Programmierung mit TI BASIC für Fortgeschrittene

Beispiele für TI-Nspire™ CX mit TI-Innovator™ Hub und TI-Innovator™ Rover

Jürgen Enders (Hrsg.)

T³ EUROPE

Teachers Teaching with Technology™

Herausgeber:
Jürgen Enders

Autoren:
Veit Berger, Oliver Gallus, Hubert Langlotz, Hans-Martin Hilbig, Ralph Huste, Christian Zöpfl

Dieses und weiteres Material steht Ihnen zum pdf-Download bereit: **www.t3europe.eu** sowie unter **www.ti-unterrichtsmaterialien.net**

Inhaltsverzeichnis

Versuche mit dem TI-Innovator™ Hub:

Versuche mit dem TI-Innovator™ Rover :

Vorwort

„Programmieren ist wie Küssen: Man kann darüber reden, man kann es beschreiben, aber man weiß erst was es bedeutet, wenn man es getan hat.“ meinte Andrée Beaulieu–Green, Gründerin des kanadischen ICARI (Institut de Création Artistique en Recherche Informatique).

Das Küssen steht bekanntlich ja nicht in den Lehrplänen und Curricula der europäischen Schulen. Beim Programmieren sieht es, zum Glück für unsere Gesellschaft, inzwischen anders aus. Zumindest in den Grundzügen gehört inzwischen die Fähigkeit zu Programmieren zu einer soliden Ausbildung dazu. Dabei ist die verwendete Programmiersprache zunächst nachrangig. Es gilt, ein Verständnis von Strukturen, sequenziellen Abläufen und Variablen zu schaffen. Alles Fähigkeiten, die auch im Mathematik–Curriculum in der einen oder anderen Form am thematisiert werden.

Nun sind aber nicht nur beim Küssen Theorie und Praxis zwei völlig verschiedene Paar Schuhe. Um Spaß am Programmieren zu haben, muss man (und dieses *man* gilt hier besonders auch für Schülerinnen) es einfach mal ausprobieren. Das erste „Hello World“, die erste selbst angesteuert blinkende LED und die erste Ausfahrt eines Rovers, all das macht dann einfach Spaß und Lust auf mehr. Nein, einen weiteren Vergleich zum Küssen wird es nicht geben.

Programmieren kann aber, gerade am Beginn, auch Quell von Frust sein, gerade wenn man in einer Programmierumgebung noch nicht firm ist oder neben der Programmierung noch externe Komponenten der „realen Welt“ ins Spiel kommen. Genau hier setzt dieses Heft an. Es zeigt an mehreren Beispielen unterschiedlicher Komplexität, wie beeindruckend einfach sich Programmierung und Regelungstechnik mit dem TI Innovator System verbinden lassen. Während im ersten Heft „*Von der Messwerterfassung bis zur Programmierung*“ [1] aus der T³–Reihe vor allem die Grundlagen von TI Basic und dem TI-Innovator™ System behandelt wurden, legt dieses Heft den Focus auf fortgeschrittene Aufgaben aus dem Bereich „Messen, Steuern, Regeln“ sowie Robotik-Programmierung mit dem TI-Innovator™ Rover.

Wir wünschen Ihnen, geschätzte Leserinnen und Leser, genauso viel Spaß beim Programmieren wie wir beim Entwickeln der Beispiele hatten und freuen uns über Ihre Anmerkungen und Hinweise.

Das Autorenteam

Ergänzende Hinweise
Bei der Erarbeitung der Experimente werden die derzeitig gültigen Sicherheitsbestimmungen für elektrische Anlagen zugrunde gelegt.
Es sei ausdrücklich darauf hingewiesen, dass die betreuende Lehrperson die Verantwortung für die Einhaltung von Sicherheitsbestimmungen in elektrischen Schaltungen trägt.
Alle Anleitungen wurden sorgfältig erarbeitet. Dennoch übernehmen die Herausgeber und Autoren für die Richtigkeit von Aufgaben, deren Lösungen, Hinweisen und Ratschlagen sowie für eventuelle Druckfehler keine Haftung. In den nachfolgenden Anleitungen werden die Bezeichnungen Rechner und Taschenrechner für entsprechende Geräte zur Messwerterfassung bzw. zur Programmierung des TI–Innovator™ Hubs genutzt. Alle Beispiele wurden in der TI–Nspire™ CX Plattform erstellt und getestet. Bei Verwendung des TI–84 Plus CE-T sind Modifikationen an den Programmen nötig.

Zu finden unter:
https://ti-unterrichtsmaterialien.net/fileadmin/user_upload/Von_der_Messwerterfassung_bis_zur_Programmierung.pdf

Versuche mit dem TI-Innovator™ Hub:

1. Rollosteuerung

Ein Fensterrollo soll in Abhängigkeit der jeweiligen Helligkeit herabgelassen bzw. hoch-gezogen werden. Mit einer Programmerweiterung soll es zusätzlich möglich sein das Rollo auch manuell zu bedienen

Überblick

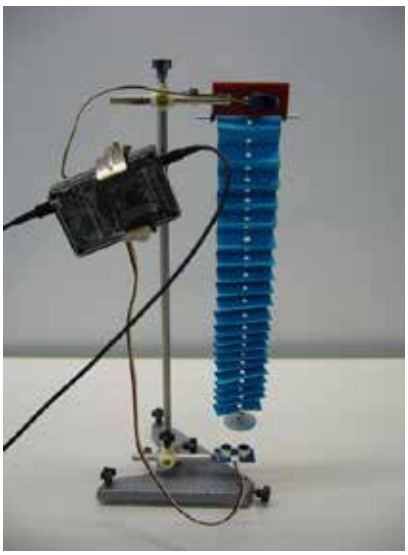
- Die Rollobewegung wird mit einen Servomotor realisiert.
- Der interne Helligkeitssensor wird zur Messung der Umgebungshelligkeit genutzt.
- Die Abschaltung des Rollos im unteren bzw. oberen Endzustand soll durch Abstandsmessung mit einem Ultraschallsensor erfolgen.
- Wahlweise soll das Umschalten vom automatischen in den manuellen Betrieb und umgekehrt möglich sein.
- Das Programm kann zu jedem Zeitpunkt abgebrochen werden.

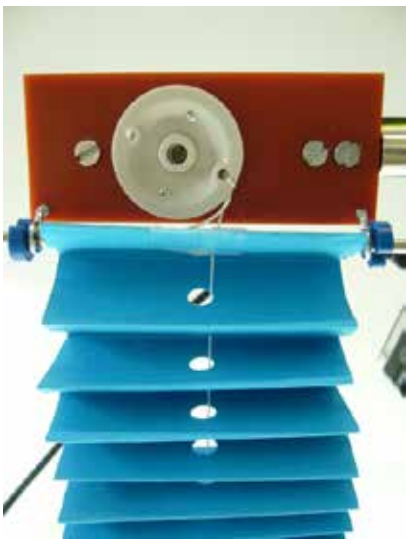
Externe Sensoren / Aktoren:

- interner Lichtsensor BRIGHTNESS
- Servomotor
- Ultraschallsensor RANGER

Aufbau

Zunächst sollte ein geeignetes Rollo-Modell hergestellt werden (s. Abbildungen). Am unteren Ende des Papier-Rollos wird ein entsprechender Ultraschall-Reflektor angebracht.
Der Servomotor muss an Out 3 angeschlossen werden und benötigt in jedem Fall eine externe Stromversorgung über den Eingang PWR. Im Weiteren wird vereinbart, dass eine Linksdrehung das Hochziehen und eine Rechtsdrehung das Herablassen des Rollos bewirken.
Der Ultraschallsensor wird an IN 1 angeschlossen.





möglicher Aufbau eines Rollo-Modells

Aufgabe 1:
Erstelle ein Programm **rollo1()**, welches ab einem Schwellenwert der Umgebungshelligkeit mit einem Servomotor das Rollo herablässt.
Zur Lösung wird ein erster Programmentwurf vorgestellt. Er besitzt aber noch wesentliche Mängel. Teste das Programm und gib diese Mängel an.

Lösung:

Deklarationen lokaler Variablen:
b ... Messwert der Helligkeit
bmin ... Helligkeitsminimum

Verbindung des Servomotors mit OUT3

Einlesen des Helligkeitswertes und Speichern des Wertes in der Variable b.

Kontrollausgabe des Helligkeitswertes.
Der Wert b muss ggf. angepasst werden.

Der Servomotor dreht sich mit einer Dauer von 1 s links herum, falls eine Helligkeits-grenze überschritten wird (der negative Wert -20 gibt auch die Geschwindigkeit mit Beträgen zwischen 0 und 100 an), anderenfalls rechts herum (positiver Wert).

Der Programmabbruch erfolgt mit Hilfe des getkey() Befehls.

Trennung der Verbindung zum Servomotor.

```
Define rollo1()=
Prgm
:Local b,bmin
:Send „CONNECT SERVO 1 TO OUT 3"
:bmin:=2
:While getKey()≠"esc"
: Send „READ BRIGHTNESS"
: Get b
: DispAt 1,"Brightness:
„,round(b,2)
: If b>bmin Then
:  Send „SET SERVO 1 TO -20 1"
: Else
:  Send „SET SERVO 1 TO 20 1"
: EndIf
: Wait 0.05
:EndWhile
:Send „DISCONNECT SERVO 1"
:EndPrgm
```

Aufgabe 2:
Verändere das Programm **rollo1()** dahingehend, dass ein Hoch- bzw. Herunterfahren erst dann wieder erfolgt, wenn ein unterer Schwellenwert der Umgebungshelligkeit unterschritten bzw. ein oberer Schwellenwert überschritten wird. Das neue Programm heißt dann **rollo2()**.
Begründe, warum diese so genannte Zweipunktregelung günstiger ist.

Lösung:

Deklarationen lokaler Vaiablen:
b ... Messwert der Helligkeit
bmin ... Helligkeitsminimum
bmax ... Helligkeitsmaximum
r ... Bewegungsrichtung

Falls die obere Helligkeitsgrenze überschritten ist und das Rollo zuvor nach oben bewegt wurde, wird das Rollo nach unten gezogen.
Falls die untere Helligkeitsgrenze unterschritten ist und das Rollo zuvor nach unten bewegt wurde, wird das Rollo nach oben gezogen.

```
Define rollo2()=
Prgm
:Local b,bmin,bmax,r
:Send „CONNECT SERVO 1 TO OUT 3"
:bmin:=2: bmax:=5: r:=0
:While getKey()≠"esc"
: Send „READ BRIGHTNESS"
: Get b
: DispAt 1,"Brightness:
„,round(b,2)
: If b>bmax and r≤0 Then
:  Send „SET SERVO 1 TO -20 2.5"
:  r:=1
: EndIf
: If b<bmin and r≥0 Then
:  Send „SET SERVO 1 TO 20 2.7"
:  r:=-1
: EndIf
: Wait 0.05
:EndWhile
:Send „DISCONNECT SERVO 1"
:EndPrgm
```

Durch das Einführen einer Richtungsvariable wird im Programm **rollo2()** jeweils eine vollständige Rollobewegung nach oben bzw. nach unten ausgeführt.
Der Übergang zur Zweipunktregelung führt zu einer Schalthysterese, die bei Schwankungen um den Helligkeitsschwellenwert ein ständiges wechselseitiges Heraufziehen und Herablassen des Rollos vermeidet.

Aufgabe 3:
Beim Test des Programms **rollo2()** fällt auf, dass die empirisch ermittelten Rollo-Laufzeiten (hier 2,5 s bzw. 2,7 s) nicht sehr zuverlässig sind.
Deshalb soll das Abschalten in den jeweiligen Endzuständen des Rollos aus einer Abstandsmessung mit dem Ultraschallsensor RANGER erfolgen.
Implementiere ein vergleichbares Programm **rollo automat()**.

Lösung:

Deklaration weiterer lokaler Variablen:
v ... Geschwindigkeit des Servos,
dt ... kleines Zeitintervall,
h ... Messwert der Höhe,
hmin ... minimale Höhe,
hmax ... maximale Höhe,

Ausgabe der gerundeten Messwerte.

Bei Unterschreitung des Helligkeits-minimums wird die Bewegungsrichtung „aufwärts“ festgelegt, bei Überschreitung des Helligkeitsmaximums entsprechend „abwärts“.

Abwärtsbewegung bei maximaler Helligkeit.

Aufwärtsbewegung bei minimaler Helligkeit.

```
Define rollo_automat()=
Prgm
:Local v,dt,r
:Local b,bmin,bmax
:Local h,hmin,hmax
:v:=20: dt:=0.05: r:=0
:bmin:=2: bmax:=5
:hmin:=0.05: hmax:=0.3
:Send „CONNECT SERVO 1 TO OUT 3“
:Send „CONNECT RANGER 1 TO IN 1“
:Send „READ BRIGHTNESS“
:While getKey()≠“esc“
: Send „READ BRIGHTNESS“
: Get b
: Send „READ RANGER 1“
: Get h
: DispAt 1,“Brightness   :
„,round(b,2)
: DispAt 2,“Ranger (cm): „,
            round(100*h,1)
: If b<bmin Then
:  r:=1
: EndIf
: If b>bmax Then
:  r:=-1
: EndIf
: If b>bmax and h>hmin and r≤0 Then
:  Send „SET SERVO 1 TO eval(-1*v)
         eval(10*dt)“
: EndIf
: If b<bmin and h<hmax and r≥0 Then
:  Send „SET SERVO 1 TO eval(v)
         eval(10*dt)“
: EndIf
: Wait dt
:EndWhile
:Send „DISCONNECT SERVO 1“
:Send „DISCONNECT RANGER 1“
:EndPrgm
```

Aufgabe 4:
Bei übermäßiger Sonneneinstrahlung besteht der Wunsch, das Rollo manuell herabzulassen. Teste zunächst eine manuelle Steuerung des Rollos. Benutze dazu den entsprechenden Schieberegler (s. Abbildung).


```
Define rollo_manuell(r)=
Prgm
:Local v,dt,h,hmin,hmax
:Send „CONNECT SERVO 1 TO OUT 3“
:Send „CONNECT RANGER 1 TO IN 1“
:v:=20: dt:=0.05
:hmin:=0.05: hmax:=0.3
:Send „READ RANGER 1“
:Get h
:While r=-1 and h>hmin or r=1 and
h<hmax
: Send „SET SERVO 1 TO eval(r*v)
eval(10*dt)“
: Send „READ RANGER 1“
: Get h
: Wait dt
:EndWhile
:Send „DISCONNECT RANGER 1“
:Send „DISCONNECT SERVO 1“
:EndPrgm
```

Aufgabe 5:
Nun wollen wir die beiden Funktionalitäten des automatischen bzw. manuellen Rollo-Betriebs miteinander verbinden.
Beachte, dass die zwei Modi konsequent unterschieden werden müssen. Die manuelle Bedienung soll mit den Tasten ↓ („down“) bzw. ↑ („up“) realisiert werden. Die Steuerung über den Helligkeitssensor muss dabei inaktiv sein. Es muss aber auch die anschließende Umschaltung in den Automatik-Modus (Taste „a“) gegeben sein.
Implementiere die komplette Funktionalität im Programm rollo_gesamt().

Lösung:

Zunächst sollte ein Unterprogramm **motor()** entwickelt werden, in dem die Steuerung des Servomotors in Abhängigkeit zur Position des Ultraschallsensors nahezu vollständig *gekapselt* erfolgt. Der Parameter r bestimmt die Drehrichtung des Servomotors und kann wieder nur die Werte 1 bzw. -1 annehmen. Nur die Variable key muss eine *globale* Variable sein, um die weitere korrekte Interaktion mit dem Hauptprogramm **rollo_gesamt()** zu gewährleisten.

```
Define motor(r)=
Prgm
:Local v,dt,h,hmin,hmax
:Send „CONNECT SERVO 1 TO OUT 3“
:Send „CONNECT RANGER 1 TO IN 1“
:v:=20: dt:=0.05
:hmin:=0.05: hmax:=0.3
:Send „READ RANGER 1“
:Get h
:While key≠“esc“ and (r=-1 and h>h-
min or r=1 and h<hmax)
: key:=getKey()
: Send „SET SERVO 1 TO eval(r*v)
eval(10*dt)“
: Send „READ RANGER 1“
: Get h
: Wait dt
:EndWhile
:Send „DISCONNECT SERVO 1“
:Send „DISCONNECT RANGER 1“
:EndPrgm
```

Die Funktion **motor()** wird nun ins Hauptprogramm eingebunden:

Deklaration einer weiteren lokalen Variable:
sensor ... Unterscheidung zwischen den beiden Betriebsmodi:
true ... automatisch
false ... manuell

Aufruf des Unterprogramms **motor(r)** mit der Drehrichtung nach links.
Damit wird das Rollo in seine Ausgangsposition nach oben gezogen.

Aufruf des Unterprogramms **motor(r)** mit der Drehrichtung nach rechts.

```
Define rollo_gesamt()=
Prgm
:Local b,bmin,bmax,r,sensor
:bmin:=2: bmax:=5: r:=0
:sensor:=true: key:="enter"
:While key≠"esc"
: key:=getKey()
: If key="a" Then
:  sensor:=true
: EndIf
: If sensor Then
:  DispAt 1,"Umschalten in manuellen
              Modus: <up>/<down>"
: Else
:  DispAt 1,"Umschalten in
              Automatikmodus: <a>"
: EndIf
: DispAt 2,"Abbruch: <esc>"
: Send „READ BRIGHTNESS"
: Get b
: DispAt 3,"Brightness: „,round(b,2)
: If b<bmin and r≥0 and sensor Then
:  motor(1)
:  r:=-1
: EndIf
: If b>bmax and r≤0 and sensor Then
:  motor(-1)
:  r:=1
: EndIf
: If key="up" and r≥0 Then
:  motor(1)
:  r:=-1: sensor:=false
: EndIf
: If key="down" and r≤0 Then
:  motor(-1)
:  r:=1: sensor:=false
: EndIf
:EndWhile
:EndPrgm
```

2. Füllstandsregelung

Grundaufgabe

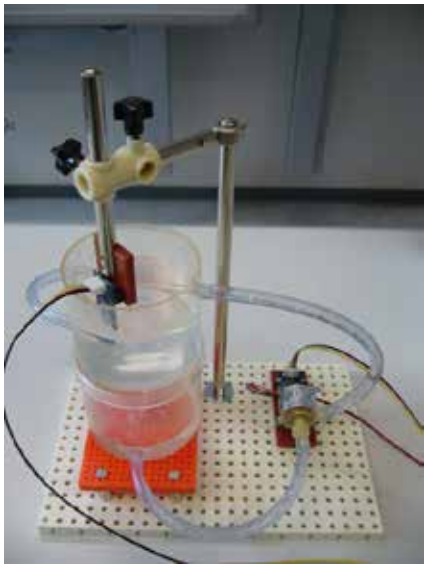
Mit dem TI Innovator™ soll mit einer geeigneten Experimentieranordnung das Modell einer Füllstandsregelung realisiert werden.

