

Inhaltsverzeichnis

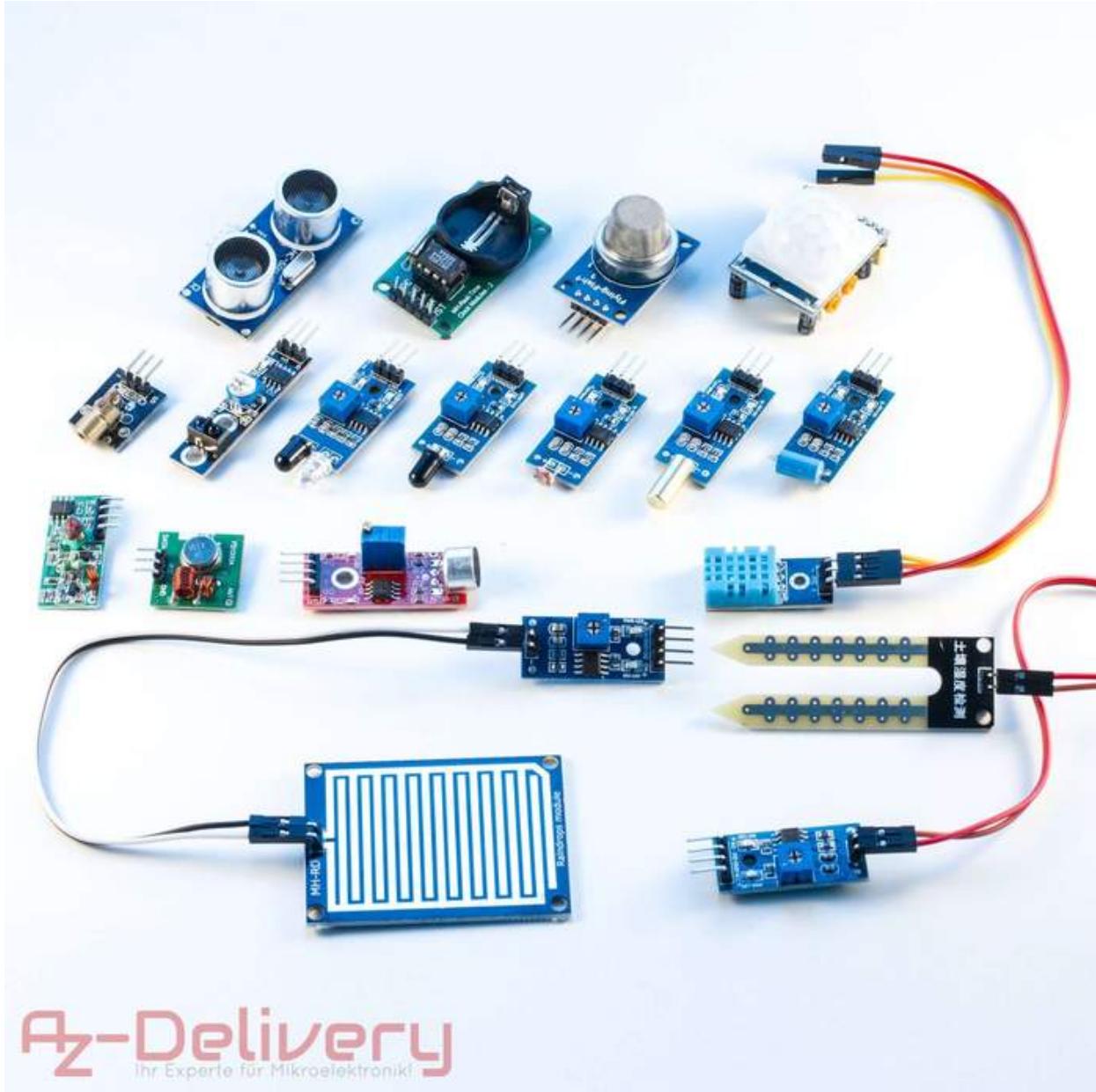
Inhaltsverzeichnis	0
1 Einführung	1
1.1 GPIO Belegung	2
2.1 433MHz Funkmodule (Sender + Empfänger)	3
2.2 DHT11 Temperatur & Luftfeuchtigkeit	7
2.3 Lasерmodul	8
2.4 Bodenfeuchtigkeit	9
2.5 Regensensor	12
2.6 Bewegungsmelder (PIR = Passive InfraRed)	15
2.7 Lichtabhängiger Widerstand LDR	17
2.8 Erschütterungssensor	18
2.9 Lagesensor	19
2.10 Klatschsensor	20
2.11 Näherungssensor	21
2.12 Flammensensor	22
2.13 Näherungssensor 2	23
2.14 Ultraschallsensor HC-SR04	24
2.15 MQ-2 Gassensor	27
2.16 Realtime-Clock	28



1 Einführung

Danke das du dich für das 16 in 1 Zubehörset für Raspberry Pi aus unseren Shop entschieden hast. Mit dieser Anleitung wollen wir dir die einzelnen Elemente näherbringen.

Wie du das Betriebssystem installierst und den Raspberry vorbereitest, kannst du in dem entsprechenden eBook über den Raspberry Pi nachlesen.



1.1 GPIO Belegung

Raspberry Pi 3 GPIO Header			
Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi



Sicherheitshinweise

bei dem Umgang mit den GPIO:



- Kurzschlüsse und falsche Beschaltung von Pins können den Raspberry Pi beschädigen!
- Raspberry Pi ausschalten, wenn am Steckboard die Schaltung geändert wird
- Die maximale Spannung an den GPIO-Pins beträgt 3.3 Volt. Niemals 5 Volt mit GPIO verbinden

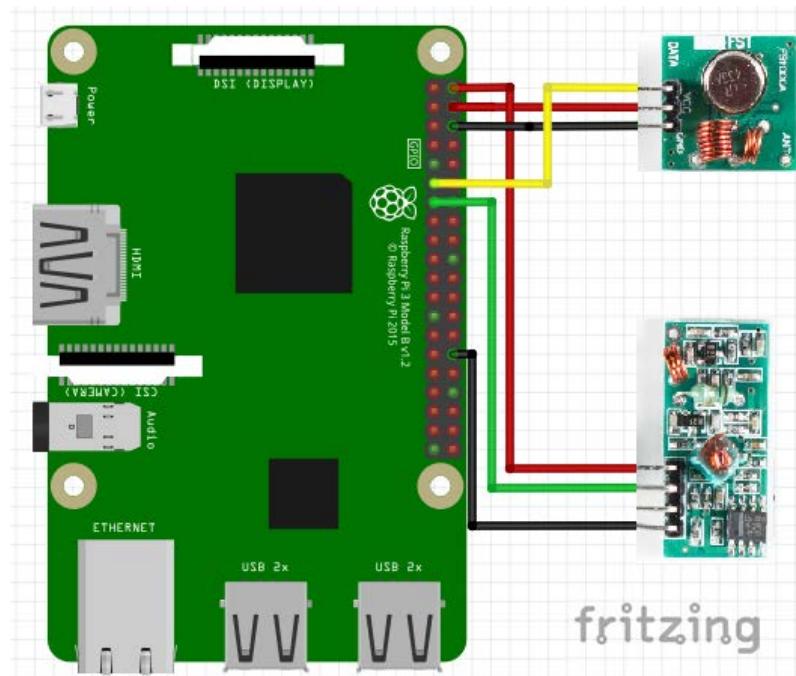
2.1 433MHz Funkmodule (Sender + Empfänger)



AZ-Delivery
Ihr Experte für Mikroelektronik!

Das Sender (XY-FST) und Empfänger (XY-MK) Funkmodulset kommuniziert auf 433,92 MHz. Mit den Modulen können beispielsweise Funksteckdosen gesteuert werden, dies werden die hier ebook erklären.

Verdrahten der Module mit dem Raspberry Pi:



Die-Module haben jeweils nur 3 Anschlüsse, VCC, GND und Data.

Bei dem Empfangsmodul sind 4 Pins angebracht, wobei die 2 mittleren (DATA) gebrückt sind.

Sendemodul (3 Pin):

VCC wird mit **PIN 2 (5V)** am Raspberry verbunden
GND wird mit **PIN 6 (GND)** verbunden
DATA wird mit **PIN 11 (GPIO 17)** verbunden

Rote Leitung
Schwarze Leitung
Gelbe Leitung

Empfangsmodul (4 Pin):

VCC wird mit **PIN 4 (5V)** am Raspberry verbunden
GND wird mit **PIN 30 (GND)** verbunden
DATA wird mit **PIN 13 (GPIO 27)** verbunden

Rote Leitung
Schwarze Leitung
Grüne Leitung

Antennen

Zu Beginn sollten wir an jedes Modul noch eine Antenne anbringen.
Berechnen wir diese kurz:

Das Modul sendet auf **433,92 MHz**.

Die Wellenlänge λ berechnet sich aus
$$\frac{\text{Lichtgeschwindigkeit}}{\text{Frequenz}}$$

$$\lambda = \frac{299.792.458 \text{ m/s}}{433.920.000 \text{ 1/s}} = 0,69 \text{ m}$$

Die Antenne sollte $\frac{1}{4} \lambda$ betragen, also $69\text{cm} / 4 \Rightarrow 17,25 \text{ cm}$

Also schließen wir am Antennenkontakt je einen Draht mit ca. 17 cm Länge an.

Nachdem alles verdrahtet ist kann der Raspberry Pi gestartet werden.

Zur Information: Diese Anleitung basiert auf dem Raspberry Pi Image vom 29.11.2017 (Stretch - Lite) – Updates können leichte Modifikationen der Anleitung notwendig machen.

Alternativ zu den von uns beschriebenen Pins am Raspberry kann jeder Masse-Pin verwendet werden, ebenso auch andere GPIO-Pins. Sollten die GPIO-Pins verändert werden, muss die Beispielsoftware angepasst und neu Kompiliert werden.

