SensorKit X40

Ausgabe: 29.04.2016

Sehr geehrter Kunde,

vielen Dank, dass Sie sich für unser Produkt entschieden haben.

Dieses hochwertige Sensor Kit wurde speziell für die gängigsten Open-Source Plattformen in Deutschland entwickelt. Es ist zu folgenden Einplatinen Computern kompatibel:

Raspberry Pi (alle Modelle), Arduino, Banana PI, Cubieboard, Cubietruck, Beaglebone, pcDuino und vielen weiteren Mikrocontroller-Systemen (Atmega, MicroChip PIC, STM32 usw.).

Die folgende Anleitung beinhaltet nicht nur die technische Beschreibung der einzelnen Sensoren, wie z.B. die Pin-Belegung oder den jeweils verwendeten Chipsatz, sondern gibt auch einen Einblick in die Funktionsweise der jeweiligen Sensormodule.

Um Sie bei Ihren eigenen Projekten zu unterstützen, haben wir im Zuge dieser Anleitung jeweils ein Code-Beispiel für die meistgenutzten Systeme Raspberry Pi (geschrieben in Python) und Arduino (geschrieben in C++) für jeden einzelnen Sensor bereitgestellt, um die Vorgehensweise der Programmierung mit den Sensoren am leichtesten zu verdeutlichen. Sie finden diese direkt als Code integriert in dieser Anleitung oder direkt unter den jeweiligen Beispiel als Download. Sie finden zudem die immer aktuellste Fassung der Beispiele unter unserem SensorKit X40 Wiki:

http://sensorkit.joy-it.net/

Hierdurch können auch Programmieranfänger ganz leicht eigene Versuche, Projekte und Experimente entwickeln und durchführen.

So haben Sie im Handumdrehen die Möglichkeit Ihren Herzschlag zu prüfen oder die Raumtemperatur und Luftfeuchtigkeit in Ihrer Umgebung zu messen.

Speziell für den Raspberry Pi liegen dem Set ein Analog-Digital Konverter [KY-053] und ein Voltage Translator [KY-051] bei - durch diese werden zwei große Einschränkungen des Raspberry Pi (kein ADC / 3,3V Spannungslevel), wenn es um Interaktion mit analogen Signalen und Sensoren allgemein geht, beseitigt.

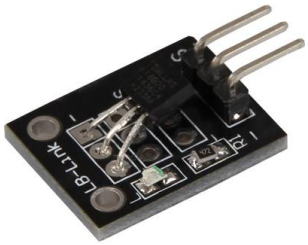
Sie können die Sensoren entweder fest verlöten oder auf ein Breadboard stecken, um an verschiedenen Schaltungen oder Experimenten zu arbeiten.

Wir wünschen Ihnen viel Freude mit dem Sensorkit X40

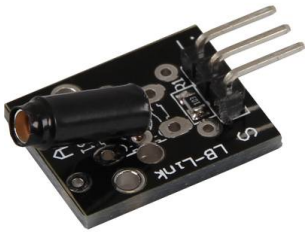
Ihr Joy-IT Team

Index

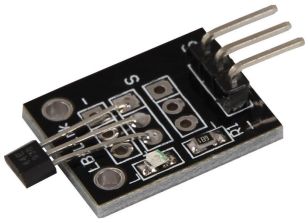
Klicken Sie auf die Beschreibung des jeweiligen Sensors, um auf die entsprechende Unterseite zu gelangen



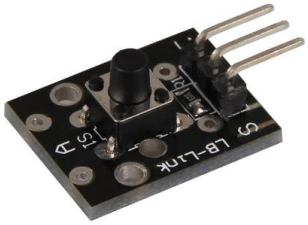
KY-001 Temperatur Sensor Modul



KY-002 Erschütterungs-Schalter Modul



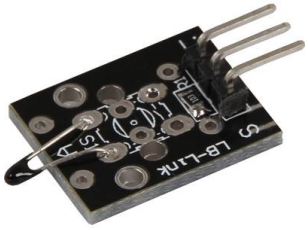
KY-003 Hall Magnetfeld-Sensor Modul



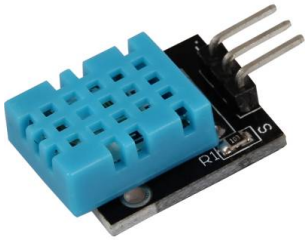
KY-004 Taster-Modul

Index

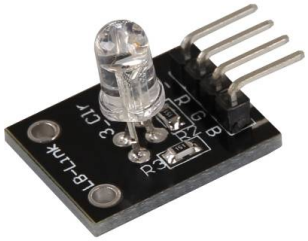
KY-012 Aktives Piezo-Buzzer Modul



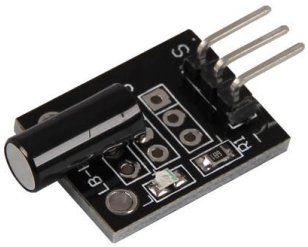
KY-013 Temperatur-Sensor Modul



KY-015 Kombi-Sensor Temperatur+Feuchtigkeit

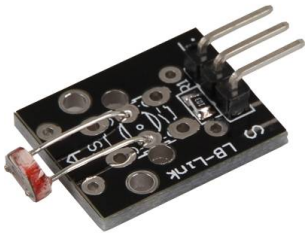


KY-016 RGB 5mm LED Modul

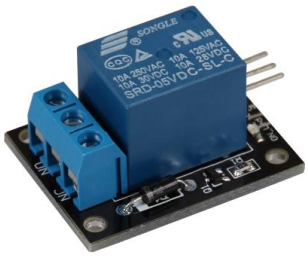


KY-017 Neigungsschalter Modul

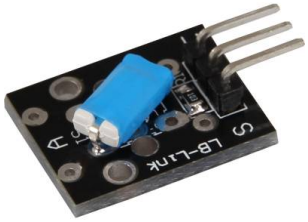
Index



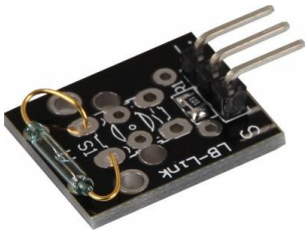
KY-018 Fotowiderstand Modul



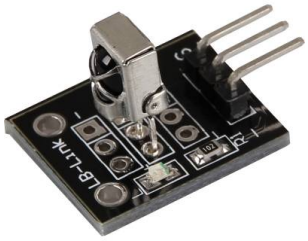
KY-019 5V Relais Modul



KY-020 Neigungs-Schalter Modul



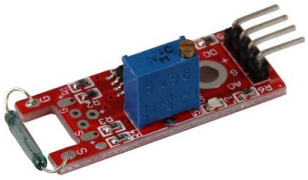
KY-021 Mini Magnet Reed Modul



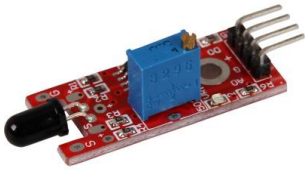
KY-022 Infrarot Receiver Modul

KY-023 Joystick Modul (XY-Achsen)

KY-024 Linear magnetic Hall Sensor

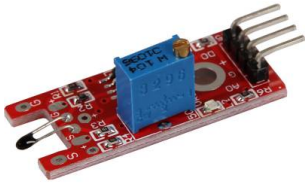


KY-025 Reed Modul



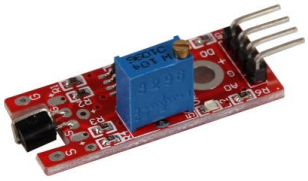
KY-026 Flamen-Sensor Modul

KY-027 Magic Light Cup Modul

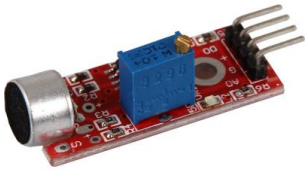


KY-028 Temperatur Sensor Modul (Thermistor)

Index

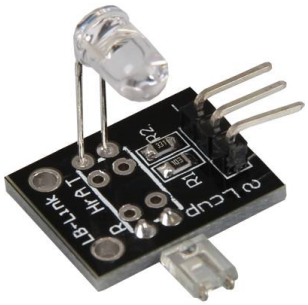


KY-036 Metall-Touchsensor Modul

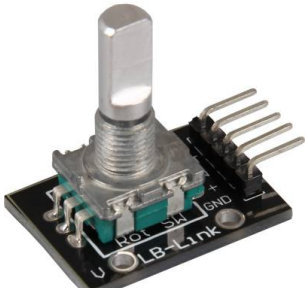


KY-037 Mikrofon Sensor Modul [hohe Empfindlichkeit]

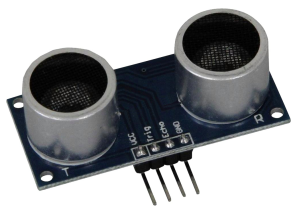
KY-038 Mikrofon Sound Sensor Modul



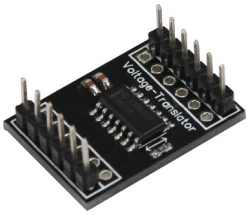
KY-039 Herzschlag Sensor Modul



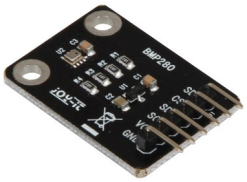
KY-040 Kodierter Drehschalter (Rotary Encoder)



KY-050 Ultraschallabstandssensor



KY-051 Voltage Translator / Level Shifter



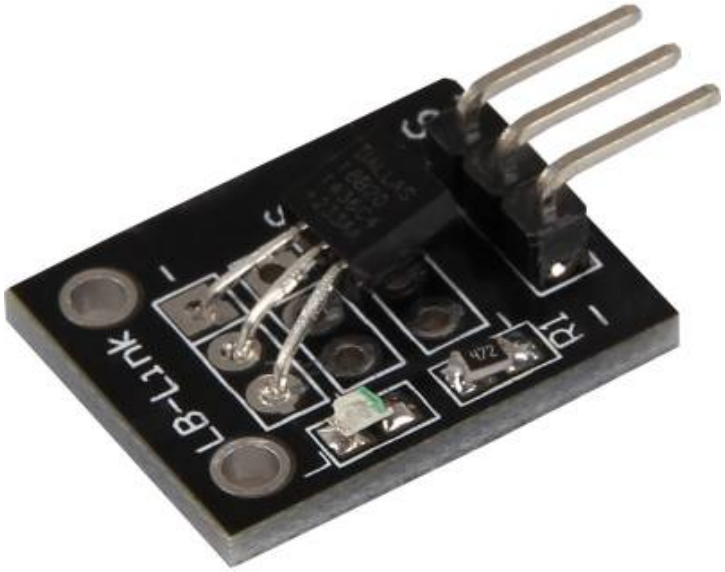
KY-052 Drucksensor / Temperatursensor [BMP280]

KY-053 Analog Digital Converter

KY-001 Temperatur Sensor Modul

Inhaltsverzeichnis

Bild

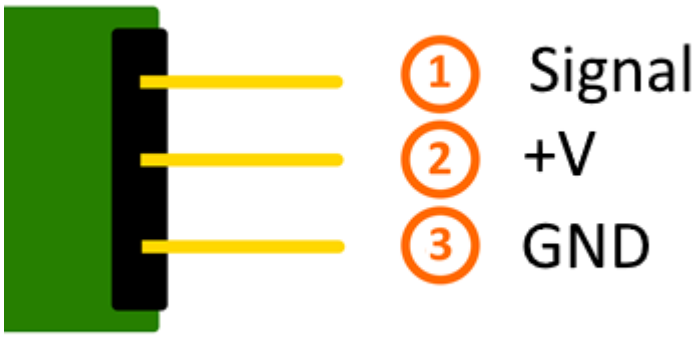


Technische Daten / Kurzbeschreibung

Chipsatz: DS18B20 | Kommunikationsprotokoll: 1-Wire

9- 12Bit genaue Temperaturmessung im Meßbereich von –55°C bis +125°C

Pin-Belegung



Codebeispiel Arduino

Für das folgende Codebeispiel werden zwei zusätzliche Libraries benötigt:

- [OneWire Library] von Paul Stoffregen | veröffentlicht unter der MIT License

- [Dallas Temperature Control Library] von Miles Burton | veröffentlicht unter LGPL

Beide Libraries sind im Paket enthalten und müssen vor dem Start der Arduino IDE in den "library"-Ordner kopiert werden.

Diesen finden Sie standardmäßig unter dem folgenden Pfad Ihrer Windows-Installation:

C:\Benutzer\[Benutzername]\Dokumente\Arduino\libraries

```
// Benötigte Libraries werden importiert
#include <DallasTemperature.h>
#include <OneWire.h>

// Hier wird der Eingangs-Pin deklariert, an dem das Sensor-Modul angeschlossen ist
#define KY001_Signal_PIN 4

// Libraries werden konfiguriert
OneWire oneWire(KY001_Signal_PIN);
DallasTemperature sensors(&oneWire);


void setup() {

        // Initialisierung Serielle Ausgabe
        Serial.begin(9600);
        Serial.println("KY-001 Temperaturmessung");

        // Sensor wird initialisiert
        sensors.begin();
```



```
}

//Hauptprogrammschleife
void loop()
{
        // Temperaturmessung wird gestartet...
        sensors.requestTemperatures();
        // ... und gemessene Temperatur ausgeben
        Serial.print("Temperatur: ");
        Serial.print(sensors.getTempCByIndex(0));
        Serial.write(176); // UniCode-Angabe eines char-Symbols für das "°-Symbol"
        Serial.println("C");

        delay(1000);    // 5s Pause bis zur nächsten Messung
}
```

Anschlussbelegung Arduino:

Sensor Signal	= [Pin 4]
Sensor +V	= [Pin 5V]
Sensor -	= [Pin GND]

Beispielprogramm Download

KY-001-TemperaturSensor.zip

One-Wire Konfiguration Raspberry Pi

Damit der Raspberry Pi mit dem One-Wire Bus, mit der Sensor DS18B20 seine Messdaten digital sendet, kommunizieren kann, muss dieser vorerst aktiviert werden. Hierbei muss die Datei "/boot/config.txt" editiert und um folgende Zeile ergänzt werden:

```
dtoverlay=w1-gpio,gpiopin=4
```

Die Datei können Sie editieren, indem Sie den Befehl...

```
sudo nano /boot/config.txt
```

... in die Konsole eingeben. Mit der Tastenkombination [STRG+X] können Sie das Editieren beenden und mit [STRG+Y] abspeichern.

Nachdem Sie den Raspberry Pi mittels...

```
sudo reboot
```

... neugestartet haben, können Sie das untenstehende Beispiel anwenden.

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python


```
# coding=utf-8
# Benoetigte Module werden importiert und eingerichtet
import glob
import time
from time import sleep
import RPi.GPIO as GPIO

# An dieser Stelle kann die Pause zwischen den einzelnen Messungen eingestellt werden
sleeptime = 1

# Der One-Wire EingangsPin wird deklariert und der integrierte
#PullUp-Widerstand aktiviert GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Nach Aktivierung des Pull-UP Widerstandes wird gewartet,
# bis die Kommunikation mit dem DS18B20 Sensor aufgebaut ist
print 'Warte auf Initialisierung...'

base_dir = '/sys/bus/w1/devices/'
while True:
    try:
        device_folder = glob.glob(base_dir + '28*')[0]
        break
    except IndexError:
        sleep(0.5)
        continue
device_file = device_folder + '/w1_slave'


# Funktion wird definiert, mit dem der aktuelle Messwert am Sensor
# ausgelesen werden kann
def TemperaturMessung():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

# Zur Initialisierung, wird der Sensor einmal "blind" ausgelesen
TemperaturMessung()

# Die Temperaturauswertung: Beim Raspberry Pi werden erkennte one-Wire Slaves im Ordner
# /sys/bus/w1/devices/ einem eigenen Unterordner zugeordnet. In diesem Ordner befindet
# sich die Datei w1-slave # in dem Die Daten, die über dem One-Wire Bus gesendet wurden gespeichert.
# In dieser Funktion werden diese Daten analysiert und die Temperatur herausgelesen und ausgegeben
def TemperaturAuswertung():
    lines = TemperaturMessung()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = TemperaturMessung()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        return temp_c

# Hauptprogrammschleife
# Die gemessene Temperatur wird in die Konsole ausgegeben - zwischen den einzelnen Messungen
# ist eine Pause, deren Länge mit der Variable "sleeptime" eingestellt werden kann
try:
    while True:
        print '---------------------------------------'
        print "Temperatur:", TemperaturAuswertung(), "°C"
        time.sleep(sleeptime)

except KeyboardInterrupt:
    GPIO.cleanup()
```


Anschlussbelegung Raspberry Pi:

Signal	=	GPIO4	[Pin 7]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

KY-001_RPi_TemperaturSensor.zip

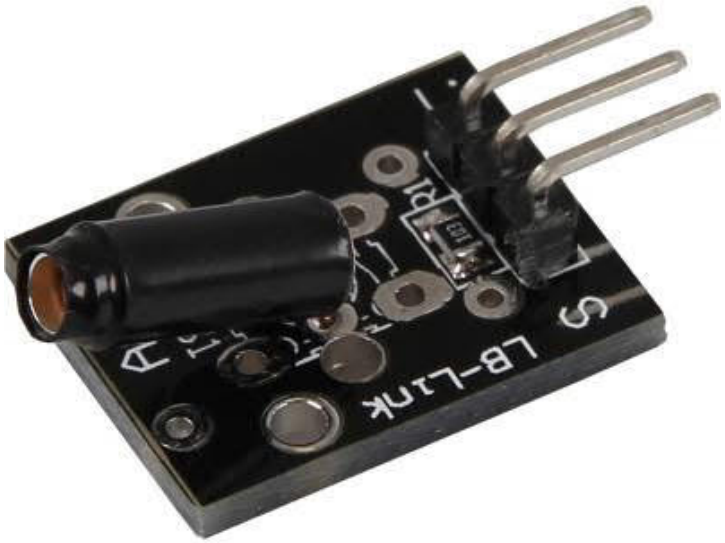
Zu starten mit dem Befehl:

```
sudo python KY-001_RPi_TemperaturSensor.py
```


KY-002 Erschütterungs-Schalter Modul

Inhaltsverzeichnis

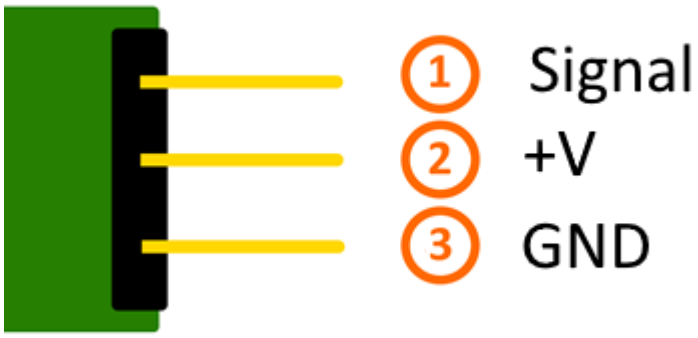
Bild



Technische Daten / Kurzbeschreibung

Bei Erschütterung wird der Kontakt zwischen den zwei Eingangspins geschlossen

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
  digitalWrite(Sensor, HIGH); // Aktivierung interner Pull-Up Widerstand
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]

Sensor +V = [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (fallende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal = GPIO24 [Pin 18]

+V = 3,3V [Pin 1]

GND = Masse [Pin 6]

Beispielprogramm Download

SensorTest_RPi.zip

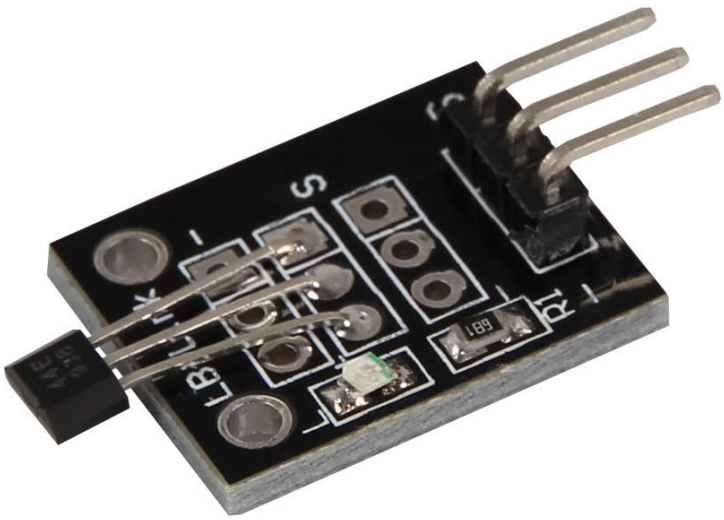
Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi.py
```


KY-003 Hall Magnetfeld-Sensor Modul

Inhaltsverzeichnis

Bild



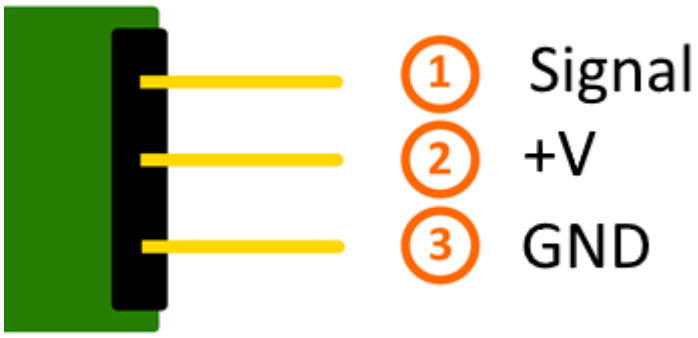
Technische Daten / Kurzbeschreibung

Chipsatz: A3141

Sensortyp: Hall Effect Transistor/Schalter

Der Transistor schaltet durch, falls das Modul in ein Magnetfeld gehalten wird. Dies kann dann am Signalausgang als analoger Spannungswert ausgelesen werden.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
  digitalWrite(Sensor, HIGH); // Aktivierung interner Pull-Up Widerstand
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]

Sensor +V = [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (fallende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal = GPIO24 [Pin 18]

+V = 3,3V [Pin 1]

GND = Masse [Pin 6]

Beispielprogramm Download

SensorTest_RPi.zip

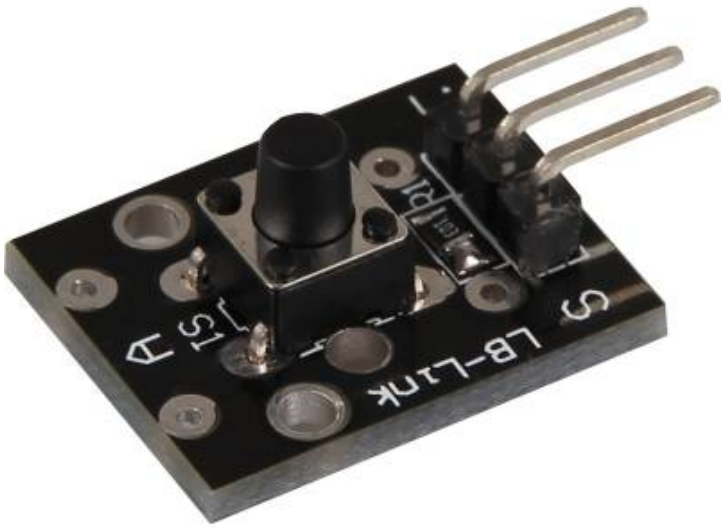
Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi.py
```


KY-004 Taster-Modul

Inhaltsverzeichnis

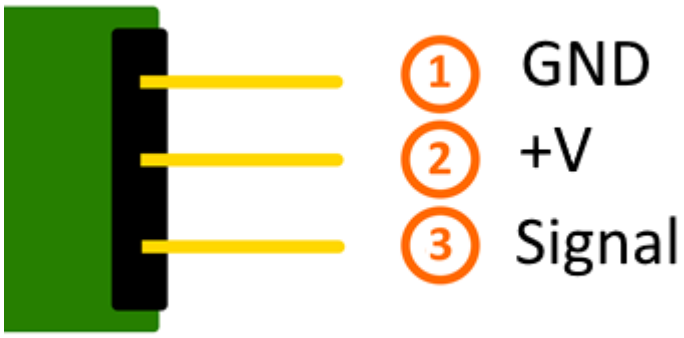
Bild



Technische Daten / Kurzbeschreibung

Beim Drücken des Tasters, werden zwei Signalausgänge miteinander kurzgeschlossen.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
  digitalWrite(Sensor, HIGH); // Aktivierung interner Pull-Up Widerstand
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]

Sensor +V = [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
#Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (fallende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal = GPIO24 [Pin 18]

+V = 3,3V [Pin 1]

GND = Masse [Pin 6]

Beispielprogramm Download

SensorTest_RPi.zip

Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi.py
```


KY-005 Infrarot Transmitter Modul

Inhaltsverzeichnis

Bild

Technische Daten / Kurzbeschreibung

Eine Leuchtdiode, die im infraroten Bereich ausstrahlt. Je nach Eingangsspannung, werden Vorwiderstände benötigt

Vf= 1,1V

If= 20mA

emittierende Wellenlänge: 940nm *(nicht sichtbares Licht)*

Vorwiderstände:

Rf (3,3V) = 120Ω

[z.B. beim Einsatz mit ARM CPU-Kern basierten Mikrokontrollern wie Raspberry-Pi]

Rf (5V) = 220Ω

[z.B. beim Einsatz mit Atmel Atmega basierten Mikrokontrollern wie Arduino]

Pin-Belegung

- *Auf der Platine gibt es die Möglichkeit den jeweils benötigten Widerstand direkt aufzulöten. In dem Falle kann dann der mittlere Pin genutz werden, der dann den Widerstand beihnaltet.

Codebeispiel Arduino

Codebeispiel ON/OFF

Diese Codebeispiel zeigt auf, wie eine LED mittels eines definierbaren Ausgangspins abwechselnd für Vier Sekunden ein- und danach zwei Sekunden ausgeschaltet werden kann.


```
int Led = 13;

void setup ()
{
  pinMode (Led, OUTPUT); // Initialisierung Ausgangspin für die LED
}

void loop () //Hauptprogrammschleife
{
  digitalWrite (Led, HIGH); // LED wird eingeschaltet
  delay (4000); // Wartemodus für 4 Sekunden
  digitalWrite (Led, LOW); // LED wird ausgeschaltet
  delay (2000); // Wartemodus für weitere zwei Sekunden in denen die LED dann ausgeschaltet ist
}
```

Beispielprogramm Download:

LedTestArduino_4On_2Off.zip

Codebeispiel Fernbedienung

Mithilfe der beiden Sensormodule KY-005 und KY-022 lässt sich ein Infrarot-Fernbedienung + Infrarot Receiver System aufbauen. Hierzu werden neben den zwei Modulen auch zwei einzelne Arduinos benötigt. Der eine fungiert hierbei dann als Sender und der andere empfängt die Signale und gibt diese dann in der seriellen Konsole aus.

Für das folgende Codebeispiel wird eine zusätzliche Library benötigt:

- [Arduino-IRremote] von Ken Shirriff | veröffentlicht unter LGPL

Die Library ist im Paket enthalten und muss vor dem Start der Arduino IDE in den "library"-Ordner kopiert werden.

Diesen finden Sie standardmäßig unter dem folgenden Pfad Ihrer Windows-Installation:

C:\Benutzer\[Benutzername]\Dokumente\Arduino\libraries

Bei Infrarot-Sendesystemen, gibt es verschiedene Protokolle, in denen die Daten versendet werden können. In dem folgenden Beispiel wird für das versenden das RC5 Protokoll verwendet - die verwendete Library "Arduino-IRremote" kümmert sich eigenständig um die Konvertierung in die richtige Datenfolge. Es gibt innerhalb der Library jedoch auch andere Protokolle/Kodierungen - diese sind in der Dokumentation/Code der Library gekennzeichnet.

Code für den Empfänger:

```
// Arduino-IRremote Iibrary wird hinzugefuegt
#include <IRremote.h>

// Hier kann der entsprechende Eingangspin für den Signalausgang
// des KY-022 deklariert werden
```



```
int RECV_PIN = 11;

// Arduino-IRremote Library wird initialisiert
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Infrarot-Receiver wird gestartet
}

// Hauptprogrammschleife
void loop() {

  // Es wird geprüft ob am Recveiver ein Signal eingegangen ist
  if (irrecv.decode(&results)) {
    // Bei Signaleingang wird das empfangene und dekodierte Signal
    // in der serriellen Konsole ausgegeben
    Serial.println(results.value, HEX);
    irrecv.resume();
  }
}
```

Code für den Sender:

```
//Arduino-IRremote Library wird hinzugefügt...
#include <IRremote.h>

//...und hier initialisiert
IRsend irsend;

// Die Einstellungen für den Ausgang werden von der Library übernommen
// Die entsprechenden Ausgänge unterscheiden sich je nach verwendeten Arduino
// Arduino UNO:  Ausgang = D3
// Arduino MEGA: Ausgang = D9
// Eine komplette Auflistung der entsprechenden Ausgänge finden Sie unter
// http://z3t0.github.io/Arduino-IRremote/
void setup()
{
}

// Hauptprogrammschleife
void loop() {
       // Der Sender verschickt in diesem Beispiel das Signal A90 (in hexdezimaler Form) in der Kodierung "RC5"
       // Dieses wird dreimal hintereinander gesendet und danach eine Pause für 5 Sekunden eingelegt
         for (int i = 0; i < 3; i++) {
       // [0xA90] zu versendetes Signal | [12] Bit-Länge des zu versendeten Signals (hex A90=1010 1001 0000)
                  irsend.sendRC5(0xA90, 12);
                  delay(40);
         }
         delay(5000); //Zwischen den Sendeimpulsen gibt es eine Pause von 5 Sekunden
}
```

Beispielprogramm Download:

Arduino_Fernbedienung.zip

Anschlussbelegung Arduino 1 [Empfänger]:

KY-022

Signal = [Pin 11]

+V = [Pin 5V]
GND = [Pin GND]

Anschlussbelegung Arduino 2 [Sender]:

KY-005

Signal = [Pin 3 (Arduino Uno) | Pin 9 (Arduino Mega)]
GND+Widerstand = [Pin GND*]
GND = [Pin GND]

- *Nur wenn Vorwiderstand auf dem Modul verlötet wurde und nicht vor dem Modul geschaltet ist

Codebeispiel Raspberry Pi

Es sind zwei Anwendungsbeispiele für dieses Sensormodul hier vorgestellt. Eines, welches die Infrarot-Transmitter Diode kurz ein und wieder ausschaltet (emittierendes Licht nicht sichtbar - kann z.B. durch eine Handykamera gesehen werden), sowie ein direktes Anwendungsbeispiel für den Raspberry Pi, wo er als entweder Infrarot Receiver für Fernbedienungen zum Steuern von z.B. der Mediencenter-Software OpenElec oder als Infrarot-Transmitter zur Software-gesteuerten Fernbedienung programmiert werden kann.

Codebeispiel ON/OFF

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
#Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert LED_PIN = 24
GPIO.setup(LED_PIN, GPIO.OUT, initial= GPIO.LOW)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
                print("LED 4 Sekunden an")
                GPIO.output(LED_PIN,GPIO.HIGH) #LED wird eingeschaltet
                time.sleep(4) #Wartemodus fuer 4 Sekunden
                print("LED 2 Sekunden aus")
                GPIO.output(LED_PIN,GPIO.LOW) #LED wird ausgeschaltet
                time.sleep(2) #Wartemodus fuer weitere zwei Sekunden,
                              # in denen die LED Dann ausgeschaltet ist

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor Signal = GPIO24 [Pin 18]
GND+Widerstand = GND* [Pin 9]

Sensor GND = Masse [Pin 6]

- *Nur wenn Vorwiderstand auf dem Modul verlötet wurde und nicht vor dem Modul geschaltet ist

Beispielprogramm Download

LedTest_RPi_4On_2Off.zip

Zu starten mit dem Befehl:

```
sudo python LedTest_RPi_4On_2Off.py
```

Codebeispiel Fernbedienung

Der Raspberry Pi besitzt mit seiner fortschrittlichen Prozessorarchitektur den Vorteil gegenüber dem Arduino, dass dieser ein komplettes Linux-Betriebssystem betreiben kann. Mit Hilfe eines Infrarot-Receivers können somit nicht nur einfache Datensignale ausgetaucht, sondern es können auch komplette Software-Programme wie z.B. die Mediencenter Software OpenElec per Fernbedienung bedient werden.

Für die Einrichtung eines Infrarot-Steuerungssystem, bietet sich unter Linux bietet die Software "lirc" an (veröffentlicht unter der LGPL - Website). Im Folgenden zeigen wir auf, wie lirc eingerichtet, eine Fernbedienung angelernt werden kann und per Infrarotdiode die erlernten Signale per Infrarot versendet werden können (um z.B. aus dem Raspberry Pi eine per Software steuerbare Infrarot Fernbedienung zu machen).

Zu diesem Zwecke werden die Module KY-005 als Infrarot-Transmitter und KY-022 als Infrarot Receiver angewendet.

Anschlussbelegung Raspberry Pi:

KY-005

Signal = GPIO17 [Pin 11]
GND+Widerstand = GND* [Pin 9]
GND = GND [Pin 6]

- *Nur wenn Vorwiderstand auf dem Modul verlötet wurde und nicht vor dem Modul geschaltet ist

KY-022

Signal = GPI18 [Pin 12]
+V = 3,3V [Pin 17]
GND = GND [Pin 25]

Lirc Installation

Als erstes öffnen wir auf dem Desktop ein Terminal oder verbinden wir uns per SSH mit dem Raspberry Pi. Dort geben Sie den folgenden Befehl ein, um lirc auf den Raspberry Pi zu installieren:


```
sudo apt-get install lirc -y
```

[Hierzu muss der Raspberry Pi mit dem Internet verbunden sein]

Damit das lirc Modul direkt zum Start des Betriebssystem verfügbar ist, müssen folgende Zeilen am Ende der Datei "/boot/config.txt" hinzugefügt werden:

```
dtoverlay=lirc-rpi,gpio_in_pin=18,gpio_out_pin=17,gpio_in_pull=up
```

Hierbei definiert "gpio_in_pin=18" den Eingangspin für den IR-Receiver, sowie "gpio_out_pin=17" den Ausgangspin für den IR-Transmitter.

Die Datei kann mit folgenden Befehl editiert werden:

```
sudo nano /boot/config.txt
```

Mit der Tastenfolge [Strg+X -> Y -> Enter] kann die Datei, nach dem hinzufügen der Zeile am unteren Ende, gespeichert und geschlossen werden.

Auch die Datei "/etc/lirc/hardware.conf" muss modifiziert werden. Hierbei ist der Befehl...

```
sudo nano /etc/lirc/hardware.conf
```

... zu verwenden. Hier müssen die die folgenden Zeilen modifiziert werden. An den entsprechenden Stellen muss dann ...

```
DRIVER="UNCONFIGURED"
--->>
DRIVER="default"
```

```
DEVICE=""
--->>
DEVICE="/dev/lirc0"
```

```
MODULES=""
--->>
MODULES="lirc_rpi"
```

...geändert werden.

Die modifizierte Datei muss dann wie folgt aussehen:

```
# /etc/lirc/hardware.conf
#
# Arguments which will be used when launching lircd
LIRCD_ARGS=""

#Don't start lircmd even if there seems to be a good config file
```



```
#START_LIRCMD=false

#Don't start irexec, even if a good config file seems to exist.
#START_IREXEC=false

#Try to load appropriate kernel modules
LOAD_MODULES=true

# Run "lircd --driver=help" for a list of supported drivers.
DRIVER="default"
# usually /dev/lirc0 is the correct setting for systems using udev
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"

# Default configuration files for your hardware if any
LIRCD_CONF=""
LIRCMD_CONF=""
```

Danach starten wir den Raspberry Pi mit dem folgenden Befehl neu:

```
sudo reboot
```

IR-Receiver Test

Um den angeschlossenen Receiver zu testen, muss vorab lirc mit dem Befehl...

```
sudo /etc/init.d/lirc stop
```

beendet werden. Danach kann mit...

```
mode2 -d /dev/lirc0
```

...getestet werden, ob am Raspberry Pi Signale detektiert werden können. Hierzu nehmen Sie eine Infrarot Fernbedienung und drücken eine beliebige Taste - es sollten Zeilen in der folgenden Form auftauchen:

```
space 95253
pulse 9022
space 2210
pulse 604
space 95246
pulse 9019
space 2211
pulse 601
space 95252
pulse 9019
space 2210
pulse 603
space 95239
pulse 9020
space 2208
pulse 603
...
```

Mit dem Befehl...


```
sudo /etc/init.d/lirc start
```

... kann der lirc-Dienst wieder gestartet werden.

Fernbedienung anlernen

Um eine Infrarot-Fernbedienung im lirc System zu registrieren, muss für die Fernbedienung die Datei "/etc/lirc/lircd.conf" konfiguriert werden. In dieser sind die jeweiligen Zuordnungen von Befehl zu empfangenen Infrarot-Codes gespeichert.

Um eine entsprechend richtig formatierte lircd.conf erstellen zu können, gibt es in der lirc-Software einen Assistenten, der die Datei automatisch erstellt. Die Vorgehensweise für diese Konfiguration ist wie folgt:

Zu aller erst, muss der lirc-Service beendet werden

```
sudo /etc/init.d/lirc stop
```

Mit dem folgenden Befehl starten wir dann den Assistenten:

```
irrecord -d /dev/lirc0 ~/MeineFernbedienung.conf
```

Dieser Assistent führt vorab eine erste Initialisierung der Fernbedienung durch - in dieser müssen mehrere Tasten abwechselnd gedrückt werden, damit das lirc System die entsprechende Kodierung der Fernbedienung erlernen kann. Bitte folgen Sie dazu die entsprechenden Anweisungen des Assistenten. Nach der Initialisierung fragt dann der Assistent nach dem Namen der Knopfzuordnung, die mit einem neuen Infrarotcode aufgezeichnet werden soll. Sie können sich hierzu die Knopfzuordnungen aus der folgenden Datei auswählen

FernbedienungsCodes.txt

Diese müssen dann in den Assistenten eingegeben und mit Enter bestätigt werden. Hiernach startet dann die Aufzeichnung des Infrarot-Codes, für die ausgewählte Taste.

Beispiel: [KEY_0] eingeben -> mit Enter bestätigen -> Taste "0" auf der Fernbedienung drücken -> warten bis der Assistent die Aufnahme bestätigt.

Sollen keine weiteren Tasten angelernt werden, so kann der Assistent mit der Enter-Taste beendet werden. Hiernach ist die Konfigurationsdatei erstellt, jedoch muss noch ein Name für die aufgezeichnete Fernbedienung vergeben werden. Hierzu öffnen wir die Datei im Editor mit:

```
sudo nano ~/MeineFernbedienung.conf
```

Hier kann dann die Zeile 17 von

```
    name  /home/pi/MeineFernbedienung.conf
```

in


```
    name  MeineFernbedienung
```

geändert werden. Beachten Sie hierbei keine Leerzeichen und Sonderzeichen innerhalb des Namens zu verwenden. Mit der Tastenfolge [Strg+X -> Y -> Enter] kann die Datei, nach dem hinzufügen der Zeile am unteren Ende, gespeichert und geschlossen werden.

Nach erstellen der Konfigurationsdatei können Sie vorab für die original lircd.conf mit folgenden Befehl ein Backup erstellen:

```
sudo mv /etc/lirc/lircd.conf /etc/lirc/lircd.conf.bak
```

und mit dem Befehl...

```
sudo cp ~/MeineFernbedienung.conf /etc/lirc/lircd.conf
```

...wird die vorab neu erstellte Konfigurationsdatei für das lirc-System eingesetzt.

Nun kann mit...

```
sudo /etc/init.d/lirc start
```

...das lirc System wieder gestartet werden.

Ab nun ist die neu angelernte Fernbedienung im System registriert und kann in kompatibler Software , wie z. B. dem Mediencenter OpenElec verwendet werden. Alternativ kann mit dem Befehl...

```
irw
```

...die Zuordnungen und die Funktion der Fernbedienung getestet werden.

Will man die Fernbedienung nun z.B. in OpenElec verwenden, kann es je nach Konfiguration der Mediensoftware sein, dass lirc in den Optionen von OpenElec vorab noch aktiviert werden muss.

Befehle senden mit dem Infrarot Transmitter

Möchte man mit dem Raspberry Pi nun Geräte wie z.B. den Fernseher per Infrarot steuern, so können nun die vorab angelernten Befehle mit Hilfe des Infrarot Transmitters wieder versendet werden. So lässt sich z.B. eine Software gesteuerte Infrarot-Steuerung aufbauen oder einzelne Geräte mittels Netzwerk/Internet ein und Ausschalten.

Zuerst überprüfen wir mit folgendem Befehl...

```
irsend LIST MeineFernbedienung ""
```

...welche Zuordnungen für die jeweilige gespeicherte Fernbedienung verfügbar sind.

Nun können wir z.B. den Befehl [KEY_0] versenden, indem wir den folgenden Befehl verwenden:


```
irsend SEND_ONCE MeineFernbedienung KEY_0
```

Auf dem Fernseher bzw. dem entsprechenden Empfänger-Endgerät, sollte sich nun eine Reaktion zeigen. Zu dem o.g. Befehl gibt es auch die Variation, dass das Signal wiederholend ausgegeben wird

```
irsend SEND_START MeineFernbedienung KEY_0
```

Hier nach wird der gesendete Code [KEY_0] so oft wiederholt bis mit...

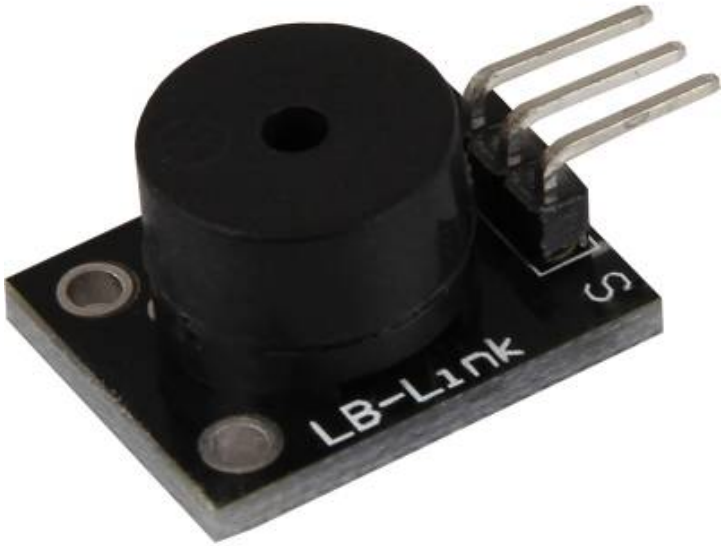
```
irsend SEND_STOP MeineFernbedienung KEY_0
```

...wieder beendet wird. Dies wird z.B. bei Befehlen wie "Lautstärke Hoch" oder "Helligkeit runter" angewendet.

KY-006 Passives Piezo-Buzzer Modul

Inhaltsverzeichnis

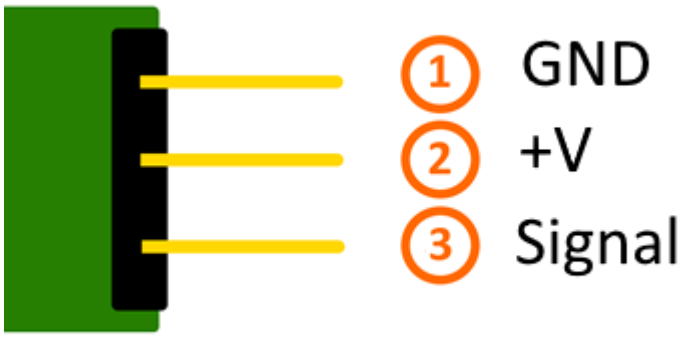
Bild



Technische Daten / Kurzbeschreibung

Mit PWM-Signalen verschiedener Frequenzen angesteuert, können mit dem passiven Piezo-Buzzer verschiedene Töne erzeugt werden.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches am Buzzer ein Alarmsignal mittels einer Rechteckspannung erzeugt.