Überblick

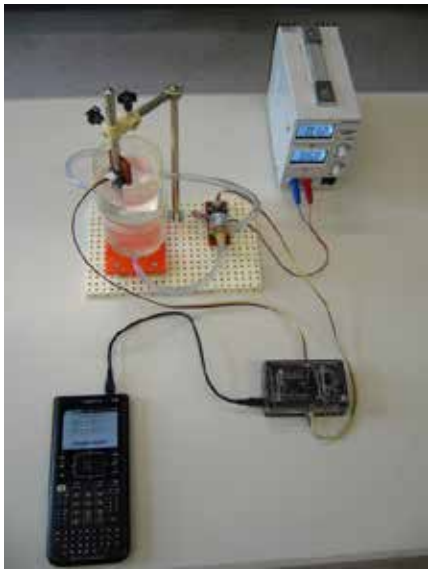
- Zunächst besteht eine anspruchsvolle Aufgabe darin, eine geeignete Versuchsanordnung (z.B. mit ausrangierten Überlaufgefäßen) herzustellen.
- Zur Füllstandsmessung wird ein Grove-Wassersensor benutzt, der die füllstandsabhängige Leitfähigkeit des Wassers erfasst.
- Die gewünschte Füllstandshöhe wird durch Eingabe eines maximalen bzw. minimalen elektrischen Widerstandes des Wassers eingestellt.
- Mit einer kleinen Wasserpumpe lässt sich dann diese Füllstandshöhe herstellen.

Aufbau und Hardware

Zwei Gefäße, die jeweils mit einem Zu- und einem Abflussstutzen ausgestattet sind, werden derart übereinander angeordnet und mit einem Schlauch verbunden, dass Wasser vom oberen in das untere Gefäß abfließen kann. Mit einer kleinen Wasserpumpe ist es weiterhin möglich, Wasser zurück vom unteren in das obere Gefäß zu pumpen (vgl. Abbildungen).
Ein Grove-Wassersensor erfasst den Wasserstand im oberen Gefäß. Über einen Grove-MOSFET-Treiber kann die Wasserpumpe mit dem TI Innovator ein- bzw. ausgeschaltet werden. Um das Wasser zu entkalken und gleichzeitig seine Leitfähigkeit zu verbessern, ist es empfehlenswert, ca. 10 ml Essig zuzugeben.
Der Wassersensor wird am Eingang IN1, der MOSFET-Treiber an den Ausgang OUT1 angeschlossen. Die Betriebsspannung für diesen Treiber beträgt etwa 6 - 8 V.
Diese Spannung kann neben einem handelsüblichen Stromversorgungsgerät auch mit einem ausrangierten Steckernetzteil bereitgestellt werden.



Detailansicht zur Füllstandsregelung



Gesamtansicht

Aufgabe 1:
Mit dem nachfolgenden Programm **fuellmessung()** wird die Wasserpumpe eingeschaltet. Beim Füllen erfolgt die Widerstandsmessung mit dem Wassersensor. Die „Systemzeit“ wird mit einem Wait-Befehl „getaktet“, dessen Argument ggf. angepasst werden muss. Die „Systemzeit“ und die Messwerte des elektrischen Widerstandes werden in Listen gespeichert und am Ende des Programmes ausgegeben. Der Programmabbruch erfolgt mit der Taste „esc“.
Teste das Programm. Achte darauf, dass in die Wasserpumpe keine Luftblasen gelangen. Beende dazu das Programm rechtzeitig.
Stelle den Widerstandsverlauf graphisch dar!

Lösung:

Löschen globaler Variablen
Deklarationen lokaler Variablen:
i ... Index der Listenelemente
dt, t0, t ... Zeitintervall / Zeitvariablen
Verbinden des Wassersensors mit IN1.
Verbinden des MOSFET-Treibers mit OUT1.
Einschalten der Pumpe.
Bestimmung der Startzeit.

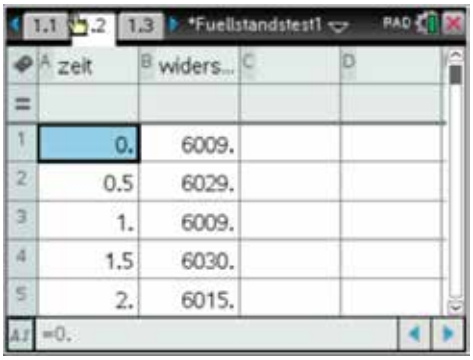
Programmabbruch mit „esc“.
Zeitmessung.
Berechnung der Messdauer in s.

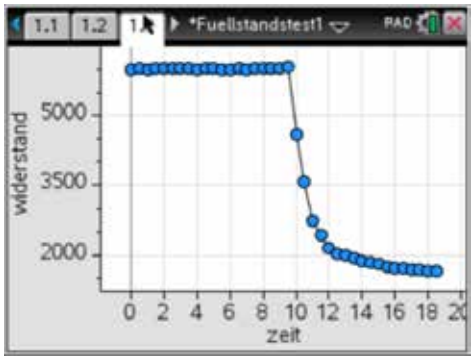
Widerstandsmessung am Wassersensor.

Ausgabe beider Messwerte.

Trennung der Hardware.

```
Define fuellmessung()=
Prgm
DelVar zeit,widerstand
Local i,t0,t,dt
dt:=0.5
Send "CONNECT ANALOG.IN 1 TO IN 1"
Send "CONNECT DIGITAL.OUT 1 TO OUT
1"
DispAt 1,"Abbruch mit <esc>"
Send "SET DIGITAL.OUT 1 ON"
Send "READ TIMER"
Get t0
i:=1
While getKey()≠"esc"
 Send "READ TIMER"
 Get t
 zeit[i]:=round(((t-t0)/(1000)),1)
 Send "READ ANALOG.IN 1"
 Get widerstand[i]
 DispAt 2,"Zeit: „,zeit[i]," s"
 DispAt 3,"Widerstand: ",wider-
stand[i]," Ohm"
 i:=i+1
 Wait dt
EndWhile
Send "DISCONNECT ANALOG.IN 1"
Send „DISCONNECT DIGITAL.OUT 1"
EndPrgm
```





Wenn der Füllstand den Wassersensor erreicht hat, nimmt der Leitungswiderstand der Flüssigkeit mit zunehmender Bedeckung des Wassersensors ab.

Aufgabe 2:
Bestimme anhand der Listen den Maximal- und den Minimalwert für den Füllstand und ermögliche im Programm die Eingabe dieser Werte. Vervollständige so das Programm zur Füllstandsregelung. Ermittle mit dem Statistik-Werkzeug „Data & Statistics“ die entsprechende Regelkurve.

Lösung:

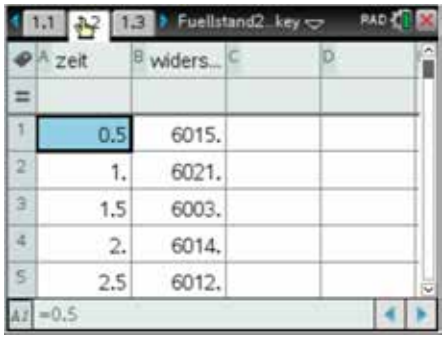
Löschen globaler Variablen
Deklarationen lokaler Variablen:

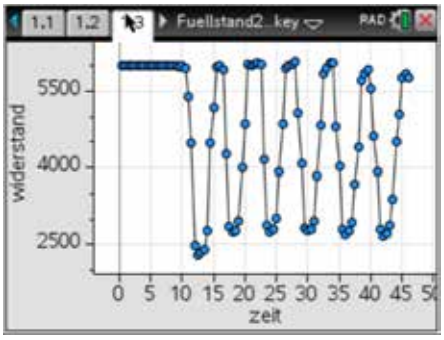
i	...	Index der Listenelemente
dt, t0, t	...	Zeitvariablen
minwert	...	Minimum des Leitungswiderstands
maxwert	...	Maximum des Leitungswiderstands
switch	...	Zustandsvariable

Falls der Widerstand den Minimalwert unterschritten hat, wird die Pumpe ausgeschaltet.

Falls der Widerstand den Maximalwert überschritten hat, wird die Pumpe eingeschaltet.

```
Define fuellstand()=
Prgm
DelVar zeit,widerstand
Local i,t0,t,dt,minwert,maxwert,s-
witch
dt:=0.5: minwert:=3000:
maxwert:=5000
Request „Minwert in Ohm:“,minwert
Request “Maxwert in Ohm:”,maxwert
Send “CONNECT ANALOG.IN 1 TO IN
1”
Send “CONNECT DIGITAL.OUT 1 TO
OUT 1”
DispAt 1,“Füllstandsregelung“
DispAt 2,“Abbruch mit <esc>“
Send “READ TIMER”
Get t0
i:=1: switch:=false
While getKey()≠”esc”
 Send “READ TIMER”
 Get t
 zeit[i]:=round(((t-t0)/(1000)),1)
 Send “READ ANALOG.IN 1”
 Get widerstand[i]
 DispAt 3,“Zeit: „,zeit[i],“ s“
 DispAt 4,”Widerstand: “,wider-
stand[i],” Ohm”
 If widerstand[i]<minwert and
switch Then
  Send “SET DIGITAL.OUT 1 OFF”
  switch:=false
 EndIf
 If widerstand[i]>maxwert and not
switch Then
  Send “SET DIGITAL.OUT 1 ON”
  switch:=true
 EndIf
 i:=i+1
 Wait dt
EndWhile
Send “DISCONNECT ANALOG.IN 1”
Send „DISCONNECT DIGITAL.OUT 1“
EndPrgm
```





Hinweis:

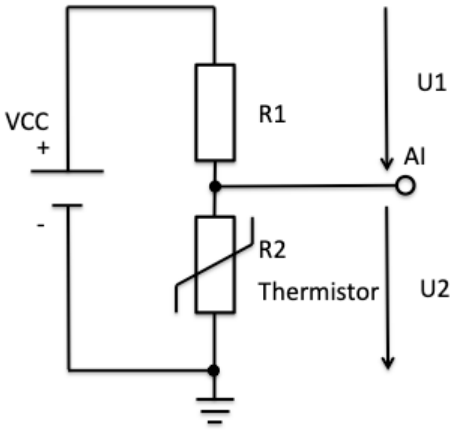
Die Zustandsvariable `switch` unterdrückt Ein- bzw. Ausschaltvorgänge, die mehrfach ausgelöst werden. So wird nach jedem Einschaltvorgang genau ein Ausschaltvorgang erzwungen und umgekehrt.

3. Kalibrierung eines Temperatursensors mit einem Thermistor

Material:

- TI-Innovator
- MuRata Thermistor aus dem TI-Innovator Breadboard Pack
- 10 kΩ Widerstand (braun/schwarz/orange)
- Steckkabel für Breadboard
- Breadboard

Schema:



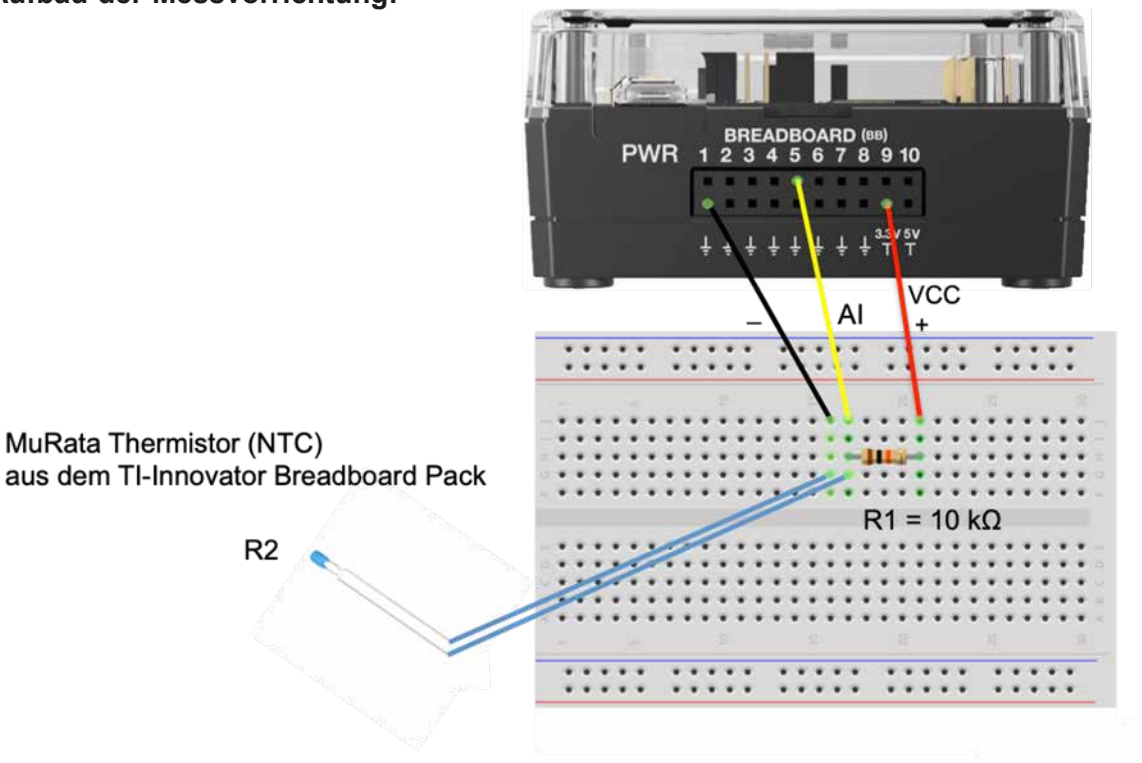
$$\frac{R1}{R2} = \frac{U1}{U2} = \frac{VCC - U2}{U2}$$

$$R2 = \frac{U2}{VCC - U2} R1$$

Methode:

Bezeichnen wir die Spannungen über den Widerständen R1 und R2 mit U1 resp. U2, dann gelten die oben stehenden Formeln.
Der Schaltkreis wird vom TI-Innovator gespeist (VCC = 3.3 V). Für R1 wird ein 10 kΩ Widerstand verwendet. **Somit kann der Widerstandswert des Thermistors (R2) via Messung der Spannung U2 mit Hilfe des Innovators ermittelt werden** (Anschluss AI (steht für Analog In)).
Der Widerstand des Thermistors hat eine nicht lineare Abhängigkeit von der Temperatur, die im folgenden beschrieben wird.
Bevor dieser Zusammenhang genauer angeschaut wird, soll der Innovator gemäss dem oben stehenden Schema angeschlossen werden.

Aufbau der Messvorrichtung:



Die Drahtverbindungen sind hier in rot, gelb und schwarz gezeigt. Die blauen Linien verweisen auf die Einsteckpunkte des Thermistors (bei diesem spielt die Polarität keine Rolle).

Anwendung der Steinhart-Hart Gleichung für einen Thermistor

Im Jahre 1968 schrieben Dr. John S. Steinhart und Dr. Stanley R. Hart, Wissenschaftler von der Carnegie Mellon University, eine Veröffentlichung, die als Standartgleichung für den Zusammenhang zwischen der absoluten Temperatur T (in Kelvin) und dem elektrischen Widerstand R (in Ohm) bei Thermistoren verwendet wird. Die allgemeine Steinhart-Hart Gleichung hat die Form

$$\frac{1}{T} = \sum_{i=0}^{\infty} a_i \ln^i(R) = a_0 + a_1 \ln(R) + a_2 \ln^2(R) + a_3 \ln^3(R) + \ldots$$

mit den vom Thermistor abhängigen Koeffizienten a_i.

Für praktische Anwendungen sind nur die Terme mit den Koeffizienten a_0, a_1 und a_3 von Bedeutung, die restlichen Koeffizienten sind in Relation so klein, dass sie vernachlässigt werden können. Entsprechend lässt sich die Gleichung (mit $a_0 = C_1$, $a_1 = C_2$ und $a_3 = C_3$) vereinfacht darstellen:

$$\frac{1}{T} = C_1 + C_2 \ln(R) + C_3 \ln^3(R)$$

In der Praxis können somit drei Widerstandsmessungen des Thermistors bei definierten Temperaturen vorgenommen werden, um diesen zu kalibrieren (üblicherweise an den zwei Endpunkten und dem Mittelpunkt des interessierenden Messbereichs). Alternativ kann eine Wertetabelle des entsprechenden Thermistors konsultiert werden.
Setzt man die drei gemessenen Punkte in die vereinfachte Gleichung ein, ergeben sich drei Gleichungen für die drei Steinhart-Hart Koeffizienten C_1, C_2 und C_3.

Wurden C_1, C_2 und C_3 durch Lösen dieses linearen Gleichungssystems ermittelt, so erhält man eine Funktion zwischen Temperatur und elektrischem Widerstand R, die für den ganzen Messbereich verwendet werden kann:

$$T = \left(C_1 + C_2 \ln(R) + C_3 \ln^3(R)\right)^{-1}$$

Die Messung der Spannung über dem Thermistor erlaubt uns nun, den Widerstand des Thermistors zu kennen und damit seine Temperatur zu berechnen.

Aufgaben und Lösungen:

Aufgabe 1 (zur Kalibrierung):

Die Kalibrierung kann durch Messen des Widerstandswerts des Thermistors bei (mindestens) drei verschiedenenTemperaturen erfolgen. Hier sollte darauf geachtet werden, dass das verwendete Thermometer möglichst genau ist und der Thermistor tatsächlich die gleiche Temperatur wie das Thermometer aufweist (thermischer Kontakt). Auf diese Vorgehensweise wird hier nicht weiter eingegangen.
Der Hersteller, **MuRata**, des hier verwendeten Thermistors aus dem TI-Innovator Breadboard Pack hat solche Messungen durchgeführt, die im Internet gefunden werden können.
Die genaue Typenbezeichnung dieses Thermistors lautet: **NXRT15XH103FA1B040**

a) Suche nach Tabellenwerten für Temperatur und Widerstand des MuRata NXRT15XH103FA1B040 Thermistors.
b) Erstelle mit Hilfe der gefundenen Daten aus a) eine Wertetabelle (Lists & Spreadsheet), die mindestens eine Spalte mit Temperaturwerten in Kelvin (K) und eine Spalte mit Widerstandswerten in Ohm (Ω) enthält.
c) Erstelle einen Graphen (Data & Statistics), der den Widerstand (in Ohm) als Funktion der Temperatur (in Kelvin) darstellt: Beobachte den Verlauf der gemessenen Punkte.

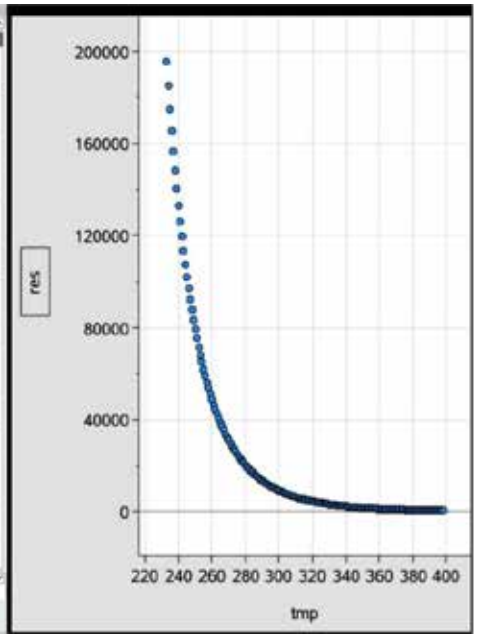
Lösung:

a) Durch Eingeben der genauen Typenbezeichnung und/oder des Herstellers ins Suchfeld einer Suchmaschine (z. B. Google) findet man relativ rasch die gesuchten Daten.
Hinweis: Die letzten drei Ziffern geben die Gesamtlänge des Thermistors an (hier: 040 bedeutet 40 mm). Alle Modelle mit NXRT15XH103FA1B___ haben die gleichen Eigenschaften.

b) und c)
Vom Hersteller werden verschiedene Grössen in Tabellenform (oder als Textdateien) gegeben (häufig sind die Temperaturen in °C gegeben). Das Übertragen dieser Werte geschieht am einfachsten durch Kopieren und Einfügen in die Wertetabelle in Lists & Spreadsheet.
Hinweis: Eine Wertetabelle in einer tns-Datei wird mit diesem Dokument zur Verfügung gestellt. Eine mögliche Lösung ist im nachfolgenden Bild gezeigt:
Der Graph zeigt eine nicht lineare Abhängigkeit des Widerstands als Funktion der Temperatur. Da der Widerstand mit zunehmender Temperatur abnimmt, spricht man von einem NTC Thermistor, wobei NTC für Negative Temperatur Coefficient steht.

