Karlsruher Institut für Technologie

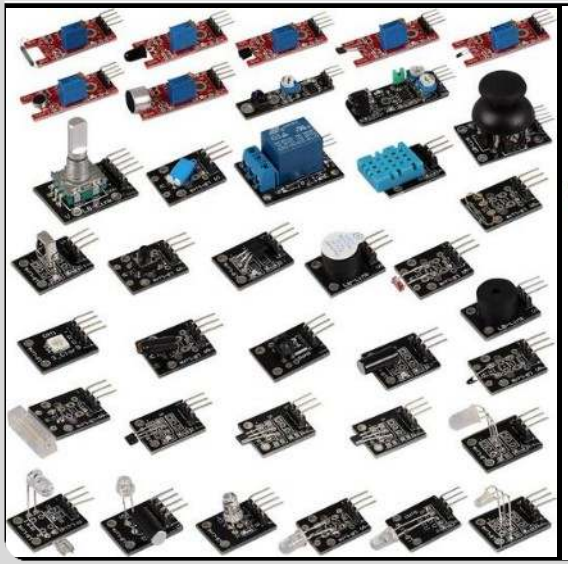
Digitale Messwerterfassung für den Physikunterricht

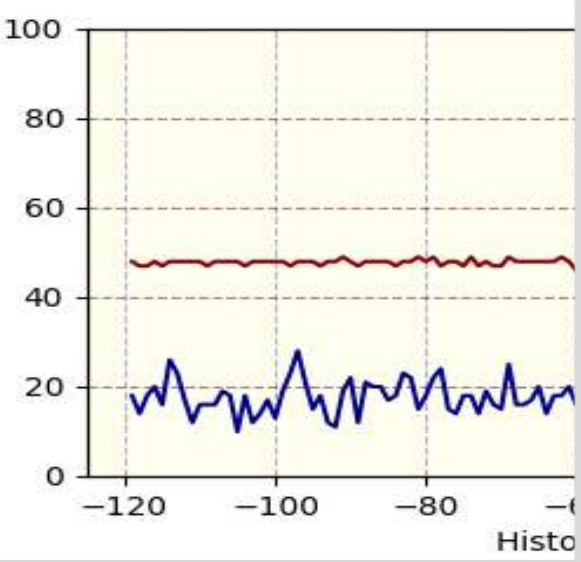
mit dem Raspberry Pi

Pforzheim

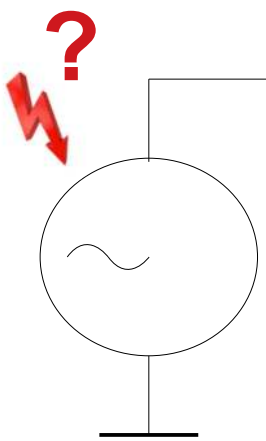
Fakultät für Physik
Institut für Experimentelle Teilchenphysik

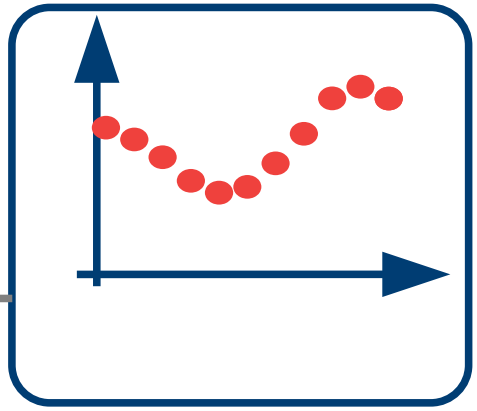
18. Februar 2020





Der Messprozess – vom Phänomen zur Erkenntnis





Sensor

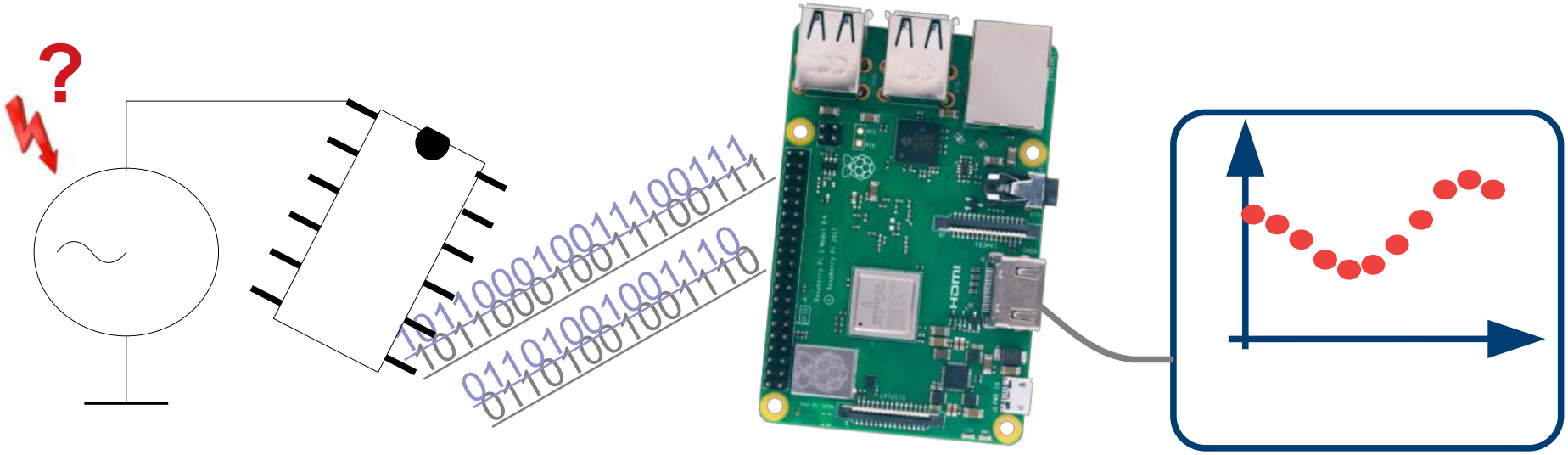
- **physikalische Größe am Eingang**

Visualisierung/ Auswertung

- **physikalische Interpretation**

Häufig gibt es eine „Black Box“ zwischen dem Phänomen und dem Messergebnis !

Der Digitale Messprozess – vom Phänomen zur Erkenntnis



Sensor

- **physikalische Größe am Eingang**
- Sonsorprinzip
- technische Umsetzung

DA-Wandlung

- Grundprinzip
- einfache Realisierungen

digitaler Datenstrom

- Binärkodierung
- Übertragungs-protokoll(e), Datenbus
- Fragen des Signaltransports auf der physikalischen Ebene

Prozessor

- Steuerung der Datenaufnahme
- digitale Filterung und Daten-verarbeitung
- Visualisierung und Auswertung
- Formatierung und Speicherung

Visualisierung/ Auswertung

- **physikalische Interpretation**

Der digitale Messprozess verknüpft das untersuchte physikalische Phänomen mit der Interpretation des Messergebnisses

Sensoren

Handel bietet eine Vielzahl an Sensoren zum Preis von 1 bis ca. 10 €, häufig als Sensor-Set:

Temperatur, Magnetfeld, Druck, Licht, Infrarot, Schall, Ultraschall, Vibration, Lage, Abstand, **...**

sowie diverse
Analog-Digital-Wandler

und Komponenten zur Steuerung und Regelung:

Digital-Analog-Wandler, Relais, Motorsteuerung, ...

PhyPiDAQ

Standardisierter Messkoffer

zur Einführung von Schülern in
die grundlegenden Konzepte der
Digitaltechnik und des digitalen Messens

Standardisierte Software-Umgebung für typische Messaufgaben:

- einheitliche Schnittstelle für eine Vielzahl von Sensoren, erweiterbar
- graphische Anzeigen
  (Voltmeter, Oszilloskop, xy-Anzeige, zeitlicher Verlauf von Messgrößen)
- Datenaufzeichnung (im CSV-Format)
- Kalibration von (nicht-linearen) Sensoren
- Anwendung von Formeln auf Messwerte

```
sensor = <DeviceClass>()
sensor.init()
sensor.acquireData(data)
  . . .
sensor.closeDevice()
```

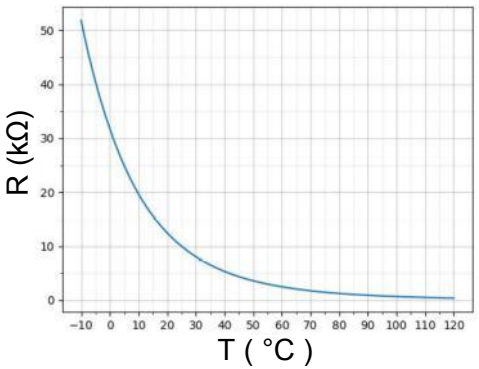
```
import numpy as np, time
from phypidaq.ADS1115Config import *

device = ADS1115Config()
device.init() # Initialisierung

dt = 1.    # Ausleseintervall in s
T0 = time.time() # Start-Zeit
dat = np.array([0.])

print(' beginne Auslese, <ctrl-C> = stop')

while True:
  device.acquireData(dat)
  dT = time.time() - T0
  print('%.2g, %.4g' %(dT, dat) )
  time.sleep(dt)
```

Code in python

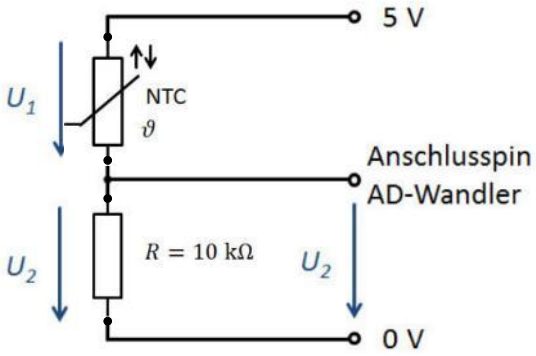
Beispiel: **Thermometer**

geführte Aufgabenstellung: Bau eines Thermometers mit nichtlinearem Sensor (NTC-Widerstand)

nicht-lineare Sensorkennlinie R(T)



