

Einführung in die Programmierung mit Python

Jürgen Enders



Teachers Teaching with Technology™



Autor:
Jürgen Enders

Dieses und weiteres Material steht Ihnen zum pdf-Download bereit: www.t3deutschland.de sowie unter www.ti-unterrichtsmaterialien.net

Dieses Werk wurde in der Absicht erarbeitet, Lehrerinnen und Lehrern geeignete Materialien für den Unterricht in die Hand zu geben. Die Anfertigung einer notwendigen Anzahl von Fotokopien für den Einsatz in der Klasse, einer Lehrerfortbildung oder einem Seminar ist daher gestattet. Hierbei ist auf das Copyright von T³-Deutschland hinzuweisen. Jede Verwertung in anderen als den genannten oder den gesetzlich zugelassenen Fällen ist ohne schriftliche Genehmigung von T³ nicht zulässig.

T³ Python Workshop

Der T³ Python Workshop vermittelt die Grundlagen der Programmierung in Python. Der Workshop wendet sich an interessierte Lehrkräfte und ist für eine Dauer von mindestens vier Unterrichtsstunden konzipiert. Vorkenntnisse in der Erstellung von Programmen sind nicht erforderlich, aber hilfreich.

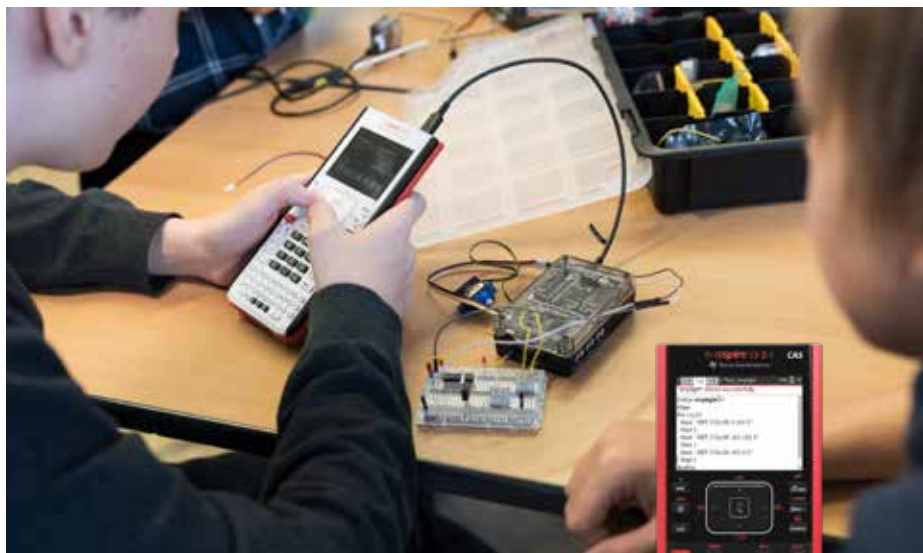
Die hier vorliegenden Aufgaben sind Teil des Workshop-Konzepts. Sie können sie mit Ihren Schülerinnen und Schülern im projektbezogenen Unterricht einsetzen.

Bei Interesse an einem Workshop, durchgeführt vor Ort an Ihrer Schule oder als Online-Seminar, schreiben Sie gerne eine E-Mail an info@t3deutschland.de

Einführung

Der TI-Innovator™ Hub mit TI Launchpad™ Board ist ein mit industriellen Komponenten aufgebautes Interface, das die Signale von Sensoren aufnehmen und Aktoren ansteuern kann. Dazu gibt es viele fertig aufgebaute Module, aber man kann auch eigene Schaltungen auf Steckplatinen (Breadboard) entwerfen und anschließen.

Der TI-Innovator™ Hub funktioniert im Zusammenspiel mit einem TI-Nspire™ CX II-T (CAS) bzw. der entsprechenden Computersoftware, da er auf die Stromversorgung dieser Geräte angewiesen ist. Auf diesen Geräten werden auch die Programme geschrieben, die für den Betrieb des TI-Innovator™ Hub notwendig sind. Die Programmierung erfolgt in TI Basic oder LUA oder - wie in den nachfolgenden Beispielen - in Python. Python ist weit verbreitet und entspricht den Anforderungen der Lehrpläne an eine objektorientierte Programmiersprache.



