1==1)       // über weißem Grund außerhalb der Spur
    { i.dreh(3) }    // Drehung von 3 Grad im Uhrzeigersinn
  else
  {
    if (i.E2==1)     // über weißem Grund außerhalb der Spur
    {
      i.dreh(-3)     // Drehung von 3 Grad gegen Uhrzeigersinn
    }
    else
    {
      i.vw(1)        // 1 Schritt vorwärts
    }
  }
}

function folgeSpur()
{
  while (i.E1+i.E2<2)     //E1+E2=2, wenn S1 und S2 außerhalb
                          //der Spur
  {
     schritt()
  }
}
```

Wesentlich ist hier, dass über die beiden Sensoren fortwährend die Position des Roboters kontrolliert wird. Hat der Roboter noch die gewünschte Position oder hat er sie bereits verlassen? Je nachdem fährt der Roboter geradeaus weiter oder dreht sich in die entsprechende Richtung. Einen solchen Vorgang bezeichnet man als ein **Regeln**.

> Regeln heißt ein Vorgang, bei dem eine gemessene Größe (die **Regelgröße**) fortlaufend mit einem vorgegebenen Wert (dem **Sollwert**) verglichen wird und bei dem durch geeignete Maßnahmen die dabei festgestellten Abweichungen verringert oder ganz beseitigt werden.

In unserem Fall setzt sich die Regelgröße aus den beiden Eingangswerten E1 und E2 zusammen. Der Sollwert ist E1 = 0 und E2 = 0, d.h. beide Sensoren befinden sich über der Spur. Weichen die gemessenen Werte (**Istwert**) von diesem Sollwert ab, so dreht sich der Roboter in geeigneter Weise; dadurch wird die Abweichung von der Spur verringert oder ganz beseitigt (Abb. 7). Es liegt hier also ein Wirkungskreis vor: Die Sensoren beeinflussen die Bewegung des Roboters und diese wiederum wirken sich auf die Lage der Sensoren und damit auch auf ihren Zustand aus.

Derartige Wirkungskreise sind typisch für Regelungen. Man spricht hier deswegen auch von **Regelkreisen**. Beim **Steuern** hingegen wird eine Größe beeinflusst, ohne dass kontrolliert wird, ob der gewünschte Wert auch tatsächlich angenommen wird. Steuerungen besitzen demnach keinen geschlossenen Wirkungskreis.

Aufgaben

1. Beschreibe den Wirkungskreis bei einer Thermostat-geregelten Heizung.

2. Nenne für die Steuerung und für die Regelung jeweils 3 weitere Beispiele.

3. Unser Roboter-Programm lässt den Roboter anhalten, wenn die Spur verloren wurde (E1 = 1 und E2 = 1). Ändere das Programm so ab, dass es in diesem Fall nach der Spur sucht.

4. Ergänze das Roboter-Programm um eine Lern- und um eine Ausführfunktion: Der Roboter soll sich die „Spur“ merken können und anschließend „blind“, d.h. ohne Spur, dieselbe Fahrt noch einmal ausführen können.