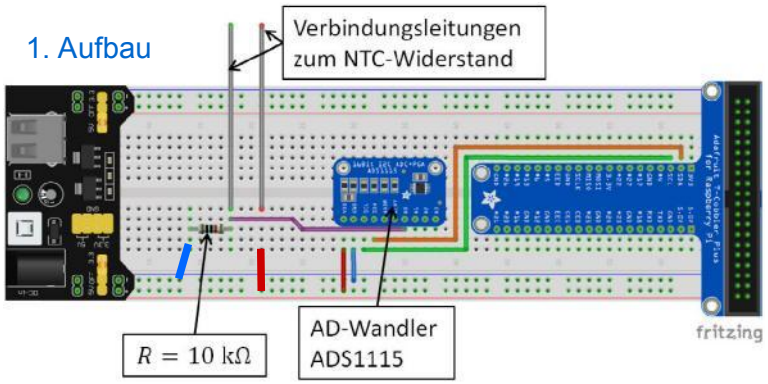
Schaltplan



- Sensor mit reproduzierbarem (nicht notwendig linearem) Zusammenhang zwischen Effekt und Messwert als Voraussetzung
- Durch Austausch des Sensors leicht in anderem physikalischen Kontext einsetzbar: Messung von Abstand, Kraft, Lichtstrom, elektr. Feldstärke, Magnetfeld, ...
- Beispiel erklärt allgemein den Weg vom Sensor-Wert zur physikalischen Größe und gibt so einen Einblick in die Black Box ; liefert Erklärungsmodell verschiedene Arten von digitalen Messprozessen

Mit Schülern der 11. und 12. Klasse in Zweiergruppen im Umfeld eines Schülerlabors in ca. 4h durchgeführt

1. Aufbau



2. Umrechnung ADC-Ausgabe in Spannung

Stufe	⇒	digitalisierte Spannung in V
0	⇒	0
32767	⇒	6,114
1	⇒	
518	⇒	
16383	⇒	

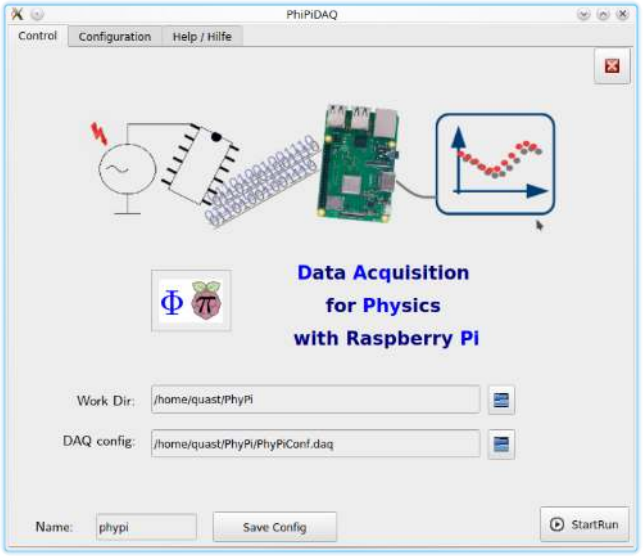
3. Kalibrationstabelle

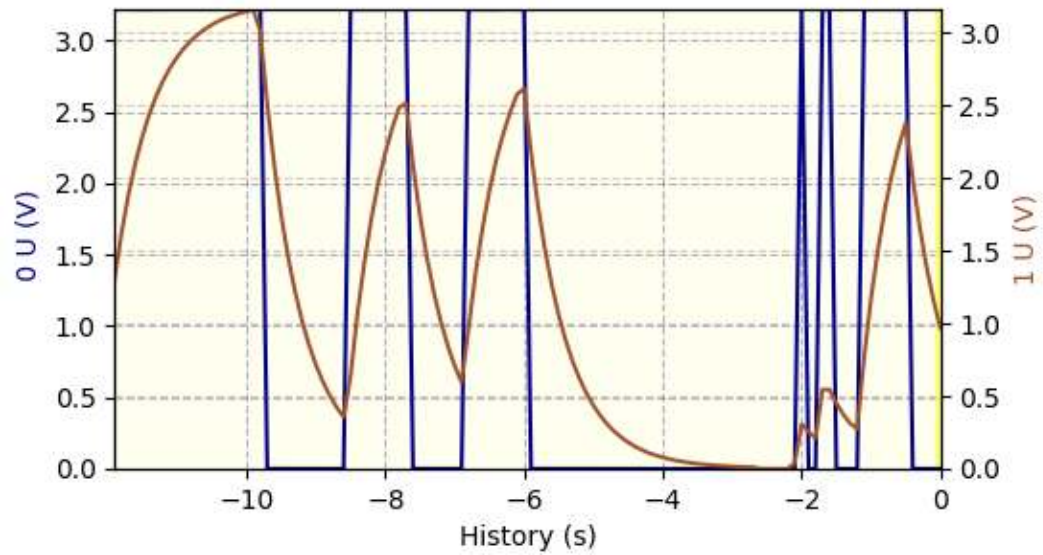
	1.	2.	3.	4.
Temperatur *T* in °C				
Spannung *U* in V				

4. Integration von AD-Wandlung und Kalibration in Software

```
while True: # Dauerschleife.
    adWert = adWandler.read_adc(0,2/3) * aufloesung # Wert Ad-Wandler.
    temperatur = kalibFunkt(adWert) # Wir berechnen aus der digitalisierten
Spannung des AD-Wandlers die Temperatur.
    temperatur = round(float(temperatur),1) # Wir runden den Temperaturwert
# auf eine Nachkommastelle.
    print("Temperatur:") # Wir geben den Text "Temperatur" aus.
    print(temperatur) # Wir geben den Wert der Temperatur aus.
    print("°C") # Wir geben die Einheit °C aus.
```

PhyPiDAQ: grafische Oberfläche





Wachsende Anzahl an vorbereiteten Konfigurationen:

- Temperaturmessung mit analogen (NTC) und digitalen Sensoren
- digitale Sensoren zur Messung von Strom, Spannung, Beschleunigung, Druck, ...
- Frequenzmessungen auf GPIO-Pins
- Analog-Digitalwandler ADS1115 (4Kanal, 16 bit) und MCP3208 (8Kanal, 12 bit)
- USB-Oszilloskop als Datenlogger und zur Registrierung von kurzen Einzelsignalen
- Kraftmessung mit Wägezelle und Dehnungsmessstreifen
- Abstandssensor (Lichtlaufzeit 0 – 4000 mm)

Schülerprojekte: Charakterisierung von GPIO-Pins, Hell-Dunkel-Schaltung, Lade- u. Entladekurven eines Kondensators, Kalibration eines Temperatursensors

Software unter openSource-Lizenz verfügbar:
s. https://github.com/GuenterQuast/PhyPiDAQ

Aufnahme analoger Signale

Präzise Messung von unipolaren Signalen

schnell Signale >10ns

Analog-Digital-Wandler

Betriebsspannung 2,0 V – 5,5 V

4 Kanäle oder 2 Kanäle differentiell

16 Bit Auflösung, programmierbarer Vorverstärker x1 – x16 (1 mv – 5 V)

interne Spannungsreferenz

I^2C – Bus

bis 860 Hz Ausleserate

USB-Oszilloskop

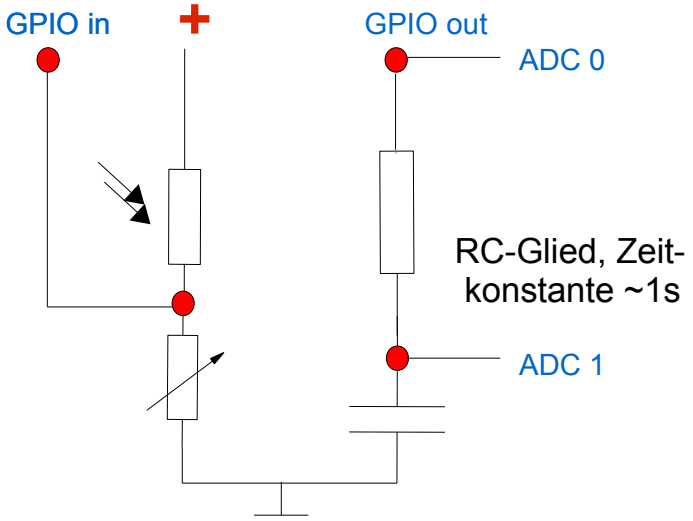
z.b. PicoScope 2204A (ab 95,-€)
- 2 Kanäle, 10 MHZ Bandbreite, 100 MS/s, 50 mV – 20 V, 8 Bit Auflösung

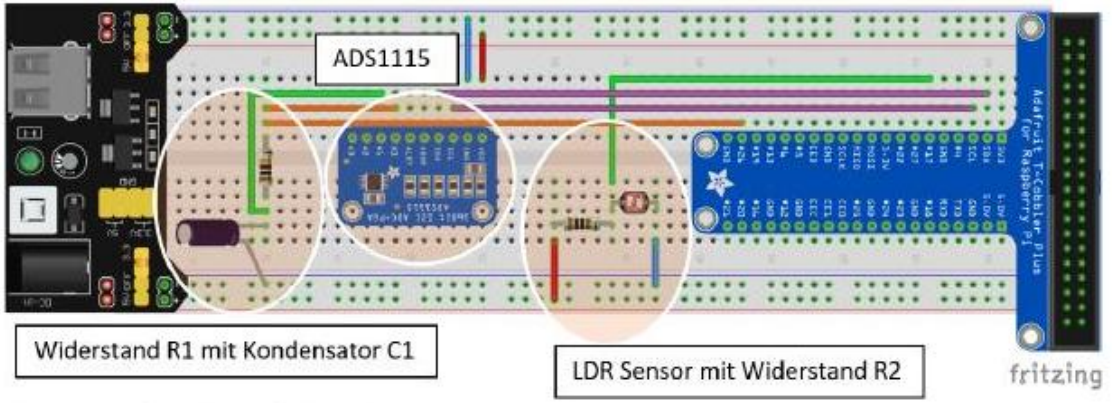
PicoScope 2204B (ab 299,- €)
- 2 Kanäle, 50 MHZ Bandbreite, 500 MS/s , 20 mV – 20 V 8 Bit Auflösung

