

PhyPiDAQ

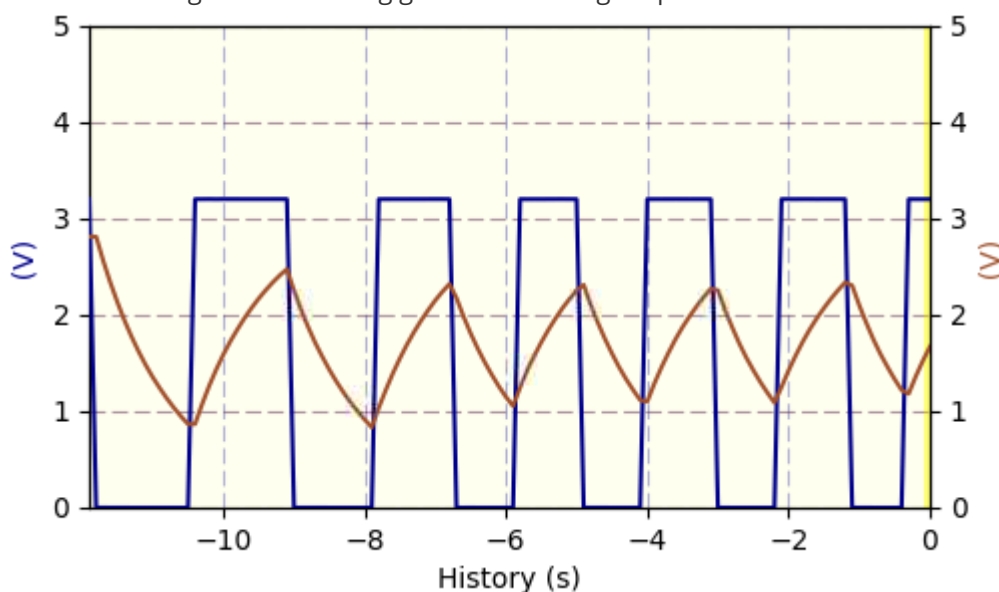
Datenerfassung und Analyse für die Physikausbildung mit Raspberry Pi

Dieser Code in der Programmiersprache *python3* bietet einige grundlegende Funktionen zur Datenerfassung und -visualisierung wie Datenlogger, Balkendiagramm, XY- oder Oszilloskopanzeige und Datenaufzeichnung auf Festplatte.

Neben den GPIO Ein- und Ausgängen des Raspberry Pi werden die Analog-Digital-Wandler ADS1115 und MCP3008 sowie USB-Oszilloskope (PicoScope der Firma picotech) als Eingabegeräte für analoge Daten sowie eine Reihe von digitalen Sensoren mit Protokollen wie I²C oder SPI unterstützt.

Das Paket bietet eine Abstraktionsschicht für Messgeräte und Sensoren, die an einen Raspberry Pi angeschlossen sind. Eigene Klassen für jedes Gerät bieten eine einfache, einheitliche Schnittstelle, die nur die Methoden `init(<config_dictionary>)`, `acquireData(buffer)` und `closeDevice()` enthalten. Einfache Beispiele mit minimalem Code veranschaulichen die Verwendung. Die grafische Benutzeroberfläche `phypi.py` und das Skript `run_phypi.py` bieten eine konfigurierbare Umgebung für komplexere Messprojekte.

Abb. 1: Darstellung der Zeitabhängigkeit von zwei Signalquellen an einem AD-Wandler



Die Beschreibung des zu Grunde liegenden Konzepts, eine Liste der empfohlenen Komponenten und ausführlich dokumentierte Beispiele finden sich in der [Masterarbeit von Moritz Aupperle](#).

Schnellstart

Nach der Installation - siehe unten - steht eine Reihe von einheitlichen Klassen für die Datenerfassung, Visualisierung und Aufzeichnung aus dem Unterverzeichnis `./phypidaq/` zur Verfügung. Jedes unterstützte Gerät benötigt eine spezifische Konfiguration, die aus Konfigurationsdateien im Unterverzeichnis `./config/` gelesen wird. Die Gesamtkonfiguration wird in Konfigurationsdateien vom Typ `.daq` angegeben, die spezifizieren, welche Geräte und Anzeigemodule verwendet werden sollen, welche

Ausleserate, Kalibrierungen oder analytische Formeln für aufgezeichnete Daten gelten sollen, oder auch Bereiche und Achsenbeschriftungen der grafischen Ausgabe.

Die grafische Benutzeroberfläche `phypi.py` hilft bei der Verwaltung der Konfigurationsoptionen und kann zum Starten der Datenerfassung verwendet werden. In diesem Fall werden Konfigurationen und erzeugte Datendateien in einem dedizierten Unterverzeichnis in `$HOME/PhyPi` abgelegt. Der Name des Unterverzeichnisses wird von einem benutzerdefinierten Tag und dem aktuellen Datum und der Uhrzeit abgeleitet.

Die Datenerfassung kann auch über die Kommandozeile gestartet werden:

```
run_phypi.py <config_file_name>\.daq
```

Wenn keine Konfigurationsdatei angegeben ist, wird der Standardwert `PhyPiConf.daq` verwendet.

Das Unterverzeichnis `./examples/` enthält eine Reihe einfacher Python-Skripte, die die Verwendung der bereitgestellten Datenerfassungs- und Anzeigemodule mit minimalem Code veranschaulichen. Außerdem sind vorbereitete Konfigurationen (*.daq*-Dateien) enthalten, die für typische Messaufgaben mit dem Script `run_phypi.py` verwendet werden können.

