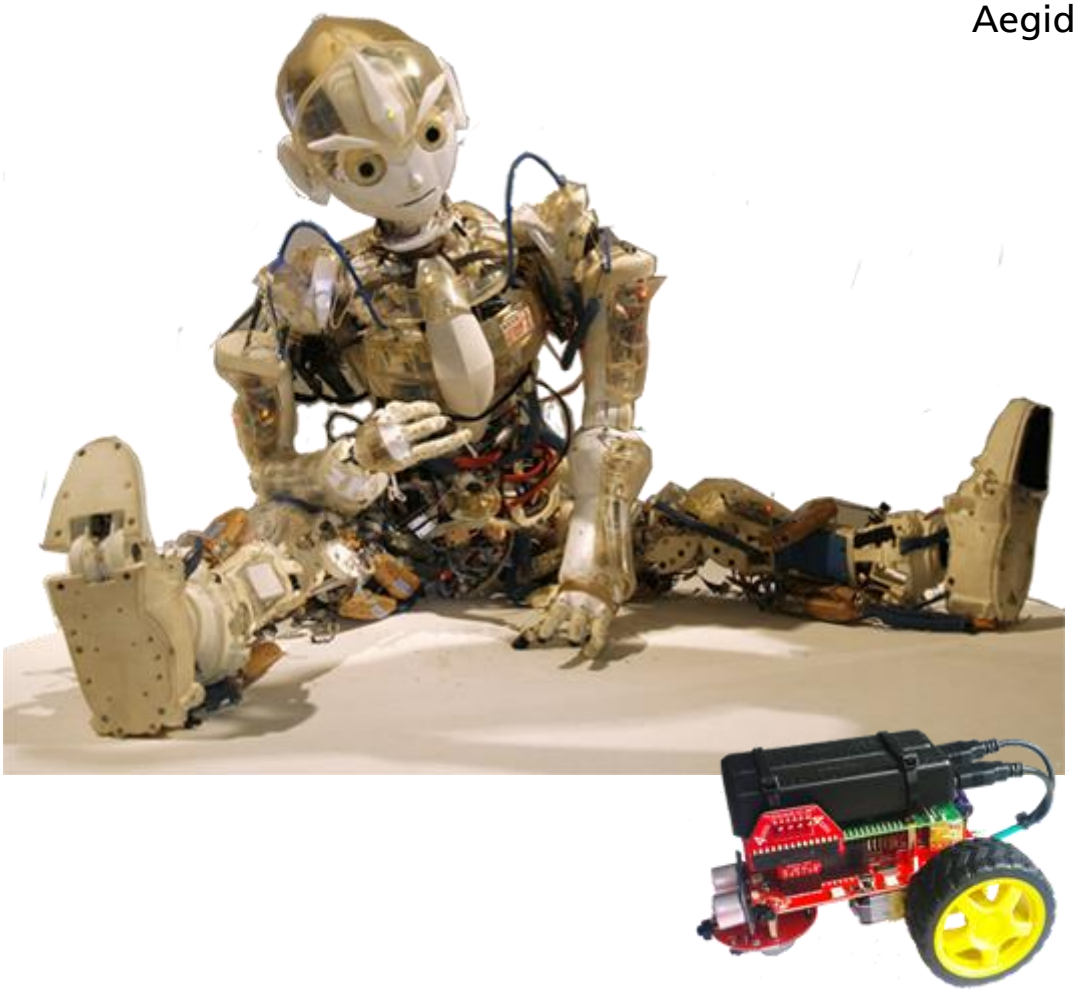
TigerJython4Kids

www.tigerjython4kids.ch

Teil 2: ROBOTIK

Jarka Arnold
Aegidius Plüss



mit TigerJython und Pi2Go

www.tigerjython4kids.ch

INHALT

Teil 2: ROBOTIK

ANHANG:

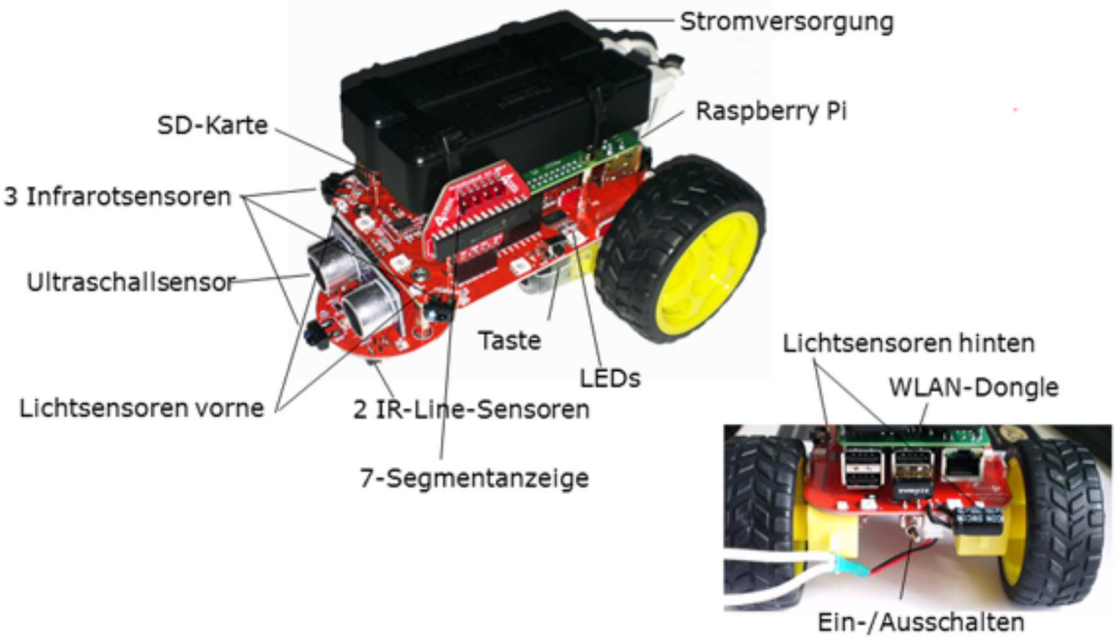
Version 2.0, August 2017

Autoren: Jarka Arnold, Aegidius Plüss
Kontakt: help@tigerjython.com

ZUSAMMENBAU

Normalerweise kriegst du den Pi2Go-Roboter als Bausatz und du musst ihn zuerst zusammenbauen. Das Standard-Modell verwendet als Stromversorgung ein Battery-Pack. Wir schlagen dir aber vor, dieses gegen zwei PowerBanks auszuwechseln. Eine Bauanleitung findest du **hier**.

Schau dir deinen Roboter genau an und identifiziere die angeschriebenen Bauteile.



SD-KARTE ERSTELLEN UND PI2GO STARTEN

Beim Einschalten des Roboters wird als erstes das Betriebssystem des Mikroprozessors von der SD-Karte geladen und gestartet. Dazu muss die SD-Karte richtig konfiguriert sein. Es ist ganz leicht, mit einem Computer, der mit dem Internet verbunden ist, die SD-Karte selbst zu erstellen. Eine Anleitung findest du **hier**:

Hast du das gemacht, so schiebst du die SD-Karten in den Pi2Go und schaltest ihn ein. Nach kurzer Zeit siehst du auf der Anzeige die aktuelle Version des Betriebssystems. Nachher versucht der Roboter, eine Verbindung mit einem Internet Hotspot aufnehmen. Da er noch keinen findet, wird als IP-Adresse 0-0-0-0 angezeigt und der Roboter geht in den Bereitschaftszustand mit einer Anzeige der Anzahl der bereits geladenen Programme: P1-3. Mit der Taste wird der Roboter bedient (Anleitung siehe Anhang).