Lade- und Entladekurven am Kondensator

Rechteckspannung an GPIO-Pin durch
Abdecken einer Fotodiode an GPIO-Pin

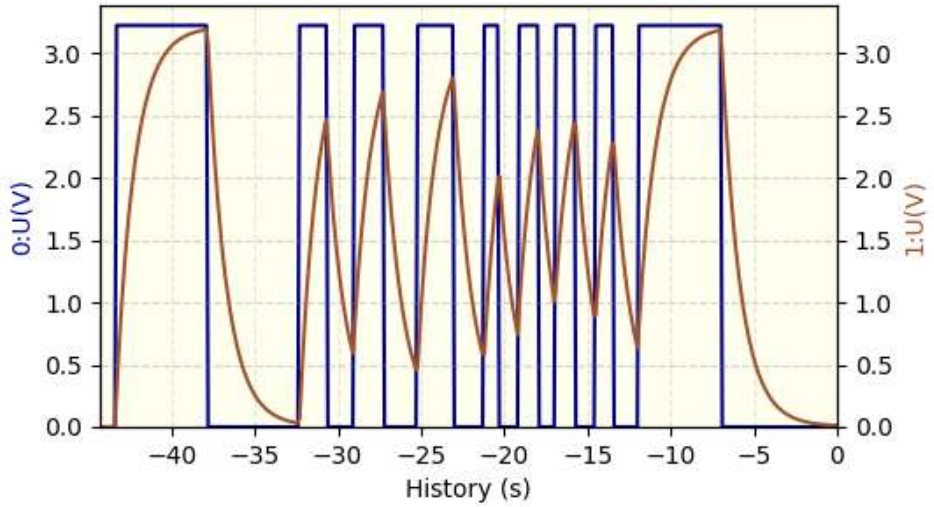
(analog Hell-Dunkelschaltung mit Kondensator statt LED; Anzeige der Spannungsverläufe mit ADS1115 und DataLogger)



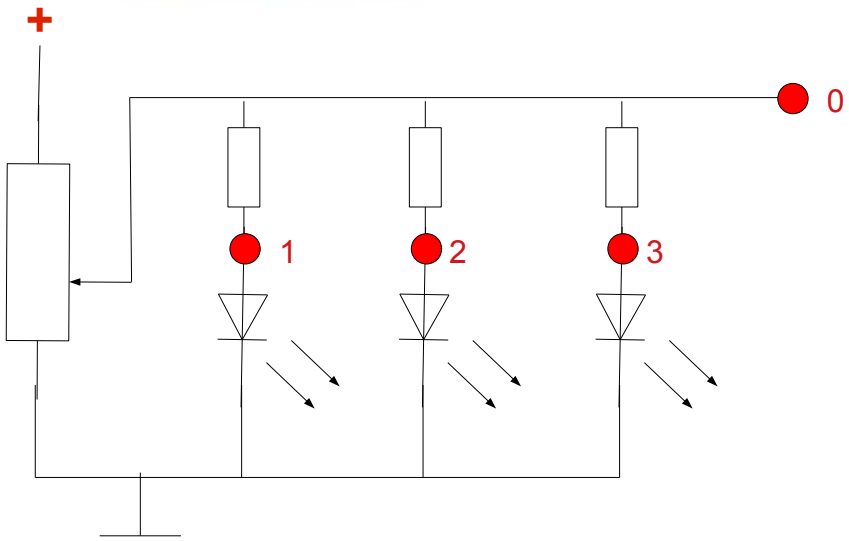


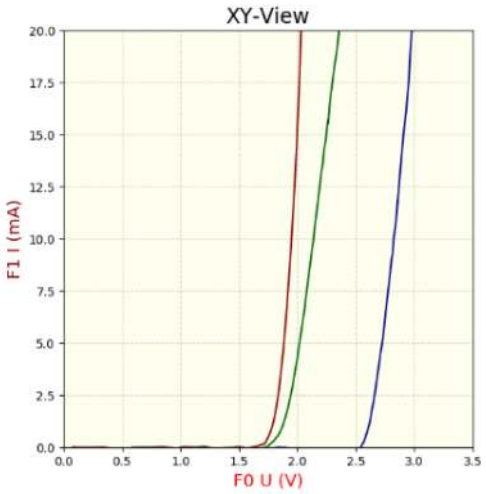
Bauteilwerte: R1=100kOhm, C1=10µF, R2=1kOhm

Ausgabe
DataLogger



Mehrere Kennlinien

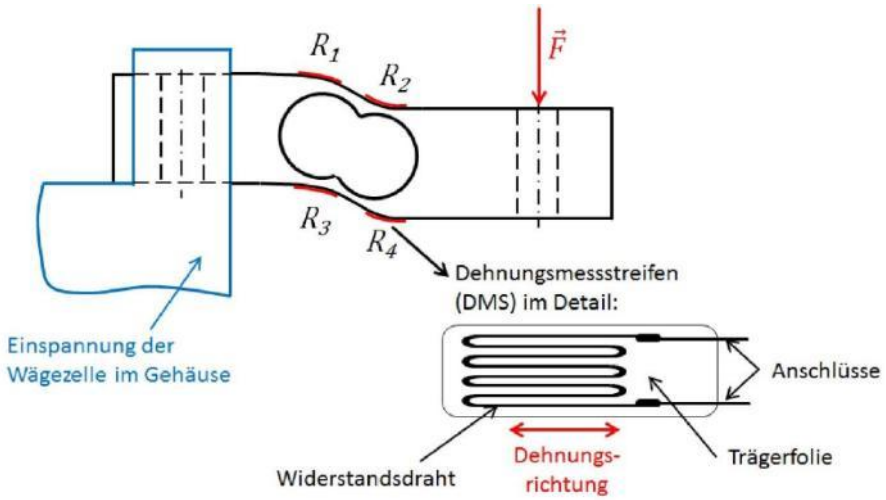




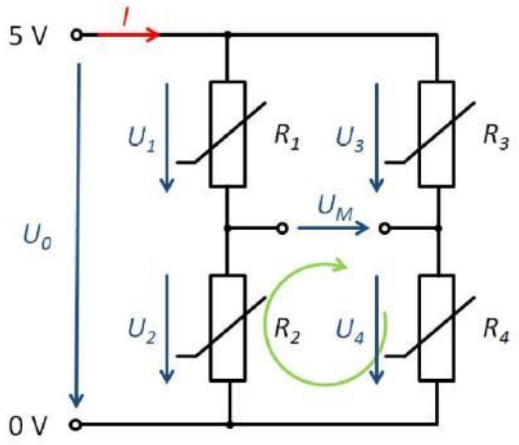

```
# Konfiguration fuer PhyPiDAQ

DeviceFile: ADS1115_Diode.yaml      # 4 aktive Kanaele mit Verst. 1

# Anwenden von Formeln auf die gemessenen Spannungen
# c0 ist Betriebsspannung, c1 - c3 Diodenspannungen
ChanFormula:
 - c1                    #  U Diode c1
 - (c0 - c1) / 0.120     #  I Diode c1
 - c2                    #  U Diode c2
 - (c0 - c2) / 0.100     #  I Diode c2
 - c3                    #  U Diode c3
 - (c0 - c3) / 0.082     #  I Diode c3

ChanLabels: [U, I, U, I, U, I]          # Labes fur Kanaele
ChanUnits: [V, mA, V, mA, V, mA]    # Einheiten
ChanColors: [red, darkred, green, darkgreen, blue, darkblue]

# Wertebereiche
ChanLimits:
 - [0., 3.5]   # U D1
 - [0., 10.]   # I D1
 - [0., 3.5]   # U D2
 - [0., 10.]   # I D2
 - [0., 3.5]   # U D3
 - [0., 10.]   # I D3

DisplayModule: DataLogger
Chan2Axes: [0,1,0,1,0,1]
XYmode:   true              # enable/disable XY-display
xyPlots:        # channels to display as x-y graph
 - [0,1]
 - [2,3]
 - [4,5]

Interval: 0.1               # Datennahme-Intervall
```

