Technology
Arts Sciences
TH Köln

Cologne Institute for Renewable Energy

Inbetriebnahme des Raspberry Pi als Spannungsphasenwinkel Messgerät

Masterprojekt

Erster Betreuer: Prof. Dr. Eberhard Waffenschmidt

Zweiter Betreuer: Dipl.-Wirt.-Ing. Christian Hotz

Studiengang: Erneuerbare Energien

Marius Kleinbach

Mat.-Nr.: 11151713

Köln, 15.03.2023

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Dies gilt auch für Quellen aus eigenen Arbeiten.

Ich versichere, dass ich diese Arbeit oder nicht zitierte Teile daraus vorher nicht in einem anderen Prüfungsverfahren eingereicht habe.

Mir ist bekannt, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs mittels einer Plagiatserkennungssoftware auf ungekennzeichnete Übernahme von fremdem geistigem Eigentum überprüft werden kann.

Diese Erklärung hat durch Abgabe der Arbeit auch ohne gescannte Unterschrift Gültigkeit.

Köln, 15.03.2023 gez. Marius Kleinbach

Inhaltsverzeichnis

Abbildungsverzeichnis

Die Verwendeten Abbildung stammen, wenn nicht weiter gekennzeichnet, aus eigenen Darstellungen, die im Rahmen des Masterprojekts erstellt wurden. Externe Bildquellen sind in als Fußnote angegeben. Online-Bildquellen wurden zuletzt geprüft am 13.03.2023.

Tabellenverzeichnis

1 Einleitung

Im Rahmen eines Masterprojekts des Studiengangs Erneuerbare Energien wurden Raspberry Pi Single Board Computer als Messystem zur Bestimmung des Spannungsphasenwinkel zwischen zwei Stellen im 50 Hz Stromnetz getestet. Hierfür werden zwei Geräte zeitsynchron betrieben und mittels Hardwareinterrupts die Nulldurchgänge der Netzspannung mit einem GPS-Zeitsignal verglichen. Aus der Zeitdifferenz zweier Signale lässt sich die Phasenverschiebung der Signale bestimmen, was Aufschlüsse über die Impedanz zwischen den beiden Knotenpunkten im Netz geben kann.

Das Messystem soll dazu dienen im Zuge des PROGRESSUS Projekts eine Topologieabschätzung des Ortsnetzes zu ermöglichen indem unter Berücksichtigung der Phasenwinkeln verschiedener Spannungsmesspunkte eine Knotenadmittanzmatrix aufgestellt wird [1]. Ob die dafür geforderte Genauigkeit von dem betrachteten Messverfahren mittels Hardwareinterrupts geleistet werden kann, wurde in dem Masterprojekt untersucht. Der finale Aufbau und die Herangehensweise zum Einsatz des Systems werden in diesem Bericht vorgestellt.

2 Wide Area Measurement System

Wide Area Measurement Systems (WAMS) liefern Informationen an die Leitstelle in modernen Stromnetzen, um die Beobachtbarkeit zu verbessern und Netzstabilität zu gewährleisten [2]. Ein Single Board Computer wie der Raspberry Pi kann zur Überwachung wichtiger Parameter eines Stromnetzes verwendet werden. Er kann zur Messung und Überwachung der Netzfrequenz verwendet werden, vorausgesetzt das Signal wird auf einen Spannungspegel zwischen 0 und 3,3 V transformiert, der die Signaleingänge des Geräts nicht beschädigt. Zur Messung des Spannungsphasenwinkels zwischen zwei Knotenpunkten muss ein Zeitgeber auf zwei Geräten zur exakt gleichen Zeit gestartet und gestoppt werden, sobald das jeweilige Spannungssignal den Nullpunkt durchläuft. Die Zeitdaten der einzelnen Geräte werden minütlich auf einer Cloud gesichert. Diese Zeitstempel können dann verglichen werden, um den Phasenwinkel zwischen den beiden Knotenpunkten zu bestimmen.