Konfigurationsdateien für PhyPiDAQ

Mit dem Skript `run_phypi.py` können sehr allgemeine Messaufgaben ausgeführt werden, ohne eigenen Code schreiben zu müssen. Die Konfigurationsoptionen für Eingabegeräte und deren Kanäle sowie für die Anzeige- und Datenspeichermodule werden in einer globalen Konfigurationsdatei vom Typ `.daq` angegeben, die Verweise auf Gerätekonfigurationsdateien vom Typ `.yaml` enthält.

Generell entspricht die in den Konfigurationsdateien verwendete Syntax der mark-up Sprache *yaml*. Insbesondere kennzeichnet Text nach einem `#`-Zeichen erklärende Kommentare oder enthält alternative, auskommentierte Konfigurationsoptionen, die durch Löschen des `#`-Zeichens aktiviert werden können.

Hauptkonfiguration

Ein typisches, ausführlich kommentiertes Beispiel für die Hauptkonfigurationsdatei sieht wie folgt aus:

Inhalt der Hauptkonfigurationsdatei `PhyPiConf.daq`

```
# -- Konfigurations-Optionen fuer PhyPiDAQ
# -----

#
# -- Konfigurationsdateien fuer Geraete
#
DeviceFile: config/ADS1115Config.yaml      # 16 bit ADC, I2C bus
## optional:
#DeviceFile: config/MCP3008Config.yaml    # 10 bit ADC, SPI-Bus
#DeviceFile: config/MCP3208Config.yaml    # 12 bit ADC, SPI-Bus
#DeviceFile: config/PSConfig.yaml         # PicoTechnology USB-Oszilloskop
#DeviceFile: config/MAX31865Config.yaml    # pt100 Temperatursensor
#DeviceFile: config/GPIOCCount.yaml       # Frequenzzaeher
#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: ToyData.yaml                # Simulierte Daten

## Beispiel für die Verwendung mehrerer Sensoren:
#DeviceFile: [config/ADS1115Config.yaml, config/GPIOCount.yaml]

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

#
# -- Konfiguration der grafischen Anzeige
#
Interval: 0.1                # Datennahme-Intervall in Sekunden
#NHHistoryPoints: 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

```

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

# Speicherung der letzten NHistoryPoints Datenpunkte
#bufferData: PhyPiData          # Dateiname fuer (optionale) Speicherung
#bufferData: null               #    null zum Ausschalten; Voreinstellung: Datei PhyPiData.dat
```

Gerätekonfigurationen

Die Gerätekonfiguration für den sehr flexibel einsetzbaren Analog-Digital-Wandler **ADS1115** mit 16 Bit Auflösung und Ausleseraten bis zu 860 Hz gibt die aktiven Kanäle und deren Wertebereiche an.

Inhalt der Konfigurationsdatei `ADS1115Config.yaml`

[illegible]

Das **USB-Oszilloskop** PicoScope kann ebenfalls als Datenlogger eingesetzt werden. In diesem Fall wird über eine Anzahl von Messungen mit sehr hoher Ausleserate gemittelt. Wählt man z.B. ein Messintervall von 20 ms, so wird 50 Hz- Rauschen effizient herausgemittelt.

Inhalt der Gerätekonfiguration `PSconfig.yaml`

```
# Konfiguration für PicoScope als Datenlogger

DAQModule: PSConfig # relevantes phypidaq-Modul

PSmodel: 2000a      # PicoScope Modell (PS2000a ist die Vorgabe)

# Konfiguration der Kanäle
picoChannels: [A, B] # Kanal A und B
ChanRanges: [2., 2.] # Wertebereich
ChanOffsets: [-1.95, -1.95] # analoger Offset, wird vor Anzeige addiert
ChanModes: [DC, DC] # Kanal-Kopplung (DC oder AC)
sampleTime: 2.0E-02 # Dauer der Datenaufnahme
Nsamples: 100       # Zahl der Messungen

# trigger
trgActive: false # Aufnahme ohne Oszilloskop-Trigger
trgChan: A

# Interner Signalgenerator
frqSG: 0. # aus
```

Beispiele für andere Geräte, wie den Analog-Digital-Wandler MCP3008, für Ratenmessungen über die GPIO - Pins des Raspberry Pi oder Temperaturmessungen mit dem digitalen 1-Wire Thermometer DS18B20, PT100-Sensoren am MAX31865 'Resistance-to-Digital Converter' oder mit Thermoelementen (Typ K) am MAX31855 'Thermocouple-to-Digital Converter' sind im Konfigurationsverzeichnis `./config/` enthalten, siehe `MCP3008Config.yaml`, `GPIOcount.yaml`, `DS18B20Config.yaml`, `MAX31865Config.yaml` oder `MAX31855Config.yaml`.

Installation von PhyPiDAQ auf dem Raspberry Pi

Beziehen des PhyPiDAQ Codes und einfache Installation

Bitte beachten Sie, dass Ihr Raspberry Pi für die folgenden Schritte mit dem Internet verbunden sein muss. Mit dem Befehl `git` lassen sich alle Dateien des Pakets `PhyPiDAQ` herunterladen. Zur Installation von `git` geben Sie nach dem Einrichten Ihres Raspberry Pi mit dem aktuellen Debian-Release *stretch* im Konsolenfenster folgenden Befehl ein:

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

Zur Installation von `PhyPiDAQ` geben Sie folgende Befehle ein:

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

PhyPiDAQ* basiert auf Code aus anderen Paketen, die die Treiber für die unterstützten Geräte und Bibliotheken für die Visualisierung bereitstellen. Die notwendigen Befehle zu deren Installation sind im Skript `installlibs.sh` zusammengefasst. Geben Sie auf der Kommandozeile folgende Befehle ein (ohne den erklärenden Text nachdem `#`-Zeichen):