	A res	B tmp	C tmpc	D
=				
1	195652	233.15	-40	
2	184917.	234.15	-39	
3	174845.	235.15	-38	
4	165391	236.15	-37	
5	156513.	237.15	-36	
6	148171	238.15	-35	
7	140330.	239.15	-34	
8	132958.	240.15	-33	
9	126022.	241.15	-32	
10	119494.	242.15	-31	
11	113347.	243.15	-30	
12	107565.	244.15	-29	
13	102116.	245.15	-28	

A res



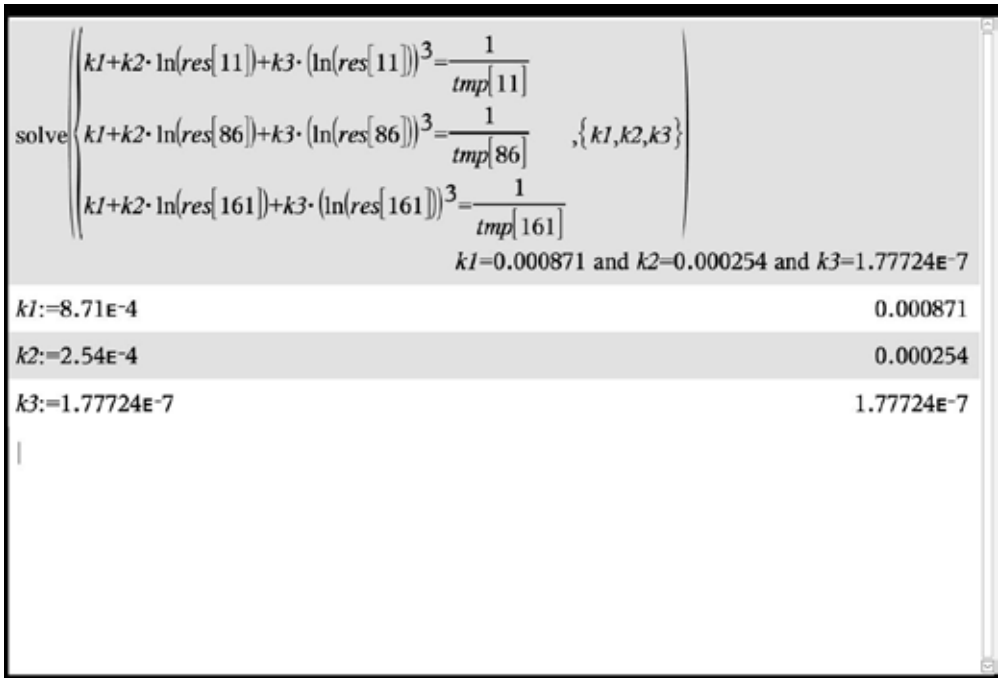
res: Widerstand in Ω
tmp: Temperatur in K
(tmpc: Temperatur in °C)

Aufgabe 2 (zur Kalibrierung):

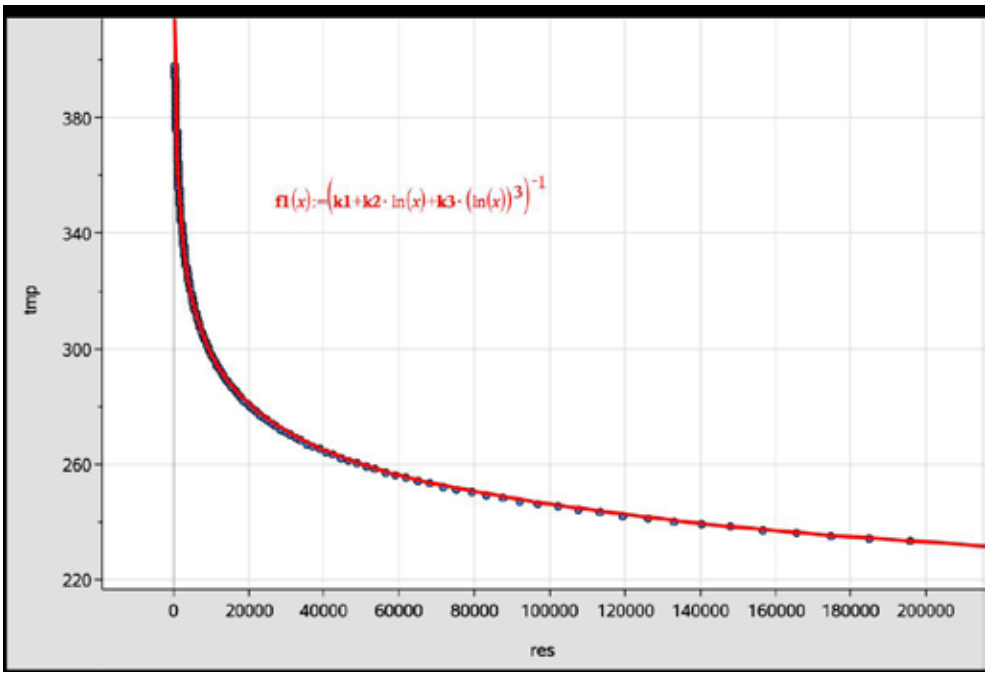
a) Wähle drei verschiedene Wertepaare für Temperatur (in Kelvin) und Widerstand (in Ohm) und erstelle damit ein Gleichungssystem für die drei gesuchten Steinhart-Hart Koeffizienten. Löse das Gleichungssystem (Calculator).
b) Erstelle einen Graphen mit den Temperaturwerten (T in Kelvin) auf der Ordinate (y-Achse) und den Widerstandswerten (R in Ohm) auf der Abszisse (x-Achse) und füge dem Graphen die die Funktion $T=\left(C_1+C_2\ln(R)+C_3\ln^3(R)\right)^{-1}$ hinzu.

Lösung:

Wählt man die Gleichung $\frac{1}{T}=C_1+C_2\ln(R)+C_3\ln^3(R)$, sowie drei geeignete Wertepaare aus der Wertetabelle, so erhält man ein lineares Gleichungssystem. Als Beispiel werden hier die Tabellenwerte bei 243.15 K (- 30 °C, Zeile 11), 318.15 K (45 °C, Zeile 86) und 393.15 K (120 °C, Zeile 161) gewählt. Die Lösungswerte werden drei globalen Variablen zugewiesen (k1 = C_1, k2 = C_2 und k3 = C_3).



b) Die Funktion T(R) ist hier mit f1(x) bezeichnet (k1 = C_1, k2 = C_2 und k3 = C_3):



Aufgabe 3 (zur Programmierung):

Erstelle ein Programm in TI Basic, dass die Messung der Temperatur in °C als Funktion der Zeit mit Hilfe der Messvorrichtung ermöglicht (Programmeditor).

Lösung:

Pin 5 des Innovators als analogen Eingang festlegen.
Globale Liste für Zeitwerte.
Globale Liste für Temperaturwerte.
Messung ein Mal pro Sekunde
Den lokalen Variablen c1,c2 und c3, die globalen Werte aus den Berechnungen zuweisen.
R2 + 100 Ohm (eingebaut im Innovator Hub).

Leerausgabe (Anzeige löschen).

Messen bis Escape Taste gedrückt wird.

Binären Wert (bei 14 Bit Auflösung zwischen 0 und 16383 = 2^14-1) für die Spannung abrufen und diese berechnen.
Widerstandswert des Thermistors berechnen.
Temperatur in Kelvin berechnen.
Temperatur in °C umrechnen.
Thermistorspannung anzeigen.

Temperatur in °C anzeigen.
Thermistorwiderstand anzeigen.

Zeitlistenelement 1 erhält Zeitwert 0...
In Temperaturliste eintragen.

```
Define temptherm()=
Prgm
:Send "CONNECT ANALOG.IN 1 TO BB5"
: zeit:={}
: temp:={}
: Local n:n:=0
: Local zeitintervall:zeitintervall:=1
: Local c1:c1:=k1
: Local c2:c2:=k2
: Local c3:c3:=k3
: Local rs:rs:=10100
: Local rt:rt:=0
: Local spannung:spannung:=0
: Local temperatur:temperatur:=0
: Local key:key:=""
: Local r
: For r,1,8
:  DispAt r,""
: EndFor
: DispAt 8,"Zum Abbrechen Esc-Taste
drücken"
: DispAt 1,"Thermistor Werte"
: While key≠"esc"
:   n:=n+1
:   Send "READ ANALOG.IN 1"
:   Get binary
:   spannung:=3.3*((binary)/(2^(14)-1))
:   rt:=((rs*spannung)/(3.3-spannung))
:   temperatur:=((1)/(c1+c2*l-
n(rt)+c3*(ln(rt))^(3)))
:   temperatur:=temperatur-273.15
:   DispAt 3,"Spannung = ",round(span-
nung,2),"V"
:   DispAt 4,"Temperatur = ",round(tem-
peratur,1),"°C"
:   DispAt 5,"Widerstand =
",round(rt,0),"Ω"
:   zeit[n]:=(n-1)*zeitintervall
:   temp[n]:=temperatur
:   Wait zeitintervall
:   key:=getKey()
:  EndWhile
:EndPrgm
```

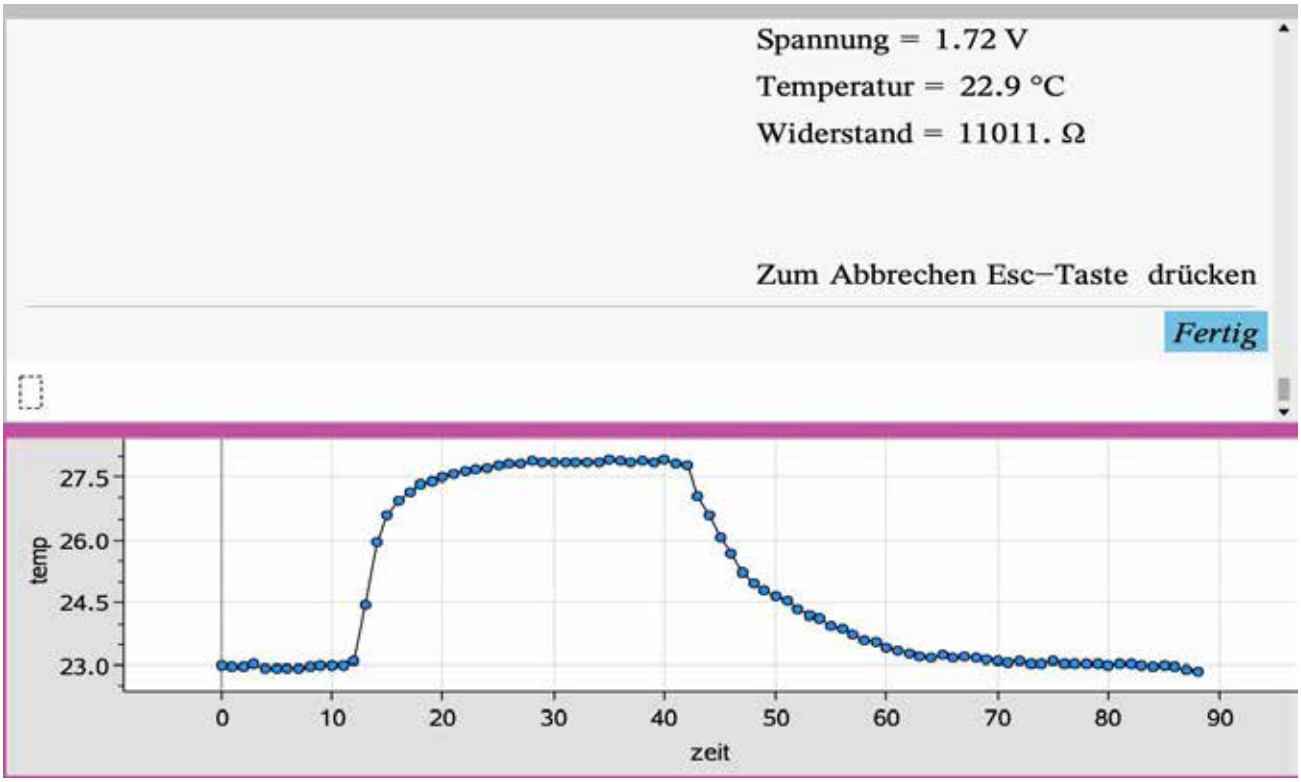
Aufgabe 4 (zu Aufbau und Messung):

Baue die Messvorrichtung auf und führe, mit Hilfe des Programms aus Aufgabe 3, eine zeitabhängige Temperaturmessung durch.

Lösung:

Im folgenden wird eine Messung für den folgenden Vorgang gezeigt:

Der Thermistor befindet sich bei Raumtemperatur und misst diese. Dann wird er zwischen Daumen und Zeigefinger geklemmt. Dadurch wird Körperwärme an den Thermistor übertragen und seine Temperatur steigt an. Nach einer gewissen Zeit erreicht er die Temperatur von Finger und Daumen und die Temperatur bleibt konstant. Danach wird der Thermistor wieder losgelassen und er kühlt sich wieder auf Raumtemperatur ab.



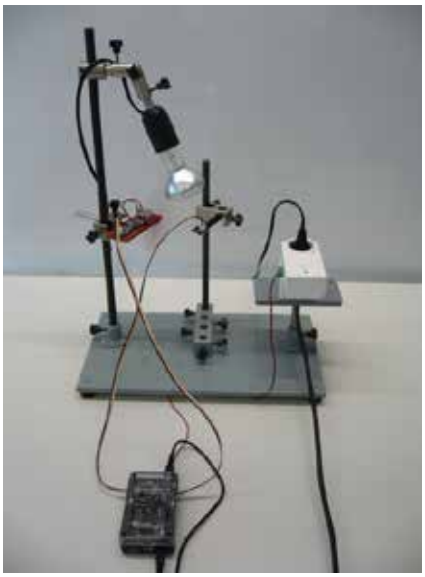
4. Temperaturregelung

Eine Temperaturregelung ist zu entwerfen, bei der ein Temperatursensor unterhalb einer Minimaltemperatur erwärmt und oberhalb einer Maximaltemperatur abgekühlt wird.

Überblick
- Die Temperaturregelung soll über einen beliebigen Zeitraum möglich sein.
- Die Minimal- bzw. Maximaltemperatur sind frei wählbar und durch den Nutzer einzugeben.
- Zur Temperaturregelung ist die so genannte Regelkurve als Temperatur(Zeit)-Diagramm aufzunehmen.
- Als Sensor wird der Grove-Sensor verwendet. Eine Kalibrierung des Grove-Sensors ist nicht notwendig.
- Zum Abkühlen soll der Arduino-Grove-Lüfter eingesetzt werden.

Externe Sensoren / Aktoren:
Als Heizquelle kann neben einer gewöhnlichen Glühlampe eine 230-V-Rotlichtlampe verwendet werden, die über ein optoelektronisches Schaltinterface vom TI Innovator ein- und ausgeschaltet wird. Dazu eignet sich das Schaltinterface SI230-3 (Komplettbausatz) des Elektronik-Versandhauses ELV.
Der kommerzielle Arduino-Grove-Lüfter wird wie ein Vibrationsmotor angesteuert.

Aufbau
Der Grove-Temperatursensor muss am Eingang IN3 angeschlossen werden. Der Arduino-Grove-Lüfter befindet sich am Ausgang OUT3. Zu seinem Betrieb benötigt der TI Innovator™ eine externe Stromversorgung (USB-Mikrosteckbuchse PWR).


Aufbau der Temperaturregelung


Lüfter und Temperatursensor

Aufgaben

Aufgabe 1:
Implementiere das Programm zur Temperaturregelung.
Dabei sollen die Zeit in Sekunden und die Temperatur in °C als Listen gespeichert werden. Die Zeitmessung wird mit der „Systemzeit“ des TI Innovators™ realisiert.
Der Programmabbruch soll mit der Taste „esc“ erfolgen.

Aufgabe 2:
Ermittle mit dem Statistik-Werkzeug „Data & Statistics“ die gesuchte Regelkurve.

Aufgabe 3:
Bestimme mit einer Funktion den Mittelwert aller Temperaturen, die in der Phase der Temperaturregelung gemessen wurden. Entwickle dazu einen Algorithmus, der nur die infrage kommenden Temperaturmesswerte auswählt.
Überprüfe, ob der Mittelwert tatsächlich zwischen der eingegebenen Minimal- bzw. Maximaltemperatur liegt.

Programmstrukturen und Hinweise zu den Aufgaben

Aufgabe 1:
Mit einer geeigneten Variablen muss zwischen den drei Temperaturzuständen $T < T_{min}$, $T_{min} \leq T \leq T_{max}$ und $T_{max} < T$ **unterschieden werden.**

Löschen globaler Variablen
Deklarationen lokaler Variablen:
i ... Index der Listenelemente
dt, t0, t ... Zeitvariablen
tempmin ... Temperaturminimum
tempmax ... Temperaturmaximum
switch ... Zustandsvariable

Verbindung zum Grove-Temperatursensor.
Verbindung zum Schaltinterface.
Verbindung zum Lüfter.
Bestimmung der Startzeit.

Programmabbruch mit „esc“.
Zeitmessung.

Berechnung der Messdauer in s.
Temperaturmessung.

Ausgabe der Messwerte.

Heizphase für $T < T_{min}$.
Einschalten des Schaltinterface.
Ausschalten des Lüfters.

Zwischenphase $T_{min} \leq T \leq T_{max}$.

Schaltinterface und Lüfter sind ausgeschaltet.

Kühlphase $T_{max} < T$.

```
Define tempregelung()=
Prgm
:DelVar temp,time
:Local i,dt,t0,t,
:Local tempmin,tempmax,switch
:dt:=1.: tempmin:=35: tempmax:=36
:Request "Temp-min in °C",tempmin
:Request "Temp-max in °C",tempmax
:DispAt 3,"Abbruch mit <esc>"
:Send "CONNECT TEMPERATURE 1 TO IN 3
       AS NTC"
:Send "CONNECT DIGITAL.OUT 1 TO BB 1"
:Send "CONNECT VIB.MOTOR 1 TO OUT 3"
:Send "READ TIMER"
:Get t0
:i:=1: switch:=0
:While getKey()≠"esc"
: Send "READ TIMER"
: Get t
: time[i]:=round(((t-t0)/(1000)),1)
: Send "READ TEMPERATURE 1"
: Get temp[i]
: DispAt 4,"Zeit: ",time[i]," s"
: DispAt 5,"Temp: ",
            round(temp[i],2)," °C"
: If temp[i]≤tempmin and switch≤0 Then
:  Send "SET DIGITAL.OUT 1 ON"
:  Send "SET VIB.MOTOR 1 OFF"
:  switch:=1
: EndIf
: If temp[i]>tempmin
     and temp[i]<tempmax
     and switch≠0 Then
:  Send "SET DIGITAL.OUT 1 OFF"
:  Send "SET VIB.MOTOR 1 OFF"
:  switch:=0
: EndIf
```

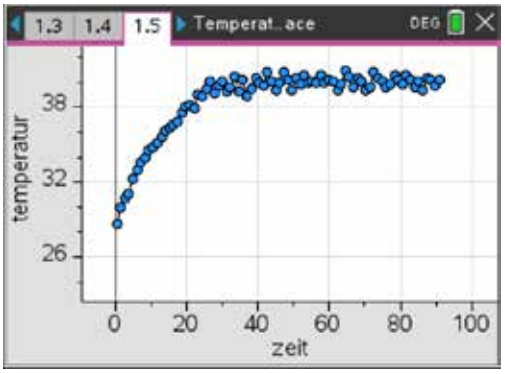
Ausschalten des Schaltinterface.
Einschalten des Lüfters.