„Programmieren“ des Raspberry Pi:

Bevor man auf dem Raspberry Pi Software installiert, sollte der Raspberry Pi noch auf den aktuellsten Stand gebracht werden:

```
sudo apt-get update  
sudo apt-get upgrade
```

Do you want to continue? [Y/n] → y (Y eingeben und mit Enter bestätigen)

Nachdem der Raspberry Pi nun aktuell ist können wir Software installieren.

```
sudo apt-get install git-core
```

git-core: Software zum Download von Software aus GIT

Do you want to continue? [Y/n] → y

wenn git fertig installiert ist,
laden (git clone) wir uns von git die Bibliothek von Ninjablocks und wiringPi.
Anschließend kompilieren wir die Pakete (./build oder make all) diese:

```
git clone git://git.drogon.net/wiringPi  
cd ~/wiringPi  
./build  
cd ~  
  
git clone --recursive https://github.com/ninjablocks/433Utils.git  
cd ~/433Utils/RPi_utils  
make all
```

Nach dem Kompilieren sind wir in dem Ordner: 433Utils/RPi_utils

```
pi@raspberrypi:~/433Utils/RPi_utils $
```

Sollte das bei dir nicht der Fall sein gebe folgenden Befehl ein:

```
cd ~/433Utils/RPi_utils
```

Testen wir als erstes den Empfänger und lesen den Code unserer Fernbedienung aus:

```
sudo ./RFSniffer
```

Der Raspberry sollte nun beim Einschalten und ausschalten der Funksteckdose etwas empfangen:

```
Received 263505  
Received 263508
```

Az-Delivery

Bei mir hat der Raspberry 263505 beim Einschalten und 263508 beim Ausschalten empfangen.

Mit der Tastenkombination STRG + C beenden wir den RFSniffer.

Um eine Steckdose auch schalten zu können, gibt es das Programm codesend. Diesem geben wir unseren Einschaltcode beim starten mit und dieses sendet dann diesen Code auf 433,92 MHz aus.

```
sudo ./codesend 263505
```

Die Funksteckdose hat sich nun eingeschalten.

Und ausschalten geht so:

```
sudo ./codesend 263508
```

Wunderbar, jetzt können wir unsere Funksteckdose steuern. Aber was machen wir wenn wir keine Fernbedienung zur Hand haben, die wir auslesen können?



den Sendebefehl send. Diesem Befehl geben den eingestellten Systemcode (die ersten 5 Schalter auf der Steckdose), gefolgt von der Steckdosensummer (1, 2, 3, ...) und 0 (AUS) oder 1 (EIN) an.

O	O	O	O	O	O	O	O	O	O
N	N	N	N	N	N	N	N	N	N
1	2	3	4	5	A	B	C	D	E
		F			F	F	F	F	F
		F			F	F	F	F	F

Dafür
gibt es
wir

In meinem Fall ist das

```
sudo ./send 11011 1 0
```

 AUS

```
sudo ./send 11011 1 1
```

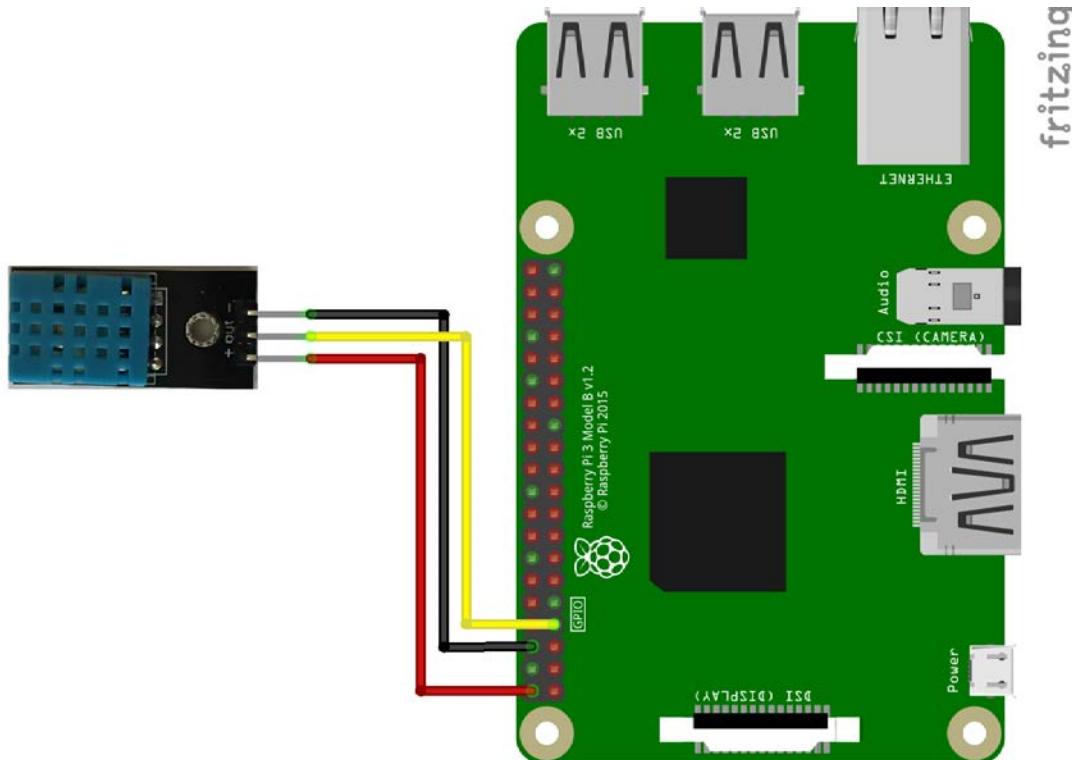
 EIN

2.2 DHT11 Temperatur & Luftfeuchtigkeit

Nun verwenden wir den Temperatur- und Luftfeuchtigkeitssensor DHT11.

Dieser wird mit 1-Wire angesteuert. Das 1-Wire Protokoll wird am Raspberry nur am GPIO4 unterstützt. Deswegen muss der 1-Wire-Anschluss des Temperatursensors (mittlere Anschluss) hier angeschlossen werden.

Bauen wir einmal die Schaltung auf.



Wenn alles entsprechend verkabelt ist, können wir die benötigten Bibliotheken installieren:

```
sudo apt-get install build-essential python-dev python-openssl git  
git clone https://github.com/adafruit/Adafruit_Python_DHT.git && cd Adafruit_Python_DHT  
sudo python setup.py install
```

Hat alles geklappt können wir bereits die Temperatur und Luftfeuchtigkeit auslesen. Am Einfachsten ist das erst einmal mit den Demofiles:

```
cd examples
```

```
sudo ./AdafruitDHT.py 11 4
```

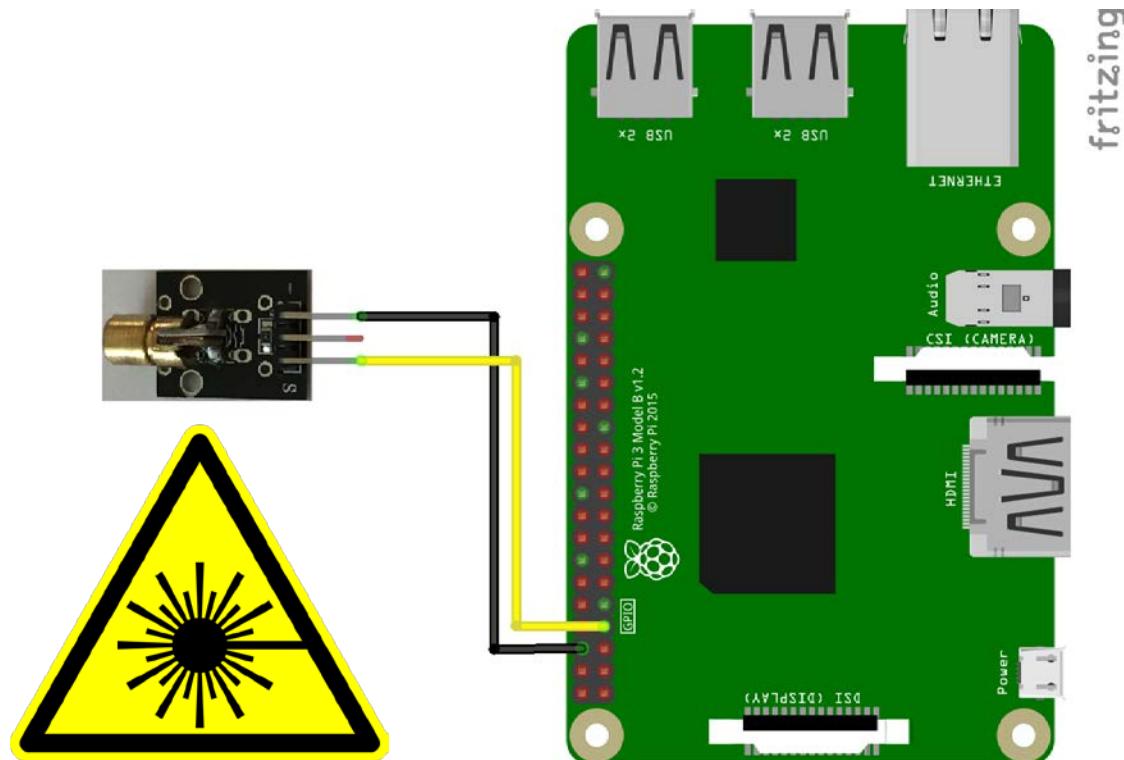
Der erste Parameter (11) gibt an, welcher Sensor benutzt wurde (22 für den DHT22) und der zweite, an welchem GPIO er angeschlossen ist (Nicht die Pin Nummer, sondern die GPIO Nummer). Wir bekommen folgende Ausgabe:

Temp=24.0* Humidity=41.0%

Achtung: Die Sensoren sind nur alle zwei Sekunden bereit. Eine Abfrage des Sensors dauert ca. 2 Sekunden, in dieser Zeit darf kein 2. Mal der Sensor abgefragt werden.