3 Verwendete Hardware

Die gesamte Rechenleistung wird von einem Raspberry Pi Einplatinencomputer erbracht. Das System wurde mit dem Modell 3B+ mit einem 1,4-GHz-Prozessor und 1 GB RAM und einem Modell 4B mit einem 1,5-GHz-Prozessor und 8 GB RAM getestet. In Abbildung 1 sind beide Geräte im Vergleich zu sehen. Es konnten keine Auswirkungen der Leistungsunterschiede zwischen den beiden Systemen festgestellt werden und die beiden Modelle können auf verschiedenen Knoten desselben WAMS verwendet werden. Dabei sind mindestens zwei

Messgeräte nötig, auch Phase-Measurement-Units (PMU) genannt, um den Spannungsphasenwinkel zu ermitteln.

Abbildung 1: Raspberry Pi Modell 3B+ (links)[1] und Modell 4B (rechts)[2]

Ein Raspberry-System benötigt zwei periphere Schaltungen, um als Teil des Messsystems zu arbeiten. Zur Umwandlung des 230-V-Wechselstromsignals in ein Rechtecksignal mit 3,3 V Pegel wird eine Komparatorschaltung verwendet, die von Tim Schäfer im Rahmen des PROGRESSUS-Projekts entwickelt wurde [3]. Sie ist in Abbildung 2 zu sehen

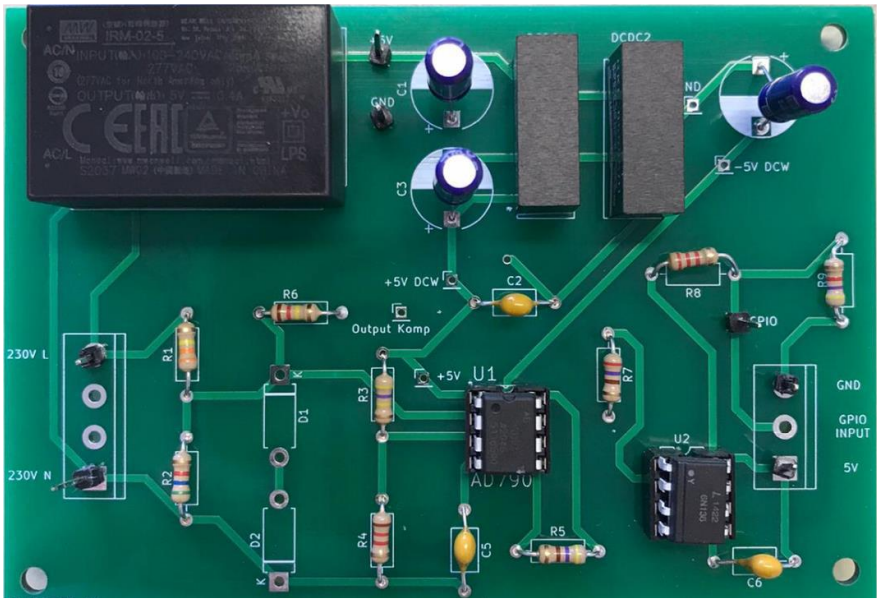

Abbildung 2: Komparatorschaltung zur Umwandlung von Wechselspannung[3]

Um sicherzustellen, dass eine Zeitmessung auf zwei verschiedenen Geräten zur gleichen Zeit gestartet werden kann, wird ein Referenzzeitsignal benötigt. Um einen identischen Zeitstempel zu erhalten, wird ein GPS-Modul (Modell GNSS 5 Click) an den Raspberry Pi angeschlossen. Diese Peripherieschaltung stellt automatisch eine Verbindung zu einem GPS-Satelliten her und empfängt ein periodisches Signal, das jede Sekunde einen Spannungsimpuls liefert, ein Pulse per Second (PPS) Signal. Jeder dieser beiden Schaltkreise ist mit einem GPIO (General

[1] Quelle: https://www.berrybase.de/raspberry-pi-3-modell-b#
[2] Quelle: https://www.distrelec.de/de/raspberry-pi-5ghz-quad-core-2gb-ram-raspberry-pi-pi4-model-2gb/p/30152780
[3] Quelle: [3]

Purpose Input/Output) des Raspberry Pi verbunden und alle drei Platinen müssen eine gemeinsame Masse haben. Zusätzlich kann der serielle Datenempfangsanschluss (RxD) des Raspberry Pi verwendet werden, um Informationen über Datum, Uhrzeit und Standort vom Datensendeanschluss (TxD) des GPS-Moduls nach dem GNRMC Standard abzufragen. Der Aufbau belegt also nur zwei bis drei GPIOs des Raspberry Pi, was viel Platz für zusätzliche Peripheriegeräte lässt. Der Aufbau und die Verschaltung eines PMU ist in Abbildung 3 dargestellt.