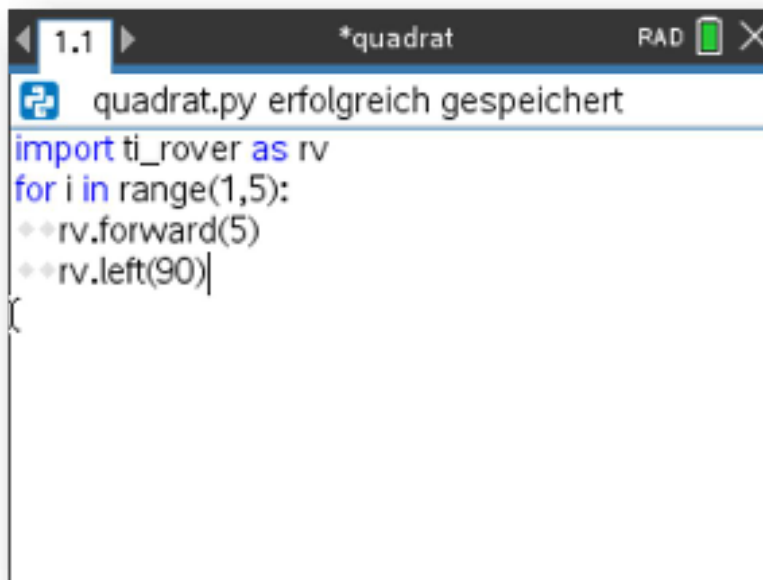
Der TI-Innovator™ Rover

Der Rover ist ein zweiräderiges Fahrzeug mit zwei Fahrmotoren und einer Kugel als dritter Stütze, die ab und zu gereinigt und ev. mit Graphit-Pulver eingestäubt werden muss, damit sie gut gleitet. Der Rover muss zwingend mit dem Innovator verbunden werden, der unter dem Chassis in ein Fach eingeschoben wird, so dass die Ports IN 1 .. IN 3 und OUT 1 .. OUT 3 zugänglich bleiben im Gegensatz zu den Breadboard-Ports und dem I²C-Port, die alle vom Rover selbst benötigt werden. Auf dem Rover ist dann der Platz für den Taschenrechner, der mit dem Innovator verbunden wird und das Programm für den Rover enthält. Der Rover verfügt über einen fest eingebauten Abstandssensor, einen RGB-Lichtsensoren mit weißer LED zur Spurverfolgung, eine RGB-LED und einen gesonderten Akku, mit dem über 6 Stunden Betrieb möglich sind. In dieser Zeit kann der Rover mit den Standardeinstellungen fast 4 km zurücklegen. Selbst wenn die Ladeanzeige nur noch 2 LEDs anzeigt, ist noch ein langer Betrieb ohne Einschränkungen möglich. Steigungen werden bis zu 12° überwunden, aber man benötigt zum Betrieb auf jeden Fall einen glatten Untergrund ohne Fugen. Ferner hat der Rover noch einen Stifthalter. Als Zeichenstifte eignen sich dünne Filzstifte vom Typ non-permanent, um eventuelle Spuren besser beseitigen zu können.

Gesteuert wird der Rover vom Taschenrechner aus über den TI-Innovator Hub. In dessen Menü findet man einen Unterpunkt ROVER, in dem sich alle Befehle befinden. Die Befehle gliedern sich dabei in zwei Gruppen: Alle Befehle, die das Fahren betreffen, werden der Reihe nach abgearbeitet wie gewohnt. Da sie aber mehr Zeit benötigen als die übrigen Befehle wie z.B. das Anschalten der LED, werden diese anderen Befehle parallel dazu ausgeführt. Man muss beim Programmieren also stets die zeitliche Komponente mit berücksichtigen und den korrekten Programmablauf immer wieder überprüfen.



Einfache Programme für den Rover



```
1.1 *quadrat RAD [X]
quadrat.py erfolgreich gespeichert
import ti_rover as rv
for i in range(1,5):
    rv.forward(5)
    rv.left(90)|
{
```

Aufgaben

1. Zwei einfache Programme als Einstieg: Fahren eines Quadrates und eines Kreises
2. Fahren im Stau
3. An einer Markierung anhalten
4. Rückwärts hinter einem Fahrzeug einparken
5. Ein Hindernis umfahren

