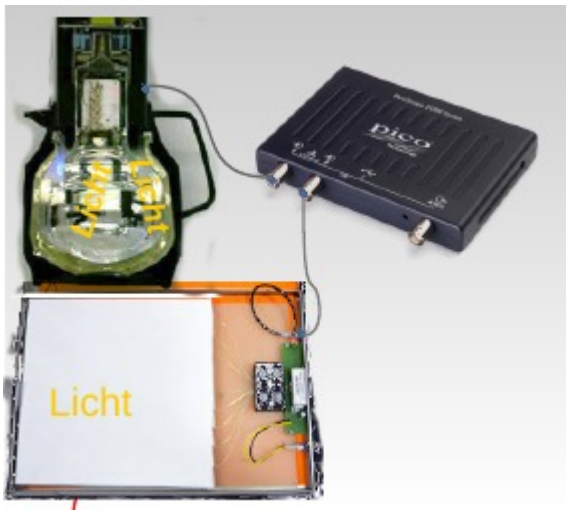
Software zur Auslese und Analyse der Experimente des Netzwerks Teilchenwelt

Kurzfassung:
Python-Script zur Aufnahme und Auswertung der Daten der CosMO-Detektoren und der Kamiokanne des Netzwerks Teilchenwelt mit einem USB-Oszilloskop der Firma PicoTechnology



Beschreibung der Funktionalität

Das Netzwerk Teilchenwelt, http://www.Teilchenwelt.de stellt Experimente zum Nachweis von Myonen aus der kosmischem Strahlung zur Verfügung. Dies sind die Szintillationszähler des CosMO-Experiments und der aus einer Kaffeekanne mit aufgesetzter Photoröhre bestehende Wasser-Cherenkov-Zähler "Kamiokanne". Diese Detektoren liefern kurze Signale von ca. 100 ns Dauer und einigen 10 bis 100 mV Pulshöhe, die mit einem Oszillographen sichtbar gemacht werden können.

Moderne USB-Oszilloskope wie das PicoScope der Firma PichoTechnology, siehe http://www.picotech.com, erlauben es, die Pulsformen nicht nur anzuzeigen, sondern auch in Echtzeit an einen Commputer zu exportieren, mit dem sie dann aufgezeichnet, angezeigt und analysiert werden können. Diesem Zweck dient das hier beschriebene Projekt "*picoCosmo*". Es ist auf Linux-Systemen und auch auf dem Raspberry Pi lauffähig und unterstützt PicoScope-Geräte mit zwei oder vier Kanälen.

picoCosmo nutzt zur Datenaufname den Puffermanager und die Echtzeit-Anzeigen des Projekts *picoDAQ* (https://github.com/Guenter.Quast/picoDAQ). Der Puffermanager von *picoDAQ* sammelt die Daten und verteilt sie an Echtzeit-Anzeigen oder weitere Prozesse zur Datenauswertung. *picoCosmo* ist eine angepasste und um umfangreiche Funktionalität zur Datenauswertung erweiterte Variante des Scripts *runDAQ.py* aus dem Projekt *picoDAQ*.

Die Analyse der aufgezeichneten Pulsformen verläuft in drei Schritten:

1. **Validierung der Trigger-Schwelle des Oszilloskops**

   Dazu wird der Signalverlauf um den Triggerzeitpunkt mit einem Musterpuls verglichen und das Signal akzeptiert, wenn die Form gut übereinstimmt und der Puls eine Mindesthöhe überschreitet.

2. **Suche nach Koinzidenzen**

   Als nächstes werden Pulse auf allen aktiven Kanälen in der Nähe des Triggerzeitpunkts gesucht. Bei mehr als einem angeschlossenen Detektor wird ein aufgezeichnetes Ereignis akzeptiert, wenn mindestens zwei in zeitlicher Koinzidenz auftreten.

3. **Suche nach verzögerten Pulsen**

   Im optionalen dritten Schritt werden weitere Pulse auf allen aktiven Kanälen gesucht und die Zeitdifferenz zum Triggerzeitpunkt festgehalten. Solche Pulse treten auf, wenn ein Myon aus der kosmischen Strahlung nach Durchgang durch den bzw. die Detektoren gestoppt und das aus dem Zerfall entstandene Elektron registriert wird. Die registrierten individuellen Lebensdauern folgen einer Exponential-Verteilung mit einer mittleren Lebensdauern von 2,2 µs, die auf diese Weise bestimmt werden kann.