PC UND PI2GO VERBINDEN

Du schreibst deine Roboter-Programme mit TigerJython. Zur Ausführung muss dein Computer eine WLAN-Verbindung mit dem Pi2Go haben. Dazu müssten beide Geräte auf dem gleichen WLAN-Hotspot (WLAN-Router, Access-Point) angemeldet sein. Am einfachsten verwendest du dein Smartphone also Hotspot. Du kannst aber den Raspi auch auf einem vorhandenen Router anmelden, den du auch sonst verwendest. Dies hat den Vorteil, dass du dann immer noch auf dem Internet bist.

Ist der Router in Betrieb, so musst du dem Raspi die SSID und den Sicherheitsschlüssel mitteilen, damit er sich dort anmelden kann. Du verwendest dazu unser Tool RaspiBrickConfig.

1. LOSLEGEN

DU LERNST HIER...

wie du ein erstes Roboterprogramm erstellst, es auf den Pi2Go hinunterlädst und dort ausführst.

DIE PI2GO-FIRMWARE BEDIENEN

In allen folgenden Beispielen gehen wir davon aus, dass du den Pi2Go mit der SD-Karte gestartet hast und die TCP-Verbindung gemäss der Anleitung funktioniert.

Auf dem Display siehst du jetzt eine Anzeige wie P1-3. P steht für "Programm" und 1-3 sagt, dass du im Augenblick das 1. Programm von insgesamt 3 Programmen zum Start ausgewählt hast. Mit einem Klick der Taste wird das Programm gestartet.

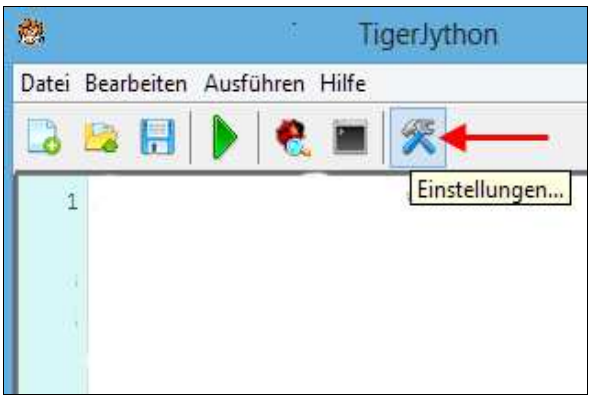
Das Programm muss immer so geschrieben werden, dass es wieder endet. Es kann von selbst enden, oder auf die Betätigung der Taste warten, beispielsweise so lange laufen, bis die Taste geklickt oder lange gedrückt wird.

Du solltest nur Programme schreiben, die diese Bedingung erfüllen!

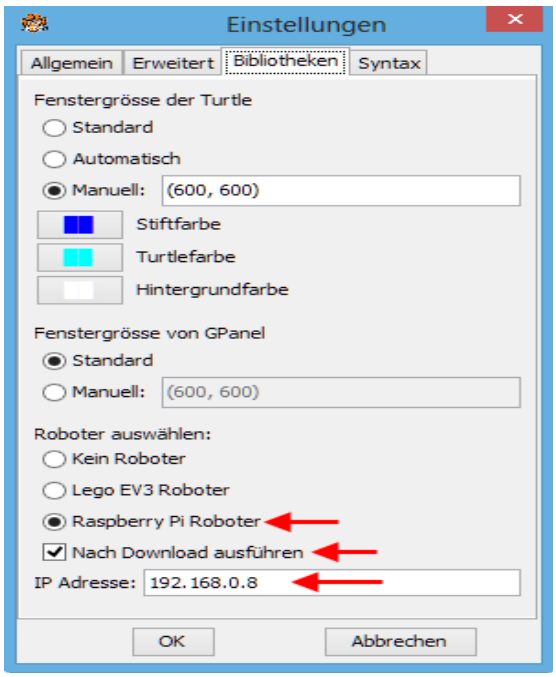
Denn um ein falsch laufendes oder blockierendes Programm abzubrechen, bleibt dir nichts anderes übrig, als den Pi2Go aus- und wieder einzuschalten.

Auf dem Brick können maximal 9 Programme gespeichert werden. Mit einem Doppelklick wird das nächste Programm ausgewählt. Wenn du also bei der Anzeige P1-3 doppelklickst, so erscheint P2-3 und mit einem Klick wird dieses Programm gestartet.

Für die Programmentwicklung startest du TigerJython und musst in den *Einstellungen* die Raspberry Pi-Robotik aktiv machen.

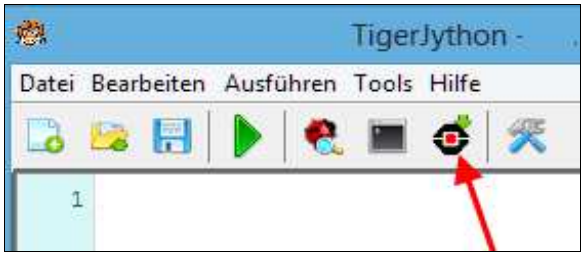


Wähle auch "Nach Download ausführen", damit das Programm ohne dein Zutun nach dem Download ausgeführt wird.



In der letzten Zeile musst du die IP-Adresse deines Roboters eingeben.

Du siehst nun im Menü von TigerJython einen zusätzlichen Button, mit dem du das Programm auf den Pi2Go downloaden und ausführen kannst.



REAL- UND SIMULATIONSMODUS

Hast du keinen realen Roboter, so kannst du einen simulierten Roboter in einer virtuellen Welt verwenden. Viele Programme laufen praktisch unverändert im Real- oder Simulationsmodus. Dazu muss einzig eine andere Bibliothek importiert werden und zwar:

`from raspibrick import *` im Realmodus

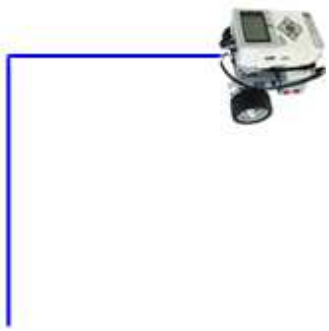
`from raspisim import *` im Simulationsmodus

Da sich die Roboter in der realen und virtuellen Welt nicht genau gleich schnell bewegen, musst du eventuell auch gewisse Zeitangaben (Timings) anpassen. Das Programm wird im Simulationsmodus mit dem grünen "Run-Pfeil" gestartet.

MUSTERBEISPIEL

Mit deinem ersten Programm bewegst du den Roboter während 2 Sekunden (2000 Millisekunden) vorwärts, während 550 Millisekunden links und während 2000 Millisekunden rückwärts.

Dazu erstellst du mit robot = Robot() ein Roboterobjekt, wobei der Bezeichner *robot* frei wählbar ist. Mit gear = Gear() erzeugst du ein Fahrwerkobjekt (bestehend aus den beiden Motoren). Nun kannst du mit Befehlen an das Fahrwerk den Roboter bewegen. Zuletzt musst du noch mit robot.exit() alle Ressourcen freigeben.



Realmodus

```
from raspibrick import *

robot = Robot()
gear = Gear()
gear.forward(2000)
gear.left(550)
gear.backward(2000)
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren)

Simulationsmodus

```
from raspisim import *

robot = Robot()
gear = Gear()
gear.forward(2000)
gear.left(550)
gear.backward(2000)
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren)

Nachdem du das Programm im Editor geschrieben oder dort eingefügt hast, kannst du den Download-Button drücken und das Programm wird auf den Roboter heruntergeladen und ausgeführt. **Pass auf, dass der Roboter nicht von Tisch herunter fällt!**

Im Simulationsmodus klickst du einfach auf den Run-Button.