1. Zwei einfache Programme als Einstieg

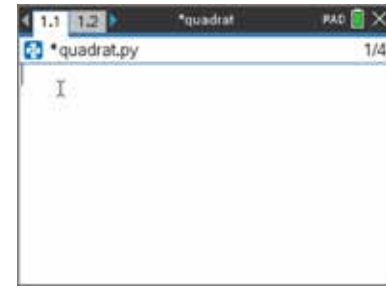
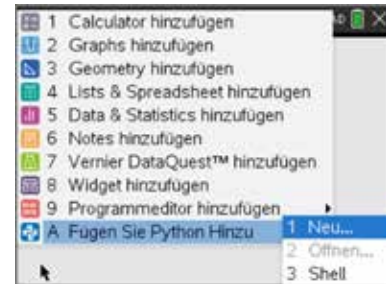
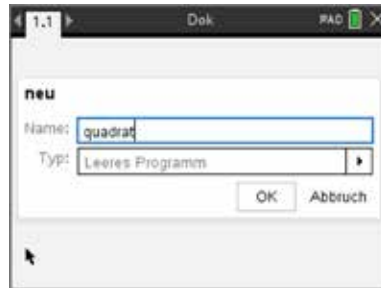
a. Ein Quadrat fahren

Anhand dieses sehr einfachen Beispiels soll die Erstellung eines Python-Programmes auf dem **TI-Nspire™ CX II-T (CAS)** erläutert werden.

Zunächst muss ein neues Dokument geöffnet und darin die App Python hinzugefügt werden. Python besteht aus 2 Teilen, dem Programmierer und der Shell. Mit **1 Neu ...** gelangt man automatisch in den Programmierer.

Hier muss man zunächst einen Namen für das Programm vergeben und mit **OK** bestätigen.

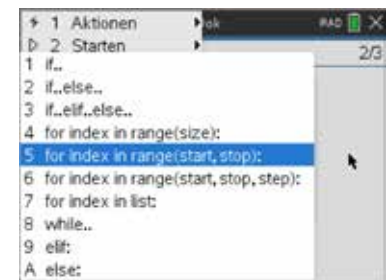
Anschließend öffnet sich der Programmierer mit einem leeren Blatt.



Python ist in seiner Basisversion eine „schlanke“ Sprache. Eine Stärke liegt darin begründet, dass man Module hinzufügen kann. Da im Beispiel der Rover benutzt werden soll, muss das entsprechende Modul über das **Menü** importiert werden. Der Import von Modulen sollte immer am Anfang eines Programmes stehen.



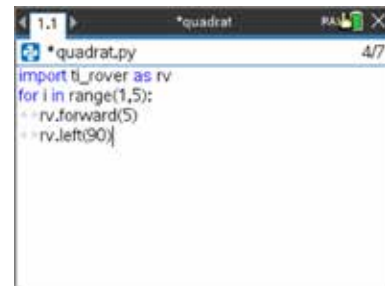
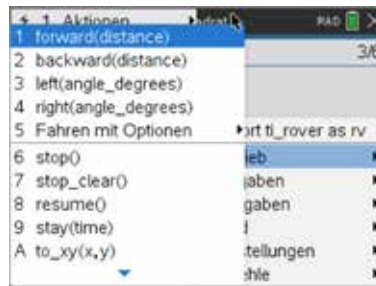
Der Rover soll ein Quadrat fahren. Er muss also viermal eine Seite abfahren und eine Drehung um 90° machen. Dazu nutzt man zweckmäßigerweise eine **For** – Schleife, die man über das **Menü** und **4 Eingebaute** (- Programm-elemente) erreicht.



Python bietet viele **Programmierhilfen** an. In **Grau** sind alle Teile dargestellt, die ausgefüllt werden müssen. Im Beispiel sind das der **Index** (die **Zählvariable**) **i** sowie der Bereich für **i**. Zu beachten ist dabei, dass sobald **i** den Wert **Stopp** erreicht hat, die Anweisungen aus dem **Block** nicht mehr ausgeführt werden. Deshalb muss – trotz der **4 Ecken** – **Stopp** hier den Wert **5** haben.