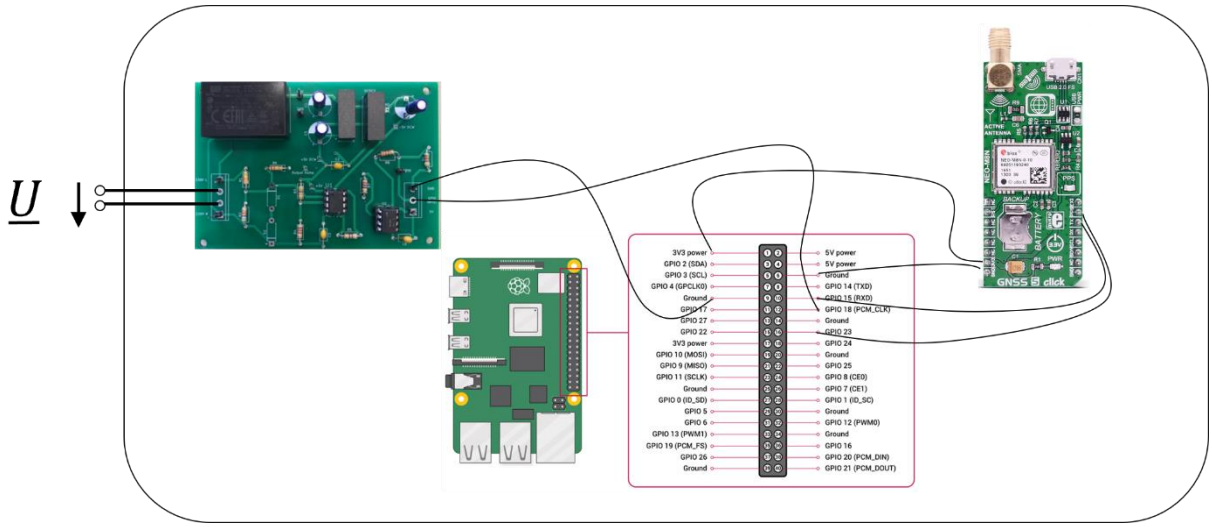

Abbildung 3: Aufbau einer Phase Measurement Unit (PMU)[4]

4 Softwarestruktur

Um die Phasenverschiebung zwischen zwei Netzknoten zu bestimmen, müssen die Ergebnisse der Zeitmessungen zweier Geräte gesammelt und auf einem Computer zur Auswertung verglichen werden. Die Auswertung kann entweder von einem externen Gerät erfolgen oder an einem der Raspberry Pi Geräte selbst durchgeführt werden.

Die Bestimmung des Spannungsphasenwinkels läuft im Wesentlichen in drei Schritten ab. Der Erfassung der Zeitdifferenzen zwischen voller Sekunde und Nulldurchgang des Spannungssignals, der Übertragung der Messwerte und schließlich einem Vergleich der Zeitstempel unterschiedlicher Messpunkte, um den Spannungsphasenwinkel zu errechnen. Die Entwickelte Software zur Durführung der einzelnen Schritte wird in den folgenden Unterkapiteln vorgestellt. Der vollständige Quellcode ist im Anhang des Berichts zu finden.

[4] Quelle: Eigene Darstellung basierend auf Bildern von [3], https://personalpages.hs-kempten.de/~vollratj/Projekte/SenseHAT/web_report.html und https://www.mikroe.com/gnss-5-click

4.1 C-Code zur Erfassung der Zeitstempel

Die Erfassung der Zeitdifferenz zwischen voller Sekunde und Nulldurchgang des Messignals findet in einem C-Programm über Hardwareinterrupts statt. Die Programmiersprache C wurde dabei gewählt, da der Code hierbei hardwarenäher und um ein Vielfaches schneller ausgeführt wird als bei benutzerfreundlicheren Programmiersprachen wie Python [4].