MERKE DIR...

Sobald du ein Programm auf den Brick hinuntergeladen hast, erhöht sich in der Anzeige die totale Anzahl der Programme (ausser es hat bereits 9 davon). Das neue Programm hat die Nummer 1 und die anderen verschieben sich um 1 nach oben. Wenn du auf der Anzeige P1-1 stehst, kannst du das zuletzt hinuntergeladene Programm mit einem Tastenklick wieder ausführen. Mit einem Doppelklick kommst du von einem Programm zum nächsten.

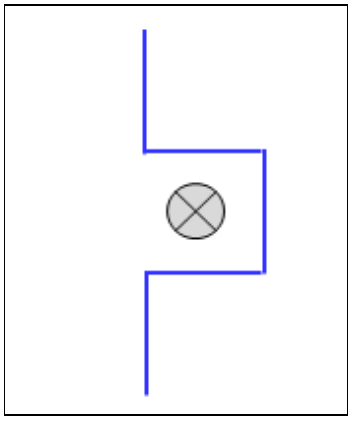
Da die Programme im Real- und Simulationsmodus sich nur in der Importzeile unterscheiden, fügen wir im Folgenden in den Musterbeispielen beide Importzeilen ein, wobei der Import für den Realmodus mit dem Kommentarzeichen # auskommentiert (deaktiviert) ist. Je nachdem, welchen Modus du verwenden willst, musst du das Zeichen # vor die nicht verwendete Importzeile setzen.

ZUM SELBST LÖSEN

1. Der Roboter soll einen Gegenstand umfahren. Er soll also:

   - 1000 ms vorwärts fahren
   - 550 ms rechts drehen
   - 1000 ms vorwärts fahren
   - 550 ms links drehen
   - 1000 ms vorwärts fahren
   - 550 ms links drehen
   - 1000 ms vorwärts fahren
   - 550 ms rechts drehen
   - 1000 ms vorwärts fahren

   und dann anhalten.



2. Der Roboter soll eine bestimmte Strecke vorwärts fahren, dann um 180 Grad drehen und wieder zurückfahren. Wie du siehst, kannst du dies im Realmodus nur ungenau erreichen, da wegen Unregelmässigkeiten der Motoren und unterschiedlicher Bodenbeschaffenheit die Bewegung nicht exakt reproduzierbar ist. Dies ist für die Robotik typisch und du musst lernen, mit solchen Ungenauigkeiten umzugehen.

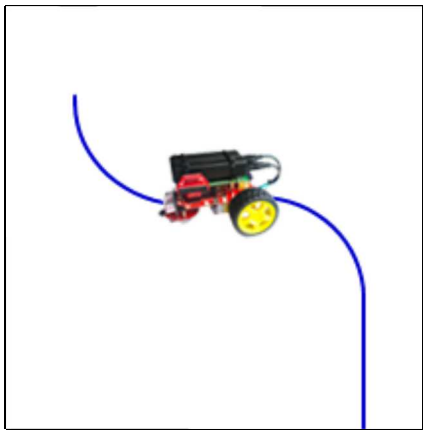
2. ROBOTER BEWEGEN

DU LERNST HIER...

einige weitere Befehle kennen, um den Roboter zu bewegen und was blockierende und nicht-blockierende Funktionen sind.

MUSTERBEISPIEL

Der Roboter soll kurz vorwärts fahren und dann eine Ausweichbewegung auf einem Linksbogen machen. Dann korrigiert es diese mit einem Rechtsbogen und hält an. Das Bogenfahren machst du mit den Befehlen leftArc(radius, time) bzw. *rightArc(radius, time),* wobei der Bogenradius in Metern und die Zeit in Millisekunden angegeben wird. Mit setSpeed(speed) kannst du noch die Geschwindigkeit im Bereich 0..100 wählen.



```
from raspisim import *
#from raspibrick import *

robot = Robot()
gear = Gear()
gear.setSpeed(40)
gear.forward(2000)
gear.leftArc(0.3, 5000)
gear.rightArc(0.3, 5000)
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

Der Aufruf *gear.forward(2000)* setzt den Roboter während 2 Sekunden in den Vorwärtszustand. Am Ende dieser Zeit stoppt der Roboter und erst jetzt fährt das Programm mit der nächsten Anweisung weiter. Das Programm blockiert also während 2 Sekunden.

In vielen Anwendungen muss der Roboter auch während seinen Bewegungen in der Lage sein, Programmanweisungen auszuführen. Dazu sind die **blockierenden Funktionen** wie *forward(2000)* nicht geeignet. Mit **nicht-blockierenden Funktionen** schreibst du das Musterprogramm wie folgt:

- Du versetzt den Roboter mit *forward()* (ohne Zeitparameter) in den Vorwärts-Zustand. Er bewegt sich jetzt vorwärts und das Programm fährt sofort mit der nächsten Anweisung weiter
- Das Programm wartet während 2 Sekunden (es könnte während dieser Zeit aber auch etwas anderes tun)
- Du versetzt den Roboter mit *leftArc()* in den Linksbogen-Zustand. Er bewegt sich jetzt auf einem Linksbogen und das Programm fährt mit nächsten Anweisung weiter
- Du wartest im Programm 5 Sekunden

- Du versetzt den Roboter mit *rightArc()* in den Rechtsbogen-Zustand. Er bewegt sich jetzt auf einem Rechtsbogen und das Programm fährt mit nächsten Anweisung weiter
- Du wartest im Programm um 5 Sekunden
- Du stoppst mit *robot.exit()* den Roboter (du könntest auch *stop()* verwenden).

Das Musterbeispiel mit nicht-blockierenden Funktionen sieht dann so aus:

```
from raspisim import *
#from raspibrick import *

robot = Robot()
gear = Gear()
gear.setSpeed(40)
gear.forward()
Tools.delay(2000)
gear.leftArc(0.3)
Tools.delay(5000)
gear.rightArc(0.3)
Tools.delay(2000)
robot.exit()
```

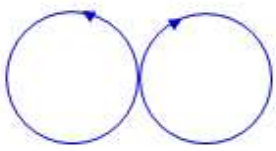
Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MERKE DIR…

Du kannst ein Programm mit blockierenden oder nicht-blockierenden Funktionen schreiben. Wenn dein Programm aber während der Bewegung andere Aufgaben erledigen muss, so musst du nicht-blockierende Funktionen verwenden.

ZUM SELBST LÖSEN

1a) Der Roboter soll sich so lange auf einen Linksbogen bewegen, bis er einen ganzen Kreis zurückgelegt hat. Danach soll er sich gleich lange auf einem rechtsbogen bewegen. Verwende blockierende Funktionen.



1b) Schreibe dasselbe Programm mit den nicht-blockierenden Funktionen *leftArc(radius)* und *rightArc(radius)*.

2a) Erstelle mit einigen Büchern einen Parcours und schreibe ein dazugehörendes Programm so, dass der Roboter vom Start zum Ziel fährt.

Für die Simulation wird mit *RobotContext.useBackground("sprites/bg.gif")* das Hintergrundbild *bg.gif* angezeigt, das sich im Unterverzeichnis *sprites* befindet.

Mit *RobotContext.setStartPosition()* kannst du den Roboter bei Programmstart an eine bestimmte Stelle setzen. (Bildschirmkoordinaten 0..500, Nullpunkt oben links, x-Achse nach rechts, y-Achse nach unten). Das Programm beginnt also mit