```
cd ~/git/PhyPiDAQ # ins Installationsverzeichnis wechseln
git pull          # optional, falls Sie PhyPiDAQ aktualisieren möchten
./installlibs.sh  # Installations-Script ausführen
```

Damit ist die Installation schon abgeschlossen und `PhyPiDAQ` ist bereit für den ersten Einsatz.

Die letzten Zeilen der Installationsvorschrift gelten auch, wenn eine schon installierte Version von `PhyPiDAQ` aktualisiert werden soll.

Um die Installation auch ohne angeschlossene Hardware oder auf einem anderen System als dem Raspberry Pi zu testen, kann PhyPiDAQ im Demo-Modus gestartet werden:

```
cd ~/git/PhyPiDAQ # ins Installationsverzeichnis wechseln
./run_phypi.py    # run_phypi.py mit PhyPiDemo.daq ausführen
```

Anmerkung

Schüler oder Studierende zu Beginn mit dem vollen Umfang des Pakets *PhyPiDAQ* zu konfrontieren, ist aus didaktischer Sicht wenig angebracht. Stattdessen wird empfohlen, ein Arbeitsverzeichnis zu erstellen und benötigte Beispiele von dort in ein eigenes Arbeitsverzeichnis zu kopieren. Dies wird durch folgende Befehle erreicht:

```
# Erzeugen eines Arbeitsverzeichnis PhyPi und Kopieren von Beispielen und
Konfigurationsdateien in das neu erzeugte Verzeichnis.
cd ~/git/PhyPiDAQ
./install_user.sh

# klickbares Symbol auf dem Desktop zum Zugang zu phypi
cp ~/git/PhyPiDAQ/phypi.desktop ~/Desktop
```

Um versehentliches Überschreiben von Dateien im Paket *PhyPiDAQ* zu vermeiden, sollte eine Verschiebung bzw. Kopieren in den Systembereich in Erwägung gezogen werden, z. B. nach `/usr/local/`:

```
sudo cp ~/git/PhyPiDAQ /usr/local/
```

Die Pfade in `~/Desktop/phypi.desktop` müssen dann ebenfalls entsprechend angepasst werden. Dies wird am einfachsten durch Klicken mit der rechten Maustaste auf das *phypi*-Symbol erreicht. Im sich dann öffnenden Menü den Dialog "Eigenschaften" wählen und alle Pfade von `~/git/` -> `/usr/local/` ändern.

Dokumentation der Abhängigkeiten von externen Paketen

Dies ist die Liste der externen Pakete, von denen `PhyPiDAQ` abhängt:

- die Adafruit Python MCP3008 Bibliothek
https://github.com/adafruit/Adafruit_Python_MCP3008
- die Adafruit Python ADX1x15 Bibliothek
https://github.com/adafruit/Adafruit_Python_ADS1x15
- die Adafruit Python MAX31855 Bibliothek
https://github.com/adafruit/Adafruit_Python_MAX31855
- die w1thermsensor Bibliothek von Timo Furrer
<https://github.com/timofurrer/w1thermsensor>
- Komponenten des picoDAQ-Projekts
<https://github.com/GuenterQuast/picoDAQ>
- das *python* Interface für die PicoScope Treiber des *pico-python*-Projekts von Colin O'Flynn
<https://github.com/colinoflynn/pico-python>
- die C-Treiber aus dem Pico Technology Software Development Kit
<https://labs.picotech.com/raspbian>

Zur Vereinfachung der Installation werden Installationsdateien für benötigte externe Pakete und für die Module dieses Pakets als Debian Installationsdateien im *.deb*-Format oder als python-Module im *pip*-Wheel-Format im Unterverzeichnis `./installlibs/` bereitgestellt.

Die Module zur Visualisierung hängen von *matplotlib.pyplot*, *Tkinter* und *pyQt5* ab, die ebenfalls noch installiert werden müssen.

Die vom oben schon verwendeten Script `installlibs.sh` ausgeführten Schritte sind die folgenden:

```
# script installlibs.sh
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-scipy
```

```

sudo apt-get install python3-matplotlib
sudo apt-get install python3-pyqt5
sudo apt-get install libatlas-base-dev # wird benoetigt für neueste Version von numpy
sudo pip3 install pyyaml
# Treiber for unterstützte Sensoren und Komponenten
sudo pip3 install installlibs/whl/*.whl
# Treiber für PicoScope 2000 und 2000B
sudo dpkg -i installlibs/picoscopelibs/*.deb