Aufbau eines Kraftsensors

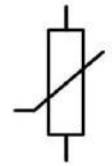
Wägezelle mit Dehnungsmesssteifen



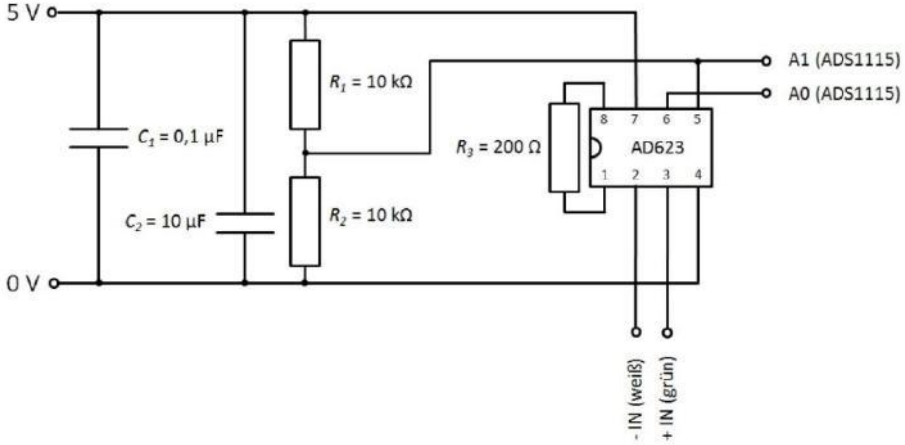
Schaltung der Dehnungsmessstreifen

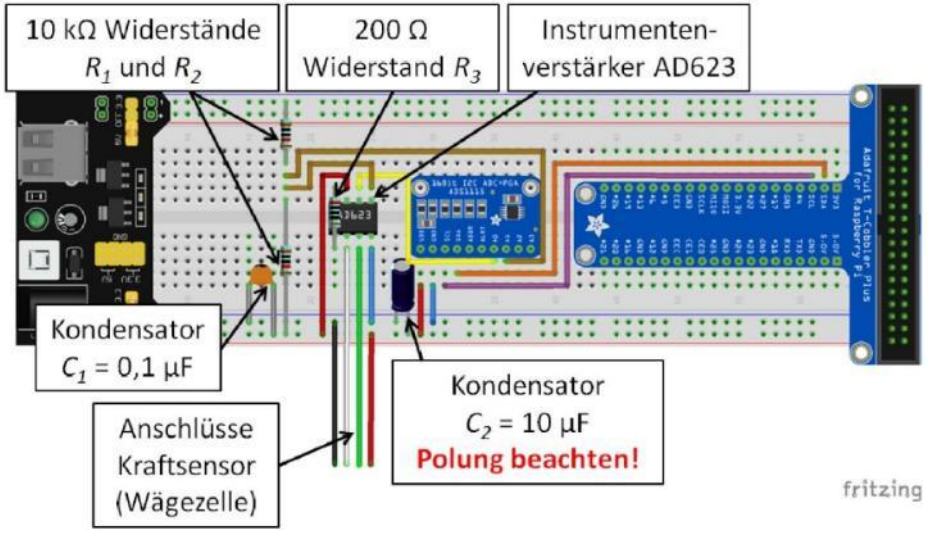


Schaltzeichen
für einen DMS



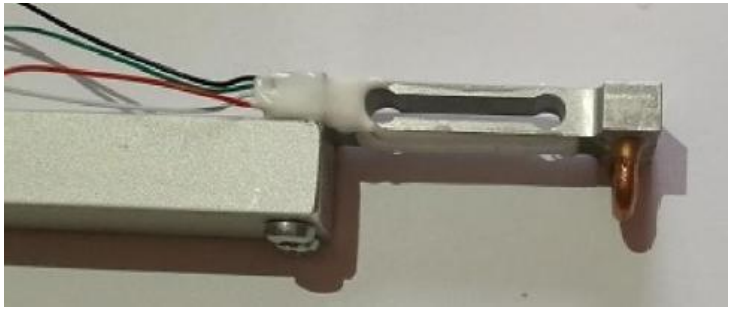
Aufbau mit Instrumentenverstärker



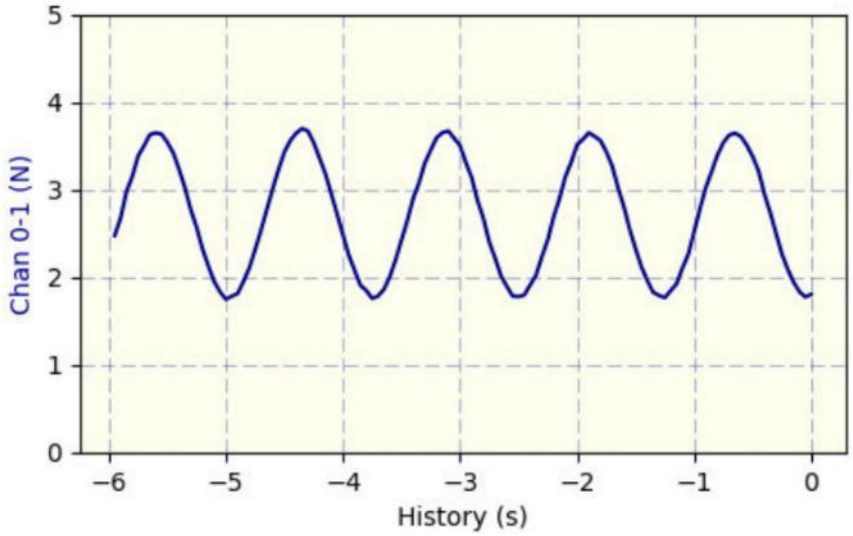


Aufbau des Kraftsensors (2)

Der Selbst-Bau Kraftsensor (Kosten ca. 15,- €)



Federpendel am Kraftsensor (Auslese mit PhyPiDAQ)



Geophon am PicoScope

Mit einem Oszilloskop lassen sich Effektivwerte von Wechselsignalen bestimmen – z.B. Lautstärke-Messung

oder – hier - ein „Erdbebensensor“ (Sm-24)

- 200 Werte in 20 ms aufnehmen
- quadratischen Mittelwert bilden
- Effektivwerte mit PhyPiDAq darstellen

Die Klasse PSConfig
erledigt Auslese und Mittelwertbildung

```
# Messung von Effektiv-Werten mit PS2000(A)
DAQModule: PSConfig

PSmodel: '2000'      # PS model 220xA
## PSmodel: '2000a'     # PS model 2x0xB

# channel configuration
picoChannels: [A]
ChanRanges: [0.05]
ChanModes: [AC]
## ChanOffsets: [-1.95, -1.95] # !!! not for A
series

sampleTime: 2.0E-02
Nsamples: 200

# trigger
trgActive: false  # true to activate
trgChan: A
trgThr: 0.
trgTyp: Rising
trgTO: 4  # set short time-out for A series
       # values<4 lead to readout instabilities

## pretrig: 0.05 # !!! not for A series

# signal generator
# frqSG: 100.E+3 # put 0. do disable
frqSG: 0.

# special flags for PhyPiDAQ
ChanAverages: ['rms'] # 'mean'
```

Gamma-Detektor GDK101

Detektor: 10 PiN-Diden mit 1cm² Fläche

Betrieb: 5V, interne Spannungsstabilisierung zur Versorgung der PiN-Dioden

Ausgänge: analog und I²C-Bus

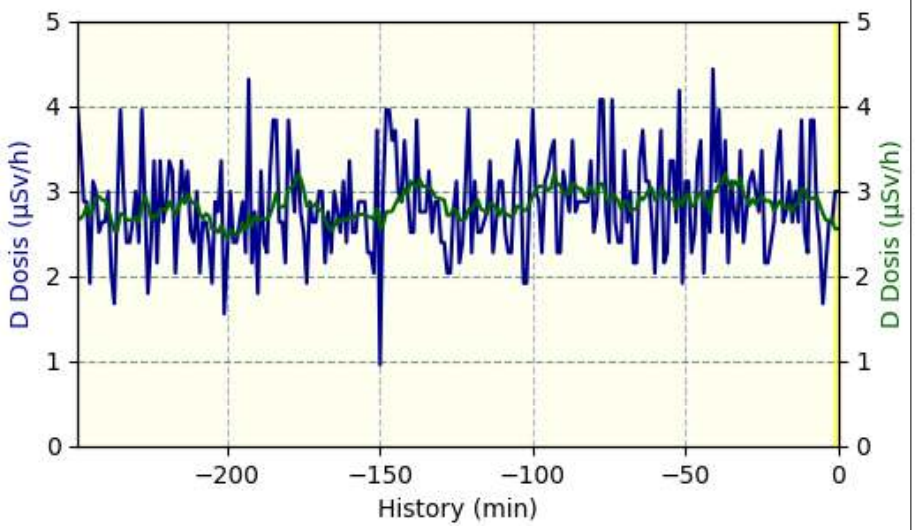
(5V, benötigt Level-Shifter)

Messbereich: 0.01 – 200 µSv/h

(1 – 1000 cnt/min)

Messmöglichkeiten:

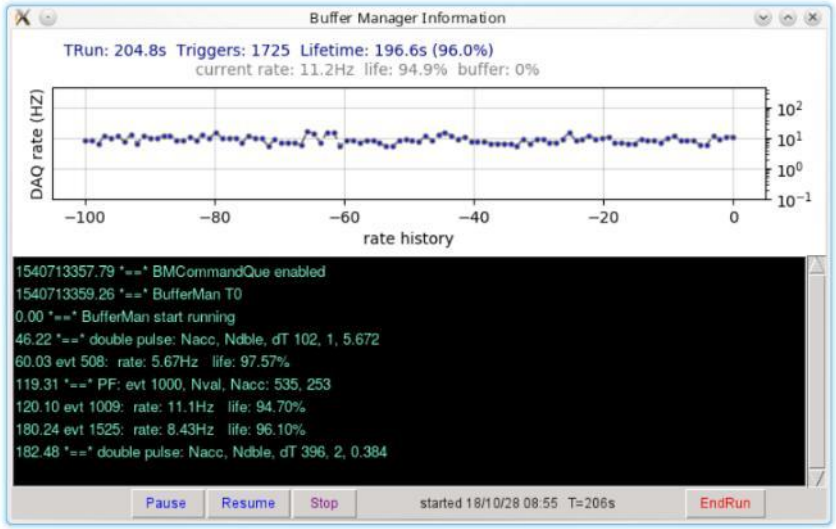
- Umgebungsradioaktivität, Rosendünger, Zigarettenasche, Uranglas, …
- Radon-Zerfallsprodukte (gesammelt z.B. auf Luftballon) und Abklingrate
- Poisson-Statistik bei radioaktiven Zerfällen