```
int buzzer = 8 ; // Deklaration des Buzzer-Ausgangspin

void setup ()
{
  pinMode (buzzer, OUTPUT) ;// Initialisierung als Ausgangspin
}


void loop ()
{
  unsigned char i;
  while (1)
  {
    // In diesem Programm wird der Buzzer abwechselnd mit zwei verschiedenen
    // Frequenzen angesteuert. Das Signal hierbei besteht aus einer Rechteckspannung.
    // Das an- und ausmachen am Buzzer generiert dann einen Ton,
    // der in etwa der Frequenz entspricht.
    // Die Frequenz definiert sich dadurch, wie lang jeweils die An- und Ausphase sind

    //Frequenz 1
    for (i = 0; i <80; i++)
    {
      digitalWrite (buzzer, HIGH) ;
      delay (1) ;
      digitalWrite (buzzer, LOW) ;
      delay (1) ;
    }
    //Frequenz 2
    for (i = 0; i <100; i++)
    {
      digitalWrite (buzzer, HIGH) ;
```



```
      delay (2) ;
      digitalWrite (buzzer, LOW) ;
      delay (2) ;
    }
  }
}
```

Anschlussbelegung Arduino:

Sensor Signal = [Pin 8]

Sensor - = [Pin GND]

Beispielprogramm Download

KY-006_Buzzer.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

Das Beispielprogramm nutzt Software-PWM, um am Ausgangspin eine Rechteckspannung mit definierbarer Frequenz zu erstellen.

Durch das An- und Ausschalten wird am Buzzer ein Ton erzeugt, der in etwa der Frequenz der Rechteckspannung entspricht.

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)


# Hier wird der Ausgangs-Pin deklariert, an dem der Buzzer angeschlossen ist.
GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.OUT)

# Das Software-PWM Modul wird initialisiert - hierbei wird die Frequenz 500Hz als Startwert genommen
Frequenz = 500 #In Hertz
pwm = GPIO.PWM(GPIO_PIN, Frequenz)
pwm.start(50)

# Das Programm wartet auf die Eingabe einer neuen PWM-Frequenz vom Benutzer.
# Bis dahin wird der Buzzer mit der vorher eingegebenen Freuqenz betrieben (Startwert 500Hz)
try:
        while(True):
                print "----------------------------------------"
                print "Aktuelle Frequenz: %d" % Frequenz
                Frequenz = input("Bitte neue Frequenz eingeben (50-5000):")
                pwm.ChangeFrequency(Frequenz)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal = GPIO24 [Pin 18]

+V = 3,3V [Pin 1]

GND = Masse [Pin 6]

Beispielprogramm Download

KY-006-RPI_PWM.zip

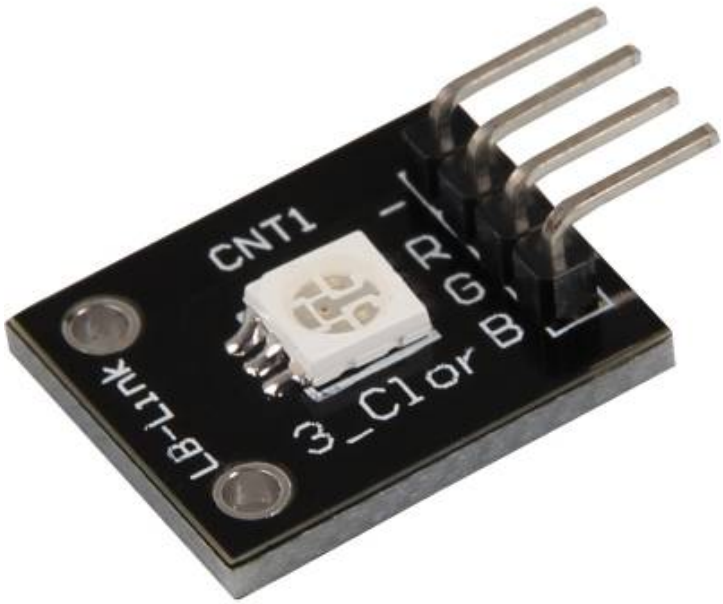
Zu starten mit dem Befehl:

```
sudo python KY-006-RPI_PWM.py
```


KY-009 RGB LED SMD Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

LED-Modul welche eine rote, blaue und grüne LED beinhaltet. Diese sind mittels gemeinsamer Kathode miteinander verbunden. Je nach Eingangsspannung, werden Vorwiderstände benötigt

Vf [Rot]= 1,8V

Vf [Gruen,Blau]= 2,8V

If= 20mA

Vorwiderstände:

Rf (3,3V) [Grün]= 100Ω

Rf (3,3V) [Rot]= 180Ω

Rf (3,3V) [Blau]= 100Ω

[z.B. beim Einsatz mit ARM CPU-Kern basierten Mikrokontrollern wie Raspberry-Pi]

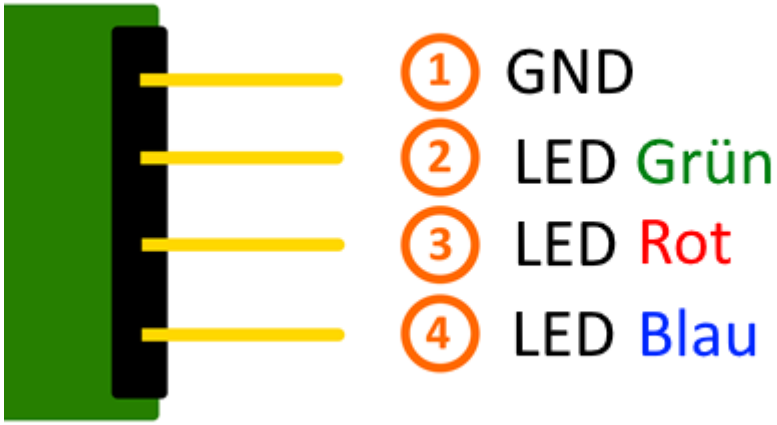
Rf (5V) [Grün] = 100Ω

Rf (5V) [Rot] = 180Ω

Rf (5V) [Blau] = 100Ω

[z.B. beim Einsatz mit Atmel Atmega basierten Mikrokontrollern wie Arduino]

Pin-Belegung



Codebeispiel Arduino

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.

```
int Led_Rot = 10;
int Led_Gruen = 11;
int Led_Blau = 12;

void setup ()
{
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
  pinMode (Led_Blau, OUTPUT);
}
```



```
void loop () //Hauptprogrammschleife
{
  digitalWrite (Led_Rot, HIGH); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Blau, LOW); // LED wird eingeschaltet
  delay (3000); // Wartemodus für 3 Sekunden

  digitalWrite (Led_Rot, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, HIGH); // LED wird eingeschaltet
  digitalWrite (Led_Blau, LOW); // LED wird eingeschaltet
  delay (3000); // Wartemodus für weitere drei Sekunden in denen die LEDs dann umgeschaltet werden

  digitalWrite (Led_Rot, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Blau, HIGH); // LED wird eingeschaltet
  delay (3000); // Wartemodus für weitere drei Sekunden in denen die LEDs dann umgeschaltet werden
}
```

Beispielprogramm ON/OFF Download:

KY-009_LED_ON-OFF.zip

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt.

```
int Led_Rot = 10;
int Led_Gruen = 11;
int Led_Blau = 12;

int val;

void setup () {
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
  pinMode (Led_Blau, OUTPUT);
}
void loop () {
   // Innerhalb einer For-Schleife werden den drei LEDs verschiedene PWM-Werte uebergeben
   // Dadurch entsteht ein Farbverlauf, in dem sich durch das Vermischen unterschiedlicher
   // Helligkeitstufen der beiden integrierten LEDs, unterschiedliche Farben entstehen
   for (val = 255; val> 0; val--)
      {
       analogWrite (Led_Blau, val);
       analogWrite (Led_Gruen, 255-val);
       analogWrite (Led_Rot, 128-val);
       delay (1);
   }
   // In der zweiten For-Schleife wird der Farbverlauf rückwärts durchgegangen
   for (val = 0; val <255; val++)
      {
      analogWrite (Led_Blau, val);
```



```
        analogWrite (Led_Blau, val);
        analogWrite (Led_Gruen, 255-val);
        analogWrite (Led_Rot, 128-val);
        delay (1);
    }
}
```

Beispielprogramm PWM Download:

KY-009_PWM.zip

Anschlussbelegung Arduino:

LED Rot	= [Pin 10]
LED Grün	= [Pin 11]
LED Blau	= [Pin 12]
Sensor GND	= [Pin GND]

Codebeispiel Raspberry Pi

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_ROT = 6
LED_GRUEN = 5
LED_BLAU = 4

GPIO.setup(LED_ROT, GPIO.OUT, initial= GPIO.LOW)
GPIO.setup(LED_GRUEN, GPIO.OUT, initial= GPIO.LOW)
GPIO.setup(LED_BLAU, GPIO.OUT, initial= GPIO.LOW)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
                        print("LED ROT 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.HIGH) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_BLAU,GPIO.LOW) #LED wird eingeschaltet
                        time.sleep(3) # Wartemodus fuer 4 Sekunden
                        print("LED GRUEN 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.HIGH) #LED wird eingeschaltet
                        GPIO.output(LED_BLAU,GPIO.LOW) #LED wird eingeschaltet
                        time.sleep(3) #Wartemodus fuer 3 Sekunden
                        print("LED BLAU 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_BLAU,GPIO.HIGH) #LED wird eingeschaltet
```



```
                        time.sleep(3) #Wartemodus fuer 3 Sekunden

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm ON/OFF Download

KY009_RPi_ON-OFF.zip

Zu starten mit dem Befehl:

```
sudo python KY009_RPI_ON-OFF.py
```

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt. Im Raspberry Pi ist nur ein Hardware-PWM Channel uneingeschränkt auf die GPIO-Pins hinausgeführt, weswegen im vorliegenden Beispiel auf Software-PWM zurückgegriffen wird.

```
# Benoetigte Module werden importiert und eingerichtet
import random, time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_Rot = 6
LED_Gruen = 5
LED_Blau = 4

# Set pins to output mode
GPIO.setup(LED_Rot, GPIO.OUT)
GPIO.setup(LED_Gruen, GPIO.OUT)
GPIO.setup(LED_Blau, GPIO.OUT)

Freq = 100 #Hz

# Die jeweiligen Farben werden initialisiert.
ROT = GPIO.PWM(LED_Rot, Freq)
GRUEN = GPIO.PWM(LED_Gruen, Freq)
BLAU = GPIO.PWM(LED_Blau, Freq)
ROT.start(0)
GRUEN.start(0)
BLAU.start(0)

# Diese Funktion generiert die eigentliche Farbe
# Mittels der jeweiligen Farbvariable, kann die Farbintensitaet geaendert werden
# Nachdem die Farbe eingestellt wurde, wird mittels "time.sleep" die Zeit definiert,
# wie lang die besagte Farbe angezeigt werden soll

def LED_Farbe(Rot, Gruen,Blau, pause):
```



```
    ROT.ChangeDutyCycle(Rot)
    GRUEN.ChangeDutyCycle(Gruen)
    BLAU.ChangeDutyCycle(Blau)
    time.sleep(pause)

    ROT.ChangeDutyCycle(0)
    GRUEN.ChangeDutyCycle(0)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife:
# Diese hat die Aufgabe fuer jede einzelne Farbe eine eigene Variable zu erstellen
# und mittels einer For-Schleife die Farbintensitaet jeder einzelnen Farbe von 0-100% zu druchlaufen
# Durch die Mischungen der verschiedenen Helligkeitsstufen der jeweiligen Farben
# entsteht somit ein Farbverlauf
try:
    while True:
        for x in range(0,2):
            for y in range(0,2):
                for z in range(0,2):
                    print (x,y,z)
                    for i in range(0,101):
                        LED_Farbe((x*i),(y*i),(z*i),.02)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm PWM Download:

KY-009_RPi_PWM.zip

Zu starten mit dem Befehl:

```
sudo python KY-009_RPi_PWM.py
```

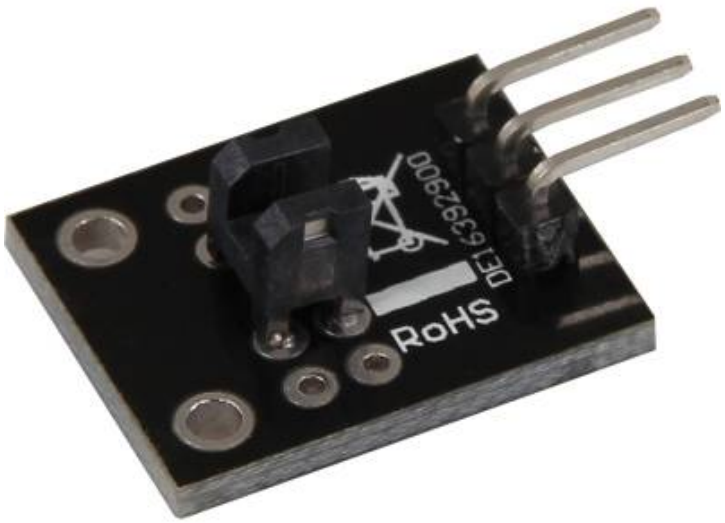
Anschlussbelegung Raspberry Pi:

LED Rot	= GPIO6 [Pin 22]
LED Grün	= GPIO5 [Pin 18]
LED Blau	= GPIO4 [Pin 16]
Sensor GND	= Masse [Pin 6]

KY-010 Lichtschranken-Modul

Inhaltsverzeichnis

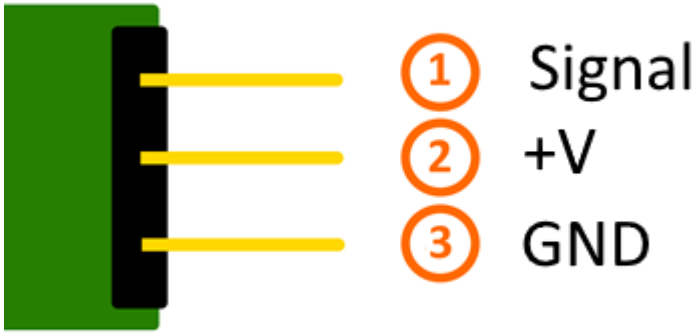
Bild



Technische Daten / Kurzbeschreibung

Die Verbindung zwischen zwei Eingangspins wird unterbrochen, falls die Lichtschranke im Schalter ebenfalls unterbrochen wird.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, HIGH);
  }
  else
  {
    digitalWrite (Led, LOW);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]
Sensor +V	= [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino_inverted.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
#Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (steigende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.RISING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal	=	GPIO24	[Pin 18]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

SensorTest_RPi_inverted.zip

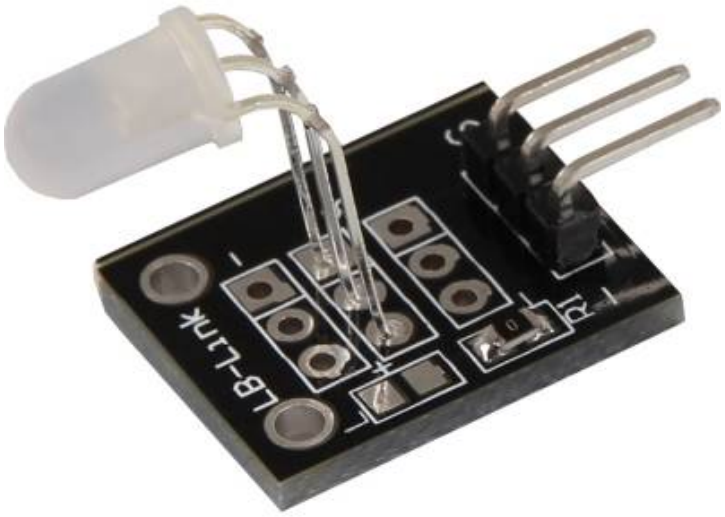
Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi_inverted.py
```


KY-011 2-Farben - Rot+Grün- 5mm LED Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

LED-Modul welche eine rote und grüne LED beinhaltet. Diese sind mittels gemeinsamer Kathode miteinander verbunden. Je nach Eingangsspannung, werden Vorwiderstände benötigt

Vf [typ]= 2,0-2,5V

If= 20mA

Vorwiderstände:

Rf (3,3V) [Grün]= 120Ω

Rf (3,3V) [Rot]= 120Ω

[z.B. beim Einsatz mit ARM CPU-Kern basierten Mikrokontrollern wie Raspberry-Pi]

Rf (5V) [Grün] = 220Ω

Rf (5V) [Rot] = 220Ω

[z.B. beim Einsatz mit Atmel Atmega basierten Mikrokontrollern wie Arduino]

Pin-Belegung

Codebeispiel Arduino

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.

```
int Led_Rot = 10;
int Led_Gruen = 11;

void setup ()
{
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
}

void loop () //Hauptprogrammschleife
{
  digitalWrite (Led_Rot, HIGH); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, LOW); // LED wird eingeschaltet
```



```
   delay (3000); // Wartemodus für 3 Sekunden

   digitalWrite (Led_Rot, LOW); // LED wird eingeschaltet
   digitalWrite (Led_Gruen, HIGH); // LED wird eingeschaltet
   delay (3000); // Wartemodus für weitere zwei Sekunden in denen die LEDs dann umgeschaltet sind
}
```

Beispielprogramm ON/OFF Download:

KY-011_LED_ON-OFF.zip

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt.

```
int Led_Rot = 10;
int Led_Gruen = 11;

int val;

void setup () {
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
}
void loop () {
   // Innerhalb einer For-Schleife werden den beiden LEDs verschiedene PWM-Werte uebergeben
   // Dadurch entsteht ein Farbverlauf, in dem sich durch das Vermischen unterschiedlicher
   // Helligkeitstufen der beiden integrierten LEDs, unterschiedliche Farben entstehen
    for (val = 255; val> 0; val--)
       {
       analogWrite (Led_Gruen, val);
       analogWrite (Led_Rot, 255-val);
       delay (15);
    }
    // In der zweiten For-Schleife wird der Farbverlauf rückwärts durchgegangen
    for (val = 0; val <255; val++)
       {
       analogWrite (Led_Gruen, val);
       analogWrite (Led_Rot, 255-val);
       delay (15);
    }
}
```

Beispielprogramm PWM Download:

KY-011_PWM.zip

Anschlussbelegung Arduino:

LED Grün = [Pin 10]
LED Rot = [Pin 11]
Sensor GND = [Pin GND]

Codebeispiel Raspberry Pi

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_ROT = 5
LED_GRUEN = 4
GPIO.setup(LED_ROT, GPIO.OUT, initial= GPIO.LOW)
GPIO.setup(LED_GRUEN, GPIO.OUT, initial= GPIO.LOW)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
                        print("LED ROT 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.HIGH) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.LOW) #LED wird eingeschaltet
                        time.sleep(3) # Wartemodus fuer 4 Sekunden
                        print("LED GRUEN 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.HIGH) #LED wird eingeschaltet
      #Wartemodus fuer weitere zwei Sekunden, in denen die LED Dann ausgeschaltet ist
                        time.sleep(3)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm ON/OFF Download

KY011_RPI_ON-OFF.zip

Zu starten mit dem Befehl:

```
sudo python KY011_RPI_ON-OFF.py
```

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt. Im Raspberry Pi ist nur ein Hardware-PWM Channel uneingeschränkt auf die GPIO-Pins hinausgeführt, weswegen im vorliegenden Beispiel auf Software-PWM zurückgegriffen wird.

```
# Benoetigte Module werden importiert und eingerichtet
import random, time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_Rot = 5
LED_Gruen = 4

# Set pins to output mode
GPIO.setup(LED_Rot, GPIO.OUT)
GPIO.setup(LED_Gruen, GPIO.OUT)

Freq = 100 #Hz

# Die jeweiligen Farben werden initialisiert.
ROT = GPIO.PWM(LED_Rot, Freq)
GRUEN = GPIO.PWM(LED_Gruen, Freq)
ROT.start(0)
GRUEN.start(0)

# Diese Funktion generiert die eigentliche Farbe
# Mittels der jeweiligen Farbvariable, kann die Farbintensitaet geaendert werden
# Nachdem die Farbe eingestellt wurde, wird mittels "time.sleep" die Zeit definiert,
# wie lang die besagte Farbe angezeigt werden soll

def LED_Farbe(Rot, Gruen, pause):
    ROT.ChangeDutyCycle(Rot)
    GRUEN.ChangeDutyCycle(Gruen)
    time.sleep(pause)

    ROT.ChangeDutyCycle(0)
    GRUEN.ChangeDutyCycle(0)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife:
# Diese hat die Aufgabe fuer jede einzelne Farbe eine eigene Variable zu erstellen
# und mittels einer For-Schleife die Farbintensitaet jeder einzelnen Farbe von 0-100% zu durchlaufen
# Durch die Mischungen der verschiedenen Helligkeitsstufen der jeweiligen Farben
# entsteht somit ein Farbverlauf
try:
        while True:
                for x in range(0,2):
                        for y in range(0,2):
                                print (x,y)
                                for i in range(0,101):
                                        LED_Farbe((x*i),(y*i),.02)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm PWM Download:

KY011_RPI_PWM.zip

Zu starten mit dem Befehl:

```
sudo python KY011_RPI_PWM.py
```

Anschlussbelegung Raspberry Pi:

LED Gruen = GPIO4 [Pin 16]
LED Rot = GPIO5 [Pin 18]

Sensor GND = Masse [Pin 6]

KY-012 Aktives Piezo-Buzzer Modul

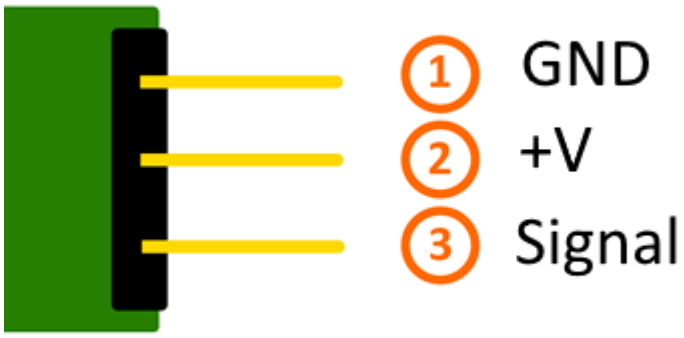
Inhaltsverzeichnis

Bild

Technische Daten / Kurzbeschreibung

Mit Spannung betrieben, erzeugt der aktive Buzzer einen Ton mit der Frequenz von 2,5kHz

Pin-Belegung



Codebeispiel Arduino

Das aktive Buzzer-Modul benötigt, im Gegensatz zum passiven Modul (KY-006) keine Rechteckspannung um einen Ton zu erzeugen - wird an seinem Signal-Pin eine Spannung von min. 3,3V angelegt, so wird im Buzzer die benötigte Rechteckspannung selbstständig erzeugt.

Dieses Codebeispiel zeigt auf, wie der Buzzer mittels eines definierbaren Ausgangspins abwechselnd für Vier Sekunden ein- und danach zwei Sekunden ausgeschaltet werden kann.

```
int Buzzer = 13;

void setup ()
{
  pinMode (Buzzer, OUTPUT); // Initialisierung Ausgangspin für den Buzzer
}

void loop () //Hauptprogrammschleife
{
  digitalWrite (Buzzer, HIGH); // Buzzer wird eingeschaltet
  delay (4000); // Wartemodus für 4 Sekunden
  digitalWrite (Buzzer, LOW); // Buzzer wird ausgeschaltet
  delay (2000); // Wartemodus für weitere zwei Sekunden in denen die LED dann ausgeschaltet ist
}
```

Anschlussbelegung Arduino:

Sensor Signal	= [Pin 13]
Sensor [N.C]	=
Sensor GND	= [Pin GND]

Beispielprogramm Download:

KY-0012_AktiverBuzzer.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

Das aktive Buzzer-Modul benötigt, im Gegensatz zum passiven Modul (KY-006) keine Rechteckspannung um einen Ton zu erzeugen - wird an seinem Signal-Pin eine Spannung von min. 3,3V angelegt, so wird im Buzzer die benötigte Rechteckspannung selbstständig erzeugt.

Dieses Codebeispiel zeigt auf, wie der Buzzer mittels eines definierbaren Ausgangspins abwechselnd für Vier Sekunden ein- und danach zwei Sekunden ausgeschaltet werden kann.

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
Buzzer_PIN = 24
GPIO.setup(Buzzer_PIN, GPIO.OUT, initial= GPIO.LOW)

print "Buzzer-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
                print("Buzzer 4 Sekunden an")
                GPIO.output(Buzzer_PIN,GPIO.HIGH) #Buzzer wird eingeschaltet
        time.sleep(4) #Wartemodus für 4 Sekunden
                print("Buzzer 2 Sekunden aus")
                GPIO.output(Buzzer_PIN,GPIO.LOW) #Buzzer wird ausgeschaltet
      #Wartemodus für weitere zwei Sekunden, in denen die LED Dann ausgeschaltet ist
                time.sleep(2)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor Signal	=	GPIO24	[Pin 18]
Sensor [N.C]	=		
Sensor GND	=	Masse	[Pin 6]

Beispielprogramm Download

LedTest_RPi_4On_2Off.zip

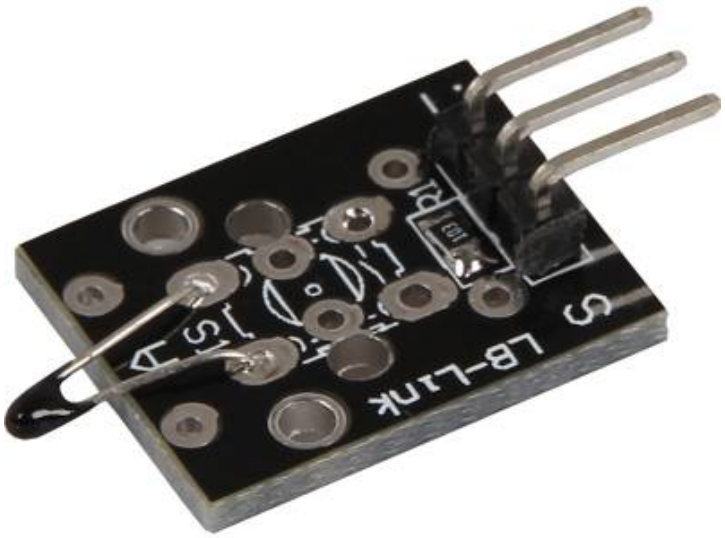
Zu starten mit dem Befehl:

```
sudo python LedTest_RPi_4On_2Off.py
```


KY-013 Temperatur-Sensor Modul

Inhaltsverzeichnis

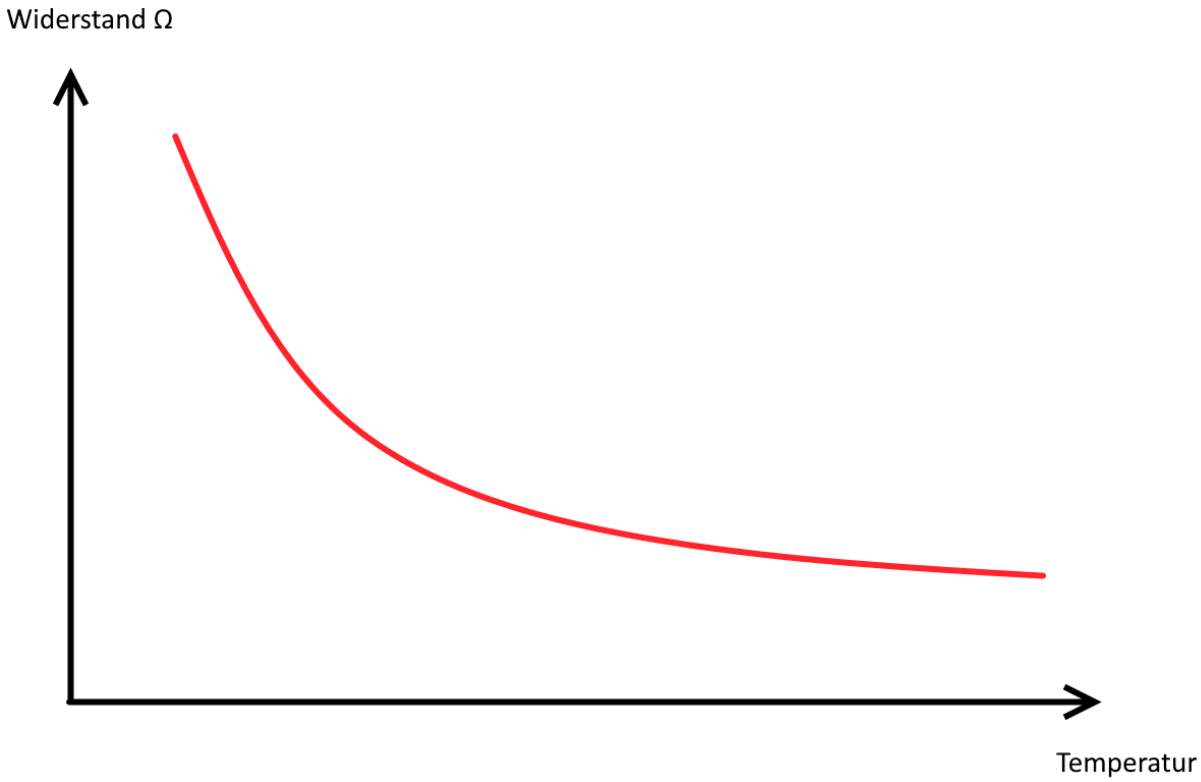
Bild



Technische Daten / Kurzbeschreibung

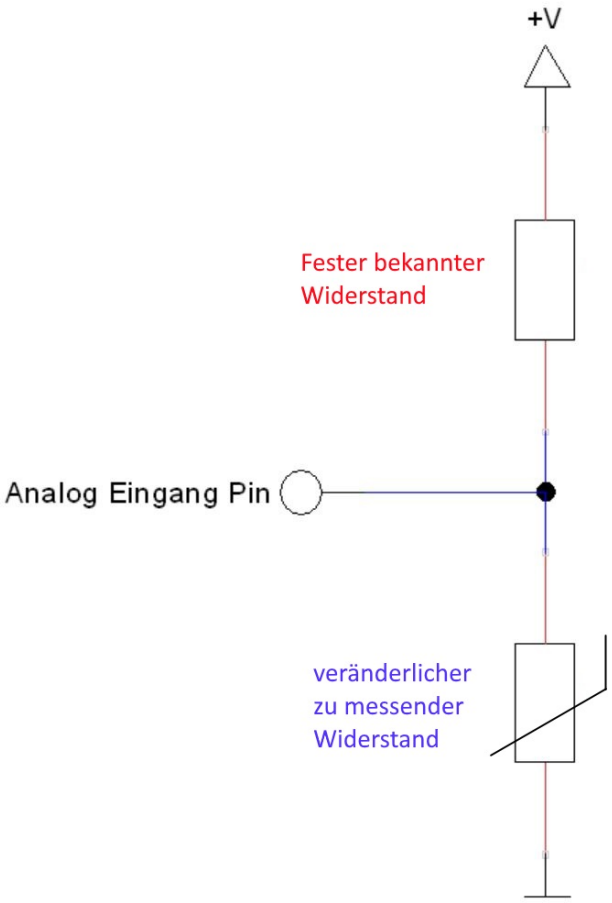
Temperaturmessbereich: -55°C / +125°C

Dieses Modul beinhaltet einen NTC Thermistor—dieser hat bei höherer Temperatur einen immer weniger werdenden Widerstandswert.

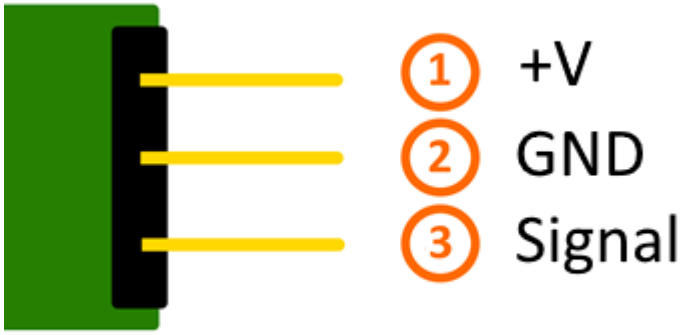


Diese Änderung des Widerstands lässt sich mathematisch annähern und in einen linearen Verlauf umrechnen und den Temperaturkoeffizienten (Abhängigkeit von Widerstandsänderung zur Temperaturänderung) bestimmen. Mittels diesen lässt sich somit dann immer die aktuelle Temperatur errechnen, wenn man den aktuellen Widerstand kennt.

Dieser Widerstand lässt sich mit Hilfe eines Spannungsteilers bestimmen, wo sich eine bekannte Spannung über einen bekannten und einen unbekannten (veränderlichen) Widerstand aufteilt. Mittels dieser gemessenen Spannung lässt sich dann der Widerstand berechnen - die genaue Berechnung ist in den unten stehenden Codebeispielen enthalten.



Pin-Belegung



Codebeispiel Arduino

Das Programm misst den aktuellen Spannungswert am NTC, berechnet die Temperatur und übersetzt das Ergebnis in °C für die serielle Ausgabe

```
#include <math.h>

int sensorPin = A5; // Hier wird der Eingangs-Pin deklariert

// Diese Funktion übersetzt den aufgenommenen analogen Messwert
// in die entsprechende Temperatur in °C und gibt diesen aus
double Thermistor(int RawADC)
{
        double Temp;
        Temp = log(10000.0 * ((1024.0 / RawADC - 1)));
        Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp * Temp )) * Temp );
        Temp = Temp - 273.15;            // Konvertierung von Kelvin in Celsius
        return Temp;
}

// Serielle Ausgabe in 9600 Baud
void setup()
{
         Serial.begin(9600);
}

// Das Programm misst den aktuellen Spannungswert am NTC
// und übersetzt das Ergebnis in °C für die serielle Ausgabe
void loop()
{
         int readVal = analogRead(sensorPin);
         double temp =  Thermistor(readVal);

         // Ausgabe auf die serielle Schnittstelle
         Serial.print("Aktuelle Temperatur ist:");
         Serial.print(temp);
         Serial.print(char(186)); //Ausgabe <°> Symbol
         Serial.println("C");
         Serial.println("---------------------------------------");

         delay(500);
}
```

Anschlussbelegung Arduino:

Sensor +V	= [Pin 5V]
Sensor GND	= [Pin GND]
Sensor Signal	= [Pin A5]

Beispielprogramm Download

KY-013_TemperaturSensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm misst mit Hilfe des ADS1115 ADC den aktuellen Spannungswert am ADC, berechnet daraus den aktuellen Widerstand des NTC, berechnet mit Hilfe vorab für diesen Sensor bestimmter Werte die Temperatur und gibt diese in die Konsole aus.

```
# coding=utf-8
#!/usr/bin/python


##################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### KY-013 Temperatur Sensor - Raspberry Pi Python Code Example
### ##############################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os, math
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC
```



```
ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
# sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

#########################################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm misst mit Hilfe des ADS1115 ADC den aktuellen Spannungswert am ADC,
# berechnet daraus den aktuellen Widerstand des NTC, berechnet mit Hilfe vorab für diesen Sensor bestimmter Werte
# die Temperatur und gibt diese in die Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen...
                voltage = adc.readADCSingleEnded(adc_channel, gain, sps)
                # ... umgerechnet ...
                temperatur = math.log((10000/voltage)*(3300-voltage))
                temperatur = 1/(0.001129148+(0.000234125+(0.0000000876741*temperatur*temperatur))*temperatur);
                temperatur = temperatur - 273.15;
                # ... und ausgegeben
                print "Temperatur:", temperatur,"°C"
                print "--------------------------------------"

                # Delay
                time.sleep(delayTime)



except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]

analoges Signal = Analog 0 [Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	=	3,3V	[Pin 17]
GND	=	Masse	[Pin 09]
SCL	=	GPIO03 / SCL	[Pin 05]
SDA	=	GPIO02 / SDA	[Pin 03]
A0	=	s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

KY-013_RPi_TemperaturSensor.zip

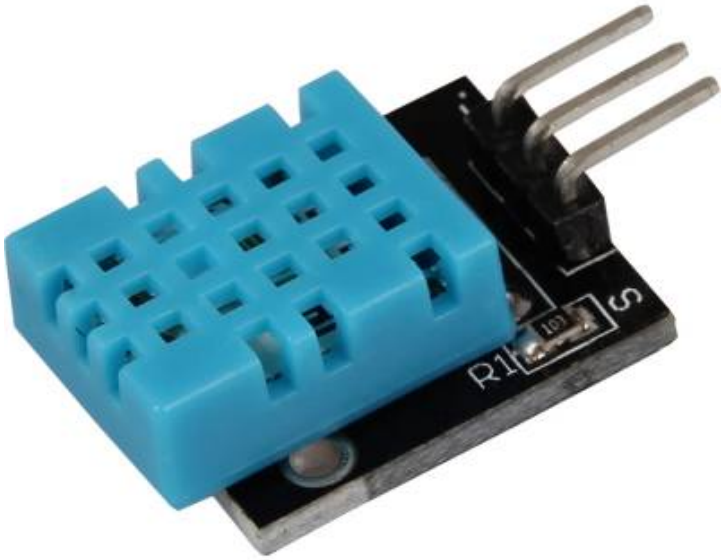
Zu starten mit dem Befehl:

```
sudo python KY-013_RPi_TemperaturSensor.py
```


KY-015 Kombi-Sensor Temperatur+Feuchtigkeit

Inhaltsverzeichnis

Bild

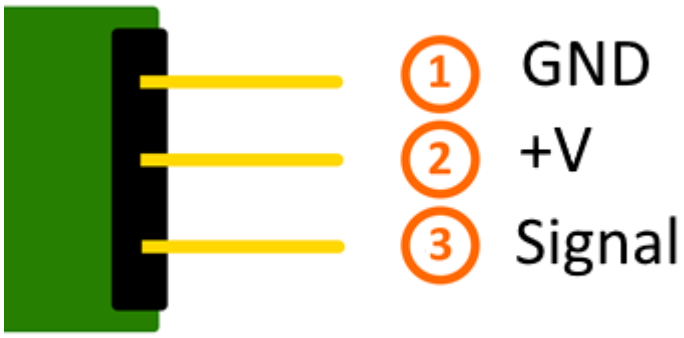


Technische Daten / Kurzbeschreibung

Chipsatz: DHT11 | Kommunikationsprotokoll: 1-Wire Messbereich Luftfeuchtigkeit: 20-90%RH Messbereich Temperatur: 0-50°C

Vorteile dieses Sensors ist die Kombination von Temperaturmessung und Luftfeuchtigkeitsmessung in einer kompakten Bauform - der Nachteil ist jedoch die geringe Abtastrate der Messung, so dass nur jede 2 Sekunden ein neues Messergebnis zur Verfügung steht - dieser Sensor ist somit sehr gut für Langzeit-Messungen geeignet.

Pin-Belegung



Software-Beispiel Arduino

Dieser Sensor gibt sein Messergebnis nicht als analoges Signal auf einen Ausgangspin aus, sondern kommuniziert diesen digital kodiert.

Zur Ansteuerung dieses Sensormoduls gibt es mehrere Möglichkeiten - als besonders zugänglich hat sich die Adafruit_DHT Library erwiesen, die die Firma Adafruit unter dem folgenden Link unter der OpenSource MIT-Lizenz veröffentlicht hat.

Das unten stehende Beispiel verwendet diese besagte Library - hierzu empfehlen wir diese von Github herunterzuladen, zu entpacken und im Arduino-Library-Ordner, welcher sich standardmäßig unter (C:\Benutzer\[Benutzername]\Dokumente\Arduino\libraries) befindet, zu kopieren, damit diese für dieses Codebeispiel und folgende Projekte zur Verfügung steht. Alternativ ist diese auch im unten stehenden Download Paket ebenfalls enthalten.

```
// Adafruit_DHT Library wird eingefügt
#include "DHT.h"

// Hier kann der jeweilige EingangsPin deklariert werden
#define DHTPIN 2

// Der Sensor wird initialisiert
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
  Serial.begin(9600);
  Serial.println("KY-015 Test - Temperatur und Luftfeuchtigkeits-Test:");

  // Messung wird gestartet
  dht.begin();
}

// Hauptprogrammschleife
// Das Programm startet die Messung und liest die gemessenen Werte aus
// Zwischen den Messungen wird eine Pause von 2 Sekunden eingelegt,
```



```
// damit beim nächsten Druchlauf eine neue Messung erfasst werden kann.
void loop() {

  // Zwei Sekunden Pause zwischen den Messungen
  delay(2000);

  // Luftfeuchtigkeit wird gemessen
  float h = dht.readHumidity();
  // Temperatur wird gemessen
  float t = dht.readTemperature();

  // Hier wird überprüft, ob die Messungen fehlerfrei druchgelaufen sind
  // Bei Detektion eines Fehlers, wird hier eine Fehelrmeldung ausgegeben
  if (isnan(h) || isnan(t)) {
    Serial.println("Fehler beim Auslesen des Sensors");
    return;
  }

  // Ausgabe in die serrielle Konsole
  Serial.println("------------------------------------------------------------");
  Serial.print("Luftfeuchtigkeit: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperatur: ");
  Serial.print(t);
  Serial.print(char(186)); //Ausgabe <°> Symbol
  Serial.println("C ");
  Serial.println("------------------------------------------------------------");
  Serial.println(" ");
}
```

Bitte beachten Sie, dass der Sensor nur etwa alle 2 Sekunden ein neues Messergebnis zur Verfügung stellt; also eher für Langzeit-Aufnahmen ausgelegt ist

Beispielprogramm Download:

KY-015-Kombi-Sensor_Temperatur_Feuchtigkeit.zip

Anschlussbelegung Arduino:

GND	=	[Pin GND]
+V	=	[Pin 5V]
Signal	=	[Pin D2]

Software-Beispiel Raspberry Pi

Das Programm nutzt zur Ansteuerung des DHT11-Sensors, der auf diesem Sensor-Modul verbaut ist, die entsprechende Adafruit_Python_DHT Library der Firma Adafruit. Diese wurden unter dem folgenden Link unter der MIT OpenSource-Lizenz veröffentlicht.

Diese muss vorab erst installiert werden. Hierzu muss folgendermaßen vorgegangen werden:

Zuerst muss, falls dies nicht auf dem Raspberry Pi geschehen ist, die GitHub-Software installiert werden:

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

Hierzu muss der Raspberry Pi mit dem Internet verbunden sein. Mit dem Befehl...