2.3 Lasermodul

Das Lasermodul wird direkt an einen GPIO angeschlossen. Mit dem einschalten des entsprechenden GPIO wird der Laser eingeschalten.



Für die Ansteuerung des GPIO brauchen wir keine besondere Software, wir nehmen einfach das vorhandene System:

```
sudo echo "4" > /sys/class/gpio/export  
sudo echo "out" > /sys/class/gpio/gpio4/direction  
sudo chmod 666 /sys/class/gpio/gpio4/value  
sudo chmod 666 /sys/class/gpio/gpio4/direction  
echo "1" > /sys/class/gpio/gpio4/value
```

echo "4" > xx	schreibe eine 4 in die Datei xx (initialisieren des GPIO 4)
echo "out" > xx	schreibe out in die Datei xx (GPIO 4 als Ausgang definieren)
chmod	Berechtigungen ändern
echo "1" >	schreibe eine 1 in die Datei xx (GPIO 4 einschalten)

Mit dem letzten Befehl (echo "1" > /sys/class/gpio/gpio4/value) schreiben wir in die Datei den Ausgangswert unseres GPIO4. Schreiben wir eine 0, so schalten wir den Pin (Laser) aus.

```
echo "0" > /sys/class/gpio/gpio4/value
```

Hinweis: Nach einem Neustart des Raspberry Pi muss die Initialisierung erneut durchgeführt werden!

2.4 Bodenfeuchtigkeit

Das Bodenfeuchtigkeitsmodul liefert einen digitalen Ausgang und einen analogen Wert. Leider hat der Raspberry Pi nur digitale Eingänge, somit können wir nur diesen verwenden.

Der digitale Ausgang sendet ein Signal, sobald ein Schwellwert übertreten wird, allerdings ist der Wert so nicht exakt bestimmbar. Den Schwellwert kann man durch ein Potentiometer verändern (falls der Wert überschritten wurde, leuchtet das grüne Lämpchen).

Zum Auslesen des GPIO verwenden wir die GPIO Bibliothek wiringPi.

```
sudo apt-get -y install git-core  
cd ~  
git clone git://git.drogon.net/wiringPi  
cd wiringPi  
./build
```

Nach der Installation von wiringPi können wir das gleich testen:

```
gpio -v
```

```
gpio readall
```

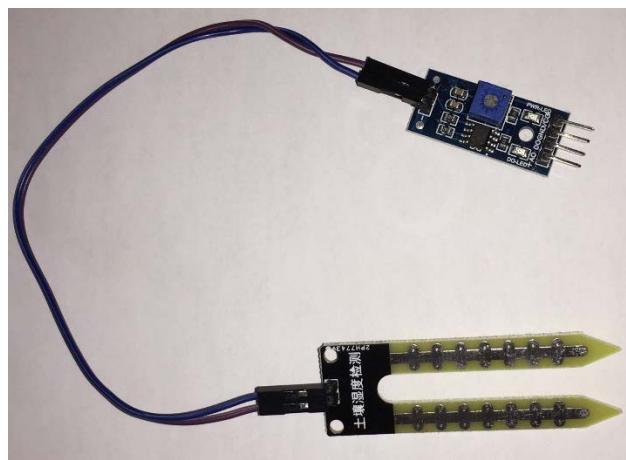
Wir bekommen folgende Ausgabe:

```
pi@raspberrypi3p:~/wiringPi $ gpio readall
```

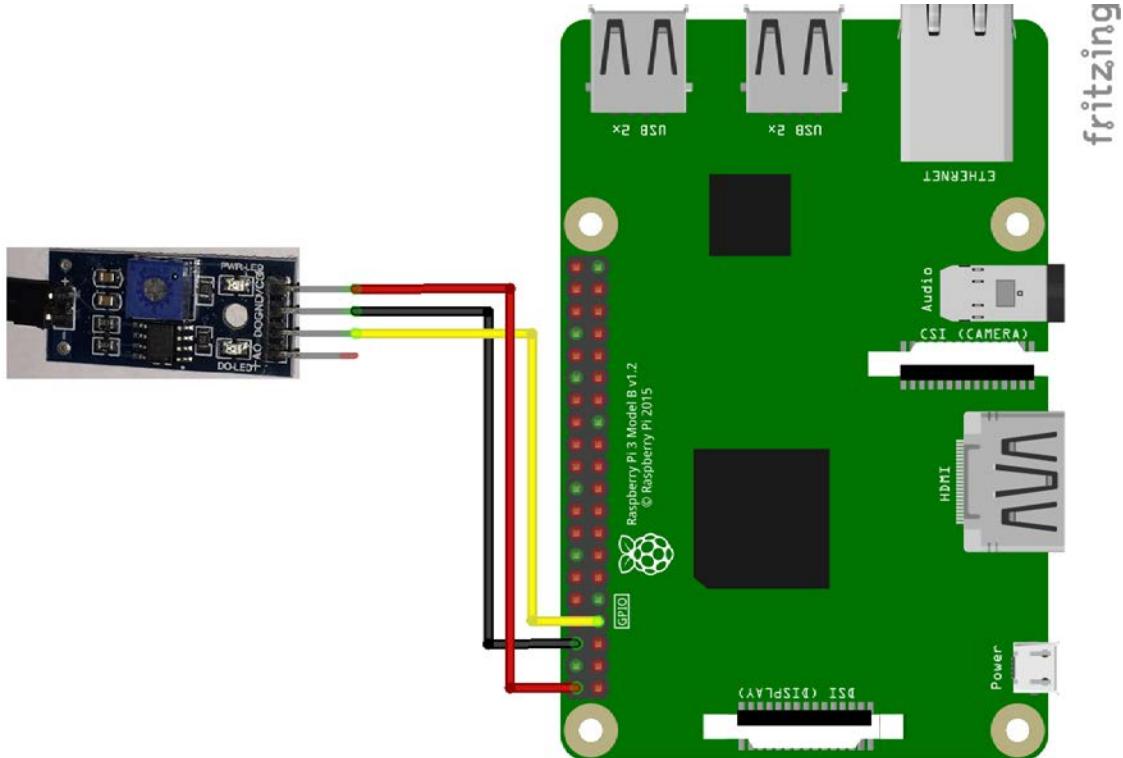
Pi 3												
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM		
		3.3v			1 2			5v				
2	8	SDA.1	IN	1	3 4			5v				
3	9	SCL.1	IN	1	5 6			0v				
4	7	GPIO. 7	IN	1	7 8	0	IN	TxD	15	14		
		0v			9 10	1	IN	RxD	16	15		
17	0	GPIO. 0	IN	0	11 12	0	IN	GPIO. 1	1	18		
27	2	GPIO. 2	IN	0	13 14			0v				
22	3	GPIO. 3	IN	0	15 16	0	IN	GPIO. 4	4	23		
		3.3v			17 18	0	IN	GPIO. 5	5	24		
10	12	MOSI	IN	0	19 20			0v				
9	13	MISO	IN	0	21 22	0	IN	GPIO. 6	6	25		
11	14	SCLK	IN	0	23 24	1	IN	CE0	10	8		
		0v			25 26	1	IN	CE1	11	7		
0	30	SDA.0	IN	1	27 28	1	IN	SCL.0	31	1		
5	21	GPIO.21	IN	1	29 30			0v				
6	22	GPIO.22	IN	1	31 32	0	IN	GPIO.26	26	12		
13	23	GPIO.23	IN	0	33 34			0v				
19	24	GPIO.24	IN	0	35 36	0	IN	GPIO.27	27	16		
26	25	GPIO.25	IN	0	37 38	0	IN	GPIO.28	28	20		
		0v			39 40	0	IN	GPIO.29	29	21		

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
Pi 3										

Nun können wir den Feuchtigkeitssensor an die Sensorplatine stecken:



Und mit dem Raspberry verbinden:



Nun programmieren wir die ein paar Zeilen zur Abfrage:

```
cd ~  
touch feuchte.sh  
nano feuchte.sh  
  
#!/bin/sh  
gpio -g mode 4 in  
while true  
do  
result=$( gpio -g read 4 )  
if [ "$result" = "0" ]; then  
echo "Boden ist Feucht"  
fi  
if [ "$result" = "1" ]; then  
echo "Boden ist Trocken"  
fi  
sleep 2  
done  
  
bash /home/pi/feuchte.sh
```

Az-Delivery

Wir bekommen jetzt alle 2 Sekunden eine Ausgabe ob der Boden Trockener oder Feuchter ist als der eingestellte Schwellwert.

```
pi@raspberrypi3p:~ $ bash /home/pi/feuchte.sh
Boden ist Trocken
Boden ist Trocken
Boden ist Trocken
Boden ist Trocken
Boden ist Feucht
Boden ist Feucht
Boden ist Trocken
Boden ist Trocken
```

Mit STRG + C wird das Programm wieder beendet.

2.5 Regensor

Der Regensor liefert einen digitalen Ausgang und einen analogen Wert. Leider hat der Raspberry Pi nur digitale Eingänge, somit können wir nur diesen verwenden.

Der digitale Ausgang sendet ein Signal, sobald ein Schwellwert übertreten wird, allerdings ist der Wert so nicht exakt bestimmbar. Den Schwellwert kann man durch das Potentiometer verändern (falls er erreicht wurde, leuchtet das grüne Lämpchen).