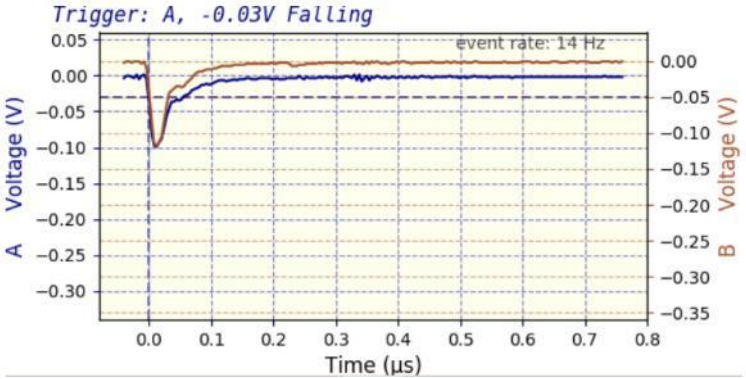
Messung mit Uranglas auf GDK101

Detektor für Kosmische Strahlung mit PicoScope

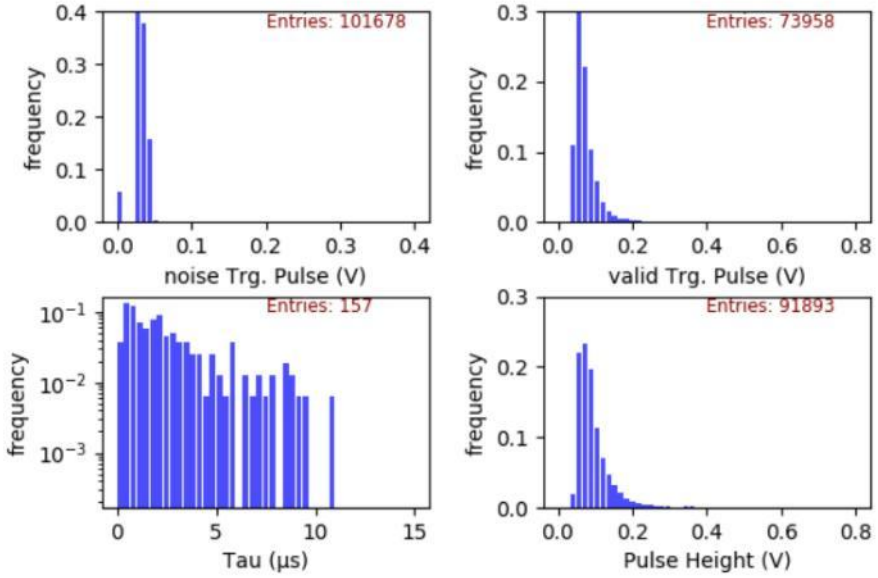
Verlauf der Datennahme



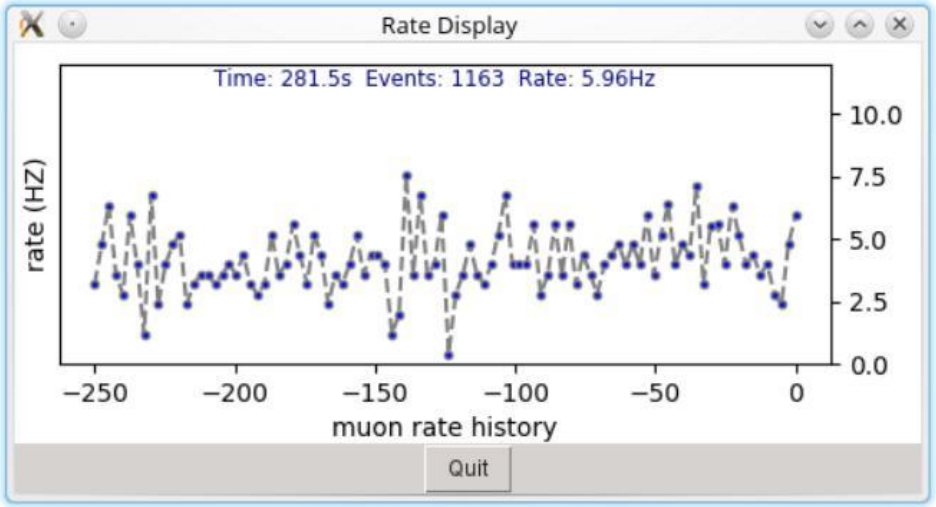
Oszilloskop-Anzeige



Häufigkeitsverteilungen: Pulshöhen u. Myon-Lebensdauern

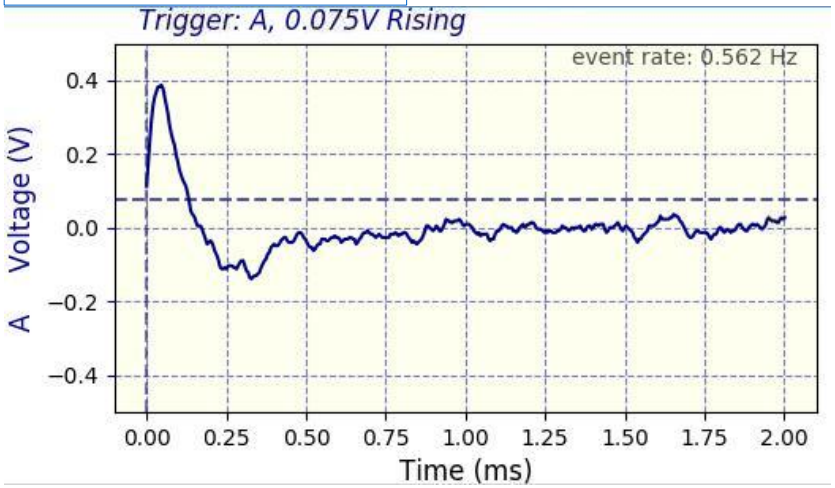


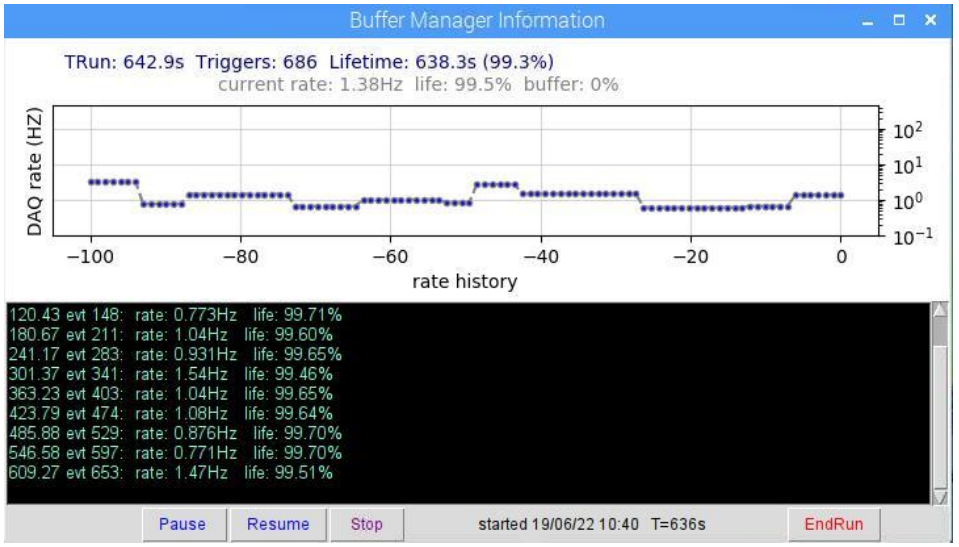
Rate der akzeptierten Myonen (Zweierkoinzidenz)



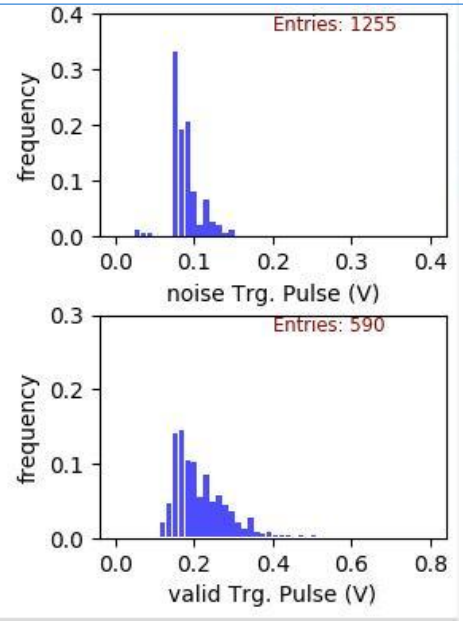
Detektor für Gamma-Strahlung mit PicoScope