sudo usermod -a -G tty pi # USB-Zugang für user pi gewaehren

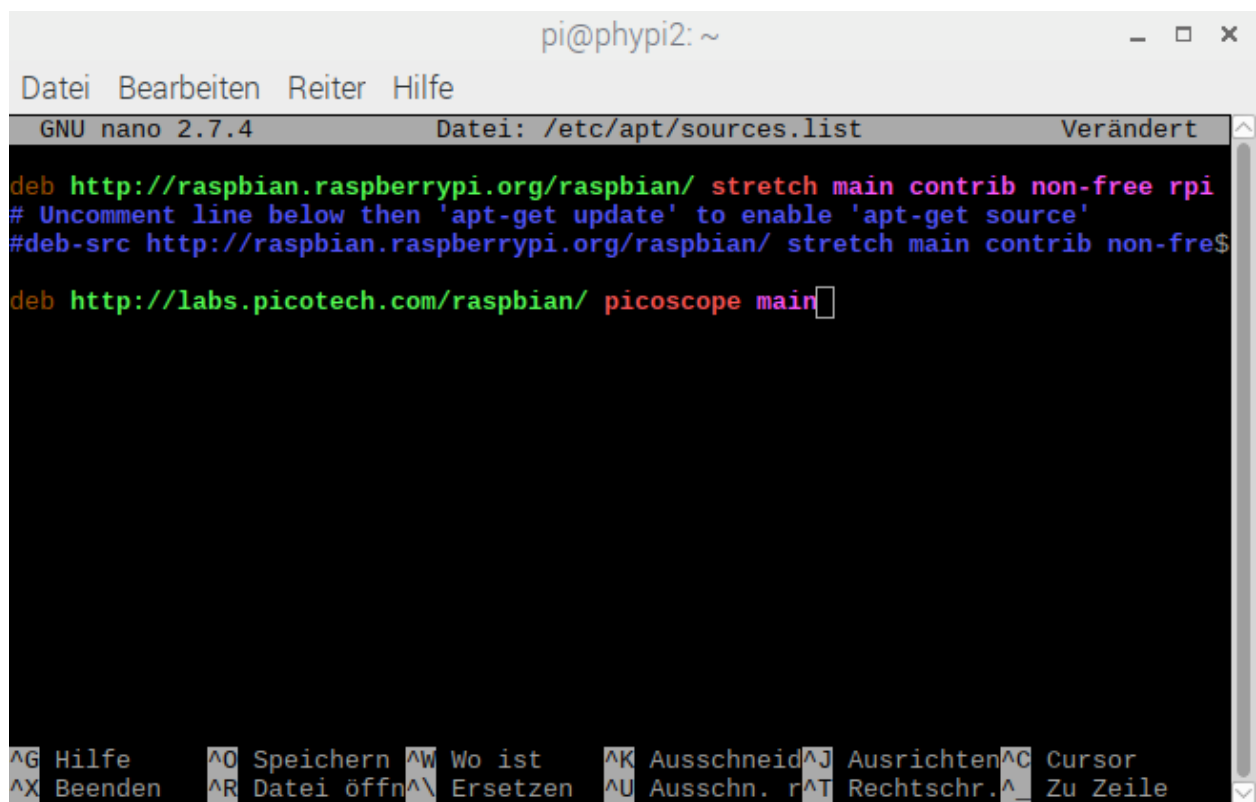
```

Die Treiber für PicoTech-Oszilloskope können auch von der Webseite des Herstellers bezogen werden; das picotech-raspbian-Repository kann dazu hinzugefügt werden:

1. Öffnen Sie die Datei /etc/apt/sources.list über die Kommandozeile mit `sudo nano`

/etc/apt/sources.list .

2. Navigieren Sie mit den Pfeiltasten in die nächste freie Zeile und ergänzen Sie den Eintrag `deb`
`http://labs.picotech.com/raspbian/ picoscope main` in der Datei /etc/apt/sources.list.



```
pi@phypi2: ~
Datei Bearbeiten Reiter Hilfe
GNU nano 2.7.4 Datei: /etc/apt/sources.list Verändert
deb http://raspbian.raspberrypi.org/raspbian/ stretch main contrib non-free rpi
# Uncomment line below then 'apt-get update' to enable 'apt-get source'
#deb-src http://raspbian.raspberrypi.org/raspbian/ stretch main contrib non-fre$
deb http://labs.picotech.com/raspbian/ picoscope main
^G Hilfe ^O Speichern ^W Wo ist ^K Ausschneid ^J Ausrichten ^C Cursor
^X Beenden ^R Datei öffn ^\ Ersetzen ^U Ausschn. r ^T Rechtschr. ^_ Zu Zeile
```

3. Speichern Sie die Datei /etc/apt/sources.list mit `Strg + O` und `Enter`.
4. Schließen Sie die Datei /etc/apt/sources.list mit `Strg + X`.

Nun können die Treiber für PicoScope-Geräte mit *apt-get* eingebunden und ggf. aktualisiert werden:

```
wget -O - http://labs.picotech.com/debian/dists/picoscope/Release.gpg.key | sudo apt-  
key add -  
sudo apt-get update  
sudo apt-get install libps2000  
sudo apt-get install libps2000a  
  