Der entwickelte Programmcode basiert auf der von Martin Gut entwickelten Software zur Spannungsnulldurchgangdetektion [5]. Das Programm wurde jedoch in wesentlichen Punkten angepasst und optimiert, um die Weiterverarbeitung der Messwerte zu erleichtern und allen voran zeitliche Verzögerungen zwischen dem Auftreten eines Nulldurchgangs und der Erfassung eines Zeitstempels zu minimieren, da diese die Messergebnisse verfälschen. Die wesentlichen Unterschiede sind in Tabelle 1 zu sehen.

Zeitstempel v1.4	**Zeitstempel v2.5**
Messwerte in einer Gesamtdatei -> Zugriff nach Programmdurchlauf	Mehrere Einzeldateien -> dynamischer Zugriff auf Dateien
Auswertung manuell mit C-Programm	Auswertung automatisiert möglich über DB
Messungen immer im Abstand von 1 Minute	Abstand in vollen Minuten einstellbar
5 Messwerte jede Minute nach Messstart	Jeweils 5 Messwerte zur *vollen* und *halben* Minute
Werte direkt nach Ausreißern gefiltert	Rohe Messdaten zur weiteren Verarbeitung
Hardwareinterrupts, um **im Hauptprogramm Zeitstempel** aufzunehmen	**Zeitstempel direkt in Hardwareinterrupt** aufgenommen

Tabelle 1: Anpassungen des C-Codes

Um Veränderungen der Messsignale zu erfassen, werden zwei Interrupt Routinen eingerichtet. Eine wird dabei durch eine fallende Flanke des Rechtecksignals der Komparatorschaltung ausgelöst und die andere durch eine steigende Flanke des PPS-Signals des GPS-Moduls. Zu jeder vollen Minute wird ein Messzyklus gestartet. Hierbei wird auf das erste sekündliche Signal des GPS-Sensors gewartet und ein Zeitstempel aufgenommen. Dieser dient als Referenz für die nachfolgenden Messungen des Nulldurchgangs des Spannungssignals. Bei jedem fallenden Nulldurchgang wird in der ersten Interrupt Routine ein weiterer Zeitstempel aufgenommen.

Die Zeitunterschiede zwischen dem PPS-Signal und dem Nulldurchgang ergeben die Rohdaten eines Geräts für die spätere Auswertung. Zu jeder vollen und halben Minute werden ein PPS-Signal und vier Nulldurgänge gemessen, um durch das Messverfahren bedingte Schwankungen über Mittelwertbildung minimieren zu können. Da ein 50 Hz Signal gemessen wird könnten theoretisch bis zu 50 Messungen pro Sekunde aufgenommen werden und das Ganze

zu jeder vollen Sekunde wiederholt werden, falls diese Ansprüche an das WAMS gesetzt werden sollten.

Die resultierenden Zeitdifferenzen für jede Minute werden in einer CSV-Datei auf der SD-Karte des jeweiligen Raspberry Pi gespeichert. Bei 5 Zeiterfassungen alle halbe Minute ergibt das 8 Messwerte pro Datei, da die erste Zeiterfassung stets als Referenz dient.

Die Namenskonvention ist dabei: Datum(JJJJMMTT)-Uhrzeit(hhmmss) gefolgt von der Bezeichnung „PMU“, der Gerätenummer und der Abkürzung „MW“ für Messwert. Die Gerätenummer wird im C-Programm als die Variable „deviceNumber“ vergeben. In den durchgeführten Labormessungen wurden jeweils nur zwei Geräte verwendet mit den Nummern 1 und 2. Ein vollständiger Dateiname lautet beispielsweise *20220628-113700_PMU1-MW.csv*. Hierbei handelt es sich um die Messwerte des Geräts 1 die am 28.06.2022 um 11:37 Uhr aufgenommen wurden.

Neben der Einstellung der Gerätenummer ist bei unterschiedlichen Geräten darauf zu achten, welcher Port für die serielle Kommunikation über das UART-Protokoll verwendet wird. UART steht für *Universal Asynchronous Receiver Transmitter* und nach diesem Standard werden die Daten des GPS-Moduls gesendet und vom Messgerät empfangen. Raspberry Pi der 3. Generation verwenden hierfür die Adresse */dev/ttyS0*, bei Raspberry Pi der 4. Generation wird die serielle Schnittstelle über */dev/ttyAMA0* geöffnet. Der in Anhang 10.1 zu findende Code berücksichtigt dies. Wenn die boolesche Variable „isPi3“ auf *true* gesetzt ist, wird von einem Raspberry Pi der 3. Generation ausgegangen, ist der Wert auf *false* wird der Port für ein Raspberry Pi 4 verwendet.