Oszilloskop-Anzeige

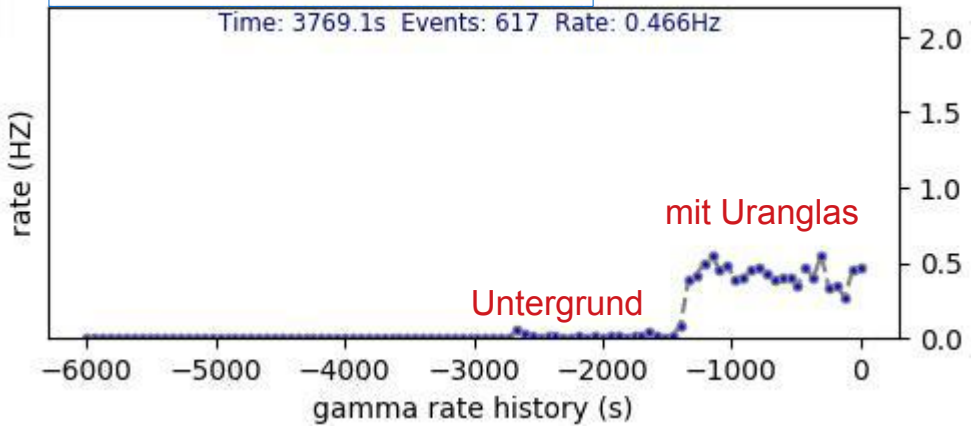




Häufigkeitsverteilungen der Pulshöhen



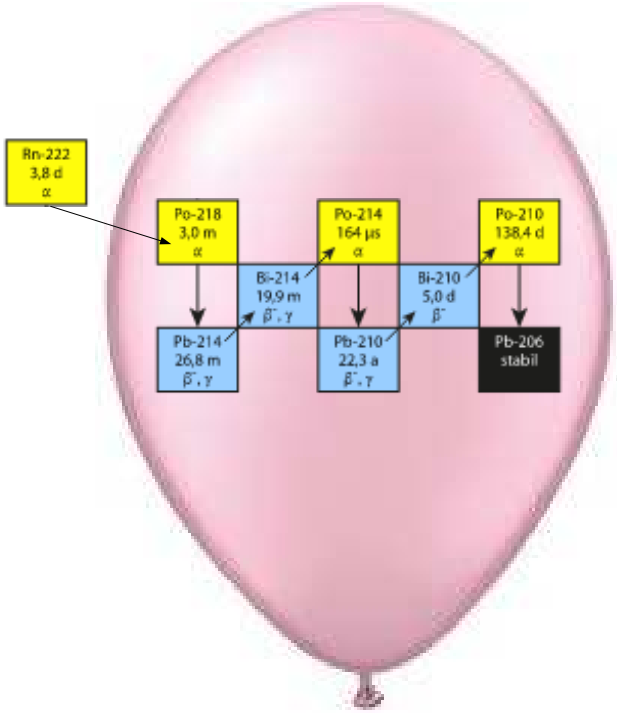
Rate der akzeptierten Pulse



Umwelt-Radioaktivität mit Luftballon und GDK101

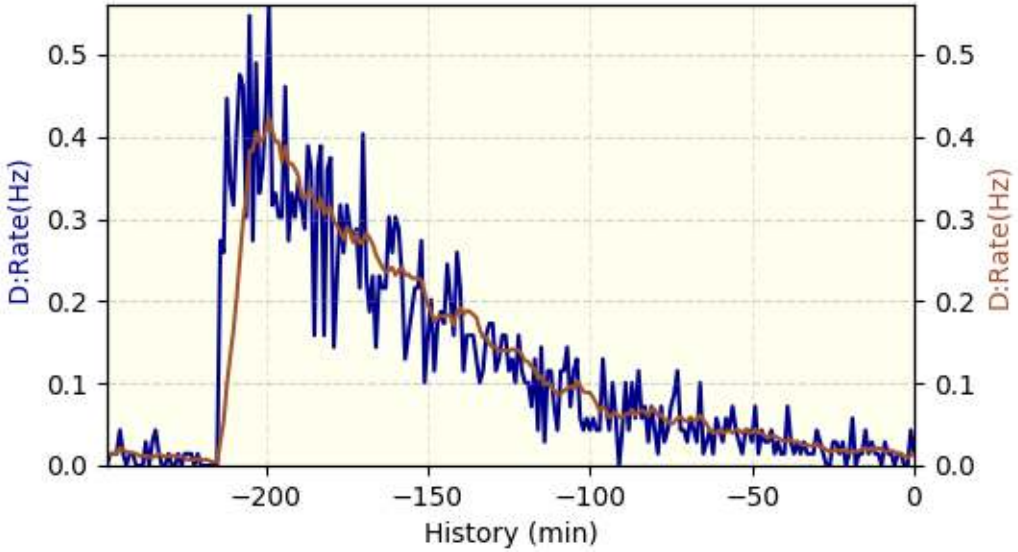
siehe Leifi Physik - Versuch zur Umweltradiaoktivität

1.) Anziehung von ^{222}Rn-Zerfallsprodukten durch elektrostatisch aufgeladenen Luftballon



2.) nach ca. 30 min Luft entweichen lassen und Ballon auf den Sensor legen

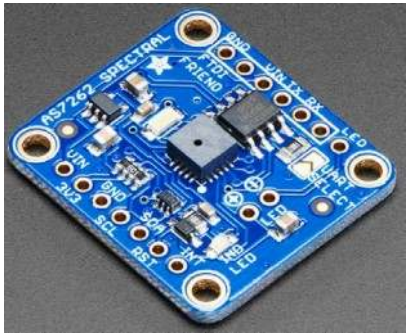
Messung der γ-Rate mit 1 min. Torzeit



Überlagerung der Zerfälle verschiedener Nuklide, daher kein reines Exponentialgesetz !

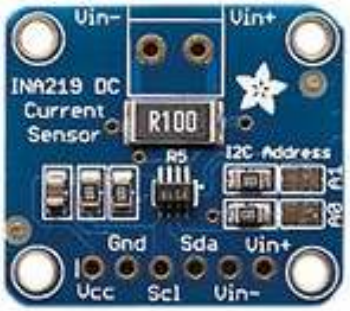
(Eine Auswahl an) Präzisionssensoren

VL53L1X
Abstandssensor
Prinzip Lichtlaufzeit



AS7262
Spektralsensor mit 6 Kanälen, je 40 nm breit

GDK101
Gamma-Detektor
10 PiN-Dioden,
1cm² Fläche



INA219
Strom- & Spannungs-Sensor, 3.2 A, 26 V

MMA7451
Beschleunigungs-Sensor

SM-24
“Geophon“,
Erdbeben- und Erschütterungs-Sensor

bieten viele preiswerte Messmöglichkeiten im Physikunterricht

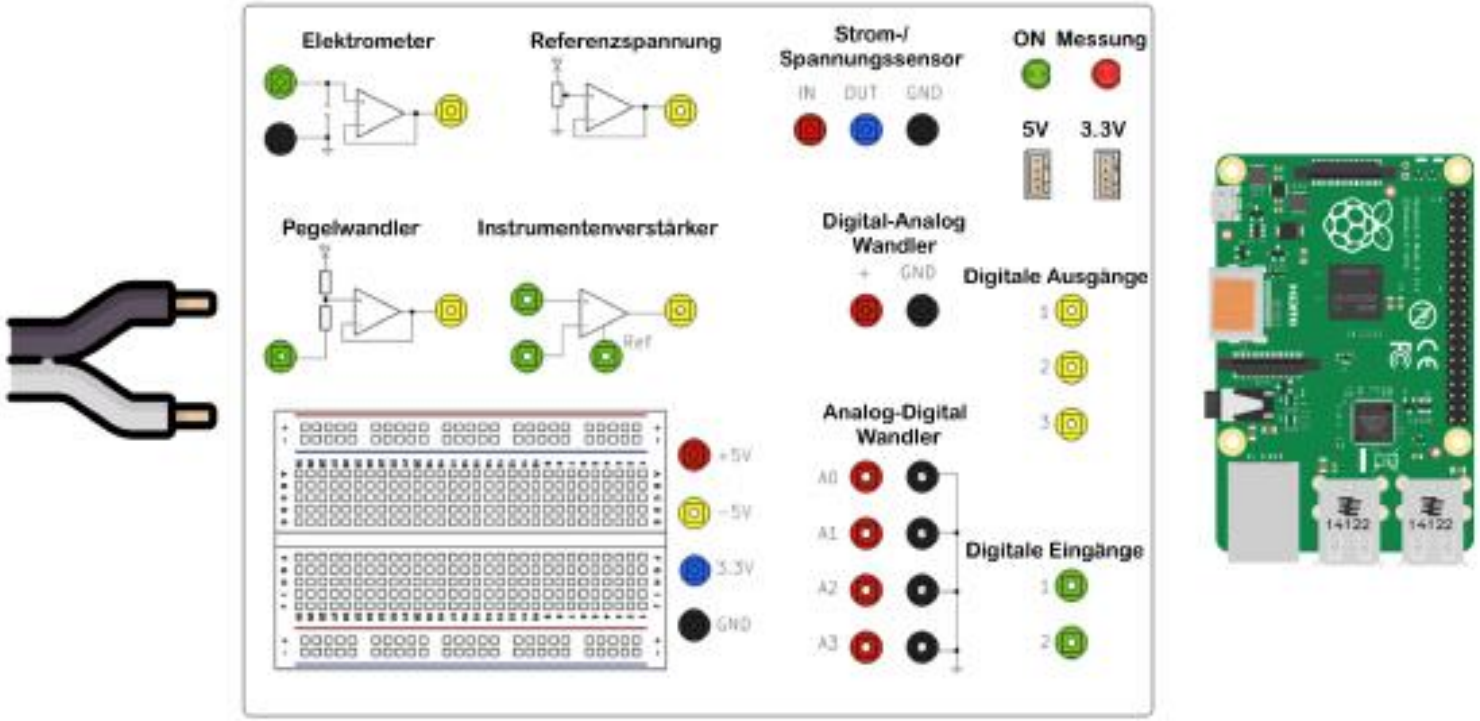
Analoge Signale

Viele physikalische Aufbauten liefern analoge Signale

Einige Beispiele:

- Ladungsmessung am Kondensator
- Photostrom
- Hall-Spannung
- Kennlinien elektrischer Bauteile
- ...

→ **benötigen geeignete Vorverstärker und Pegelanpassungen**

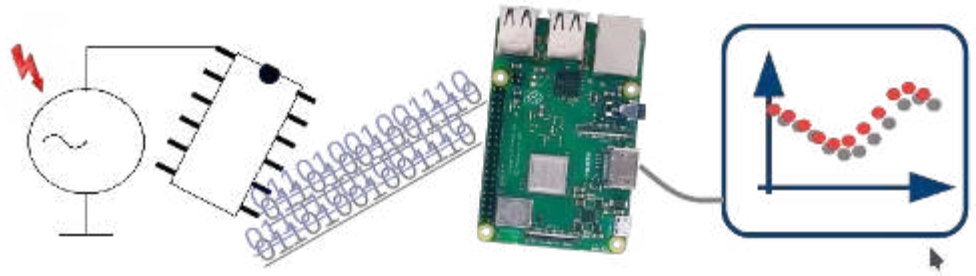


Analogplatine, Bachelor-Arbeit Dominik Braig, KIT Jan. 2020

Links

- Masterarbeit von Moritz Aupperle:
  Konzeption und Gestaltung eines digitalen Messwertefassungsystem für den Physikunterricht
  http://ekp-invenio.physik.uni-karlsruhe.de/record/49063

- Digitales Messen im Physikunterricht mit Raspberry Pi
  https://github.com/GuenterQuast/PhyPiDAQ



Weitere Projekte der Gruppe:

- Auslese von PicoScope USB-Oszilloskopen
  https://github.com/GuenterQuast/picoDAQ

- Auslese von Detektoren für Kosmische Strahlung
  https://github.com/GuenterQuast/picoCosmo

- Netzwerk Teilchenwelt
  http://www.teilchenwelt.de

NETZWERK TEILCHENWELT

Anhang 1: Installation

Installation des Betriebssystems auf dem Raspberry Pi:

einfachste Möglichkeit „NOOBS“:
https://www.raspberrypi.org/downloads/noobs/

Installation von PhyPiDAQ:

Raspberry Pi muss mit dem Internet verbunden sein !