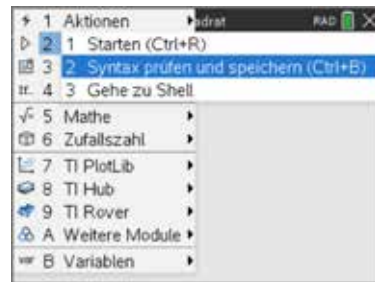


Jetzt kann der **Block** mit den **Fahrbefehlen** gefüllt werden, die man im **Menü** für den Rover unter **Antrieb** findet. Die voreingestellte **Masseinheit** für die Entfernung in **rv.forward()** ist dabei **dm**; das Quadrat hat also eine Seitenlänge von 0,5 m. Python kennt **keine Befehle** wie EndFor oder EndIf. Ein **zusammenhängender Block** von Befehlen wird durch die **gemeinsame Einrückung** um **2 Stellen** gekennzeichnet.

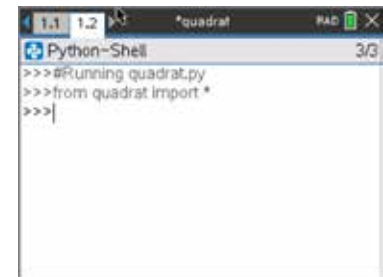


Das Programm ist schon fertig!

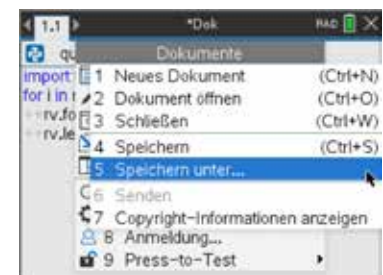
Unbedingt anschließen muss sich nun aus dem **Menü 2 Starten** und dann **2 Syntax prüfen und speichern**, um Fehler zu erkennen und die aktuellste Version des Programmes gesichert zu haben.



Nun kann man das Programm entweder über **1 Starten** oder durch die Tastenkombination **/R** ablaufen lassen. Python wechselt nun in die **Shell**. Die letzte Zeile mit dem blinkenden Cursor zeigt dabei an, dass das Programm fertig ist, obwohl der Rover noch fährt. Das liegt daran, dass die Fahrbefehle im Rover gespeichert und nacheinander abgearbeitet werden, was eine längere Zeit dauert. Das Programm kann mit **/R** auch aus der **Shell** heraus gestartet werden.



Das Programm ist allerdings noch nicht dauerhaft im TI-Nspire™ CX II-T (CAS) gespeichert. Mit der Taste **~** öffnet sich ein Menü, das das Abspeichern ermöglicht.



b. Einen Kreis fahren

Die Drehbefehle für den Rover erzeugen eine Drehung um den Mittelpunkt zwischen den beiden Antriebsrädern, indem die Räder sich entgegengesetzt gleich schnell drehen. Sie eignen sich also nicht für Kreis- oder Bogenfahrten, bei denen die Antriebsräder sich ja unterschiedlich schnell drehen müssen.

Dazu muss man den rechts hervorgehobenen Befehl aus dem Rover-Menü verwenden, der beide Motoren direkt anspricht.

In die Formatvorlage muss für jeden Motor die Drehrichtung eingetragen werden:

cw: clockwise, in Uhrzeigerrichtung

ccw: counterclockwise, in Gegenuhrzeigerrichtung

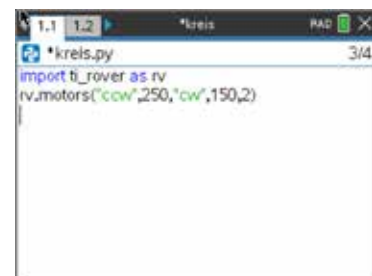
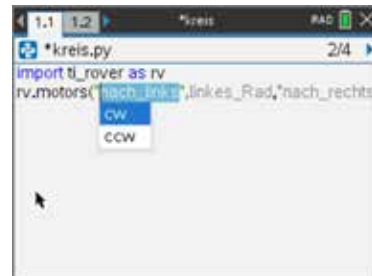
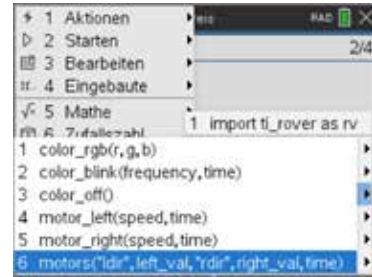
Man muss dabei aufpassen: bedingt durch die Montage der Motoren im Rover, müssen sich die Motoren bei Geradeausfahrt in unterschiedliche Richtungen drehen!