```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git
```

... kann die aktuelle Version der Adafruit_BM085 Library heruntergeladen und entpackt werden

Danach wechseln wir mit...

```
cd Adafruit_Python_BMP/
```

... in den heruntergeladenen Ordner und intallieren mit...

```
sudo python setup.py install
```

... die Library. Hiernach kann die Library genutzt werden.

Damit der Raspberry Pi mit dem Sensor auf dem I2C-Bus kommunizieren kann, muss auch vorab die I2C-Funktion beim Raspberry Pi aktiviert werden. Hierzu müssen folgende Zeilen am Ende der Datei "/boot /config.txt" hinzugefügt werden:

```
    dtparam=i2c_arm=on
```

Die Datei kann mit folgenden Befehl editiert werden:

```
sudo nano /boot/config.txt
```

Mit der Tastenfolge [Strg+X -> Y -> Enter] kann die Datei, nach dem hinzufügen der Zeile am unteren Ende, gespeichert und geschlossen werden.

Außerdem werden zusätzliche Bibliotheken benötigt, um I2C innerhalb Python nutzen zu können. Um diese zu installieren muss folgender Befehl in die Konsole eingegeben werden:

```
sudo apt-get install python-smbus i2c-tools -y
```

Hiernach kann das folgende Python-Code Beispiel verwendet werden. Das Programm startet die Messung am Sensor und gibt die gemessenen Werte für den Luftdruck, der Temperatur und der Höhe überm Meeresspiegel aus.

```
#!/usr/bin/python
# coding=utf-8

# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import Adafruit_DHT
import time

# Die Pause von zwei Sekunden zwischen den Messungen wird hier eingestellt
sleeptime = 2

# Sensor should be set to Adafruit_DHT.DHT11,
# Adafruit_DHT.DHT22, or Adafruit_DHT.AM2302.
DHTSensor = Adafruit_DHT.DHT11
```



```
# Hier kann der Pin deklariert werden, an dem das Sensormodul angeschlossen ist
GPIO_Pin = 23

print('KY-015 Sensortest - Temperatur und Luftfeuchtigkeit')

try:
    while(1):
        # Messung wird gestartet und das Ergebnis in die entsprechenden Variablen geschrieben
        Luftfeuchte, Temperatur = Adafruit_DHT.read_retry(DHTSensor, GPIO_Pin)

        print("-----------------------------------------------------------------")
        if Luftfeuchte is not None and Temperatur is not None:

            # Das gemessene Ergebnis wird in der Konsole ausgegeben
            print('Temperatur = {0:0.1f}°C  | rel. Luftfeuchtigkeit = {1:0.1f}%'.format(Temperatur, Luftfeuchte))

        # Da der Raspberry Pi aufgrund des Linux-Betriebsystems für Echtzeitanwendungen benachteiligt ist,
        # kann es sein, dass aufgrund von Timing Problemen die Kommunikation scheitern kann.
        # In dem Falle wird eine Fehlermeldung ausgegeben - ein Ergebnis sollte beim nächsten Versuch vorliegen
        else:
            print('Fehler beim Auslesen - Bitte warten auf nächsten Versuch!')
        print("-----------------------------------------------------------------")
        print("")
        time.sleep(sleeptime)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
    GPIO.cleanup()
```

Bitte beachten Sie, dass der Sensor nur etwa alle 2 Sekunden ein neues Messergebnis zur Verfügung stellt; also eher für Langzeit-Aufnahmen ausgelegt ist

Anschlussbelegung Raspberry Pi:

GND	=	GND	[Pin 06]
+V	=	3,3V	[Pin 01]
Signal	=	GPIO23	[Pin 16]

Beispielprogramm Download

KY-015-RPi_Kombi-Sensor_Temperatur_Feuchtigkeit.zip

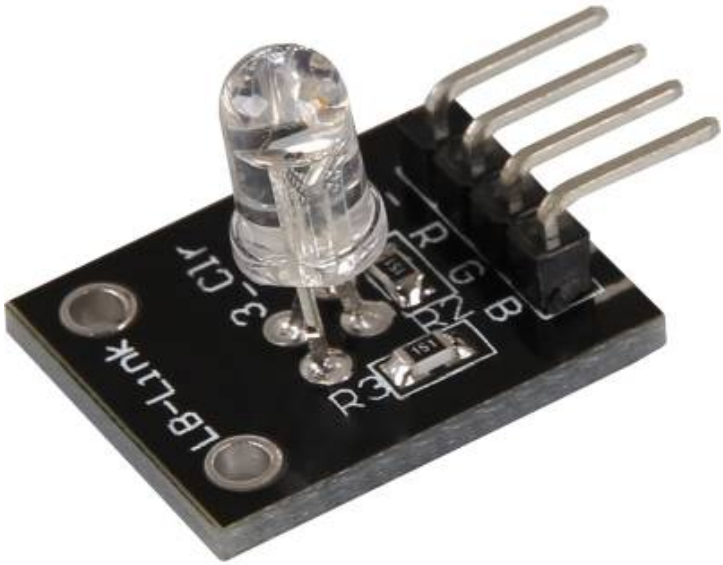
Zu starten mit dem Befehl:

```
sudo python KY-015-RPi_Kombi-Sensor_Temperatur_Feuchtigkeit.py
```


KY-016 RGB 5mm LED Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

LED-Modul welche eine rote, blaue und grüne LED beinhaltet. Diese sind mittels gemeinsamer Kathode miteinander verbunden. Je nach Eingangsspannung, werden Vorwiderstände benötigt

Vf [Rot]= 1,8V

Vf [Gruen,Blau]= 2,8V

If= 20mA

Vorwiderstände:

Rf (3,3V) [Grün]= 100Ω

Rf (3,3V) [Rot]= 180Ω

Rf (3,3V) [Blau]= 100Ω

[z.B. beim Einsatz mit ARM CPU-Kern basierten Mikrokontrollern wie Raspberry-Pi]

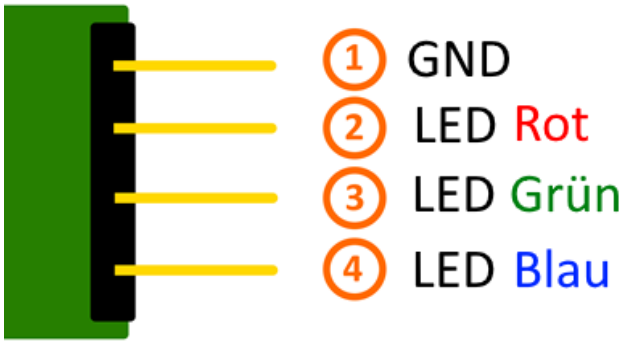
Rf (5V) [Grün] = 100Ω

Rf (5V) [Rot] = 180Ω

Rf (5V) [Blau] = 100Ω

[z.B. beim Einsatz mit Atmel Atmega basierten Mikrokontrollern wie Arduino]

Pin-Belegung



Codebeispiel Arduino

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.

```
int Led_Rot = 10;
int Led_Gruen = 11;
int Led_Blau = 12;

void setup ()
{
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
  pinMode (Led_Blau, OUTPUT);
}

void loop () //Hauptprogrammschleife
{
  digitalWrite (Led_Rot, HIGH); // LED wird eingeschaltet
```



```
  digitalWrite (Led_Gruen, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Blau, LOW); // LED wird eingeschaltet
  delay (3000); // Wartemodus für 3 Sekunden

  digitalWrite (Led_Rot, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, HIGH); // LED wird eingeschaltet
  digitalWrite (Led_Blau, LOW); // LED wird eingeschaltet
  delay (3000); // Wartemodus für weitere drei Sekunden in denen die LEDs dann umgeschaltet werden

  digitalWrite (Led_Rot, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Blau, HIGH); // LED wird eingeschaltet
  delay (3000); // Wartemodus für weitere drei Sekunden in denen die LEDs dann umgeschaltet werden
}
```

Beispielprogramm ON/OFF Download:

KY-016_LED_ON-OFF.zip

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt.

```
int Led_Rot = 10;
int Led_Gruen = 11;
int Led_Blau = 12;

int val;

void setup () {
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
  pinMode (Led_Blau, OUTPUT);
}
void loop () {
   // Innerhalb einer For-Schleife werden den drei LEDs verschiedene PWM-Werte uebergeben
   // Dadurch entsteht ein Farbverlauf, in dem sich durch das Vermischen unterschiedlicher
   // Helligkeitstufen der beiden integrierten LEDs, unterschiedliche Farben entstehen
   for (val = 255; val> 0; val--)
      {
       analogWrite (Led_Blau, val);
       analogWrite (Led_Gruen, 255-val);
       analogWrite (Led_Rot, 128-val);
       delay (1);
   }
   // In der zweiten For-Schleife wird der Farbverlauf rückwärts durchgegangen
   for (val = 0; val <255; val++)
      {
```



```
      analogWrite (Led_Blau, val);
      analogWrite (Led_Gruen, 255-val);
      analogWrite (Led_Rot, 128-val);
      delay (1);
   }
}
```

Beispielprogramm PWM Download:

KY-016_PWM.zip

Anschlussbelegung Arduino:

LED Rot	= [Pin 10]
LED Grün	= [Pin 11]
LED Blau	= [Pin 12]
Sensor GND	= [Pin GND]

Codebeispiel Raspberry Pi

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_ROT = 6
LED_GRUEN = 5
LED_BLAU = 4

GPIO.setup(LED_ROT, GPIO.OUT, initial= GPIO.LOW)
GPIO.setup(LED_GRUEN, GPIO.OUT, initial= GPIO.LOW)
GPIO.setup(LED_BLAU, GPIO.OUT, initial= GPIO.LOW)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
                        print("LED ROT 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.HIGH) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_BLAU,GPIO.LOW) #LED wird eingeschaltet
                        time.sleep(3) # Wartemodus fuer 4 Sekunden
                        print("LED GRUEN 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.HIGH) #LED wird eingeschaltet
                        GPIO.output(LED_BLAU,GPIO.LOW) #LED wird eingeschaltet
                        time.sleep(3) #Wartemodus fuer 3 Sekunden
                        print("LED BLAU 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_BLAU,GPIO.HIGH) #LED wird eingeschaltet
```



```
                        time.sleep(3) #Wartemodus fuer 3 Sekunden

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm ON/OFF Download

Zu starten mit dem Befehl:

```
sudo python KY016_RPI_ON-OFF.py
```

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt. Im Raspberry Pi ist nur ein Hardware-PWM Channel uneingeschränkt auf die GPIO-Pins hinausgeführt, weswegen im vorliegenden Beispiel auf Software-PWM zurückgegriffen wird.

```
# Benoetigte Module werden importiert und eingerichtet
import random, time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_Rot = 6
LED_Gruen = 5
LED_Blau = 4

# Set pins to output mode
GPIO.setup(LED_Rot, GPIO.OUT)
GPIO.setup(LED_Gruen, GPIO.OUT)
GPIO.setup(LED_Blau, GPIO.OUT)

Freq = 100 #Hz

# Die jeweiligen Farben werden initialisiert.
ROT = GPIO.PWM(LED_Rot, Freq)
GRUEN = GPIO.PWM(LED_Gruen, Freq)
BLAU = GPIO.PWM(LED_Blau, Freq)
ROT.start(0)
GRUEN.start(0)
BLAU.start(0)

# Diese Funktion generiert die eigentliche Farbe
# Mittels der jeweiligen Farbvariable, kann die Farbintensitaet geaendert werden
# Nachdem die Farbe eingestellt wurde, wird mittels "time.sleep" die Zeit definiert,
# wie lang die besagte Farbe angezeigt werden soll

def LED_Farbe(Rot, Gruen,Blau, pause):
    ROT.ChangeDutyCycle(Rot)
```



```
    GRUEN.ChangeDutyCycle(Gruen)
    BLAU.ChangeDutyCycle(Blau)
    time.sleep(pause)

    ROT.ChangeDutyCycle(0)
    GRUEN.ChangeDutyCycle(0)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife:
# Diese hat die Aufgabe fuer jede einzelne Farbe eine eigene Variable zu erstellen
# und mittels einer For-Schleife die Farbintensitaet jeder einzelnen Farbe von 0-100% zu druchlaufen
# Durch die Mischungen der verschiedenen Helligkeitsstufen der jeweiligen Farben
# entsteht somit ein Farbverlauf
try:
    while True:
        for x in range(0,2):
            for y in range(0,2):
                for z in range(0,2):
                    print (x,y,z)
                    for i in range(0,101):
                        LED_Farbe((x*i),(y*i),(z*i),.02)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm PWM Download:

KY-016_RPi_PWM.zip

Zu starten mit dem Befehl:

```
sudo python KY-016_RPi_PWM.py
```

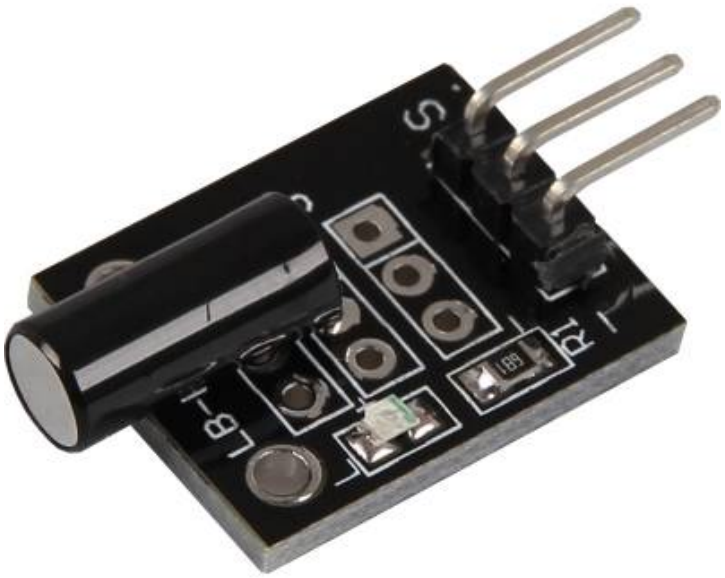
Anschlussbelegung Raspberry Pi:

LED Rot	= GPIO6 [Pin 22]
LED Grün	= GPIO5 [Pin 18]
LED Blau	= GPIO4 [Pin 16]
Sensor GND	= Masse [Pin 6]

KY-017 Neigungsschalter Modul

Inhaltsverzeichnis

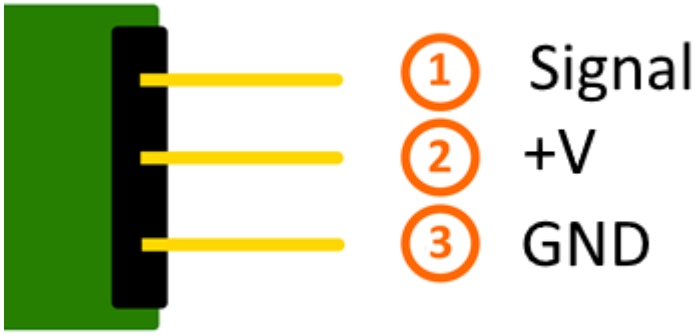
Bild



Technische Daten / Kurzbeschreibung

Je nach Neigung, schließt ein Schalter die Eingangspins kurz.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]
Sensor +V	= [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino_withoutPullUP.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (fallende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal	=	GPIO24	[Pin 18]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

SensorTest_RPi_withoutPullUP.zip

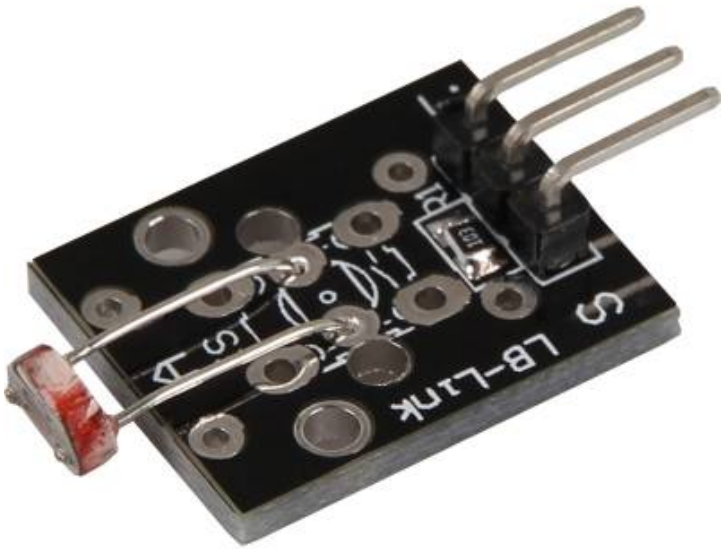
Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi_withoutPullUP.py
```


KY-018 Fotowiderstand Modul

Inhaltsverzeichnis

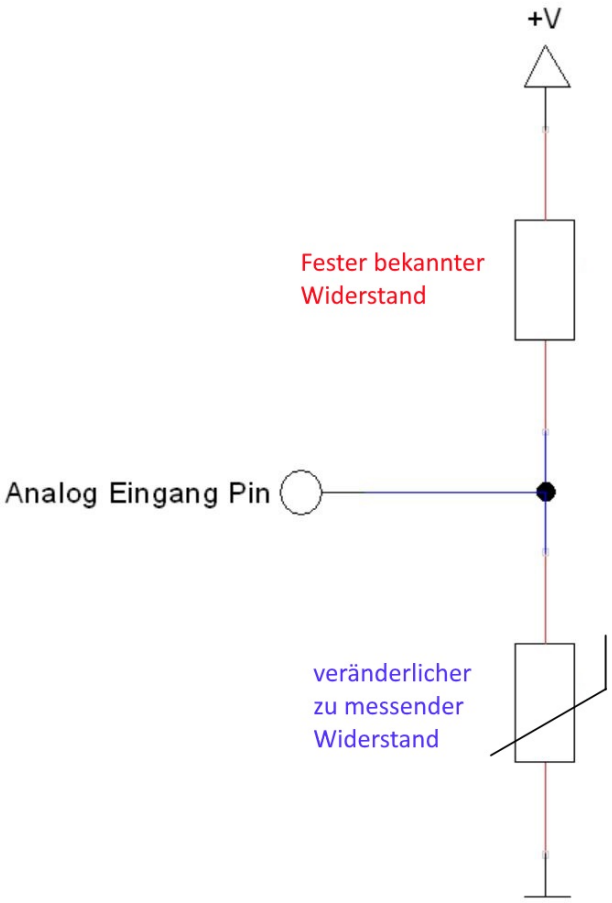
Bild



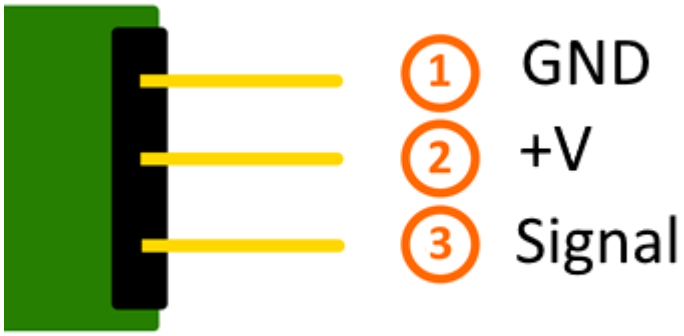
Technische Daten / Kurzbeschreibung

Beinhaltet einen LDR-Widerstand, dessen Widerstandswert bei hellerer Umgebung kleiner wird.

Dieser Widerstand lässt sich mit Hilfe eines Spannungsteilers bestimmen, wo sich eine bekannte Spannung über einen bekannten und einen unbekannten (veränderlichen) Widerstand aufteilt. Mittels dieser gemessenen Spannung lässt sich dann der Widerstand berechnen - die genaue Berechnung ist in den unten stehenden Codebeispielen enthalten.



Pin-Belegung



Codebeispiel Arduino

Das Programm misst den aktuellen Spannungswert am Sensor, berechnet aus diesen und dem bekannten Serienwiderstand den aktuellen Widerstandswert des Sensors und gibt die Ergebnisse auf der serielle Ausgabe aus.

```
int sensorPin = A5; // Hier wird der Eingangs-Pin deklariert

// Serielle Ausgabe in 9600 Baud
void setup()
{
        Serial.begin(9600);
}

// Das Programm misst den aktuellen Spannungswert am Sensor,
// berechnet aus diesen und dem bekannten Serienwiderstand den aktuellen
// Widerstandswert des Sensors und gibt die Ergebnisse auf der serielle Ausgabe aus

void loop()
{
        // Aktueller Spannungswert wird gemessen...
        int rawValue = analogRead(sensorPin);
        float voltage = rawValue * (5.0/1023) * 1000;

        float resitance = 10000 * ( voltage / ( 5000.0 - voltage) );

        // ... und hier auf die serielle Schnittstelle ausgegeben
        Serial.print("Spannungswert:");       Serial.print(voltage); Serial.print("mV");
        Serial.print(", Widerstandswert:"); Serial.print(resitance);Serial.println("Ohm");
        Serial.println("----------------------------------------");

        delay(500);
}
```

Anschlussbelegung Arduino:

Sensor GND	= [Pin GND]
Sensor +V	= [Pin 5V]
Sensor Signal	= [Pin A5]

Beispielprogramm Download

Single_Analog_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm misst den aktuellen Spannungswert am Sensor, berechnet aus diesen und dem bekannten Serienwiderstand den aktuellen Widerstandswert des Sensors und gibt die Ergebnisse auf der serielle Ausgabe aus.

```
# coding=utf-8
#!/usr/bin/python


##################################################################################################
 ### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Single Analog Sensor - Raspberry Pi Python Code Example
### ##############################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os, math
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2
voltageMax = 3300 # maximal möglicher Spannungswert am ADC

# Adresszuweisung ADS1x15 ADC

ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
```



```
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
# sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

##################################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm misst mit Hilfe des ADS1115 ADC den aktuellen Spannungswert am ADC, berechnet
# aus diesem und den bekannten Widerstandswert des Serien-Vorwiderstands den aktuellen
# Widerstandswert des Sensors und gibt diese in der Konsole aus.

try:
        while True:
                #Aktueller Wert wird aufgenommen,...
                voltage = adc.readADCSingleEnded(adc_channel, gain, sps)

                # ... der Widerstand wird berechnet...
                resitance = 10000 * voltage/(voltageMax - voltage)

                # ... und beides hier in die Konsole ausgegeben
                print "Spannungswert:", voltage,"mV, Widerstand:", resitance,"Ω"
                print "--------------------------------------"

                # Delay
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

GND	= GND	[Pin 06 (RPi)]
+V	= 3,3V	[Pin 01 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 17]
GND	= Masse	[Pin 09]

SCL	=	GPIO03 / SCL	[Pin 05]
SDA	=	GPIO02 / SDA	[Pin 03]
A0	=	s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_Single_Analog_Sensor.zip

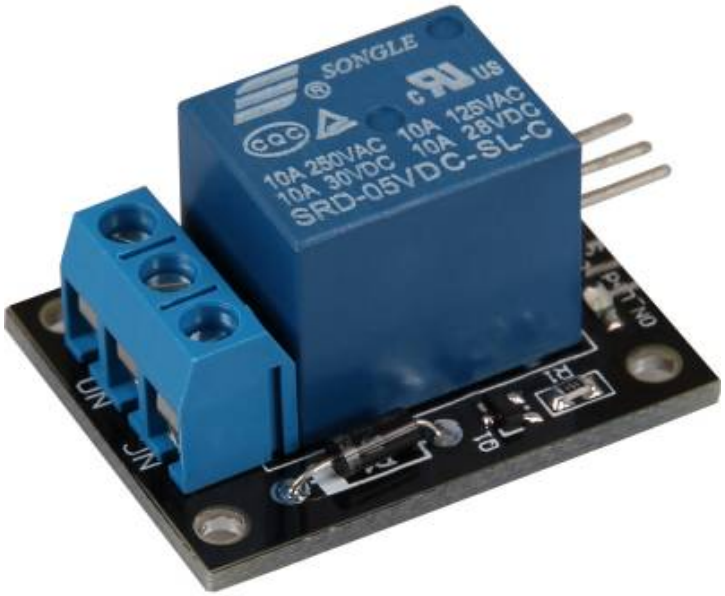
Zu starten mit dem Befehl:

```
sudo python RPi_Single_Analog_Sensor.py
```


KY-019 5V Relais Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

Spannungsbereich: 240VAC / 10A | 28VDC / 10AEin Relais zum schalten von höherer Spannungen mittels eines 5V Ausgangs.

!!!!! Warnung !!!!!

Das Arbeiten mit Spannungen >30v und vor allem bei Netzspannung (230V) kann zu körperlichen Schäden führen bis sogar tödlich sein.

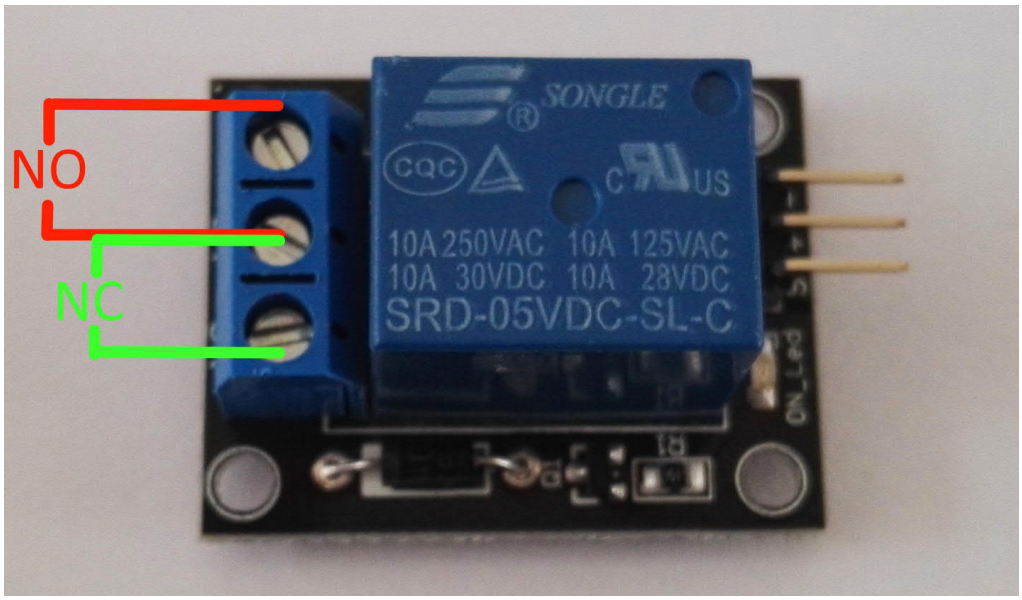
Wir raten dringend dazu erst mit höheren Spannungen zu Arbeiten, wenn Sie die fachliche Kompetenz dazu besitzten.

!!!!! Warnung !!!!!

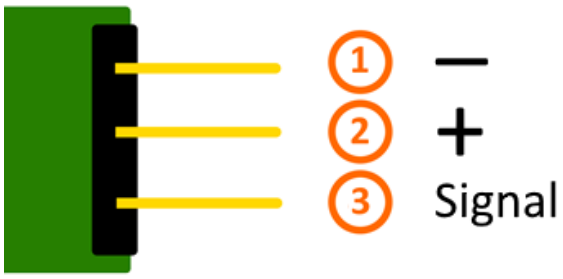
Die Ausgangsleiste des Relais besitzt zwei Ausgangsterminals:

- Das eine welches mit "NC" für "normally closed" gekennzeichnet ist, was bedeutet dass dieser Durchgang ohne elektrische Umschaltung am Relais standardmäßig kurzgeschlossen ist.

- Das andere welches mit "NO" für "normally open" gekennzeichnet ist, was bedeutet dass dieser Durchgang ohne elektrische Umschaltung am Relais standardmäßig offen bzw. getrennt ist.



Pin-Belegung



Codebeispiel Arduino

Das Programm bildet einen Blinker nach - es schaltet das Relais in vorher definierter Zeit (delayTime) zwischen den beiden Zuständen (bzw. Ausgangsterminals) um.


```
int relay = 10; // Hier wird der Pin deklariert, an dem das Relay angeschlossen ist

delayTime = 1 // Wert in Sekunden, wie lange zwischen den Umschaltungen gewartet werden soll

void setup ()
{
  pinMode (relay, OUTPUT); // Der Pin wird als Ausgang deklariert
}

// Das Programm bildet einen Blinker nach - es schaltet das Relais in vorher definierter
// Zeit (delayTime) zwischen den beiden Zuständen (bzw. Ausgangsterminals) um.
void loop ()
{
  digitalWrite (relay, HIGH); // "NO" ist nun kurzgeschlossen;
  delay (delayTime * 1000);
  digitalWrite (relay, LOW); // "NC" ist nun kurzgeschlossen;
  delay (delayTime * 1000);
}
```

Anschlussbelegung Arduino:

Sensor -	= [Pin GND]
Sensor +	= [Pin 5V]
Sensor Signal	= [Pin 10]

Beispielprogramm Download

KY-019_Relais.zip

Codebeispiel Raspberry Pi

Das Programm bildet einen Blinker nach - es schaltet das Relais in vorher definierter Zeit (delayTime) zwischen den beiden Zuständen (bzw. Ausgangsterminals) um.

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
# Hier wird die Pause (in Sekunden) zwischen dem Umschalten deklariert
delayTime = 1

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert RELAIS_PIN = 21
GPIO.setup(RELAIS_PIN, GPIO.OUT)
GPIO.output(RELAIS_PIN, False)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"


# Hauptprogrammschleife
try:
        while True:
            GPIO.output(RELAIS_PIN, True) # NO ist nun kurzgeschlossen
            time.sleep(delayTime)
            GPIO.output(RELAIS_PIN, False) # NC ist nun kurzgeschlossen
            time.sleep(delayTime)
```



```
# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Relais -	=	GND	[Pin 06]
Relais +	=	5V	[Pin 2]
Relais Signal	=	GPIO24	[Pin 18]

Beispielprogramm Download

KY-019_RPi_Relais.zip

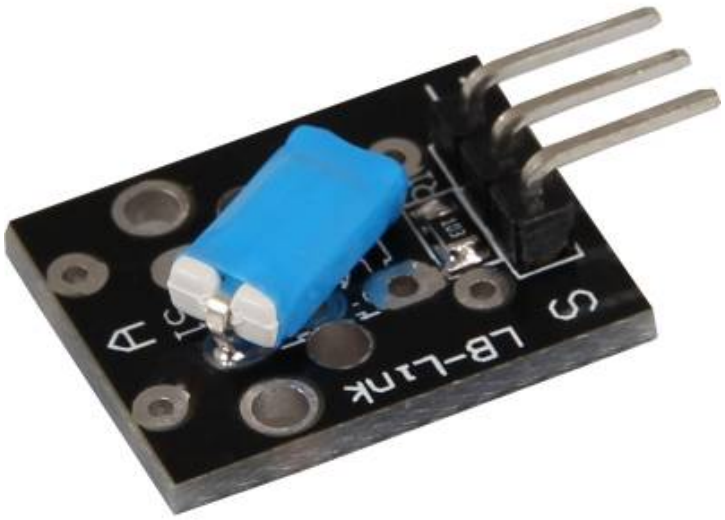
Zu starten mit dem Befehl:

```
sudo python KY-019_RPi_Relais.py
```


KY-020 Neigungs-Schalter Modul

Inhaltsverzeichnis

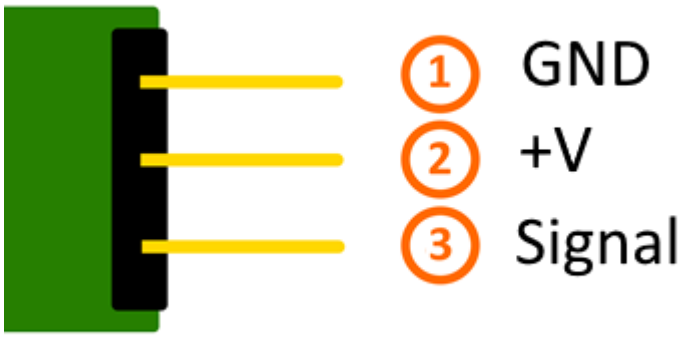
Bild



Technische Daten / Kurzbeschreibung

Je nach Neigung, schließt ein Schalter die Eingangspins kurz.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]
Sensor +V	= [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino_withoutPullUP.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (fallende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal	=	GPIO24	[Pin 18]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

SensorTest_RPi_withoutPullUP.zip

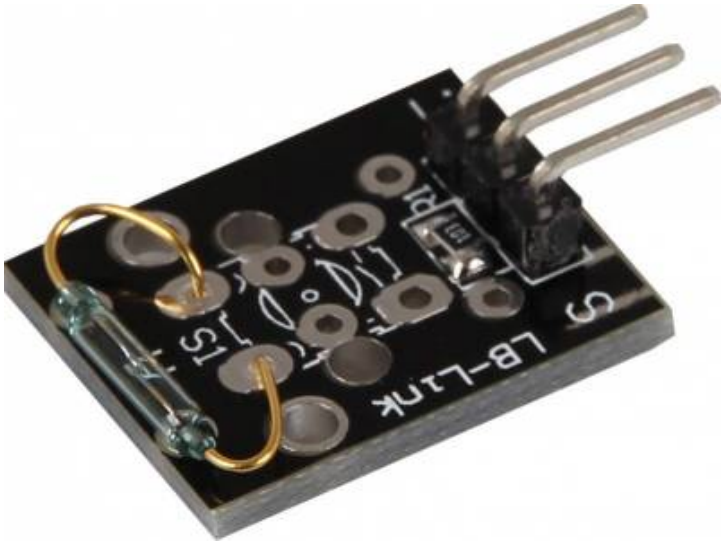
Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi_withoutPullUP.py
```


KY-021 Mini Magnet Reed Modul

Inhaltsverzeichnis

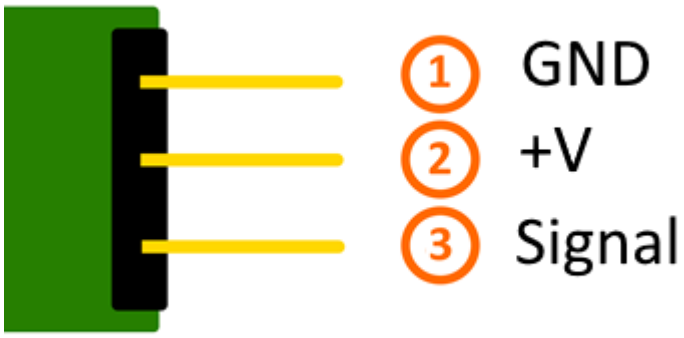
Bild



Technische Daten / Kurzbeschreibung

Wird ein Magnetfeld detektiert, so werden die beiden Eingangspins kurzgeschlossen.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]
Sensor +V	= [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino_withoutPullUP.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (fallende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal	=	GPIO24	[Pin 18]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

SensorTest_RPi_withoutPullUP.zip

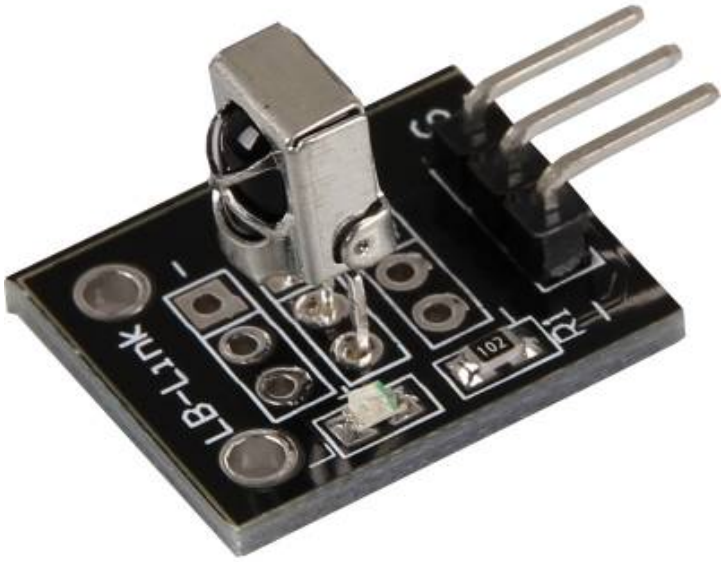
Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi_withoutPullUP.py
```


KY-022 Infrarot Receiver Modul

Inhaltsverzeichnis

Bild

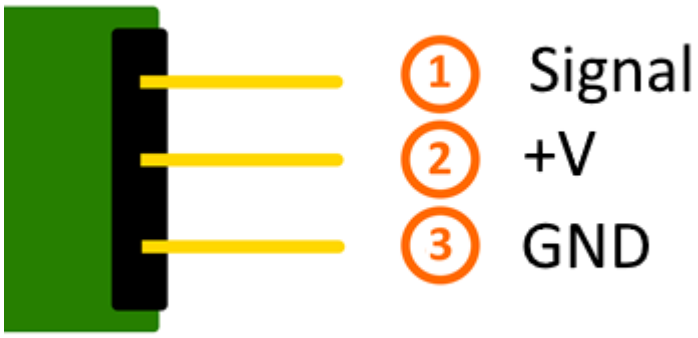


Technische Daten / Kurzbeschreibung

Trägerfrequenz: 38kHz Kann Infrarotsignale empfangen und gibt diese am Signalausgang als digitale Abfolge aus.

Zusätzlich blinkt die auf dem Modul integrierte LED kurz auf, wenn ein Infrarot-Signal detektiert wurde.

Pin-Belegung



Codebeispiel Arduino

Mithilfe der beiden Sensormodule KY-005 und KY-022 lässt sich ein Infrarot-Fernbedienung + Infrarot Receiver System aufbauen. Hierzu werden neben den zwei Modulen auch zwei einzelne Arduinos benötigt. Der eine fungiert hierbei dann als Sender und der andere empfängt die Signale und gibt diese dann in der seriellen Konsole aus.

Für das folgende Codebeispiel wird eine zusätzliche Library benötigt:

- [Arduino-IRremote] von Ken Shirriff | veröffentlicht unter LGPL

Die Library ist im Paket enthalten und muss vor dem Start der Arduino IDE in den "library"-Ordner kopiert werden.

Diesen finden Sie standardmäßig unter dem folgenden Pfad Ihrer Windows-Installation:

C:\Benutzer\[Benutzername]\Dokumente\Arduino\libraries

Bei Infrarot-Sendesystemen, gibt es verschiedene Protokolle, in denen die Daten versendet werden können. In dem folgenden Beispiel wird für das versenden das RC5 Protokoll verwendet - die verwendete Library "Arduino-IRremote" kümmert sich eigenständig um die Konvertierung in die richtige Datenfolge. Es gibt innerhalb der Library jedoch auch andere Protokolle/Kodierungen - diese sind in der Dokumentation/Code der Library gekennzeichnet.

Code für den Empfänger:

```
// Arduino-IRremote Iibrary wird hinzugefuegt
#include <IRremote.h>

// Hier kann der entsprechende Eingangspin für den Signalausgang
```



```
// des KY-022 deklariert werden
int RECV_PIN = 11;

// Arduino-IRremote Library wird initialisiert
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Infrarot-Receiver wird gestartet
}

// Hauptprogrammschleife
void loop() {

  // Es wird geprüft ob am Recveiver ein Signal eingegangen ist
  if (irrecv.decode(&results)) {
     // Bei Signaleingang wird das empfangene und dekodierte Signal
    // in der serriellen Konsole ausgegeben
    Serial.println(results.value, HEX);
    irrecv.resume();
  }
}
```

Code für den Sender:

```
//Arduino-IRremote Library wird hinzugefügt...
#include <IRremote.h>

//...und hier initialisiert
IRsend irsend;

// Die Einstellungen für den Ausgang werden von der Library übernommen
// Die entsprechenden Ausgänge unterscheiden sich je nach verwendeten Arduino
// Arduino UNO:  Ausgang = D3
// Arduino MEGA: Ausgang = D9
// Eine komplette Auflistung der entsprechenden Ausgänge finden Sie unter
// http://z3t0.github.io/Arduino-IRremote/
void setup()
{
}

// Hauptprogrammschleife
void loop() {
        // Der Sender verschickt in diesem Beispiel das Signal A90
        // (in hexdezimaler Form) in der Kodierung "RC5". Dieses wird dreimal
        //  hintereinander gesendet und danach eine Pause für 5 Sekunden eingelegt
        for (int i = 0; i < 3; i++) {
// [0xA90] zu versendetes Signal | [12] Bit-Länge des zu versendeten Signals (hex A90=1010 1001 0000)
                irsend.sendRC5(0xA90, 12);
                delay(40);
        }
        delay(5000); //Zwischen den Sendeimpulsen gibt es eine Pause von 5 Sekunden
}
```

Beispielprogramm Download:

Arduino_Fernbedienung.zip

Anschlussbelegung Arduino 1 [Empfänger]:

KY-022

Signal = [Pin 11]
+V = [Pin 5V]
GND = [Pin GND]

Anschlussbelegung Arduino 2 [Sender]:

KY-005

Signal = [Pin 3 (Arduino Uno) | Pin 9 (Arduino Mega)]
GND+Widerstand = [Pin GND*]
GND = [Pin GND]

- *Nur wenn Vorwiderstand auf dem Modul verlötet wurde und nicht vor dem Modul geschaltet ist

Codebeispiel Raspberry Pi

Der Raspberry Pi besitzt mit seiner fortschrittlichen Prozessorarchitektur den Vorteil gegenüber dem Arduino, dass dieser ein komplettes Linux-Betriebssystem betreiben kann. Mit Hilfe eines Infrarot-Receivers können somit nicht nur einfache Datensignale ausgetaucht, sondern es können auch komplette Software-Programme wie z.B. die Mediencenter Software OpenElec per Fernbedienung bedient werden.

Für die Einrichtung eines Infrarot-Steuerungssystem, bietet sich unter Linux bietet die Software "lirc" an (veröffentlicht unter der LGPL - Website). Im Folgenden zeigen wir auf, wie lirc eingerichtet, eine Fernbedienung angelernt werden kann und per Infrarotdiode die erlernten Signale per Infrarot versendet werden können (um z.B. aus dem Raspberry Pi eine per Software steuerbare Infrarot Fernbedienung zu machen).

Zu diesem Zwecke werden die Module KY-005 als Infrarot-Transmitter und KY-022 als Infrarot Receiver angewendet.