- Anmelden als Nutzer „pi“
- Herunterladen des Pakets PhyPiDAQ

  ```
  mkdir ~/git
  cd ~/git
  git clone https://github.com/GuenterQuast/PhyPiDAQ
  ```

- Installation aller Abhängigkeiten

  ```
  cd ~/PhyPiDAQ
  git pull
  ./installlibs.sh
  ```

  (die mittlere Zeile dient dazu, eine bereits vorhandene Installation zu aktualisieren)

- Aufsetzen eines Arbeitsverzeichnisses

  ```
  ./install_user.sh  <Verzeichnis>
  ```

  wenn kein Verzeichnis angegeben wird, ist die Vorgabe `/home/pi/PhyPi`

Anhang 2: Die Konfigurationsdatei

```
# -- Konfigurations-Optionen fuer PhyPiDAQ
DeviceFile: config/ADS1115Config.yaml    # 16 bit ADC, I2C bus

# -- Konfigurationsoptionen fuer Kanaele
##  Meta-Daten fuer jeden Kanal
ChanLabels: [U, U]              # Namen
ChanUnits: [V, V]               # Einheiten
ChanColors: [darkblue, sienna] # Farbzuordnung in der Anzeige

## ggf. werden hier die Informationen aus der Geraete-Konfiguration ueberschrieben
#ChanLimits:
# - [0., 1.]   # chan 0
# - [0., 1.]   # chan 1
# - [0., 1.]   # chan 2

## ggf. Kalibration der Rohmessungen
#ChanCalib:
#  - null    oder  - <Faktor> or  - [ [ <wahre Werte> ], [ <Rohwerte> ] ]
#  - 1.                      # chan0: ein einfacher Faktor fuer Kanal 0
#  - [ [0.,1.], [0., 1.] ]    # chan1: Interpolation [wahr]([roh] )
#  - null                     # chan2: Keine Kalibration

## Formel auf Werte anwenden
#ChanFormula:
#  - c0 + c1     # chan0 = Summe von Kanal 0 und 1
#  - c1          # chan1 : = Kanal 1 (Keine Aenderung)
#  - null        # chan2 : Keine Formel
```

Zeichen nach ‚#‘ werden ignoriert;
‚#‘ dient auch zum Auskommentieren von Befehlen

Anhang 2: Die Konfigurationsdatei (2)

```
# -- Konfiguation der grafischen Anzeige
#
Interval: 0.1                # Datennahme-Intervall in Sekunden
#NHistoryPoints: 120         # Anzahl Datenpunkte im Verlaufspuffer (Vorgabe 120)
DisplayModule: DataLogger    # zeitlicher Verlauf der Messgroessen
# DisplayModule: DataGraphs  # text, Balkendiagramm, zeitlicher Verlauf und xy-Darstellung
#Title: Demo                 # Titel fuer die Anzeige
## falls mehr als zwei Kanaele aktiv sind:
Chan2Axes: [0, 1, 0]         # Kanal auf linker(0) oder rechter(1) Achse
                             #     Voreinstellung [0, 1, 1, ...]
XYmode:     false            # XY-Darstellung ein/aus
## falls mehr als zwei Kanaele aktiv sind:
#xyPlots:      # Paare von Kanaelen als xy-Grafik
# - [0, 1]     # x: Kanal 0, y: Kanal 1
# - [0, 2]     # x: Kanal 0, y: Kanal 2 (falls vorhanden)
# - [2, 3]     # Voreinstellung [0,1], [0,2], ..., [0, n-1] bei n aktiven Kanaelen

#
# -- start in running or paused mode
# startActive: true  # start in running mode
```

Anhang 2: Die Konfigurationsdatei (3)

```
# -- Konfiguration fuer Ausgabe in Dateien
#
# Name der Ausgabedatei im CSV-Format
#DataFile:   testfile.csv     # Dateiname
DataFile:   null             #   null falls keine Ausgebe gewuenscht
#CSVseparator: ';'            # Feld-Trenner auf ';' setzen, Vorgabe ist ','

# Speicherung der letzen NHistoryPoints Datenpunke
#bufferData: PhyPiData    # Dateiname für (optionale) Speicherung
#bufferData: null         #  null zum Ausschalten; Voreinstellung: Datei PhyPiData.dat

# Ausgabe in Linux fifo (pipe) zum Senden von Daten an andere Prozesse
DAQfifo: null
#DAQfifo: PhyPiDAQ.fifo
```

Anhang 2: Die Konfigurationsdatei (4)

Zur Zeit unterstützte Sensoren:

```
DeviceFile: config/ADS1115Config.yaml     # 16 bit ADC, I2C bus
#DeviceFile: config/MCP3008Config.yaml    # 10 bit ADC, SPI-Bus
#DeviceFile: config/MCP3208Config.yaml    # 12 bit ADC, SPI-Bus
#DeviceFile: config/groveADCConfig.yaml   # 12 bit ADC auf Grove RPI Shield
#DeviceFile: config/PSConfig.yaml         # PicoTechnology USB-Oszilloskop
#DeviceFile: config/MAX31865Config.yaml   # pt100 Temperatursensor
#DeviceFile: config/GPIOCount.yaml        # Frequenzzaehler
#DeviceFile: config/DS18B20Config.yaml    # digitaler Temperatursensor
#DeviceFile: config/MAX31855Config.yaml   # Thermoelement
#DeviceFile: config/BMP180Config.yaml     # Druck-/Temperatursensor
#DeviceFile: config/INA219Config.yaml     # Strom-/Spannungssensor
#DeviceFile: config/MMA8451Config.yaml    # Beschleunigungssensor
#DeviceFile: config/VL53LxConfig.yaml     # Abstandssensor
## Beispiel für die Verwendung mehrerer Sensoren:
#DeviceFile: [config/ADS1115Config.yaml, config/GPIOCount.yaml]
## Demo options:
#DeviceFile: ToyDataConfig.yaml           # simulierte Daten
#DeviceFile: config/ReplayConfig.yaml     # Daten aus Datei
```

Anhang 3: Gerätekonfiguration

Beispiel einer (komplexen) Gerätekonfiguration für 16-Bit ADC mit Vorverstärkung

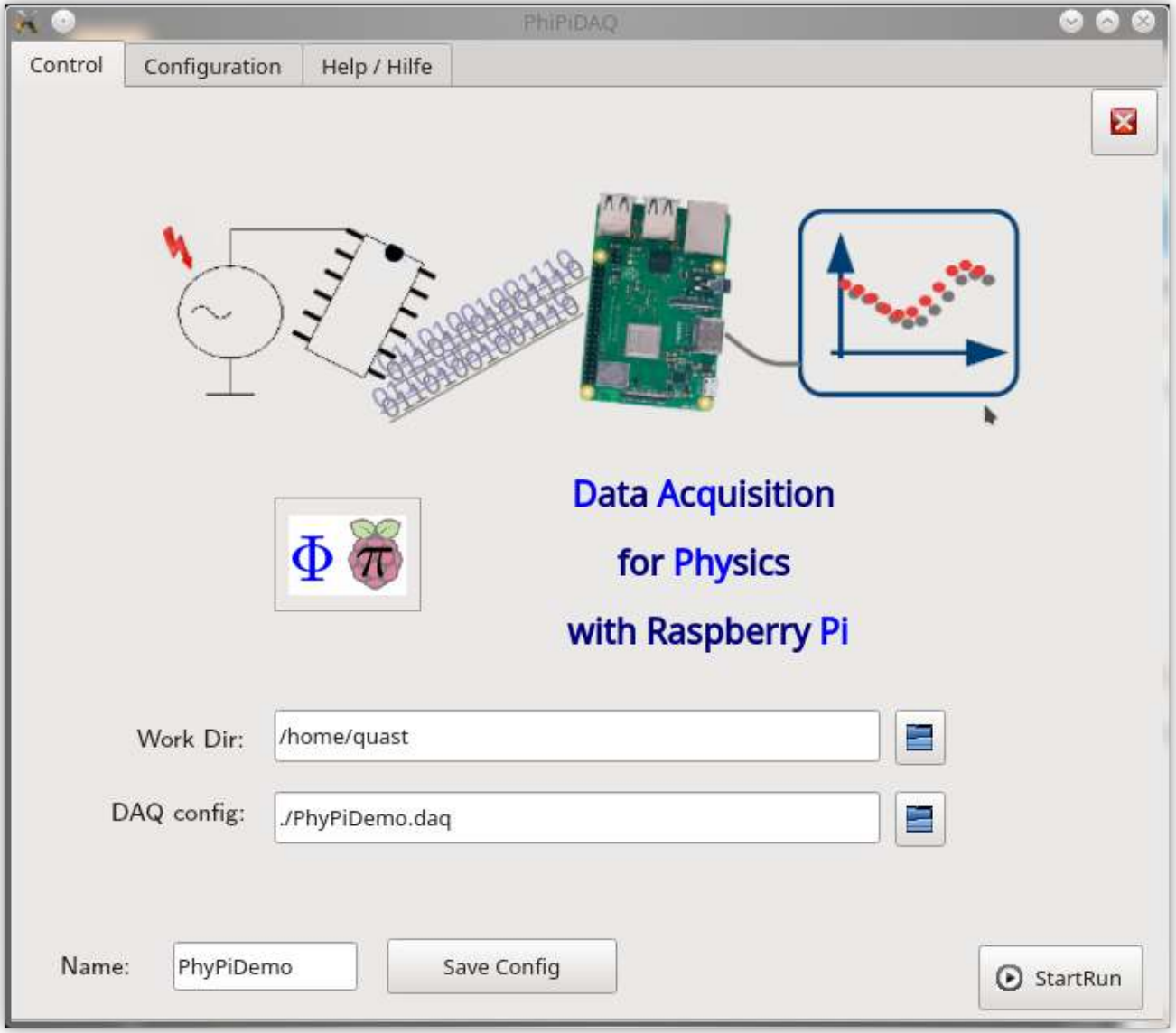
```
# Beispiel einer Konfiguration fuer den Analog-Digital-Wandler ADS1115
DAQModule: ADS1115Config    # relevantes phypidaq-Modul
ADCChannels: [0, 3]         # aktive ADC-Kanaele
                    # moegliche Werte: 0, 1, 2, 3
                    #  in differentiellem Modus:
                       #    -  0 = ADCChannel 0
                       #           minus ADCChannel 1
                       #    -  1 = ADCChannel 0
                       #           minus ADCChannel 3
                       #    -  2 = ADCChannel 1
                       #           minus ADCChannel 3
                       #    -  3 = ADCChannel 2
                       #           minus ADCChannel 3
DifModeChan: [true, true] # differentiellen Modus einschalten
Gain: [2/3, 2/3]          # programmierbarer Verstaerkungsfaktor
                     #   moegliche Werte:
                     #     - 2/3 = +/-6.144V
                     #     -   1 = +/-4.096V
                     #     -   2 = +/-2.048V
                     #     -   4 = +/-1.024V
                     #     -   8 = +/-0.512V
                     #     -  16 = +/-0.256V
sampleRate: 860           # programmierbare Datenrate des ADS1115
                     #    moegliche Werte:
                     #    8, 16, 32, 64, 128, 250, 475, 860
```