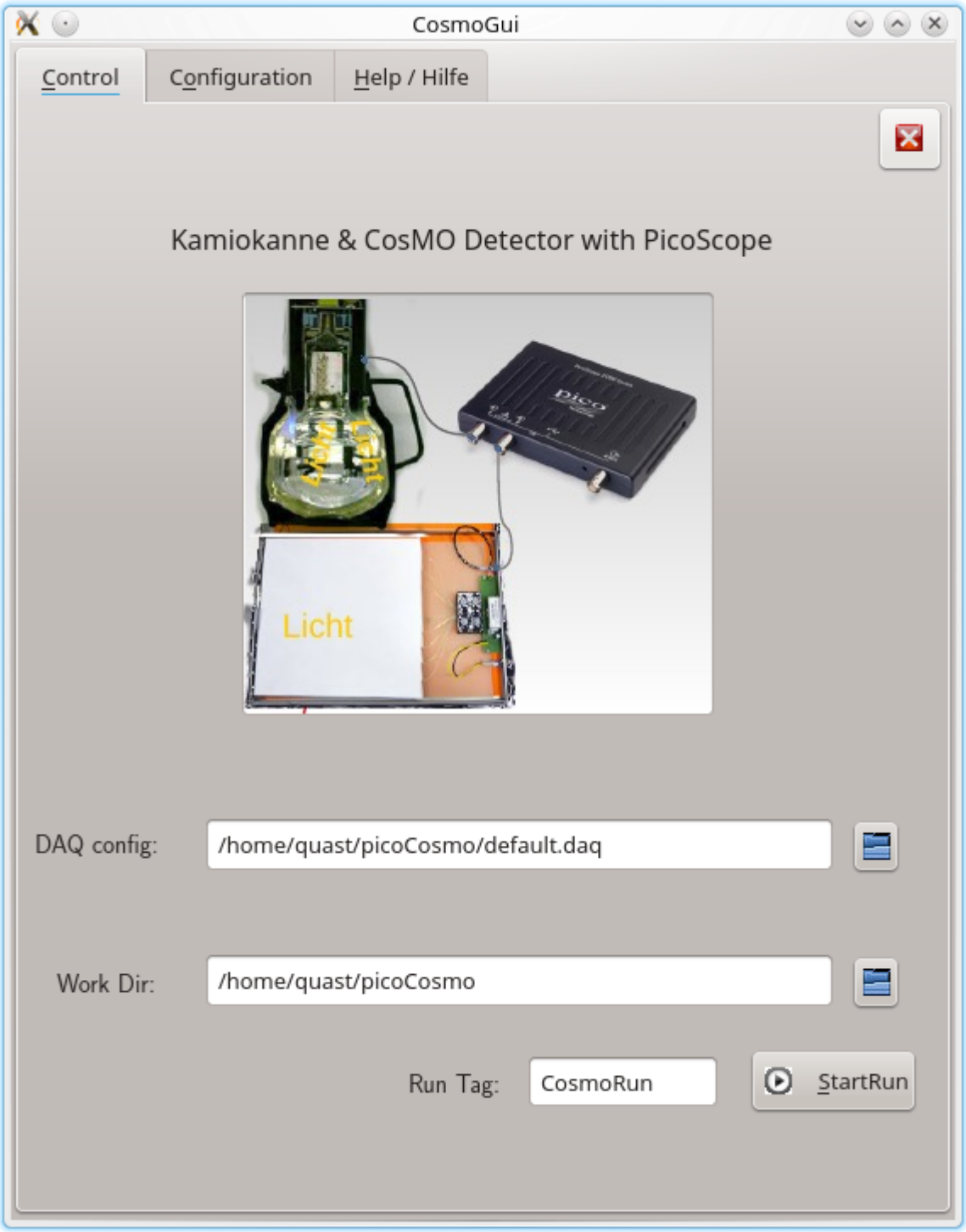
Die Software bietet Echtzeit-Anzeigen der Myon-Rate, der aufgenommenen Pulshöhen und der Myon-Lebensdauern. Zusätzlich können Mehrfach-Pulse als Rohdaten der registrierten Pulsformen oder als Bilder im *.png*-Format gespeichert werden.

Details zu Abhängigkeiten und zur Installation finden sich in der Datei README_de.md.