Ferner müssen noch Zahlen aus dem Bereich von 0 bis 255 für die **Drehgeschwindigkeiten** der Motoren eingetragen werden. Als letzter Parameter muss noch ein Wert für die **Laufzeit** der Motoren in **s** eingegeben werden.

So sieht das ganze Programm aus.

- Allerdings fährt der Rover nur einen Bogen – was muss man ändern, damit es ein vollständiger Kreis wird?
- Was muss man ändern, um einen größeren/kleineren Kreis zu bekommen?
- Was ändert sich, wenn der Kreis in die andere Richtung gefahren werden soll?
- Wie sieht das Programm für eine Schlangenlinie aus?

Der Parameter Zeit ist optional. Lässt man ihn weg, so fährt der Rover so lange mit den eingestellten Werten, bis sie geändert werden oder der Befehl **rv.stop()** erreicht wird. Die Laufzeit wird dabei durch den Befehl **sleep()** (in s) eingestellt. Dieser Befehl ist im Modul **ti_hub** enthalten, das also zusätzlich geladen werden muss.

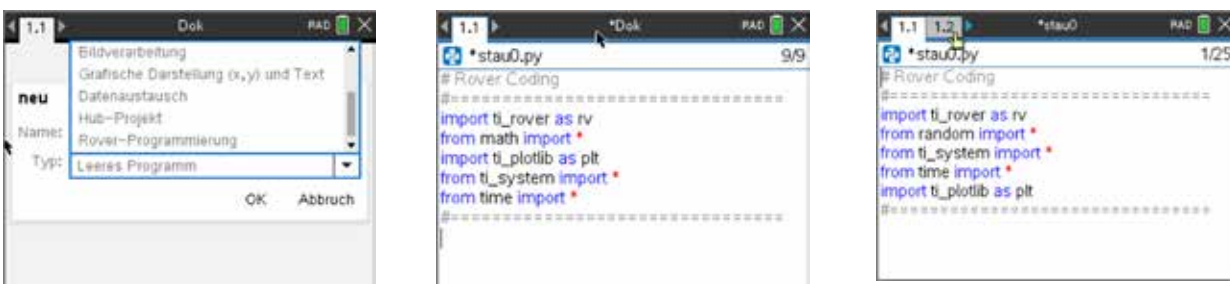


2. Fahren im Stau

Ein Stau kann auf verschiedene Weise entstehen, z.B. wenn man hinter einem Trecker herfahren muss, vor einer roten Ampel oder bei dichtem Verkehr auf der Autobahn. In allen Fällen bedeutet dies ein fortwährendes ermüdendes Abbremsen und Anfahren, das man gerne einem Automaten überlassen würde.

In jedem Fall hat man **ein Fahrzeug vor sich**, an dem man sich orientiert. Dessen Bewegung soll in einem ersten Programm **stau0** simuliert werden. Der Rover fährt dabei zufallsgesteuert 10 cm oder 20 cm vorwärts oder bleibt stehen.

Bei der Anlage dieses Programmes kann man statt **Leeres Programm** auch **Rover Programmierung** wählen (Bild links). Dann werden automatisch die wichtigsten Module geladen (Bild mitte). Im Beispielfall wurde das Modul **from math import** durch das Modul **from random import** ersetzt (Bild rechts), indem das eine Modul gelöscht und das andere eingefügt wurde, denn der Editor funktioniert i.w. wie ein normaler Texteditor. Das Einbinden von Modulen vergrößert den Programmcode.



stau0.py

Module:

ti_rover: Rover-Befehle

random: Zufallszahlen, usw.

ti_system: Tastaturabfragen, usw.

time: sleep, usw.

ti_plotlib: statt der Shell wird der komfortablere Grafikschild genutzt.

Löscht den Grafikschild.

Zeigt den zentrierten Text in den Zeilen 2,4 und 6 an.

Zentrale **while**-Schleife: die Befehle des **Blocks** werden solange ausgeführt, bis die Taste **d** gedrückt wird. **!=** steht in Python für **≠**.

Die Variable **f** enthält nach jedem Durchgang des **Blocks** die **Zufallszahlen 0,1 oder 2**.

f=0 : Rover stoppt.

f=1 : Rover fährt 10 cm.

f=2 : Rover fährt 20 cm vorwärts.