Anhang 4: Grafische Oberfläche

Grafische Oberfläche zur Verwaltung und Erstellung der Konfigurationen

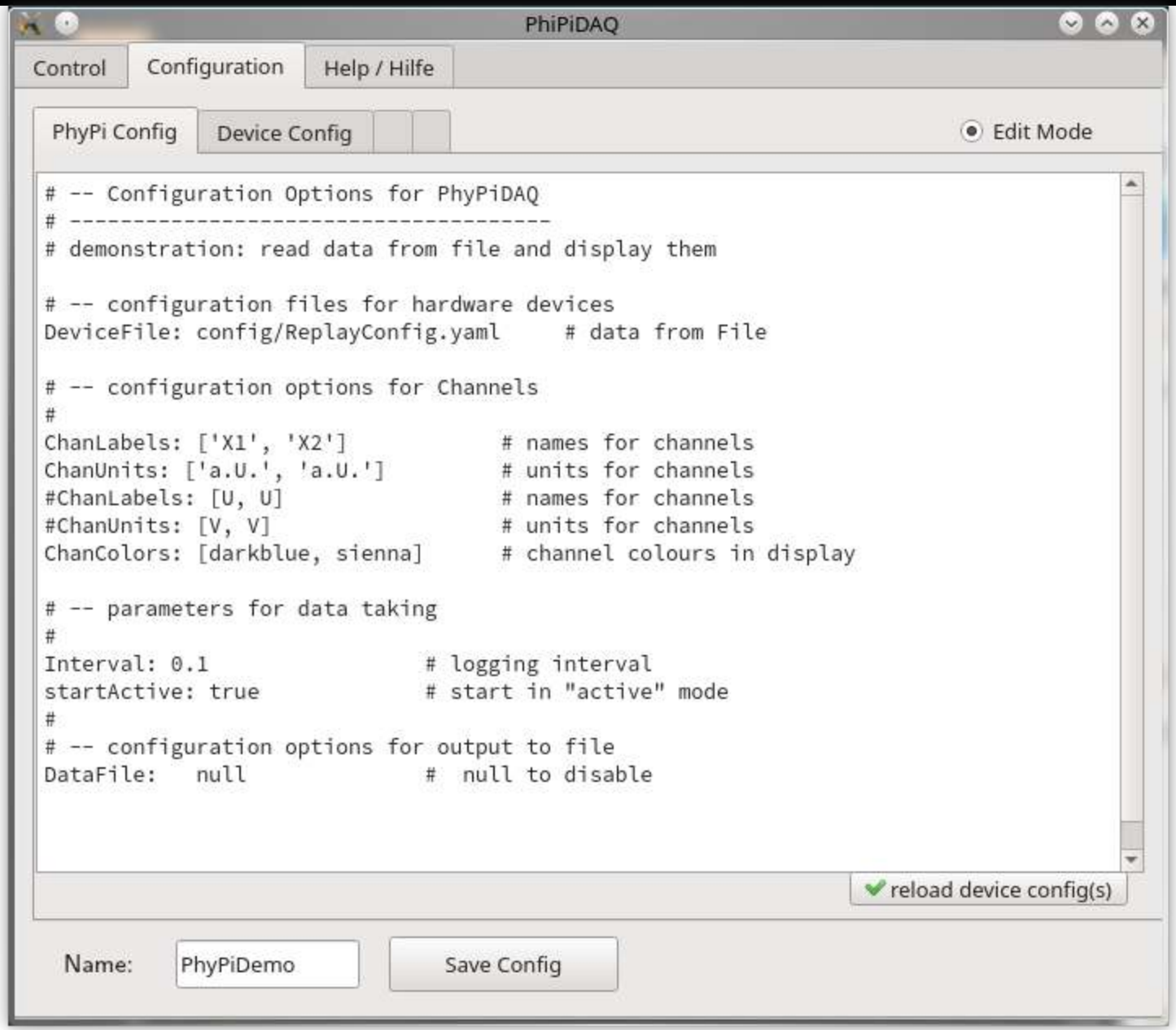
python Code

phypi.py

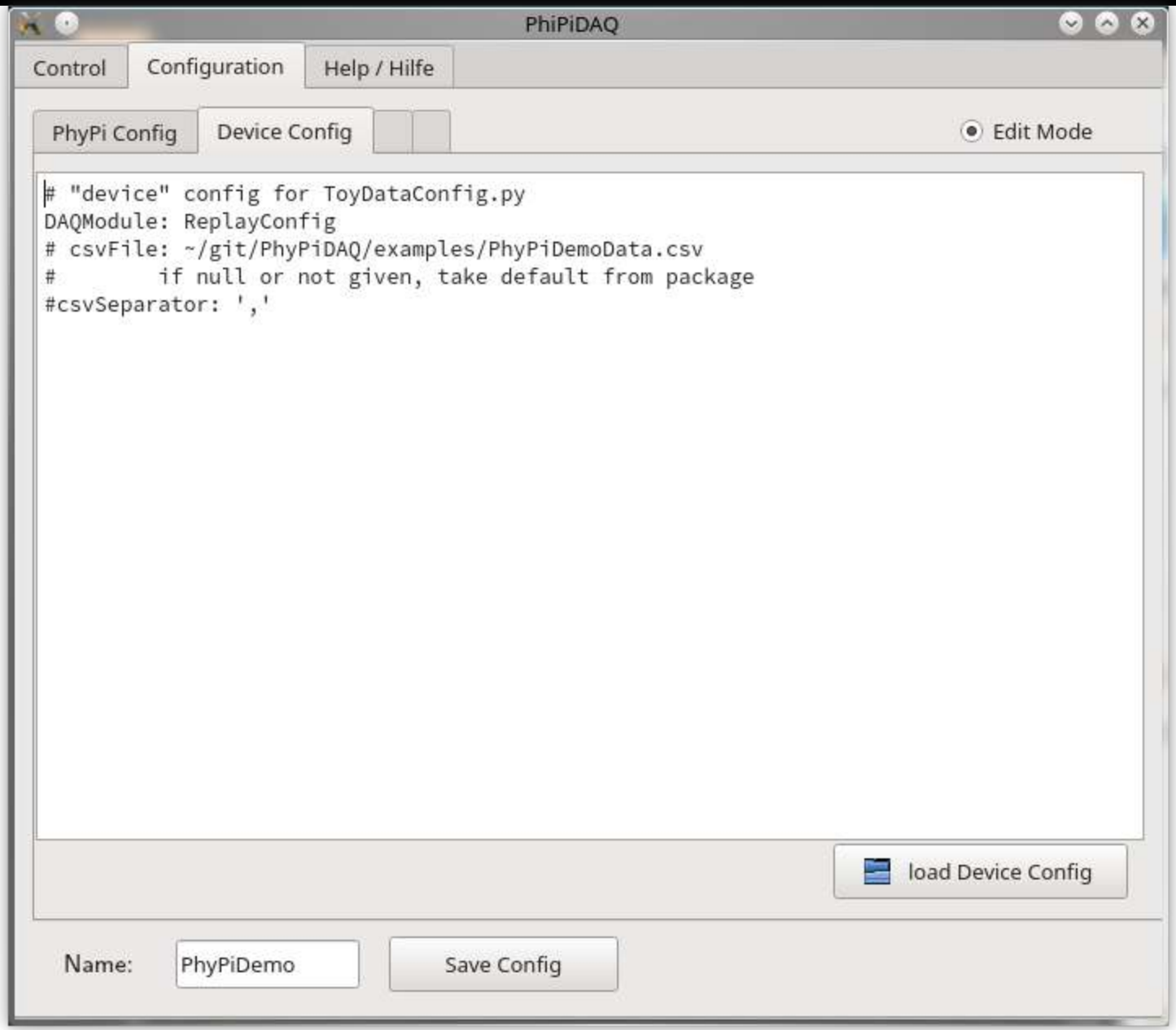


Klick auf StartRun führt run_phypi.py mit der erstellten Konfiguration aus

Anhang 4: Grafische Oberfläche (2)



Anhang 4: Grafische Oberfläche (3)



Anhang 5: Datennahme mit python-Code (2)

Beispiel

display_analog2.py

Code in PhyPiDAQ ist so strukturiert, dass die Komponenten auch leicht in eigenen Code eingebunden werden können

→ Einführung in die **textuelle Programmierung**

```
#!/usr/bin/env python3
'''read_analog.py
    this script illustrates the general usage of package phypidaq
    prints data read from an analog channel
'''
import time, numpy as np
# import module controlling readout device
from phypidaq.ADS1115Config import *
# create an instance of the device
device = ADS1115Config()
# initialize the device
device.init()
# reserve space for data (here only one channel)
dat = np.array([0.])
# read-out interval in s
dt = 1.
# start time
T0 = time.time()
print(' starting readout,      type <ctrl-C> to stop')
# readout loop, stop with <crtl>-C
while True:
  device.acquireData(dat)
  dT = time.time() - T0
  print('%.2g, %.4g' %(dT, dat) )
  time.sleep(dt)
```

Anhang 5: Datennahme mit python-Code (2)

Datennahme mit grafischer Ausgabe

Beispiel

read_analog.py

```
#!/usr/bin/env python3
import time, numpy as np
from phypidaq.ADS1115Config import *
from phypidaq.Display import *
# create device and display ...
device = ADS1115Config( {'ADCChannels': [0,1]} ) # channels 0 and 1
  # dictionary with graphics options
ddict = {'NChannels': 2, 'XYmode': False} # configuration options
display = Display( interval=0.1, confdict=ddict) # display 2 channels
device.init()
display.init()
dat = np.array([0., 0.])  # reserve space for data (two channels)

print(' starting readout,      type <ctrl-C> to stop')
T0 = time.time() # start time
# readout loop, stop with <crtl>-C
try:
  while True:
    device.acquireData(dat)
    dT = time.time() - T0
    print('%.2g, %.4g %.4g' %(dT, dat[0], dat[1]) )
    display.show(dat)
except KeyboardInterrupt:
  print('ctrl-C received - ending')
  device.closeDevice()
  display.close()
```