Zum Auslesen des GPIO verwenden wir die GPIO Bibliothek wiringPi.

```
sudo apt-get -y install git-core
cd ~
git clone git://git.drogon.net/wiringPi
cd wiringPi
./build
```

Nach der Installation von wiringPi können wir das gleich testen:

```
gpio -v
gpio readall
```

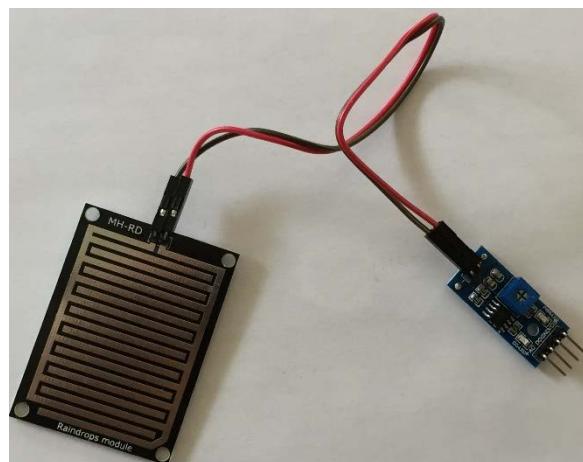
Wir bekommen folgende Ausgabe:

```
pi@raspberrypi3p:~/wiringPi $ gpio readall
```

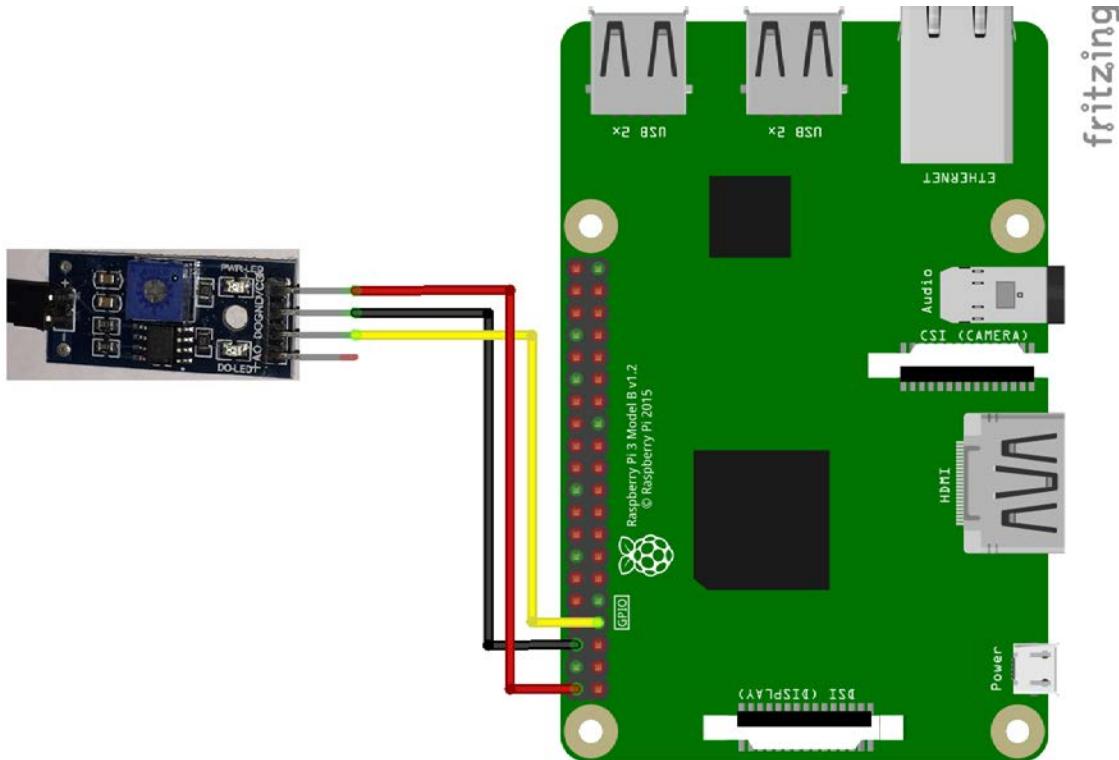
Pi 3+-----Pi 3-----+												
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM		
		3.3v			1 2			5v				
2	8	SDA.1	IN	1	3 4			5v				
3	9	SCL.1	IN	1	5 6			0v				
4	7	GPIO. 7	IN	1	7 8	0	IN	TxD	15	14		
		0v			9 10	1	IN	RxD	16	15		
17	0	GPIO. 0	IN	0	11 12	0	IN	GPIO. 1	1	18		
27	2	GPIO. 2	IN	0	13 14			0v				
22	3	GPIO. 3	IN	0	15 16	0	IN	GPIO. 4	4	23		
		3.3v			17 18	0	IN	GPIO. 5	5	24		
10	12	MOSI	IN	0	19 20			0v				
9	13	MISO	IN	0	21 22	0	IN	GPIO. 6	6	25		
11	14	SCLK	IN	0	23 24	1	IN	CE0	10	8		
		0v			25 26	1	IN	CE1	11	7		
0	30	SDA.0	IN	1	27 28	1	IN	SCL.0	31	1		
5	21	GPIO.21	IN	1	29 30			0v				
6	22	GPIO.22	IN	1	31 32	0	IN	GPIO.26	26	12		
13	23	GPIO.23	IN	0	33 34			0v				
19	24	GPIO.24	IN	0	35 36	0	IN	GPIO.27	27	16		
26	25	GPIO.25	IN	0	37 38	0	IN	GPIO.28	28	20		
		0v			39 40	0	IN	GPIO.29	29	21		

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
Pi 3+-----Pi 3-----+										

Nun können wir den Regensensor an die Sensorplatine stecken:



Und mit dem Raspberry verbinden:



Nun programmieren wir die ein paar Zeilen zur Abfrage:

```
cd ~  
touch regen.sh  
nano regen.sh  
  
#!/bin/sh  
gpio -g mode 4 in  
while true  
do  
result=$( gpio -g read 4 )  
if [ "$result" = "0" ]; then  
echo "Es regnet"  
fi  
if [ "$result" = "1" ]; then  
echo "Kein Regen"  
fi  
sleep 2  
done  
  
bash /home/pi/regen.sh
```

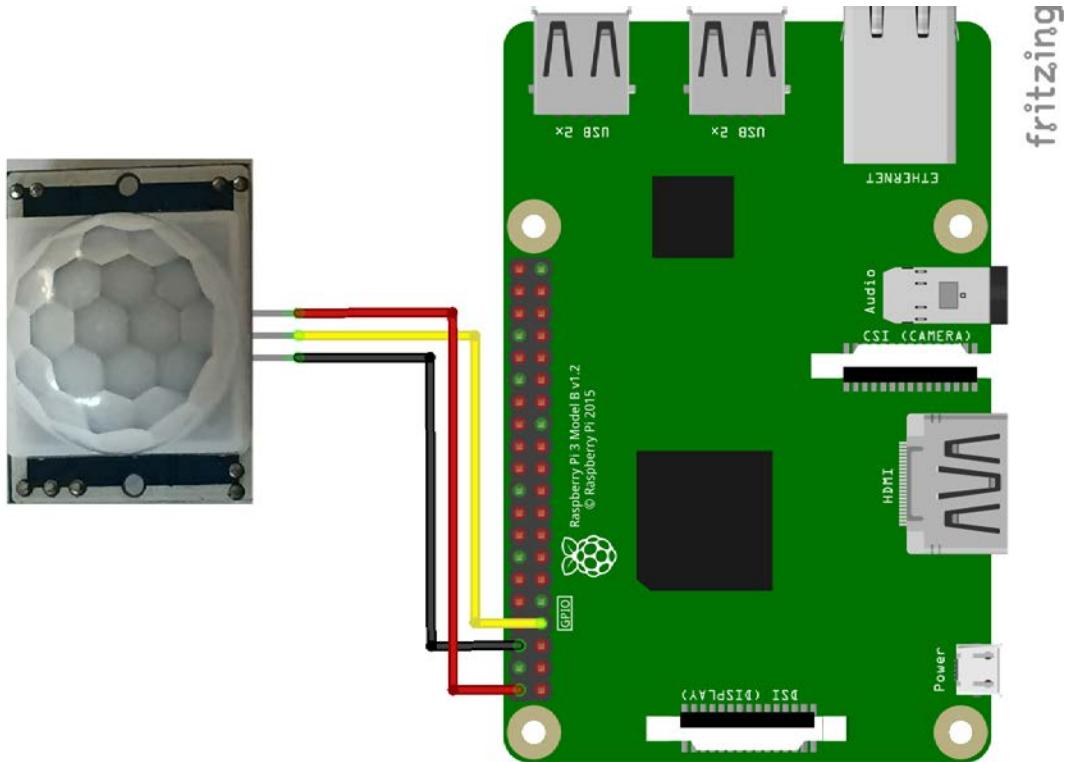
Wir bekommen jetzt alle 2 Sekunden eine Ausgabe ob es regnet oder nicht, bzw. ob der eingestellte Schwellwert erreicht wurde oder nicht.

```
pi@raspberrypi3p:~ $ bash /home/pi/regen.sh
Kein Regen
Kein Regen
Es regenet
Es regenet
Kein Regen
Kein Regen
```

Mit STRG + C wird das Programm wieder beendet.

2.6 Bewegungsmelder (PIR = Passive InfraRed)

Der Bewegungsmelder liefert ein Signal, wenn sich das erfasste Infrarotfeld ändert.



Diesen Sensor können wir auch mit einem Shell Programm abfragen, aber dieses Mal zeigen wir euch das in einem Python Programm.

Az-Delivery

```
cd ~
```

```
touch pir.py
```

```
nano pir.py
```

```
import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

def bewegung(channel):
    print('Es gab eine Bewegung!')

try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=bewegung)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()
```

```
python pir.py
```

Jedes Mal wenn der Bewegungsmelder etwas erkennt wird bei dem Python Programm ein Interrupt ausgeführt und „Es gabe eine Bewegung!“ ausgegeben.