```
from simrobot import *
RobotContext.useBackground("sprites/bg.gif")
RobotContext.setStartPosition(310, 480)
```

2b) Erstelle ein eigenes Bild, das deinem realen Parcours entspricht (Grösse 501x501 Pixels) und kopiere es in das Unterverzeichnis *sprites*, in dem sich dein Programm befindet. Schreibe ein dazu gehörendes Programm im Real- und Simulationsmodus.

Anmerkung: Du kannst im Realmodus die Zeilen mit *RobotContext* stehen lassen, da sie keinen Einfluss auf die Programmausführung haben.

3. REMOTESTEUERUNG

DU LERNST HIER...

wie ein Roboterprogramm, das auf dem PC läuft, den Roboter steuern kann (wir nennen dies den "Remote Modus"). Die Programme im Remote Modus sind exakt die gleichen wie im autonomen Modus, sie werden aber auf dem PC mit dem grünen *Run*-Button gestartet. Da die Programme im Remote und autonomen Modus die gleichen sind, nennen wir sie auch "*Programme für den Realmodus*".

MUSTERBEISPIEL

Um zu zeigen, dass du Programme zwischen den verschiedenen Modi austauschen kannst, nehmen wir das Musterbeispiel aus Kapitel 1. Der Roboter soll die Bewegung aber viel mal wiederholen. Das kannst du mit einer *for-Schleife* programmieren. Mit for i in range(4): werden die nachfolgenden eingerückten Zeilen 4 mal wiederholt.

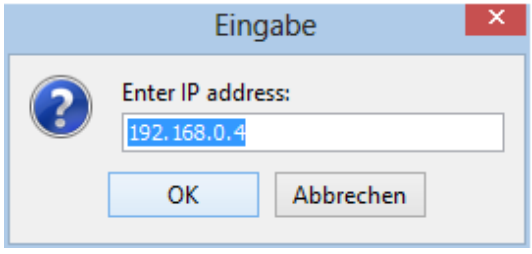
Normalerweise ist der Pi2Go nach dem Booten bereit, autonome Programme auszuführen.

Damit er remotegesteuert funktioniert, musst du auf dem Pi2Go den sogenannten Brickgate-Server starten. Um den Brickgate-Server zu starten, drückst du den Button so lange, bis ***HOLd*** angezeigt wird.

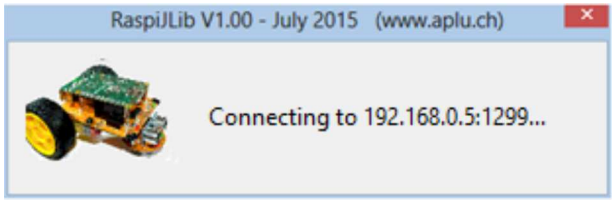

```
# Ro3a.py

from raspisim import *
#from raspibrick import *

robot = Robot()
gear = Gear()
for i in range(4):
    gear.forward(2000)
    gear.left(550)
    gear.backward(2000)
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

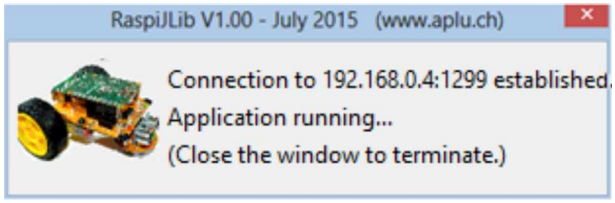
Statt wie vorhin den Download-Button zu klicken, klickst du jetzt den grünen Run-Button. Es erscheint ein Dialogfenster, wo du die IP-Adresse des Pi2Go eingeben musst.



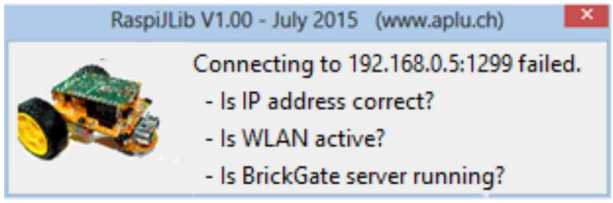
Nach dem Klicken des OK-Buttons öffnet sich ein weiteres Dialogfenster.



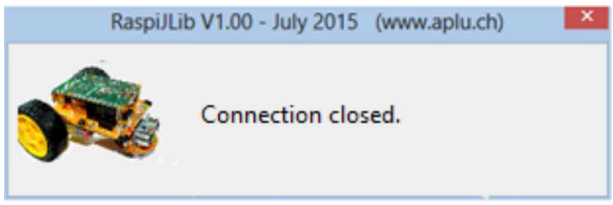
Falls die Verbindung hergestellt werden kann, so startet nach kurzer Zeit das Programm und im Dialogfenster wird eine Erfolgsmeldung ausgeschrieben. Auf dem Pi2Go-Display erscheint kurz *Conn*.



Misslingt die Verbindung, so siehst du die nebenstehende Meldung und du musst nach der Ursache suchen. (Meist ist der Brickgate-Server nicht gestartet oder die WLAN-Verbindung nicht erstellt.)



Ein Programm im Remote Modus wird durch Schliessen des Dialogfensters beendet. Die Verbindung zum Roboter wird automatisch unterbrochen.



MERKE DIR...

Im Remote-Modus wird das Programm im TigerJython mit dem grünen *Run*-Button gestartet und auf dem PC ausgeführt. Während der Programmausführung werden die Befehle via WLAN zum Roboter gesendet. Da die Programme auf diese Art schneller ausgeführt werden, kannst du während der Programmentwicklung den Remote-Modus verwenden und anschliessen das Programm unverändert im autonomen Modus auf den Pi2Go herunterladen.

FERNSTEUERUNG MIT DER TASTATUR

Im Remote Modus kannst du den Roboter mit der Tastatur (oder mit einem anderen am PC angeschlossenen Eingabegerät, z.B. der Maus oder einem Touchscreen) fernsteuern, indem du die entsprechenden Befehle an den Roboter sendest. Im folgenden Programm macht es Spass, mit den Cursortasten den Roboter zu bewegen. Das Programmkonzept ist wichtig und du solltest es dir gut merken. Es funktioniert so:

In einer while-Schleife prüfst du mit einer geschachtelten if-Struktur ständig, ob eine der Cursortasten oder die Enter-Taste gedrückt wurde. Je nachdem wird dem Fahrwerk der entsprechende Befehl gesendet. Du brichst die *while*-Schleife und damit das Programm ab, wenn die Escape-Taste gedrückt wurde.

```
from raspisim import *
# from raspibrick import *

robot = Robot()
gear = Gear()

while not robot.isEscapeHit():
    if robot.isDownHit():
        gear.backward()
```

```
    elif robot.isUpHit():
        gear.forward()
    elif robot.isLeftHit():
        gear.leftArc(0.1)
    elif robot.isRightHit():
        gear.rightArc(0.1)
    elif robot.isEnterHit():
        gear.stop()
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MERKE DIR…

Wir werden diese Programm-Konstruktion mit der while-Schleife, die mit der Escape-Taste (bzw. im autonomen Mode mit der Taste des Pi2Go) abgebrochen werden kann, noch sehr oft verwenden.

Bevor ein Programm im Remote Modus gestartet wird, muss der Brickgate-Server gestartet sein und auf dem Display *HOLd* erscheinen. Du kannst den Brickgate-Server beenden, indem du die Taste lange drückst. Der Roboter ist nun wieder für den autonomen Modus bereit.

ZUM SELBST LÖSEN

1a) Erstelle wieder einen lustigen Parcours und versuche, ihn mit der PC-Fernsteuerung zu durchlaufen. Verwende dazu für die Links- bzw. Rechtsdrehung den Befehl *gear.left()* bzw. *gear.right()*. Du kannst auch mit *gear.setSpeed(speed)* und einem Wert von *speed* zwischen 0 und 100 die Geschwindigkeit ändern und einen Wettbewerb veranstalten, wer den Parcours in kürzester Zeit durchläuft.

1b) Verwende wie in Kapitel 1 das Hintergrundbild *bg.gif* und führe das Spiel im Simulationsmodus durch.