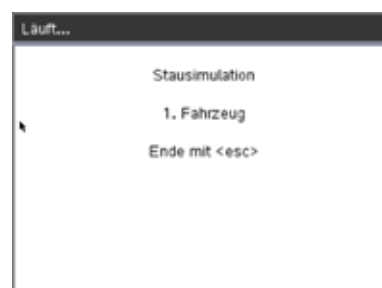
== steht in Python für das **logische =**.

Anhalten der Programmausführung für **2s**, damit Rover Zeit zum Fahren hat.

```
# Rover Coding
#=====
import ti_rover as rv
from random import *
from ti_system import *
from time import *
import ti_plotlib as plt
#=====

plt.cls()
plt.text_at(2,"Stausimulation","center")
plt.text_at(4,"1. Fahrzeug","center")
plt.text_at(6,"Ende mit <esc>","center")
while get_key() != "esc":
    f=randint(0,2)
    if f==0:
        rv.stop()
    if f==1:
        rv.forward(1)
    if f==2:
        rv.forward(2)
    sleep(2)
```

Während der Programmausführung wird der Grafikschild angezeigt. Ist das Programm durch Drücken von **d** beendet, so gelangt man durch **Í** wieder auf die Shell und damit auch wieder in den Editor.



Für alle Fahrzeuge, die **im Stau** stehen, ist das Programm **stau1** gedacht.

Hier wird der im Rover vorne eingebaute **Ultraschall-Abstandssensor** RANGER genutzt. Der Abstand zum vorausfahrenden Fahrzeug wird kontinuierlich gemessen. Unterschreitet der Abstand einen bestimmten Wert, so hält das Fahrzeug an, andernfalls fährt es weiter.

An der Oberfläche des Rovers befindet sich eine **RGB-LED**. Die **drei Farbkanäle** dieser LED (rot grün blau) können einzeln angesteuert werden; dabei sind Werte zwischen 0 und 255 erlaubt. Die Kombination 255 255 255 erzeugt weißes Licht, 0 0 0 schaltet alle Farbkanäle aus. 255 0 0 lässt die LED rot leuchten, 0 255 0 grün und 255 255 0 gelb.

Als Programmtyp wurde wieder **Rover Programming** gewählt; das Modul Time wurde entfernt.

Erläuternder Text im Grafikschild.

Zentrale **while**-Schleife: Programmabbruch mit **d**

Abspeichern der gemessenen Entfernung in der Variablen **d**.

d<0,1m: Rover hält an.

RGB-LED auf dem Rover leuchtet rot.

d>0,1m: Rover fährt wieder an bzw. weiter.

RGB-LED leuchtet grün.

Nach Drücken von **d** wird Rover angehalten.

RGB-LED wird ausgeschaltet.

stau1.py

```
# Rover Coding
#=====
import ti_rover as rv
from math import *
from ti_system import *
import ti_plotlib as plt
#=====
plt.cls()
plt.text_at(2, "Stausimulation", "center")
plt.text_at(4, "Folgefahzeug", "center")
plt.text_at(6, "Ende mit <esc>", "center")
while get_key() != "esc":
    d=rv.ranger_measurement()
    if d<0.1:
        rv.stop()
        rv.color_rgb(255,0,0)
    else:
        rv.forward()
        rv.color_rgb(0,255,0)
rv.stop()
rv.color_rgb(0,0,0)
```

Stehen viele Rover zur Verfügung, kann man einen längeren „Stau“ damit simulieren. Nur das Fahrzeug an der Spitze erhält dann das Programm **stau0**. Dort wird auch das Programm als letztes gestartet. Dann kann man beobachten, wie sich die Bewegung der Folgefahrzeuge im Stau entwickelt.

3. An einer Markierung anhalten

Sehr oft befinden sich auf der Fahrbahn dicke weiße Streifen, die eine Haltelinie darstellen. Der Rover hat vorne am Fahrzeugboden einen **Farbsensor** und eine **weiße LED**, mit der der Boden permanent beleuchtet wird. Durch das vom Untergrund reflektierte Licht ist der Farbsensor in der Lage, die Farbe des Untergrundes zu erkennen. Mithin kann also auch eine weiße **Haltelinie** erkannt werden.

Der Untergrund sollte deutlich dunkel sein, damit die Haltelinie sicher erkannt werden kann. Als Haltelinie kann einfach ein weißes Blatt Papier genommen werden, auf das der Rover zufährt. Das Anhalten wird durch die rot leuchtende **RGB-LED** signalisiert. Zusätzlich wechselt auf dem Grafikschild ein grüner Kreis zu rot. Dazu muss das Modul **ti_draw** geladen werden.