Trennung der angeschlossenen Hardware.

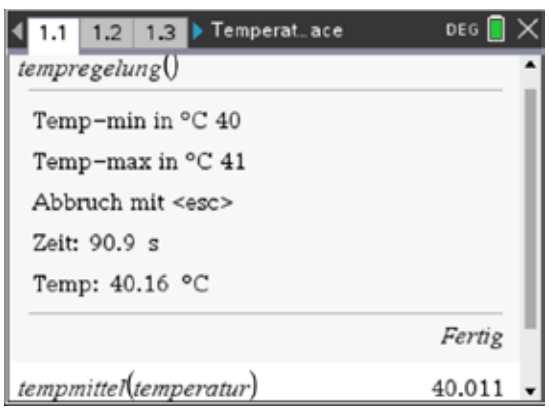
```
: If temp[i]≥tempmax and switch≥0 Then
:  Send „SET DIGITAL.OUT 1 OFF“
:  Send „SET VIB.MOTOR 1 TO 255“
:  switch:=-1
: EndIf
: i:=i+1
: Wait dt
:EndWhile
:Send „DISCONNECT TEMPERATURE 1“
:Send „DISCONNECT DIGITAL.OUT 1“
:Send „DISCONNECT VIB.MOTOR 1“
:EndPrgm
```

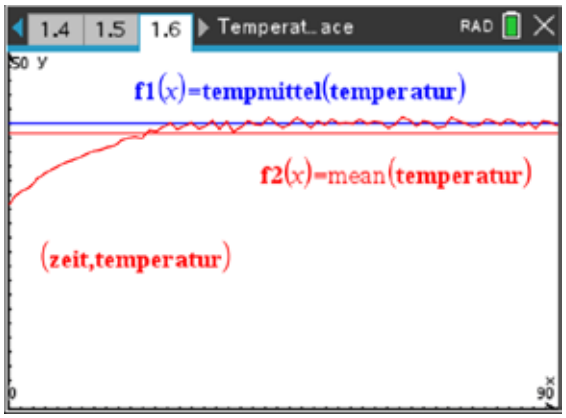
Aufgabe 2:

Zur abgebildeten Regelkurve wurden eine Minimaltemperatur von 40.0 °C und eine Maximaltemperatur von 41.0 °C eingegeben. Eine deutliche Abkühlung tritt ein, wenn sich oberhalb der Maximaltemperatur der Lüfter einschaltet.



Aufgabe 3:
Eine einfacher Lösungsalgorithmus besteht darin, die Anstiege zwischen den Temperaturmesswerten T[i], T[i+1] und T[i+1], T[i+2] zu vergleichen. Ein negatives Produkt dieser Anstiege entspricht einem Vorzeichenwechsel. Damit könnte ein erster lokaler Extremwert gefunden sein. Der Mittelwert aller weiteren Temperaturmesswerte entspricht dann dem gesuchten Mittelwert der Temperaturregelung.
Bei sehr großen Messwertschwankungen können auch mehrere Vorzeichenwechsel „abgezählt" werden, bevor die Mittelwertsbestimmung vorgenommen wird.





Funktionsaufruf / Darstellung der Temperaturmittel im Vergleich zur Regelkurve

```
Define tempmittel(temp)=
Func
:Local i,j,maxindex,temp0,whlg
:maxindex:=dim(temp)
:If maxindex≤2 Then
: Return mean(temp)
:Else
: i:=1: whlg:=10
: While i<maxindex-2 and whlg>0
:  If (temp[i+1]-temp[i])*(temp[i+2]-temp[i+1])≤0 Then
:   whlg:=whlg-1
:  EndIf
:  i:=i+1
: EndWhile
: For j,i,maxindex
:  temp0[j-i+1]:=temp[j]
: EndFor
: Return mean(temp0)
:EndIf
:EndFunc
```

5. Bewegungsmelder

Bewegungsmelder kennen wir aus zahlreichen Anwendungen im Alltag. Sie reagieren auf Ortsänderungen einer (Infrarot-)Wärmequelle und ermöglichen z.B. bei Dunkelheit das zeitlich begrenzte Einschalten einer Hausbeleuchtung.

Externe Sensoren / Aktoren:
- Arduino-Grove PIR-Bewegungssensor, TM2291 (https://www.reichelt.de)
- ELV 230-V-Schaltinterface SI230-3, Komplettbausatz (https://www.elv.de)
- interner Lichtsensor BRIGHTNESS

Aufbau:
- Bewegungssensor an Digitaleingang IN 1
- Schaltinterface an Digitalausgang BB 1

Aufgabe 1:

Implementiere das einfache Modell eines Bewegungsmelders zunächst unabhängig von der Umgebungshelligkeit.
Die Schaltzeit für die Beleuchtung soll frei wählbar in Sekunden eingegeben werden.

Lösung:

Deklarationen lokaler Variablen:
t, dt ... Schaltzeit, kleines Zeitintervall
pegel ... Messwert des Sensors
switch ... Zustandsvariable
Eingabe der Schaltzeit in Sekunden.

Initialisierung des digitalen Ein- und Ausgangs.

Programmstruktur zur permanenten Wiederholung des Regelalgorithmus, der Abbruch erfolgt mit ESC.
Wertübergabe und Ausgabe des Messwertes des Bewegungssensors.
Falls dieser Messwert 1 ist und das Einschalten möglich ist, wird der digitale Ausgang eingeschaltet, die Schaltzeit abgewartet und ein weiterer Einschalt-vorgang unterdrückt.
Bei dem Messwert 0 und bestehendem Einschaltzustand wird der Digitalausgang ausgeschaltet und der Bewegungsmelder für einen weiteren Einschaltvorgang freigegeben.

Freigabe des digitalen Ein- und Ausgangs.

```
Define bewegung1()=
Prgm
:Local t,dt,pegel,switch
:t:=2: dt:=0.2
:Request „Einschaltdauer in s“,t
:DispAt 1,“Bewegungsmelder“
:DispAt 2,“Abbruch mit <esc>“
:Send “CONNECT DIGITAL.IN 1 TO IN 1”
:Send “CONNECT DIGITAL.OUT 2 TO BB
1”
:switch:=true
:While getKey()≠”esc”
: Send “READ DIGITAL.IN 1”
: Get pegel
: DispAt 3,”Messwert: “,pegel
: If pegel=1 and switch Then
:  Send “SET DIGITAL.OUT 2 ON”
:  Wait t
:  switch:=false
: EndIf
: If pegel=0 and not switch Then
:  Send “SET DIGITAL.OUT 2 OFF”
:  switch:=true
: EndIf
: Wait dt
:EndWhile
:Send “DISCONNECT DIGITAL.IN 1”
:Send „DISCONNECT DIGITAL.OUT 2“
:EndPrgm
```

Hinweis:

Die Zustandsvariable `switch` unterdrückt Ein- bzw. Ausschaltvorgänge, die mehrfach ausgelöst werden. So wird nach jedem Einschaltvorgang genau ein Ausschaltvorgang erzwungen und umgekehrt.

Aufgabe 2:
Erweitere das Programm derart, dass der Bewegungsmelder nur unterhalb eines frei wählbaren Helligkeitsminimums aktiv ist. Nutze dazu den internen Helligkeitssensor (BRIGHTNESS) des TI Innovator™ Hubs.
Beachte dabei, dass ein evtl. notwendiger Ausschaltvorgang unabhängig von der Umgebungshelligkeit erfolgen muss.

Lösung:

Deklaration weiterer lokaler Variablen:
hell ... Helligkeitsmesswert,
hellmin ... Helligkeitsminimum

Eingabe des Helligkeitsminimums.

Messung, Wertübergabe und Ausgabe des Helligkeitsmesswertes.

Falls der Helligkeitswert unterhalb des Helligkeitsminimums liegt oder (noch) ein Einschaltzustand vorliegt, dann starte den o.g. Algorithmus zur Bewegungsmeldung.

```
Define bewegung2()=
Prgm
:Local hell,hellmin
:Local t,dt,pegel,switch
:hell:=0: hellmin:=3
:t:=2: dt:=0.2: pegel:=0
:Request „Helligkeitsminimum“,hellmin
:Request „Einschaltdauer in s“,t
:DispAt 1,“Bewegungsmelder“
:DispAt 2,“Abbruch mit <esc>“
:DispAt 3,“Helligkeit: „,round(hell,1)
:DispAt 4,”Messwert: “,pegel
:Send “CONNECT DIGITAL.IN 1 TO IN 1”
:Send “CONNECT DIGITAL.OUT 2 TO BB 1”
:switch:=true
:While getKey()≠”esc”
: Send “READ BRIGHTNESS”
: Get hell
: DispAt 3,”Helligkeit:
“,round(hell,1)
: If hell<hellmin or not switch Then
:  Send “READ DIGITAL.IN 1”
:  Get pegel
:  DispAt 4,”Messwert: “,pegel
:  If pegel=1 and switch Then
:   Send “SET DIGITAL.OUT 2 ON”
:   Wait t
:   switch:=false
:  EndIf
:  If pegel=0 and not switch Then
:   Send “SET DIGITAL.OUT 2 OFF”
:   switch:=true
:  EndIf
: EndIf
: Wait dt
:EndWhile
:Send “DISCONNECT DIGITAL.IN 1”
:Send „DISCONNECT DIGITAL.OUT 2“
:EndPrgm
```

Hinweis:

Eine interessante Alternative zum Gebrauch des Schaltinterfaces mit einer Glühlampe stellt der Einsatz des RGB-Moduls dar. Nachfolgend wird die Lösung zur **Aufgabe 2** mit dem RGB-Modul angegeben:

```
Define bewegung2()=
Prgm
:Local hell,hellmin
:Local t,dt,pwr,pegel,switch
:hell:=0: hellmin:=3
:t:=2: dt:=0.2: pegel:=0
:Request „Helligkeitsminimum“,hellmin
:Request „Einschaltdauer in s“,t
:DispAt 1,“Bewegungsmelder“
:DispAt 2,“Abbruch mit <esc>“
:DispAt 3,“Helligkeit: „,round(hell,1)
:DispAt 4,“Messwert: „,pegel
:Send „CONNECT DIGITAL.IN 1 TO IN 1“
```

Abfrage des PWR-Anschlusses.

```
:Send „READ PWR“
:Get pwr
```

Bei einer zusätzlichen Stromversorgung wird das RGB-Modul im Power-Modus initialisiert, anderenfalls im normalen Betriebsmodus.

```
:If pwr=1 Then
: Send „CONNECT RGB LAMP“
:Else
: Send „CONNECT RGB“
:EndIf
:switch:=true
:While getKey()≠“esc“
: Send „READ BRIGHTNESS“
: Get hell
: DispAt 3,“Helligkeit:
„,round(hell,1)
: If hell<hellmin or not switch Then
:  Send „READ DIGITAL.IN 1“
:  Get pegel
:  DispAt 4,“Messwert: „,pegel
```

Einschalten des RGB-Moduls.
Alle LEDs leuchten in weißer Farbe.

```
:  If pegel=1 and switch Then
:   Send „SET RGB ALL 255 255 255“
:   Wait t
:   switch:=false
:  EndIf
```

Ausschalten des RGB-Moduls.

```
:  If pegel=0 and not switch Then
:   Send „SET RGB ALL 0 0 0“
:   switch:=true
:  EndIf
: EndIf
: Wait dt
```

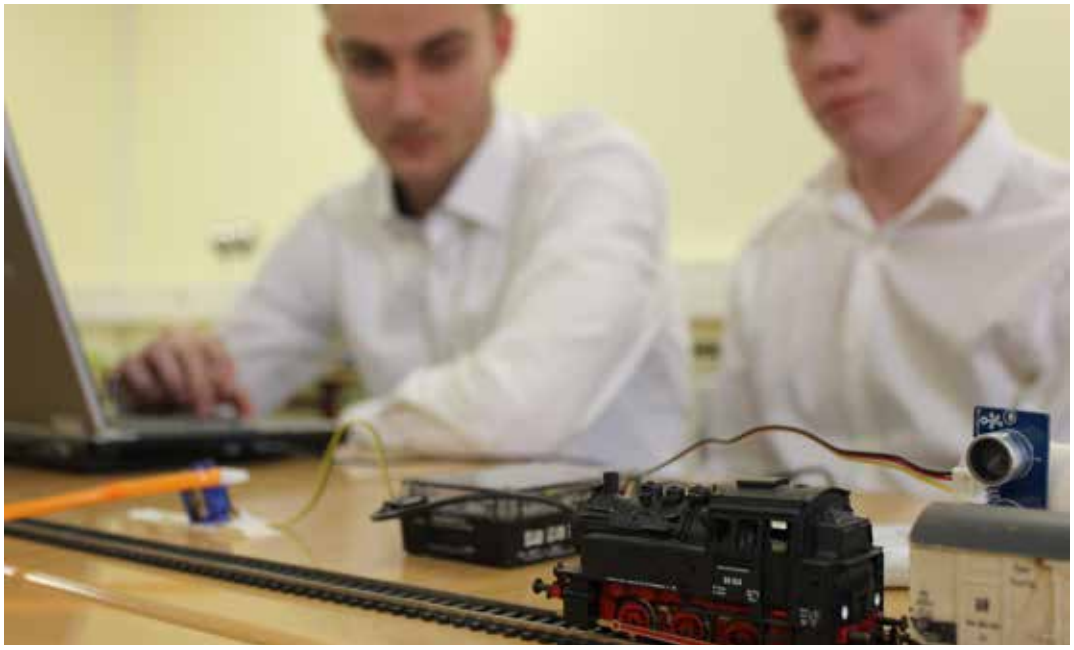
Freigabe des RGB-Moduls.

```
:EndWhile
:Send „DISCONNECT DIGITAL.IN 1“
:Send „DISCONNECT RGB“
:EndPrgm
```

6. Automatische Bahnschranke

Grundidee

An einem anschaulichen und realitätsnahen Beispiel soll ein einfacher Regelkreis mit verschiedenen Stellgliedern mittels des TI Innovator™ realisiert werden. An einem Bahnübergang soll dabei der TI Innovator™ die Steuerung der Bahnschranke und des Blocksignals übernehmen. Ein aktive Zugbeeinflussung ist nicht vorgesehen, würde sich aber in einer optionalen Erweiterung realisieren lassen.



Überblick

- Eine einfache Modelleisenbahn soll zur Veranschaulichung dienen (siehe dazu auch die pädagogischen Kommentare).
- Der herankommende Zug soll über einen Ultraschallsensor erkannt werden.
- Die Bahnschranke soll über eine Servo verstellt werden, eine rote LED soll die Warnblinkanlage und der Lautsprecher die Warnglocke simulieren.
- Ein Blocksignal soll (ohne aktive Zugbeeinflussung) den Streckenabschnitt mit dem Bahnübergang freigeben, wenn die Schranke geschlossen wird.

Der Aufbau ist so gestaltet, dass möglichst viele unterschiedliche Typen von I/0 Ports verwendet werden, um die Programmieraufgaben möglichst abwechslungsreich zu gestalten.

Modellbahn

Um eine anschauliche, realitätsnahe Umsetzung der Aufgabe zur erreichen, wurde eine Holzeisenbahn mit selbstfahrender Lok in einem einfachen Oval aufgebaut. Um die Verstellung der Bahnschranke zu erreichen, wurde ein fertiger Bahnübergang der Firma Brio mit einem Modellbauservo ergänzt. Die Schranken könnten aber auch wesentlich kostengünstiger über einen Pappstreifen, der auf das Servohorn geklebt wird, realisiert werden.

Hardware

Neben dem TI Handheld, in meinen Fall ein TI Nspire™ CX CAS, und dem TI Innovator™ werden für den Aufbau ein Ultraschall Abstandssensor, ein Servomotor, 3 LEDs (2x rot und 1x grün), einige Kabel und ein Powerpack für den Betrieb des Servos benötigt.

Programmstruktur und Regelkreiselemente

Kontinuierlicher Ablauf mit Abbruchbedingungen

Grundsätzlich soll das Programm kontinuierlich ablaufen können. Um aber einer Endlosschleife vorzubeugen, soll eine Abbruchbedingung integriert sein. Dafür lässt sich beispielsweise der im TI Nspire™ integrierten Helligkeitssensor verwenden.
In einer ersten Schleife wird also die Umgebungshelligkeit gemessen. Bei „normalen“ Lichtbedingungen wird ein Programmdurchlauf ausgeführt, bei Dunkelheit wird das Programm beendet. So lässt sich das Programm kontinuierlich ausführen und durch einfaches Abdecken des Sensors mit dem Finger beenden.

Abfrage Abstandssensor

Als Input wird nun der am Abstandssensor gemessen Abstand ausgewertet. Liegt dieser über einem Schwellwert, hier 5cm, geht die Steuerung von einem leeren Gleisabschnitt aus. Der Servomotor öffnet die Schranke und am Blocksignal wird „Halt“ (rote LED) signalisiert.
Andernfalls wird der Zustand „vorbeifahrender Zug“ angenommen. Die Schranke wird über den Servomotor geschlossen, der Warnblinker (rote LED, blinkt) und die Warnglocke werden aktiviert und am Blocksignal „Fahrt frei“ (grüne LED) signalisiert. Nach einer vorgegebenen Zeitspanne wird eine neuerliche Messung durchgeführt.

Vereinfachungen und Hinweise

In dieser ersten Umsetzung wurden bewusst mehrere Vereinfachungen getroffen, die hier nicht unerwähnt bleiben sollen.

Fahrtrichtung

Da nur ein Abstandssensor verwendet wurde, funktioniert die Steuerung nur in einer Fahrtrichtung. Der Zug muss den Sensor daher mit Fahrtrichtung zum Bahnübergang passieren, also vor dem Bahnübergang.

Öffnen der Schranke

Um eine möglichst einfache Programmstruktur zu erreichen, wird das Öffnen über einen Timer gesteuert, der auf die Geschwindigkeit und die Länge des Zugs abgestimmt ist. Eine aktive Überprüfung, ob der Bahnübergang frei ist, wird nicht durchgeführt.

Zugbeeinflussung

Das Blocksignal hat hier keinen Einfluss auf den vorbeifahrenden Zug, da eine Holzeisenbahn verwendet wird.