Starten des Programms

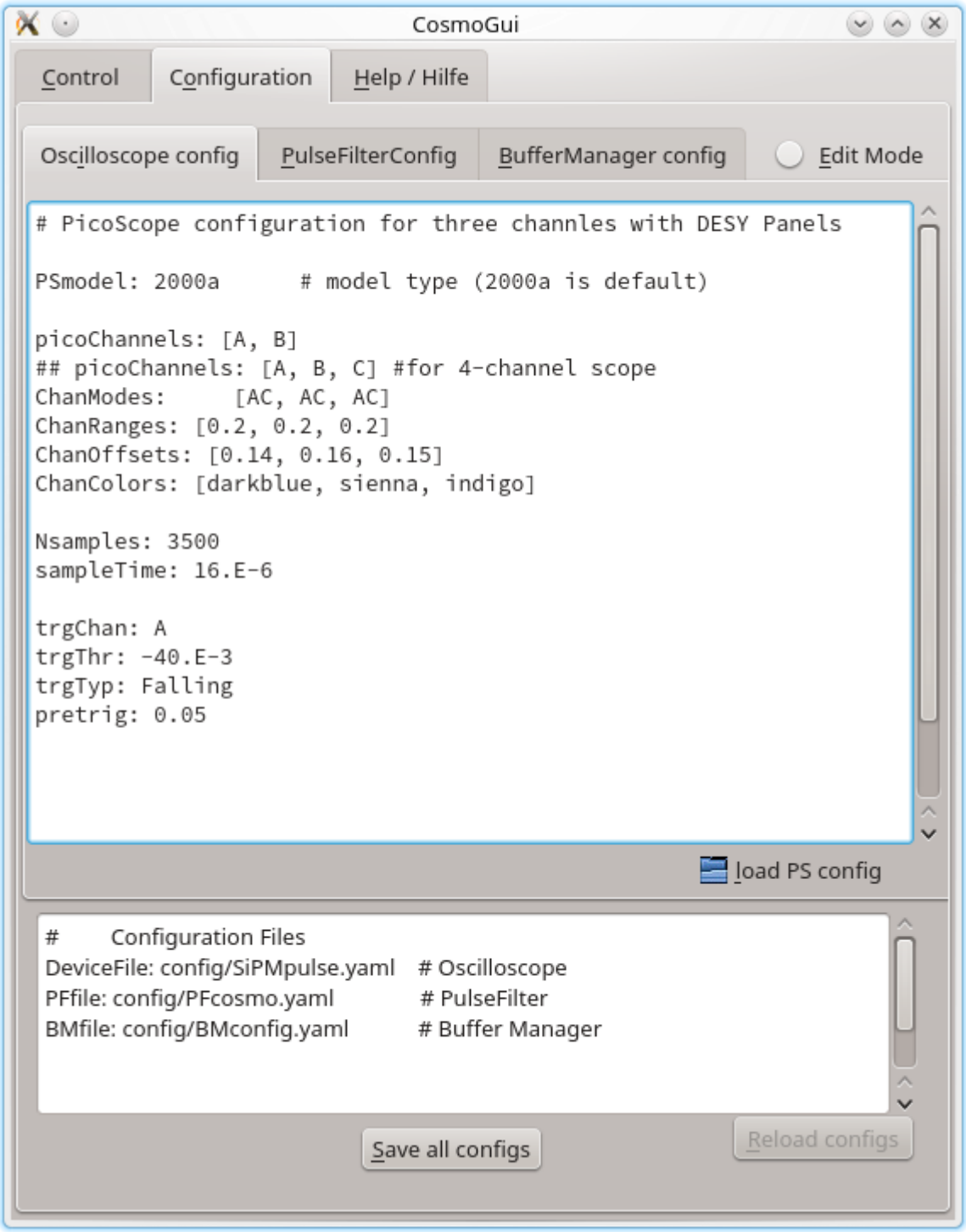
Der Code kann entweder auf der Linux-Kommandozeile über das Script *runCosmo.py* oder über eine grafische Oberfläche, *CosmoGui.py*, gestartet werden. Auf vielen Systemen ist ein *Icon* auf der grafischen Oberfläche vorhanden, mit dem durch anklicken das Programm gestartet werden kann.

Hier ein Bild der grafischen Oberfläche:



In den Feldern mit Dateinamen steht zunächst die Haupt-Konfigurationsdatei, in der alle weiteren Konfigurationsdateien enthalten sind, sowie das Arbeitsverzeichnis, in dem modifizierte Konfigurationen und die aufgezeichneten Daten abgelegt werden. Im Feld *Run Tag* steht ein Name, der der aktuellen Messung zugeordnet ist und aus dem die Dateinamen für Konfigurations- und Ausgabedateien und abgeleitet werden.