Weitere Benutzereingaben, die an dieser Stelle zu Beginn der main() Schleife vorgenommen werden können sind die integer Variablen „time_gap“ und „no_meas“. Hiermit kann eingestellt werden, in welchem Minutenabstand ein Messzyklus gestartet werden soll und ob das System nach einer definierten Anzahl an Messungen selbstständig stoppt.

4.2 Python-Skript zum Dateiupload

Um den Spannungsphasenwinkel zwischen zwei Messstellen zu bestimmen, müssen die Zeitstempelmessungen, die auf jedem Gerät als CSV-Dateien generiert werden, an einer zentralen Stelle gesammelt werden. Hierzu wurde für die Testmessungen eine cloudbasierte Lösung gewählt. Das ermöglicht eine Auswertung von einem beliebigen Computer mit Zugang zu diesem Cloud Konto. Aus verschiedenen Alternativen wurde der Anbieter DropBox gewählt, da sich dieses System leicht über Python Code ansprechen lässt.

Zum Upload der CSV-Dateien dient ein kurzes Python-Skript, das bei Aufruf sämtliche Dateien im Ordner der Messwerte in die DropBox hochlädt und anschließend die lokalen Dateien auf der SD-Karte des Raspberry Pi löscht. Dadurch gibt es kein Risiko, dass Langzeitmessungen

aufgrund einer überfüllten Speicherkarte abgebrochen werden. Das besagte Python-Programm ist in Anhang 10.3 zu finden. Am Ende jedes Messzyklus wird im C-Programm das Python-Skript zum Upload über einen Kommandozeilenbefehlt aufgerufen. Das ermöglicht eine Auswertung der ersten Messergebnisse, ohne die laufende Messung zu beeinträchtigen.

4.3 Python-Skript zur Auswertung der Messergebnisse

Um einen großen Satz an Messdaten auszuwerten wird ein Python-Skript eingesetzt, das die Zeitstempel der Nulldurchgänge zweier Geräte zur gleichen Messsekunde miteinander vergleicht. Die Auswertungssoftware wurde von Christian Hotz entwickelt und im Rahmen des Masterprojekts um einen Filteralgorithmus erweitert. Die schlussendlich eingesetzte Software ist in Anhang 10.4 zu sehen.

In folgenden 4 Stufen kann die Nachbereitung der Messwerte über die Variable „filter_level“ eingestellt werden:

0. Keine Filterung.
1. Ergebnisse, bei denen ein Nulldurchgang vor dem Beginn der Messung liegt und einer danach werden um 360° verschoben (z.B. -355° wird zu -5°).
2. Glättung der Messwert (Rauschunterdrückung) indem jeder Messwert durch den Mittelwert der benachbarten Messwerte ersetzt wird. Wie groß dieses Fenster zur Glättung ist, wird mit der Variable „filter_window“ festgelegt.
3. Der erste Messpunkt zu Beginn eines Messzyklus wird nicht in der Auswertung mitberücksichtigt, da diese vergleichsweise stark vom Erwartungswert abweichen.
4. Der Mittelwert der gefilterten Messergebnisse wird zusätzlich im Graphen dargestellt.

4.4 Python-Skript zur Auswertung der Messgenauigkeit

Um zu überprüfen, wie präzise das vorgestellte Messverfahren unter idealen Bedingungen sein kann, wurden mit einem definierten Rechtecksignal aus einem Funktionsgenerator Vergleichsmessungen durchgeführt. Ziel dieser Untersuchung ist es, die Standardabweichung bei der Messung von zwei aufeinanderfolgenden Spannungsflanken zu ermitteln, deren zeitlicher Versatz bekannt ist. Damit kann bewertet werden, ob ein Spannungsphasenwinkel unter Idealbedingungen mit ausreichender Genauigkeit gemessen werden kann. Der genaue Aufbau ist in Kapitel 5 beschrieben. Der dafür angepasste C-Code ist in Anhang 10.2 zu sehen.