Lösung:

```
Define bahn()=
Prgm
 Send "BEGIN"
 DelVar iostr.str0
 GetStr iostr.str0
 Disp iostr.str0
 Send "CONNECT LED 1 TO BB1"
 Send "CONNECT LED 2 TO BB2"
 Send "CONNECT LED 3 TO BB3"
 Send "CONNECT RANGER 1 TO IN 1"
 Send "CONNECT SERVO 1 TO OUT 3"
 Send "SET SERVO 1 TO 10"
 Send "SET LED 1 OFF"
 Send "SET LED 2 OFF"
 Send "SET LED 3 OFF"
 d:=1
 l:=1
 Disp „Schranke aktiv"
 Loop
  Send „READ BRIGHTNESS „
  Get l
  If l>0.5 Then
   Send „READ RANGER 1"
   Get d
   If d>0.05 Then
    Send „SET LED 1 OFF"
    Send „SET LED 2 OFF"

    Send "SET LED 3 ON"
    Send "SET SERVO 1 TO 70"
   Else
    Send "SET LED 2 ON"
    Send "SET LED 3 OFF"
    Send "SET SERVO 1 TO 10"
    For n,1,7
     Send "SET LED 1 ON"
     Send "SET SOUND 300 TIME 0.7"
     Wait 0.7
     Send "SET LED 1 OFF"
     Wait 0.3
    EndFor
   EndIf
  Else
   Disp „Schranke deaktiviert"
   Exit
  EndIf
 EndLoop
 Send „DISCONNECT LED 1"
 Send „DISCONNECT LED 2"
 Send „DISCONNECT LED 3"
 Send „DISCONNECT RANGER 1"
 Send „DISCONNECT SERVO 1"
EndPrgm
```

Initialisierung der Ports

Festlegung des Anfangszustandes

Schleifenbeginn
Einlesen der Helligkeit *l* (für Programmabbruch)

Einlesen der Entfernung *d* bei nicht-abgedunkeltem Helligkeitssensor

d>0,05m : kein Zug
Blocksignal grün (LED 3
Schranke auf

Sonst:
Blocksignal rot (LED 2)
Schranke geschlossen
LED 1 an Schranke blinkt und Warnsignal ertönt insgesamt 7mal

Helligkeit l<0,5: Schleifenabbruch

Freigeben der Ports

7. Beschleunigte Bewegung auf einer geneigten Ebene

Grundidee

Die gleichmäßig beschleunigte Bewegung eines Experimentierwagens ist zu untersuchen. Mit dem TI-Innovator™ soll die Versuchsdurchführung unterstützt werden, in dem die erforderliche Kurzzeitmessung mit einer externen Digitaluhr gesteuert wird.

Überblick

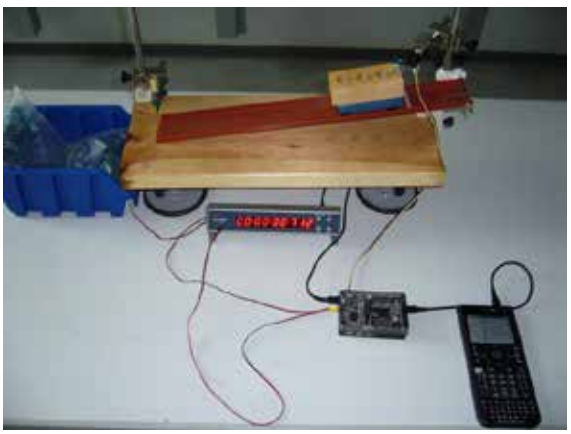
- Der Experimentierwagen wird mit einem Haltemagneten fixiert, der durch den TI-Innovator™ freigegeben werden kann.
- Bei Freigabe des Experimentierwagens wird die Digitaluhr gestartet.
- Das Stopp-Signal soll mit einem mechanischen Schalter ausgelöst werden.
- Das Experiment soll beliebig oft wiederholbar sein.
- Mit dem TI-Nspire™ sollen bei Bedarf die Messwerte für den Weg und die Zeit erfasst und in entsprechenden Listen zur Auswertung bereit gehalten werden.

Hardware

Als Haltemagnet kommt das Grove-Relais zum Einsatz. Zur Kurzeitmessung ist die kommerzielle LED-Stoppuhr LSU 100 des Elektronik-Versands ELV geeignet.
Prinzipiell lassen sich aber auch vergleichbare Modelle verwenden, die über externe Start-/Stopp-Anschlüsse in TTL-Pegeln verfügen.
Alle weiteren Experimentiergeräte können den gängigen Schülerexperimentiersätzen entnommen werden. Am Experimentierwagen muss ein kleiner ferromagnetischer Aufnehmer (Schraube, Nagel o. ä.) für den Haltemagneten angebracht sein.

Aufbau

Der Haltemagnet sollte am Ausgang OUT3 (5 V) angeschlossen werden. Zu seinem Betrieb benötigt der TI Innovator™ eine externe Stromversorgung (USB-Mikrosteckbuchse PWR).
Er wird wie ein Relais zu- oder abgeschaltet.
Das Startsignal für die externe Digitaluhr wird am Breadboard-Anschluss BB1 generiert.
Im vorliegenden Fall kann die Anzeige der Uhr elektronisch zurückgesetzt werden Das erforderlich Signal wird am Breadboard-Anschluss BB2 erzeugt.
Natürlich ist das Zurücksetzen auch manuell möglich. Dann kann auf diese zweite Signalleitung verzichtet werden.



Aufgabe 1:

Zunächst soll der Versuchsaufbau getestet werden. I. A. sind Digitaluhren mit externen Start-/Stopp-Anschlüssen flankengesteuert. Deshalb ist es notwendig, den Betriebsmodus zu ermitteln, bei dem je eine aufsteigende Signalflanke die Digitaluhr startet bzw. stoppt. Benutze dazu das nachfolgende **Testprogramm**:

	`Define zeitmessung1()=`
	`Prgm`
Definieren lokaler Variablen.	`Local key,test`
Verbindung zur Digitaluhr.	`Send „CONNECT DIGITAL.OUT 1 TO BB 1"`
	`Send „CONNECT DIGITAL.OUT 2 TO BB 2"`
Verbindung zum Haltemagneten.	`Send „CONNECT RELAY 1 TO OUT 3"`
Einschalten des Haltemagneten.	`Send „SET RELAY 1 ON"`
	`key=:"n": test:=true`
Programmabbruch mit „esc".	`While key≠"esc"`
Tastaturabfrage.	`  key:=getKey()`
	`  If key="enter" and test Then`
Start der Zeitmessung mit der Enter-Taste (Null-Setzung, Abschalten des Haltemagneten, Start der Digitaluhr).	`    Send „SET DIGITAL.OUT 2 ON"`
	`    Send „SET RELAY 1 OFF"`
	`    Send „SET DIGITAL.OUT 1 ON"`
	`    test:=false`
	`  EndIf`
Kurze Zeitverzögerung	`  Wait 0.2`
	`  If not test Then`
Zuschalten des Haltemagneten.	`    Send „SET RELAY 1 ON"`
Zurücksetzen der Signalpegel.	`    Send „SET DIGITAL.OUT 1 OFF"`
	`    Send „SET DIGITAL.OUT 2 OFF"`
	`    test:=true`
	`  EndIf`
	`EndWhile`
Trennung der angeschlossenen Hardware.	`Send „DISCONNECT DIGITAL.OUT 1"`
	`Send „DISCONNECT DIGITAL.OUT 2"`
	`Send „DISCONNECT RELAY 1"`
	`EndPrgm`

Aufgabe 2:

Ergänze im vorliegenden Testprogramm die Möglichkeit, Werte zur Weg- bzw. Zeitmessung einzugeben und diese in Listen zu speichern.

Aufgabe 3:

Ermittle nun mit der komplettierten Versuchsanordnung das s(t)-Diagramm für die geneigte Ebene und bestimme daraus die Beschleunigung des Experimentierwagens.

Hinweise und Lösungen

Aufgabe 1:

In der LED-Stoppuhr LSU 100 muss der Modus 7 eingestellt werden, um eine korrekte Start-/ Stopp-Funktion zu realisieren. Dann sollten reproduzierbare Ablaufzeiten zu einer konstanten Ablaufstrecke messbar sein.

Aufgabe 2:

Die Messwerteingabe erfolgt nach einem speziellen Tastendruck (hier Taste „m“).
Da der Experimentierwagen ohne Anfangsgeschwindigkeit gestartet wird, ist es sinnvoll, neben den Messwerten für den Weg s und die Zeit t den Quotienten für die Beschleunigung $a = \frac{2 \cdot s}{t^2}$ in einer weiteren Liste zu speichern.

Löschen globaler (Feld-)Variablen.

Zuweisung der Ports

Definieren des Feldindex sowie einem Zeitintervall für Programmunterbrechungen.

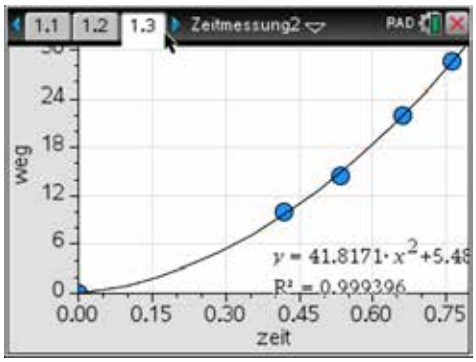
Falls Taste „m“ gedrückt wird,
- Inkrementieren des Feldindex,
- Wegeingabe in cm,
- Zeiteingabe in s,
- Berechnung der Beschleunigung.

```
Define zeitmessung2()=
Prgm
DelVar weg,zeit,beschl
Local key,test,i,dt
Send „CONNECT DIGITAL.OUT 1 TO BB 1“
Send „CONNECT DIGITAL.OUT 2 TO BB 2“
Send „CONNECT RELAY 1 TO OUT 3“
Send „SET RELAY 1 ON“
i:=0: dt:=0.1
key:=“n“: test:=true
While key≠“esc“
 key:=getKey()
 If key=“enter“ and test Then
   Send „SET DIGITAL.OUT 2 ON“
   Send „SET DIGITAL.OUT 1 ON“
   Send „SET RELAY 1 OFF“
   test:=false
 EndIf
 Wait dt
 If key=“m“ Then
  i:=i+1
  Request „Weg in cm“,weg[i]
  Request „Zeit in s“,zeit[i]
  beschl[i]:=((2*weg[i])/
  (100*zeit[i]^(2)))
 EndIf
 Wait dt
 If not test Then
  Send „SET DIGITAL.OUT 2 OFF“
  Send „SET DIGITAL.OUT 1 OFF“
  Send „SET RELAY 1 ON“
  test:=true
 EndIf
EndWhile
Send „DISCONNECT DIGITAL.OUT 1“
Send „DISCONNECT DIGITAL.OUT 2“
Send „DISCONNECT RELAY 1“
EndPrgm
```

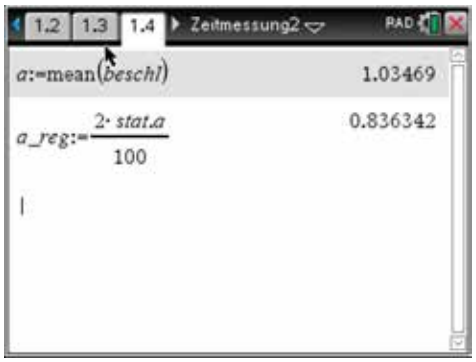
Aufgabe 3:

Zur Weg-Zeit-Messung ist das Wertepaar s_0 = 0 cm bzw. t_0 = 0 s sinnvoll. Nun kann die Beschleunigung sowohl als Mittelwert aller Einzelwerte als auch aus der quadratischen Regression bestimmt werden.

Messwerte



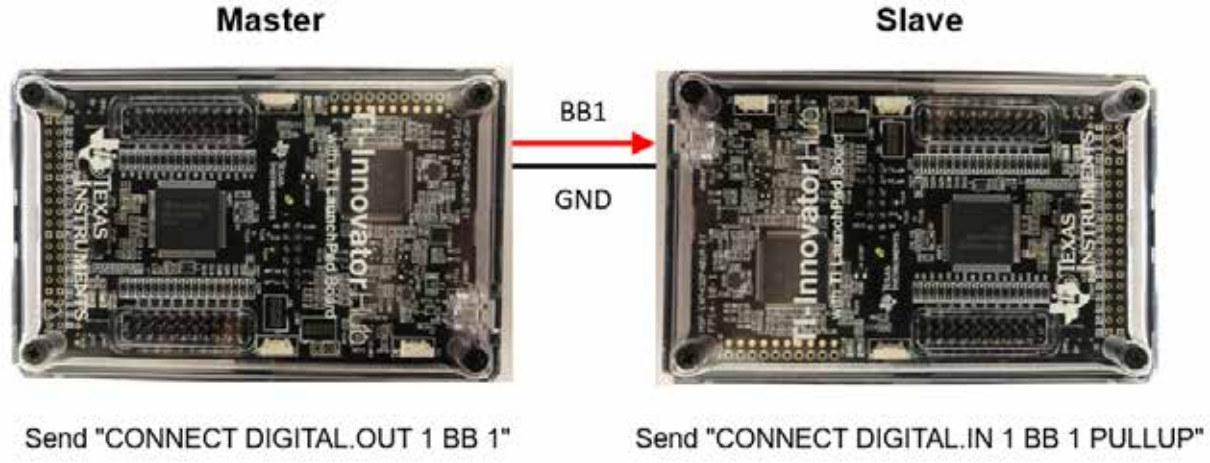
s(t) - Diagramm



Bestimmung der Beschleunigung

8. Vernetzen zweier Hubs über BB-Ports

TI-Innovator™ Systeme können per Kabel miteinander vernetzt werden. In der einfachsten Form erfolgt die Vernetzung über eine Signalleitung und eine Masseleitung (GND).



Die Master-Slave Verbindung ist für Steuerungen geeignet, bei denen der Slave einfache Aufgaben ausführen soll, die immer in einem bestimmten Zeitraum erledigt werden können. Der Master gibt den Start an den Slave weiter, indem er den Logikzustand seines BB1 Ausgangs ändert. Eine Rückmeldung des Slaves an den Master ist bei solchen Aufgaben nicht erforderlich. Wenn es sich um komplexere Steuerungen handelt oder eine bidirektionale Kommunikation der Hubs erforderlich ist, muss man zwei Signalleitungen verwenden. Sollen Daten übertragen werden, erfolgt die Kommunikation der Hubs seriell über die Signalleitungen, nach einem individuell zu erstellenden Protokoll.
Die BB pins verwenden eine 3.3V LV-TTL Logik, die Abstände von bis zu 2m zwischen den Hubs störungsfrei übertragen kann. Es empfiehlt sich, den Eingang des Slave Hubs mit einem Pullup Widerstand zu programmieren, um ein Schwingen der Signalleitung bei offenem Eingang zu vermeiden.

Programmieraufgabe für eine Master-Slave Verbindung:

Erstelle ein Programm, bei dem ein Master-Hub über eine Signalleitung im Sekundentakt die rote Leuchtdiode in einem Slave-Hub fortlaufend ein- bzw. ausschaltet, bis am Master eine beliebige Taste auf dem Rechner gedrückt wird.

Lösung:

Programm für Master-Hub:

BB1 als Ausgang
Endlosschleife, bis Taste gedrückt wird

```
Define msmaster()=
Prgm
:Send "CONNECT DIGITAL.OUT 1 BB 1"
:While getKey()=""
:  Send "SET DIGITAL.OUT 1 ON"
:  Wait 1
:  Send "SET DIGITAL.OUT 1 OFF"
:  Wait 1
:EndWhile
:EndPrgm
```

Programm für den Slave Hub:

BB1 als Eingang mit Pullup Widerstand
Endlosschleife, bis Taste gedrückt wird
Zustand des Eingangs lesen
1 bedeutet, am Eingang liegt 'HIGH' an
LED soll eingeschaltet sein

Sonst:
Ausschalten

LED Ausschalten vor Ende des Programms

```
Define msslave()=
Prgm
:Send „CONNECT DIGITAL.IN 1 BB 1
PULLUP"
:While getKey()=""
:  Send „READ DIGITAL.IN 1 „
:  Get l
:  If l=1 Then
:   Send „SET LIGHT ON"
:  Else
:   Send „SET LIGHT OFF"
:  EndIf
:EndWhile
:Send „SET LIGHT OFF"
:EndPrgm
```

9. Digitale Motorsteuerung

Grundidee

Eine digitale Motorsteuerung ist zu entwerfen, mit der zunächst die Drehrichtung eines Elektromotors gesteuert werden kann. Im Weiteren soll auch seine Drehzahl zu variieren sein.

Überblick

- Der Elektromotor soll über einen beliebigen Zeitraum eingeschaltet werden können.
- Mit den Cursortasten „up“ / „down“ soll er eingeschaltet bzw. seine Drehrichtung geändert werden. Mit der Taste „enter“ soll er ausgeschaltet werden können
- Zur Leistungsverstärkung kommt ein kommerzielles Motortreiber-Modul zum Einsatz.

Hardware

Die digitale Steuerung des Elektromotors erfolgt über eine sogenannte H-Brücke.
Abbildung 1 zeigt ihr Funktionsprinzip. Ist jeweils nur ein Schalter geschlossen, dreht sich der Motor. Die Drehrichtung ist davon abhängig, ob Schalter 1 oder Schalter 2 geschlossen wird. Sind beide Schalter geöffnet oder geschlossen, bleibt der Motor stehen. Abbildung 2 zeigt die Umsetzung dieses Funktionsprinzips in einem kommerziellen Motortreiber-Modul.

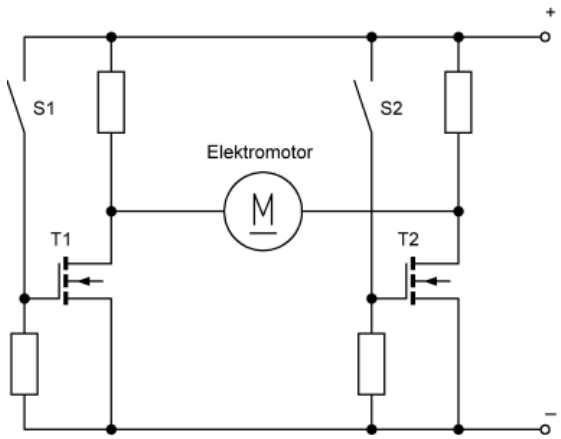


Abbildung 1

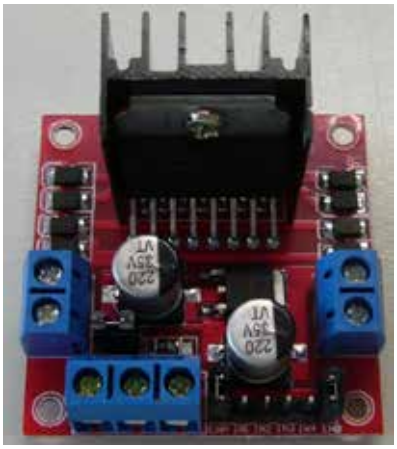


Abbildung 2

Wird nun jeweils ein Transistor mit einer Impulsfolge angesteuert, deren Tastverhältnis (Verhältnis der Ein- und Ausschaltzeit) variiert wird, kann auch die Drehzahl in jeder Drehrichtung geändert werden. Der Elektromotor selbst fungiert als Digital-Analog-Umsetzer.

Aufbau

Zunächst werden die beiden Breadboardanschlüsse BB 4 und BB 8 als digitale Ausgabekanäle genutzt. Mit dem oben beschriebenen Funktionsprinzip lässt sich damit das Modell einer Krananlage realisieren.
Da die beiden Anschlüsse BB 4 und BB 8 auch zur Pulsweitenmodulation (PWM) vorgesehen sind, soll dann die Erweiterung zur Drehzahländerung erfolgen. Damit ist dann z.B. die Steuerung einer Modelleisenbahn möglich.