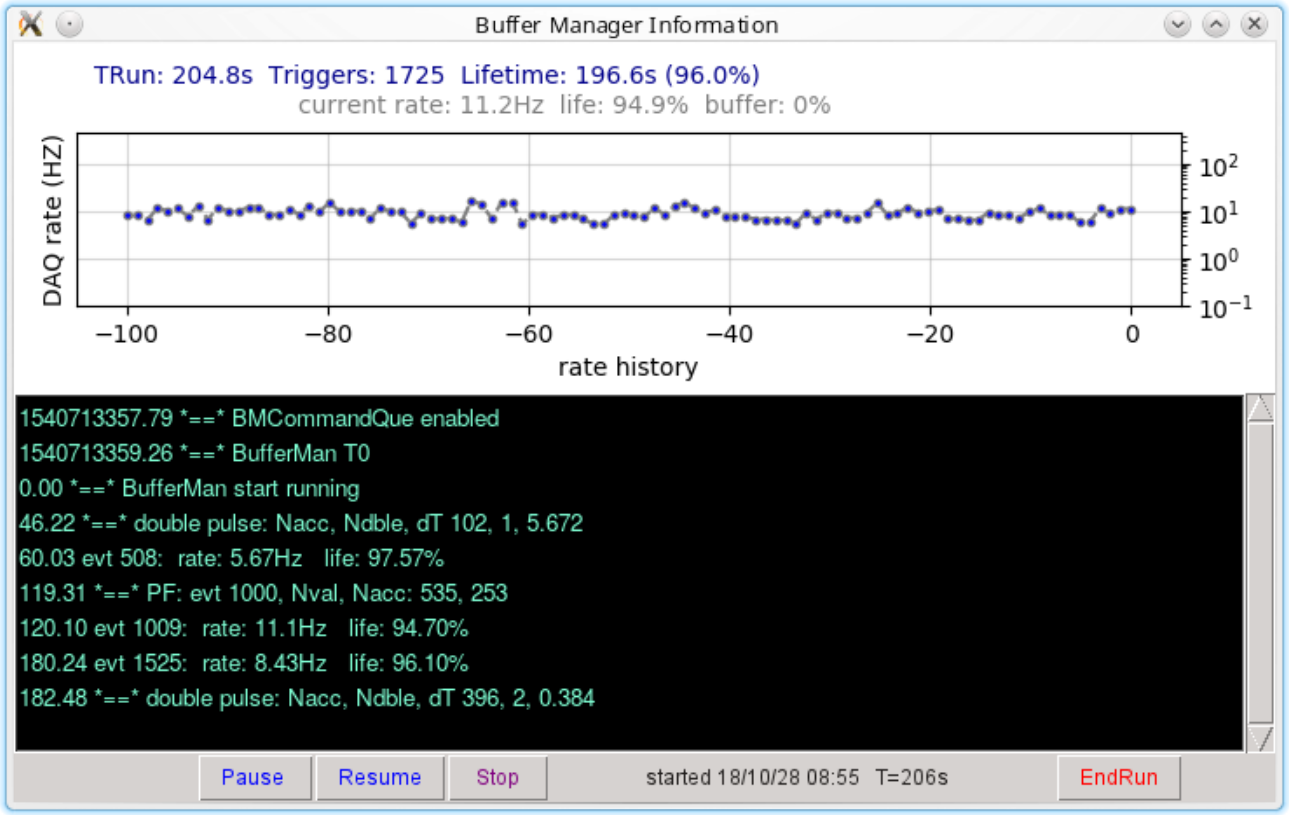
Bei Klick auf den Reiter *Configuration* öffnet sich die Anzeige der aktuellen Konfigurationsdateien:



Ganz unten im Fenster wird die Hauptkonfiguration angezeigt, die lediglich die Namen der Konfigurationsdateien für das USB-Oszilloskops, die Pulsanalyse und die Datennahme enthält. Die Konfigurationsdateien können mittels der grafischen Oberfläche ausgewählt (Klick auf das Dateisymbol) oder verändert werden (Knopf *EditMode* aktivieren). Wenn alle Konfigurationsdateien erstellt sind, können sie mit dem Feld `Save all configs` im Arbeitsverzeichnis unter den in der Hauptkonfiguration angegebenen Namen gespeichert werden. Der Name der Hauptkonfigurationsdatei ist dabei der im Feld `Run Tag` gesetzte Text mit der Erweiterung *.daq*. Vor dem Abspeichern erfolgt eine Überprüfung auf syntaktische Richtigkeit - sollte eine Fehlermeldung angezeigt werden, kann die betroffene Datei erneut modifiziert und dann die gesamte Konfiguration abgespeichert werden.

Das Starten der Datennahme erfolgt mit dem Knopf `Start Run` im ersten Reiter. Es wird ein eigenes Unterverzeichnis im Arbeitsverzeichnis erzeugt, dessen Name aus dem im Feld `Run Tag` eingetragenen Text und dem aktuelle Datum abgeleitet wird. Auch die komplette Konfiguration wird dort abgespeichert, so dass jederzeit ersichtlich ist, unter welchen Bedingungen die Daten im Verzeichnis aufgenommen wurden.

Nach dem Abspeichern beendet sich die grafische Oberfläche, und die eigentliche Datennahme (engl. "Run") beginnt mit dem Start der grafischen Oberfläche des Puffer-Managers und den in dessen Konfiguration festgelegten Echtzeitanzeigen. Die grafische Oberfläche ist hier gezeigt:



Über die Kontrollflächen des Puffer-Managers kann die Datennahme pausiert (*Pause*), wieder aufgenommen (*Resume*) oder beendet werden (*Stop* und **EndRun*). In gestopptem Zustand werden die Ausgabedateien geschlossen, aber alle Fenster bleiben noch geöffnet, so dass Grafiken betrachtet oder gespeichert und statistische Information ausgewertet werden können. Wird der Run beendet, verschwinden alle Fenster.