Das Programm wird mit der Taste **d** beendet.

Geladene Module.

Zusätzliches Modul **ti_draw**.

Grafikschirm mit Überschrift.

Befehle aus **ti_draw**, erzeugen einen grünen Kreis mit **M(160/100)** und **r=20**.

Rover fährt los.

Grauwert der Farbe des Untergrundes wird fortlaufend gemessen.

g>100: heller Untergrund (also weiß).

Rover hält an, RGB-LED leuchtet rot.

Farbwechsel des Kreises zu rot.

RGB-LED ausschalten.

haltelinie.py

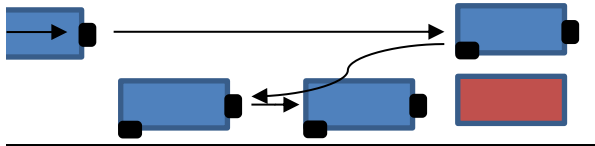
```
# Rover Coding
#=====
import ti_rover as rv
from math import *
import ti_plotlib as plt
from ti_system import *
from ti_draw import *
#=====
plt.cls()
plt.text_at(2, "Haltelinie", "center")
plt.text_at(4, "Ende mit
<esc>", "center")
set_color(0,255,0)
fill_circle(160,100,20)
rv.motors("ccw",150,"cw",150)
while get_key() != "esc":
    g=rv.gray_measurement()
    if g>100:
        rv.stop()
        rv.color_rgb(255,0,0)
        set_color(255,0,0)
        fill_circle(160,100,20)
rv.color_rgb(0,0,0)
```

4. Rückwärts hinter einem Fahrzeug einparken

Das ist der Glücksfall für jemanden, der einen **Parkplatz am Straßenrand** sucht, denn hierbei spielt die Breite der Parklücke keine Rolle. Das **Fahrmanöver** sieht dann folgendermaßen aus:

1. Neben dem Fahrzeug anhalten, hinter dem man einparken will
2. Blinken
3. Eine S-Kurve rückwärts an den Straßenrand fahren
4. Abstand zum Vordermann korrigieren

Um zu erkennen, ob man für das Einparken richtig steht, muss man das Heck des Vordermannes detekieren. Dazu dient ein weiterer **Ultraschall-Bewegungssensor**, der mit dem Eingang **IN1** des **TI-Innovator™ Hub** verbunden und hinten rechts seitlich am Rover befestigt wird. Es reicht schon, wenn man das Verbindungskabel entsprechend knickt. Als parkendes Fahrzeug kann ein kleiner Karton oder ein Bücherstapel dienen.



Zusätzliches Modul **ti_hub**.

Überschrift.

Der Ultraschall-Bewegungssensor RANGER des Innovator™-Hubs wird als Objekt der Variablen **d** zugewiesen.

Rover fährt langsam los; der seitliche Abstand **d1** wird fortlaufend gemessen.

d1 < 0,1m: Rover stoppt.

Für **1s** leuchtet die RGB-LED rot („Stopplicht“).
Die RGB-LED blinkt gelb („Blinker“).

Rover fährt für **1s** rückwärts, dann eine S-Kurve.
Um diese Manöver ausführen zu können, wird das Programm **4,6s** angehalten.

Blinker aus.

Rover fährt geradeaus, bis der RANGER des Rovers **d2 < 0,1m** misst.

Rover stoppt.

RGB-LED leuchtet für **1s** rot („Bremslicht“).
RGB-LED wird ausgeschaltet.

einparken.py

```
# Rover Coding
#=====
import ti_rover as rv
from math import *
import ti_plotlib as plt
from ti_system import *
from time import *
from ti_hub import *
#=====
plt.cls()
plt.text_at(2, "Einparken", "center")
d=ranger("IN 1")
rv.motors("ccw",150,"cw",150)
d1=d.measurement()
while d1>0.1:
    d1=d.measurement()
rv.stop()
rv.color_rgb(255,0,0)
sleep(1)
rv.color_rgb(255,255,0)
rv.color_blink(2,20)
rv.motors("cw",150,"ccw",150,1)
rv.motors("cw",180,"ccw",50,1.8)
rv.motors("cw",50,"ccw",180,1.8)
sleep(4.6)
rv.color_off()
d2=rv.ranger_measurement()
while d2>0.1:
    rv.motors("ccw",150,"cw",150)
    d2=rv.ranger_measurement()
rv.stop()
rv.color_rgb(255,0,0)
sleep(1)
rv.color_off()
```

