

## Software zur Auslese und Analyse der Experimente des Netzwerks Teilchenwelt

### *Kurzfassung:*

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

### **Beschreibung der Software**

Das Netzwerk Teilchenwelt, <http://www.Teilchenwelt.de> stellt Experimente zum Nachweis von Myonen aus der kosmischen 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 Oszilloskopen sichtbar gemacht werden können.

Moderne USB-Oszilloskope wie das PicoScope der Firma PicoTechnology, siehe <http://www.picotech.com>, erlauben es, die Pulsformen nicht nur anzuzeigen, sondern auch in Echtzeit aufzuzeichnen und zu analysieren. Diesem Zweck dient das hier beschriebene Projekt „picoCosmo“. Es ist auf Linux-Systemen und auch auf dem Raspberry Pi lauffähig.

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 eine Mindest-Pulshöhe überschritten wird.

#### **1.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.

#### **1.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 Lebensdauer von 2,2  $\mu$ s, 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.

## Abhängigkeiten von anderen Paketen

Der hier bereit gestellte Code hängt von anderen Software-Paketen ab, die zuvor auf Ihrem System installiert sein müssen:

- das Datennahme-Paket ('Data Acquisition') *picoDAQ*, <https://github.com/Guenter-Quast/picoDAQ>
- das *python*-Interface *pico-python*, <https://github.com/colinoflynn/pico-python>
- die Gerätetreiber und C-Bibliotheken aus dem 'Pico Technology Software Development Kit', das zusammen mit der PicoScope-Software installiert wird, siehe <https://www.picotech.com/downloads>

Auch auf dem Raspberry Pi können die Treiber installiert werden, die grafische Oberfläche PicoScope ist allerdings (noch) nicht verfügbar.

## Programmausführung

Der Code kann entweder auf der Linux-Kommandozeile über das Script `runCosmo.py` oder über eine grafische Oberfläche, `CosmoGui.py`, gestartet werden.

Die benötigte Information zur Konfiguration des USB-Oszilloskops, der Pufferverwaltung zur Bereitstellung der Daten und die Pulsanalyse werden in Konfigurationsdateien im `.yaml`-Format bereit gestellt. Die für eine spezielle Konfiguration verwendeten Dateien sind in einer Datei im `.yaml`-Format mit der Endung `.daq` enthalten.

Der Start der Software auf der Kommandozeile erfolgt mit dem Befehl

```
./runCosmo xxx.daq
```

- xxx = Cosmo : Konfiguration für Cosmo-Panels
- xxx = Kanne : Konfiguration für die Kamiokanne

Die Datei enthält die Namen der Konfigurationsdateien, die für das PicoScope, den Puffer-Manager und die Pulsanalyse verwendet werden.

Auch die grafische Oberfläche kann analog mit einer Konfiguration initialisiert werden:

```
./CosmoGui xxx.daq
```

Die Konfigurationsdatei kann aber auch mittels der grafischen Oberfläche geladen und die Konfiguration editiert werden. Über die grafische Oberfläche kann ein Name für die Datennahme festgelegt werden. Alle für eine Datennahme (einen sogenannten Run) benötigten Konfigurationsdateien und die Programmausgaben werden in einem eigenen Verzeichnis abgelegt, deren Name aus dem Namen für die Datennahme und der Startzeit abgeleitet wird.

## Installation der Software

Zunächst werden alle Dateien dieses Projekts über den Link <https://github.com/GuenterQuast/picoCosmo> herunter geladen und in einem Arbeitsverzeichnis abgelegt. Vor der Verwendung müssen die folgenden Pakete installiert werden:

- PicoTech Software Development Kit:  
<https://www.picotech.com/library/oscilloscopes/picoscope-software-development-kit-sdk>.
- das pico-python Paket:  
<https://github.com/colinoflynn/pico-python>.
- das picoDAQ Paket, vers. >= 0.7.2: <https://github.com/GuenterQuast/picoDAQ>.

Zur Vereinfachung sind im Unterverzeichnis `whl/` kompatible Versionen der Module `pico-scope` aus dem Paket `pico-python` und `picodaqa` aus dem Paket `picoDAQ` als `python-wheels` enthalten, die mittels

```
pip install package-<vers\>--<tags\>.whl
```

installiert werden können.

## Details zu Konfiguration

Wie oben beschrieben, wird die Datenaufnahme und Analyse entweder über die grafische Oberfläche (`./CosmoGui.py xxx.daq`) oder über die Kommandozeile (`./runCosmo xxxx.daq`) gestartet.