Anschlussbelegung Raspberry Pi:

KY-005

Signal = GPIO17 [Pin 11]
GND+Widerstand = GND* [Pin 9]
GND = GND [Pin 6]

- *Nur wenn Vorwiderstand auf dem Modul verlötet wurde und nicht vor dem Modul geschaltet ist

KY-022

Signal = GPI18 [Pin 12]
+V = 3,3V [Pin 17]
GND = GND [Pin 25]

Lirc Installation

Als erstes öffnen wir auf dem Desktop ein Terminal oder verbinden wir uns per SSH mit dem Raspberry Pi. Dort geben Sie den folgenden Befehl ein, um lirc auf den Raspberry Pi zu installieren:

```
sudo apt-get install lirc -y
```

[Hierzu muss der Raspberry Pi mit dem Internet verbunden sein]

Damit das lirc Modul direkt zum Start des Betriebssystem verfügbar ist, müssen folgende Zeilen am Ende der Datei "/boot/config.txt" hinzugefügt werden:

```
dtoverlay=lirc-rpi,gpio_in_pin=18,gpio_out_pin=17,gpio_in_pull=up
```

Hierbei definiert "gpio_in_pin=18" den Eingangspin für den IR-Receiver, sowie "gpio_out_pin=17" den Ausgangspin für den IR-Transmitter.

Die Datei kann mit folgenden Befehl editiert werden:

```
sudo nano /boot/config.txt
```

Mit der Tastenfolge [Strg+X -> Y -> Enter] kann die Datei, nach dem hinzufügen der Zeile am unteren Ende, gespeichert und geschlossen werden.

Auch die Datei "/etc/lirc/hardware.conf" muss modifiziert werden. Hierbei ist der Befehl...

```
sudo nano /etc/lirc/hardware.conf
```

... zu verwenden. Hier müssen die die folgenden Zeilen modifiziert werden. An den entsprechenden Stellen muss dann ...

```
DRIVER="UNCONFIGURED"
--->>
DRIVER="default"
```

```
DEVICE=""
--->>
DEVICE="/dev/lirc0"
```

```
MODULES=""
--->>
MODULES="lirc_rpi"
```

...geändert werden.

Die modifizierte Datei muss dann wie folgt aussehen:


```
# /etc/lirc/hardware.conf
#
# Arguments which will be used when launching lircd
LIRCD_ARGS=""

#Don't start lircmd even if there seems to be a good config file
#START_LIRCMD=false

#Don't start irexec, even if a good config file seems to exist.
#START_IREXEC=false

#Try to load appropriate kernel modules
LOAD_MODULES=true

# Run "lircd --driver=help" for a list of supported drivers.
DRIVER="default"
# usually /dev/lirc0 is the correct setting for systems using udev
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"

# Default configuration files for your hardware if any
LIRCD_CONF=""
LIRCMD_CONF=""
```

Danach starten wir den Raspberry Pi mit dem folgenden Befehl neu:

```
sudo reboot
```

IR-Receiver Test

Um den angeschlossenen Receiver zu testen, muss vorab lirc mit dem Befehl...

```
sudo /etc/init.d/lirc stop
```

beendet werden. Danach kann mit...

```
mode2 -d /dev/lirc0
```

...getestet werden, ob am Raspberry Pi Signale detektiert werden können. Hierzu nehmen Sie eine Infrarot Fernbedienung und drücken eine beliebige Taste - es sollten Zeilen in der folgenden Form auftauchen:

```
space 95253
pulse 9022
space 2210
pulse 604
space 95246
pulse 9019
space 2211
pulse 601
space 95252
pulse 9019
space 2210
```



```
pulse 603
space 95239
pulse 9020
space 2208
pulse 603
...
```

Mit dem Befehl...

```
sudo /etc/init.d/lirc start
```

... kann der lirc-Dienst wieder gestartet werden.

Fernbedienung anlernen

Um eine Infrarot-Fernbedienung im lirc System zu registrieren, muss für die Fernbedienung die Datei "/etc /lirc/lircd.conf" konfiguriert werden. In dieser sind die jeweiligen Zuordnungen von Befehl zu empfangenen Infrarot-Codes gespeichert.

Um eine entsprechend richtig formatierte lircd.conf erstellen zu können, gibt es in der lirc-Software einen Assistenten, der die Datei automatisch erstellt. Die Vorgehensweise für diese Konfiguration ist wie folgt:

Zu aller erst, muss der lirc-Service beendet werden

```
sudo /etc/init.d/lirc stop
```

Mit dem folgenden Befehl starten wir dann den Assistenten:

```
irrecord -d /dev/lirc0 ~/MeineFernbedienung.conf
```

Dieser Assistent führt vorab eine erste Initialisierung der Fernbedienung durch - in dieser müssen mehrere Tasten abwechselnd gedrückt werden, damit das lirc System die entsprechende Kodierung der Fernbedienung erlernen kann. Bitte folgen Sie dazu die entsprechenden Anweisungen des Assistenten. Nach der Initialisierung fragt dann der Assistent nach dem Namen der Knopfzuordnung, die mit einem neuen Infrarotcode aufgezeichnet werden soll. Sie können sich hierzu die Knopfzuordnungen aus der folgenden Datei auswählen

FernbedienungsCodes.txt

Diese müssen dann in den Assistenten eingegeben und mit Enter bestätigt werden. Hiernach startet dann die Aufzeichnung des Infrarot-Codes, für die ausgewählte Taste.

Beispiel: [KEY_0] eingeben -> mit Enter bestätigen -> Taste "0" auf der Fernbedienung drücken -> warten bis der Assistent die Aufnahme bestätigt.

Sollen keine weiteren Tasten angelernt werden, so kann der Assistent mit der Enter-Taste beendet werden. Hiernach ist die Konfigurationsdatei erstellt, jedoch muss noch ein Name für die aufgezeichnete Fernbedienung vergeben werden. Hierzu öffnen wir die Datei im Editor mit:

```
sudo nano ~/MeineFernbedienung.conf
```


Hier kann dann die Zeile 17 von

```
name /home/pi/MeineFernbedienung.conf
```

in

```
name MeineFernbedienung
```

geändert werden. Beachten Sie hierbei keine Leerzeichen und Sonderzeichen innerhalb des Namens zu verwenden. Mit der Tastenfolge [Strg+X -> Y -> Enter] kann die Datei, nach dem hinzufügen der Zeile am unteren Ende, gespeichert und geschlossen werden.

Nach erstellen der Konfigurationsdatei können Sie vorab für die original lircd.conf mit folgenden Befehl ein Backup erstellen:

```
sudo mv /etc/lirc/lircd.conf /etc/lirc/lircd.conf.bak
```

und mit dem Befehl...

```
sudo cp ~/MeineFernbedienung.conf /etc/lirc/lircd.conf
```

...wird die vorab neu erstellte Konfigurationsdatei für das lirc-System eingesetzt.

Nun kann mit...

```
sudo /etc/init.d/lirc start
```

...das lirc System wieder gestartet werden.

Ab nun ist die neu angelernte Fernbedienung im System registriert und kann in kompatibler Software , wie z. B. dem Mediencenter OpenElec verwendet werden. Alternativ kann mit dem Befehl...

```
irw
```

...die Zuordnungen und die Funktion der Fernbedienung getestet werden.

Will man die Fernbedienung nun z.B. in OpenElec verwenden, kann es je nach Konfiguration der Mediensoftware sein, dass lirc in den Optionen von OpenElec vorab noch aktiviert werden muss.

Befehle senden mit dem Infrarot Transmitter

Möchte man mit dem Raspberry Pi nun Geräte wie z.B. den Fernseher per Infrarot steuern, so können nun die vorab angelernten Befehle mit Hilfe des Infrarot Transmitters wieder versendet werden. So lässt sich z.B. eine Software gesteuerte Infrarot-Steuerung aufbauen oder einzelne Geräte mittels Netzwerk/Internet ein und Ausschalten.

Zuerst überprüfen wir mit folgendem Befehl...


```
irsend LIST MeineFernbedienung ""
```

...welche Zuordnungen für die jeweilige gespeicherte Fernbedienung verfügbar sind.

Nun können wir z.B. den Befehl [KEY_0] versenden, indem wir den folgenden Befehl verwenden:

```
irsend SEND_ONCE MeineFernbedienung KEY_0
```

Auf dem Fernseher bzw. dem entsprechenden Empfänger-Endgerät, sollte sich nun eine Reaktion zeigen. Zu dem o.g. Befehl gibt es auch die Variation, dass das Signal wiederholend ausgegeben wird

```
irsend SEND_START MeineFernbedienung KEY_0
```

Hier nach wird der gesendete Code [KEY_0] so oft wiederholt bis mit...

```
irsend SEND_STOP MeineFernbedienung KEY_0
```

...wieder beendet wird. Dies wird z.B. bei Befehlen wie "Lautstärke Hoch" oder "Helligkeit runter" angewendet.

KY-023 Joystick Modul (XY-Achsen)

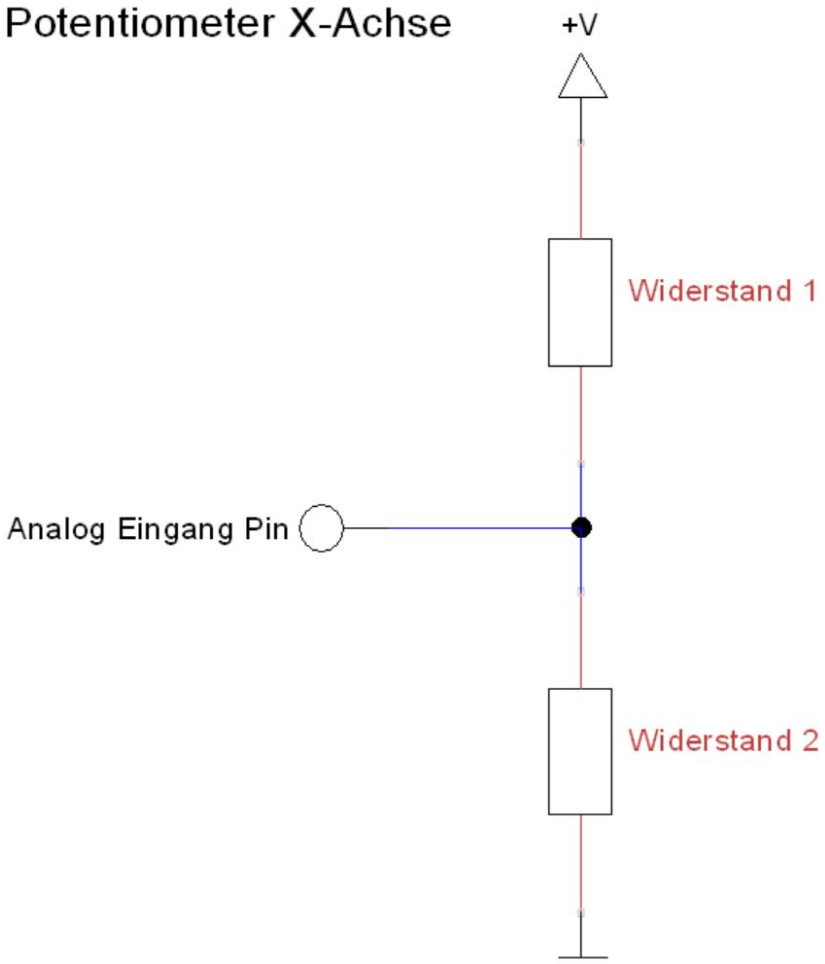
Inhaltsverzeichnis

Bild

Technische Daten / Kurzbeschreibung

X und Y Position des Joysticks, werden als analoge Spannung auf den Ausgangspins ausgegeben.

In diesem Joystick wurde für die X-Achse, sowie für die Y-Achse, ein eigenes Potentiometer verbaut. Diese ergeben einen Spannungsteiler, wie dieser im folgendem Bild aufgezeigt wird

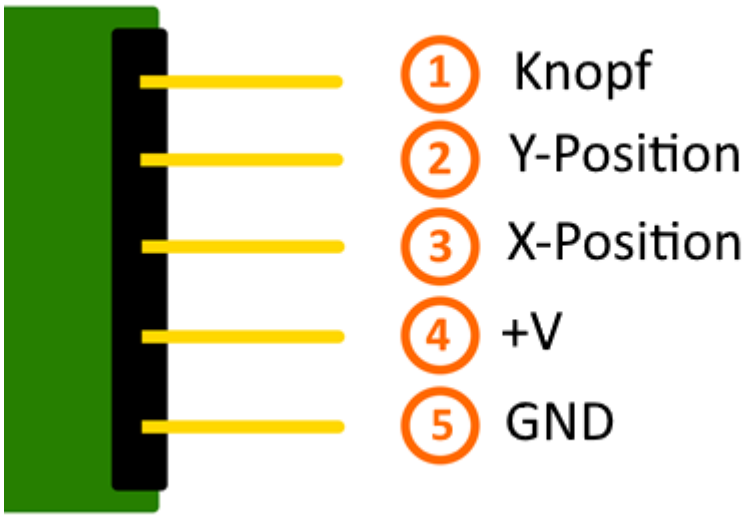


Im Ruhezustand befinde
der Mitte, so dass Wide
auch die angelegte Spa
beide verteilt - z.B Mess

Wird jetzt die Position v
verändert, so ändern sic
abhängig zur aktuellen
eine Richtung so wird W
Widerstand 2 größer, ge
so wird dann Widerstan
kleiner.

Je nachdem wie sich die
aufteilen, resultiert dies
Spannungswert, den ma
Widerständen (beim Pot
messen und somit die P
kann.

Pin-Belegung



Codebeispiel Arduino

Das Programm liest die aktuellen Werte der Eingang-Pins, konvertiert diese in eine Spannung (0-1023 -> 0V-5V) und gibt diese auf der seriellen Ausgabe aus.

```
// Deklaration und Initialisierung der Eingang-Pins
int JoyStick_X = A0; // X-Achse-Signal
int JoyStick_Y = A1; // Y-Achse-Signal
int Button = 3; // Knopf

void setup ()
{
  pinMode (JoyStick_X, INPUT);
  pinMode (JoyStick_Y, INPUT);
  pinMode (Button, INPUT);

  // Da der Knopf das Signal beim druecken auf Masse zieht,
  // schalten wir hiermit den PullUp-Widerstand ein
  digitalWrite(Button, HIGH);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float x, y;
  int Knopf;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  x = analogRead (JoyStick_X) * (5.0 / 1023.0);
  y = analogRead (JoyStick_Y) * (5.0 / 1023.0);
  Knopf = digitalRead (Button);
```



```
  //... und an dieser Stelle ausgegeben
  Serial.print ("X-Achse:"); Serial.print (x, 4);  Serial.print ("V, ");
  Serial.print ("Y-Achse:"); Serial.print (y, 4);  Serial.print ("V, ");
  Serial.print ("Knopf:");

  if(Knopf==1)
  {
      Serial.println (" nicht gedrueckt");
  }
  else
  {
      Serial.println (" gedrueckt");
  }
  delay (200);
}
```

Anschlussbelegung Arduino:

Knopf	=	[Pin 3]
Y-Position	=	[Pin A1]
X-Position	=	[Pin A0]
Sensor +V	=	[Pin 5V]
Sensor GND	=	[Pin GND]

Beispielprogramm Download

KY-023_Joystick_Modul.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

```
#
#!/usr/bin/python
# coding=utf-8

########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### KY-023 Joystick Module - Raspberry Pi Python Code Example
### ####################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC

ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
x_adc_channel = 0    # Channel 0 für die x-Achse
y_adc_channel = 1    # Channel 1 für die y-Achse
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten ADC handelt es
# sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

Button_PIN = 24
GPIO.setup(Button_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```



```
# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                  x   =   adc.readADCSingleEnded(x_adc_channel,   gain,   sps)
                  y   =   adc.readADCSingleEnded(y_adc_channel,   gain,   sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Button_PIN) == True:
                        print "X-Achse:", x,"mV, ","Y-Achse:", y,"mV, Button: nicht gedrückt"
                else:
                        print "X-Achse:", x, "mV, ", "Y-Achse:", y, "mV, Button: gedrückt"
                print "----------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)



except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor KY-023

Knopf	=	GPIO24	[Pin 18 (RPi)]
Y-Position	=	Analog 1	[Pin A1 (ADS1115 - KY-053)]
X-Position	=	Analog 0	[Pin A0 (ADS1115 - KY-053)]
+V	=	3,3V	[Pin 1 (RPi)]
GND	=	Masse	[Pin 6 (RPi)]

ADS1115 - KY-053:

VDD	=	3,3V	[Pin 01]
GND	=	Masse	[Pin 09]
SCL	=	GPIO03 / SCL	[Pin 05 (RPi)]
SDA	=	GPIO02 / SDA	[Pin 03 (RPi)]
A0	=	s.o.	[Sensor: X-Position (KY-023)]
A1	=	s.o.	[Sensor: Y-Position (KY-023)]

Beispielprogramm Download

KY-023_RPi_JoystickModule.zip

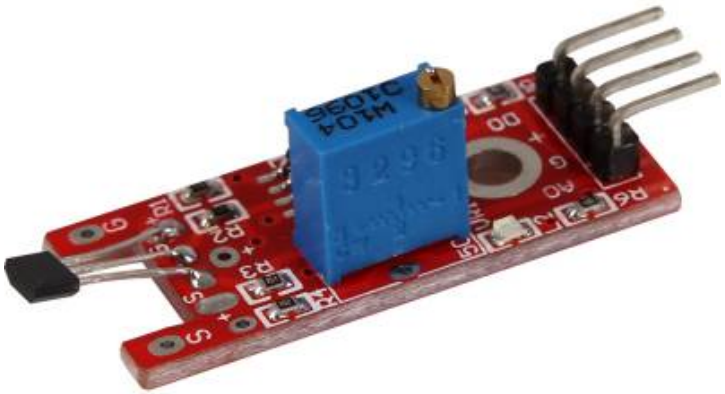
Zu starten mit dem Befehl:

```
sudo python KY-023_RPi_JoystickModule.py
```


KY-024 Linear magnetic Hall Sensor

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

Chipsatz: A3141 | OP-Verstärker: LM393

Das Magnetfeld wird vom Sensor gemessen und als analoger Spannungswert ausgegeben. Die Empfindlichkeit des Sensors kann mittels des Potentiometers geregelt werden.

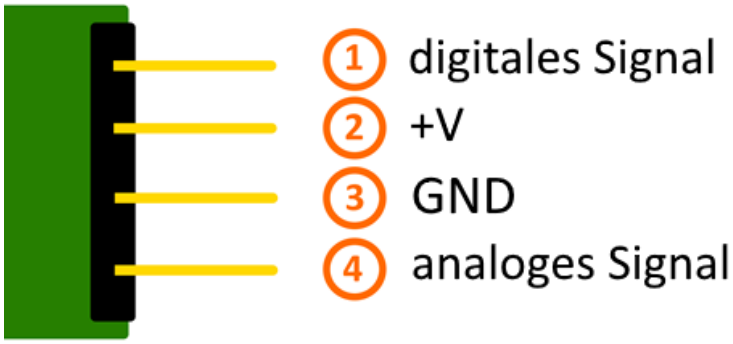
Digitaler Ausgang: Wird ein Magnetfeld erkannt, so wird hier ein Signal ausgegeben

Analoger Ausgang: Direkter Messwert der Sensoreinheit

LED1: Zeigt an, dass der Sensor mit Spannung versorgt ist

LED2: Zeigt an, dass ein Magnetfeld detektiert wurde

Pin-Belegung

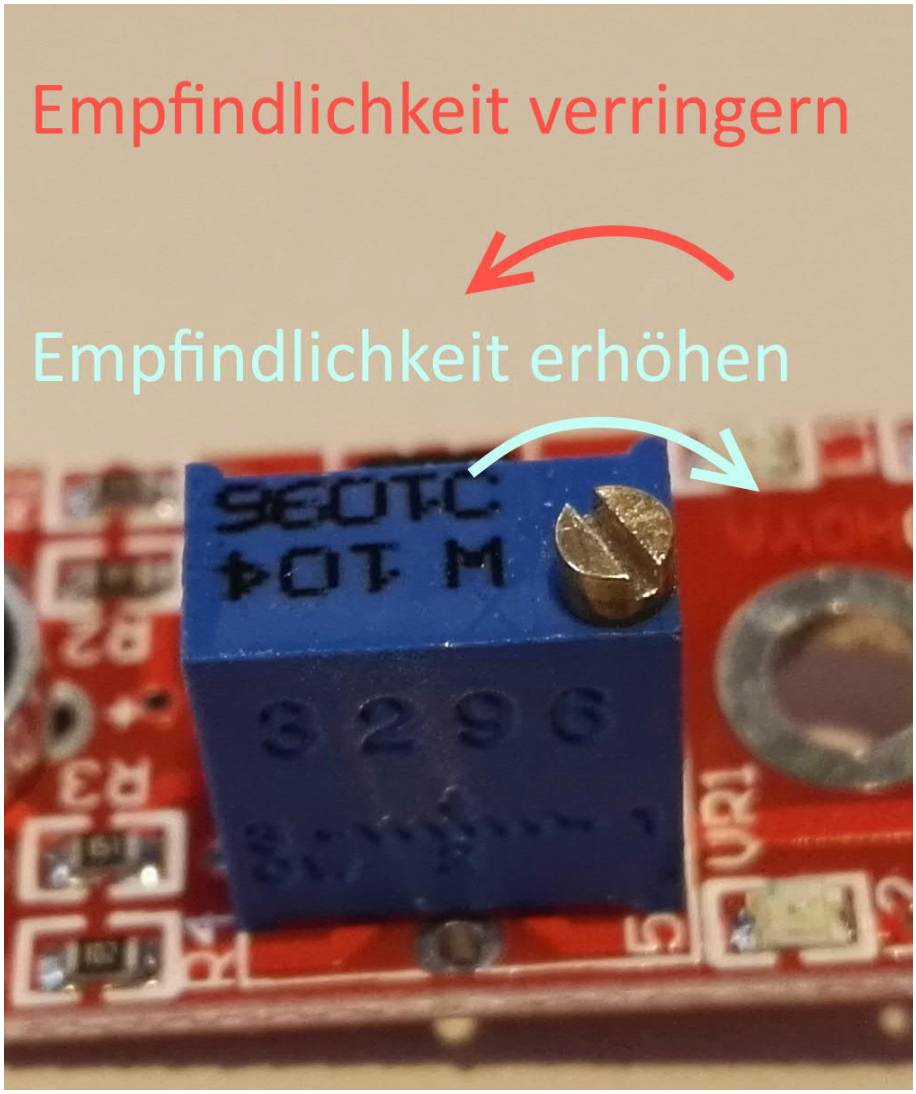


Funktionsweise des Sensors

Dieser Sensor besitzt auf seiner Platine drei funktionelle Bestandteile. Die ist die Sensoreinheit vorne am Modul, welche das aktuelle Umfeld physikalisch misst und als analoges Signal auf die zweite Einheit, dem Verstärker, ausgibt. Dieser verstärkt das Signal abhängig vom eingestellten Widerstand am Drehpotentiometer und leitet es auf den analogen Ausgang des Moduls.

Hierbei ist zu beachten: Das Signal ist invertiert; wird ein hoher Wert gemessen, so resultiert dies in einen niedrigeren Spannungswert am analogen Ausgang.

Die dritte Einheit stellt einen Komparator dar, welcher den digitalen Ausgang und die LED schaltet, wenn das Signal unter einen bestimmten Wert fällt. Mittels des Drehpotentiometers kann somit die Empfindlichkeit eingestellt werden, wie es im folgenden Bild aufgezeigt wird:



Dieser Sensor gibt somit keine absoluten Werte aus (z.B. genau gemessene Temperatur in °C oder Magnetfeldstärke in mT) , sondern es handelt sich hierbei um eine Relativ-Messung: Man definiert hierbei einen Grenzwert relativ zur gegebenen normalen Umweltsituation und es wird ein Signal ausgegeben was weiterverarbeitet werden kann, falls dieser Grenzwert überschritten bzw. ein anderer Zustand als der Normalfall eingetreten ist.

Dieses Verhalten eignet sich hervorragend zur Temperaturüberwachung (KY-028), Näherungsschalter (KY-024, KY-025, KY-036), Alarmüberwachungen (KY-037, KY-038) oder Drehgeber (KY-026).

Codebeispiel Arduino

Das Programm liest den aktuellen Spannungswert aus, der am analogen Ausgang gemessen werden kann, und gibt diesen auf der seriellen Schnittstelle aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.


```
// Deklaration und Initialisierung der Eingang-Pins
int Analog_Eingang = A0; // X-Achse-Signal
int Digital_Eingang = 3; // Knopf

void setup ()
{
  pinMode (Analog_Eingang, INPUT);
  pinMode (Digital_Eingang, INPUT);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float Analog;
  int Digital;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  Analog = analogRead (Analog_Eingang) * (5.0 / 1023.0);
  Digital = digitalRead (Digital_Eingang);

  //... und an dieser Stelle ausgegeben
  Serial.print ("Analoger Spannungswert:"); Serial.print (Analog, 4);  Serial.print ("V, ");
  Serial.print ("Grenzwert:");

  if(Digital==1)
  {
      Serial.println (" erreicht");
  }
  else
  {
      Serial.println (" noch nicht erreicht");
  }
  Serial.println ("-----------------------------------------------------------------");
  delay (200);
}
```

Anschlussbelegung Arduino:

digitales Signal	= [Pin 3]
+V	= [Pin 5V]
GND	= [Pin GND]
analoges Signal	= [Pin 0]

Beispielprogramm Download

Ard_Analoger_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Analog Sensor + ADS1115 ADC - Raspberry Pi Python Code Example
### ######################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC
```

```
ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
#ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

# Hier waehlt man den Eingangs-Pin des digitalen Signals aus
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)

##################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                analog = adc.readADCSingleEnded(adc_channel, gain, sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Digital_PIN) == False:
                        print "Analoger Spannungswert:",analog,"mV","Grenzwert: noch nicht erreicht"
                else:
                        print "Analoger Spannungswert:", analog, "mV, ", "Grenzwert: erreicht"
                print "--------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

digitales Signal	= GPIO 24	[Pin 18 (RPi)]
+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 01]
GND	= Masse	[Pin 09]
SCL	= GPIO03 / SCL	[Pin 05]
SDA	= GPIO02 / SDA	[Pin 03]
A0	= s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_AnalogSensor.zip

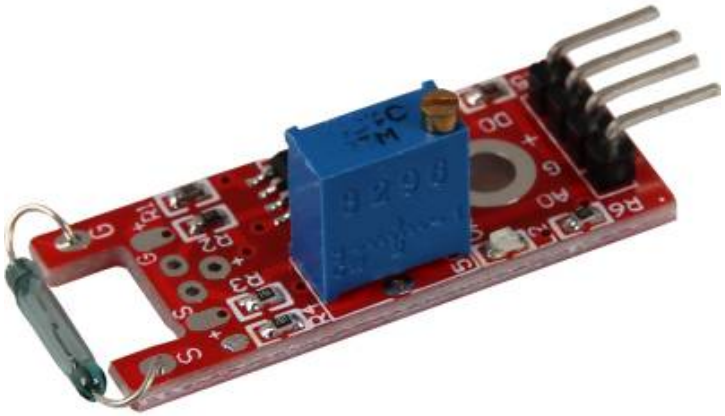
Zu starten mit dem Befehl:

```
sudo python RPi_AnalogSensor.py
```


KY-025 Reed Modul

Inhaltsverzeichnis

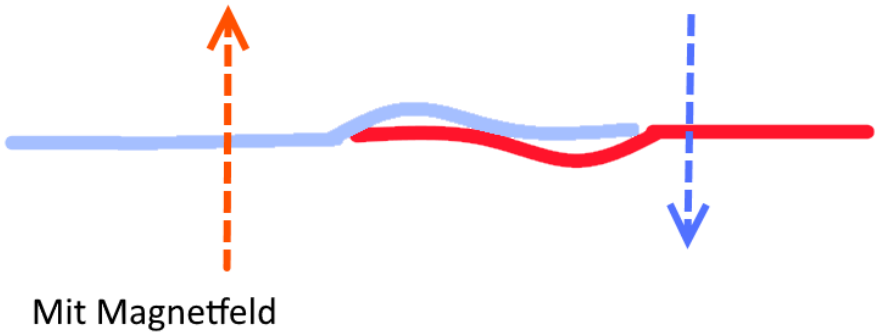
Bild



Technische Daten / Kurzbeschreibung

Wird ein Magnetfeld detektiert, so wird dies am digitalen Ausgang ausgegeben.

Reed-Kontakte haben die Eigenschaft, dass deren zwei dünnen Kontaktfedern, die sich im Inneren des Glasröhrchen befinden, aufeinander zu bewegen, falls ein Magnetfeld in der Nähe ist.



So wird ein elektrischer Kontakt geschlossen, welcher dann das Signal durchschaltet.

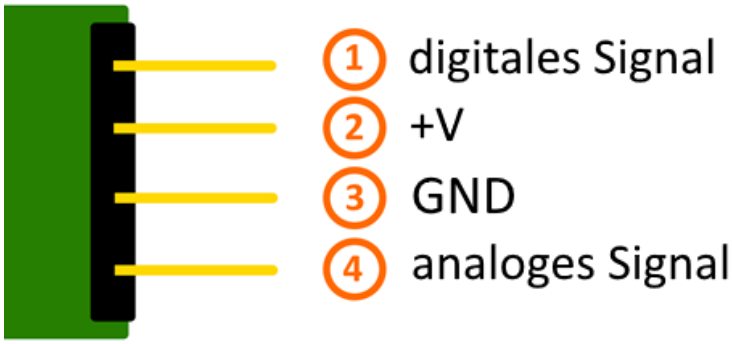
Digitaler Ausgang: Wird ein Magnetfeld erkannt, so wird hier ein Signal ausgegeben

Analoger Ausgang: Direkter Messwert der Sensoreinheit

LED1: Zeigt an, dass der Sensor mit Spannung versorgt ist

LED2: Zeigt an, dass ein Magnetfeld detektiert wurde

Pin-Belegung

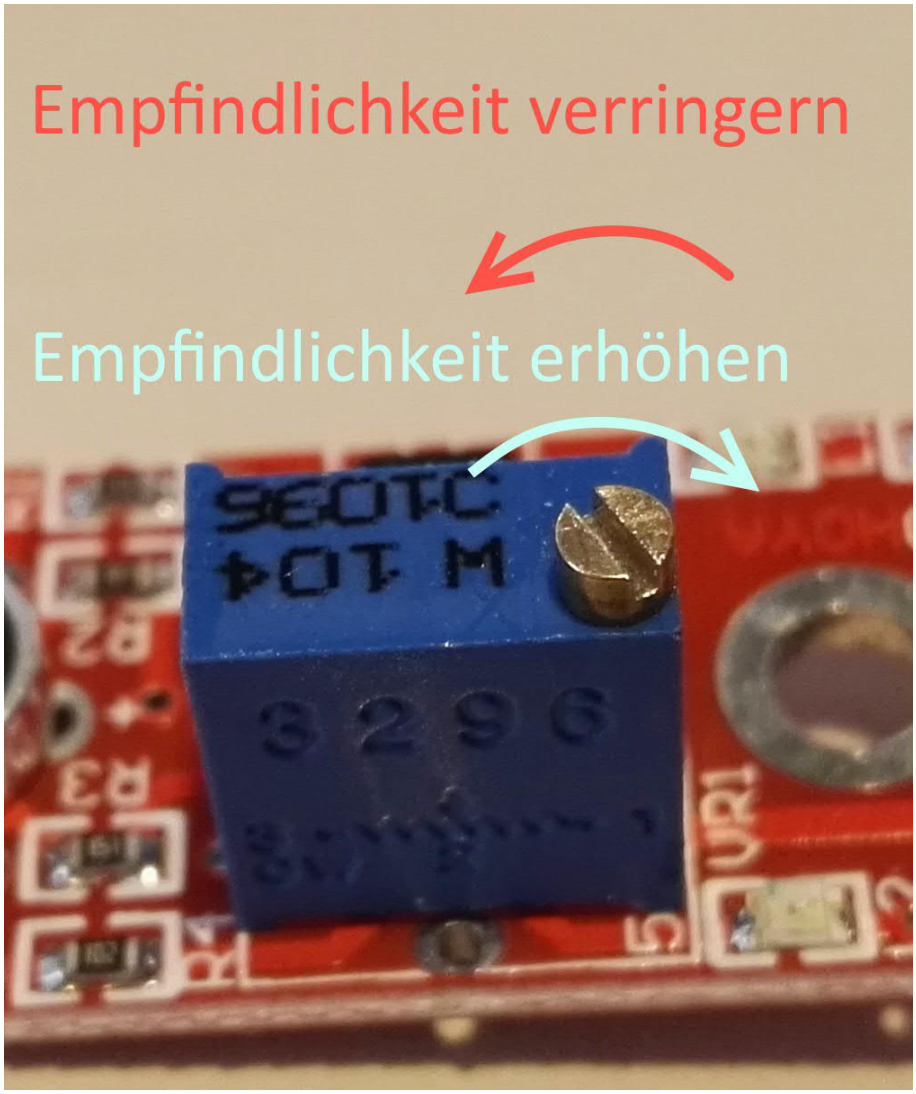


Funktionsweise des Sensors

Dieser Sensor besitzt auf seiner Platine drei funktionelle Bestandteile. Die ist die Sensoreinheit vorne am Modul, welche das aktuelle Umfeld physikalisch misst und als analoges Signal auf die zweite Einheit, dem Verstärker, ausgibt. Dieser verstärkt das Signal abhängig vom eingestellten Widerstand am Drehpotentiometer und leitet es auf den analogen Ausgang des Moduls.

Hierbei ist zu beachten: Das Signal ist invertiert; wird ein hoher Wert gemessen, so resultiert dies in einen niedrigeren Spannungswert am analogen Ausgang.

Die dritte Einheit stellt einen Komparator dar, welcher den digitalen Ausgang und die LED schaltet, wenn das Signal unter einen bestimmten Wert fällt. Mittels des Drehpotentiometers kann somit die Empfindlichkeit eingestellt werden, wie es im folgenden Bild aufgezeigt wird:



Dieser Sensor gibt somit keine absoluten Werte aus (z.B. genau gemessene Temperatur in °C oder Magnetfeldstärke in mT) , sondern es handelt sich hierbei um eine Relativ-Messung: Man definiert hierbei einen Grenzwert relativ zur gegebenen normalen Umweltsituation und es wird ein Signal ausgegeben was weiterverarbeitet werden kann, falls dieser Grenzwert überschritten bzw. ein anderer Zustand als der Normalfall eingetreten ist.

Dieses Verhalten eignet sich hervorragend zur Temperaturüberwachung (KY-028), Näherungsschalter (KY-024, KY-025, KY-036), Alarmüberwachungen (KY-037, KY-038) oder Drehgeber (KY-026).

Codebeispiel Arduino

Das Programm liest den aktuellen Spannungswert aus, der am analogen Ausgang gemessen werden kann, und gibt diesen auf der seriellen Schnittstelle aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
// Deklaration und Initialisierung der Eingang-Pins
int Analog_Eingang = A0; // X-Achse-Signal
int Digital_Eingang = 3; // Knopf

void setup ()
{
  pinMode (Analog_Eingang, INPUT);
  pinMode (Digital_Eingang, INPUT);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float Analog;
  int Digital;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  Analog = analogRead (Analog_Eingang) * (5.0 / 1023.0);
  Digital = digitalRead (Digital_Eingang);

  //... und an dieser Stelle ausgegeben
  Serial.print ("Analoger Spannungswert:"); Serial.print (Analog, 4);Serial.print("V, ");
  Serial.print ("Grenzwert:");

  if(Digital==1)
  {
      Serial.println (" erreicht");
  }
  else
  {
      Serial.println (" noch nicht erreicht");
  }
  Serial.println ("-----------------------------------------------------------------");
  delay (200);
}
```


Anschlussbelegung Arduino:

digitales Signal	= [Pin 3]
+V	= [Pin 5V]
GND	= [Pin GND]
analoges Signal	= [Pin 0]

Beispielprogramm Download

Ard_Analoger_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Analog Sensor + ADS1115 ADC - Raspberry Pi Python Code Example
### ######################################################################################################
```

```
# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC

ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

# Hier waehlt man den Eingangs-Pin des digitalen Signals aus
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)
########################
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                analog = adc.readADCSingleEnded(adc_channel, gain, sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Digital_PIN) == False:

                        print "Analoger Spannungswert:",analog,"mV,","Grenzwert: noch nicht erreicht"
```



```
                        print "Analoger Spannungswert:" analog,"mV","Grenzwert: noch nicht erreicht"
                else:
                        print "Analoger Spannungswert:", analog, "mV, ", "Grenzwert: erreicht"
                print "---------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)



except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

digitales Signal	= GPIO 24	[Pin 18 (RPi)]
+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 01]
GND	= Masse	[Pin 09]
SCL	= GPIO03 / SCL	[Pin 05]
SDA	= GPIO02 / SDA	[Pin 03]
A0	= s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_AnalogSensor.zip

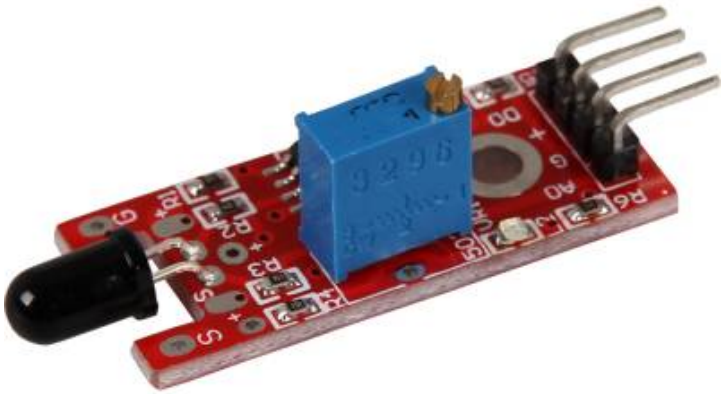
Zu starten mit dem Befehl:

```
sudo python RPi_AnalogSensor.py
```


KY-026 Flamen-Sensor Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

Die angebrachte Fotodiode ist empfindlich auf den Spektralbereich von Licht, welches von offenen Flamen erzeugt wird.

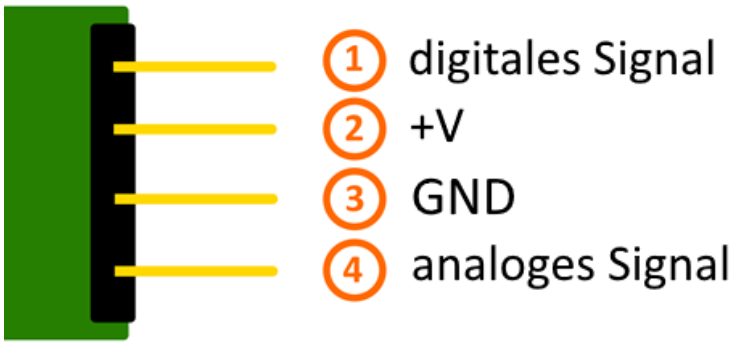
Digitaler Ausgang: Wird eine Flame erkannt, wird hier ein Signal ausgegeben

Analoger Ausgang: Direkter Messwert der Sensoreinheit

LED1: Zeigt an, dass der Sensor mit Spannung versorgt ist

LED2: Zeigt an, dass ein Magnetfeld detektiert wurde

Pin-Belegung

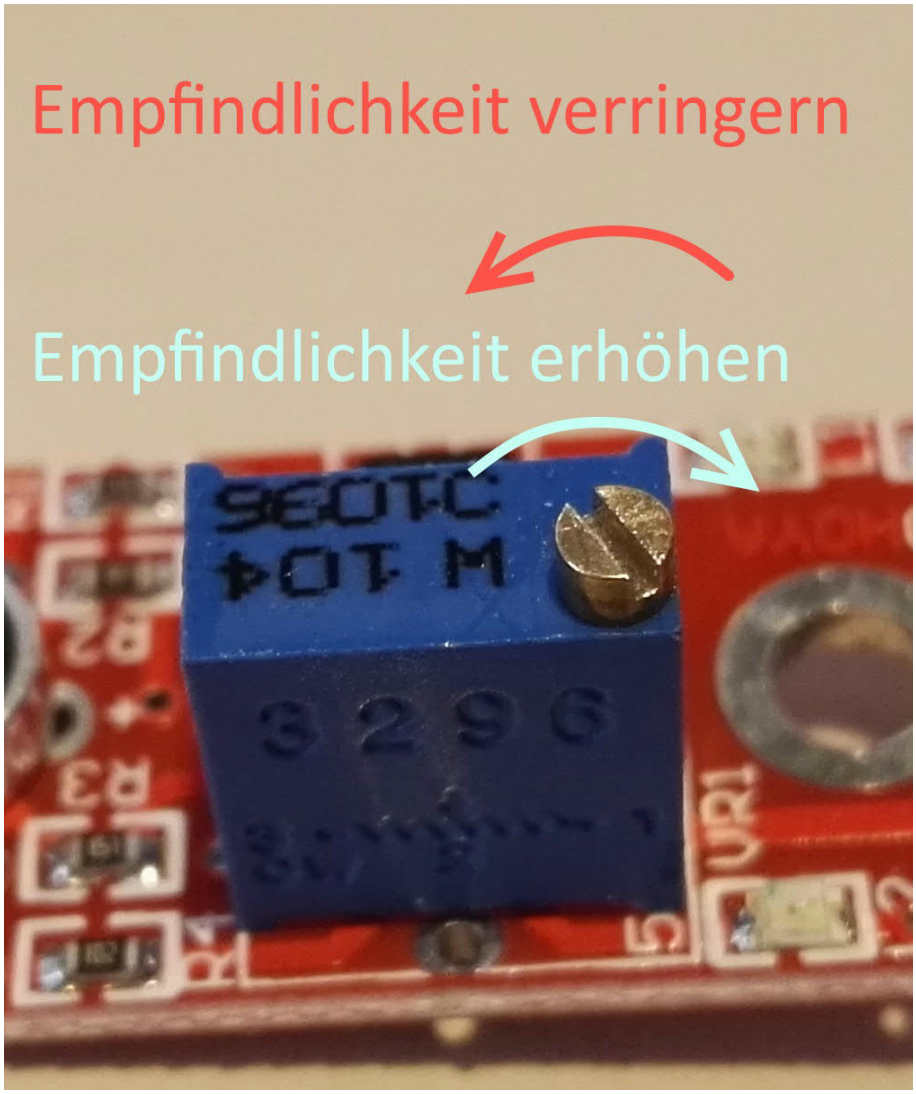


Funktionsweise des Sensors

Dieser Sensor besitzt auf seiner Platine drei funktionelle Bestandteile. Die ist die Sensoreinheit vorne am Modul, welche das aktuelle Umfeld physikalisch misst und als analoges Signal auf die zweite Einheit, dem Verstärker, ausgibt. Dieser verstärkt das Signal abhängig vom eingestellten Widerstand am Drehpotentiometer und leitet es auf den analogen Ausgang des Moduls.

Hierbei ist zu beachten: Das Signal ist invertiert; wird ein hoher Wert gemessen, so resultiert dies in einen niedrigeren Spannungswert am analogen Ausgang.

Die dritte Einheit stellt einen Komparator dar, welcher den digitalen Ausgang und die LED schaltet, wenn das Signal unter einen bestimmten Wert fällt. Mittels des Drehpotentiometers kann somit die Empfindlichkeit eingestellt werden, wie es im folgenden Bild aufgezeigt wird:



Dieser Sensor gibt somit keine absoluten Werte aus (z.B. genau gemessene Temperatur in °C oder Magnetfeldstärke in mT) , sondern es handelt sich hierbei um eine Relativ-Messung: Man definiert hierbei einen Grenzwert relativ zur gegebenen normalen Umweltsituation und es wird ein Signal ausgegeben was weiterverarbeitet werden kann, falls dieser Grenzwert überschritten bzw. ein anderer Zustand als der Normalfall eingetreten ist.

Dieses Verhalten eignet sich hervorragend zur Temperaturüberwachung (KY-028), Näherungsschalter (KY-024, KY-025, KY-036), Alarmüberwachungen (KY-037, KY-038) oder Drehgeber (KY-026).

Codebeispiel Arduino

Das Programm liest den aktuellen Spannungswert aus, der am analogen Ausgang gemessen werden kann, und gibt diesen auf der seriellen Schnittstelle aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.