Zur Auswertung wurde auch hierfür ein Python-Skript genutzt. Der zugrunde liegende Code zum Einlesen der CSV-Dateien wurde ebenfalls wie die in Kapitel 4.3 beschrieben Software von Christian Hotz entwickelt und um eine Datenauswertung erweitert. Je nachdem welche Darstellung der Ergebnisse gewünscht ist kann zwischen 8 Einstellungen über die Variable „analyze_setting“ gewählt werden:

1. Darstellung als Zeitreihe: Alle Ausreißer, die einem Wert des Winkels > 10° entsprechen würden gelöscht
2. Darstellung als Zeitreihe: Glättung über Fenstergröße mittels Filteralgorithmus
3. Darstellung als Zeitreihe: Darstellung der gefilterten und ungefilterten Messwerte im selben Diagramm
4. Darstellung als Zeitreihe: Betrachtung der Zeit Δt, anstatt des hypothetischen Winkels
5. Darstellung als Histogramm: Erster Messpunkt jedes Messzyklus nicht berücksichtigt
6. Darstellung als Histogramm: Absolute Abweichung vom Erwartungswert
7. Darstellung als Histogramm: Absolute Werte abzüglich Offset des Mittelwerts
8. Darstellung als Zeitreihe: Darstellung der gefilterten und ungefilterten Messwerte im selben Diagramm über den gesamten Zeitbereich

Das entsprechende Python-Skript ist in Anhang 10.5 beigefügt.

5 Aufbau zur Überprüfung der Messgenauigkeit eines Geräts

Die Herausforderung bei der Entwicklung der Spannungsphasenwinkelmessungen über Hardware-Interrupts eines Raspberry Pi Single Board Computern ist es einen Zeitstempel unmittelbar nach dem zu messenden Ereignis aufzunehmen. In einem 50 Hz Netz verfälscht jede zeitliche Abweichung Δt zwischen Spannungsflanke am Messsignal und Aufnahme der Zeitmessung im C-Code den ermittelten Phasenwinkel φ nach Formel (5.1.

$$\Delta\varphi = 360° \cdot 50\,Hz \cdot \Delta t = \frac{360°}{20000\mu s}\Delta t = 0{,}018\frac{°}{\mu s}\,\Delta t \qquad (5.1)$$

Ziel der Optimierungen am Programmcode war es diese Zeitabweichungen so zu reduzieren, dass der Spannungsphasenwinkel bis auf ± 0,3° genau gemessen werden kann, um eine verlässliche Knotenadmittanzmatrix für die Topologieabschätzung zu ermöglichen [1]. Aus Formel (5.1 folgt, dass die Zeitmessungen für diese Genauigkeit um ± 16,67 µs schwanken dürfen. Eine Standardabweichung der Messwerte um 16,67 µs ist also gerade noch akzeptabel.

Raspberry Pi Computer arbeiten in Regel auf einem Linux basierten Betriebssystem. In diesem Projekt wurde mit dem Raspbian Betriebssystem mit grafischer Benutzeroberfläche gearbeitet. Da die Geräte nicht echtzeitfähig sind und Hintergrundprozesse die Ausführung von Programmcode verzögern können, ist es nicht möglich zu garantieren, dass ein Codeabschnitt nach einem Hardware-Interrupt in einem festen Zeitfenster ausgeführt wird [6].

Um zu überprüfen ob die zeitlichen Abweichungen mit einem minimalen Messaufbau und ohne zusätzliche Prozesse auf dem Raspberry Pi ausreicht um erforderliche Messgenauigkeit zu erzielen, wurden die Interrupt Routinen an einem Funktionsgenerator getestet. Um mögliche Zeitverzögerungen durch die Komparatorschaltung oder den GPS-Empfänger auszuschließen, wurden die Interrupt Pins direkt an das 50 Hz Rechtecksignal eines Rohde & Schwarz

HM8150 Funktionsgenerators angeschlossen. Die Signalgüte wurde gleichzeitig mit einem Agilent DSO1024A Oszilloskops überwacht. Der gesamte Messaufbau ist in Abbildung 4 zu sehen.