# Benutzer pi Zugriff auf den USB-Port ermöglichen  
sudo usermod -a -G tty pi
```

Experimente und Messungen mit PhyPiDAQ

Die hier bereit gestellte Software soll es sowohl Lernenden als auch Lehrenden ermöglichen, typische Messaufgaben im Physikunterricht durchzuführen. Dank der Realisierung mit Sensoren, die auch in Alltagsgeräten eingesetzt werden, und dem Raspberry Pi als Datennahme-Rechner können kostengünstige Einführungssets für Schülerversuche bereit gestellt werden.

Ein Vorschlag von Komponenten zur Grundausstattung wird in der Datei *Komponenten_fuer_PhyPi.pdf* beschreiben. Die zum Umgang damit notwendigen Grundkenntnisse werden in einem Einführungskurs erarbeitet, der in der Datei *Kurs_digitale_Messwerterfassung_mit_PhyPiDAQ.pdf* beschreiben ist. Zum Verständnis unumgänglich sind einige Grundkenntnisse in der Sprache *python* und die Vorgehensweise zum Ansprechen der GPIO-Pins des Raspberry Pi. Es folgt eine Einführung in die Analog-Digital-Wandlung und die Verwendung des Wandlerbausteins ADS1115. Am Ende steht die Kalibration und Verwendung eines NTC-Widerstands als Temperatursensor. Die letzte Stufe des Einführungskurses bildet ein mit einer Wägezelle und einem Instrumentenverstärker realisierter Kraftsensor.

Um das Erstellen von eigenem Code für jede Messaufgabe zu vermeiden, liefert das Paket *PhyPiDAQ* eine einheitliche Programmierschnittstelle, die verschiedene Sensoren unterstützt und Standard-Anzeigen für die Datenaufnahme bereit stellt.

Auslese eines Analog-Digitalwandlers

Ein einfaches Beispiel zur Auslese des Digital-Analog-Wandlers *ADS1115* illustriert die Anwendung (Script

`read_analog.py`):

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
'''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 <ctrl>-C
while True:
    device.acquireData(dat)
    dT = time.time() - T0
    print('%.2g, %.4g' %(dT, dat) )
    time.sleep(dt)

```

Werden andere Bausteine zur Dateneingabe verwendet, wie zum Beispiel der Analog-Digital-Wandler *MCP3208*, der digitale Temperatursensor *18B20*, der Temperatur- und Drucksensor *BMP180* oder der Beschleunigungssensor *MMA8451*, so müssen nur zwei Zeilen am Anfang des Scripts angepasst werden

```

from phypidaq.<sensor> import *
device = <sensor>

```

Auslese eines Analog-Digitalwandlers und Visualisierung der Daten

Ein einfaches Beispiel zur Auslese von zwei Kanälen eines Analog-Digitalwandlers sowie die Echtzeit-Anzeige der Daten als Verlaufsdiagramm ist das Script `display_analog2.py`:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
'''display_analog2.py
    illustrates the general usage of package phypidaq
    prints and displays data read from 2 analog channels
'''
import time, numpy as np
# import module controlling readout device
from phypidaq.ADS1115Config import *
# import display
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
# ... and initialize
device.init()
display.init()

# reserve space for data (two channels)

```

```

dat = np.array([0., 0.])

print(' starting readout,      type <ctrl-C> to stop')
# start time
T0 = time.time()
try:
# readout loop, stop with <ctrl>-C
    while True:
        device.acquireData(dat)
        dT = time.time() - T0
        print('%0.2g, %0.4g %0.4g' % (dT, dat[0], dat[1]) )
        display.show(dat)
except KeyboardInterrupt:
    print('ctrl-C received - ending')
    device.closeDevice()
    display.close()

```

Datenaufnahme mit *phypi.py* oder *_run_phypi.py*

Das Script *run_phypi* stellt eine sehr allgemein und weitgehend konfigurierbare Auslese und Anzeige von Sensordaten bereit. Die Konfigurationsdateien im Dateiformat *.daq* enthalten dabei die notwendigen Informationen zum verwendeten Sensor, zu den Anzeigoptionen als Echtzeitanzeige, Verlaufsdiagramm oder xy-Darstellung sowie zur Kalibration oder Umrechnung von Sensordaten. Ein allgemeines Beispiel einer solchen Hauptkonfigurationsdatei wurde bereits oben vorgestellt. Mit entsprechend vorbereiteten Konfigurationsdateien lassen sich sehr flexibel die für bestimmte Experimente notwendigen Messungen und Anzeigen vorbereiten und durchführen. Mit Hilfe der grafische Oberfläche *phypi.py* lassen sich die Konfigurationen bequem anschauen, anpassen und abspeichern und die Datenaufnahme kann gestartet werden. Konkrete Beispiele sind im Verzeichnis *examples/* enthalten.

Barometer

Zur Messung und Aufzeichnung von Temperatur und Luftdruck mit einem *BMP180* -Sensor müssen nur vier Leitungen vom Sensor (+3,3V, Masse und die Signale SDA und SCL des I²C-Busees) an den Raspberry Pi angeschlossen werden. Als Beispiel ist hier die Konfigurationsdatei *Barometer-daq* gezeigt:

```

# Konfigurationsdatei Barometer.daq für PhyPiDAQ
# Temperatur und Luftdruck mit BMP180

DeviceFile: BMP180Config.yaml

DisplayModule: DataGraphs
Title: Temperatur & Luftdruck

ChanLabels: [Temperatur, Druck]           # Namen der Messgrößen
ChanLimits: [[0., 30.], [970., 1030.]]    # Wertebereich
ChanUnits: ['°C', 'hPa']                  # Einheiten
ChanColors: [darkblue, darkgreen]         # Farben in der Anzeige

Interval: 120.                            # Intervall für Datenaufnahme und Anzeige (in s)

```

Gleichzeitige Darstellung mehrerer LED-Kennlinien

Etwas aufwändiger ist die simultane Darstellung mehrerer Diodenkennlinien mit einem Analog-Digital-Wandler. Dazu werden die Dioden mit jeweils einem Vorwiderstand versehen parallel an eine variable Versorgungsspannung angeschlossen, die mit Hilfe eines Potentiometers aus der Betriebsspannung von 5 V des Raspberry Pi erzeugt werden kann. Gemessen werden auf Kanal 0 des ADS1115 die Versorgungsspannung sowie die Spannungen über den Dioden auf den Kanälen 1-3. Der Strom durch jede der Dioden wird aus dem Spannungsabfall über dem jeweiligen (bekannten) Vorwiderstand bestimmt. Dies ist dank der Möglichkeit, Formeln auf die Eingangsspannungen anzuwenden, leicht mit PhyPiDAQ realisierbar.