Aufgabe 1:

Implementiere das Programm zur digitalen Motorsteuerung und teste es an einem „Modellkran“. Mit der Taste „enter“ wird der Motor gestoppt, der Programmabbruch erfolgt mit der Taste „esc“.

Aufgabe 2:

Erweitere das Programm mit einer Pulsweitenmodulation. Die Cursortasten „up“ und „down“ sollen dabei zur schrittweisen Drehzahländerung genutzt werden. Wird z.B. die Taste „down“ auch dann gedrückt, wenn der Motor bereits zum Stillstand gekommen ist, erfolgt ein Drehrichtungswechsel und die Drehzahl wird in der entgegengesetzten Richtung erhöht. Analog erfolgt der erneute Richtungswechsel, wenn nun die Taste „up“ über den Motorstillstand hinaus betätigt wird.
Der „Schnellstopp“ mit der Taste „enter“ soll weiterhin möglich sein. Mit der Taste „esc“ wird das Programm wieder beendet.

Lösungen und Hinweise zu den Aufgaben

Aufgabe 1:

	`Define kran()=`
	`Prgm`
Definieren lokaler Variablen.	`Local key,test`
Initialisierung der Breadboardausgänge BB 4 und BB 8.	`Send "CONNECT DIGITAL.OUT 1 TO BB 4"`
	`Send "CONNECT DIGITAL.OUT 2 TO BB 8"`
	`key:="enter": test:=true`
Programmabbruch mit „esc“.	`While key≠"esc"`
Tastaturabfrage.	`  key:=getKey()`
Falls Taste „down“ gedrückt, dann aktiviere Digitalausgang 1, deaktiviere Digitalausgang 2.	`  If key="down" and test Then`
	`   Send "SET DIGITAL.OUT 1 ON"`
	`   Send "SET DIGITAL.OUT 2 OFF"`
	`   test:=false`
	`  EndIf`
Falls Taste „enter“ gedrückt, dann deaktiviere beide Digitalausgänge.	`  If key="enter" and not test Then`
	`   Send "SET DIGITAL.OUT 1 OFF"`
	`   Send "SET DIGITAL.OUT 2 OFF"`
	`   test:=true`
	`  EndIf`
Falls Taste „up“ gedrückt, dann deaktiviere Digitalausgang 1, aktiviere Digitalausgang 2.	`  If key="up" and test Then`
	`   Send "SET DIGITAL.OUT 1 OFF"`
	`   Send "SET DIGITAL.OUT 2 ON"`
	`   test:=false`
	`  EndIf`
	`EndWhile`
Trennung der angeschlossenen Hardware.	`Send "DISCONNECT DIGITAL.OUT 1"`
	`Send „DISCONNECT DIGITAL.OUT 2"`
	`EndPrgm`

Mit der Variablen *test* wird verhindert, dass von einer in die andere Drehrichtung umgeschaltet werden kann, ohne den Motor zuvor zu stoppen. Weiterhin werden unnötige Mehrfachaktivierungen bzw. Deaktivierungen vermieden.

Aufgabe 2:

Definieren lokaler Variablen.
Initialisierung der Breadboardausgänge BB 4 und BB 8.
Variable zum „Schnellstopp“.
Variablen zur Festlegung der Fahrtrichtung.
Variablen zur Drehzahl bzw. Drehzahländerung.

Falls Taste „down“ gedrückt, dann reduziere den pwm-Wert um dpwm und begrenze ihn auf -255.

Falls der pwm-Wert kleiner Null ist, dann organisiere den Fahrtrichtungswechsel.

Ausgabe des pwm-Wertes.

Falls Taste „enter“ gedrückt, dann lege die Variablenwerte für einen „Schnellstopp“ fest.

Falls Taste „up“ gedrückt, dann erhöhe den pwm-Wert um dpwm und begrenze ihn auf 255.

Falls der pwm-Wert größer Null ist, dann organisiere den erneuten Fahrtrichtungswechsel.

Programmunterbrechung.
Ausgabe des modifizierten pwm-Wertes an beiden analogen Breadboardkanälen.

Trennung der angeschlossenen Hardware.

```
Define eisenbahn()
Prgm
Local key,vor,rueck,pwm,dpwm
Send „CONNECT ANALOG.OUT 1 TO BB 4“
Send „CONNECT ANALOG.OUT 2 TO BB 8“
key:=“enter“
vor:=0: rueck:=0
pwm:=0: dpwm:=10
DispAt 1,“-- Motorsteuerung --“
DispAt 2,“Steuertasten: up / down“
DispAt 3,“Stopp: enter / Abbruch:
esc“
While key≠“esc“
 key:=getKey()
 If key=“down“ Then
  pwm:=pwm-dpwm
  If pwm<-255 Then
   pwm:=-255
  EndIf
  If pwm<0 Then
   vor:=0: rueck:=1
  EndIf
  DispAt 4,“Geschwindigkeit: „,pwm
 EndIf
 If key=“enter“ Then
  pwm:=0: vor:=0: rueck:=0
  DispAt 4,“Geschwindigkeit: „,pwm
 EndIf
 If key=“up“ Then
  pwm:=pwm+dpwm
  If pwm>255 Then
   pwm:=255
  EndIf
  If pwm>0 Then
   vor:=1: rueck:=0
  EndIf
  DispAt 4,“Geschwindigkeit: „,pwm
 EndIf
 Wait 0.05
 Send „SET ANALOG.OUT 1
eval(abs(vor*pwm))“
 Send „SET ANALOG.OUT 2
eval(abs(rueck*pwm))“
EndWhile
Send „DISCONNECT ANALOG.OUT 1“
Send „DISCONNECT ANALOG.OUT 2“
EndPrgm
```

Versuche mit dem TI-Innovator™ Rover:

10. Linienfolger

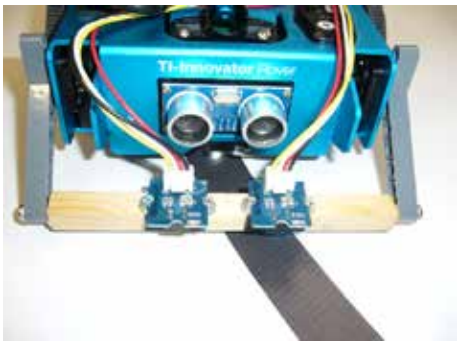
Von vergleichbaren mobilen Robotern kennen wir die klassische Anwendung als Linienfolger.
Dabei soll der Rover in möglichst kurzer Zeit einer vorgezeichneten dunklen Linie mit beliebigen Krümmungsradien fehlerfrei folgen.
Wir wollen Programmstrukturen entwickeln, mit denen dieses Konzept auch mit dem TI Innovator™ Rover umsetzbar ist.

Externe Sensoren:

- 2 Arduino-Grove Linienfinder, LTH-1550-01 (https://www.reichelt.de)
- Anschluss der Linienfinder-Sensoren an die Digitaleingänge IN 1 / IN 2.

  Diese Sensoren müssen im Abstand der Linienbreite positioniert werden.

  Der Anbau an den Rover kann mit geeigneten Lego-Bauteilen erfolgen.
- Die Funktionalität des integrierten Ultraschallsensors soll dabei erhalten bleiben.



Vorüberlegungen:

- Der Rover soll sich kontinuierlich durch den Parcours bewegen. Deshalb wird für ihn der Pulsweitenmodulations-Antrieb (PWM) gewählt.
- Wegen der spiegelsymmetrischen Anordnung der Antriebsmotoren müssen bei geradliniger Bewegung des Rovers die Vorzeichen der PWM-Werte für das linke bzw. rechte Antriebsrad verschieden sein.
- Lenkbewegungen werden dadurch realisiert, indem man das PWM-Signal des kurvenäußeren Rades um den Betrag Dpwm erhöht, während gleichzeitig das Signal des kurveninneren Rades um diesen Betrag reduziert wird.
- Die Linienfinder-Sensoren generieren bei hellem Untergrund den logischen Pegel 0 und bei dunklem Untergrund den logischen Pegel 1.

Aufgabe 1:

Implementiere einen einfachen Algorithmus eines Linienfolgers.
Teste ihn auf einem geeigneten Parcours und optimiere dabei seine Bewegungsparameter.
Experimentiere dazu mit geeigneten Werten für das PWM-Signal und seine Änderung ∆pwm bei Kurvenfahrten.

Lösung:

Deklarationen lokaler Vaiablen:
pwm ... Geschwindigkeitswert
dpwm ... Änderung der Geschwindigkeitswerte bei Kurvenfahrten,
cl, cr ... Messwerte der Sensoren,
Festlegung er Bewegungsparameter.

Programmstruktur zum Start des Rovers mit der Taste „up“.
Mit der Taste „esc“ kann das Programm verlassen werden.

Bis zum Programmabbruch mit „esc“ werden die digitalen Pegel der beiden Linienfinder-Sensoren erfasst, daraus die entsprechenden PWM-Werte bestimmt und an den Rover gesendet. Zunächst werden die Variablen cl und cr mit den Messwerten der Sensoren belegt.
Falls der linke Sensor die dunkle Linie erfasst, soll sich der Rover nach links bewegen.

Anderenfalls bewegt sich der Rover nach rechts oder geradlinig weiter.

```
Define linienfolger1()=
Prgm
:Local pwm,dpwm,key,cl,cr
:Send "CONNECT RV"
:Send "CONNECT DIGITAL.IN 1 TO IN 1"
:Send "CONNECT DIGITAL.IN 2 TO IN 2"
:pwm:=110: dpwm:=55
:DispAt 1,"Start mit <up> /
            Abbruch mit <esc>"
:key:="enter"
:While key≠"up" and key≠"esc"
: key:=getKey()
: If key="up" Then
:  Send „RV.MOTORS
          LEFT eval(-1*pwm)
          RIGHT eval(pwm)"
: EndIf
:EndWhile
:While key≠"esc"
: key:=getKey()
: cl:=0: cr:=0
: Send „READ DIGITAL.IN 1"
: Get cl
: Send „READ DIGITAL.IN 2"
: Get cr
: If cl=1 Then
:  Send „RV.MOTORS
          LEFT eval(-1*pwm+cl*dpwm)
          RIGHT eval(pwm+cl*dpwm)"
: Else
:  Send „RV.MOTORS
          LEFT eval(-1*pwm-cr*dpwm)
          RIGHT eval(pwm-cr*dpwm)"
: EndIf
:EndWhile
:Send „DISCONNECT DIGITAL.IN 1"
:Send „DISCONNECT DIGITAL.IN 2"
:Send „DISCONNECT RV"
:EndPrgm
```

Hinweis:

Eine separate Detektierung einer geradlinigen Bewegung ist nicht notwendig, da in diesem Fall die Sensor-Variable cr den Wert 0 annimmt.

Aufgabe 2:

Beim Test fällt auf, dass bei Kurven sehr kleiner Krümmungsradien der Rover den Parcours vorzeitig verlässt, da entweder die Geschwindigkeit zu hoch oder die Lenkbewegung zu gering ist. Optimiere den Algorithmus derart, dass der Rover seine Lenkbewegung dem Krümmungsradius der Kurve anpasst.

Lösung:

Deklaration weiterer lokaler Variablen:
kl, kr ... Vervielfachung der Geschwindigkeitsänderung Dpwm,
dk ... Zunahme der Faktoren kl, kr,
Festlegung geeigneter Startwerte.

```
Define linienfolger2()=
Prgm
:Local pwm,dpwm
:Local key,cl,cr,kl,kr,dk
:Send "CONNECT RV"
:Send "CONNECT DIGITAL.IN 1 TO IN 1"
:Send "CONNECT DIGITAL.IN 2 TO IN 2"
:pwm:=120: dpwm:=40
:kl:=1: kr:=1: dk:=0.12
:DispAt 1,"Start mit <up> /
            Abbruch mit <esc>"
:key:="enter"
:While key≠"up" and key≠"esc"
: key:=getKey()
: If key="up" Then
:  Send "RV.MOTORS
          LEFT eval(-1*pwm)
          RIGHT eval(pwm)"
: EndIf
:EndWhile
:While key≠"esc"
: key:=getKey()
: cl:=0: cr:=0
: Send "READ DIGITAL.IN 1"
: Get cl
: Send "READ DIGITAL.IN 2"
: Get cr
```

Falls der Pegel des linken Sensors 1 ist, werden die PWM-Werte für eine Linkskurve modifiziert und der Vervielfachungswert kl schrittweise erhöht.
Anderenfalls wird dieser Vervielfachungsfaktor wieder auf den Wert 1 zurückgesetzt.

Analog dazu wird der Pegel 1 des rechten Sensors verarbeitet.

Zusätzlich muss hier der Fall untersucht werden, bei dem keiner der Sensoren den Pegel 1 liefert, um wieder die Geradeausfahrt festzulegen.

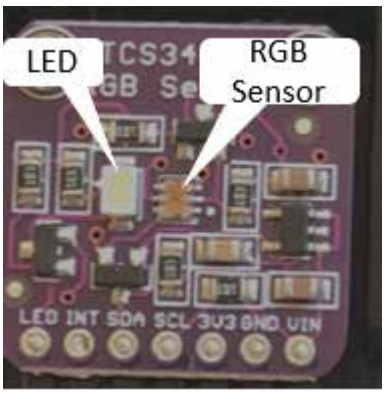
```
: If cl=1 Then
:  Send "RV.MOTORS
       LEFT eval(-1*pwm+kl*dpwm)
       RIGHT eval(pwm+kl*dpwm)"
:  kl:=kl+dk
: Else
:  kl:=1
: EndIf
: If cr=1 Then
:  Send "RV.MOTORS
       LEFT eval(-1*pwm-kr*dpwm)
       RIGHT eval(pwm-kr*dpwm)"
:  kr:=kr+dk
: Else
:  kr:=1
: EndIf
: If cl=0 and cr=0 Then
:  Send „RV.MOTORS
       LEFT eval(-1*pwm)
       RIGHT eval(pwm)"
: EndIf
:EndWhile
:Send „DISCONNECT DIGITAL.IN 1"
:Send „DISCONNECT DIGITAL.IN 2"
:Send „DISCONNECT RV"
:EndPrgm
```

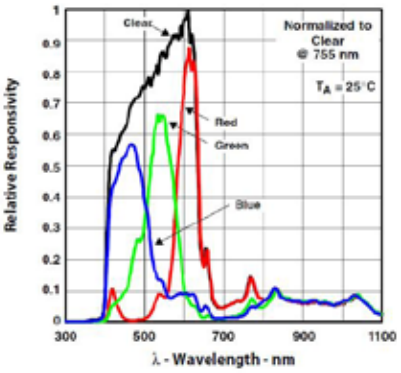
Hinweise:

Eine weitere Optimierung wäre durch die Anpassung der Geschwindigkeit des Rovers (PWM-Signal) an den Krümmungsradius der Kurve denkbar. Strukturell ist die Umsetzung in einem vergleichbaren Algorithmus möglich. Natürlich lassen sich auch beide Ideen miteinander kombinieren. Die kürzeste Laufzeit durch den Parcours könnte dann über die günstigste Problemlösung entscheiden. Mit dem im Rover integrierten Ultraschallsensor können natürlich weitere Funktionalitäten ergänzt werden, wie. z.B. das Erfassen von Hindernissen.

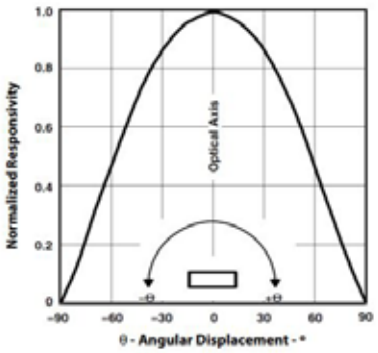
11. Rasenmäher und Farberkennung

Auf der Unterseite des Rovers vor der Vorderachse befindet sich ein RGB Sensor zur Erkennung der Farbe des Bodens, auf dem der Rover fährt. Die wesentlichen Komponenten der RGB Sensor-Platine sind eine weiße LED als neutrale Lichtquelle und ein RGB Sensor vom Typ TCS34725 der Firma AMS. Dieser Sensor besteht aus vier Photodioden mit spektralen Filtern für die Farben Rot, Grün und Blau. Die vierte Photodiode zum Messen der Helligkeit wird im Rover nicht verwendet. Das von der Bodenoberfläche reflektierte Licht der weißen LED wird mit den drei Photodioden gemessen und ist als Zahlenwert über den Befehl <Send(„READ RV.COLORINPUT.x “)> abrufbar. Man kann auch den „Grauwert“ des Bodens über den Befehl <Send(„READ RV.COLORINPUT.GRAY“)> abrufen, jedoch ist dies kein direkter Messwert, sondern eine Berechnung aus den RGB Werten nach der Formel: Grau= 0.3*Rot + 0.59*Grün + 0.11*Blau.
Aufgrund der hohen spektralen Empfindlichkeit des Sensors für die Farben Rot, Grün und Blau wird empfohlen, Übungen mit diesen drei Grundfarben zu erstellen, statt eine schwarze oder weiße Linie zu verwenden.





Der Abstand des Sensors vom Boden beträgt knapp 10mm. Die Empfindlichkeit des Sensors ist auch im Bereich von +/- 30 Grad ausserhalb seiner optischen Achse grösser als 80%. Daher gibt es eine minimale Linienbreite, die der Sensor noch einwandfrei erkennt (siehe Übungen).



Aufgabe 1:

Erstelle ein Programm, das fortlaufend die Rot- ,Grün- ,Blau- und Grauwerte des RGB Sensors ausliest, auf dem CX Display anzeigt und die RGB LED des Rovers entsprechend ansteuert. Das Programm soll durch Drücken einer beliebigen Taste auf dem CX Rechner gestoppt werden können.

Lösung:

Den Rover verbinden.
Endlos-Schleife, bis Taste gedrückt wird.
Rot-Wert lesen.
Rot-Wert anzeigen.

Grün-Wert lesen.
Grün-Wert anzeigen.

Blau-Wert lesen.
Blau-Wert anzeigen.

Grau-Wert lesen.
Grau-Wert anzeigen.

RGB-Werte an RGB LED des Rovers senden.

Vor Ende des Programms RGB LED ausschalten.

```
Define readcolors()=
Prgm
:Send "CONNECT RV"
:While getKey()=""
:  Send "READ RV.COLORINPUT.RED"
:  Get xred
:  DispAt 1,"Rot Wert: ",xred
:  Send "READ RV.COLORINPUT.GREEN"
:  Get xgreen
:  DispAt 2,"Grün Wert: ",xgreen
:  Send "READ RV.COLORINPUT.BLUE"
:  Get xblue
:  DispAt 3,"Blau Wert: ",xblue
:  Send "READ RV.COLORINPUT.GRAY"
:  Get xgrey
:  DispAt 4,"Grau Wert: ",xgrey
:  Send "SET RV.COLOR eval(xred)
eval(xgreen) eval(xblue)"
:EndWhile
:Send "SET RV.COLOR 0 0 0"
:EndPrgm
```

Übungen:

(1) Starte das Programm und stelle den Rover auf ein weisses Blatt Papier. Notiere die Werte. Male ca 5x5cm grosse Quadrate mit rotem, grünen, blauem und schwarzem Filzstift auf das Papier. Male die Quadrate mit den entsprechenden Filzstiften komplett aus. Platziere den Rover mit dem RGB Sensor jeweils über jedes der vier Quadrate und notiere die Werte. Beende Dein Programm durch Drücken einer Taste auf dem CX Rechner.
Welche Erkenntnisse hast Du aus dieser Übung gewonnen?