```
// Deklaration und Initialisierung der Eingang-Pins
int Analog_Eingang = A0; // X-Achse-Signal
int Digital_Eingang = 3; // Knopf

void setup ()
{
  pinMode (Analog_Eingang, INPUT);
  pinMode (Digital_Eingang, INPUT);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float Analog;
  int Digital;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  Analog = analogRead (Analog_Eingang) * (5.0 / 1023.0);
  Digital = digitalRead (Digital_Eingang);

  //... und an dieser Stelle ausgegeben
  Serial.print ("Analoger Spannungswert:"); Serial.print (Analog, 4);Serial.print ("V, ");
  Serial.print ("Grenzwert:");

  if(Digital==1)
  {
      Serial.println (" erreicht");
  }
  else
  {
      Serial.println (" noch nicht erreicht");
  }
  Serial.println ("-----------------------------------------------------------------");
  delay (200);
}
```

Anschlussbelegung Arduino:

digitales Signal	= [Pin 3]
+V	= [Pin 5V]
GND	= [Pin GND]
analoges Signal	= [Pin 0]

Beispielprogramm Download

Ard_Analoger_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Analog Sensor + ADS1115 ADC - Raspberry Pi Python Code Example
### ######################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC

ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V
```



```
# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

# Hier waehlt man den Eingangs-Pin des digitalen Signals aus
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)

##########################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                analog = adc.readADCSingleEnded(adc_channel, gain, sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Digital_PIN) == False:
                        print "Analoger Spannungswert:",analog,"mV,","Grenzwert: noch nicht erreicht"
                else:
                        print "Analoger Spannungswert:", analog, "mV, ", "Grenzwert: erreicht"
                print "----------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

digitales Signal	= GPIO 24	[Pin 18 (RPi)]
+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	=	3,3V	[Pin 01]
GND	=	Masse	[Pin 09]
SCL	=	GPIO03 / SCL	[Pin 05]
SDA	=	GPIO02 / SDA	[Pin 03]
A0	=	s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_AnalogSensor.zip

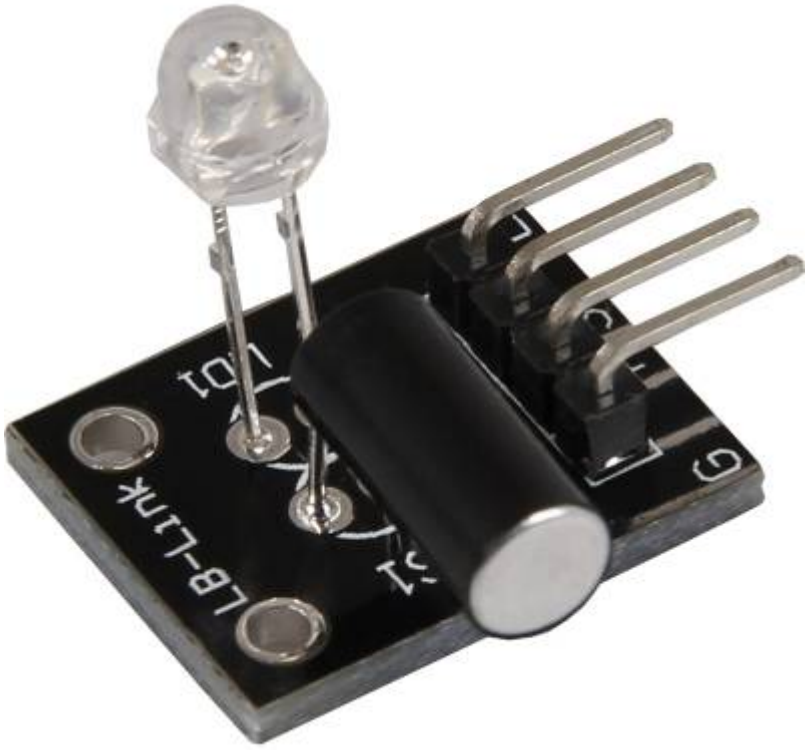
Zu starten mit dem Befehl:

```
sudo python RPi_AnalogSensor.py
```


KY-027 Magic Light Cup Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

Die LED wird bei Erschütterung an– oder ausgeschaltet. Das Signal, wann die LED an ist, wird an einen Signalausgang gegeben. Je nach Eingangsspannung, werden Vorwiderstände benötigt

Vorwiderstand:

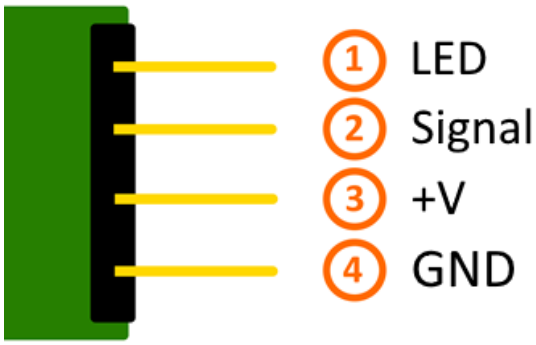
Rf (3,3V) [Rot]= 120Ω

[z.B. beim Einsatz mit ARM CPU-Kern basierten Mikrokontrollern wie Raspberry-Pi]

Rf (5V) [Rot] = 220Ω

[z.B. beim Einsatz mit Atmel Atmega basierten Mikrokontrollern wie Arduino]

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches die LED vom Magic-LightCup-Modul zum Leuchten bringt, wenn am Sensor eine Neigung detektiert wurde.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
  digitalWrite(Sensor, HIGH); // Aktivierung interner Pull-Up Widerstand
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED + = [Pin 13]

LED - = [Pin GND]
Sensor Signal = [Pin 10]
Sensor +V = [Pin 5V]
Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

In diesem Programmbeispiel Leuchtet die LED auf, wenn am Neigungsschalter, ein Signal detektiert wird.

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier werden die beiden Pins deklariert, an dem der Sensor und die LED angeschlossen sind,
LED_PIN = 24
Sensor_PIN = 23
GPIO.setup(Sensor_PIN, GPIO.IN)
GPIO.setup(LED_PIN, GPIO.OUT)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        GPIO.output(LED_PIN, True)

# Beim Detektieren eines Signals wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(Sensor_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=10)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)
                # Ausgang wird wieder zurueckgesetzt, falls der Neigungschalter
                 # wieder auf die andere Seite geneigt wird
                if GPIO.input(Sensor_PIN):
                        GPIO.output(LED_PIN,False)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

LED = GPIO24 [Pin 18]
Signal = GPIO23 [Pin 16]
+V = 3,3V [Pin 1]
GND = Masse [Pin 6]

Beispielprogramm Download

KY-027-RPi-MagicLightCup.zip

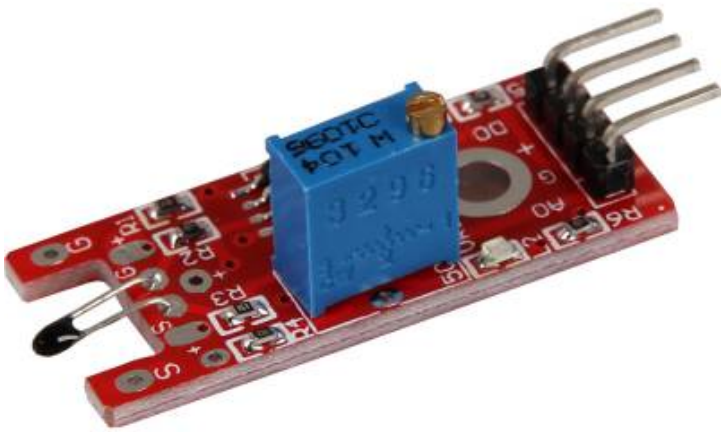
Zu starten mit dem Befehl:

```
sudo python KY-027-RPi-MagicLightCup.py
```


KY-028 Temperatur Sensor Modul (Thermistor)

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

Temperaturmessbereich: -55°C / +125°C Dieses Modul beinhaltet einen NTC Thermistor—dieser hat bei höherer Temperatur einen immer weniger werdenden Widerstandswert.

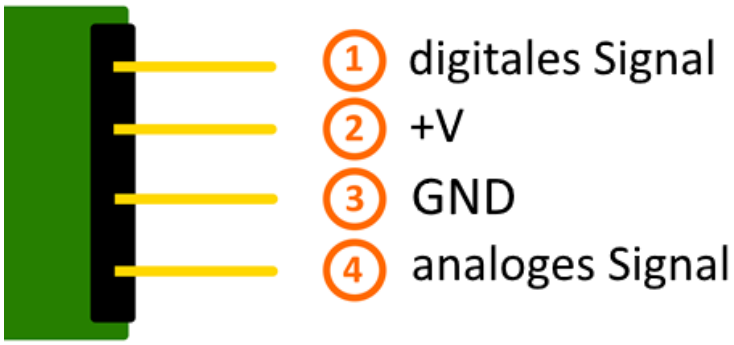
Digitaler Ausgang: Wird eine Temperatur über einen Grenzwert gemessen so wird dieses hier ausgegeben—der Grenzwert kann mittels des Potentiometers eingestellt werden

Analoger Ausgang: Direkter Messwert der Sensoreinheit

LED1: Zeigt an, dass der Sensor mit Spannung versorgt ist

LED2: Zeigt an, dass ein Magnetfeld detektiert wurde

Pin-Belegung

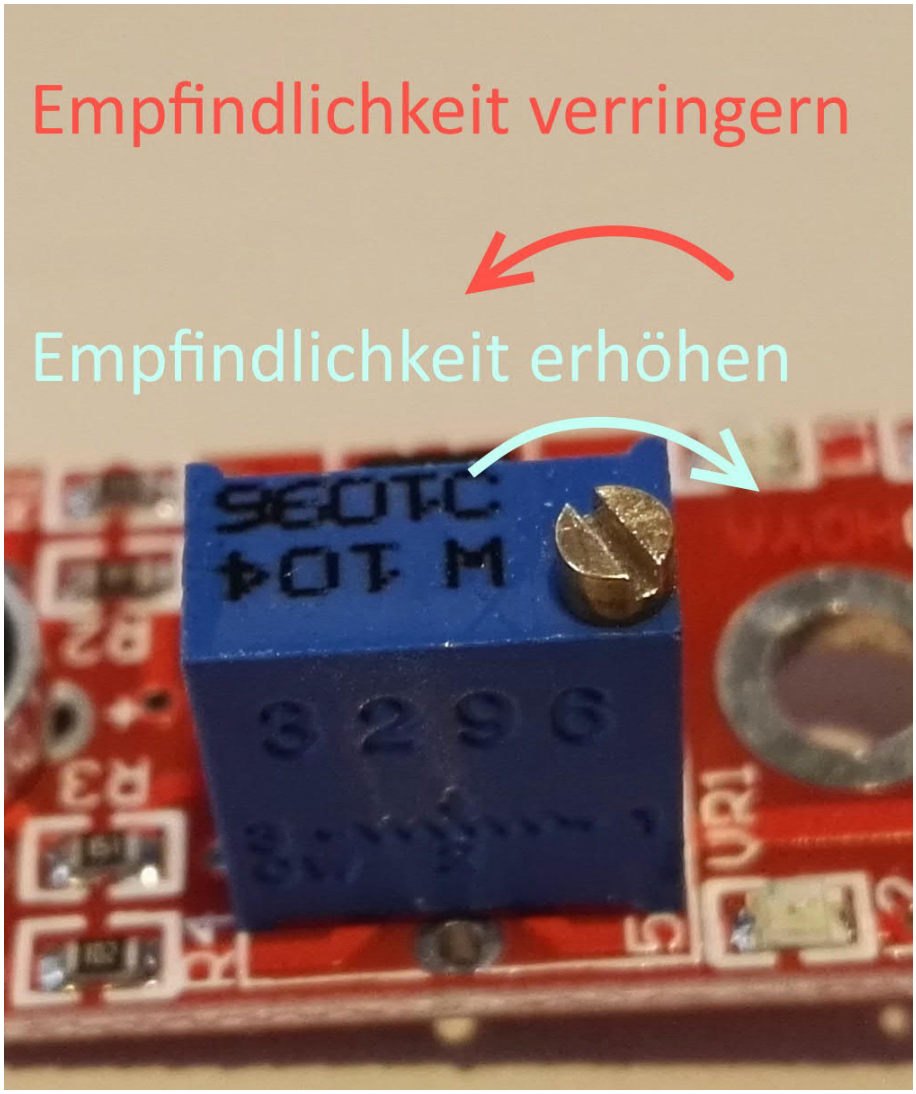


Funktionsweise des Sensors

Dieser Sensor besitzt auf seiner Platine drei funktionelle Bestandteile. Die ist die Sensoreinheit vorne am Modul, welche das aktuelle Umfeld physikalisch misst und als analoges Signal auf die zweite Einheit, dem Verstärker, ausgibt. Dieser verstärkt das Signal abhängig vom eingestellten Widerstand am Drehpotentiometer und leitet es auf den analogen Ausgang des Moduls.

Hierbei ist zu beachten: Das Signal ist invertiert; wird ein hoher Wert gemessen, so resultiert dies in einen niedrigeren Spannungswert am analogen Ausgang.

Die dritte Einheit stellt einen Komparator dar, welcher den digitalen Ausgang und die LED schaltet, wenn das Signal unter einen bestimmten Wert fällt. Mittels des Drehpotentiometers kann somit die Empfindlichkeit eingestellt werden, wie es im folgenden Bild aufgezeigt wird:



Dieser Sensor gibt somit keine absoluten Werte aus (z.B. genau gemessene Temperatur in °C oder Magnetfeldstärke in mT) , sondern es handelt sich hierbei um eine Relativ-Messung: Man definiert hierbei einen Grenzwert relativ zur gegebenen normalen Umweltsituation und es wird ein Signal ausgegeben was weiterverarbeitet werden kann, falls dieser Grenzwert überschritten bzw. ein anderer Zustand als der Normalfall eingetreten ist.

Dieses Verhalten eignet sich hervorragend zur Temperaturüberwachung (KY-028), Näherungsschalter (KY-024, KY-025, KY-036), Alarmüberwachungen (KY-037, KY-038) oder Drehgeber (KY-026).

Codebeispiel Arduino

Das Programm liest den aktuellen Spannungswert aus, der am analogen Ausgang gemessen werden kann, und gibt diesen auf der seriellen Schnittstelle aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.


```
// Deklaration und Initialisierung der Eingang-Pins
int Analog_Eingang = A0; // X-Achse-Signal
int Digital_Eingang = 3; // Knopf

void setup ()
{
  pinMode (Analog_Eingang, INPUT);
  pinMode (Digital_Eingang, INPUT);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float Analog;
  int Digital;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  Analog = analogRead (Analog_Eingang) * (5.0 / 1023.0);
  Digital = digitalRead (Digital_Eingang);

  //... und an dieser Stelle ausgegeben
  Serial.print ("Analoger Spannungswert:"); Serial.print (Analog, 4);Serial.print ("V, ");
  Serial.print ("Grenzwert:");

  if(Digital==1)
  {
      Serial.println (" erreicht");
  }
  else
  {
      Serial.println (" noch nicht erreicht");
  }
  Serial.println ("-----------------------------------------------------------------");
  delay (200);
}
```

Anschlussbelegung Arduino:

digitales Signal	= [Pin 3]
+V	= [Pin 5V]
GND	= [Pin GND]
analoges Signal	= [Pin 0]

Beispielprogramm Download

Ard_Analoger_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Analog Sensor + ADS1115 ADC - Raspberry Pi Python Code Example
### ######################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC
```

```
ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

# Hier waehlt man den Eingangs-Pin des digitalen Signals aus
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)

##############################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                analog = adc.readADCSingleEnded(adc_channel, gain, sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Digital_PIN) == False:
                        print "Analoger Spannungswert:",analog,"mV,","Grenzwert: noch nicht erreicht"
                else:
                        print "Analoger Spannungswert:", analog, "mV, ", "Grenzwert: erreicht"
                print "---------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

digitales Signal	= GPIO 24	[Pin 18 (RPi)]
+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 01]
GND	= Masse	[Pin 09]
SCL	= GPIO03 / SCL	[Pin 05]
SDA	= GPIO02 / SDA	[Pin 03]
A0	= s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_AnalogSensor.zip

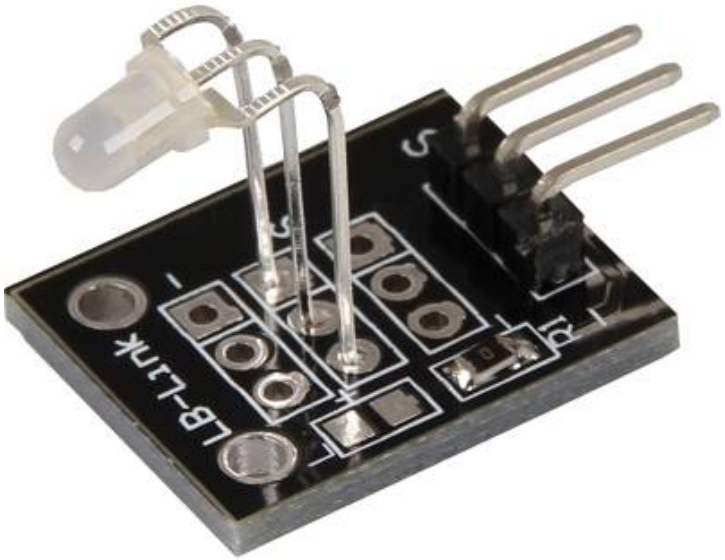
Zu starten mit dem Befehl:

```
sudo python RPi_AnalogSensor.py
```


KY-029 2-Farben - Rot+Grün - 3mm LED Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

LED-Modul welche eine rote und grüne LED beinhaltet. Diese sind mittels gemeinsamer Kathode miteinander verbunden. Je nach Eingangsspannung, werden Vorwiderstände benötigt

Pin-Belegung

Codebeispiel Arduino

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.

```
int Led_Rot = 10;
int Led_Gruen = 11;

void setup ()
{
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
}

void loop () //Hauptprogrammschleife
{
  digitalWrite (Led_Rot, HIGH); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, LOW); // LED wird eingeschaltet
  delay (3000); // Wartemodus für 3 Sekunden

  digitalWrite (Led_Rot, LOW); // LED wird eingeschaltet
  digitalWrite (Led_Gruen, HIGH); // LED wird eingeschaltet
  delay (3000); // Wartemodus für weitere zwei Sekunden in denen die LEDs dann umgeschaltet sind
}
```

Beispielprogramm ON/OFF Download:

KY-029_LED_ON-OFF.zip

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt.

```
int Led_Rot = 10;
int Led_Gruen = 11;

int val;

void setup () {
  // Initialisierung Ausgangspins für die LEDs
  pinMode (Led_Rot, OUTPUT);
  pinMode (Led_Gruen, OUTPUT);
}
void loop () {
   // Innerhalb einer For-Schleife werden den beiden LEDs verschiedene PWM-Werte uebergeben
   // Dadurch entsteht ein Farbverlauf, in dem sich durch das Vermischen unterschiedlicher
   // Helligkeitstufen der beiden integrierten LEDs, unterschiedliche Farben entstehen
    for (val = 255; val> 0; val--)
       {
       analogWrite (Led_Gruen, val);
       analogWrite (Led_Rot, 255-val);
       delay (15);
    }
    // In der zweiten For-Schleife wird der Farbverlauf rückwärts durchgegangen
    for (val = 0; val <255; val++)
       {
       analogWrite (Led_Gruen, val);
       analogWrite (Led_Rot, 255-val);
       delay (15);
    }
}
```

Beispielprogramm PWM Download:

KY-029_PWM.zip

Anschlussbelegung Arduino:

LED Grün	= [Pin 10]
LED Rot	= [Pin 11]
Sensor GND	= [Pin GND]

Codebeispiel Raspberry Pi

Codebeispiel ON/OFF

Dieses Codebeispiel zeigt auf, wie die integrierten LEDs mittels eines definierbaren Ausgangspins abwechselnd, in 3 Sekunden Takt, angeschaltet werden können.


```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_ROT = 5
LED_GRUEN = 4
GPIO.setup(LED_ROT, GPIO.OUT, initial= GPIO.LOW)
GPIO.setup(LED_GRUEN, GPIO.OUT, initial= GPIO.LOW)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
                        print("LED ROT 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.HIGH) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.LOW) #LED wird eingeschaltet
                        time.sleep(3) # Wartemodus fuer 4 Sekunden
                        print("LED GRUEN 3 Sekunden an")
                        GPIO.output(LED_ROT,GPIO.LOW) #LED wird eingeschaltet
                        GPIO.output(LED_GRUEN,GPIO.HIGH) #LED wird eingeschaltet
      #Wartemodus fuer weitere zwei Sekunden, in denen die LED Dann ausgeschaltet ist
                        time.sleep(3)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm ON/OFF Download

KY029_RPI_ON-OFF.zip

Zu starten mit dem Befehl:

```
sudo python KY029_RPI_ON-OFF.py
```

Codebeispiel PWM

Mittels Puls-Weiten-Modulation [PWM] lässt sich die Helligkeit einer LED regulieren - dabei wird die LED in bestimmten Zeitintervallen ein und ausgeschaltet, wobei das Verhältnis der Einschalt- und Ausschaltzeit einer relativen Helligkeit entspricht - aufgrund der Trägheit des menschlichen Sehvermögens, interpretieren die menschlichen Augen ein solches Ein-/Ausschaltverhalten als Helligkeitsänderung. Nähere Informationen zu diesem Thema finden Sie in diesem [Artikel von mikrokontroller.net].

In diesem Modul sind mehrere LEDs integriert - durch die Überlagerung von unterschiedlichen Helligkeitsstufen lassen sich somit verschiedene Farben kreieren. Dieses wird im folgenden Codebeispiel gezeigt. Im Raspberry Pi ist nur ein Hardware-PWM Channel uneingeschränkt auf die GPIO-Pins hinausgeführt, weswegen im vorliegenden Beispiel auf Software-PWM zurückgegriffen wird.

```
# Benoetigte Module werden importiert und eingerichtet
import random, time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

# Hier werden die Ausgangs-Pin deklariert, an dem die LEDs angeschlossen sind.
LED_Rot = 5
```



```
LED_Gruen = 4

# Set pins to output mode
GPIO.setup(LED_Rot, GPIO.OUT)
GPIO.setup(LED_Gruen, GPIO.OUT)

Freq = 100 #Hz

# Die jeweiligen Farben werden initialisiert.
ROT = GPIO.PWM(LED_Rot, Freq)
GRUEN = GPIO.PWM(LED_Gruen, Freq)
ROT.start(0)
GRUEN.start(0)

# Diese Funktion generiert die eigentliche Farbe
# Mittels der jeweiligen Farbvariable, kann die Farbintensitaet geaendert werden
# Nachdem die Farbe eingestellt wurde, wird mittels "time.sleep" die Zeit definiert,
# wie lang die besagte Farbe angezeigt werden soll

def LED_Farbe(Rot, Gruen, pause):
    ROT.ChangeDutyCycle(Rot)
    GRUEN.ChangeDutyCycle(Gruen)
    time.sleep(pause)

    ROT.ChangeDutyCycle(0)
    GRUEN.ChangeDutyCycle(0)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife:
# Diese hat die Aufgabe fuer jede einzelne Farbe eine eigene Variable zu erstellen
# und mittels einer For-Schleife die Farbintensitaet jeder einzelnen Farbe von 0-100% zu durchlaufen
# Durch die Mischungen der verschiedenen Helligkeitsstufen der jeweiligen Farben
# entsteht somit ein Farbverlauf
try:
        while True:
                for x in range(0,2):
                        for y in range(0,2):
                                print (x,y)
                                for i in range(0,101):
                                        LED_Farbe((x*i),(y*i),.02)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Beispielprogramm PWM Download:

KY029_RPI_PWM.zip

Zu starten mit dem Befehl:

```
sudo python KY029_RPI_PWM.py
```

Anschlussbelegung Raspberry Pi:

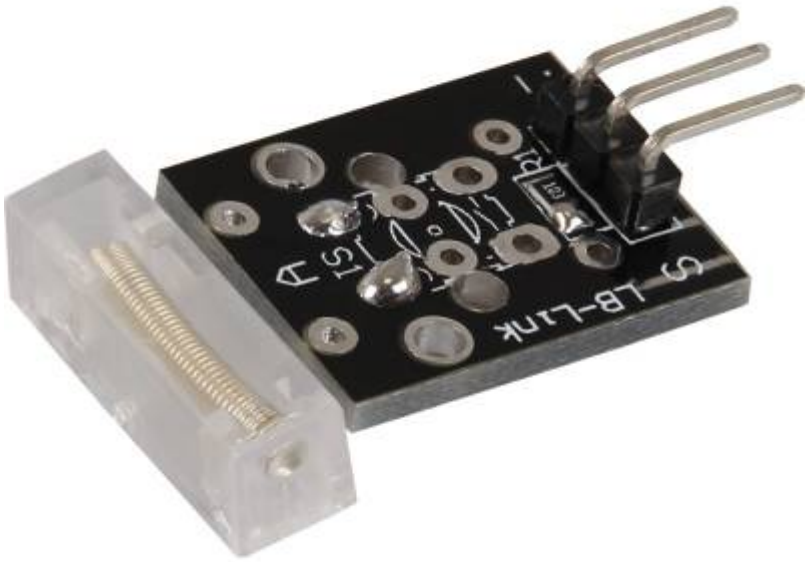
LED Gruen = GPIO4 [Pin 16]
LED Rot = GPIO5 [Pin 18]

Sensor GND = Masse [Pin 6]

KY-031 Klopf-Sensor Modul

Inhaltsverzeichnis

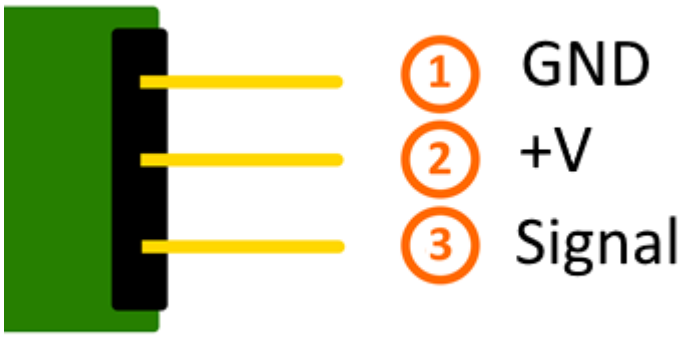
Bild



Technische Daten / Kurzbeschreibung

Wird der Sensor einem Klopfen/einer Erschütterung ausgesetzt, so werden die beiden Ausgangspins kurzgeschlossen.

Pin-Belegung



Codebeispiel Arduino

Hier bei handelt es sich um ein Beispielprogramm, welches eine LED zum Leuchten bringt, wenn am Sensor ein Signal detektiert wurde. Als LED können z.B. auch unter anderem die Module KY-011, KY-016 oder KY-029 verwendet werden.

```
int Led = 13 ;// Deklaration des LED-Ausgangspin
int Sensor = 10; // Deklaration des Sensor-Eingangspin
int val; // Temporaere Variable

void setup ()
{
  pinMode (Led, OUTPUT) ; // Initialisierung Ausgangspin
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
}

void loop ()
{
  val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    digitalWrite (Led, LOW);
  }
  else
  {
    digitalWrite (Led, HIGH);
  }
}
```

Anschlussbelegung Arduino:

LED +	= [Pin 13]
LED -	= [Pin GND]
Sensor Signal	= [Pin 10]
Sensor +V	= [Pin 5V]

Sensor - = [Pin GND]

Beispielprogramm Download

SensorTest_Arduino_withoutPullUP.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN)

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
        print("Signal erkannt")

# Beim Detektieren eines Signals (fallende Signalflanke) wird die Ausgabefunktion ausgeloest
GPIO.add_event_detect(GPIO_PIN, GPIO.FALLING, callback=ausgabeFunktion, bouncetime=100)

# Hauptprogrammschleife
try:
        while True:
                time.sleep(1)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal	= GPIO24	[Pin 18]
+V	= 3,3V	[Pin 1]
GND	= Masse	[Pin 6]

Beispielprogramm Download

SensorTest_RPi_withoutPullUP.zip

Zu starten mit dem Befehl:

```
sudo python SensorTest_RPi_withoutPullUP.py
```


KY-032 Hindernis Detektor Modul

Inhaltsverzeichnis

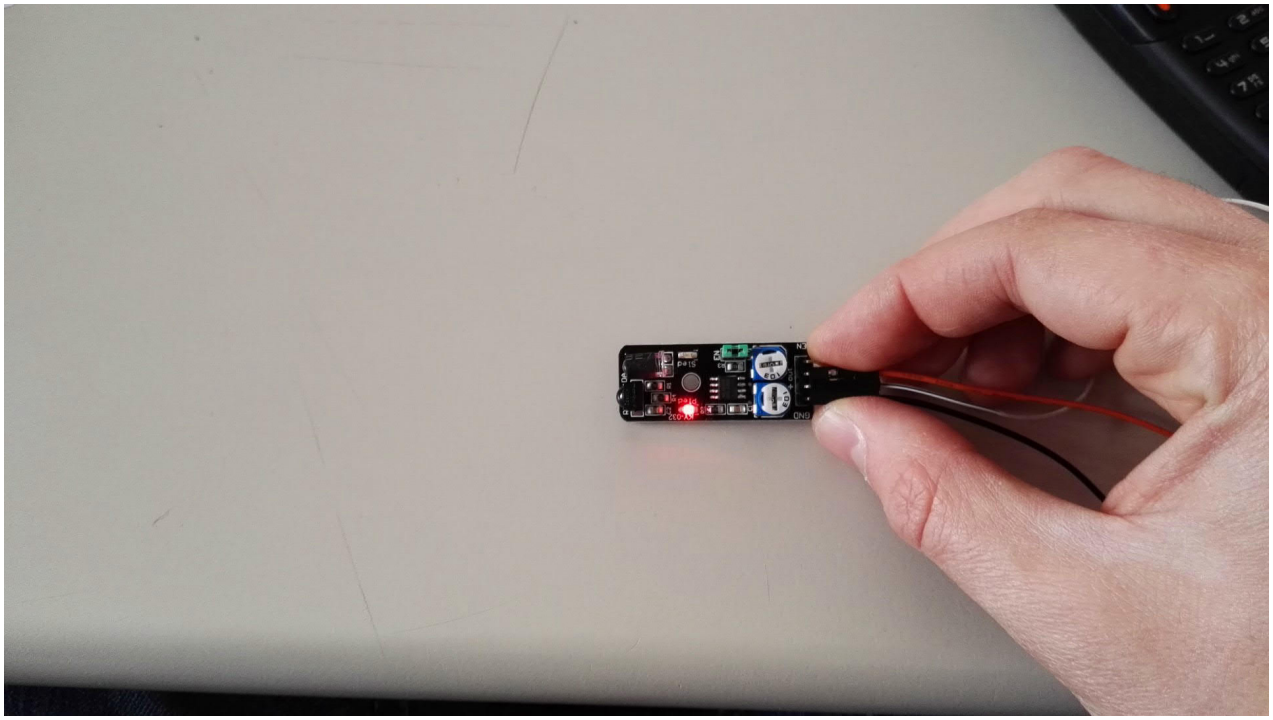
Bild

Technische Daten / Kurzbeschreibung

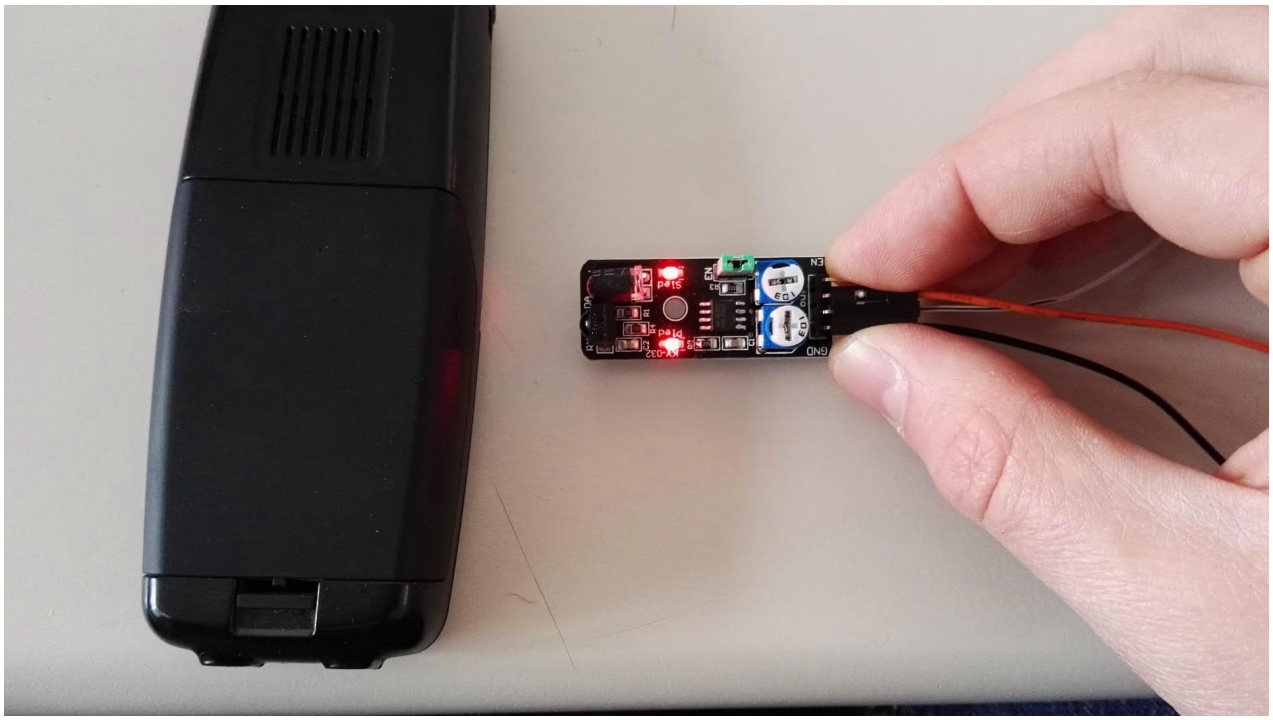
Trifft das ausgesendete Infrarot Licht auf ein Hindernis, so wird dieses reflektiert und von der Fotodiode detektiert. Die Entfernung, die zum detektieren erreicht werden muss, kann mit den beiden Reglern justiert werden.

Dieses Verhalten kann man in Steuerungen einsetzten, wie sie z.B. bei Robotern Verwendung finden, um autonom stoppen zu können, falls sie auf ein Hindernis zu fahren würden.

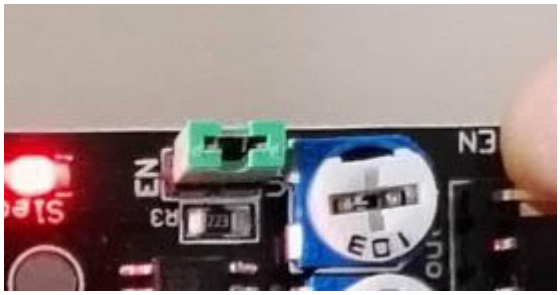
Zustand 1: Vor dem Detektor ist kein Hindernis [LED auf dem Modul: Aus] [Sensor Signal= Digital Ein]



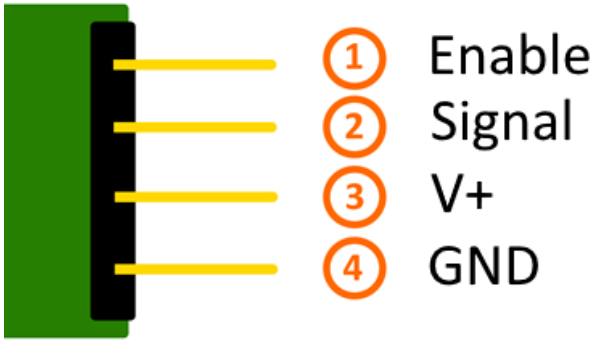
Zustand 1: Detektor hat Hindernis erkannt [LED auf dem Modul: Ein] [Sensor Signal= Digital Aus]



Dieser Sensor besitzt mit dem zusätzlichen Pin "Enable" die Möglichkeit die Hindernis-Detektion mittels Kontroller zu aktivieren oder zu deaktivieren. Standardmäßig ist bei diesem Sensor dieses aktiviert, also dauerhaft wird die Detektion durchgeführt - möchte man dies nicht, da die gewünschte Programmierung z.B. nicht vorsieht, so muss man die Steckbrücke (siehe grün im Bild) mit der Beschriftung "EN" entfernen und auf den "Enable-Pin" ein Steuersignal aktivieren.



Pin-Belegung



Codebeispiel Arduino

Das Programm liest den aktuellen Status des Sensor-Pins aus und gibt in der seriellen Konsole aus, ob der Hindernis Detektor sich aktuell vor einem Hindernis befindet oder nicht

```
int Sensor = 10; // Deklaration des Sensor-Eingangspin

void setup ()
{
  Serial.begin(9600); // Initialisierung serielle Ausgabe
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
}

// Das Programm liest den aktuellen Status des Sensor-Pins aus und
// gibt in der seriellen Konsole aus, ob ein Hindernis aktuell erkannt wird
// oder ob kein Hindernis sich vor dem Sensor befindet
void loop ()
{
  bool val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
```



```
  {
    Serial.println("Kein Hindernis");
  }
  else
  {
    Serial.println("Hindernis erkannt");
  }
  Serial.println("-----------------------------------");
  delay(500); // Pause zwischen der Messung von 500ms
}
```

Anschlussbelegung Arduino:

Sensor Enable	=	[N.C. (Steckbrücke gesteckt)]
Sensor Signal	=	[Pin 10]
Sensor +V	=	[Pin 5V]
Sensor GND	=	[Pin GND]

Beispielprogramm Download

KY-032_HindernisDetektor.zip

Codebeispiel Raspberry Pi

Das Programm liest den aktuellen Status des Sensor-Pins aus und gibt in der seriellen Konsole aus, ob der Hindernis Detektor sich aktuell vor einem Hindernis befindet oder nicht

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)

# Pause  zwischen der Ausgabe des Ergebnisses wird definiert (in Sekunden)
delayTime = 0.5

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
            if GPIO.input(GPIO_PIN) == True:
                print "Kein Hindernis"
            else:
                print "Hindernis erkannt"
            print "--------------------------------------"

            # Reset + Delay
            time.sleep(delayTime)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```


Anschlussbelegung Raspberry Pi:

Enable	=	-	[N.C. (Steckbrücke gesteckt)]
Signal	=	GPIO24	[Pin 18]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

KY-032_RPi_HindernisDetektor.zip

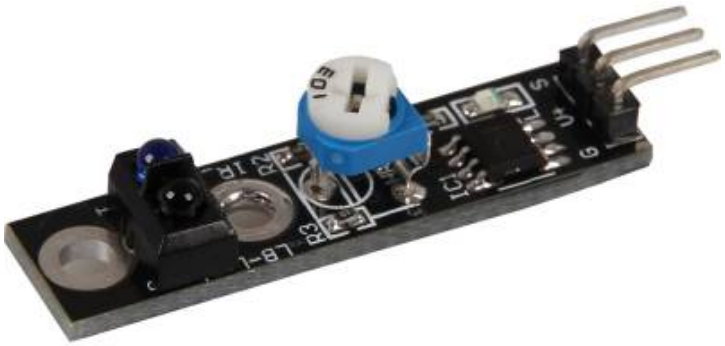
Zu starten mit dem Befehl:

```
sudo python KY-032_RPi_HindernisDetektor.py
```


KY-033 Tracking Sensor Modul

Inhaltsverzeichnis

Bild

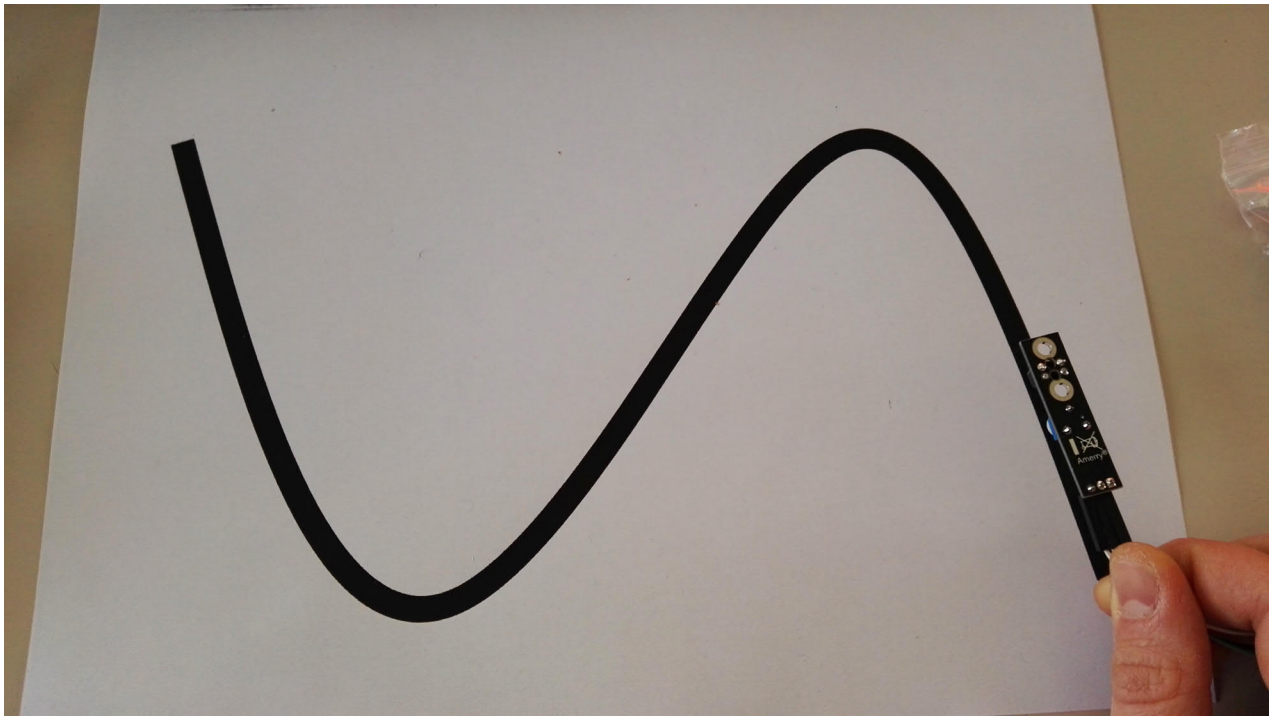


Technische Daten / Kurzbeschreibung

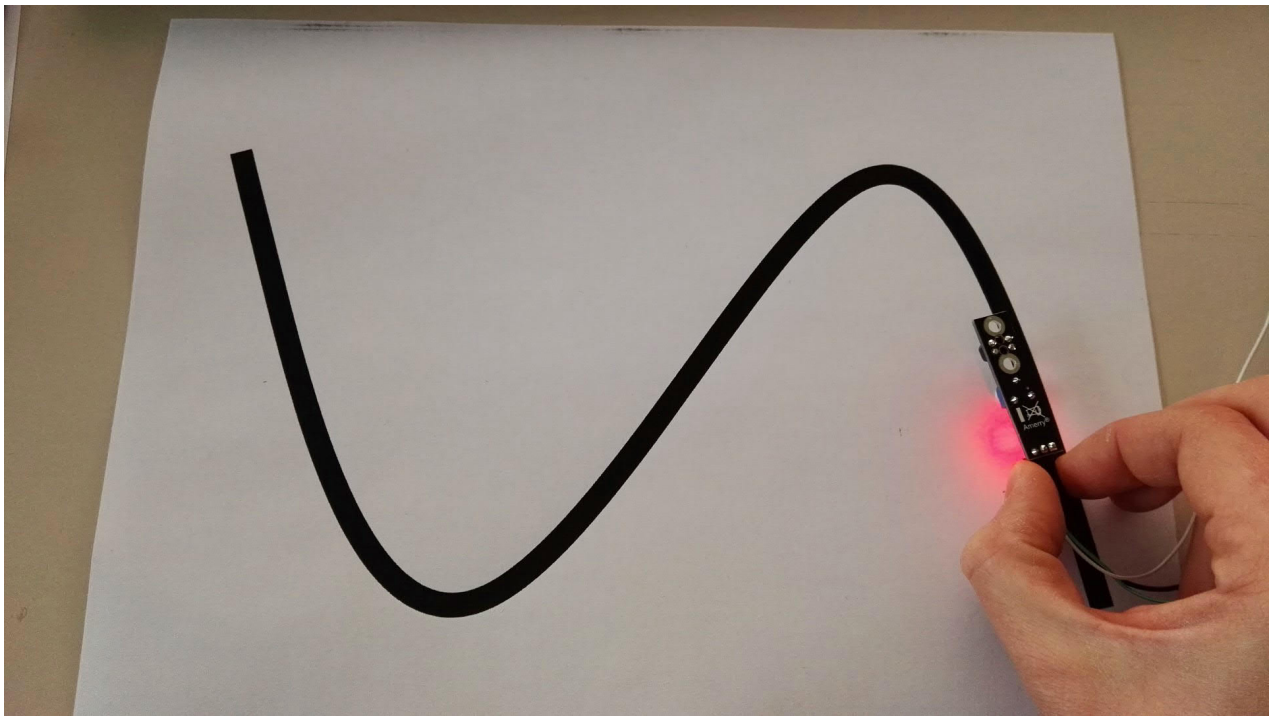
Das Sensor-Modul erkennt ob sich eine lichtreflektierende oder lichtabsorbierende Fläche vor dem Sensor befindet. Was aktuell der Fall ist, gibt das Modul an seinem digitalen Ausgang aus, wie es in den unteren Bildern aufgezeigt ist. Die Empfindlichkeit (resultierender Mindestabstand) des Sensors, kann hierbei mit dem Regler reguliert werden.

Dieses Verhalten kann man in Steuerungen einsetzten, wie sie z.B. bei Robotern Verwendung finden, um damit autonom einer Linie folgen zu können.

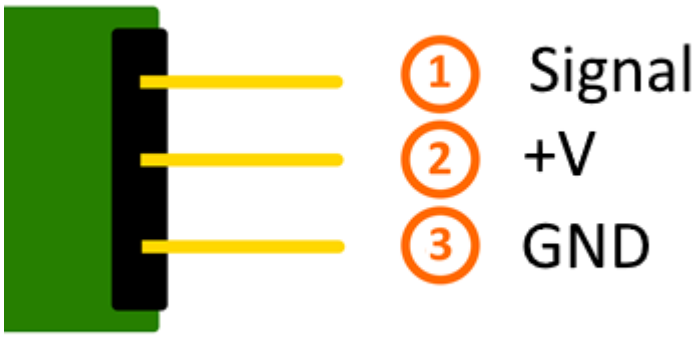
Zustand 1: Line Tracker ist über einer Line (nicht reflektierenden Fläche) [LED auf dem Modul: Aus] [Sensor Signal= Digital Ein]



Zustand 2: Line Tracker ist außerhalb einer Line (reflektierenden Fläche) [LED auf dem Modul: AN] [Sensor Signal= Digital Aus]



Pin-Belegung



Codebeispiel Arduino

Das Programm liest den aktuellen Status des Sensor-Pins aus und gibt in der seriellen Konsole aus, ob der Linetracker sich aktuell auf der Linie befindet oder nicht

```
int Sensor = 10; // Deklaration des Sensor-Eingangspin

void setup ()
{
  Serial.begin(9600); // Initialisierung serielle Ausgabe
  pinMode (Sensor, INPUT) ; // Initialisierung Sensorpin
}

// Das Programm liest den aktuellen Status des Sensor-Pins aus und
// gibt in der seriellen Konsole aus, ob der Linetracker sich aktuell
// auf der Linie befindet oder nicht
void loop ()
{
  bool val = digitalRead (Sensor) ; // Das gegenwärtige Signal am Sensor wird ausgelesen

  if (val == HIGH) // Falls ein Signal erkannt werden konnte, wird die LED eingeschaltet.
  {
    Serial.println("LineTracker ist ueber der Linie");
  }
  else
  {
    Serial.println("Linetracker ist ausserhalb der Linie");
  }
  Serial.println("-----------------------------------");
  delay(500); // Pasuse zwischen der Messung von 500ms
}
```

Anschlussbelegung Arduino:

Sensor Signal	= [Pin 10]
Sensor +V	= [Pin 5V]
Sensor GND	= [Pin GND]

Beispielprogramm Download

KY-033_TrackingSensor.zip

Codebeispiel Raspberry Pi

Das Programm liest den aktuellen Status des Sensor-Pins aus und gibt in der seriellen Konsole aus, ob der Linetracker sich aktuell auf der Linie befindet oder nicht

```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
GPIO_PIN = 24
GPIO.setup(GPIO_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)

# Pause  zwischen der Ausgabe des Ergebnisses wird definiert (in Sekunden)
delayTime = 0.5

print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
            if GPIO.input(GPIO_PIN) == True:
                print "LineTracker ist ueber der Linie"
            else:
                print "Linetracker ist ausserhalb der Linie"
            print "--------------------------------------"

            # Reset + Delay
            time.sleep(delayTime)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Signal	=	GPIO24	[Pin 18]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

KY-033_RPi_Trackingsensor.zip

Zu starten mit dem Befehl:

```
sudo python KY-033_RPi_Trackingsensor.py
```


KY-034 7 Farben LED Flash-Modul

Inhaltsverzeichnis

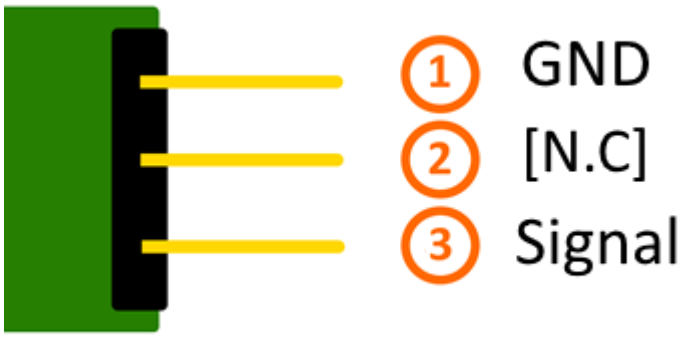
Bild

Technische Daten / Kurzbeschreibung

Wird dieses Modul mit Spannung versorgt, so wird automatisch eine Abfolge von Farbwechseln von der LED ausgestrahlt, die 7 verschiedene Farben beinhaltet.

Spannungsbereich: 3,3V - 5V

Pin-Belegung



Codebeispiel Arduino

Diese Codebeispiel zeigt auf, wie eine LED mittels eines definierbaren Ausgangspins abwechselnd für Vier Sekunden ein- und danach zwei Sekunden ausgeschaltet werden kann.

```
int Led = 13;

void setup ()
{
  pinMode (Led, OUTPUT); // Initialisierung Ausgangspin für die LED
}

void loop () //Hauptprogrammschleife
{
  digitalWrite (Led, HIGH); // LED wird eingeschaltet
  delay (4000); // Wartemodus für 4 Sekunden
  digitalWrite (Led, LOW); // LED wird ausgeschaltet
  delay (2000); // Wartemodus für weitere zwei Sekunden in denen die LED dann ausgeschaltet ist
}
```

Anschlussbelegung Arduino:

Sensor Signal	= [Pin 13]
Sensor [N.C]	=
Sensor GND	= [Pin GND]

Beispielprogramm Download:

LedTestArduino_4On_2Off.zip

Codebeispiel Raspberry Pi

Programmierbeispiel in der Programmiersprache Python


```
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier wird der Eingangs-Pin deklariert, an dem der Sensor angeschlossen ist.
# Zusaetzlich wird auch der PullUP Widerstand am Eingang aktiviert LED_PIN = 24
GPIO.setup(LED_PIN, GPIO.OUT, initial= GPIO.LOW)

print "LED-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
                print("LED 4 Sekunden an")
                GPIO.output(LED_PIN,GPIO.HIGH) #LED wird eingeschaltet
                time.sleep(4) #Wartemodus fuer 4 Sekunden
                print("LED 2 Sekunden aus")
                GPIO.output(LED_PIN,GPIO.LOW) #LED wird ausgeschaltet
      #Wartemodus fuer weitere zwei Sekunden, in denen die LED Dann ausgeschaltet ist
                time.sleep(2)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor Signal = GPIO24 [Pin 18]

Sensor [N.C] =

Sensor GND = Masse [Pin 6]

Beispielprogramm Download

LedTest_RPi_4On_2Off.zip

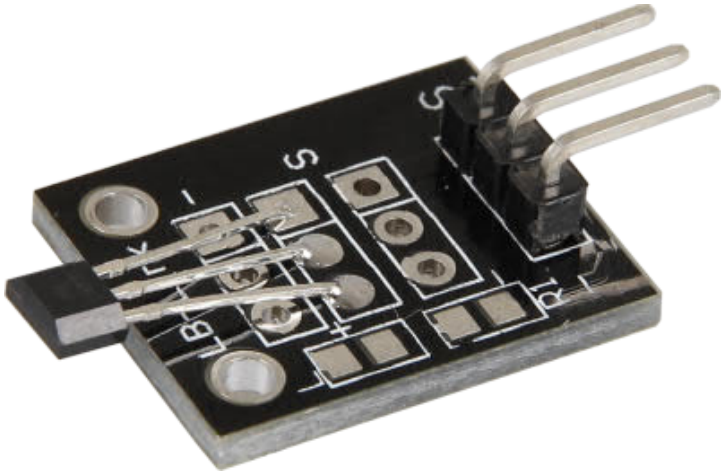
Zu starten mit dem Befehl:

```
sudo python LedTest_RPi_4On_2Off.py
```


KY-035 Bihor Magnet Sensor Modul

Inhaltsverzeichnis

Bild

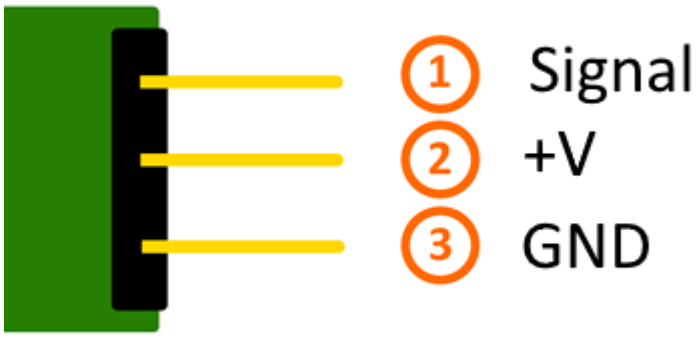


Technische Daten / Kurzbeschreibung

Chipsatz: AH49E

Der Sensor gibt über seinen Ausgang ein analoges Spannungssignal, welches die Stärke des Magnetfelds angibt.

Pin-Belegung



Codebeispiel Arduino

Das Programm misst den aktuellen Spannungswert am Sensor, berechnet aus diesen und dem bekannten Serienwiderstand den aktuellen Widerstandswert des Sensors und gibt die Ergebnisse auf der serielle Ausgabe aus.

```
int sensorPin = A5; // Hier wird der Eingangs-Pin deklariert

// Serielle Ausgabe in 9600 Baud
void setup()
{
        Serial.begin(9600);
}

// Das Programm misst den aktuellen Spannungswert am Sensor,
// berechnet aus diesen und dem bekannten Serienwiderstand den aktuellen
// Widerstandswert des Sensors und gibt die Ergebnisse auf der serielle Ausgabe aus

void loop()
{
        // Aktueller Spannungswert wird gemessen...
        int rawValue = analogRead(sensorPin);
        float voltage = rawValue * (5.0/1023) * 1000;

        float resitance = 10000 * ( voltage / ( 5000.0 - voltage) );

        // ... und hier auf die serielle Schnittstelle ausgegeben
        Serial.print("Spannungswert:");       Serial.print(voltage); Serial.print("mV");
        Serial.print(", Widerstandswert:");Serial.print(resitance); Serial.println("Ohm");
        Serial.println("---------------------------------------");

        delay(500);
}
```

Anschlussbelegung Arduino:

Sensor GND = [Pin GND]

Sensor +V = [Pin 5V]
Sensor Signal = [Pin A5]

Beispielprogramm Download

Single_Analog_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm misst den aktuellen Spannungswert am Sensor, berechnet aus diesen und dem bekannten Serienwiderstand den aktuellen Widerstandswert des Sensors und gibt die Ergebnisse auf der serielle Ausgabe aus.

```
# coding=utf-8
#!/usr/bin/python


##################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Single Analog Sensor - Raspberry Pi Python Code Example
### ##############################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
```

```
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os, math
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2
voltageMax = 3300 # maximal möglicher Spannungswert am ADC

# Adresszuweisung ADS1x15 ADC

ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
# sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

##########################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm misst mit Hilfe des ADS1115 ADC den aktuellen Spannungswert am ADC,
# berechnet aus diesem und den bekannten Widerstandswert
#  des Serien-Vorwiderstands den aktuellen Widerstandwert des Sensors
# und gibt diese in der Konsole aus.

try:
        while True:
                #Aktueller Wert wird aufgenommen,...
                voltage = adc.readADCSingleEnded(adc_channel, gain, sps)

                # ... der Widerstand wird berechnet...
                resitance = 10000 * voltage/(voltageMax - voltage)

                # ... und beides hier in die Konsole ausgegeben
                print "Spannungswert:", voltage,"mV, Widerstand:", resitance,"Ω"
                print "----------------------------------------"

                # Delay
                time.sleep(delayTime)
```



```
except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

GND	= GND	[Pin 06 (RPi)]
+V	= 3,3V	[Pin 01 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 17]
GND	= Masse	[Pin 09]
SCL	= GPIO03 / SCL	[Pin 05]
SDA	= GPIO02 / SDA	[Pin 03]
A0	= s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_Single_Analog_Sensor.zip

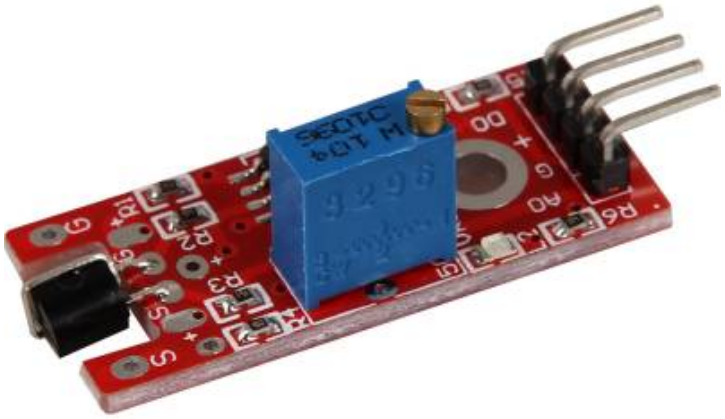
Zu starten mit dem Befehl:

```
sudo python RPi_Single_Analog_Sensor.py
```


KY-036 Metall-Touchsensor Modul

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

Gibt ein Signal aus, falls die vordere Metallspitze des Sensors berührt wird. Die Empfindlichkeit des Sensors kann mittels Regler justiert werden.

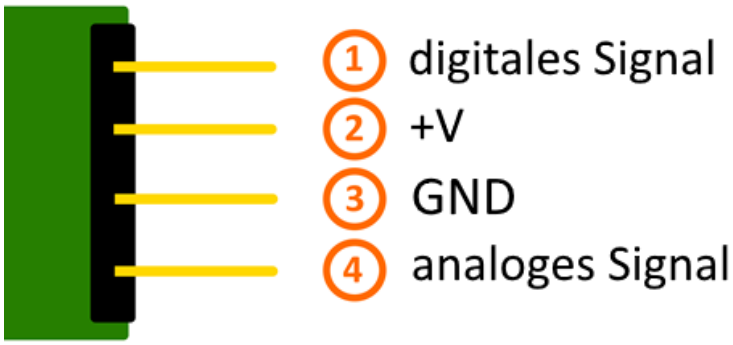
Digitaler Ausgang: Wird eine Berührung detektiert, wird hier ein Signal ausgegeben

Analoger Ausgang: Direkter Messwert der Sensoreinheit

LED1: Zeigt an, dass der Sensor mit Spannung versorgt ist

LED2: Zeigt an, dass ein Magnetfeld detektiert wurde

Pin-Belegung

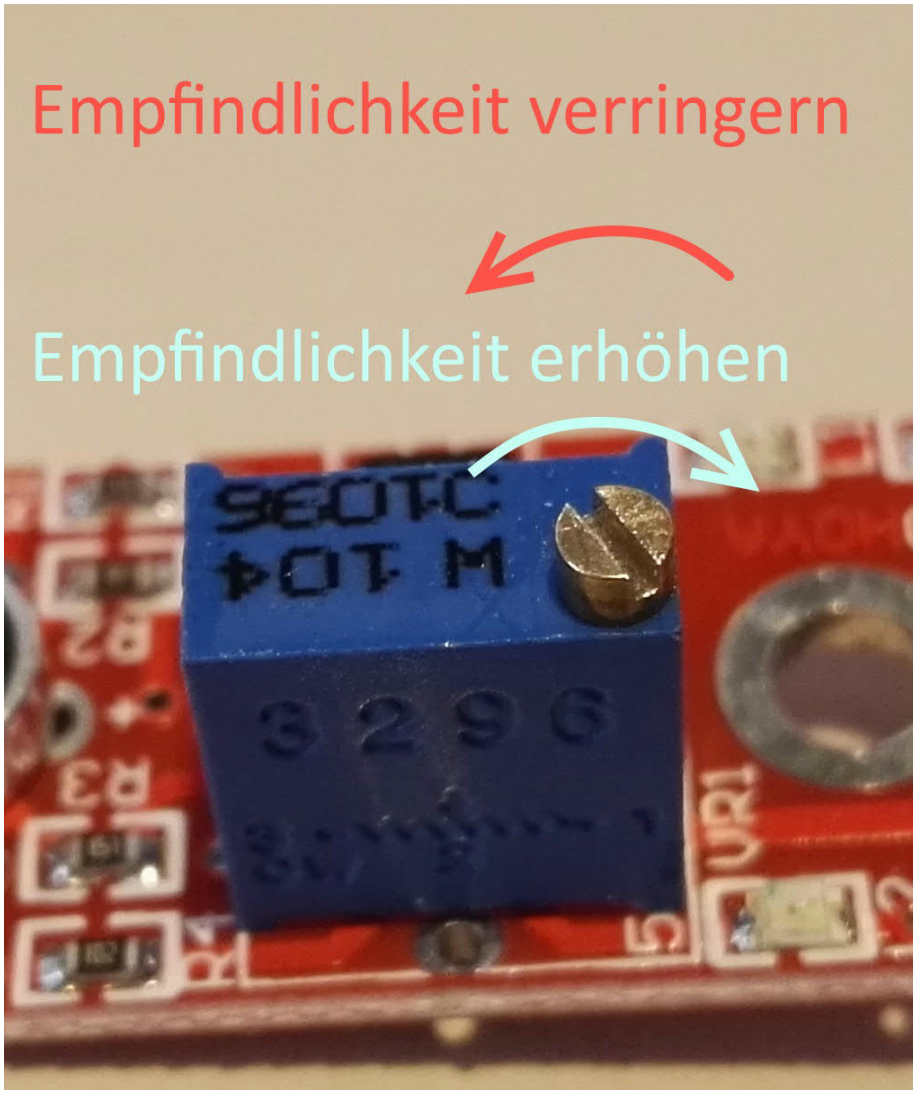


Funktionsweise des Sensors

Dieser Sensor besitzt auf seiner Platine drei funktionelle Bestandteile. Die ist die Sensoreinheit vorne am Modul, welche das aktuelle Umfeld physikalisch misst und als analoges Signal auf die zweite Einheit, dem Verstärker, ausgibt. Dieser verstärkt das Signal abhängig vom eingestellten Widerstand am Drehpotentiometer und leitet es auf den analogen Ausgang des Moduls.

Hierbei ist zu beachten: Das Signal ist invertiert; wird ein hoher Wert gemessen, so resultiert dies in einen niedrigeren Spannungswert am analogen Ausgang.

Die dritte Einheit stellt einen Komparator dar, welcher den digitalen Ausgang und die LED schaltet, wenn das Signal unter einen bestimmten Wert fällt. Mittels des Drehpotentiometers kann somit die Empfindlichkeit eingestellt werden, wie es im folgenden Bild aufgezeigt wird:



Dieser Sensor gibt somit keine absoluten Werte aus (z.B. genau gemessene Temperatur in °C oder Magnetfeldstärke in mT) , sondern es handelt sich hierbei um eine Relativ-Messung: Man definiert hierbei einen Grenzwert relativ zur gegebenen normalen Umweltsituation und es wird ein Signal ausgegeben was weiterverarbeitet werden kann, falls dieser Grenzwert überschritten bzw. ein anderer Zustand als der Normalfall eingetreten ist.

Dieses Verhalten eignet sich hervorragend zur Temperaturüberwachung (KY-028), Näherungsschalter (KY-024, KY-025, KY-036), Alarmüberwachungen (KY-037, KY-038) oder Drehgeber (KY-026).

Codebeispiel Arduino

Das Programm liest den aktuellen Spannungswert aus, der am analogen Ausgang gemessen werden kann, und gibt diesen auf der seriellen Schnittstelle aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.


```
// Deklaration und Initialisierung der Eingang-Pins
int Analog_Eingang = A0; // X-Achse-Signal
int Digital_Eingang = 3; // Knopf

void setup ()
{
  pinMode (Analog_Eingang, INPUT);
  pinMode (Digital_Eingang, INPUT);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float Analog;
  int Digital;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  Analog = analogRead (Analog_Eingang) * (5.0 / 1023.0);
  Digital = digitalRead (Digital_Eingang);

  //... und an dieser Stelle ausgegeben
  Serial.print ("Analoger Spannungswert:"); Serial.print (Analog, 4);Serial.print ("V, ");
  Serial.print ("Grenzwert:");

  if(Digital==1)
  {
      Serial.println (" erreicht");
  }
  else
  {
      Serial.println (" noch nicht erreicht");
  }
  Serial.println ("-----------------------------------------------------------------");
  delay (200);
}
```

Anschlussbelegung Arduino:

digitales Signal	= [Pin 3]
+V	= [Pin 5V]
GND	= [Pin GND]
analoges Signal	= [Pin 0]

Beispielprogramm Download

Ard_Analoger_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Analog Sensor + ADS1115 ADC - Raspberry Pi Python Code Example
### ######################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC
```



```
ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

# Hier waehlt man den Eingangs-Pin des digitalen Signals aus
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)

##################################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                analog = adc.readADCSingleEnded(adc_channel, gain, sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Digital_PIN) == False:
                        print "Analoger Spannungswert:",analog,"mV,","Grenzwert: noch nicht erreicht"
                else:
                        print "Analoger Spannungswert:", analog, "mV, ", "Grenzwert: erreicht"
                print "---------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

digitales Signal	= GPIO 24	[Pin 18 (RPi)]
+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 01]
GND	= Masse	[Pin 09]
SCL	= GPIO03 / SCL	[Pin 05]
SDA	= GPIO02 / SDA	[Pin 03]
A0	= s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_AnalogSensor.zip

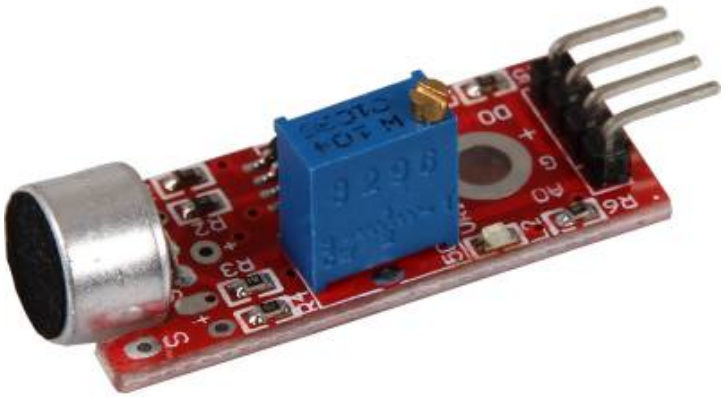
Zu starten mit dem Befehl:

```
sudo python RPi_AnalogSensor.py
```


KY-037 Mikrofon Sensor Modul - hohe Empfindlichkeit

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

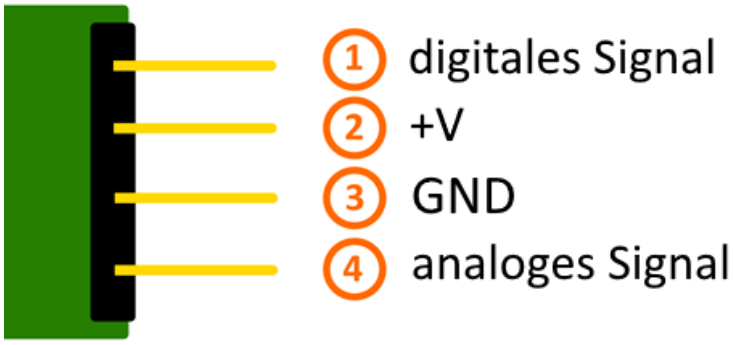
Digitaler Ausgang: Über das Potentiometer, kann ein Grenzwert für den empfangenen Schall eingestellt werden, bei dem der digitale Ausgang schalten soll.

Analoger Ausgang: Direktes Mikrofon-Signal als Spannungspegel

LED1: Zeigt an, dass der Sensor mit Spannung versorgt ist

LED2: Zeigt an, dass ein Magnetfeld detektiert wurde

Pin-Belegung

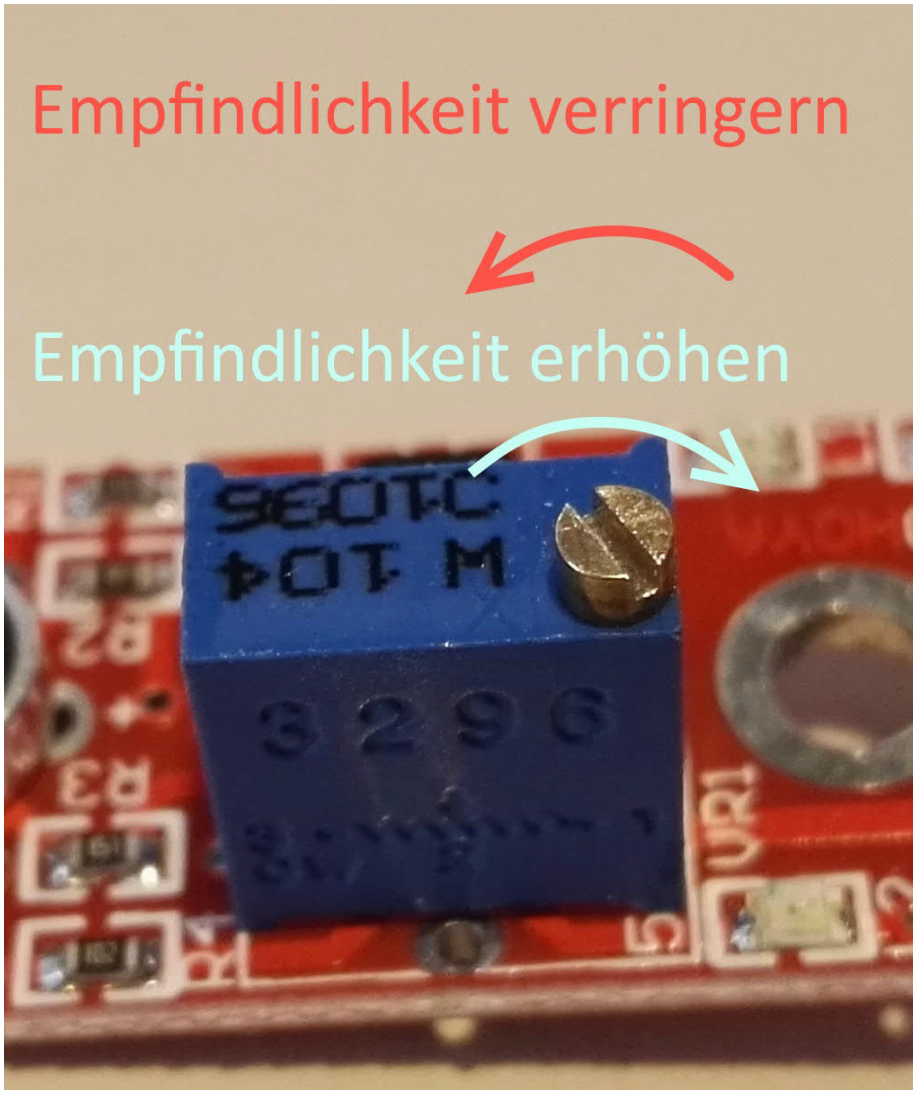


Funktionsweise des Sensors

Dieser Sensor besitzt auf seiner Platine drei funktionelle Bestandteile. Die ist die Sensoreinheit vorne am Modul, welche das aktuelle Umfeld physikalisch misst und als analoges Signal auf die zweite Einheit, dem Verstärker, ausgibt. Dieser verstärkt das Signal abhängig vom eingestellten Widerstand am Drehpotentiometer und leitet es auf den analogen Ausgang des Moduls.

Hierbei ist zu beachten: Das Signal ist invertiert; wird ein hoher Wert gemessen, so resultiert dies in einen niedrigeren Spannungswert am analogen Ausgang.

Die dritte Einheit stellt einen Komparator dar, welcher den digitalen Ausgang und die LED schaltet, wenn das Signal unter einen bestimmten Wert fällt. Mittels des Drehpotentiometers kann somit die Empfindlichkeit eingestellt werden, wie es im folgenden Bild aufgezeigt wird:



Dieser Sensor gibt somit keine absoluten Werte aus (z.B. genau gemessene Temperatur in °C oder Magnetfeldstärke in mT) , sondern es handelt sich hierbei um eine Relativ-Messung: Man definiert hierbei einen Grenzwert relativ zur gegebenen normalen Umweltsituation und es wird ein Signal ausgegeben was weiterverarbeitet werden kann, falls dieser Grenzwert überschritten bzw. ein anderer Zustand als der Normalfall eingetreten ist.

Dieses Verhalten eignet sich hervorragend zur Temperaturüberwachung (KY-028), Näherungsschalter (KY-024, KY-025, KY-036), Alarmüberwachungen (KY-037, KY-038) oder Drehgeber (KY-026).

Codebeispiel Arduino

Das Programm liest den aktuellen Spannungswert aus, der am analogen Ausgang gemessen werden kann, und gibt diesen auf der seriellen Schnittstelle aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.


```
// Deklaration und Initialisierung der Eingang-Pins
int Analog_Eingang = A0; // X-Achse-Signal
int Digital_Eingang = 3; // Knopf

void setup ()
{
  pinMode (Analog_Eingang, INPUT);
  pinMode (Digital_Eingang, INPUT);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float Analog;
  int Digital;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  Analog = analogRead (Analog_Eingang) * (5.0 / 1023.0);
  Digital = digitalRead (Digital_Eingang);

  //... und an dieser Stelle ausgegeben
  Serial.print ("Analoger Spannungswert:"); Serial.print (Analog, 4);Serial.print ("V, ");
  Serial.print ("Grenzwert:");

  if(Digital==1)
  {
      Serial.println (" erreicht");
  }
  else
  {
      Serial.println (" noch nicht erreicht");
  }
  Serial.println ("-----------------------------------------------------------------");
  delay (200);
}
```

Anschlussbelegung Arduino:

digitales Signal	= [Pin 3]
+V	= [Pin 5V]
GND	= [Pin GND]
analoges Signal	= [Pin 0]

Beispielprogramm Download

Ard_Analoger_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Analog Sensor + ADS1115 ADC - Raspberry Pi Python Code Example
### ######################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC
```



```
ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

# Hier waehlt man den Eingangs-Pin des digitalen Signals aus
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)

##################################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                analog = adc.readADCSingleEnded(adc_channel, gain, sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Digital_PIN) == False:
                        print "Analoger Spannungswert:",analog,"mV,","Grenzwert: noch nicht erreicht"
                else:
                        print "Analoger Spannungswert:", analog, "mV, ", "Grenzwert: erreicht"
                print "---------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

digitales Signal	= GPIO 24	[Pin 18 (RPi)]
+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 01]
GND	= Masse	[Pin 09]
SCL	= GPIO03 / SCL	[Pin 05]
SDA	= GPIO02 / SDA	[Pin 03]
A0	= s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_AnalogSensor.zip

Zu starten mit dem Befehl:

```
sudo python RPi_AnalogSensor.py
```


KY-038 Mikrofon Sound Sensor Modul

Inhaltsverzeichnis

Bild

Technische Daten / Kurzbeschreibung

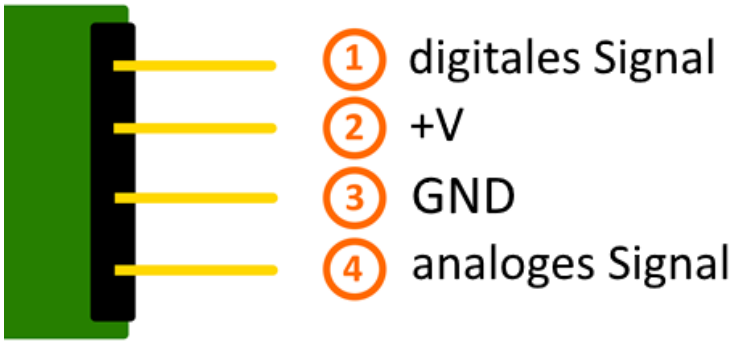
Digitaler Ausgang: Über das Potentiometer, kann ein Grenzwert für den empfangenen Schall eingestellt werden, bei dem der digitale Ausgang schalten soll.

Analoger Ausgang: Direktes Mikrofon-Signal als Spannungspegel

LED1: Zeigt an, dass der Sensor mit Spannung versorgt ist

LED2: Zeigt an, dass ein Magnetfeld detektiert wurde

Pin-Belegung

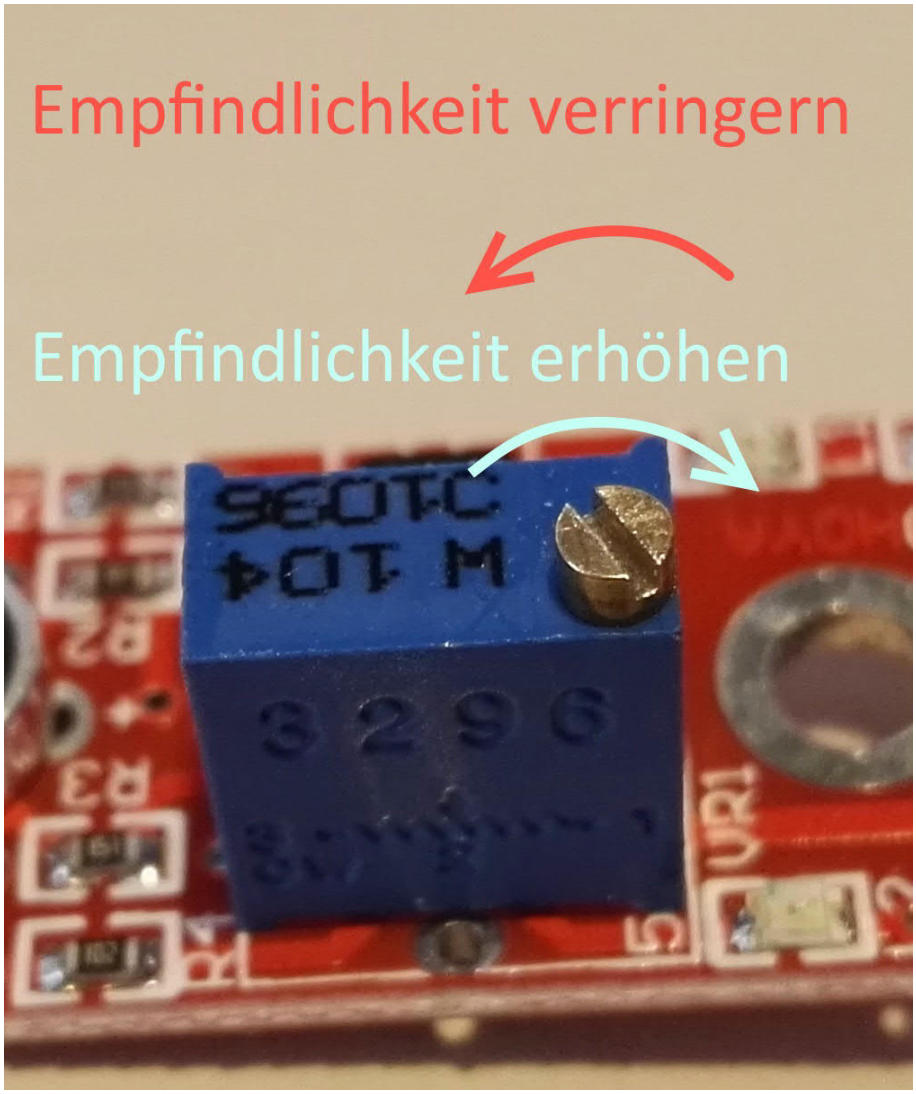


Funktionsweise des Sensors

Dieser Sensor besitzt auf seiner Platine drei funktionelle Bestandteile. Die ist die Sensoreinheit vorne am Modul, welche das aktuelle Umfeld physikalisch misst und als analoges Signal auf die zweite Einheit, dem Verstärker, ausgibt. Dieser verstärkt das Signal abhängig vom eingestellten Widerstand am Drehpotentiometer und leitet es auf den analogen Ausgang des Moduls.

Hierbei ist zu beachten: Das Signal ist invertiert; wird ein hoher Wert gemessen, so resultiert dies in einen niedrigeren Spannungswert am analogen Ausgang.

Die dritte Einheit stellt einen Komparator dar, welcher den digitalen Ausgang und die LED schaltet, wenn das Signal unter einen bestimmten Wert fällt. Mittels des Drehpotentiometers kann somit die Empfindlichkeit eingestellt werden, wie es im folgenden Bild aufgezeigt wird:



Dieser Sensor gibt somit keine absoluten Werte aus (z.B. genau gemessene Temperatur in °C oder Magnetfeldstärke in mT) , sondern es handelt sich hierbei um eine Relativ-Messung: Man definiert hierbei einen Grenzwert relativ zur gegebenen normalen Umweltsituation und es wird ein Signal ausgegeben was weiterverarbeitet werden kann, falls dieser Grenzwert überschritten bzw. ein anderer Zustand als der Normalfall eingetreten ist.

Dieses Verhalten eignet sich hervorragend zur Temperaturüberwachung (KY-028), Näherungsschalter (KY-024, KY-025, KY-036), Alarmüberwachungen (KY-037, KY-038) oder Drehgeber (KY-026).

Codebeispiel Arduino

Das Programm liest den aktuellen Spannungswert aus, der am analogen Ausgang gemessen werden kann, und gibt diesen auf der seriellen Schnittstelle aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.


```
// Deklaration und Initialisierung der Eingang-Pins
int Analog_Eingang = A0; // X-Achse-Signal
int Digital_Eingang = 3; // Knopf

void setup ()
{
  pinMode (Analog_Eingang, INPUT);
  pinMode (Digital_Eingang, INPUT);

  Serial.begin (9600); // Serielle Ausgabe mit 9600 bps
}

// Das Programm liest die aktuellen Werte der Eingang-Pins
// und gibt diese auf der seriellen Ausgabe aus
void loop ()
{
  float Analog;
  int Digital;

  //Aktuelle Werte werden ausgelesen, auf den Spannungswert konvertiert...
  Analog = analogRead (Analog_Eingang) * (5.0 / 1023.0);
  Digital = digitalRead (Digital_Eingang);

  //... und an dieser Stelle ausgegeben
  Serial.print ("Analoger Spannungswert:"); Serial.print (Analog, 4);Serial.print ("V, ");
  Serial.print ("Grenzwert:");

  if(Digital==1)
  {
      Serial.println (" erreicht");
  }
  else
  {
      Serial.println (" noch nicht erreicht");
  }
  Serial.println ("-----------------------------------------------------------------");
  delay (200);
}
```

Anschlussbelegung Arduino:

digitales Signal	= [Pin 3]
+V	= [Pin 5V]
GND	= [Pin GND]
analoges Signal	= [Pin 0]

Beispielprogramm Download

Ard_Analoger_Sensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuellen Werte der Eingang-Pins und gibt diese in der Konsole als Wert in [mV] aus.