```
# Konfiguration Diodenkennlinie.daq für PhyPiDAQ

DeviceFile: ADS1115_Diode.yaml # definiert 4 aktive Kanäle mit Verstärkung 1

# Anwenden von Umrechnungsformeln auf die Eingangskanäle
# aus den Messgrößen an den Kanälen c0 und c1-c3 werden 6 Werte berechnet
ChanFormula:
- c1 # U Diode c1
- (c0 - c1) / 0.120 # I Diode c1
- c2 # U Diode c2
- (c0 - c2) / 0.120 # I Diode c2
- c3 # U Diode c3
- (c0 - c3) / 0.120 # I Diode c3

# Namen, Messgrößen und Einheiten für die sechs Ausgabewerte
ChanNams: ['F0', 'F1', 'F2', 'F3', 'F4', 'F5'] # Namen der Kanäle
ChanLabels: [U, I, U, I, U, I] # Messgrößen
ChanUnits: [V, mA, V, mA, V, mA] # Einheiten
ChanColors: [black, red, black, green, black, blue] # Anzeigefarben
ChanLimits: # Wertebereiche
- [0., 3.1] # U D1
- [0., 30.] # I D1
- [0., 3.1] # U D2
- [0., 30.] # I D2
- [0., 3.1] # U D3
- [0., 30.] # I D3

DisplayModule: DataLogger
Chan2Axes: [0,1,0,1,0,1] # Kanal auf linker(0) oder rechter(1) Achse
# Voreinstellung [0,1,1,...]

Interval: 0.1 # Anzeige-Intervall
XYmode: true # XY-Darstellung
xyPlots: # Paare von Kanälen als xy-Grafik
- [0,1] # U0 - I0
- [2,3] # U1 - I1
- [4,5] # U2 - I2
```

Datenaufnahme mittels USB-Oszilloskop

Digitaloszilloskope mit USB-Anschluss sind außer für die reine Anzeige von Signalen auch sehr flexibel einsetzbare Datenaufnahme-Systeme. `PhyPiDAQ` unterstützt USB-Oszilloskope der aus PicoScope-Reihe der Firma PicoTech, für die es eine gut dokumentierte Schnittstelle für Anwendungsprogramme gibt. Die Installation der notwendigen Bibliotheken wurde weiter oben beschrieben und findet sich auch in der Datei *doc/Einrichten_des_Raspberry_pi.pdf*. Die preisgünstigste Variante mit einer Bandbreite von 10 MHz gibt es im Handel bereits für ca. 100,-€; empfehlenswert für praktisch alle Anwendungen im Physikunterricht ist die 50 MHz-Variante, mit der auch kurze Signale von Einzelphotondetektoren dargestellt werden können.

Setzt man das Oszilloskop als Datenlogger ein, so werden viele Messwerte über eine kurze Zeit von wenigen Millisekunden aufgezeichnet und entweder der Mittelwert (bei langsam veränderlichen Signalen) oder der Effektivwert (bei Wechselspannungen) als Datenpunkte übergeben. Die Realisierung einer Lautstärkemessung illustriert die Datei *NoiseMeter.daq*:

```
# Konfiguration NoiseMeter.daq für PhyPiDAQ
#   Ausgabe des Effektivwerts eines Schallsignals

# Konfiguration des Oszilloskops
DeviceFile: PSConfig_sound.yaml          # PS2000B -Typen
#DeviceFile: PSConfig2000A_sound.yaml    # PS2000A - Typ

#DisplayModule: DataLogger
DisplayModule: DataGraphs
Title: Noisemeter
Interval: 0.05                          # logging interval

ChanLabels: [U_eff]                     # Messgröße
ChanUnits: [V]                          # Einheit
ChanColors: [darkblue]                  # Farbe
ChanLimits:
- [0., 0.035] # scope at 50mV, eff. Voltage is smaller
## - [0., 1.]

DataFile: null                          # file name for output file
#DataFile: testfile                     # file name for output file
```

Ein Blick in die Konfiguration des Oszilloskops ist an dieser Stelle hilfreich. Benötigt werden die für ein Oszilloskop notwendigen Informationen wie Kanalwahl, Messbereich, AC- oder DC-Kopplung, Zeitbasis und Triggereinstellung. Die in der Konfigurationsdatei *NoiseMeter.daq* spezifizierte Datei *PSConfig_sound.yaml* sieht wie folgt aus:

```
# Konfiguration eines PicoScope 2000

DAQModule: PSConfig

#PSmodel: '2000'      # PS model 220xA
PSmodel: '2000a'      # PS model 2y0xB

# channel configuration
picoChannels: [A]      # Kanals A
ChanRanges: [0.05]     # +/- 50 mV
ChanModes: [AC]        # AC-Kopplung
```

```

sampleTime: 2.0E-02 # 20 ms Datenaufnahme
Nsamples: 200      # mit 200 Messpunkten