(2) Nimm ein weiteres Blatt Papier und male schwarze, gerade Linien in verschiedenen Breiten, jeweils in einem Abstand von mindestens 6cm voneinander. Bewege den Rover über die Linien, senkrecht zur Linienrichtung. Welche ist die kleinste Linienbreite, die der Rover noch erkennt?

(3) Wiederhole die Übung (2), aber mit einem roten, grünen oder blauen Filzstift.

(4) Halte den Sensor des Rovers direkt an einen Finger Deiner Hand. Welcher Farbwert R/G/B ist am höchsten?

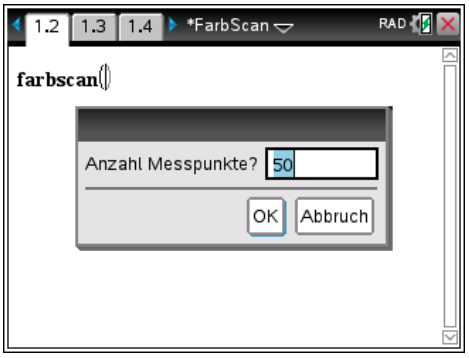
(5) Halte den Sensor des Rovers gegen ein Kleidungsstück. Kann er die Farbe erkennen?

Aufgabe 2:

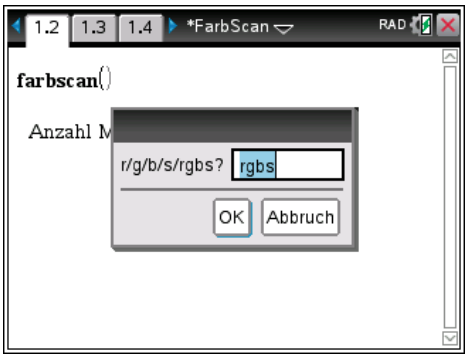
Erstelle ein Programm, das fortlaufend die Rot- ,Grün- ,Blau- und Grauwerte des RGB Sensors ausliest, während der Rover fährt. Zu Beginn soll die Anzahl der Messpunkte per Eingabefenster abgefragt werden sowie die Farbkanäle (R/G/B/Grau), die gemessen werden sollen. Die Messdaten sollen in einer Tabelle gespeichert und anschliessend in Grafik-Fenstern auf dem Rechner angezeigt werden.

Lösung:

Das Eingabefenster für die Anzahl der Messpunkte.



Das Eingabefenster für die Auswahl der zu messenden Farbkanäle



```
                                Define farbscan()=
                                Prgm
Den Rover verbinden.            :Send "CONNECT RV"
Eventuell vorhandene Variablen löschen.  :DelVar rot,gruen,blau,grau,x
Anzahl der Messpunkte abfragen. :Request „Anzahl Messpunkte?",n
Die Listen mit Nullen füllen:   :For i,1,n
Liste für roten Farbsensor      :  rot[i]:=0
Liste für grünen Farbsensor     :  gruen[i]:=0
Liste für blauen Farbsensor     :  blau[i]:=0
Liste für die Grau-Werte        :  grau[i]:=0
Messpunkte-Zähler               :  x[i]:=i
                                :EndFor
Auswahl der Farbsensoren        :RequestStr „r/g/b/s/rgbs?",farb
Rover Messfahrt starten, auf 4m begrenzen.  :Send „RV FORWARD  DISTANCE 40  SPEED
                                0.20  M/S „
Mess-Schleife                   :For i,1,n
Wenn Rot selektiert, Rot abfragen  : If inString(farb,"r")>0 Then
                                :   Send „READ RV.COLORINPUT.RED"
                                :   Get xred
Wert in der Liste abspeichern   :   rot[i]:=xred
                                : EndIf
Wenn Grün selektiert, Grün abfragen  : If inString(farb,"g")>0 Then
                                :   Send „READ RV.COLORINPUT.GREEN"
                                :   Get xgreen
Wert in der Liste abspeichern   :   gruen[i]:=xgreen
                                : EndIf
Wenn Blau selektiert, Blau abfragen  : If inString(farb,"b")>0 Then
                                :    Send „READ RV.COLORINPUT.BLUE"
                                :       Get xblau
                                :       blau[i]:=xblau
Wert in der Liste abspeichern   :  EndIf
                                :  If inString(farb,"s")>0 Then
Wenn Grau selektiert, Grau abfragen  :     Send „READ RV.COLORINPUT.GRAY"
                                :        Get xgray
                                :        grau[i]:=xgray
Wert in der Liste abspeichern   :         EndIf
                                :EndFor
Alle Messwerte aufgenommen?     :Send „READ RV.WAYPOINT.XYTHDRN"
Zeitdauer der Messfahrt ermitteln  :Get t
                                :Send „RV STOP „
Messfahrt stoppen               :Disp „Dauer: „,t[3],"Sekunden"
Messdauer auf dem Rechnerbildschirm anzeigen  :EndPrgm
```

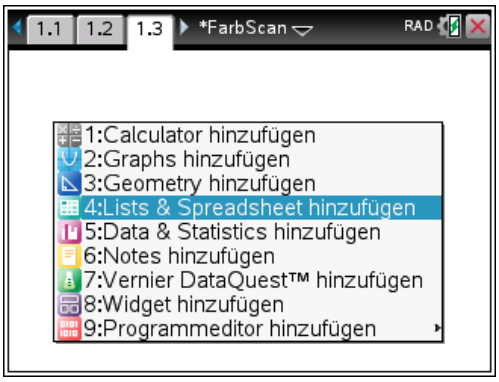
Anmerkung:

Die Dauer (und damit die Länge) der Messfahrt hängt von der Anzahl der Messpunkte, der Geschwindigkeit des Rovers und der Anzahl der selektierten Farbkanäle ab. Die maximale Länge der Messfahrt sollte den örtlichen Gegebenheiten (z.B. Länge des Tisches) angepasst werden und im < Send „RV FORWARD DISTANCE 40 SPEED 0.20 M/S „> Befehl entsprechend programmiert werden. Der Rover hält dann automatisch an, auch wenn die gewünschte Anzahl der Messpunkte noch nicht erreicht ist.

Übungen:

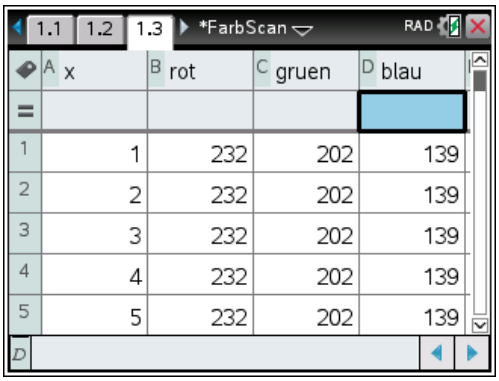
(1) Starte das Programm und wähle zu Beginn eine kleine Anzahl der Messpunkte, z.B. 10. Selektiere nur einen Farbkanal (z.B. Rot). Notiere die Zeit, die der Rover zum Messen von 10 Messwerten gebraucht hat.

Füge ein Listen-Fenster zu Deinem Programm hinzu.

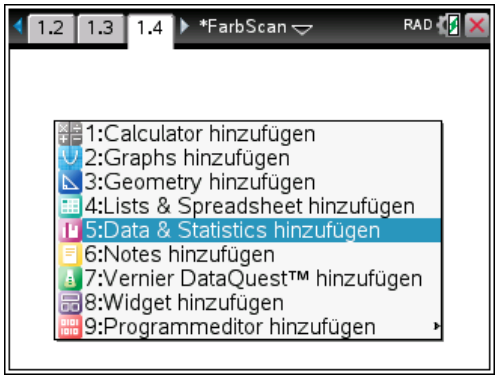


Verknüpfe die Spalte A Deiner Liste mit der Variablen x (die Messwerte-Zählvariable).

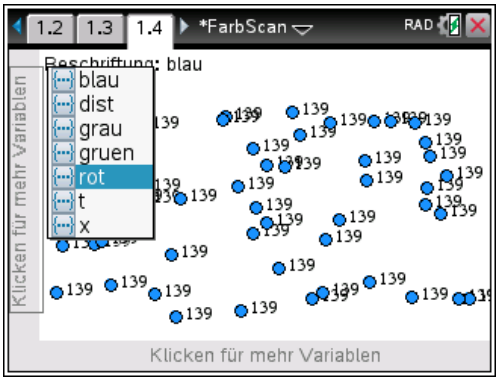
Ordne die Variablen der Farbkanäle den Spalten B,C,D,E zu.



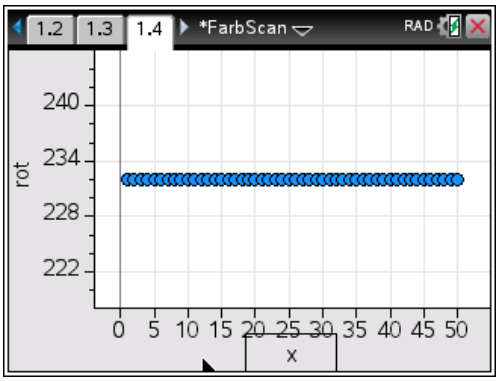
Addiere ein Daten & Statistik-Fenster, um die Datenpunkte aller Farbkanäle anzuzeigen.



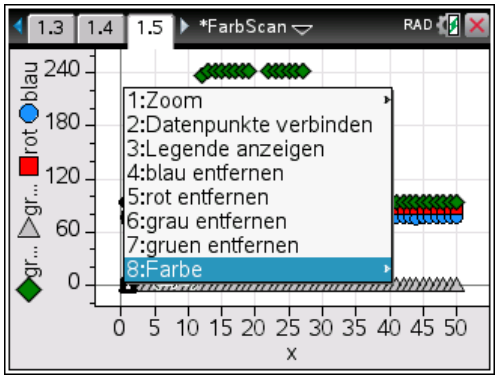
Ordne die Variable des Farbkanals der Y-Achse zu.



Ordne die Messwerte-Zählervariable der X-Achse zu.



Wiederhole diesen Vorgang für die übrigen Farbkanäle (R/G/B/Grau) und ordne den Datenpunkten die korrekte Farbe im Grafik-Fenster zu.



Schau Dir die Messdaten im Grafik-Fenster an. Erkläre die Messwerte anhand der Farbbeschaffenheit des Bodens, den Du für die Messfahrt des Rovers benutzt hast.

(2) Führe eine Messfahrt für jeden einzelnen Farbkanal mit 10,20,50 und 100 Messpunkten durch. Notiere jeweils die für die Messfahrt benötigte Zeit. Rechne die Anzahl der Messungen pro Sekunde aus. Rechne die Anzahl der Messungen pro cm Messfahrt aus. Ändere die Geschwindigkeit der Messfahrt im Befehl < Send „RV FORWARD DISTANCE 40 SPEED 0.20 M/S „> . Der zulässige Geschwindigkeitsbereich liegt zwischen 0.14 und 0.23 M/S. Wie viele Messungen der Farbe Schwarz kann der Rover pro cm bei langsamster Messfahrt machen?

(3) Zeichne verschieden breite, parallele Linien auf ein grosses Papier, quer zur Fahrtrichtung des Rovers. Alternativ kannst Du auch farbiges Klebeband (Gaffa-Tape) nehmen. Führe ein Messfahrt mit minimaler und maximaler Geschwindigkeit durch. Achte darauf, dass Du die maximale Länge der Messfahrt den Gegebenheiten anpasst (z.B. Tischlänge). Wiederhole den Versuch 5-mal und schaue Dir jeweils das Resultat im Grafik-Fenster an. Welches ist die kleinste Linienbreite, die der Rover wiederholbar sicher erkennt?

Aufgabe 3

TI-Innovator-Rover als Rasenmäher-Roboter

Moderne Rasenmäher-Roboter fahren über eine Rasenfläche, bis sie eine Begrenzung der Fläche erkennen. Dies können Koordinaten sein, die mit einem GPS Signal verglichen werden oder eine in den Erdboden verlegte Metallschleife, die über einen Sensor im Roboter erkannt wird, wenn sie überfahren wird. Der Rasenmäher-Roboter wendet und setzt seine Mähaufgabe fort, indem er in einem neuen Winkel über den Rasen fährt, bis er die Grenze der Fläche an einem anderen Punkt des Gartens erneut erreicht.

Erstelle ein Programm, bei dem der TI-Rover sich wie ein Rasenmäher-Roboter verhält und eine durch eine farbige Linie begrenzte beliebige Fläche im Laufe der Zeit komplett durchfährt.

Hinweise:

Es empfiehlt sich, als Farbe für die Begrenzungslinie eine der drei Grundfarben des RGB Sensors (rot, grün oder blau) zu verwenden, da sie am sichersten vom Sensor erkannt werden.
Die Linie sollte eine gewisse Mindestbreite haben, die vom fahrenden Rover sicher erkannt wird. Gegebenenfalls kann man die Mindestbreite mit dem RGB-Sensorprogramm aus Aufgabe 2 experimentell ermitteln.
Wenn der RGB-Sensor des Rovers die Begrenzungslinie erkannt hat, dauert es noch eine gewisse Zeit, bis das “RV STOP“ Kommando ausgeführt wurde und der Rover zum Stillstand gekommen ist. Es kann sein, daß der Rover zu diesem Zeitpunkt die Linie bereits komplett überfahren hat. Daher muß der Rover als Erstes ein Stück rückwärts fahren, bevor er wendet und seine Fahrt über die Rasenfläche fortsetzt. Die Länge bzw Zeit des Rückwärtsfahrens ist ebenfalls experimentell zu ermitteln.

Lösung:

Hilfsvariable für Programmabbruch
Mähen, solange kein Programmabbruch
Vorwärts fahren

… bis rote Linie gefunden wird.

Auch in dieser Schleife könnte Programmabbruch passieren.

Linie gefunden oder Programmabbruch
Rover anhalten

Wenn kein Programmabbruch,
Rover rückwärts fahren

… und um 45 Grad nach links drehen.

Immer noch auf der roten Linie?

Dann stattdessen um 45 Grad nach rechts drehen.

Auch beim Wenden könnte ein Programmabbruch gewünscht sein…

Bei Abbruch Rover anhalten und Programm beenden.

```
Define mowlab3()=
Prgm
:Send „CONNECT RV"
:breakkey:=0
:While breakkey=0
: Send „RV FORWARD 90"
: Send „READ RV.COLORINPUT.RED"
: Get xred
: While xred<180 and breakkey=0
:  Send „READ RV.COLORINPUT.RED"
:  Get xred
:  If getKey()≠"" Then
:   breakkey:=1
:  EndIf
: EndWhile
: Send „RV STOP „
: Wait 1
: If breakkey=0 Then
:  Send „RV BACKWARD TIME 1"
:  Wait 1
:  Send „RV LEFT 45"
:  Wait 1
:  Send „READ RV.COLORINPUT.RED"
:  Get xred
:  If xred>180 Then
:   Send „RV RIGHT 90"
:   Wait 1
:  EndIf
:  If getKey()≠"" Then
:   breakkey:=1
:  EndIf
: EndIf
:EndWhile
:Send „RV STOP „
:EndPrgm
```

Übungen:

(1) Filme den Rasenmäher-Roboter mit einem Smartphone im Zeitraffer-Modus über eine längere Zeit (länger als eine Minute) und schau Dir das Video an. Ist der Rasenmäher wirklich über die gesamte Wiese gefahren? Hat er Teile der Fläche ausgelassen? Ist er besonders lang oder häufig über eine Stelle gefahren? Wie könnte man das Programm ändern oder erweitern, damit er gleichmässiger und vollständiger über die gesamte Fläche fährt?
(2) Verändere das Programm, daß der Rover rein zufällig die Wenderichtung nach Erkennen der Begrenzungslinie wählt. Tipp: Verwende die randInt() Funktion, um die Drehrichtung nach dem Zufallsprinzip zu wählen.
(3) Versuche die Mähgeschwindigkeit zu optimieren. Bis zu welcher Geschwindigkeit funktioniert der Mähroboter noch zuverlässig? Minimiere die diversen Wait-Zeiten im Programm. Wie ändert sich die Zuverlässigkeit?

12. Punkte anfahren

In dieser Aufgabe soll der Rover nicht über seinen Ultraschallsensor oder den RGB-Sensor gesteuert werden, sondern über sein virtuelles Koordinatensystem. Es gelten die gleichen Regeln wie für die in den anderen Aufgaben verwendeten Rover Fahrbefehle: Alle Fahrbefehle werden in einem lokalen Pufferspeicher im Rover abgelegt und dort der Reihe nach abgearbeitet, unabhängig vom Hauptprogramm im Rechner. Die Abarbeitung kann über ein “RV STOP” Kommando unterbrochen werden. Neue Fahrbefehle nach einem Stop-Kommando werden wieder in den Pufferspeicher gelegt und abgearbeitet.
Das virtuelle Koordinatensystem basiert auf einem 10cm Rastermaß, es kann aber ein beliebiges anderes Rastermaß eingestellt werden.

Beachte:

XY Koordinaten werden auf dem kürzesten Weg, also per “Luftlinie“ vom Rover angefahren. Wird eine orthogonale Fahrstrecke gewünscht, muss man die Fahrstrecke in zwei GOTOXY Befehle aufteilen, zuerst die X-Richtung bei gleicher Y-Koordinate und dann die Y-Richtung, bei gleicher X-Koordinate.

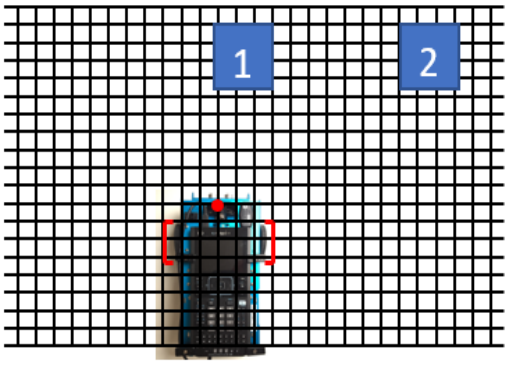
Hinweis:

Zur besseren Orientierung bei der Programmierung empfiehlt es sich, Karopapier als Skizzenvorlage zu nehmen. Die Unterlage des Rovers beim Programmablauf kann beliebig gewählt werden, da der Rover die Radumdrehungen (bzw zurückgelegte Wegstrecke) seiner Antriebsräder als Referenz verwendet. Die Startposition und Ausrichtung des Rovers sollte jedoch markiert werden und beim Start des Programms immer dieselbe sein, da diese Stelle den Ursprung des Koordinatensystems des Rovers bildet.

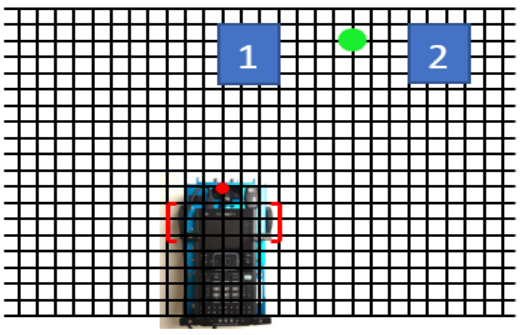
Aufgabe:

Erstelle ein Programm, bei dem der Rover rückwärts in die mit einem grünen Punkt markierte Parklücke fährt. Gehe dabei folgendermaßen vor:

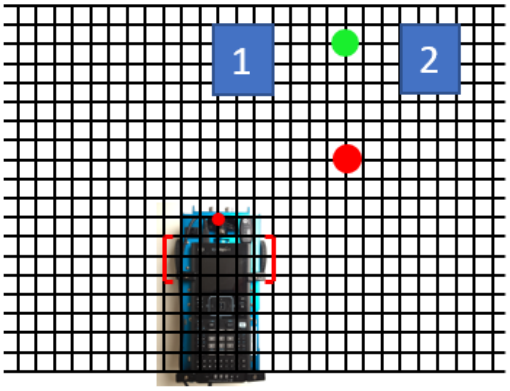
Stelle zwei Hindernisse auf den Tisch und markiere Ausgangspunkt und Ausrichtung des Rovers.