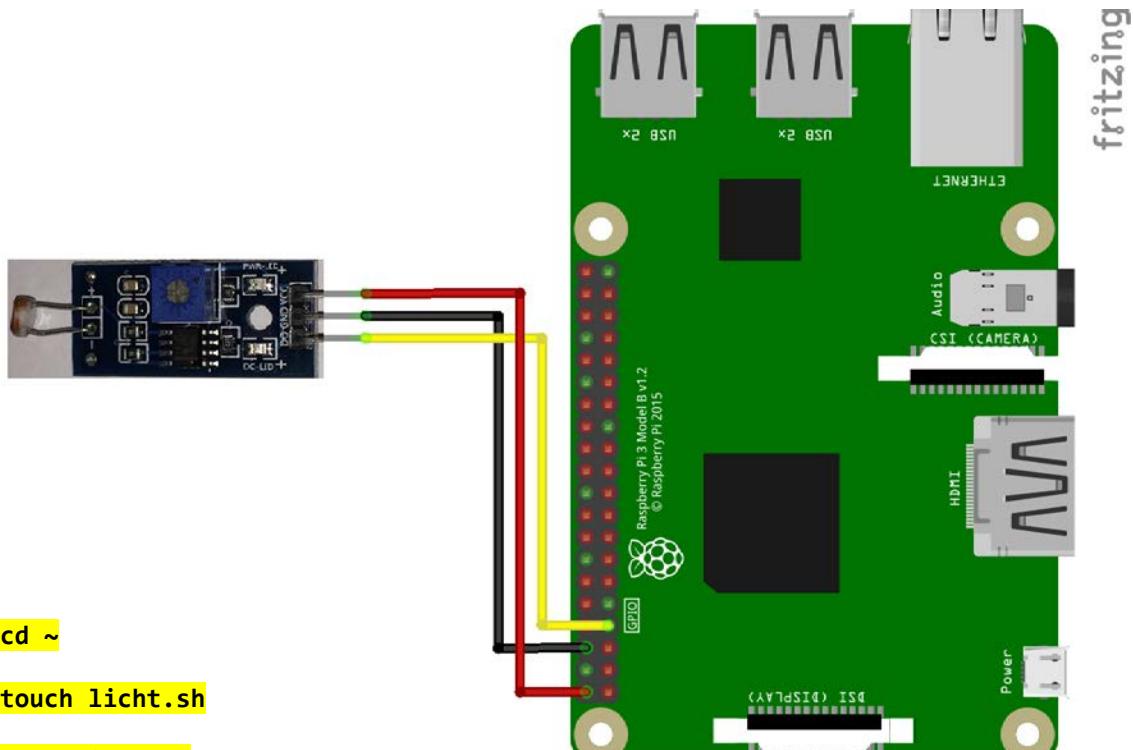
Evtl. muss du noch an den 2 Potentiometer drehen und den Sensor justieren.

```
pi@raspberrypi3p:~ $ python pir.py
Es gab eine Bewegung!
Es gab eine Bewegung!
Es gab eine Bewegung!
Es gab eine Bewegung!
```

2.7 Lichtabhängiger Widerstand LDR

Der Lichtabhängige Widerstand verändert mit der Lichtintensität seinen Widerstand. Sobald der eingestellte Sollwert am Potentiometer überschritten wurde, wird der Ausgang geschalten.

Im Kapitel „2.4 Bodenfeuchtigkeit“ wird beschrieben wie ihr wiringPi installiert.



```
cd ~
```

```
touch licht.sh
```

```
nano licht.sh
```

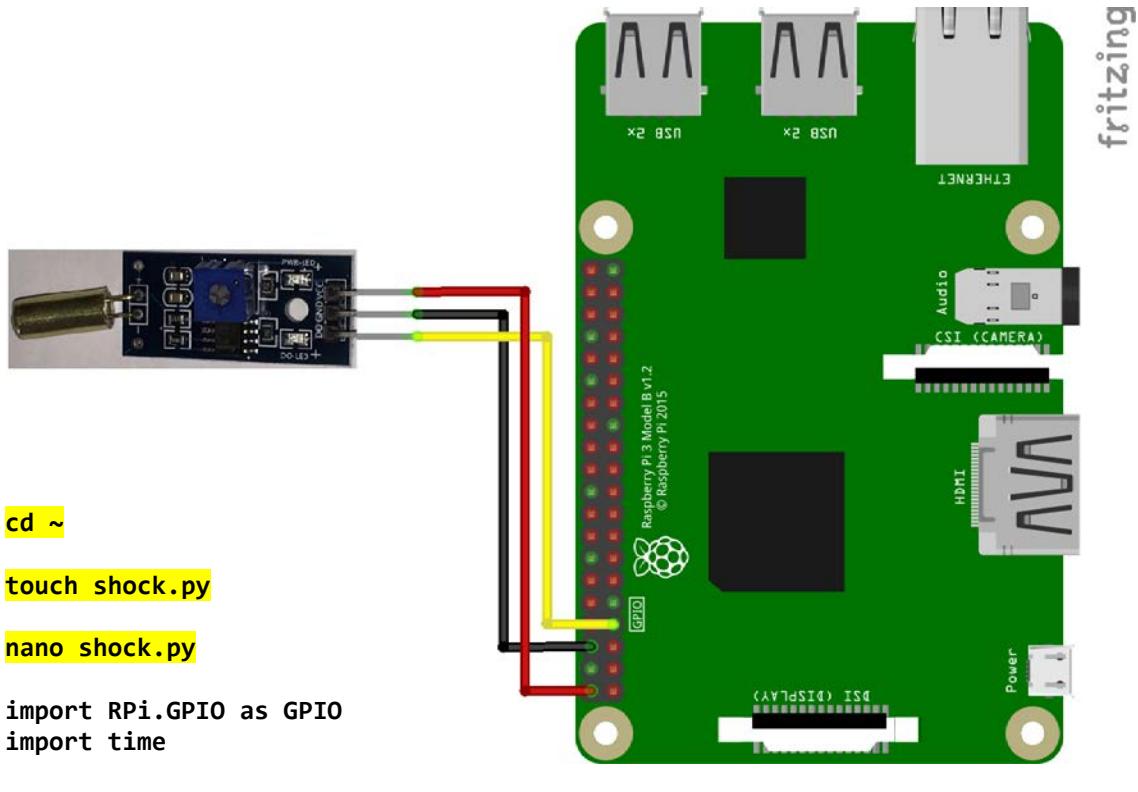
```
#!/bin/sh
gpio -g mode 4 in
while true
do
result=$( gpio -g read 4 )
if [ "$result" = "0" ]; then
echo "Es ist hell"
fi
if [ "$result" = "1" ]; then
echo "Es ist dunkel"
fi
sleep 2
done
```

```
bash /home/pi/licht.sh
```

```
pi@raspberrypi3p:~ $ bash /home/pi/licht.sh
Es ist hell
Es ist dunkel
Es ist dunkel
Es ist hell
Es ist hell
Es ist hell
Es ist hell
```

2.8 Erschütterungssensor

Der Erschütterungssensor gibt ein Signal, wenn eine Erschütterung über einen bestimmten Schwellwert erkannt wurde.



```

cd ~
touch shock.py
nano shock.py

import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

def shock(channel):
    print('Es gab eine Erschuetterung!')
try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=shock)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()

```

python shock.py

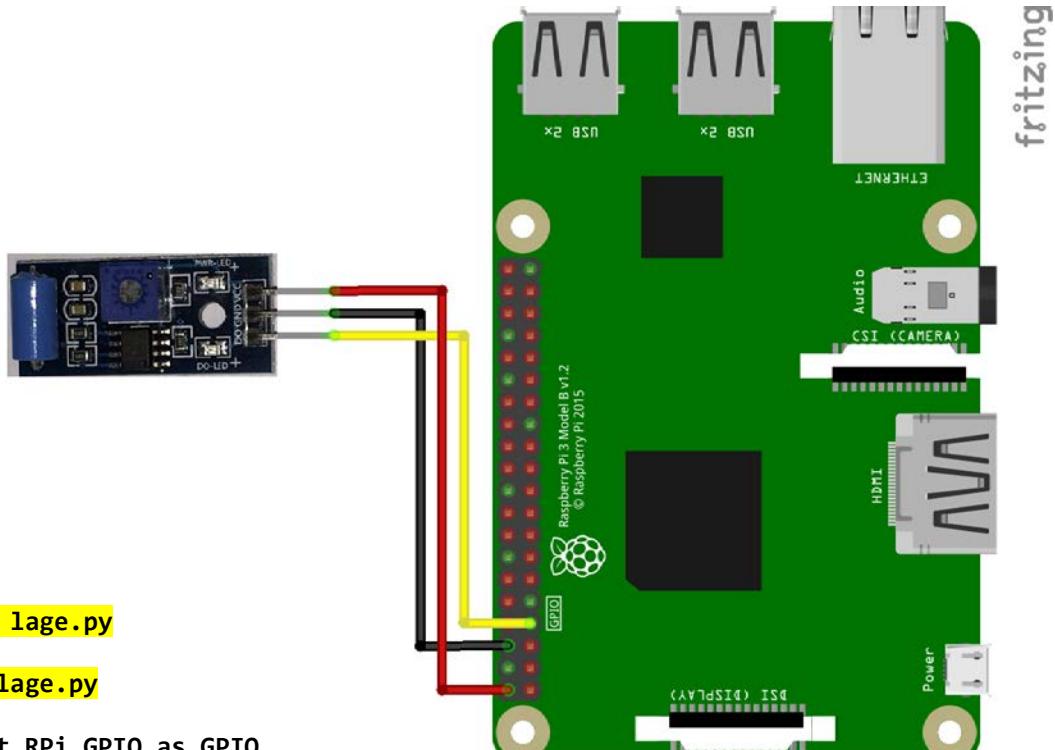
```

pi@raspberrypi3p:~ $ python shock.py
Es gab eine Erschuetterung!
Es gab eine Erschuetterung!
Es gab eine Erschuetterung!
Es gab eine Erschuetterung!