Das Programm wird in einem Konsolenfenster ausgeführt, in dem vielfältige Informationen zur Initialisierung, Konfiguration und zum Start einzelner, jeweils als Hintergrundprozessen ausgeführten Programmkomponenten angezeigt werden. Die Kontrolle ist auch über Eingabe einzelner Kommandos mit der Tastatur möglich, wenn das Ausgabefenster vorher durch Anklicken aktiviert wurde:

```
type -> E(nd), P(ause), S(top) or R(esume) + <ret>
```

Als einer der Datenkonsumenten des Puffermanagers startet neben den diversen Echtzeitanzeigen auch der Pulsfilter zur Echtzeit-Analyse der vom Oszilloskop ausgelesenen Daten mit den in dessen Konfiguration festgelegten Echtzeit-Anzeigen.

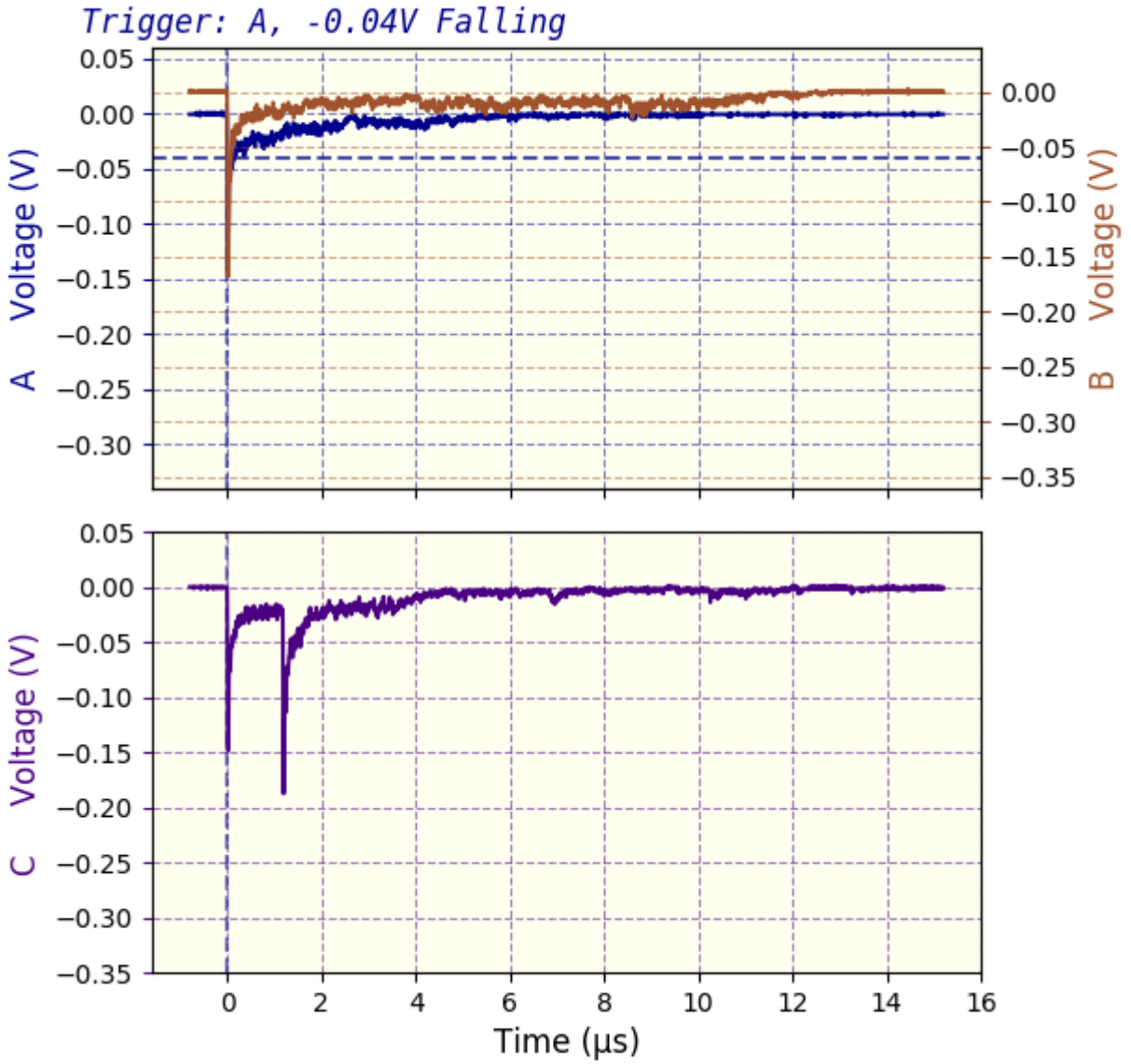
Informationen über die im Pulsfilter erkannten Signale werden laufend in Dateien auf der Festplatte abgelegt:

- Dateien mit dem Namensanfang *pFilt* enthalten Informationen zu allen aufgezeichneten Pulsen, die die Stufe der Triggervalidierung passiert haben.
- Dateien mit dem Namensanfang *dpFilt* enthalten Informationen zu den aufgezeichneten Doppelpulsen im CSV-Format:

  # Nacc, Ndble, Tau, delT(iChan), ... , V(iChan), ...

  - *Nacc* : Zahl der akzeptierten Pulse
  - *Ndble*: Zahl der akzeptierten Doppelpulse
  - *Tau*: Zeitlicher Abstand zwischen Triggerpuls und (erstem) Folgepuls (= gemessene μ-Lebensdauer)
  - *delT(iChan)* : zeitlicher Abstand des Doppelpulses vom Triggerpuls in Kanal *iChan*
  - *V(iChan)*: Pulshöhe in mV des Folgepulses in Kanal *iChan*
- Falls eingeschaltet, werden auch die vollständigen Rohdaten von allen erkannten Doppelpulsen in der Dateien mit Namensbeginn *dpRaw* im *.yaml*-Format abgelegt. Damit können eigene Analysen der Rohdaten ausgeführt werden.

Es ist auch möglich, grafische Darstellungen von Doppelpulsen im Verzeichnis mit Namensbeginn *dpFig* abzulegen. Ein Beispiel für einen Doppelpuls im dritten (untersten) Panel bei einer Datennahme mit drei Cosmo-Panels ist hier gezeigt:

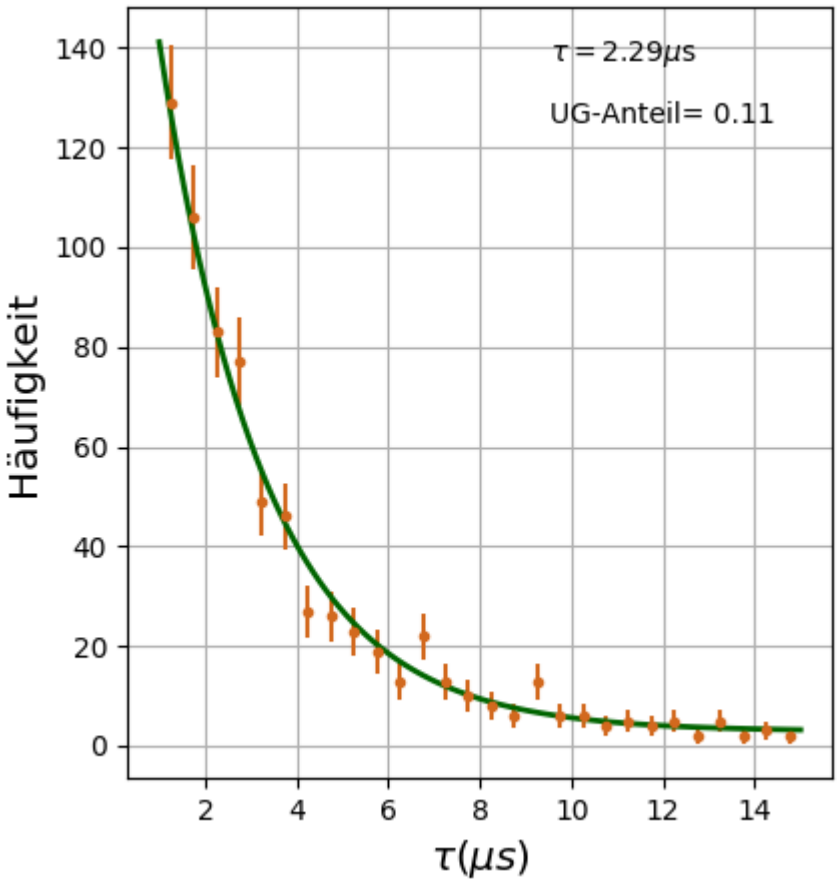


Zwei Hilfsanwendungen, *plotDoublePulses.py* und *makeFigs.py* ermöglichen das Einlesen der abgespeicherten Pulsformen und deren graphische Anzeige bzw. Abspeichern als Grafikdateien im *.png*-Format.

Eine weitere Hilfsanwendung, *fit_dpData.py*, führt die Anpassung einer Exponentialfunktion an die in Dateien mit Namensanfang *dpFilt* abgelegten individuellen µ-Lebensdauern. Zum Start einer Anpassung an die in der Datei dpFilt.dat abgelegten Daten im Bereich von 1.0 bis 15µs folgenden Befehl auf der Kommandozeile eingeben:

```
./fit_dpData.py dpFilt<Name> 1.0 15.
```

Ein typisches Ergebnis mit etwa 700 mit den Cosmo-Panels aufgezeichneten Doppelpulsen im Bereich von 1µs - 15µs ist hier gezeigt:



##Details zu Konfiguration

Die Konfigurationsdateien für das USB-Oszilloskop, den Puffer-Manager und die Signalanalyse sind in jeweils einer Datei vom Typ *.yaml* im Unterverzeichnis *./config/* festgelegt. Die Dateinamen sind in Dateien vom Typ *.daq* enthalten, also `Kanne.daq` für Kamiokanne and *Cosmo.daq* für die CosMO-Panels.