Zudem wird ebenfalls der Zustand des digitalen Pins in der Konsole angegeben, was bedeutet ob der Grenzwert unterschritten wurde oder nicht.

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Analog Sensor + ADS1115 ADC - Raspberry Pi Python Code Example
### ######################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.2

# Adresszuweisung ADS1x15 ADC
```



```
ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

# Hier waehlt man den Eingangs-Pin des digitalen Signals aus
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)

##################################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                analog = adc.readADCSingleEnded(adc_channel, gain, sps)

                # Ausgabe auf die Konsole
                if GPIO.input(Digital_PIN) == False:
                        print "Analoger Spannungswert:",analog,"mV,","Grenzwert: noch nicht erreicht"
                else:
                        print "Analoger Spannungswert:", analog, "mV, ", "Grenzwert: erreicht"
                print "---------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor

digitales Signal	= GPIO 24	[Pin 18 (RPi)]
+V	= 3,3V	[Pin 1 (RPi)]
GND	= Masse	[Pin 06 (RPi)]
analoges Signal	= Analog 0	[Pin A0 (ADS1115 - KY-053)]

ADS1115 - KY-053:

VDD	= 3,3V	[Pin 01]
GND	= Masse	[Pin 09]
SCL	= GPIO03 / SCL	[Pin 05]
SDA	= GPIO02 / SDA	[Pin 03]
A0	= s.o.	[Sensor: analoges Signal]

Beispielprogramm Download

RPi_AnalogSensor.zip

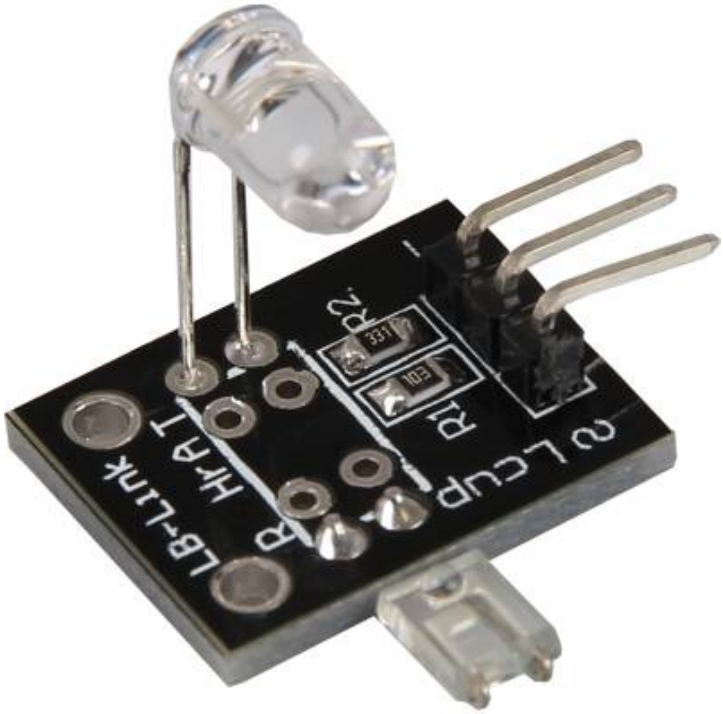
Zu starten mit dem Befehl:

```
sudo python RPi_AnalogSensor.py
```


KY-039 Herzschlag Sensor Modul

Inhaltsverzeichnis

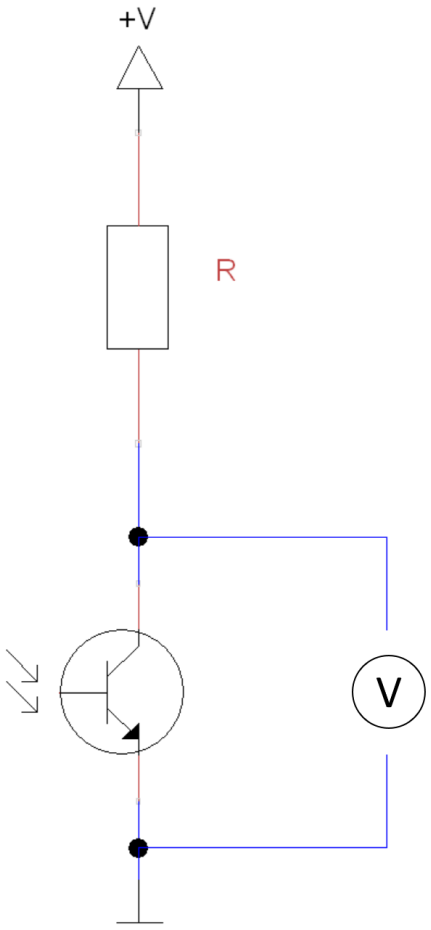
Bild



Technische Daten / Kurzbeschreibung

Wird ein Finger zwischen der Infrarot-Leuchtdiode und dem Foto-Transistor gehalten, so kann am Signalausgang der Puls detektiert werden.

Die Funktionsweise eines Fototransistors ist wie folgt erklärt: Dieser funktioniert in der Regel wie ein normaler Transistor - so wird ein höherer Strom durch ihn durchgelassen, je höher die Steuerspannung ist, die an ihn angelegt wird. Bei einem Fototransistor stellt jedoch das einfallende Licht die Steuerspannung dar - je höher das einfallende Licht, so höher der durchgelassene Strom.

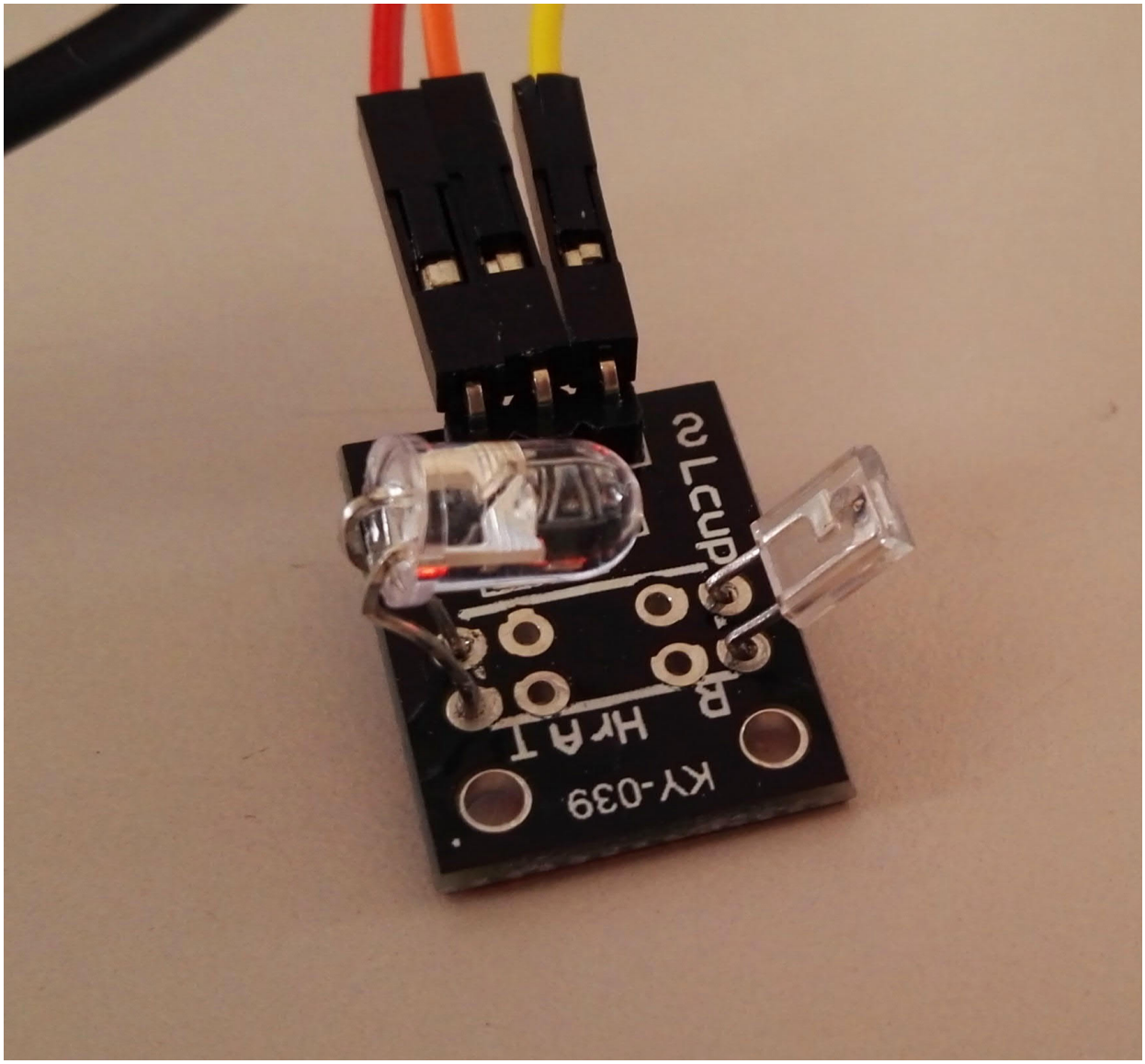


Schaltet man vor dem Transistor einen Widerst
Reihe, so ergibt sich folgendes Verhalten, wenn
die Spannung über den Transistor misst: Scheir
den Transistor viel Licht bzw. ist es außen hell,
man eine niedrige Spannung nahe 0V gemesse
der Transistor im Dunklen, so lässt dieser einen
kleinen Strom durch und man misst eine Spann
nahe +V.

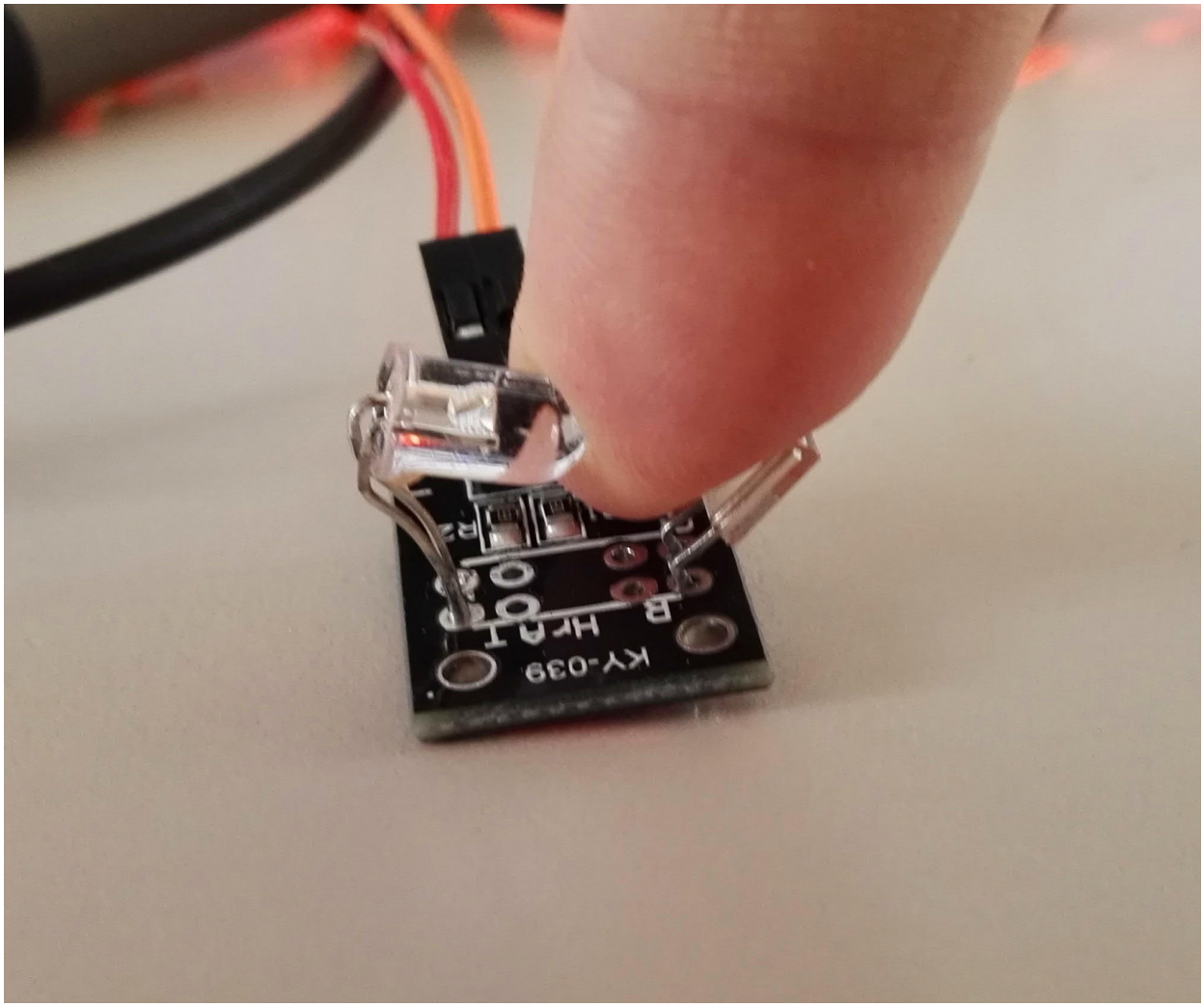
Der bei diesem Sensormodul aufgebaute Messaufbau mit Infrarotdiode und Fototransistor ermöglicht uns nun den Puls zu messen, indem ein Finger zwischen Diode und Transistor gelegt wird. Erklärung: Genau so wie man es von der Taschenlampe kennt, kann die Haut an der Hand durchleuchtet werden. Trifft man beim Durchleuchten auf eine Blutader, so kann man ganz schwach das Pumpen des Blutes erkennen. Dieses Pumpen erkennt man, da das Blut an unterschiedlichen Stellen in der Ader eine andere Dichte besitzt und somit Helligkeitsunterschiede beim Blutfluss erkennbar sind. Genau diese Unterschiede in der Helligkeit kann man mit dem Sensormodul aufnehmen und somit den Puls erkennen. Deutlich wird dieses beim Betrachten des folgenden Oszilloskop-Bildes.

Dieses zeigt auf der Y-Achse die Veränderung der Spannung am Fototransistor - somit die Helligkeitsveränderungen hervorgerufen durch das fließende Blut. Die oben gekennzeichneten Spitzen ergeben somit das Schlagen vom Herz. Rechnet man nun die registrierten Schläge pro aufgenommener Zeit, so kommt man auf einen Puls von ca. 71 Schläge/Minute (bpm)

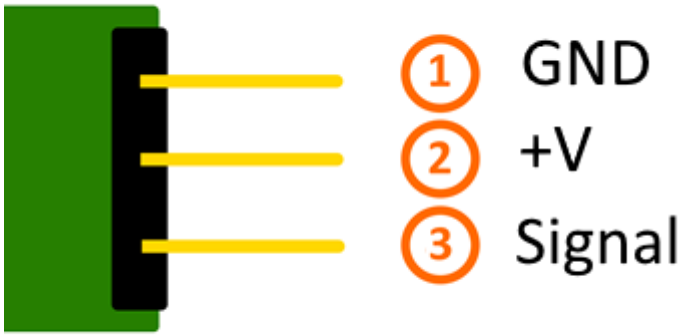
Wie man auf dem oberen Oszilloskop-Bild zudem sehen kann, ist das aufgenommene Signal relativ klein bzw. der Transistor sehr empfindlich, um das schwache Signal aufnehmen zu können.Um ein optimales Ergebnis zu erhalten, empfehlen wir das Modul zum messen so vorzubereiten, wie im folgenden Bild gezeigt:



Zudem sollte beim messen am besten der kleine Finger verwendet werden, sowie darauf geachtet werden, dass zwischen Diode und Transistor kein blockierender Knochen sondern eher größenteils die Haut aufgenommen wird.



Pin-Belegung



Codebeispiel Arduino

Das folgende Code-Beispiel stammt aus der Feder von Dan Truong, welcher diesen Code unter [folgenden Link] veröffentlicht hat. Dieser steht unter der [|MIT OpenSource Lizenz] zur Verfügung. Die unten stehende Version ist die übersetzte deutsche Fassung - das original steht unten zum Download zur Verfügung.

Dieser Code stellt eine sog. Peak-Detection dar. Es wird kein Herzschlagverlauf aufgezeichnet, sondern es wird innerhalb der aufgezeichneten Daten nach "Peaks" (Spitzen) gesucht, als Herschlag erkannt und per LED angezeigt. Mittels der bekannten Delay Abstände, kann somit grob der Puls errechnet werden.

Wird der Finger beim messen neu aufgelegt oder stark bewegt, so kann es etwas dauern, bis das Programm sich auf die neue Gegebenheit kalibriert und wieder den richtigen Wert ausgibt.

```
////////////////////////////////////////////////////////////////////////
/// Copyright (c)2015 Dan Truong
/// Permission is granted to use this software under the MIT
/// licence, with my name and copyright kept in source code
/// http://http://opensource.org/licenses/MIT
///
/// KY039 Arduino Heartrate Monitor V1.0 (April 02, 2015)
////////////////////////////////////////////////////////////////////////

// German Comments by Joy-IT


//////////////////////////////////////////////////////////////////////
/// @param[in] IRSensorPin Analog PI an welchen der Sensor angeschlossen ist
/// @param[in] delay (msec) Die Verzoegerung zwischen den Aufrufen der Abtastfunktion.
//                  Die besten Ergebnisse erhaelt man, wenn man 5 mal Pro Herzschlag abtastet.
///                 Nicht langsamer als 150mSec für z.B. 70 BPM Puls
///                 Besser waere 60 mSec für z.B. bis zu einen Puls von 200 BPM.
///
/// @Kurzbeschreibung
/// Dieser Code stellt eine sog. Peak-Detection dar.
/// Es wird kein Herzschlagverlauf aufgezeichnet, sondern es
/// wird innerhalb der aufgezeichneten Daten nach "Peaks" (Spitzen) gesucht,
/// und per LED angezeigt. Mittels der bekannten Delay Abstaende, kann somit
/// grob der Puls errechnet werden.
//////////////////////////////////////////////////////////////////////

int rawValue;

bool
heartbeatDetected(int IRSensorPin, int delay)
{
  static int maxValue = 0;
  static bool isPeak = false;


  bool result = false;

  rawValue = analogRead(IRSensorPin);
 // Hier wird der aktuelle Spannungswert am Fototransistor ausgelesen+in der rawValue-Variable gespeichert
  rawValue *= (1000/delay);

  // Sollte der aktuelle Wert vom letzten maximalen Wert zu weit abweichen
  // (z.B. da der Finger neu aufgesetzt oder weggenommen wurde)
  // So wird der MaxValue resetiert, um eine neue Basis zu erhalten.
  if (rawValue * 4L < maxValue) {    maxValue = rawValue * 0.8;  }
    if (rawValue > maxValue - (1000/delay)) {
    // Hier wird der eigentliche Peak detektiert. Sollte ein neuer RawValue groeßer sein
        // als der letzte maximale Wert, so wird das als Spitze der aufgezeichneten Daten erkannt.
    if (rawValue > maxValue) {
```



```
    if (rawValue > maxValue) {
      maxValue = rawValue;
    }
    // Zum erkannten Peak soll nur ein Herzschlag zugewiesen werden
    if (isPeak == false) {
      result = true;
    }
    isPeak = true;
  } else if (rawValue < maxValue - (3000/delay)) {
    isPeak = false;
    // Hierbei wird der maximale Wert bei jeden Durchlauf
    // etwas wieder herabgesetzt. Dies hat den Grund, dass
    // nicht nur der Wert sonst immer stabil bei jedem Schlag
    // gleich oder kleiner werden wuerde, sondern auch,
    // falls der Finger sich minimal bewegen sollte und somit
    // das Signal generell schwaecher werden wuerde.
    maxValue-=(1000/delay);
 }
  return result;
}


////////////////////////////////////////////////////////////////////////
// Arduino main code
////////////////////////////////////////////////////////////////////////
int ledPin=13;
int analogPin=0;

void setup()
{
  // Die eingebaute Arduino LED (Digital 13), wird hier zur Ausgabe genutzt
  pinMode(ledPin,OUTPUT);

  // Serielle Ausgabe Initialisierung
  Serial.begin(9600);
  Serial.println("Heartbeat Detektion Beispielcode.");
}

const int delayMsec = 60; // 100msec per sample

// Das Hauptprogramm hat zwei Aufgaben:
// - Wird ein Herzschlag erkannt, so blinkt die LED kurz auf
// - Der Puls wird errechnet und auf der serriellen Ausgabe ausgegeben.

void loop()
{
  static int beatMsec = 0;
  int heartRateBPM = 0;
      Serial.println(rawValue);
  if (heartbeatDetected(analogPin, delayMsec)) {
    heartRateBPM = 60000 / beatMsec;
        // LED-Ausgabe bei Herzschlag
    digitalWrite(ledPin,1);

    // Serielle Datenausgabe
    Serial.print(rawValue);
    Serial.print(", ");
    Serial.println(heartRateBPM);

    beatMsec = 0;
  } else {
    digitalWrite(ledPin,0);
  }
  delay(delayMsec);
  beatMsec += delayMsec;
}
```


Anschlussbelegung Arduino:

Sensor Signal	= [Pin 0]
Sensor +V	= [5V]
Sensor -	= [Pin GND]

Beispielprogramm Download

KY-039-HeartBeatDetector original by DanTruong

KY-039-HeartBeatDetector deutsche Version by Joy-It

Codebeispiel Raspberry Pi

!! Achtung !! Analoger Sensor !! Achtung !!

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser *Sensorkit X40* mit dem **KY-053** ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Somit empfehlen wir, bei analogen Sensoren dieses Sets das KY-053 Modul mit dem besagten ADC dazwischenzuschalten. Nähere Informationen finden Sie auf der Informationsseite zum **KY-053** Analog Digital Converter

!! Achtung !! Analoger Sensor !! Achtung !!

Das Programm sieht vor, dass im Abstand der eingestellten "delayTime" (Standard: 10ms) die Funktion zur Herzschlagdetektion aufgerufen wird. Wurde ein Herzschlag erkannt, so wird der Puls ausgegeben. Zusätzlich kann man am eingestellten LED_Pin (Standard: GPIO24) eine LED anschließen, um den detektierten Herzschlag auch visuell auszugeben.

Wird der Finger beim messen neu aufgelegt oder stark bewegt, so kann es etwas dauern (3-5 Sekunden), bis das Programm sich auf die neue Gegebenheit kalibriert und wieder den richtigen Wert ausgibt.

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der BSD-Lizenz [Link] veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

```
#!/usr/bin/python
# coding=utf-8
```



```
##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### Parts of Code based on Dan Truong's KY039 Arduino Heartrate Monitor V1.0
### [https://forum.arduino.cc/index.php?topic=209140.msg2168654] Message #29
##########################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
beatsPerMinute = 0
isPeak = False
result = False
delayTime = 0.01
maxValue = 0
schwelle = 25
beatTime = 0
oldBeatTime = 0

# Adresszuweisung ADS1x15 ADC

ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
# sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel = 0    # Channel 0
# adc_channel = 1    # Channel 1
# adc_channel = 2    # Channel 2
# adc_channel = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz adc = ADS1x15(ic=ADS1115)

# Hier wird der Ausgangs-Pin deklariert, an dem die LED angeschlossen ist.
LED_PIN = 24
GPIO.setup(LED_PIN, GPIO.OUT, initial= GPIO.LOW)
```



```
##########################################################################################

# Hier wird der Ausgangs-Pin deklariert, an dem der Buzzer angeschlossen ist.
def heartBeatDetect(schwelle):
        global maxValue
        global isPeak
        global result
        global oldBeatTime

        # Hier wird der aktuelle Spannungswert am Fototransistor ausgelesen
        # und in der rawValue - Variable zwischengespeichert
        # Mit "adc_channel" wird der am ADC angeschlossene Channel ausgewaehlt
        rawValue = adc.readADCSingleEnded(adc_channel, gain, sps)

        # Reset der Ergebnis Variable
        if result == True:
            result = False

        # Sollte der aktuelle Wert vom letzten maximalen Wert zu weit abweichen
        # (z.B.da der Finger neu aufgesetzt oder weggenommen wurde)
        # So wird der MaxValue resetiert, um eine neue Basis zu erhalten.
# Hier wurd der eigentliche Peak detektiert. Sollte ein neuer RawValue groeßer sein
# als der letzte maximale Wert, so wird das als Spitze der aufgezeichneten Daten erkannt.
        if rawValue * 4 < maxValue:              maxValue = rawValue * 0.8;
        if rawValue > (maxValue - schwelle):

              if rawValue > maxValue:
                    maxValue = rawValue
              # Zum erkannten Peak soll nur ein Herzschlag zugewiesen werden
              if isPeak == False:
                    result = True

              isPeak = True

        else:
            if rawValue < maxValue - schwelle:
              isPeak = False
              # Hierbei wird der maximale Wert bei jedem Durchlauf
              # etwas wieder herabgesetzt. Dies hat den Grund, dass
              # nicht nur der Wert sonst immer stabil bei jedem Schlag
              # gleich oder kleiner als maxValue sein wuerde, sondern auch,
              # falls der Finder sich minimal bewegen sollte und somit
              # das Signal generell schwaecher sein sollte.
            maxValue = maxValue - schwelle/2

       # Wurde in der oberen Abfrage ein Herzschlag detektiert, so wird nun die Ausgabe freigegeben
       if result == True:

           # Berechnung des Puls
           # Hierbei wird bei jedem registrierten Herzschlag die System-Zeit aufgenommen
           # Beim naechsten Herzschlag wird dann die aktuelle Systemzeit mit der gespeicherten verglichen
           # Die Differenz der beiden ergibt dann die Zeit zwischen den Herz-Schlaegen
           # womit man dann auch den Puls berechnen kann.
           beatTime = time.time()
           timedifference = beatTime - oldBeatTime
           beatsPerMinute = 60/timedifference
           oldBeatTime = beatTime

           # Neben der Berechnung des Puls, wird der Herzschlag auch auf eine LED
          # als kurzes Aufblinken ausgegeben
           GPIO.output(LED_PIN, GPIO.HIGH)
           time.sleep(delayTime*10)
           GPIO.output(LED_PIN, GPIO.LOW)

           # Erechneter Puls wird der Funktion übergeben
           return beatsPerMinute
```



```
# ########
# Hauptprogrammschleife
# ########
# Das Programm sieht vor, dass im Abstand der eingestellten "delayTime" (Standard: 10ms)
# die Funktion zur Herzschlagdetektion aufgerufen wird. Wurde ein Herzschlag erkannt,
# so wird der Puls ausgegeben.

try:
        while True:
                time.sleep(delayTime)
                beatsPerMinute = heartBeatDetect(schwelle)
                if result == True:
                    print "---Herzschlag erkannt !--- Puls:", int(beatsPerMinute),"(bpm)"



except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor KY-039

Signal	=	Analog 0	[Pin A0 (ADS1115 - KY-053)]
+V	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

ADS1115 - KY-053:

VDD	=	3,3V	[Pin 01]
GND	=	Masse	[Pin 09]
SCL	=	GPIO03 / SCL	[Pin 05]
SDA	=	GPIO02 / SDA	[Pin 03]
A0	=	s.o.	[Sensor: Signal]

Beispielprogramm Download

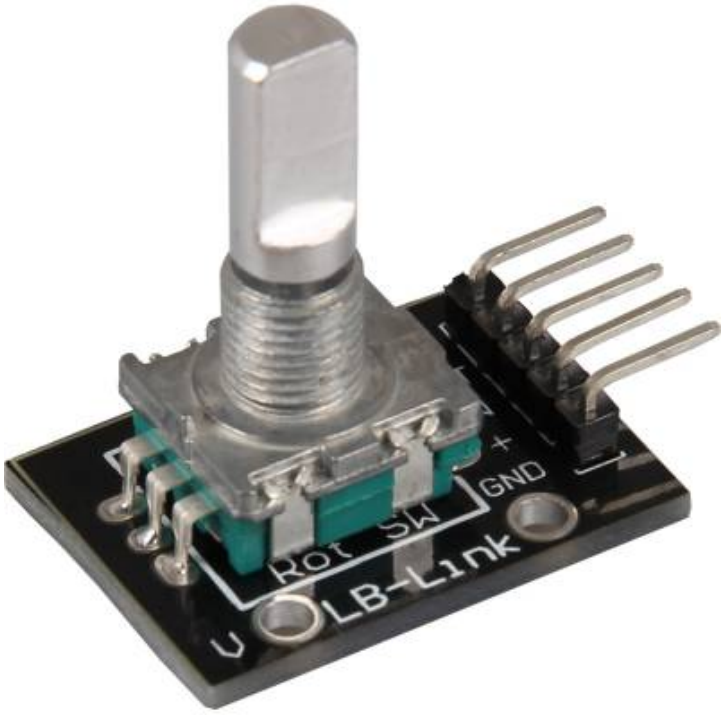
Zu starten mit dem Befehl:

```
sudo python KY-0039_HeartBeatDetector.py
```


KY-040 Kodierter Drehschalter (Rotary Encoder)

Inhaltsverzeichnis

Bild



Technische Daten / Kurzbeschreibung

Die aktuelle Position des Drehschalters wird kodiert über die Ausgänge gegeben.

Kodierung

Die Idee bei einem Drehschalter/Drehgeber ist es, dass zu jedem gedrehten "Schritt", sich der Zustand jeweils immer nur einer der beiden Ausgangs-Pins ändert. Je nachdem welhcer der beiden sich zuerst geändert hat, so kann man auf die Drehrichtung schließen, wenn man auf die folgende Kodierung achtet.

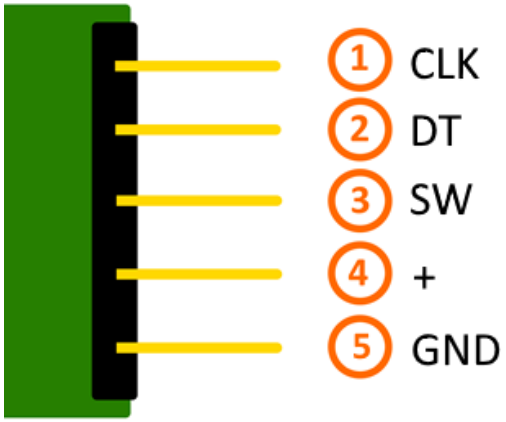
Im Uhrzeigersinn [A ändert sich zuerst] -> Pin_CLK

A	B
0	0
1	0
1	1
0	1
0	0

Gegen den Uhrzeigersinn [B ändert sich zuerst] -> Pin_DT

A	B
0	0
0	1
1	1
1	0
0	0

Pin-Belegung



Codebeispiel Arduino

Das Programm überprüft, falls eine Änderung der Pin-Zustände sich ereignet hat, welcher der beiden Pins sich zuerst geändert hatte, was auf die Drehrichtung schließen lässt. Diese Information erhält man, in dem man einen der beiden Pin-Werte aus einem vorherigen Durchlauf mit dem Wert des aktuellen Durchlaufs vergleicht.

Nachdem die Richtung festgestellt wurde, werden die Schritte von der Startposition an gezählt und ausgegeben. Ein Drücken auf den Knopf des Drehgebers resettet die aktuelle Position.

Für die serielle Ausgabe: Baudrate= 115200

```
// Initialisierung benötigter Variablen
int Counter = 0;
boolean Richtung;
int Pin_clk_Letzter;
int Pin_clk_Aktuell;

// Definition der Eingangs-Pins
int pin_clk = 3;
int pin_dt = 4;
int button_pin = 5;


void setup()
{
   // Eingangs-Pins werden initialisiert...
   pinMode (pin_clk,INPUT);
   pinMode (pin_dt,INPUT);
   pinMode (button_pin,INPUT);

   // ...und deren Pull-Up Widerstände aktiviert
   digitalWrite(pin_clk, true);
   digitalWrite(pin_dt, true);
   digitalWrite(button_pin, true);

   // Initiales Auslesen des Pin_CLK
   Pin_clk_Letzter = digitalRead(pin_clk);
   Serial.begin (115200);
 }

// Das Programm überprüft, falls eine Änderung der Pin-Zustände sich ereignet hat, welcher der beiden
// Pins sich zuerst geändert hatte, was auf die Drehrichtung schließen lässt.
// Diese Information erhält man, in dem man einen der beiden Pin-Werte aus einem vorherigen
// Durchlauf mit dem Wert des aktuellen Durchlaufs vergleicht.
// Nachdem die Richtung festgestellt wurde, werden die Schritte von der Startposition
// an gezählt und ausgegeben. Ein Drücken auf den Knopf des Drehgebers resettet die aktuelle Position.

void loop()
{
   // Auslesen des aktuellen Statuses
   Pin_clk_Aktuell = digitalRead(pin_clk);

   // Überprüfung auf Änderung
   if (Pin_clk_Aktuell != Pin_clk_Letzter)
   {

                if (digitalRead(pin_dt) != Pin_clk_Aktuell)
                {
                        // Pin_CLK hat sich zuerst verändert
                        Counter ++;
                        Richtung = true;
                }

                else
                {       // Andernfalls hat sich Pin_DT zuerst verändert
                        Richtung = false;
                        Counter--;
                }
                Serial.println ("Drehung erkannt: ");
                Serial.print ("Drehrichtung: ");


                if (Richtung)
```



```
                if (Richtung)
                {
                   Serial.println ("Im Uhrzeigersinn");
                }
                else
                {
                   Serial.println("Gegen den Uhrzeigersinn");
                }

                Serial.print("Aktuelle Position: ");
                Serial.println(Counter);
                Serial.println("-----------------------------");

   }

   // Vorbereitung für den nächsten Druchlauf:
   // Der Wert des aktuellen Durchlaufs ist beim nächsten Druchlauf der vorherige Wert
   Pin_clk_Letzter = Pin_clk_Aktuell;

   // Reset-Funktion um aktuelle Position zu speichern
   if (!digitalRead(button_pin) && Counter!=0)
     {
       Counter = 0;
       Serial.println("Position resettet");
     }

 }
```

Anschlussbelegung Arduino:

CLK	=	[Pin 3]
DT	=	[Pin 4]
Button	=	[Pin 5]
+	=	[Pin 5V]
GND	=	[Pin GND]

Beispielprogramm Download

KY-040_RotaryEncoder.zip

Codebeispiel Raspberry Pi

Das Programm überprüft, falls eine Änderung der Pin-Zustände sich ereignet hat, welcher der beiden Pins sich zuerst geändert hatte, was auf die Drehrichtung schließen lässt.Diese Information erhält man, in dem man einen der beiden Pin-Werte aus einem vorherigen Durchlauf mit dem Wert des aktuellen Durchlaufs vergleicht.

Nachdem die Richtung festgestellt wurde, werden die Schritte von der Startposition an gezählt und ausgegeben. Ein Drücken auf den Knopf des Drehgebers resettet die aktuelle Position.

```
# coding=utf-8
# Benoetigte Module werden importiert und eingerichtet
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Hier werden die Eingangs-Pins deklariert, an dem der Sensor angeschlossen ist.
PIN_CLK = 16
```



```
PIN_DT = 15
BUTTON_PIN = 14

GPIO.setup(PIN_CLK, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(PIN_DT, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)

# Benötigte Variablen werden initialisiert
Counter = 0
Richtung = True
PIN_CLK_LETZTER = 0
PIN_CLK_AKTUELL = 0
delayTime = 0.01

# Initiales Auslesen des Pin_CLK
PIN_CLK_LETZTER = GPIO.input(PIN_CLK)

# Diese AusgabeFunktion wird bei Signaldetektion ausgefuehrt
def ausgabeFunktion(null):
    global Counter

    PIN_CLK_AKTUELL = GPIO.input(PIN_CLK)

    if PIN_CLK_AKTUELL != PIN_CLK_LETZTER:

        if GPIO.input(PIN_DT) != PIN_CLK_AKTUELL:
            Counter += 1
            Richtung = True;
        else:
            Richtung = False
            Counter = Counter - 1

        print "Drehung erkannt: "

        if Richtung:
            print "Drehrichtung: Im Uhrzeigersinn"
        else:
            print "Drehrichtung: Gegen den Uhrzeigersinn"

        print "Aktuelle Position: ", Counter
        print "------------------------------"

def CounterReset(null):
    global Counter

    print "Position resettet!"
    print "------------------------------"
    Counter = 0

# Um einen Debounce direkt zu integrieren, werden die Funktionen zur Ausgabe mittels
# CallBack-Option vom GPIO Python Modul initialisiert
GPIO.add_event_detect(PIN_CLK, GPIO.BOTH, callback=ausgabeFunktion, bouncetime=50)
GPIO.add_event_detect(BUTTON_PIN, GPIO.FALLING, callback=CounterReset, bouncetime=50)


print "Sensor-Test [druecken Sie STRG+C, um den Test zu beenden]"

# Hauptprogrammschleife
try:
        while True:
            time.sleep(delayTime)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
        GPIO.cleanup()
```


Anschlussbelegung Raspberry Pi:

CLK	=	GPIO16	[Pin 36]
DT	=	GPIO15	[Pin 10]
SW	=	GPIO14	[Pin 8]
+	=	3,3V	[Pin 1]
GND	=	Masse	[Pin 6]

Beispielprogramm Download

KY-040_RPi_RotaryEncoder.zip

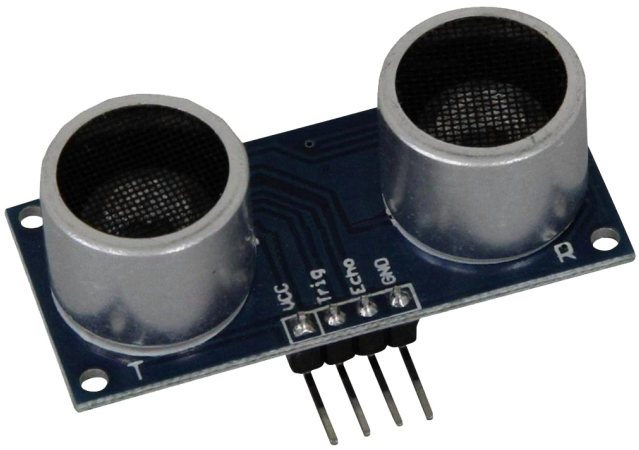
Zu starten mit dem Befehl:

```
sudo python KY-040_RPi_RotaryEncoder.py
```


KY-050 Ultraschallabstandssensor

Inhaltsverzeichnis

Bild



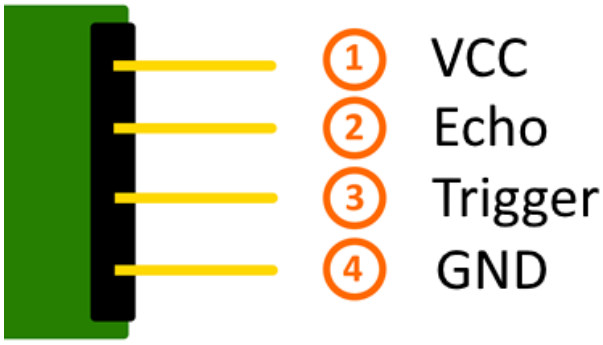
Technische Daten / Kurzbeschreibung

Wird am Trigger-Eingang ein Signal (fallende Flanke) eingegeben, so wird eine Abstandsmessung durchgeführt und am Echo-Ausgang als PWM-TTL Signal ausgegeben

messbare Distanz: 2cm—300cm **Messauflösung**: 3mm

min. Zeit zwischen den Messungen 50µs

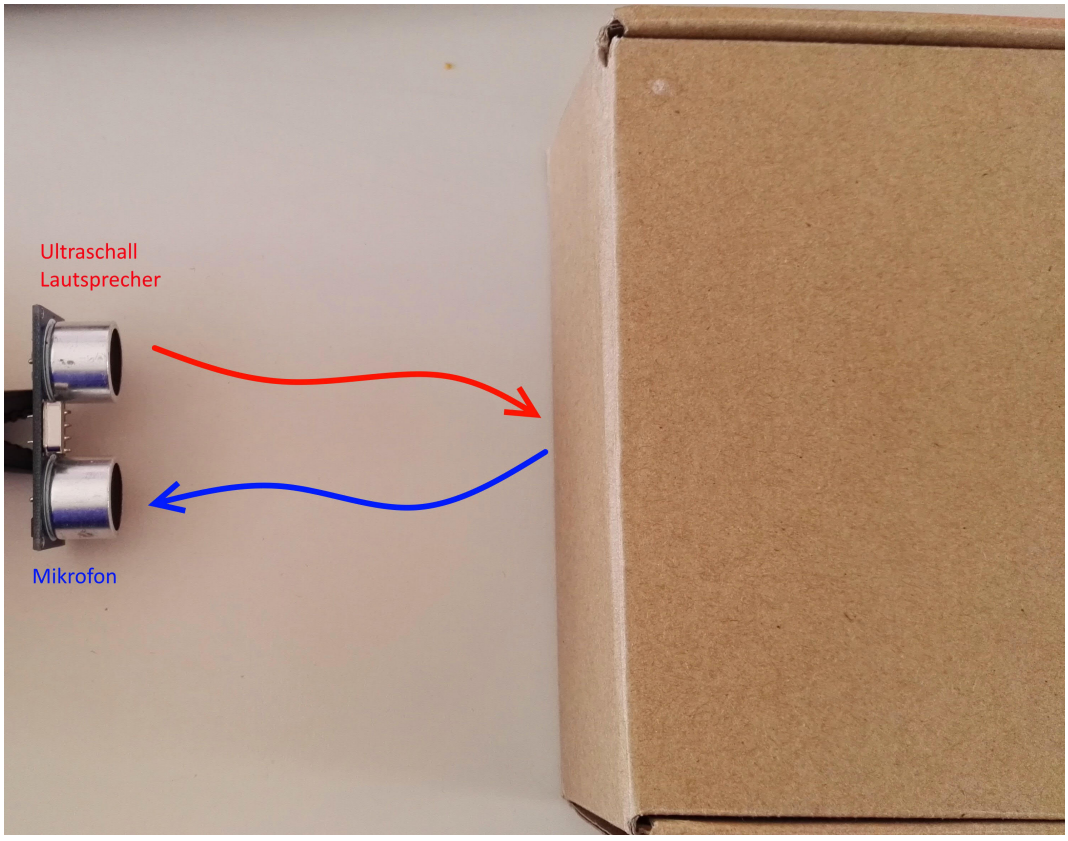
Pin-Belegung



Funktionsprinzip

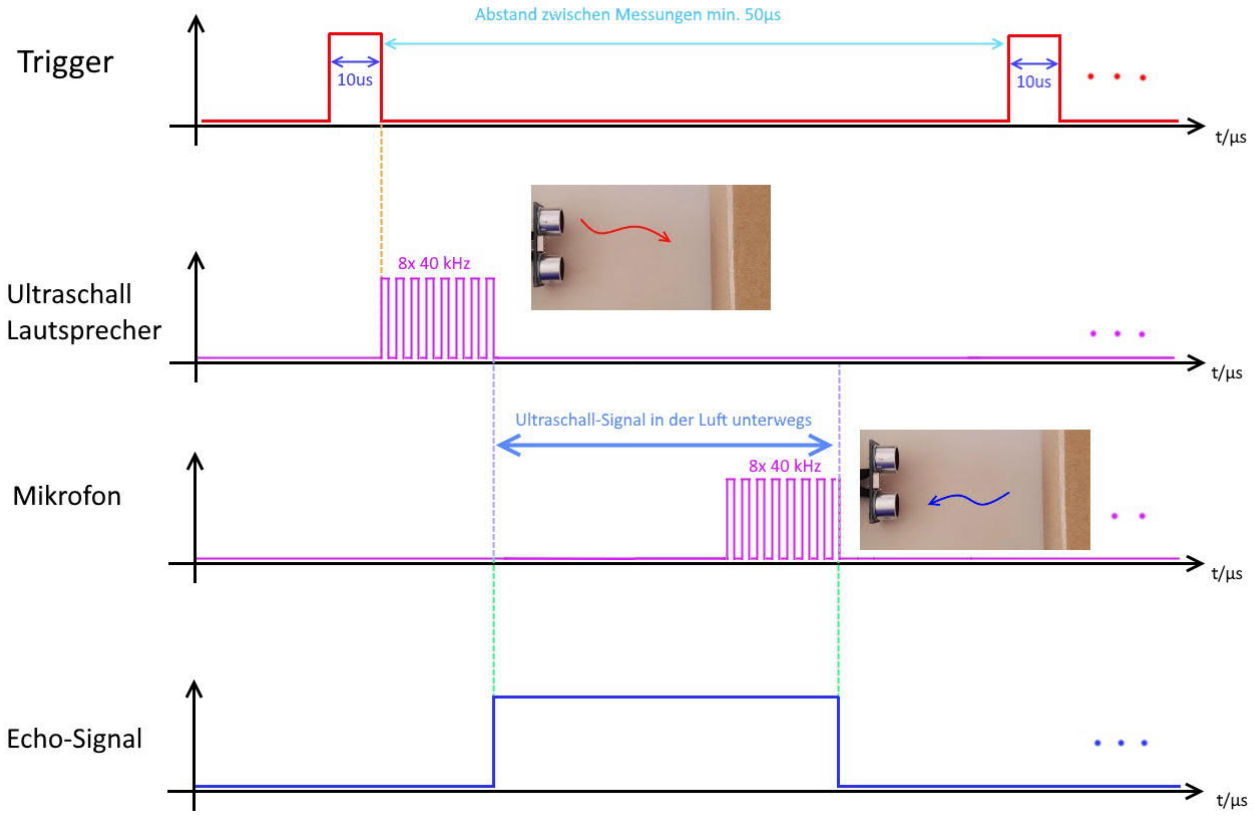
Dieses Modul zeigt auf, wie man mittels eines Ultraschalllautsprechers und eines Mikrofons den Abstand berührungslos zu einem Objekt messen kann. Das Prinzip basiert darauf, dass die Schallgeschwindigkeit in der Luft bei gleichbleibender Temperatur nahezu konstant bleibt - bei 20°C beträgt sie 343,2m/s.

Aus diesem Fakt kann man die Abstandsmessung in eine Zeitmessung überführen, welche dann von Mikrokontrollern einfach übernommen werden kann.



Im hier vorgestellten Sensormodul sendet der Ultraschalllautsprecher acht 40KHz Signale aus, welche dann von einem Gegenstand reflektiert und vom Mikrofon aufgenommen werden können. Ultraschall wird verwendet, da es sich außerhalb des Hörbereiches des menschlichen Gehörsinns befindet (grob 20Hz-22.000 Hz).

Das Aussenden des Ultraschallsignals wird gestartet, in dem am "Trigger Eingangs-Pin" ein 10µs langes Startsignal (ActiveHigh) empfangen wird. Nach dem Aussenden wird am "Echo Ausgang-Signal Pin" das Signal aktiviert (ActiveHigh). Wird nun am Mikrofon das reflektierte Signal wieder aufgenommen, so wird nach der Detektion das Echo-Signal wieder deaktiviert. Die Zeit zwischen der Aktivierung und der Deaktivierung des Echosignals kann gemessen und in den Abstand umgerechnet werden, da dies auch der Zeit entspricht, wie lang das Ultraschallsignal gebraucht hat um in der Luft die Strecke zwischen Lautsprecher->reflektierende Wand -> Mikrofon zu überwinden. Die Umrechnung erfolgt dann über die Annäherung einer konstanten Luftgeschwindigkeit - der Abstand ist dann folglich die Hälfte der zurückgelegten Strecke.



Codebeispiel Arduino

Das Beispielprogramm aktiviert nach o.g. Prinzip die Abstandsmessung und misst mit Hilfe der Arduino Funktion pulseIn die Zeit, wie lang das Ultraschallsignal in der Luft ist. Diese Zeit wird dann für die Umrechnung des Abstands als Basis genommen - das Ergebnis wird danach in der seriellen Ausgabe ausgegeben. Sollte das Signal außerhalb des Messbereichs sein, wird eine entsprechende Fehlermeldung ausgegeben.