```

2.9 Lagesensor

Der Lagesensor gibt ein Signal, wenn die Lage über einen bestimmten Schwellwert erkannt wurde. Der Lagesensor ist eigentlich auch ein Erschütterungssensor.



```
cd ~
```

```
touch lage.py
```

```
nano lage.py
```

```
import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

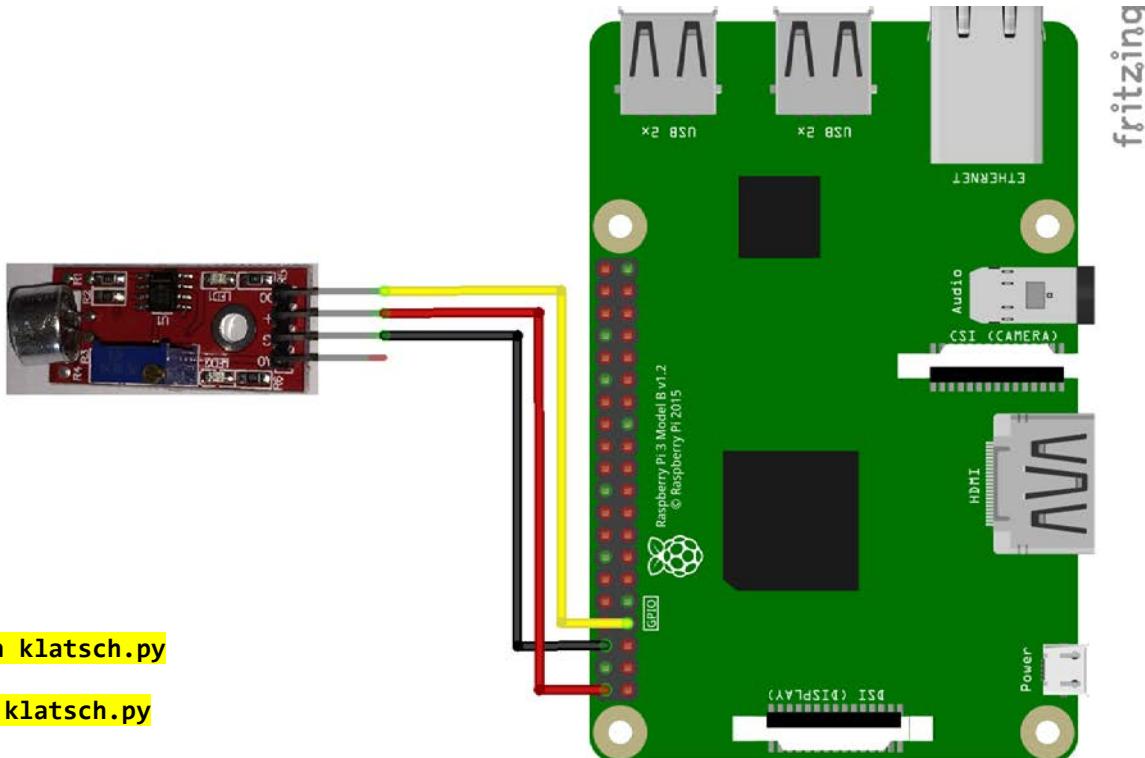
def lage(channel):
    print('Lageänderung erkannt')
try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=lage)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()
```

```
python lage.py
```

```
pi@raspberrypi3p:~ $ python lage.py
Lageänderung erkannt
Lageänderung erkannt
Lageänderung erkannt
```

2.10 Klatschsensor

Der Klatschsensor gibt ein Signal, wenn ein Lauter Ton, der über einen bestimmten Schwellwert liegt, erkannt wurde.



```
cd ~
```

```
touch klatsch.py
```

```
nano klatsch.py
```

```
import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

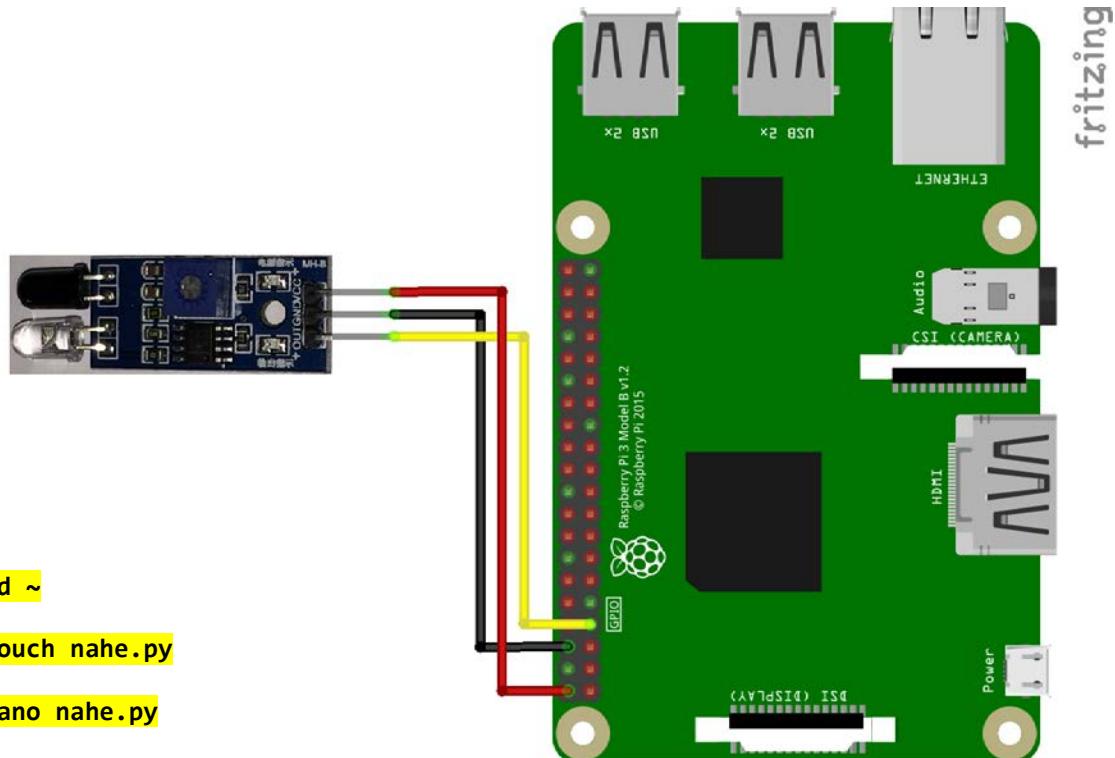
def klatsch(channel):
    print('klatschen erkannt')
try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=klatsch)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()
```

```
python klatsch.py
```

```
pi@raspberrypi3p:~ $ python klatsch.py
klatschen erkannt
klatschen erkannt
klatschen erkannt
klatschen erkannt
```

2.11 Näherungssensor

Der Näherungssensor sendet ein IR-Signal aus und wenn die IR-Empfangsdiode eine Reflektion erkennt, wird der digitale Ausgang geschalten. Die Entfernung kann mit dem Potentiometer eingestellt werden.



```
cd ~
touch nahe.py
nano nahe.py
```

```
import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

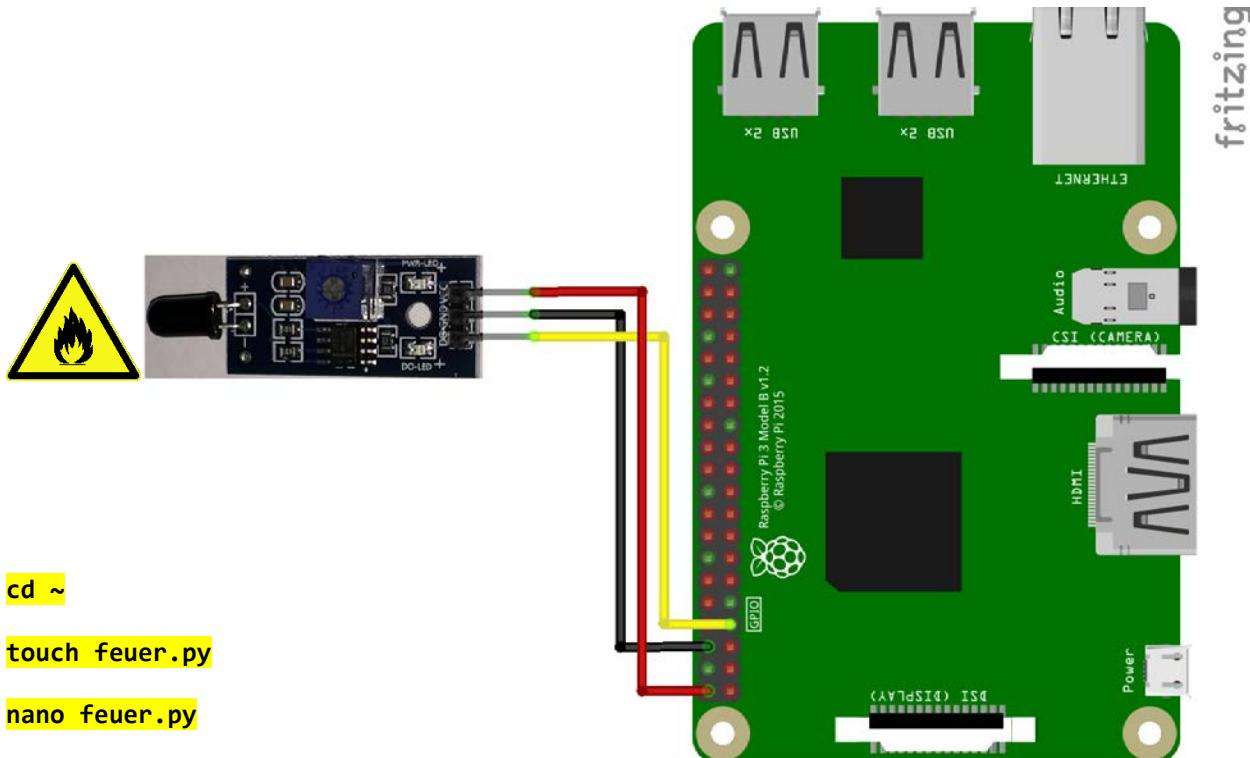
def nahe(channel):
    print('Ein Gegenstand ist nahe dem Sensor')
try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=nahe)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()
```

```
python nahe.py
```

```
pi@raspberrypi3p:~ $ python nahe.py
Ein Gegenstand ist nahe dem Sensor
```