Die folgenden Beispiele gelten für den Kamiokanne-Detektor. Generell entspricht die in den Konfigurationsdateien verwendete Syntax der Markup-Sprache *yaml*. Insbesondere kennzeichnet Text nach einem `#` -Zeichen erklärende Kommentare oder enthält alternative, auskommentierte Konfigurationsoptionen, die durchLöschen des `#` -Zeichens aktiviert werden können.

Inhalt der Datei Kanne.daq:

```
# file Kanne.daq
# --------------------
# Konfigurationsdateien für den Kamiokanne-Detektor

DeviceFile: config/PMpulse.yaml   # Konfiguration des Oszilloskops
BMfile:     config/BMconfig.yaml  # Konfiguration des Puffer-Managers
PFfile:     config/PFconfig.yaml  # Konfiguration des Pulsfilters
```

Die Oszilloskop-Konfiguration enthält Informationen zum Typ des Oszilloskops, die aktiven Kanäle und zum Trigger.

Inhalt der Datei PMpulse.yaml:

```
# file PMpulse.yaml
# -----------------
# Konfigurationsdatei für PicoScope an Photoroehre

PSmodel: 2000a      # Modeltyp (2000a ist voreingestellt)

picoChannels:      [A]         # aktiver Kanal, [A,B] aktiviert beide Kanaele
ChanRanges:        [0.5, 0.2]  # Messbereich
ChanOffsets:       [0.4, 0.45] # analoger Offset, der vor Anzeige addiert wird.

sampleTime:   16.E-6 # Zeit zwischen zwei Messpunkten in s
     # Zahl im wissenschaftlichen Format mit '.' und Exponent mit Vorzeichen
Nsamples:     3500   # Anzahl der aufzunehmenden Messpunkte

trgChan:    A        # Kanal, auf den der Trigger wirkt
trgThr:     -45.E-3  # Schwelle
trgTyp:     Falling  # fallend (Falling) oder ansteigend (Rising)
trgTO:      5000     # Timeout, nach dieser Zeit wird einmal ausgelesen
pretrig:    0.05     # Anteil der vor dem Trigger ausgelesenen Daten
ChanColors: [darkblue, sienna, indigo] # Farben für Darstellung der Kanäle
```

Die Konfiguration der Pulsanalyse spezifiziert die gewünschten Ausgabedateien und gibt die Pulsform und die Pulshöhe für jeden Kanal sowie die zu startenden Anzeige-Module an. Sie enthält auch die Spezifikation der Echtzeit-Histogramme für Pulshöhen, Myon-Rate und Lebensdauer. Ein Beispiel ist hier gezeigt:

Inhalt der Datei BFconfig.yaml:

```
# file PFKanne.yaml
# -------------------
# Konfigurationsdtei für den PulseFilter mit Kamiokanne

#logFile: pFilt     # speichere Angaben zu allen gefundenen Pulsen
logFile: Null      #     Null falls keine Ausgebe erwuenscht
logFile2: dpFilt   # speichere nur  Doppelpulse, Null falls nicht erwuenscht
rawFile:  rawDP    # speichere Rohdaten von Doppelpulsen, put Null if not wanted
pictFile: pictDP   # Speichere Bilder von Doppelpulsen

# Puls-Parameter
#         ______
#        /      \
#     _ /_ _ _ _ \_
#       r    on  f
#  r = rise (Anstiegszeit), on (Haltezeit), f = falling (Abfallzeit)

# Pulsformen für die aktiven Kanaele
   #      falls nur eine angegeben, gilt sie fuer all Kanaele
pulseShape:
 - pheight: -0.035   # Pulshoehe
   taur   : 20.E-9   # Anstiegszeit
   tauon  : 12.E-9   # Haltezeit
   tauf   : 128.E-9  # Abfallzeit
```

```
# Pulsform fuer Triggerpuls
#         optional - falls nicht angegeben, nutze pulseShape
trgPulseShape:
 - pheight: -0.045   # Pulshoehe
   taur   : 20.E-9   # Anstiegszeit
   tauon  : 12.E-9   # Haltezeit
   tauf   : 128.E-9  # Abfallzeit
```

```
# Anzeigen, die gestartet werden sollen
modules: [RMeter, Display, Hists]  # Rate, Pulsform, Histogramme

# Definition der Histogramme
histograms:
 # min  max Nbins ymax    title              lin/log
 - [0., 0.4, 50, 20., "noise Trg. Pulse (V)", 0]
 - [0., 0.8, 50, 15., "valid Trg. Pulse (V)", 0]
 - [0., 15.,  45, 7.5, "Tau (µs)", 1]
 - [0., 0.8, 50, 15., "Pulse Height (V)", 0]

doublePulse: True  # Doppelpulssuche ein, False falls nicht erwuenscht
```