```
RobotContext.setStartPosition(310, 480)
RobotContext.useBackground("sprites/bg.gif")
```

1c) Erstelle ein eigenes Hintergrundbild für das Spiel.

2. Schreibe ein Programm, bei dem du mit den Up- und Down-Tasten die Vorwärts-Geschwindigkeit des Roboters wählen kannst. Beachte, dass eine mit *gear.setSpeed()* gesetzte Geschwindigkeit erst beim nächsten Bewegungsbefehl übernommen wird und dass bei negativen Geschwindigkeiten der Roboter rückwärts fährt. Die Left- und Right-Tasten sollen den Roboter auf Links- bzw. Rechtsbogen bewegen. Durchfahre damit wieder einen Parcours.

   Anleitung: Es ist eine gute Idee, das Programm zuerst im Simulationsmodus zu entwickeln. Setze beim Start die Geschwindigkeit auf 0.

3*. Erfinde ein eigenes Fernsteuerungsprogramm.

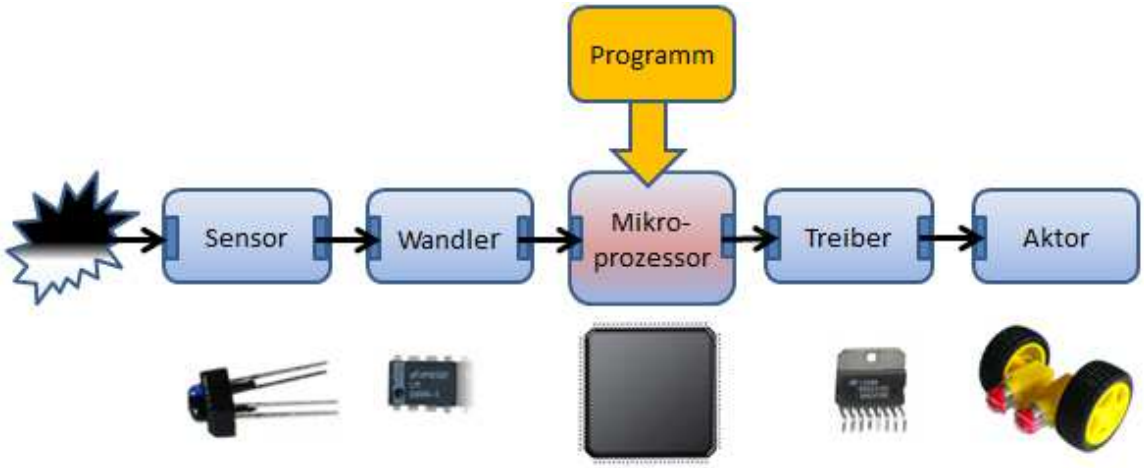
4. INFRAROT-LICHTSENSOR

DU LERNST HIER...

dass die meisten Roboter mit Sensoren ausgerüstet sind, die wie unsere menschlichen Sinne die Umgebung erfassen und wie Messgeräte die Messdaten an den eingebauten Mikroprozessor melden, der sie auswertet und entsprechende Aktionen ausführt.

MUSTERBEISPIEL

Dein Roboter soll erkennen, wenn er auf einer schwarzen oder weissen Unterlage fährt. Dazu muss die Helligkeit mit einem Lichtsensor (hier für infrarotes Licht) gemessen und der digitalisierte Wert dem Mikroprozessor übergeben werden. Dieser soll dann über einen Motortreiber den linken und rechten Motor so steuern, dass der Roboter der Kante entlang fährt. Du brauchst dazu folgende Komponenten:



Softwaremässig ist ein Infrarot-Lichtsensor ein Objekt *InfraredSensor* und die beiden Motoren ein Objekt *Gear*, die du mit den Variablen *irLeft, irRight* und *gear* definierst. Um einen Messwert zu erhalten, rufst du die Funktion *getValue()* auf, was dir entweder 0 oder 1 zurückgibt, je nachdem ob der Sensor ein kleine (dunkel) oder eine grosse (hell) Lichtintensität feststellt.

In der virtuellen Welt musst du noch ein Bild des Hintergrunds angeben, sowie festlegen, wo sich der Roboter beim Start befindet.

Die Strategie hättest auch du selbst herausgefunden. Damit das Fahrzeug auf der Kante mit der schwarzen Fläche auf der rechten Seite fährt, musst du geradeaus fahren, wenn der linke Sensor hell und der rechte Sensor dunkel sieht. Du fährst einen Rechtsbogen, wenn beide Sensoren hell sehen und einen Linksbogen, wenn beide Sensoren dunkel "sehen".

```
from raspisim import *
#from raspibrick import *

RobotContext.useBackground("sprites/oval.gif")
RobotContext.setStartPosition(250, 460)

robot = Robot()
gear = Gear()
irLeft = InfraredSensor(IR_LINE_LEFT)
irRight = InfraredSensor(IR_LINE_RIGHT)
gear.forward()

while not isEscapeHit():
    vLeft = irLeft.getValue()
    vRight = irRight.getValue()
    if vLeft == 1 and vRight == 0:
        gear.forward()
    elif vLeft == 1 and vLeft == 1:
        gear.rightArc(0.1)
    elif vLeft == 0 and vRight == 0:
        gear.leftArc(0.1)
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MERKE DIR…

dass Sensoren als Objekte aufgefasst werden und man mit der Funktion *getValue()* einen Messwert holen kann.

Du kannst das Programm im autonomen, fremdgesteuerten oder Simulationsmodus laufen lassen.

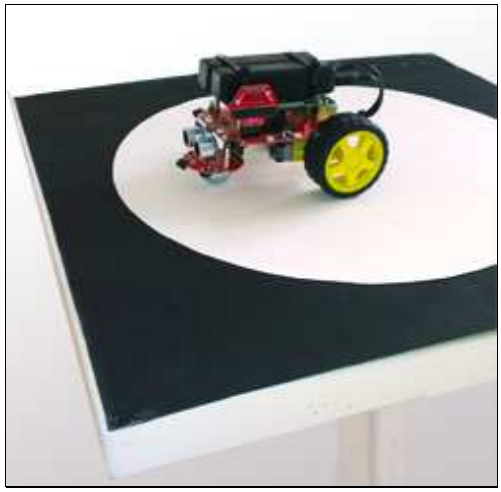
Beim Drücken der Escape-Taste wird die Bedingung *robot.isEscapeHit()* wahr und die while-Schleife bricht ab.

ZUM SELBST LÖSEN

1a) Untersuche, ob das Programm im Musterbeispiel auch richtig ist, wenn der Roboter sich in der anderen Richtung bewegen soll.

1b) Untersuche, wie sich die Wahl des Kurvenradius (hier 0.1) auf die Bewegung auswirkt. Verstehst du das Verhalten?

2. Der Roboter soll sich auf quadratischen Tisch mit weissem Innenkreis bewegen, ohne herunterzufallen. (Es kann sich auch um eine Papiervorlage auf dem Boden handeln.) Dabei startet er in der Mitte und fährt geradeaus. Erkennt er den Rand, so fährt er rückwärts, dreht um ungefähr 90 Grad nach links und fährt dann wieder vorwärts.