2.12 Flammensensor

Der Flammensensor erkennt Feuer. Wenn eine Flamme eines Feuers (Feuerzeug, ...) erkannt wird, dann wird der Ausgang geschaltet.



```
cd ~
```

```
touch feuer.py
```

```
nano feuer.py
```

```
import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

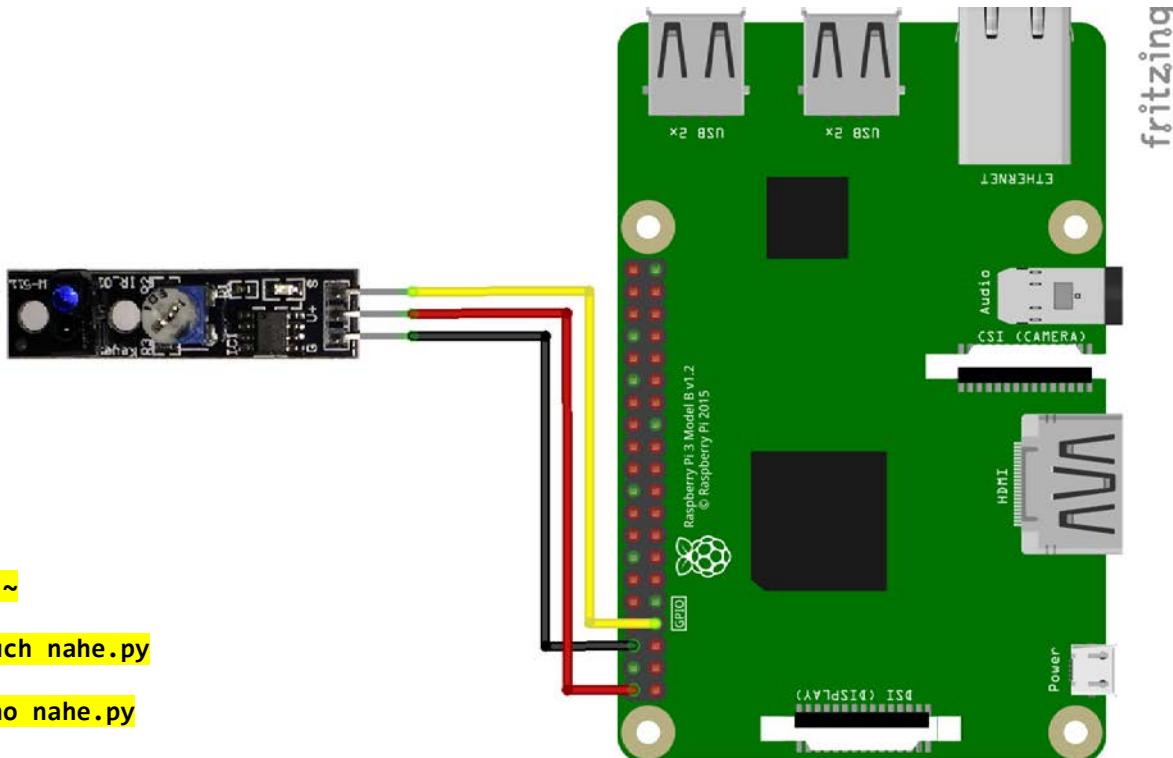
def feuer(channel):
    print('Es brennt eine Flamme!')
try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=feuer)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()
```

```
python feuer.py
```

```
pi@raspberrypi3p:~ $ python feuer.py
Es brennt eine Flamme!
Es brennt eine Flamme!
```

2.13 Näherungssensor 2

Der Näherungssensor sendet ein IR-Signal aus und wenn die IR-Empfangsdiode eine Reflektion erkennt, wird der digitale Ausgang geschalten. Die Entfernung kann mit dem Potentiometer eingestellt werden.



```
cd ~
touch nahe.py
nano nahe.py
```

```
import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

def nahe(channel):
    print('Ein Gegenstand ist nahe dem Sensor')
try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=nahe)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()
```

```
python nahe.py
```

```
pi@raspberrypi3p:~ $ python nahe.py
Ein Gegenstand ist nahe dem Sensor
```

2.14 Ultraschallsensor HC-SR04

Mit dem Ultraschallsensor kann ein Abstand zwischen Sensor und Gegenstand anhand eines Echos gemessen werden.

Der Messvorgang wird mit einer Flanke am Trigger-Eingang gestartet. Das vorhergehende High-Signal muss mindestens 10µs anliegen.

Anschließend wird am Echo-Ausgang ein High-Signal für die Dauer des Echos anliegen. Diese Zeit des High-Signal müssen wir nun messen und in eine Entfernung umrechnen.

Die Umrechnung funktioniert auf Basis der Physik. Der Schall legt in einer Sekunde ca. 330 Meter zurück. Dieser Wert ist abhängig von der Temperatur. Die Ausbreitungsgeschwindigkeit kann man ausrechnen. Bei Raumtemperatur 20°C berechnet man dies so:

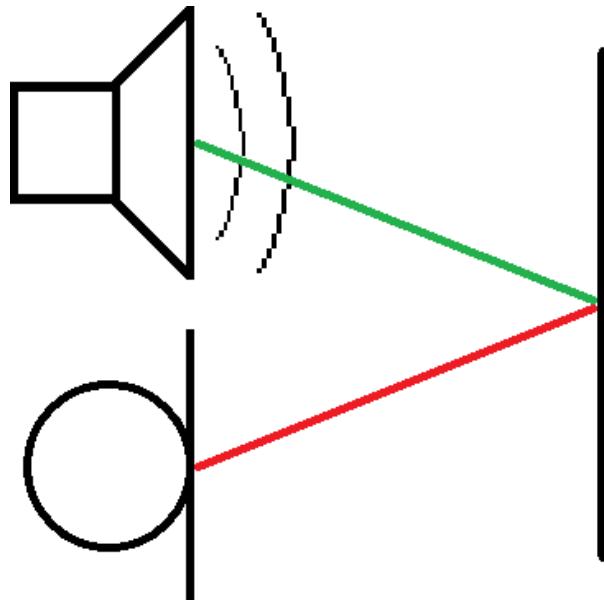
$$\text{Entfernung} = 331,5 \text{m} + (0,6 * \text{Temperatur}) = 331,5 \text{m/s} + (0,6 * 20) = 343,5 \text{ m/s}$$

Bei Raumtemperatur beträgt die Schallgeschwindigkeit also:

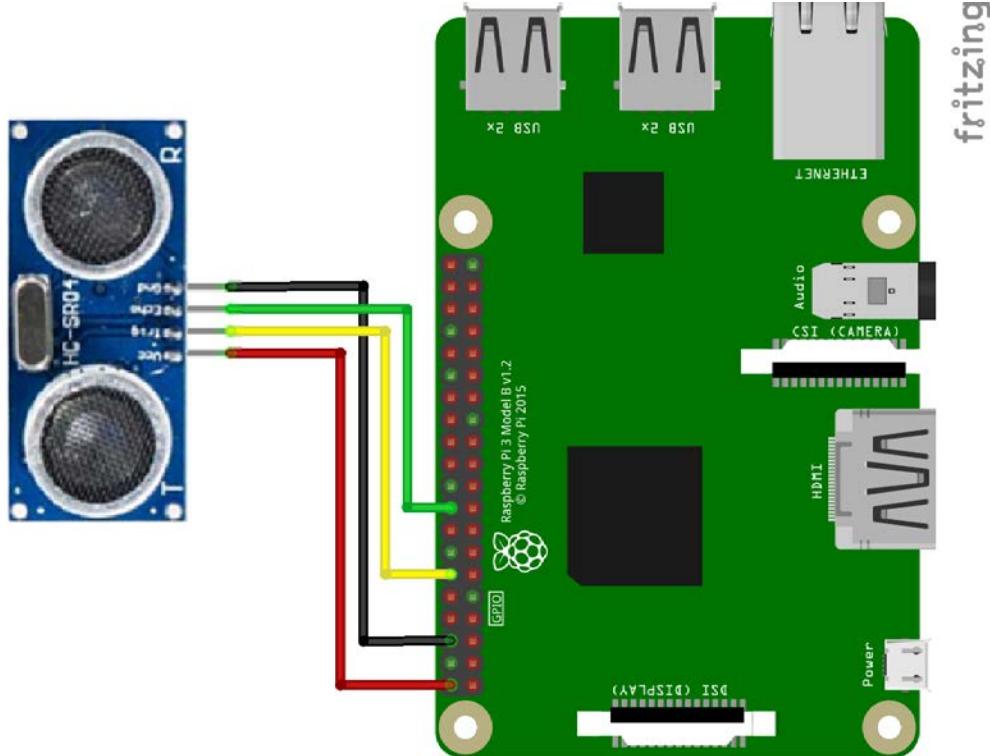
$$343,5 \text{ m/s} \Rightarrow 0,03435 \text{ cm/}\mu\text{s}$$

Nun können wir die Laufzeit des Schalls durch 2 teilen, denn der Schall wird ausgesendet und dann wieder reflektiert, der Weg muss zweimal zurückgelegt werden. Wir wollen aber nur den Abstand, nicht den Weg den der Schall benötigt.

$$\text{Entfernung} = (\text{Zeit} / 2) * \text{Schallgeschwindigkeit} (20^\circ)$$



Außerdem muss man für eine Störungsfreie Messung sorgen, deswegen werden die Interrupts während der Messung deaktiviert.