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

# frqSG: 100.E+3 # put 0. do disable
frqSG: 0. # Signalgenerator aus

# spezielle Option für PhyPiDAQ
ChanAverages: ['rms'] # ['mean'] für Mittelwert oder ['rms'] für Effektivwert

```

Analog zu einem Mikrofon lässt sich auch ein Geophon, z.B. das SM-24, anschließen, um einen Erschütterungs- oder Erdbebendetektor zu realisieren.

Verwendet man statt der Option *ChanAverages: ['rms']* die Option *ChanAverages: ['mean']* (letzteres ist die Voreinstellung), so ergibt sich ein sehr flexibler Datenlogger mit der vollen Flexibilität eines Oszilloskops, also in weiten Bereichen konfigurierbare Messbereiche auch für negative Spannungen und Überspannungsfestigkeit. Mittelt man Messwerte über 20 ms (wie oben in der Konfigurationsdatei voreingestellt), so werden Störungen durch die Frequenz von 50Hz des Stromnetzes herausgemittelt und man erhält sehr saubere Messwerte.

Test der Oszilloskop-Funktion

Leider läuft die Oszilloskop-Software der Firma PicoTech (noch) nicht auf der Raspberry Pi. Als Test der Funktionalität eines PicoScopes gibt es daher das *python*-Script *examples/runOsci.py*, das eine Oszillografenanzeige darstellt. Das Script verwendet Funktionalität aus dem Paket *picoDAQ* und stellt bis auf fehlende interaktive Einstellmöglichkeiten ein vollwertiges Oszilloskop für den Raspberry Pi bereit. Die notwendigen Einstellungen finden sich in der Steuerdatei *PSOsci.yaml*, die genau so aufgebaut ist wie das Beispiel oben:

```

# Konfiguration für PicoScope
picoChannels: [A, B] # Kanäle A und B aktiv
ChanModes: [AC, AC] # mit AC-Kopplung
ChanRanges: [0.5, 0.5] # Messbereich +/- 0.5 V
ChanColors: [darkblue, sienna]

sampleTime: 40.E-3 # 40 ms
Nsamples: 400 # 400 Samples, d.h. 1 Datenpunkt alle 10 µs

trgChan: A # Trigger-Kanal A
trgThr: 0.05 # Triggerschwelle 0.050 V
trgTyp: Above # Above, Below, Rising, Falling ....
trgTO: 500 # 500 ms Timeout
trgActive: true # Trigger aktiv

```

```
trgDelay: 0      # keine Triggerverögerung
pretrig: 0.05    # 5% der Daten vor Triggerzeitpunkt anzeigen