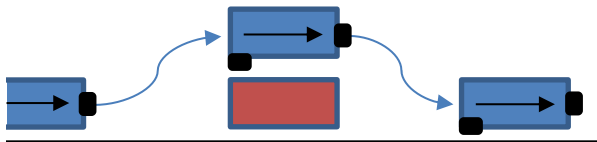
5. Ein Hindernis umfahren

Eine Standardsituation: da hält jemand in zweiter Reihe, ein Hindernis ragt in den Fahrweg oder ein langsames Fahrzeug verhindert ein schnelleres Vorankommen. In jedem dieser Fälle ist das **Fahrmanöver** gleich:

1. Langsam an das Hindernis herankommen.
2. Übrigen Verkehr die ganze Zeit beobachten.
3. Blinken.
4. Eine S-Kurve fahren.
5. Das Hindernis zügig überholen.
6. Blinken.
7. Mit einer weiteren S-Kurve vor dem Hindernis wieder auf den ursprünglichen Weg kommen.
8. Schnell weiterfahren.

Bis auf die Beobachtung des Verkehrs kann ein solches Fahrmanöver mit dem Rover nachvollzogen werden.

Um zu erkennen, wann der Überholvorgang beendet ist, muss man das Hindernis detektieren. Dazu dient wieder ein weiterer **Ultraschall-Bewegungssensor**, der mit dem Eingang **IN1** des **TI-Innovator™ Hub** verbunden und hinten rechts seitlich am Rover befestigt wird. Als Hindernis reichen ein kleiner Karton oder ein Bücherstapel. Das Modul **ti_hub** muss zusätzlich geladen werden.



Zusätzliches Modul **ti_hub**.

Der Ultraschall-Bewegungssensor des Innovators™ wird als **Objekt** der Variablen **d** zugewiesen.

d1 ist die Entfernung zum Hindernis;
solange **d1 > 0,5m** fährt Rover mit großer
Geschwindigkeit.

Solange **0,5m > d1 > 0,3m** fährt Rover mit verringerter
Geschwindigkeit und die „Bremsleuchte“ leuchtet.

Die RGB-LED **blinkt** jetzt **gelb**.

S-Kurve links-rechts **je 2s**.

Geradausfahrt am Hindernis vorbei für **2s**, damit
der Sensor am Heck das Hindernis erfassen kann;
für dieses Manöver die Programmausführung **5,6s**
anhalten.

Messwert für den Hecksensor in **d2** speichern und
langsam am Hindernis vorbeifahren, solange
d2 < 0,1m.

S-Kurve in der entgegengesetzten Richtung fahren.
„Blinker“ ausschalten.

Schnelle Geradausfahrt für **2s** am Manöverende.
Programmausführung wieder für **5,6s**
anhalten.
Rover anhalten.

ausweichen.py

```
# Rover Coding
#=====
import ti_rover as rv
from math import *
import ti_plotlib as plt
from ti_system import *
from time import *
from ti_hub import *
#=====
plt.cls()
plt.text_at(2, "Ausweichen /
Überholen", "center")
d=ranger("IN 1")
d1=rv.ranger_measurement()
while d1>0.5:
    rv.motors("ccw",255,"cw",255)
    d1=rv.ranger_measurement()
while d1>0.3:
    rv.motors("ccw",150,"cw",150)
    d1=rv.ranger_measurement()
    rv.color_rgb(255,0,0)
    rv.color_rgb(255,255,0)
    rv.color_blink(2,20)
    rv.motors("ccw",50,"cw",180,1.8)
    rv.motors("ccw",180,"cw",50,1.8)
    rv.motors("ccw",150,"cw",150,2)
    sleep(5.6)
    d2=d.measurement()
while d2<0.1:
    rv.motors("ccw",150,"cw",150)
    d2=d.measurement()
    rv.motors("ccw",180,"cw",50,1.8)
    rv.motors("ccw",50,"cw",180,1.8)
    rv.color_off()
    rv.motors("ccw",255,"cw",255,2)
    sleep(5.6)
rv.stop()
```




www.t3deutschland.de

education.ti.com



Teachers Teaching with Technology™

 TEXAS INSTRUMENTS