```
#define Echo_EingangsPin 7 // Echo Eingangs-Pin
#define Trigger_AusgangsPin 8 // Trigger Ausgangs-Pin

// Benoetigte Variablen werden definiert
int maximumRange = 300;
int minimumRange = 2;
long Abstand;
long Dauer;

void setup() {
 pinMode(Trigger_AusgangsPin, OUTPUT);
 pinMode(Echo_EingangsPin, INPUT);
 Serial.begin (9600);
}

void loop() {

 // Abstandsmessung wird mittels des 10us langen Triggersignals gestartet
 digitalWrite(Trigger_AusgangsPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(Trigger_AusgangsPin, LOW);

 // Nun wird am Echo-Eingang gewartet, bis das Signal aktiviert wurde
 // und danach die Zeit gemessen, wie lang es aktiviert bleibt
 Dauer = pulseIn(Echo_EingangsPin, HIGH);

 // Nun wird der Abstand mittels der aufgenommenen Zeit berechnet
 Abstand = Dauer/58.2;

 // Überprüfung ob gemessener Wert innerhalb der zulässingen Entfernung liegt
 if (Abstand >= maximumRange || Abstand <= minimumRange)
 {
    // Falls nicht wird eine Fehlermeldung ausgegeben.
      Serial.println("Abstand außerhalb des Messbereichs");
      Serial.println("---------------------------------");
 }

 else
 {
    // Der berechnete Abstand wird in der seriellen Ausgabe ausgegeben
      Serial.print("Der Abstand betraegt:");
      Serial.print(Abstand);
      Serial.println("cm");
      Serial.println("---------------------------------");
 }
  // Pause zwischen den einzelnen Messungen
 delay(500);
}
```

Anschlussbelegung Arduino:

VCC	= [Pin 5V]
Echo	= [Pin 7]
Trigger	= [Pin 8]
Sensor GND	= [Pin GND]

Beispielprogramm Download

KY-050-UltraschallabstandSensor.zip

Codebeispiel Raspberry Pi

!! Achtung !! 5V Spannungslevel !! Achtung !!

Der Raspberry Pi arbeitet mit seinem ARM-Prozessorkern, anders als der auf Atmel Atmega basierende Arduino, mit 3,3V Spannungslevel, anstatt mit 5V - dieser Sensor funktioniert jedoch nur mit dem höheren Spannungslevel. Würde man den Sensor uneingeschränkt am Raspberry Pi ohne Vorsichtsmaßnahmen betreiben, könnten dies bei den Eingängen des Raspberry Pi's permanente Schäden hervorrufen.

Für solche Fälle Grund besitzt dieses Sensorkit-Set mit dem KY-051 einen Voltage-Translator, welcher die Spannungslevel anpasst und somit einen sicheren Betrieb gewährleistet. Dieser muss bei diesem Sensor zwischen den Rasperry Pi und dem Sensor zwischengeschaltet sein.

Nähere Informationen entnehmen Sie der Informationsseite zum KY-051 Voltage Translator / Level Shifter

!! Achtung !! 5V Spannungslevel !! Achtung !!

Das Beispielprogramm aktiviert nach o.g. Prinzip die Abstandsmessung und misst mit Hilfe einer Art Stoppuhr die Zeit, wie lang das Ultraschallsignal in der Luft ist. Diese Stoppuhr wird realisiert, indem beim Zeitpunkt des Umschaltens des Echosignals, die aktuelle Systemzeit aus time.time() herausgelesen wird; die Differenz zwischen der Einschaltzeit und der Ausschaltzeit ist die gesuchte Zeit wie lang das Signal unterwegs ist.
Diese Zeit wird dann für die Umrechnung des Abstands als Basis genommen - das Ergebnis wird danach in der Kosnole ausgegeben. Sollte das Signal außerhalb des Messbereichs sein, wird eine entsprechende Fehlermeldung ausgegeben.

```
# coding=utf-8
# Benötigte Module werden eingefügt und konfiguriert
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

# Hier können die jeweiligen Eingangs-/Ausgangspins ausgewählt werden
Trigger_AusgangsPin = 17
Echo_EingangsPin    = 27

# Die Pause zwischen den einzelnen Messugnen kann hier in Sekunden eingestellt werden
sleeptime = 0.8

# Hier werden die Ein-/Ausgangspins konfiguriert
GPIO.setup(Trigger_AusgangsPin, GPIO.OUT)
GPIO.setup(Echo_EingangsPin, GPIO.IN)
GPIO.output(Trigger_AusgangsPin, False)

# Hauptprogrammschleife
try:
    while True:
        # Abstandsmessung wird mittels des 10us langen Triggersignals gestartet
        GPIO.output(Trigger_AusgangsPin, True)
        time.sleep(0.00001)
        GPIO.output(Trigger_AusgangsPin, False)

        # Hier wird die Stopuhr gestartet
        EinschaltZeit = time.time()
        while GPIO.input(Echo_EingangsPin) == 0:
```



```
            EinschaltZeit = time.time() # Es wird solange die aktuelle Zeit gespeichert, b

        while GPIO.input(Echo_EingangsPin) == 1:
            AusschaltZeit = time.time() # Es wird die letzte Zeit aufgenommen, wo noch das

        # Die Differenz der beiden Zeiten ergibt die gesuchte Dauer
        Dauer = AusschaltZeit - EinschaltZeit
        # Mittels dieser kann nun der Abstand auf Basis der Schallgeschwindigkeit der Abst
        Abstand = (Dauer * 34300) / 2

        # Überprüfung, ob der gemessene Wert innerhalb der zulässigen Entfernung liegt
        if Abstand < 2 or (round(Abstand) > 300):
            # Falls nicht wird eine Fehlermeldung ausgegeben
            print("Abstand außerhalb des Messbereich")
            print("------------------------------")
        else:
            # Der Abstand wird auf zwei Stellen hinterm Komma formatiert
            Abstand = format((Dauer * 34300) / 2, '.2f')
            # Der berechnete Abstand wird auf der Konsole ausgegeben
            print("Der Abstand beträgt:"), Abstand,("cm")
            print("------------------------------")

        # Pause zwischen den einzelnen Messungen
        time.sleep(sleeptime)

# Aufraeumarbeiten nachdem das Programm beendet wurde
except KeyboardInterrupt:
    GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

Sensor KY-050:

VCC	=	5V	[Pin 2 (RPi)]
Trigger	=	Pin B1	[KY-051-Voltage Translator]
Echo	=	Pin B2	[KY-051-Voltage Translator]
GND	=	Masse	[Pin 6 (RPi)]

KY-051- Voltage Translator:

VCCb	=	5V	[Pin 04(RPi)]
Pin B1	=	Trigger	[KY-050-UltraschallSensor]
Pin B2	=	Echo	[KY-050-UltraschallSensor]
VCCa	=	3,3V	[Pin 01(RPi)]
Pin A1	=	GPIO17	[Pin 11(RPi)]
Pin A2	=	GPIO27	[Pin 13(RPi)]
GND	=	Masse	[Pin 06(RPi)]

- Alle restlichen Pins am KY-051-Voltage-Translator-Modul müssen nicht angeschlossen werden (OE,B3,B4, A3,A4).

Beispielprogramm Download

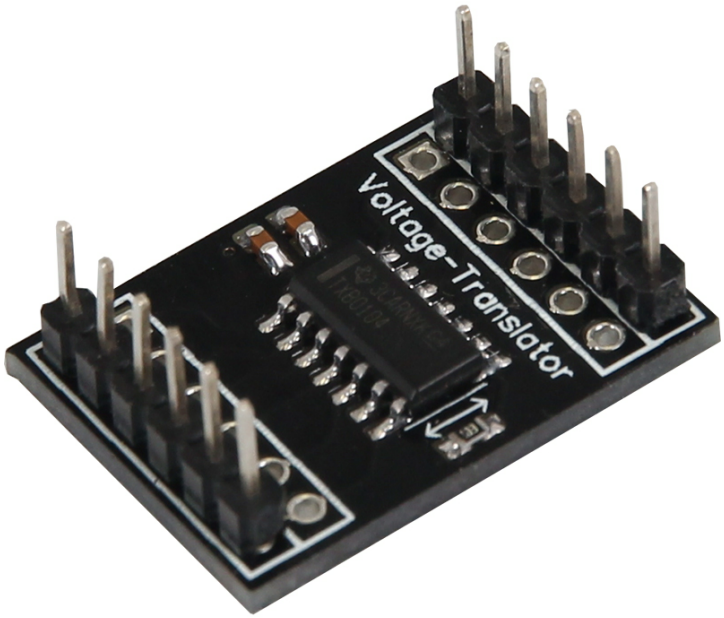
KY-050-RPi_UltraschallAbstandSensor.zip

Zu starten mit dem Befehl:


```
sudo python KY-050-RPi_UltraschallAbstandSensor.py
```


KY-051 Voltage Translator / Level Shifter

Bild



Technische Daten / Kurzbeschreibung

Dieser Level-Shifter wandelt digitale Signale von einer Spannung in eine andere herunter bzw. herauf. Hierzu gibt es 4 verfügbare Kanäle, die umgewandelt werden können.

In der heutigen Zeit gibt es eine Vielzahl ein Mikrokontrollersystemen, die in verschiedenen Spannungsbereichen operieren: So arbeiten z.B. die Ein- und Ausgänge von älteren Systemen wie Arduino, basierend auf einen Atmega-Controller, mit 5V-Level, die neueren Raspberry-Pi Computer die auf einen ARM-Controller basieren, mit einem Spannungslevel von 3,3V.

Dies ist damit zu begründen, dass bei der Kommunikation zwischen Mikrokontrollern früher mit höheren Spannungen gearbeitet werden musste, da die Eingangsstufen aufgrund von Rauschen/Störungen in den Leitungen einen höheren Abstand zwischen dem Spannungslevel für [Digital EIN] = 5V und für [Digital AUS] = 0V benötigt haben - Im Zuge der Modernisierung sind Ein-/Ausgangsstufen der Kontroller bedeutend besser geworden und man möchte heutzutage die eingesetzte Spannung so gering wie möglich halten, damit Wärmeentwicklung und Stromverbauch sinken. So ist heutzutage auch der Einsatz von 1,8V Systemen nicht unüblich.

Will man jedoch zwischen zwei Systemen mit zwei verschiedenen Spannungslevel kommunizieren (wie das untere Beispiel Arduino -> Raspberry Pi), so muss der Spannungslevel "geshiftet" werden - Wird dieses nicht getan, so muss dasjenige System mit dem niedrigeren Spannungslevel die überschüssige Spannung an den Eingangsstufen "in Wärme verbrauchen". Dieses kann je nach System zu dauerhaften Schäden am System führen.

Pin-Belegung

Die Pin-Belegung ist auf der Modulplatine aufgedruckt.

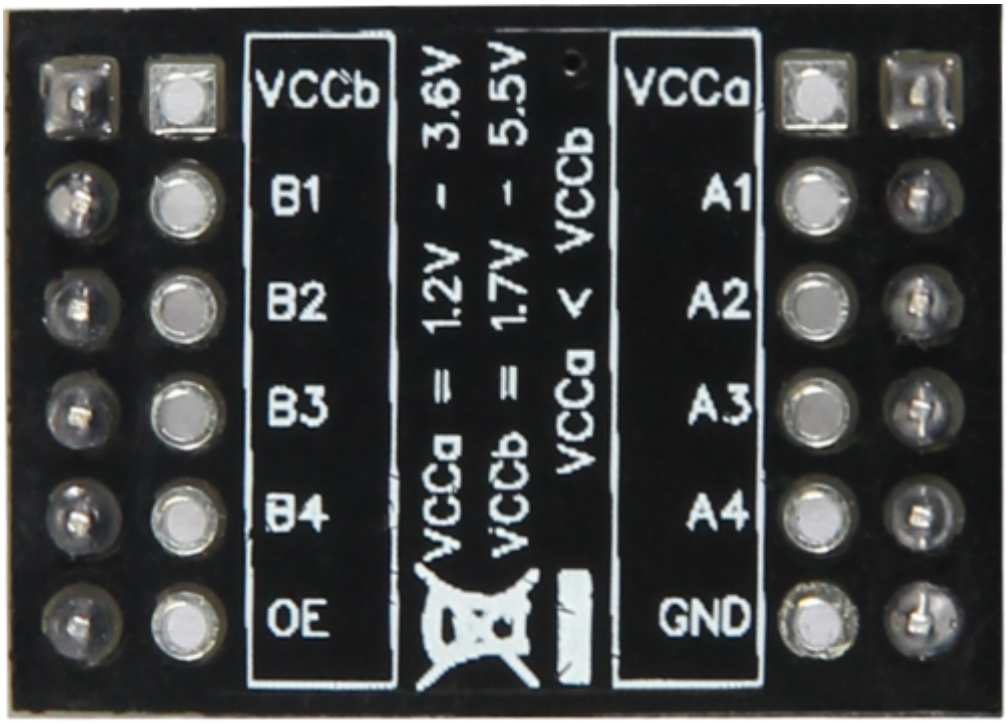
Die Signale an den Ein-/Ausgängen A1-A4, sowie B1-B4 werden auf das jeweilige Spannungslevel geshiftet (VCCa -> A1-A4 | VCCb -> B1-B4)

Beispiel:

Arduino Ausgang -> Digital [EIN] = 5V @ B1 >>>>>>> 3.3V @ B2 -> Raspberry Pi Eingang

Eine zusätzliche Software bzw. Code wird nicht zum Betrieb benötigt; das Modul arbeitet autonom.

Bitte beachten Sie, dass VCCb größer/gleich sein muss als VCCa (Beispiel VCCb=5V - VCCa=3,3V)



Beispiel Anschluss-Belegung zwischen Arduino Raspberry Pi:

Anschlussbelegung Arduino:

VCCb	=	[Pin 5V]
B1	=	[Pin 03]
B2	=	[Pin 04]
B3	=	[Pin 05]
B4	=	[Pin 06]
GND	=	[Pin GND]

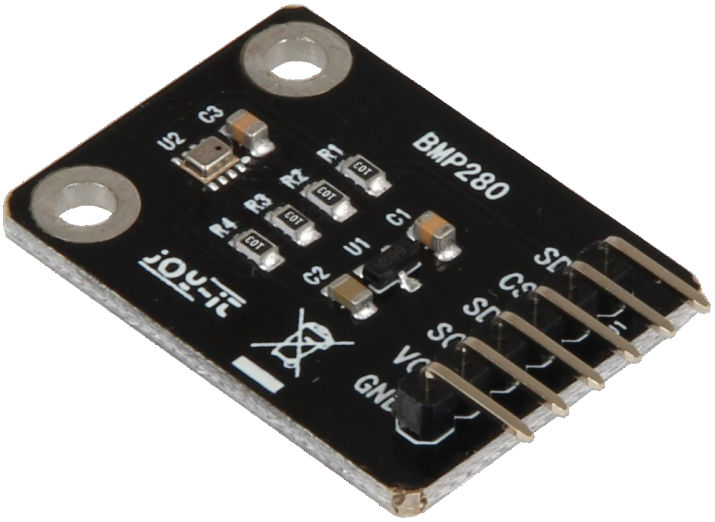
Anschlussbelegung Raspberry Pi:

VCCa	=	3,3V	[Pin 01]
A1	=	GPIO18	[Pin 12]
A2	=	GPIO03 / SCL	[Pin 05]
A3	=	GPIO02 / SDA	[Pin 01]
A4	=	GPIO14	[Pin 08]
GND	=	GND	[Pin 09]

Bitte achten Sie, dass beide Systeme am selben GND (Masse) verbunden sind - OE muss bei diesem Modul nicht verbunden werden.

KY-052 Drucksensor / Temperatursensor - BMP280 -

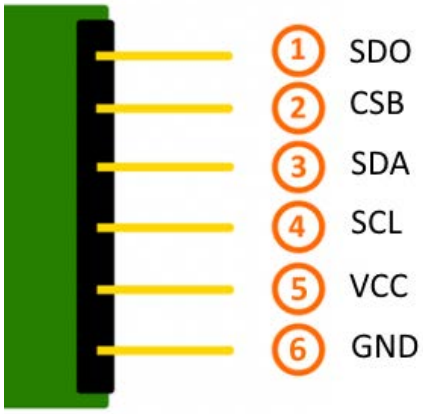
Bild



Technische Daten / Kurzbeschreibung

Dieser Drucksensor misst den Luftdruck am Sensorausgang (kleines Loch am silbernen Sensorgehäuse) und gibt das Ergebnis kodiert auf den I2C-Bus aus. **Für diesen Sensor wird eine entsprechende Software benötigt.**

Pin-Belegung



*Dieser Sensor erlaubt es sowohl an 5V System, sowie an 3,3V Systemen angeschlossen und betrieben zu werden. Hierbei ist zu beachten, dass nur einer der jeweiligen Spannungsversorgungspins angeschlossen wird; passend zum Spannungslevel des verwendeten Systems.

Software-Beispiel Arduino

Dieser Sensor gibt sein Messergebnis nicht als Signal auf einen Ausgangspin aus, sondern kommuniziert diesen per I2C-Bus. Über diesen lässt sich der Sensor ansteuern und die jeweiligen Messungen zum Druck und der Temperatur starten und auswerten. Zur Ansteuerung dieses Sensormoduls gibt es mehrere Möglichkeiten - als besonders zugänglich hat sich die Adafruit_BMP280 Library erwiesen, die die Firma Adafruit unter dem folgenden Link unter der OpenSource BSD-Lizenz BSD-Lizenz veröffentlicht hat.
Das unten stehende Beispiel verwendet diese besagte Library - hierzu empfehlen wir diese von Github herunterzuladen, zu entpacken und im Arduino-Library-Ordner, welcher sich standardmäßig unter (C:\Benutzer\[Benutzername]\Dokumente\Arduino\libraries) befindet, zu kopieren, damit diese für dieses Codebeispiel und folgende Projekte zur Verfügung steht. Alternativ ist diese auch im unten stehenden Download Paket ebenfalls enthalten.

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>

#define BMP_SCK 13
#define BMP_MISO 12
```

```
#define BMP_MOSI 11
#define BMP_CS 10

Adafruit_BMP280 bmp; // I2C
//Adafruit_BMP280 bmp(BMP_CS); // hardware SPI
//Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO,  BMP_SCK);

void setup() {
  Serial.begin(9600);
  Serial.println(F("BMP280 test"));
if (!bmp.begin()) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
    while (1);
  }
}

void loop() {
    Serial.print(F("Temperature = "));
    Serial.print(bmp.readTemperature());
    Serial.println(" *C");

    Serial.print(F("Pressure = "));
    Serial.print(bmp.readPressure());
    Serial.println(" Pa");

    Serial.print(F("Approx altitude = "));
    Serial.print(bmp.readAltitude(1013.25)); // this should be adjusted to your local forcase
    Serial.println(" m");

    Serial.println();
    delay(2000);
}
```

Beispielprogramm Download: KY-052.zip

Anschlussbelegung Arduino:

SDO = [Pin 5V]
CSB = [Pin 5V]
SDA = [Pin A4]
SCL = [Pin A5]
VCC = [Pin 5V]
GND = [Pin GND]

Software-Beispiel Raspberry Pi

Das Programm nutzt zur Ansteuerung des BMP280, der auf diesem Sensor-Modul verbaut ist, die entsprechenden BMP280 und I2C Python-Libraries der Firma Adafruit, welche anschließend von Bastien Wirtz modifiziert wurden. Diese wurden unter dem folgenden Link unter der MIT OpenSource-Lizenz veröffentlicht.

Diese muss vorab erst installiert werden. Hierzu muss folgendermaßen vorgegangen werden:

Zuerst muss, falls dies nicht auf dem Raspberry Pi geschehen ist, die GitHub-Software installiert werden:

sudo apt-get install git

Hierzu muss der Raspberry Pi mit dem Internet verbunden sein. Mit dem Befehl...

git clone https://github.com/bastienwirtz/Adafruit_Python_BMP.git

... kann die aktuelle Version der Adafruit_BM280 Library heruntergeladen und entpackt werden

Danach wechseln wir mit...

cd Adafruit_Python_BMP/

... in den heruntergeladenen Ordner und installieren mit...

sudo python setup.py install

... die Library. Hiernach kann die Library genutzt werden.

Damit der Raspberry Pi mit dem Sensor auf dem I2C-Bus kommunizieren kann, muss auch vorab die I2C-Funktion beim Raspberry Pi aktiviert werden. Hierzu müssen folgende Zeilen am Ende der Datei "/boot/config.txt" hinzugefügt werden:

dtparam=i2c_arm=on

Die Datei kann mit folgenden Befehl editiert werden:

sudo nano /boot/config.txt

Mit der Tastenfolge [Strg+X -> Y -> Enter] kann die Datei, nach dem hinzufügen der Zeile am unteren Ende, gespeichert und geschlossen werden.

Außerdem werden zusätzliche Bibliotheken benötigt, um I2C innerhalb Python nutzen zu können. Um diese zu installieren muss folgender Befehl in die Konsole eingegeben werden:

sudo apt-get install python-smbus i2c-tools -y

Hiernach kann das folgende Python-Code Beispiel verwendet werden. Das Programm startet die Messung am Sensor und gibt die gemessenen Werte für den Luftdruck, der Temperatur und der Höhe überm Meeresspiegel aus.

```
#!/usr/bin/python
# Author: Bastien Wirtz <bastien.wirtz@gmail.com>

# Can enable debug output by uncommenting:
#import logging
#logging.basicConfig(level=logging.DEBUG)

import Adafruit_BMP.BMP280 as BMP280

sensor = BMP280.BMP280()

print 'Temp = {0:0.2f} *C'.format(sensor.read_temperature())
print 'Pressure = {0:0.2f} Pa'.format(sensor.read_pressure())
print 'Altitude = {0:0.2f} m'.format(sensor.read_altitude())
print 'Sealevel Pressure = {0:0.2f} Pa'.format
(sensor.read_sealevel_pressure())
```

Anschlussbelegung Raspberry Pi:

SDO	=	3.3V	[Pin 01]
CSB	=	3.3V	[Pin 01]
SDA	=	GPIO02 / SCA	[Pin 03]
SCL	=	GPIO03 / SCL	[Pin 05]
VCC	=	3,3V	[Pin 01]
GND	=	Masse	[Pin 06]

Beispielprogramm Download: KY-052-RPi.zip

Zu starten mit dem Befehl:

sudo python KY-052-RPi_DruckSensor_TemperaturSensor.py

KY-053 Analog Digital Converter

Inhaltsverzeichnis

Bild

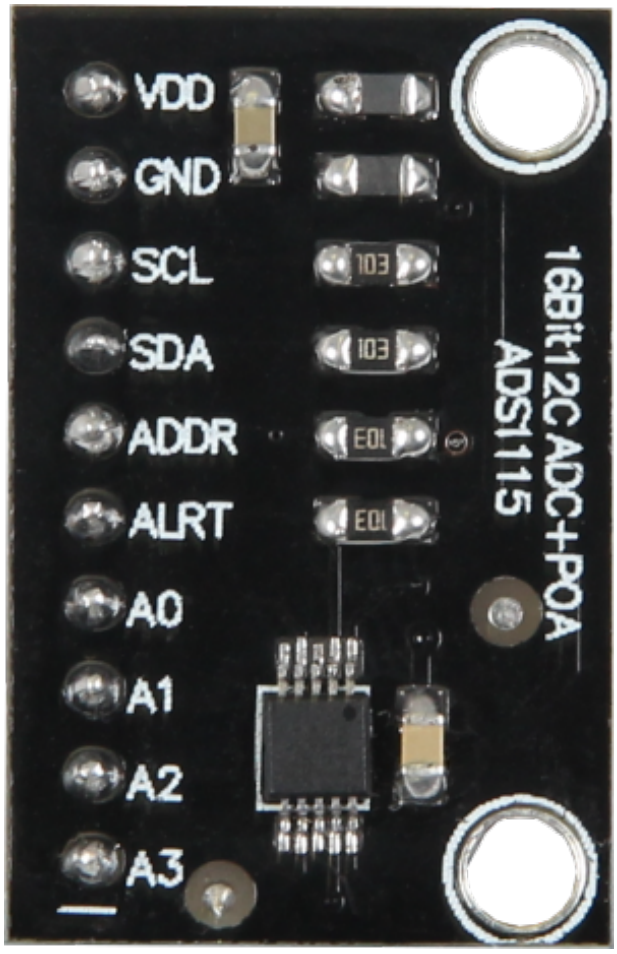
Technische Daten / Kurzbeschreibung

Per entsprechenden Befehlen auf den I2C-Bus, können auf bis zu 4 Eingängen analoge Spannungswerte mit bis zu 16-Bit Genauigkeit gemessen werden. Das Messergebnis wird kodiert auf den I2C-Bus ausgegeben.

Für dieses Modul wird eine entsprechende Software benötigt - siehe Codebeispiele unten.

Pin-Belegung

Die Pin-Belegung ist auf der Modulplatine aufgedruckt



Codebeispiel Arduino

Die Arduino-Boards besitzen von Haus aus einen 10 Bit-ADC mit 6 Kanälen. Benötigt man jedoch mehr Kanäle oder eine höhere Genauigkeit, dann kann man den Arduino mittels des KY-053 Analog Digital Converter Moduls um 4 ADC Kanäle mit 12-Bit Genauigkeit erweitern, welches per I2C an den Arduino angeschlossen wird.

Zur Ansteuerung dieses Moduls gibt es mehrere Möglichkeiten - als besonders zugänglich haben sich die ADS1X15 Libraries erwiesen, die die Firma Adafruit unter [https://github.com/adafruit/Adafruit_ADS1X15] unter der [BSD-Lizenz] veröffentlicht hat.

Das unten stehende Beispiel verwendet diese besagte Library - hierzu empfehlen wir diese von Github herunterzuladen, zu entpacken und im Arduino-Library-Ordner, welcher sich standardmäßig unter (C:\Benutzer\[Benutzername]\Dokumente\Arduino\libraries) befindet, zu kopieren, damit diese für dieses Codebeispiel und folgende Projekte zur Verfügung steht. Alternativ ist diese auch im unten stehenden Download Paket ebenfalls enthalten.


```
#include <Wire.h>
#include <Adafruit_ADS1015.h>

// ADS1115 Modul wird initialisiert - alle folgenden Operationen mit dem ADC
// können mit Hilfe des Objektes "ads" ausgeführt werden.
Adafruit_ADS1115 ads;

void setup(void)
{
  Serial.begin(9600);

  Serial.println("Werte der analogen Eingaenge des ADS1115 (A0..A3) werden ausgelesen und ausge
  Serial.println("ADC Range: +/- 6.144V (1 bit = 0.1875mV)");

  // Dieses Modul besitzt an seinen analogen Eingängen Signalverstärker, deren
  // Verstärkung per Software in den unten stehenden Bereichen konfiguriert werden
  // können.
  // Dieses ist in dem Fall gewünscht, wenn ein bestimmter Spannungsbereich
  // als messergebnis erwartet wird und man so eine höhere Auflösung des Signals
  // erhält.
  // Als Standardverstärkung ist Gain=[2/3] gewählt und kann durch Auskommentieren
  // auf eine andere Verstärkung umgestellt werden.
  //                                                                ADS1115
  //                                                                -------
  ads.setGain(GAIN_TWOTHIRDS);  // 2/3x gain +/- 6.144V  1 bit = 0.1875mV
  // ads.setGain(GAIN_ONE);        // 1x gain   +/- 4.096V  1 bit = 0.125mV
  // ads.setGain(GAIN_TWO);        // 2x gain   +/- 2.048V  1 bit = 0.0625mV
  // ads.setGain(GAIN_FOUR);       // 4x gain   +/- 1.024V  1 bit = 0.03125mV
  // ads.setGain(GAIN_EIGHT);      // 8x gain   +/- 0.512V  1 bit = 0.015625mV
  // ads.setGain(GAIN_SIXTEEN);    // 16x gain  +/- 0.256V  1 bit = 0.0078125mV

  ads.begin();
}

void loop(void)
{
  uint16_t adc0, adc1, adc2, adc3;
  float voltage0, voltage1, voltage2, voltage3;
  float gain_conversion_factor;

  // Der Befehl "ads.readADC_SingleEnded(0)" ist die eigentliche Operation, die die Messung im ADC startet.
  // die "0" als Variable für diese Funktion definiert den verwendeten Channel, der gemessen werden soll
  // Soll z.B. der dritte Channel gemessen werden, so muss diese mit der "3" ausgetauscht werden
  adc0 = ads.readADC_SingleEnded(0);
  adc1 = ads.readADC_SingleEnded(1);
  adc2 = ads.readADC_SingleEnded(2);
  adc3 = ads.readADC_SingleEnded(3);

  // Dieser Wert wird für die Umrechnung in eine Spannung benötigt - dieser ´
  // hängt von der eingestellten Verstärkung ab. Der passende Wert zur Verstärkung
  // sollte aus der oben aufgezeigten Tabelle entnommen werden
  gain_conversion_factor= 0.1875;

  // Umrechnung der aufgezeichneten Werte in eine Spannung
  voltage0 = (adc0 * gain_conversion_factor);
  voltage1 = (adc1 * gain_conversion_factor);
  voltage2 = (adc2 * gain_conversion_factor);
  voltage3 = (adc3 * gain_conversion_factor);

  // Ausgabe der Werte auf die serielle Schnittstelle
    Serial.print("Analog   Eingang   0:   ");   Serial.print(voltage0);Serial.println("mV");
    Serial.print("Analog   Eingang   1:   ");   Serial.print(voltage1);Serial.println("mV");
    Serial.print("Analog   Eingang   2:   ");   Serial.print(voltage2);Serial.println("mV");
```



```
  Serial.print("Analog Eingang 3: "); Serial.print(voltage3);Serial.println("mV");
  Serial.println("-----------------------");

  delay(1000);
}
```

Beispielprogramm Download:

KY-053-AnalogDigitalConverter.zip

Anschlussbelegung Arduino:

VDD = [Pin 5V]
GND = [Pin GND]
SCL = [Pin SCL]
SDA = [Pin SDA]
ADDR = [N.C.]
ALRT = [N.C.]
A0 = [Messspitze Analog 0]
A1 = [Messspitze Analog 1]
A2 = [Messspitze Analog 2]
A3 = [Messspitze Analog 3]

Codebeispiel Raspberry Pi

Der Raspberry Pi besitzt im Gegensatz zum Arduino keine analogen Eingänge bzw. es ist kein ADC (analog digital Converter) im Chip des Raspberry Pi's integriert. Dies schränkt den Raspberry Pi ein, wenn man Sensoren einsetzen möchte, wo nicht digital Werte ausgegeben werden [Spannungswert überschritten -> digital EIN | Spannungswert unterschritten -> digital AUS | Beispiel: Knopf gedrückt [EIN] Knopf losgelassen [AUS]], sondern es sich hier um einen kontinuierlichen veränderlichen Wert handeln sollte (Beispiel: Potentiometer -> Andere Position = Anderer Spannungswert)

Um diese Problematik zu umgehen, besitzt unser Sensorkit X40 mit dem KY-053 ein Modul mit 16 Bit genauen ADC, welches Sie am Raspberry nutzen können, um diesen um 4 analoge Eingänge erweitern zu können. Dieses wird per I2C an den Raspberry Pi angeschlossen, übernimmt die analoge Messung und gibt den Wert digital an den Raspberry Pi weiter.

Das Programm nutzt zur Ansteuerung des ADS1115 ADC die entsprechenden ADS1x15 und I2C Python-Libraries der Firma Adafruit. Diese wurden unter dem folgenden Link [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code] unter der MIT OpenSource-Lizenz veröffentlicht. Die benötigten Libraries sind im unteren Download-Paket enthalten.

Das Programm liest die aktuelle Spannung aus, die an den 4 Kanälen des ADS1115 ADC anliegen, und zeigt diese in der Konsole an. Über die Variable "delayTime", lässt sich die Pause zwischen den Messungen einstellen.

Damit der Raspberry Pi mit dem Sensor auf dem I2C-Bus kommunizieren kann, muss vorab die I2C-Funktion beim Raspberry Pi aktiviert werden. Hierzu müssen folgende Zeilen am Ende der Datei "/boot/config.txt" hinzugefügt werden:

```
dtparam=i2c_arm=on
```


Die Datei kann mit folgenden Befehl editiert werden:

```
sudo nano /boot/config.txt
```

Mit der Tastenfolge [Strg+X -> Y -> Enter] kann die Datei, nach dem hinzufügen der Zeile am unteren Ende, gespeichert und geschlossen werden.

Außerdem werden zusätzliche Bibliotheken benötigt, um I2C innerhalb Python nutzen zu können. Um diese zu installieren muss folgender Befehl in die Konsole eingegeben werden:

```
sudo apt-get install python-smbus i2c-tools -y
```

Hiernach kann das folgende Python-Code Beispiel verwendet werden:

```
#
#!/usr/bin/python
# coding=utf-8

##########################################################################################################
### Copyright by Joy-IT
### Published under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
### Commercial use only after permission is requested and granted
###
### KY-053 Analog Digital Converter - Raspberry Pi Python Code Example
### ######################################################################################################



# Dieser Code nutzt die ADS1115 und die I2C Python Library fuer den Raspberry Pi
# Diese ist unter folgendem Link unter der BSD Lizenz veroeffentlicht
# [https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code]
from Adafruit_ADS1x15 import ADS1x15
from time import sleep

# Weitere benoetigte Module werden importiert und eingerichtet
import time, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Benutzte Variablen werden initialisiert
delayTime = 0.5 # in Sekunden

# Adresszuweisung ADS1x15 ADC

ADS1015 = 0x00  # 12-bit ADC
ADS1115 = 0x01  # 16-bit

# Verstaerkung (Gain) wird ausgewaehlt
gain = 4096  # +/- 4.096V
# gain = 2048  # +/- 2.048V
# gain = 1024  # +/- 1.024V
# gain = 512   # +/- 0.512V
# gain = 256   # +/- 0.256V

# Abtasterate des ADC (SampleRate) wird ausgewaehlt
# sps = 8    # 8 Samples pro Sekunde
# sps = 16   # 16 Samples pro Sekunde
# sps = 32   # 32 Samples pro Sekunde
```



```
sps = 64   # 64 Samples pro Sekunde
# sps = 128  # 128 Samples pro Sekunde
# sps = 250  # 250 Samples pro Sekunde
# sps = 475  # 475 Samples pro Sekunde
# sps = 860  # 860 Samples pro Sekunde

# ADC-Channel (1-4) wird ausgewaehlt
adc_channel_0 = 0    # Channel 0
adc_channel_1 = 1    # Channel 1
adc_channel_2 = 2    # Channel 2
adc_channel_3 = 3    # Channel 3

# Hier wird der ADC initialisiert - beim KY-053 verwendeten
# ADC handelt es sich um einen ADS1115 Chipsatz
adc = ADS1x15(ic=ADS1115)

Button_PIN = 24
GPIO.setup(Button_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)

##########################################################################################

# ########
# Hauptprogrammschleife
# ########
# Das Programm liest die aktuellen Werte der Eingang-Pins
# und gibt diese in der Konsole aus

try:
        while True:
                #Aktuelle Werte werden aufgenommen
                adc0 = adc.readADCSingleEnded(adc_channel_0, gain, sps)
                adc1 = adc.readADCSingleEnded(adc_channel_1, gain, sps)
                adc2 = adc.readADCSingleEnded(adc_channel_2, gain, sps)
                adc3 = adc.readADCSingleEnded(adc_channel_3, gain, sps)

                # Ausgabe auf die Konsole
                print "Messwert Channel 0:", adc0, "mV "
                print "Messwert Channel 1:", adc1, "mV "
                print "Messwert Channel 2:", adc2, "mV "
                print "Messwert Channel 3:", adc3, "mV "
                print "---------------------------------------"

                # Reset + Delay
                button_pressed = False
                time.sleep(delayTime)


except KeyboardInterrupt:
        GPIO.cleanup()
```

Anschlussbelegung Raspberry Pi:

VDD	=	3,3V	[Pin 01]
GND	=	Masse	[Pin 06]
SCL	=	GPIO03 / SCL	[Pin 05]
SDA	=	GPIO02 / SDA	[Pin 03]
ADDR	=	N.C.	[-]
ALRT	=	N.C.	[-]
A0	=	Messspitze Analog 0	[Zu messende Spannung | z.B. Sensorausgang]
A1	=	Messspitze Analog 1	[Zu messende Spannung | z.B. Sensorausgang]
A2	=	Messspitze Analog 2	[Zu messende Spannung | z.B. Sensorausgang]

A3 = Messspitze Analog 3 [Zu messende Spannung | z.B. Sensorausgang]

Beispielprogramm Download

KY-053_RPi_AnalogDigitalConverter.zip

Zu starten mit dem Befehl:

```
sudo python KY-053_RPi_AnalogDigitalConverter.py
```

Erweiterte Funktionen des ADS1115 ADC

Die Funktion des ADS1115, die in den oben aufgezeigten Codebeispielen zur Verwendung kommt, nennt sich "Single Ended Conversion" und besagt, dass eine Messung am einzelnen ausgewählten Kanal gegen Masse erfolgt.

Neben dieser Art von Messung besitzt der ADS1115 ADC auch z.B. die Funktion der differentiellen Messung, sodass eine Differenz-Spannung zwischen zwei Eingängen gemessen wird (Beispiel: Spannung zwischen A0 und A1). Zusätzlich zur Singe-Ended Messung lässt sich auch die Comparator Funktion aktivieren, welche erst ein Messergebnis liefert, wenn eine Spannungsschwelle überschritten wird.

Diese Funktionen und Funktionen, wie z.B. die Änderung der Abstastrate (Samplerate), sind in den Adafruit Libraries zur Konfiguration einprogrammiert - Näheres finden Sie in der Dokumentation der Adafruit Libraries.