Für den Simulationsmodus benötigst du folgenden Context:

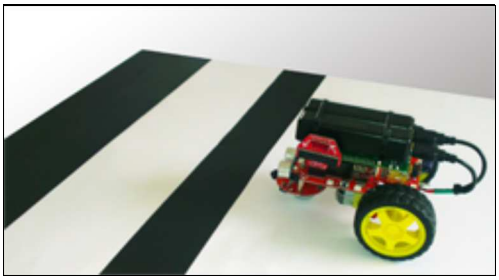
```
RobotContext.useBackground("sprites/circle.gif")
RobotContext.setStartPosition(250, 200)
```

Verwende wieder eine Schleife

```
while not robot.isEscapeHit():
```

Du kannst einen oder zwei Sensoren verwenden. Mit zwei ist die Bewegung aber regelmässiger.

3. Der Roboter soll auf weisser Unterlage starten und dann beim Erkennen des ersten schwarzen Streifens 2 Sekunden anhalten und weiterfahren (wie bei einer Haltestelle). Beim Erkennen des zweiten Streifens stoppt er definitiv (Endbahnhof).

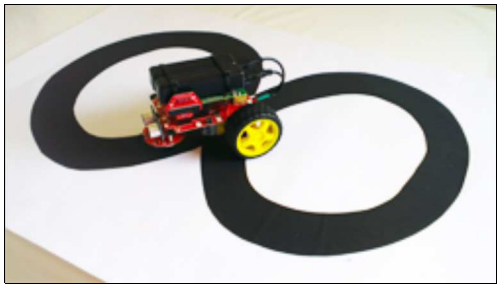


Für den Simulationsmodus benötigst du folgenden Context:

```
RobotContext.useBackground("sprites/twostrips.gif")
RobotContext.setStartPosition(250, 450)
```

Bemerkung: Die Lösung ist nicht ganz einfach, denn wenn der Roboter bei der Haltestelle anhält, weil er schwarz sieht, so sieht er immer noch schwarz, wenn er weiterfährt. Du musst dir mit einer Variablen s merken, in welchem Zustand der Roboter gerade ist, z.B. s = 0: erstes Fahren auf weiss, s = 1: Haltestelle angehalten, s = 2: zwischen Haltestelle und Endbahnhof auf weiss, s = 3: am Endbahnhof. Falls du beim Programmieren stecken bleibst, so kannst du hier ein wenig spicken .

4*. Das Fahrzeug soll in eine bestimmten Richtung die Achterbahn durchfahren. Folgende Strategie hat sich bewährt:



- Sehen beide Lichtsensoren schwarz, so fährst du gerade aus
- Sieht der linke weiss und der rechte schwarz, so fährst du einen Rechtsbogen
- Sieht der rechte weiss und der linke schwarz, so fährst du einen Linksbogen
- Sehen beide weiss, so fährst du rückwärts

Für den Simulationsmodus benötigst du folgenden Context:

```
RobotContext.useBackground("sprites/track.gif")
RobotContext.setStartPosition(250, 450)
```

Verwende wieder eine Schleife

```
while not robot.isEscapeHit():
```

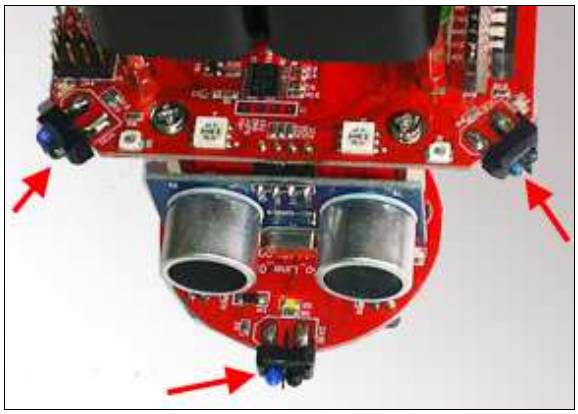
5. HINDERNISDETEKTOREN

DU LERNST HIER...

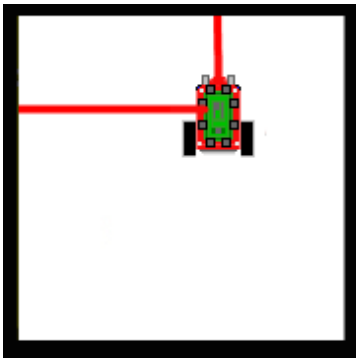
wie du die drei oberen Infrarotsensoren dafür einsetzen kannst, Hindernisse zu erkennen und entsprechend zu reagieren.

MUSTERBEISPIEL

Mit den oberen Infrarotsensoren kann der Pi2Go erkennen, wenn sich in seiner Nähe ein Hindernis befindet. In der Pi2Go-Bibliothek werden diese Sensoren mit IR_CENTER, IR_LEFT und IR_RIGHT bezeichnet.



In diesem Beispiel soll sich der Roboter in einer quadratischen oder rechteckigen Kiste so bewegen, dass er ungefähr rechtwinklig zueinander stehende Bahnen abfährt.



Um den Zustand des Sensors zu erfassen, rufst du in einer *while*-Schleife die Funktion getValue() auf. Diese liefert 1, falls sich das Hindernis in der Nähe befindet, sonst liefert sie 0. Für die Simulation musst du noch in der virtuellen Welt mit *RobotContext.useObstacle()* eine Kiste erstellen.

```
from raspisim import *
#from raspibrick import *

RobotContext.useObstacle("sprites/field1.gif", 250, 250)

robot = Robot()
gear = Gear()
ir = InfraredSensor(IR_CENTER)
gear.forward()

while not robot.isEscapeHit():
    if ir.getValue() == 1:
        gear.backward(500)
        gear.left(550)
        gear.forward()
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

MERKE DIR…

Dein Programm "bemerkt" das Hindernis nur, falls es *getValue()* aufruft. Ist es in den Funktionen *backward(500)* oder *left(550)* blockiert, kannst du mit der Hand ein Hindernis vortäuschen und es geschieht rein gar nichts.

ZUM SELBST LÖSEN

1. Der Roboter bewegt sich in einer Kiste mit einem Ausgang. Du lässt ihn in der Mitte der Kiste starten und er soll möglichst rasch den Ausgang finden. Dabei darfst du aber nicht davon ausgehen, dass der Roboter von sich aus schon "weiss", wo sich der Ausgang befindet.

   Für die Simulation benötigst du folgenden Context:

```
RobotContext.useObstacle("sprites/trap.gif", 250, 250)
```

2. Ein Kanalroboter soll sich in einem rechtwinklig gebogenen Röhrensystem selbständig, also ohne Fernsteuerung, zurechtfinden.

   Wie bei vielen Aufgaben ist es sinnvoll, das Problem zuerst im Simulationsmodus zu lösen. Verwende dazu die Infrarotsensoren IR_LEFT und IR_RIGHT. Wenn du Lust hast, so kannst du aber auch einen echten Parcours bauen.

   Für die Simulation lautet der Context:

```
RobotContext.useBackground("sprites/racetrack.gif", 250, 250)
RobotContext.setStartPosition(420, 460)
```

6. ULTRASCHALLSENSOR

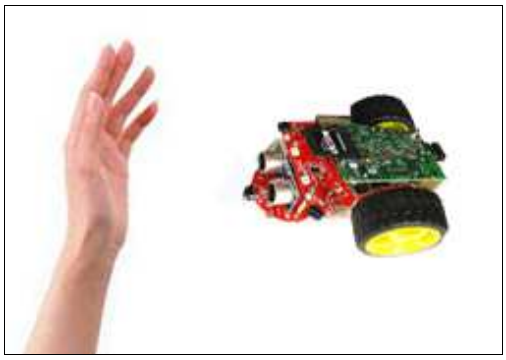
DU LERNST HIER...

wie du einen Ultraschallsensor dafür verwenden kannst, um die Distanz zu einem Objekt zu bestimmen. Dazu sendet der Sensor ähnlich wie ein Radar einen kurzen Ultraschall-Puls aus und bestimmt die Zeit, die dieser benötigt, um zum Objekt und wieder zurück zu laufen. Daraus kann er aus der bekannten Schallgeschwindigkeit die Distanz ermitteln Der Sensor liefert die Distanz in cm im Bereich von ca. 5 cm und 100 cm, wobei -1 abgegeben wird, wenn kein Objekt im Messbereich ist.

MUSTERBEISPIEL

Dein Programm soll dafür sorgen, dass der Roboter möglichst gut in einer bestimmten Distanz zu deiner Hand bleibt. Ist er zu Nahe, soll er rückwärts fahren, sonst soll er vorwärts fahren.

Wiederum wird der Sensor durch ein Objekt UltrasonicSensor in das Programm eingebunden. Dein Programm fragt in einer Wiederholschleife jede halbe Sekunde mit getDistance() die aktuelle Distanz ab. Je nach dem erhaltenen Wert, sagst du dem Roboter, dass er vorwärts oder rückwärts fahren muss.



```
from raspibrick import *

robot = Robot()
gear = Gear()
us = UltrasonicSensor()
gear.setSpeed(15)
gear.forward()

while not isEscapeHit():
    d = us.getValue()
    if d < 20:
        gear.backward()
    else:
        gear.forward()
robot.exit()
```

Programmcode markieren (Ctrl+C kopieren, Ctrl+V einfügen)

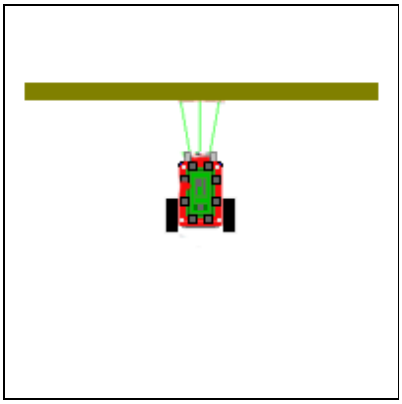
MERKE DIR...

dass der Ultraschallsensor nicht häufiger als ungefähr alle ½ Sekunde einen Messwert liefern kann. Darum solltest du ihn in der Wiederholschleife auch nicht öfter abfragen. Du kannst das Programm nur für den Realmodus verwenden.

ZUM SELBST LÖSEN

1. Der Roboter soll in senkrechter Richtung zu einer Wand fahren und bei rund 10 cm zur Wand den Rückwärtsgang einschalten und 3 s. rückwärts fahren. Nachher soll er wieder vorwärts fahren.

   Für die Simulation verwendest du an Stelle der Wand einen horizontalen Balken, den du mit folgendem Context erzeugt



```
target = [[200, 10], [-200, 10], [-200, -10], [200, -10]]
RobotContext.useTarget("sprites/bar0.gif", target, 250, 100)
```

2a. Ein Roboter mit einem Ultraschallsensor soll ein höherer Gegenstand (Säule aus Pappkarton, Kerze, leichte Büchse...) finden, hinzufahren und diesen umstossen. Der Roboter steht beim Start in einer beliebigen Richtung, d.h. er muss zuerst am Ort drehen, bis er den Gegenstand registriert.

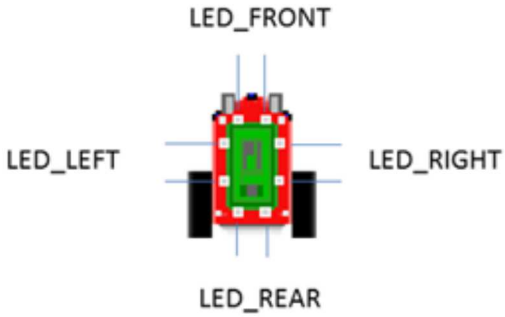
2b. Dasselbe, aber der Roboter soll wenige cm vor der Säule anhalten

2c. Dasselbe, aber der Roboter soll mehrere Säulen umstossen.
Organisiere einen Wettbewerb. Wem gelingt es am schnellsten?

7. LEDs

DU LERNST HIER...

wie du die 8 farbigen LEDs ansprechen kannst, die vorne, seitwärts links, hinten und seitwärts rechts angebracht sind. Diese werden immer als ein Paar verwendet und mit den Konstanten *LED_FRONT, LED_LEFT, LED_REAR, RED_RIGHT* bzw. den Zahlen 0, 1, 2, 3 identifiziert.



MUSTERBEISPIEL

Mit deinem Programm willst du ein Lauflicht erstellen, wobei die vorderen LEDs rot, die linken grün, die hinteren blau und die rechten gelb leuchten. Ein LED-Paar ist softwaremässig wiederum ein Objekt, das du mit Angabe seiner Identifkation erstellst, also:

Position	Mit Konstanten	Mit Zahlen
Vorne	ledFront = Led(LED_FRONT)	ledFront = Led(0)
Links	ledLeft = Led(LED_LEFT)	ledLeft = Led(1)
Hinten	ledRear = Led(LED_REAR)	ledRear = Led(2)
Rechts	ledRight = Led(LED_RIGHT)	ledRight = Led(3)

Um ein LED-Paar in einer bestimmten Farbe einzuschalten, verwendest du die Funktion *setColor(color),* wobei du die Farbe als String (X11-Color) angibst, also beispielsweise

ledFront.setColor("red").

Du kannst auch *setColor(r, g, b)* mit den drei RGB-Werten (je im Bereich 0...255) verwenden, also für rot:

```
ledFront.setColor(255, 0, 0).
```

Um ein LED-Paar auszuschalten, verwendest du die Farbe *"black"* bzw. den RGB-Wert (0, 0, 0).

```
from raspisim import *
#from raspibrick import *

robot = Robot()
ledFront = Led(LED_FRONT)
ledLeft = Led(LED_LEFT)
ledRight = Led(LED_RIGHT)
ledRear = Led(LED_REAR)

while not robot.isEscapeHit():
    ledFront.setColor("red")
    Tools.delay(2000)
    ledFront.setColor("black")
    ledLeft.setColor("green")
    Tools.delay(2000)
```

```
    ledLeft.setColor("black")
    ledRear.setColor("blue")
    Tools.delay(2000)
    ledRear.setColor("black")
    ledRight.setColor("white")
    Tools.delay(2000)
    ledRight.setColor("black")
robot.exit()
```

MERKE DIR…

dass du auch alle LEDs miteinander ansprechen kannst. Dazu verwendest du beim Aufruf nicht eine Objektvariable, sondern den Präfix *Led*.

Alle LEDs mit einer bestimmten Farbe einschalten:
`Led.setColorAll(farbe)` oder `Led.setColorAll(r, g, b)`

Alle LEDs ausschalten:
`Led.clearAll()` oder `Led.setColorAll("black")`

Beispielsweise kannst du alle LEDs wie folgt blinken lassen:

```
from raspisim import *
#from raspibrick import *

robot = Robot()
while not robot.isEscapeHit():
    Led.setColorAll("white")
    Tools.delay(200)
    Led.clearAll()
    Tools.delay(200)
robot.exit()
```

ZUM SELBST LÖSEN

1. Der Roboter soll in senkrechter Richtung zu einer Wand fahren und bei rund 10 cm zur Wand den Rückwärtsgang einschalten und 3 s. rückwärts fahren. Nachher soll er wieder vorwärts fahren.

2. Lasse den Roboter einer kurvenreichen Kante entlang fahren. Wenn er auf einem Rechtsbogen fährt, so schalte die rechten LEDs ein, wenn er auf einem Linksbogen fährt, so schalte die linken LEDs ein und wenn er geradeaus fährt, dann schalte alle LEDs aus.

   Verwende im Simulationsmodus den Context:

```
RobotContext.useBackground("sprites/border.gif")
RobotContext.setStartPosition(230, 490)
```

ANHANG : INSTALLATION UND BEDIENUNG

Verwende für die Vorbereitung des Pi2Go-Roboters die Online-Version dieses Lehrgangs. Hier findest du nur eine gekürzte Anleitung.

SD-KARTE ERSTELLEN

Hole dir das gezippte Image von hier (raspibrick.zip, 2.4 GB) herunter und packe es aus. (Du benötigst dazu ein Unzipper, der in der Lage ist, mit sehr grossen Dateien umzugeben, z.B. WinRar). Lege eine mindestens 8 GB grosse SD-Karte in den Kartenleser und starte ein Werkzeug, mit dem man img-Dateien auf SD-Karten kopieren kann. (Empfehlung: Unter Windows *USB Image Tool,* unter Mac OS oder Linux *ddutility*.) Achtung: Wähle wirklich den USB-Datenträger der SD-Karte, sonst zerstörst du möglicherweise einen falschen Datenträger.

WLAN-ROUTER (HOTSPOT) KONFIGURIEREN

Für den Einsatz im Unterricht ist es vorteilhaft, einen eigenen WLAN-Router zu verwenden, den man selbst konfigurieren kann.

Ein Internet-Zugang ist nicht unbedingt erforderlich. Ein günstiges und von uns erprobtes Modell ist TP-LINK 300MBps Wireless Router TL-WR841N. Die folgende Beschreibung bezieht sich auf dieses Modell, kann aber leicht sinngemäss auf ein anderes Modell übertragen werden.

- Verbinde eine der LAN-Buchsen des Routers mit deinem Computer (verwende nicht die besonders ausgezeichnete WAN-Buchse)

- Starte einen Web-Browser und gib die standardmässige IP-Adresse des Routers ein (oft 192.168.0.1 oder 192.168.1.1). Der Router muss sich mit einer Einloggmaske ankündigen

- Login beim TP-Link: user: admin, password: admin (aus der Anleitung entnehmen)
  Gehe in das Menü *Wireless/Wireless Setting* und gebe eine SSID ein (wenige Zeichen, die man sich gut merken kann)
  Wähle unter *Wireless/Wireless Security* WPA/WPA2-Personal und ein Wireless Password

- Nur für Router, die diese Option unterstützen: Man kann jedem Raspberry Pi eine fixe IP-Adresse zuteilen, die bei jedem Einschalten gleich bleibt. Dies ist für dich zuhause, aber insbesondere auch im Klassenverband vorteilhaft.

  Wähle im Menü *DHCP client list*. Notiere dir die MAC-Adresse des Raspberry Pi. Unter *Address Reservation* kann man dieser Mac-Adresse eine fixe IP-Adresse zuordnen z.B. 192.168.0.20

BEDIENUNG DES PI2GO-ROBOTERS MIT DER TASTE

Aktion	Zustand	Anzeige
Kippschalter auf On	Einschalten	leer
	Bereit für den autonomen Modus Zeigt aktuell gewähltes und Anzahl gespeicherter Programme (P1-3: erstes von drei Programmen gewählt)	P1-3
Kurzer Klick auf Taste	Startet gespeichertes Programm	
Doppelklick auf Taste	Programm auswählen	P2-3
Langes Drücken der Taste	Startet BrickGate Server	HOLd
Langes Drücken der Taste	Falls BrickGate Server läuft: Stoppt den BrickGate Server	P1-3
Langes Drücken der Taste	Falls BrickGate Server nicht läuft und der vordere IR-Sensor aktiviert ist:: Fährt das System nach einer Bestätigung herunten. Ausschalten ca. nach 30 Sekunden	oooo BYE
Kurzer Klick auf Taste	Falls rechter IR-Sensor aktiviert	IP-Adresse
Doppelklick auf Taste	Falls vorderer IR-Sensor aktiviert, löscht alle gespeicherte Programme	dEL

ÜBER DIE AUTOREN

Jarka Arnold war als Dozentin an der Pädagogischen Hochschule Bern für die Informatikausbildung angehender Lehrkräfte für die Sekundarstufe 1 tätig. Sie hat dabei Informatikgrundkonzepte und das Programmieren mit Java, PHP und Python vermittelt. Ihre langjährige Erfahrung in der Aus-und Weiterbildung von Informatiklehrpersonen und viele Musterbeispiele sind in diesen Lehrgang eingeflossen. Sie ist zudem verantwortlich für den Webauftritt dieses Lehrgangs.

Aegidius Plüss war an der Universität Bern Professor für Informatik und deren Didaktik und hat in dieser Tätigkeit viele Informatiklehrkräfte aus- und weitergebildet, die heute aktiv an den Schulen tätig sind. Er gilt als Urgestein in der Informatikausbildung und hat eine grosse Erfahrung mit vielen Programmiersprachen und Computersystemen. Er ist für die textliche Formulierung dieses Lehrgangs und für die didaktischen Libraries in TigerJython verantwortlich.

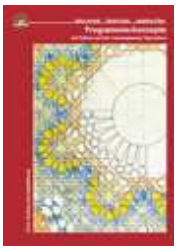
KONTAKT

Die Entwicklergruppe von TigerJython4Kids ist dankbar für jede Art von Rückmeldungen, insbesondere für Fehlermeldungen und Richtigstellungen, Anregungen und Kritik. Wir bieten auch Hilfe und Beratung bei fachlichen oder didaktischen Fragen zu Python und den in TigerJython integrierten Libraries, sowie zur Robotik-Hardware.

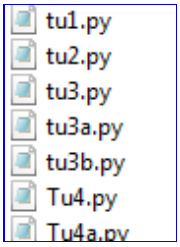
Schreiben Sie ein Email an:

help@tigerjython.com

LINKS



J. Arnold, T. Kohn, A. Plüss: Programmierkonzepte mit Python
Umfangreiches Online-Lehrmittel, mit vielen Beispielen aus verschiedenen Gebieten, geeignet für den Einsatz in weiterführenden Schulen und zum Selbststudium.
www.tigerjython.ch



Programmbeispiele
Sourcecodes aller Programmbeispiele aus *TigerJython4Kids*.
http://www.tigerjython4kids.ch/download/examples.zip

Jarka Arnold: Turtlegrafik, Robotik und Spiele mit Python
Online-Lernprogramm mit vielen lauffähigen Programmbeispielen und Aufgaben für den Einsatz im Unterricht auf der Sekundarstufe 1 und 2.
http://www.jython.ch

Tobias Kohn: Python
Eine Einführung in die Computer-Programmierung.
Ein Skript im PDF-Format.
http://jython.tobiaskohn.ch/PythonScript.pdf (166 Seiten)

Jarka Arnold, Aegidius Plüss: Python exemplarisch
Ergänzende Python-Tutorials: Visualisierung, Raspberry PI und Data Mining
http://www.python-exemplarisch.ch