Nach dem Start eines Runs startet die grafische Oberfläche des Puffer-Managers und die in der Konfiguration festgelegten Echtzeit-Anzeigen. Über die Kontrollflächen des Puffer-Managers kann die Datennahme pausiert (Pause), wieder aufgenommen (Resume) oder beendet werden kann (Stop und EndRun). In gestopptem Zustand werden die Ausgabe-dateien 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.

Eine Hilfsanwendung, `plotDoublePulses.py` ermöglicht das Einlesen der abgespeicherten Pulsformen und deren Anzeige als Grafiken. Code zum Speichern im `.png`-Format ist enthalten, aber auskommentiert.

Die Konfigurationsdateien für das USB-Oszilloskop, den Puffer-Manager und die Signalanalyse werden 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 und `Cosmo.daq` für die CosMO-Panels.

Die folgenden Beispiele gelten für den Kamiokanne-Detektor:

```
# file Kanne.daq
# -----
# configuration files for Kamiokanne

DeviceFile: config/PMpulse.yaml # Oscilloscope configuration
BMfile:     config/BMconfig.yaml # Buffer Manager configuration
PFfile:     config/PFconfig.yaml # Pulse Filter Configuration
```

Die Oszilloskop-Konfiguration enthält Informationen zum Typ des Oszilloskops, den aktiven Kanälen und zum Trigger:

```
# file PMpulse.yaml
# -----
# configuration file for PicoScope 2000 Series connected to a PM
tube

PSmodel: 2000a # model type here (2000a is default)

picoChannels: [A]
ChanRanges:   [0.5, 0.2]
ChanOffsets:  [0.4, 0.45]

sampleTime: 16.E-6 # scientific format ('.' and signed exponent)
Nsamples:   3500

trgChan:     A
trgThr:      -45.E-3
trgTyp:      Falling
trgT0:       5000 # time-out after which read-out occurs
pretrig:     0.05
ChanColors:  [darkblue, sienna, indigo]
```

Die Datei für den Puffer-Manager gibt an, wie viele Puffer verwendet werden, welche Anzeige-Module gestartet werden und ob ein Log-File erstellt werden soll:

```
# file BMconfig.yaml
# -----
# configuration of picoDAQ Buffer Manager

NBuffers: 16 # number of buffers to store raw waveforms
BMmodules: [mpOsci] # BufferMan modules to start
verbose: 1 # set verbosity level
LogFile: BMsum # write log-file entries with current statistics
```

Die Konfiguration der Pulsanalyse gibt die Pulsform, die Pulshöhe, die zu startenden An-

zeige-Module an und enthält die Spezifikation der Echtzeit-Histogramme für Pulshöhen, Myon-Rate und Lebensdauer:

```
# file PFconfig.yaml
# -----
# Configuration file for PulseFilter with Kamiokanne

#logFile: pFilt      # store all pulses, put Null if no output wanted
logFile: Null        # store all pulses, put Null if no output wanted
logFile2: dpFilt     # store double-pulses only, put Null if not
rawFile:  rawDP      # store raw wave forms, put Null if not wanted
pictFile: pictDP     # save pictures of double-pulse wave forms

# pulse parameters
#
#
#      /-----\
#     /         \
#    /           \
#   /             \
#  /               \
# /                 \
# r      on  f  f2   off  r2
#
#                f2 - r2 for bi-polar only

pulseShape:
# trigger pulse
- pheight: -0.045
  taur    : 20.E-9
  tauon   : 12.E-9
  tauf    : 128.E-9
  mode    : 0          # 0:uni-polar  1: bipolar

# other pulses (optional, if not given, use same as for trigger)
- pheight: -0.035
  taur    : 20.E-9
  tauon   : 12.E-9
  tauf    : 128.E-9

# Display Modules to be started
modules: [RMeter, Display, Hists]

# Definition of Histograms
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 # switch for double-pulse search
```

## Beispielausgabe

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

Die gepackte Datei `rawDP_<date>.dat.zip` enthält die Rohdaten der aufgezeichneten Pulsformen für erkannte Doppelpulse. Das Script `plotDoublePulses.py` erlaubt das Einlesen der entpackten Datei und die grafische Darstellung der Pulse. Die Bilder sind in der gepackten Datei `dpFigs-<date>.zip` enthalten. Die aus den Doppelpulsen bestimmten Lebensdauern sind in der Datei `dpKanne2_180403.dat` enthalten. Eine Anpassung einer Exponentialfunktion an gemessene Lebensdauern zwischen  $1.5\ \mu\text{s}$  und  $15\ \mu\text{s}$  kann mit dem Skript `fit_dpData.py` ausgeführt werden; das Ergebnis zeigt die Grafikdatei `life-ofMU_180403.png`.