Ermittle die Koordinaten der Parklücke (grüner Punkt).



Ermittle einen Punkt vor der Parklücke, wo der Rover sich ausrichten kann, ohne die Hindernisse zu berühren (roter Punkt).



Lösung:

Rover mit dem Rechner verbinden.
Ggf neuen Massstab einstellen – hier 2.5cm.
Rover fährt zum Ausrichtpunkt
… und wendet vor der Parklücke.
Rückwärts in die Lücke. Die Zahl ist die Differenz in Kästchen zwischen Ausrichtpunkt und Parklücke.

```
Define roverxy5b()=
Prgm
:Send „CONNECT RV"
:Send „SET RV.GRID.M/UNIT 0.025"
:Send „RV TO XY 3 -7"
:Send „RV TO ANGLE 180"
:Send „RV BACKWARD 6"
:EndPrgm
```

13. Fahrt zum nächsten Gegenstand

Im Umfeld des Rover sind einige Objekte platziert. Die Aufgabe des Rover besteht nun darin, das Objekt zu finden, das ihm am nächsten steht, auf dieses zuzufahren und vor dem Objekt stehen zu bleiben.

Mögliche Umsetzung:

Der Rover dreht sich einmal um seine eigene Achse, also um 360°, und misst in vorgegebenen Intervallen jeweils den Abstand. Die jeweiligen Winkelgrößen und die zugehörigen gemessenen Abstände werden in einer Liste fortlaufend gespeichert.
In der Liste der Abstände wird dann das Minimum gesucht und über den Index der zugehörige Winkel ermittelt. Auf die entsprechenden Listenwerte greift man über den Index *liste[index]* zu. Um das Minimum der Liste zu finden, deklariert man eine neue Variable *minimum* und weist ihr den ersten Listenwert zu.
Anschließend wird jeder folgende Listenwert mit der Variable *minimum* verglichen. Findet man einen Wert, der kleiner als der Wert der Variable *minimum* ist, wird der neue Wert gespeichert. Der entsprechende Index des Abstandes entspricht dem Index des Winkels aus der Liste der Winkelgrößen. Der Rover dreht sich um diesen Winkel, fährt auf das Objekt zu und stoppt nach einer gewissen Fahrstrecke.

Bemerkungen:

- Die Bodenbeschaffenheit beeinflusst stark die Bewegung des Rovers. Im Beispielprogramm verwenden wir zur Berechnung der Länge der Anfahrstrecke den Korrekturfaktor 10. Dieser Wert muss u. U. angepasst werden.
- Die Drehung des Rovers erfolgt in Schritten von 20°. Zur Verfeinerung kann natürlich auch ein kleinerer Winkelwert verwendet werden.

Mögliche Lösung:

```
Define find_item()=
Prgm
:Local abstand
:Local winkel
:Local minimum
:Local index
:
:abstand:={}
:winkel:={}
:
:Send „CONNECT RV“
:For i,1,18
:   w:=(i-1)*20
:   Wait 2
:   Send „READ RV.RANGER“
:   Get d
:   winkel:=augment(winkel,{w})
:   abstand:=augment(abstand,{d})
:   Disp w,d
:   Send „RV LEFT 20“
:EndFor
:Wait 3
:
:minimum:=abstand[1]
:index:=1
: For j,1,18
:   If abstand[j]<minimum Then
:            minimum:=abstand[j]
:            index:=j
:   EndIf
:EndFor
:
:ww:=winkel[index]
:aa:=abstand[index]
:Disp ww,aa
:Send „RV LEFT eval(winkel[index])“
:Send „RV FORWARD
eval(10*abstand[index])“
:
:EndPrgm
```

Festlegung lokaler Variablen

Erzeugung der anfangs leeren Listen

Drehung des Rovers in Schritten von 20°

Auffüllung der Listen

Darstellung der Messwerte

Bestimmung des minimalen Abstandes

Darstellung von Winkel und minimalem Abstand

Drehung und Vorwärtsbewegung des Rovers

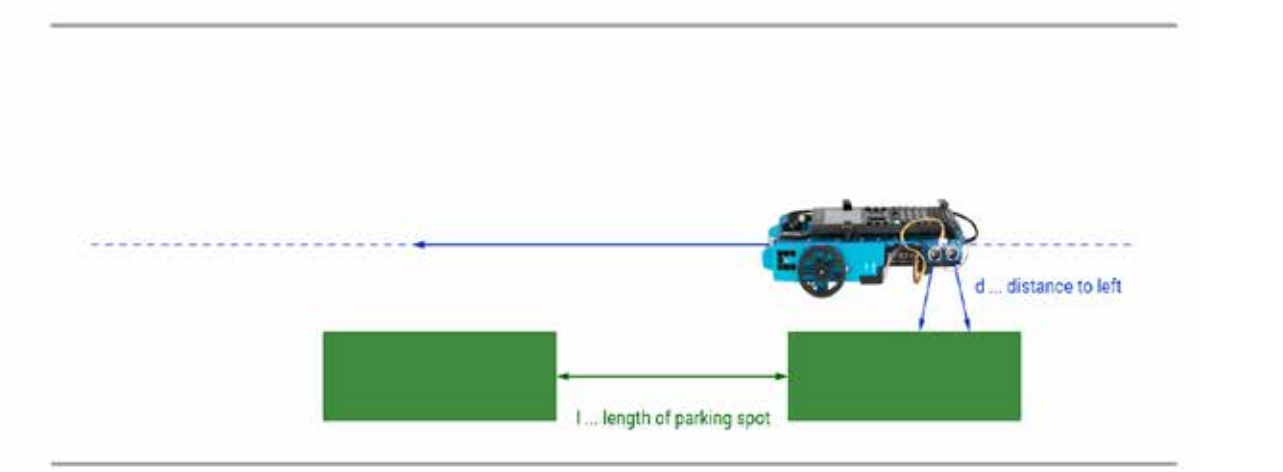
Der Korrekturfaktor 10 muss u.U. den Bodenverhältnissen angepasst werden.

14. Parallel einparken

Der Rover soll so programmiert werden, dass er selbstständig eine passende Parklücke findet und in diese rückwärts einparkt.
Dazu wird am Rover seitlich ein weiterer Ultraschallsensor angebracht (z. B. mit doppelseitigem Klebeband).
Dieser Ultraschallsensor wird mit dem Hub über den Eingang *IN 1* verbunden.

Aufgabe 1 (vorbereitende Aufgabe):

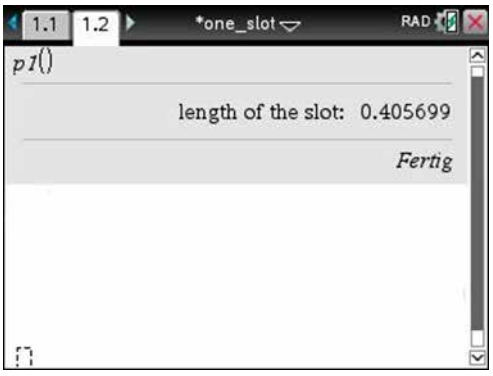
Der Rover soll an zwei parkenden Objekten vorbeifahren, die Länge der jeweiligen Parklücke messen und ausgeben (Programm **p1**).



Der Rover beginnt seine Fahrt vor einem parkenden Objekt. Der Abstand zum Objekt wird fortlaufend gemessen. Ändert sich der gemessene Abstand stark, fährt der Rover solange an einer Parklücke vorbei, bis sich der Abstand wieder stark ändert. Wir haben als Grenze für die Änderungen 30 cm verwendet.
Zur Bestimmung der Länge der Parklücke verwenden wir den Befehl

RV.WAYPOINT.DISTANCE .

Dieser gibt die Entfernung zwischen einem Startpunkt und dem jeweils aktuellen Wegpunkt als kumulierte Länge zurück.



Lösung:

d: Entfernung zum linken Parkobjekt
l : Länge der Parklücke

vorbeifahren, solange keine Parklücke da ist

anhalten und setzen der Anfangsmarke

an der Parklücke vorbeifahren

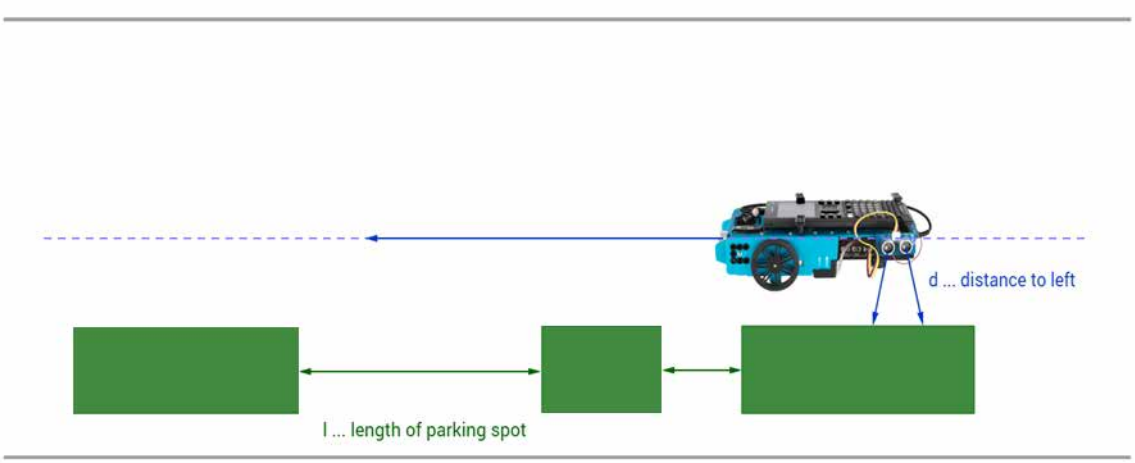
Länge der Parklücke bestimmen und anzeigen

anhalten und neue Anfangsmarke setzen

```
Define p1()=
Prgm
:Local d,l
:d:=0: l:=0
:
:Send „CONNECT RV"
:Send „CONNECT RANGER 1 TO IN 1"
:Wait 0.1
:
:While d<0.3
:      Send „RV FORWARD"
:      Send „READ RANGER 1"
:      Get d
:EndWhile
:Send „RV STOP CLEAR"
:
:While d≥0.3
:          Send „RV FORWARD"
:          Send „READ RANGER 1"
:          Get d
:EndWhile
:Send „READ RV.WAYPOINT.DISTANCE"
:Get l
:Disp „length of the slot:  „,l
:Send „RV STOP CLEAR"
:
:EndPrgm
```

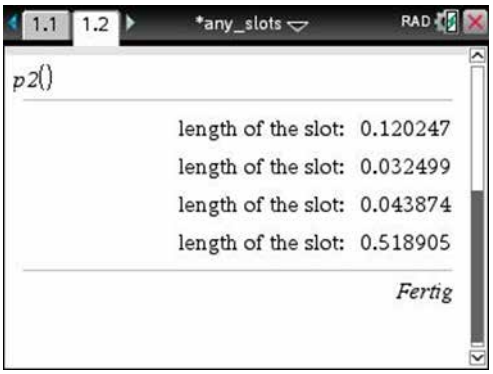
Aufgabe 2:

Nun sollen die Längen mehrerer Parklücken gemessen und ausgegeben werden (Programm **p2**). Ist die gemessene Länge kleiner als 50 cm, nutzen wir für die neuen Messungen einen rekursiven Aufruf des Programms. Ist die Parklücke mindestens 50 cm lang, stoppt der Rover.
Am Anfang steht der Rover wieder neben einem parkenden Objekt.



Lösung:

Der erste Teil von **p2** entspricht vollständig dem ersten Teil von **p1**.

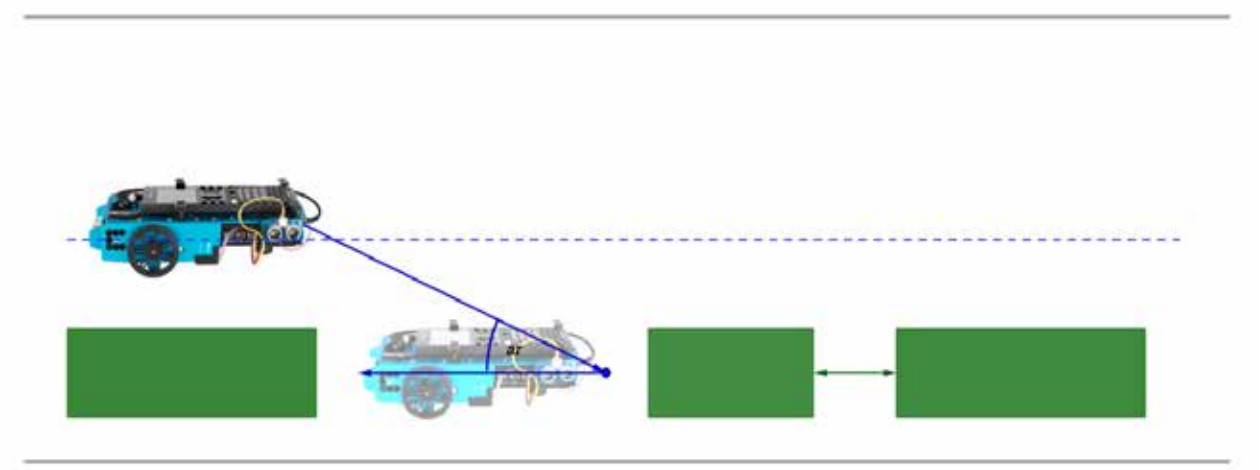


Beginn der **Rekursion**: ist die Parklücke zu klein (*l<0.5*), so wird **p2** erneut ausgeführt, bis l≥0,5 ist.

```
Define p2()=
Prgm
:Local d,l
:d=0:l=0
:Send „CONNECT RV"
:Send „CONNECT RANGER 1 TO IN 1"
:Wait 0.1
:While d<0.3
:     Send „RV FORWARD"
:     Send „READ RANGER 1"
:     Get d
:EndWhile
:Send „RV STOP CLEAR"
:While d≥0.3
:     Send „RV FORWARD"
:     Send „READ RANGER 1"
:     Get d
:EndWhile
:Send „READ RV.WAYPOINT.DISTANCE"
:Get l
:Disp „length of the slot:  „,l
:
:If l<0.5 Then
:     p2()
:Else
:     Send „RV STOP CLEAR"
:EndIf
:
:EndPrgm
```

Aufgabe 3:

Jetzt soll das Einparken an der ersten geeigneten Parklücke programmiert werden (Programm **p3**).
Nachdem der Rover eine geeignete Parklücke gefunden hat, stoppt er zunächst.
Am Anfang steht der Rover wieder neben einem geparkten Objekt.



Für das rückwärtige Einparken steuern wir mit dem Heck des Rovers einen Punkt an, der in der Mitte der Tiefe und im hinteren Drittel der Parklücke liegt.
Dazu werden der entsprechende Einschlagwinkel und die Länge der Rückfahrstrecke berechnet.
Diese Werte werden wegen Länge und Breite des Rovers etwas korrigiert.
Zum Schluss fährt der Rover in der Parklücke etwas vor.

Zur mathematischen Umsetzung verwendeten wir folgende Variablen:

physikalische Größe		Beschreibung		Variablenname
d		gemessener Abstand nach links		*d*
l		vom Rover gemessene Länge der Parklücke		*l*
l_{korr}		korrigierte Länge der Parklücke		*lcorr*
$\alpha_{rück}$		Winkel, um den sich der Rover vor dem Rückfahren dreht		*ar*
$l_{rück}$		Länge der Rückfahrstrecke		*back*
$l_{rück_{korr}}$		korrigierte Länge der Rückfahrstrecke		*backcorr*

Nach mehreren Versuchen haben wir uns für folgende Korrekturen entschieden:

- die korrigierte Länge der Parklücke $l_{korr} = l + \frac{d}{2}$
- „Rückfahrwinkel“ $\alpha_{rück} = \frac{\pi}{180} \cdot \arctan\frac{3 \cdot d}{2 \cdot l_{korr}}$
- Länge der Rückfahrstrecke $l_{rück} = \sqrt{d^2 + \frac{4}{9} {l_{korr}}^2}$
- für kleine Winkel erfolgt eine weitere Anpassung der Länge der Rückfahrstrecke.

Bemerkung:

Die Bodenbeschaffenheit beeinflusst stark die Bewegung des Rovers. Deshalb m**ü**ssen die Korrekturwerte für die Länge der Rückfahrstrecke experimentell ermittelt werden.

Lösung:

Variablen:
d: Entfernung zum linken Parkobjekt
l : Länge der Parklücke
lcorr: korrigierte Länge
ar: Winkel der Rechtsdrehung
back: Länge der Rückwärtsfahrt
backcorr: Länge der korrigierten Rückwärtsfahrt

Suchphase – wie bei **p1** und **p2**

Längenbestimmung der L**ücke**

rekursive Bestimmung der Parklücke wie in **p2**

geeignete Parklücke gefunden:

- Längenkorrektur

- Winkelbestimmung für Rückwärtsfahrt

- Länge der Rückwärtsfahrt

- kurze Rückwärtsfahrt für ar<15

- Korrektur der Länge der Rückwärtsfahrt

```
Define p3()=
Prgm
:Local d,l
:d:=0
:l:=0
:Send "CONNECT RV"
:Send "CONNECT RANGER 1 TO IN 1"
:Wait 0.1
:
:While d<0.3
:     Send "RV FORWARD"
:     Send "READ RANGER 1"
:     Get d
:EndWhile
:Send "RV STOP CLEAR"
:While d≥0.3
:      Send "RV FORWARD"
:      Send "READ RANGER 1"
:      Get d
:EndWhile
:Send "READ RV.WAYPOINT.DISTANCE"
:Get l
:Disp "length of the slot:  ",l
:
:If l<0.5 Then
:       p3()
:Else
:
:    Send "RV STOP"
:    Disp "length of the last slot:  ",l
:    lcorr:=l+((d)/(2))
:    Wait 2
:    Disp "corrected length: ",lcorr
:
:    ar:=((tan⁻¹(((3*d)/(2*lcorr)))*180)/(π))
:    Disp „angle right: „,ar
:    back:=√(d^(2)+((4)/(9))*lcorr^(2))
:
:    If ar<15 Then
:         Send „RV BACKWARD 0.1 M"
:         backcorr:=back+((0.04)/(d))
:    Else
:         backcorr:=back+((0.06)/(d))
:    EndIf
```

- Drehung nach rechts um *ar*
- Rückwärtsfahrt um *backcorr*
- Befehlsausführung abwarten
- Drehung nach links um *ar*
- kurze Vorwärtsfahrt in der Parklücke

```
:      Send „RV RIGHT EVAL(ar)“
:
:      Send „RV BACKWARD EVAL(backcorr) M“
:
:      Wait 1
:
:      Send „RV LEFT EVAL(ar)“
:
:      Send „RV FORWARD 0.3“
:
:EndIf
:EndPrgm
```

www.t3europe.eu

education.ti.com/de

T³ EUROPE

Teachers Teaching with Technology™

Texas Instruments