Die Konfigurationsdatei für den Puffer-Manager muss eigentlich selten geändert werden. Sie gibt an, wie viele Puffer verwendet werden, welche Anzeige-Module gestartet werden und ob ein Log-File erstellt werden soll:

Inhalt der Datei BMconfig.yaml:

```
# file BMconfig.yaml
# ------------------
# Konfigurationsdatei des picoDAQ Puffermanagers

NBuffers: 16         # Anzahl der Puffer für aufgezeichnete Pulsformen
BMmodules: [mpOsci]  # BufferMan- Module, die gestartet werden sollen
verbose: 1           # setze Niveau der ausgegebenen Nachrichten (0, 1, 2)
LogFile: BMsum       # Schreibe log-Datei mit laufenden Angaben
```

Beispielausgabe

Das Verzeichnis *./output* enthält Ergebnisse einer Langzeitmessung (ca. 20 Tage) mit der Kanne und einer etwa eintägigen Messung mit zwei Cosmo-Panels.

Die gepackte Datei *rawDP_.dat.zip* enthält die Rohdaten der aufgezeichneten Pulsformen für erkannte Doppelpulse. Die Scripte *plotDoublePulses.py* und *makeFigs.py* erlaubt das Einlesen der gepackten Datei und die grafische Darstellung der Doppelpulse bzw. die Speicherung als Grafikdateien im *.png*-Format. Die aus den Doppelpulsen bestimmten Lebensdauern sind in der Datei *dpKanne2_180403.dat* enthalten. Eine

Anpassung einer Exponentialfunktion an gemessene Lebensdauern zwischen 1.5 µs and 15. µs kann mit dem Skript *fit_dpData.py* ausgeführt werden; das Ergebnis zeigt die Grafikdatei *life-ofMU_180403.png*.

Liste der Dateien im Projekt *picoCosmo*

- `CosmoGui.py`
  Grafische Benutzeroberfläche zum Editieren der Konfigurationsdateien
  und Starten des Skripts `runCosmo.py`
- `runCosmo.py`
  Datennahme und Anzeigen wie in den Konfigurationsdateien (Vorgabe *defaut.daq* für zwei Cosmo-Panels mit Messung der µ-Lebensdauer
- `README_de.md` bzw. `README_de.pdf`
  Deutschsprachige Beschreibung
- `Anleitung.md` bzw. `Anleitung.pdf`
  Deutschsprachige Anleitung

Module

- picocosmo/PulseFilter.py
  Analyse der vom Oszillographen gelieferten Pulsformen;
  Auslese und Anzeige mittels der Module im Projekt *picoDAQ*

Konfigurationsdateien

- `default.daq`
  Konfiguration für zwei Cosmo-Panels
- `Cosmo.daq`
  Konfiguration für zwei Cosmo-Panels
- `Kanne.daq`
  Konfiguration für eine Kanne mit Photoröhre und Pulslänge 150 ns
- `config/BMconfig.yaml`
  Konfiguration für den Puffermanager
- `SiPMpulse.yaml'
  Konfiguration des Picoscopes für SiPM-Pulse
- `PMpulse.yaml'
  Konfiguration des Picoscopes für Photomultiplier-Pulse
- `SiPMpulse2000.yaml'
  Konfiguration eines Picoscopes 2202A für SiPM-Pulse
- `PFcosmo.yaml'
  Konfiguration des PulseFilters für Cosmo-Panels
- `PFKanne.yaml'
  Konfiguration des PulseFilters für eine Kanne
- `PFcosmo2000.yaml'
  Konfiguration des PulseFilters für Cosmo-Panels an Picoscope 2202A

##Beispiele

- `output/CosmoPanels_180514`
  Beispielausgabe einer Datennahme mit den Cosmo-Panels
  - dpFilt_180514_1806.dat enthält die Daten von aufgezeichneten Doppelpulsen
  - rawDP_180403.zip enthält gepackte Rohdaten der Doppelpulse
- `output/Kanne_180403`
  Beispielausgabe einer Datennahme mit einer Kanne
  - dpKanne2_180403.dat enthält die Daten von aufgezeichneten Doppelpulsen
  - rawDP_180514.zip enthält gepackte Rohdaten der Doppelpulse
- `output/fit_dpData.py`
  pyhton-Skript zur Anpassung einer Exponentialfunktion an Daten in Dateien *dpFilt*.dat*
- `output/makeFigs.py`
  Erzeugen von Grafiken aus Dateien *dpRaw*.dat*
- `output/plotDoublePulses.py`
  Anzeigen von Doppelpulsen aus Dateien *dpRaw.dat* als Grafiken auf dem Bildschirm