Wenn man nun das Programm von der nächsten Seite ausführt bekommt man folgende Ausgabe.

```
pi@raspberrypi3p:~ $ python us.py
Entfernung = 104.2 cm
Entfernung = 106.4 cm
Entfernung = 106.5 cm
Entfernung = 16.9 cm
Entfernung = 15.6 cm
Entfernung = 16.2 cm
Entfernung = 14.0 cm
Entfernung = 104.1 cm
Entfernung = 108.2 cm
```

Jede Sekunde wird ein neuer Messvorgang gestartet und ausgegeben.

Az-Delivery

```
cd ~
```

```
touch us.py
```

```
nano us.py
```

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO_TRIGGER = 18
GPIO_ECHO = 24

GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distanz():
    GPIO.output(GPIO_TRIGGER, True)
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)
    StartZeit = time.time()
    StopZeit = time.time()
    while GPIO.input(GPIO_ECHO) == 0:
        StartZeit = time.time()
    while GPIO.input(GPIO_ECHO) == 1:
        StopZeit = time.time()
    TimeElapsed = StopZeit - StartZeit
    distanz = (TimeElapsed * 34300) / 2
    return distanz

if __name__ == '__main__':
    try:
        while True:
            abstand = distanz()
            print ("Entfernung = %.1f cm" % abstand)
            time.sleep(1)

    except KeyboardInterrupt:
        print("Messung gestoppt")
        GPIO.cleanup()
```

```
python us.py
```

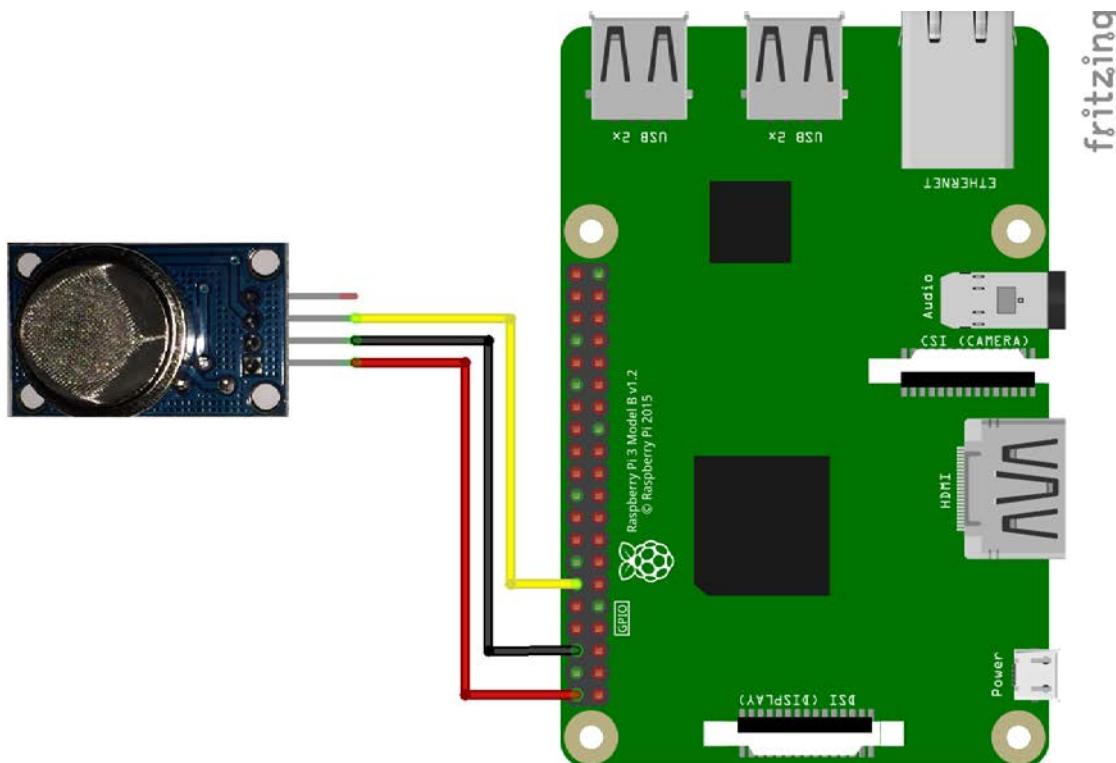
2.15 MQ-2 Gassensor

Der Gassensor kann zum Messen der Konzentration von LPG, i-Butan, Propan, Methan, Alkohol, Wasserstoff und Rauch in der Luft verwendet werden. Der MQ-2 misst eine Gaskonzentration von 100 bis 10000ppm und ist ideal zum Erkennen eines Gaslecks oder als Gas-Alarm.

Der Sensor MQ-2 benutzt ein kleines Heizelement mit einem elektronisch-chemischen Sensor. Sie sind empfindlich gegenüber verschiedenster Gase und eignen sich nur zur Verwendung in Räumen. ACHTUNG: Durch das Heizelement wird der Sensor warm!!

Der Sensor hat eine hohe Empfindlichkeit und schnelle Reaktionszeit, benötigt allerdings einige Minuten bis er genaue Messwerte ausgibt, da der Sensor sich erst aufheizen muss.

Leider hat der Raspberry Pi keinen Analogen Eingang, weshalb wir nur einen Schwellwert überwachen können.



```
cd ~
```

```
touch gas.py
```

```
nano gas.py
```

```

import RPi.GPIO as GPIO
import time

SENSOR_PIN = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

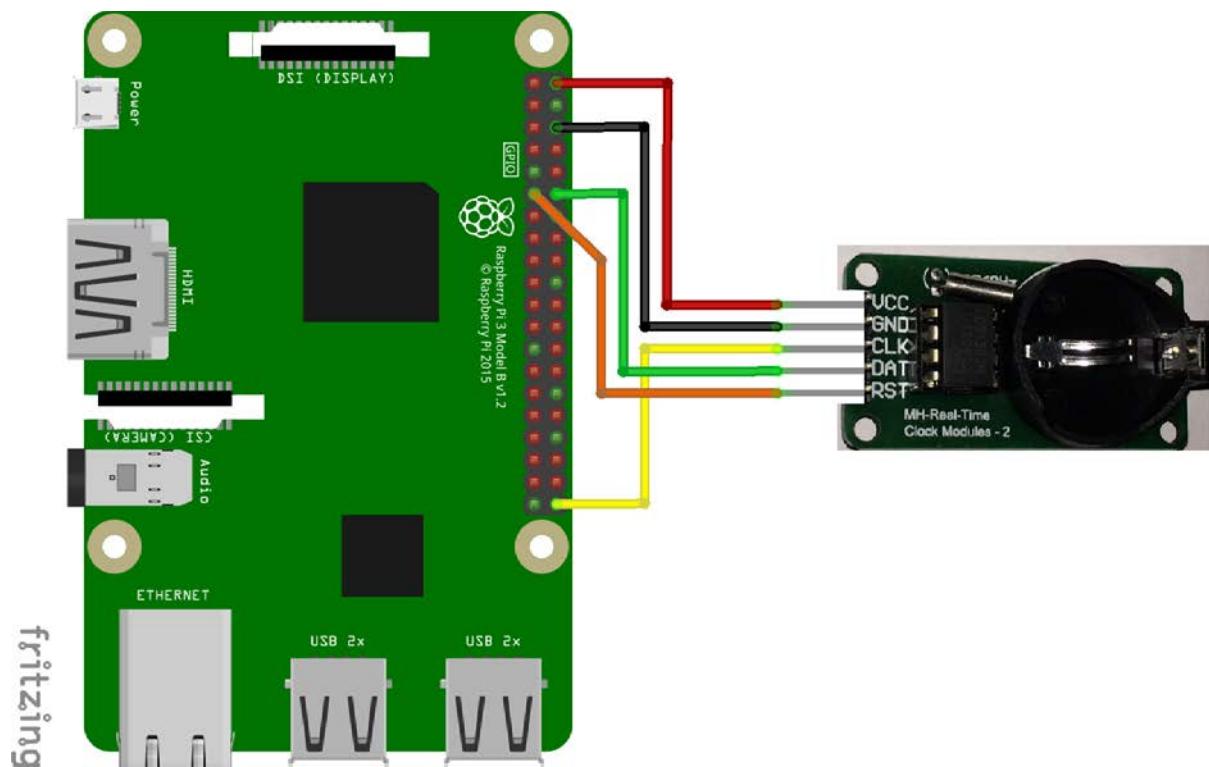
def gas(channel):
    print('Die Gaskonzentration ist zu hoch!')
try:
    GPIO.add_event_detect(SENSOR_PIN , GPIO.RISING, callback=gas)
    while True:
        time.sleep(100)
except KeyboardInterrupt:
    print "Programm wurde beendet."
GPIO.cleanup()

```

python gas.py

2.16 Realtime-Clock

Mit der Realtime-Clock bekommt der Raspberry Pi, auch ohne Internet, nach einem Neustart die korrekte Zeit, denn in dem RTC Modul läuft die Uhr weiter. Für das Modul benötigst du noch eine CR2032 Batterie!



Az-Delivery

```
sudo apt-get install wget
```

```
wget https://www.dropbox.com/s/6ugi82e0oa7x12j/rtc.c
```

```
sudo cc rtc.c -o rtc
```

```
sudo chmod +x rtc
```

```
sudo ./rtc
```

Wenn die Uhrzeit eures Raspberry Pi richtig ist können wir diese direkt auf die RTC schreiben:

```
sudo ./rtc `date +"%Y%m%d%H%M%S"`
```

Alternativ können wir diese auch Manuell setzen:

```
sudo ./rtc YYYMMDDHHMMSS
```

```
20180519164500      >>      19. Mai 2018 16:45:00
```

Mit dem Befehl

```
sudo ./rtc
```

können wir die aktuelle Uhrzeit aus dem RTC-Modul lesen und automatisch die Systemzeit korrigieren.

Du hast es geschafft, du kannst nun viele neue Projekte mit dem Raspberry Pi verwirklichen!

Ab jetzt heißt es Experimentieren.

Und für mehr Hardware sorgt natürlich dein Online-Shop auf:

<https://az-delivery.de>

Viel Spaß!
Impressum

<https://az-delivery.de/pages/about-us>