#frqSG: 10.E+3 # Frequenz des Signalgenerators in Hz
frqSG: 0.0      # Signalgenerator aus
```

Zum Test reicht ein offenes Kabelende am Eingangskabel zu Kanal A, über das 50Hz-Einstreuungen aus dem Stromnetz aufgefangen werden. Höhere Störfrequenzen findet man in der Nähe von Schaltnetzteilen, z. B. dem Steckernetzteil des Raspberry Pi.

Übersicht über Dateien im Paket PhyPiDAQ

Programme

- `run_phypi.py`
Datennahme und Anzeige wie in Konfigurationsdateien angegeben (Vorgabe `PhyPiConf.daq` und `.yaml`-Dateien im Verzeichnis `config/`)
- `phypi.py`
graphische Oberfläche zum Editieren der Konfiguration und Starten des Skripts `run_phypi.py`

Module

- `phypidaq/__init__.py`
Initialisierung für das Paket *phypidaq*
- `phypidaq/_version_info.py`
Versionsinformation für das Paket *phypidaq*
- `phypidaq/ADS1115Config.py`
Klasse zur Handhabung des Analog-Digital-Wandlers ADS1115
- `phypidaq/MCP3008Config.py`
Klasse zur Handhabung des Strom- und Spannungssensors INA219
- `phypidaq/INA219Config.py`
Klasse für Analog-Digital-Wandlers MCP3008 /MCP3208
- `phypidaq/DS18B20Config.py`
Klasse zur Handhabung des digitalen Thermometers DS18B20
- `phypidaq/BMPx80Config.py`
Klasse zur Handhabung des digitalen Temperatur- und Drucksensors BMP180/280 oder BME280
- `phypidaq/MMA8451Config.py`
Klasse zur Handhabung des digitalen Beschleunigungssensors MMA8451
- `phypidaq/GPIOCount.py`
Klasse zur Ratenmessung an GPIO-Pins
- `phypidaq/MAX31855Config.py`
Klasse für Thermolement-Wandlers MAX31855
- `phypidaq/MAX31865Config.py`
Klasse für den Widerstand-nach-digital-Wandler MAX31865
- `phypidaq/PSConfig.py`
Klasse für PicoScope USB-Oszilloskope
- `phypidaq/TCS34725Config`
Klasse für TCS34725 RGB Farbsensor
- `phypidaq/VL53L1XConfig`
Klasse zur Ansteuerung des Abstandssensors VL53L1X
- `phypidaq/AS7262Config`
Klasse für sechs-Kanal Farbsensor AS7262
- `phypidaq/ToyDataConfig.py`
Klasse zur Erzeugung simulierter Daten (für Test, Fehlersuche oder Übungsaufgaben)
- `phypidaq/ReplayConfig`
Klasse zur Wiedergabe von Daten aus Datei
- `phypidaq/Display.py`
Interface und Hintergrund-Prozess zur Datenvisualisierung
- `phypidaq/DataLogger.py`
Klasse zur Anzeige von Datenverlauf und xy-Diagrammen
- `phypidaq/DataGraphs.py`
allgemeine Klasse zur Anzeige von Balkendiagrammen, Datenverläufen und xy-Diagrammen
- `phypidaq/DataRecorder.py`
Speichern von Daten im CSV-Format
- `runPhyPiDAQ.py`
Klasse für das Script `run_phypi.py`

- `runPhyPyUI.py`
Klasse für die grafische Oberfläche `phypi.py`, abgeleitet von `phypiUI`
- `phypiUI`
mit `pyuic5` aus `phypi.ui` erzeugte Basis-Klasse für grafische Oberfläche
- `phypi.ui`
Ausgabe von `designer-qt5`, beschreibt die grafischen Oberfläche

Konfigurationsdateien

- `PhyPiConf.daq`
Hauptkonfigurationsdatei, hängt von Dateien im Unterverzeichnis *config/* ab
- `config/ADS1115Config.yaml` 16 Bit Analog-Digitalwandler
- `config/MCP3008Config.yaml` 10 Bit Analog Digitalwandler
- `config/MCP3208Config.yaml` 12 Bit Analog Digitalwandler
- `config/INA219Config.yaml` Strom- Spannungssensor
- `config/DS18B20Config.yaml` digitaler Temperatursensor
- `config/BMP180Config.yaml` Temperatur- und Drucksensor
- `config/BMP280Config.yaml` Temperatur- und Drucksensor
- `config/GPIOCOUNT.yaml` Frequenzmessung an GPIO-Pin
- `config/MAX31855Config.yaml` Konverter für Thermoelement
- `config/MAX31865Config.yaml` Konverter für PT-100
- `config/TCS34752Config.yaml` RGB-Sensor
- `config/AS7262Config.yaml` 6-Kanal Farbsensor
- `config/VL53LxConfig.yaml` Abstandssensor
- `config/PSConfig.yaml` PicoScope USB-Oszilloskop

Beispiele

- `examples/read_analog.py`
sehr minimalistisches Beispiel zum Auslesen eines Kanals von einem Analog-Digital-Wandler
- `examples/display_analog.py`
Beispiel zum Auslesen eines Kanals von einem Analog-Digital-Wandler mit grafischer Anzeige des zeitlichen Verlaufs
- `examples/display_analog2.py`
Beispiel zum Auslesen von zwei Kanälen von einem Analog-Digital-Wandler mit grafischer Anzeige des zeitlichen Verlaufs
- `examples/read_INA210.py`
Beispiel zum Auslesen des Strom- und Spannungssensors INA219
- `examples/read_18B20.py`
Auslese des digitalen Temperatursensors DS18B20
- `examples/readBMP180.py`
Auslese des digitalen Temperatur- und Drucksensors BMP180/280
- `examples/readMMA8451.py`
Auslese des digitalen Beschleunigungssensors MMA8451
- `examples/runOsci.py`
Oszillogrammanzeigen wie in *.yaml*-Datei zur Konfiguration angegeben (Vorgabe `PSOsci.yaml`)

- `examples/GPIO-In-out.py`
Beispiel zur Ansteuerung der GPIO-Pins: Erzeugung einer Rechteckspannung am Ausgabe-Pin durch Verändern der Spannung am Eingabe-Pin
- `examples/poissonLED.py`
erzeugt ein zufälliges Signal an GPIO-Pin gemäß Poisson-Prozess
- `examples/FreqGen.py`
erzeugt Signal fester Frequenz an GPIO-Pin

Konfigurationsdateien für *run_phypi.py*

- `examples/Amperemeter.daq`
simultane Messung und Darstellung von Strom und ggf. Spannung mit dem Strom- und Spannungssensor INA219
- `examples/DiodenKennlinie.daq`
simultane Messung und Darstellung von drei Diodenkennlinien mit einem ADS1115 Digital-Analog-Wandler
- `examples/Barometer.daq`
nutzt Sensoren BMB180 der BMP280 zur Anzeige von Temperatur und Luftdruck
- `examples/Accelerometer.daq`
nutzt den Sensor MMA8451 zur Anzeige der x-, y- und z-Komponente der Beschleunigung.
- `examples/NoiseMeter.daq`
Messung der Lautstärke mit einem an ein PicoScope USB-Oszilloskop angeschlossenen Mikrofon; angezeigt werden die Effektivwerte von 200 in einem Zeitraum von 20 ms aufgezeichneten Messungen der Schallamplitude. Kann auch mit dem Geophon SM-24 verwendet werden
- `examples/RGBsensor.daq`
RGB Farbsensor
- `examples/ColorSpectrum.daq`
sechs-Kanal Farbsensor
- `examples/ToyData.daq` Erzeugung und Anzeige von simulierten Daten
- `examples/ReplayData.daq`
Daten aus Datei (für Demo-Zwecke)

Dokumentation

- `doc/Kurs_digitale_Messwerterfassung_mit_PhyPiDAQ.md (.pdf)`
Einführungskurs für Schüler zum Messen mit dem Raspberry Pi
- `doc/Einrichten_des_Raspberry_Pi.md (.pdf)`
Aufsetzen des Raspberry Pi für dieses Projekt
- `doc/Komponenten_fuer_PhyPi.md (.pdf)`
empfohlene Komponenten für dieses Projekt
- `doc/Bauanleitung_Kraftsensor.md (.pdf)`
Bauanleitung für Kraftsenor