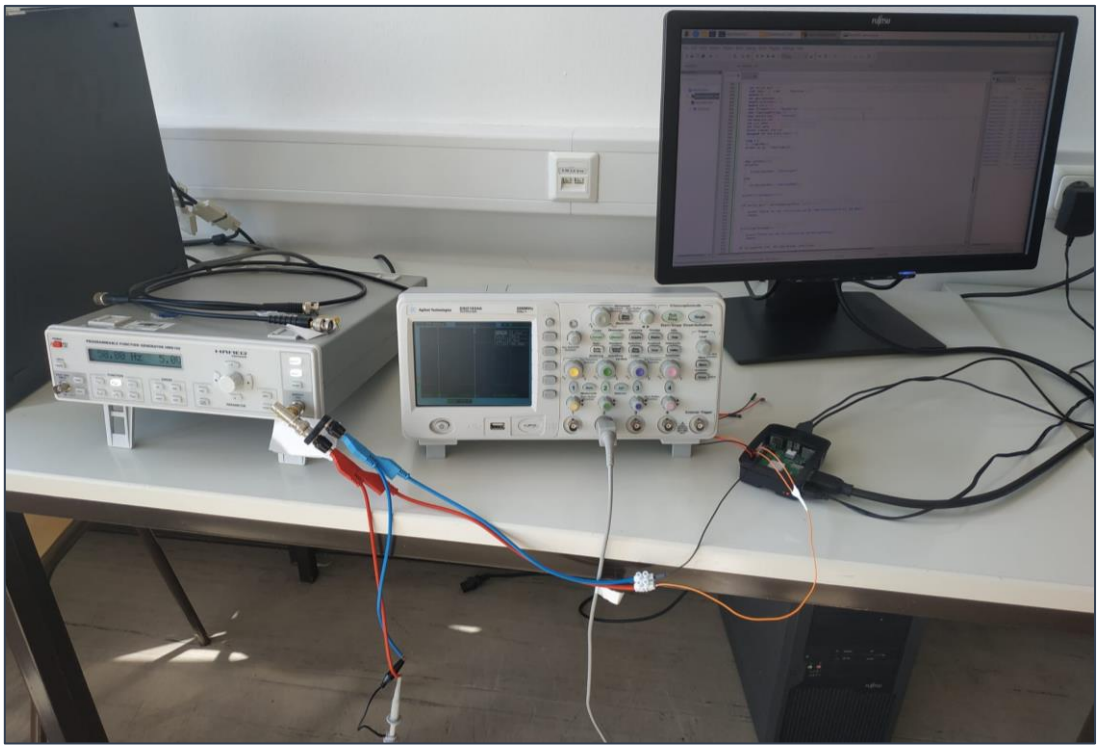

Abbildung 4: Aufbau zur Überprüfung der Interrupt-Genauigkeit

Das gleiche Rechtecksignal wird hierbei in an beiden Interrupt Eingängen des Raspberry Pi angelegt und die Software in Anhang 10.2 so angepasst, dass ein Eingang auf eine steigende und der andere auf eine fallende Spannungsflanke reagiert. Das System sollte daher immer eine Zeitdifferenz von einer halben Periode messen, was bei 50 Hz einer Zeit von 10 ms entspricht. Die Ergebnisse dieser Untersuchung, die mithilfe der in Kapitel 4.3 beschriebenen Software ausgewertet wurden, werden in Kapitel 7 betrachtet.

6 Aufbau zur Spannungsphasenwinkelmessung

Um zu prüfen, ob das Messverfahren geeignet ist, um den Spannungsphasenwinkel zwischen verschiedenen Spannungsknoten zu bestimmen, wird ein Messaufbau mit einer definierten komplexen Impedanz verwendet. Die Schaltwand im Smart-Grid Labor der Technischen Hochschule Köln bietet hierzu die Möglichkeit verschiedene veränderbare Widerstände und Kondensatoren mit einer Kapazität von bis zu 82 µF zu verschalten. Ein PMU bestehend aus Raspberry Pi, Komparatorschaltung und GPS-Modul wird dabei direkt an die 230 V Netzspannung angeschlossen und ein weiteres PMU mit eigener Komparatorschaltung hinter einer Kapazität parallel zu einem Lastwiderstand. Der vereinfachte Schaltplan für den Messaufbau ist in Abbildung 5 zu sehen.

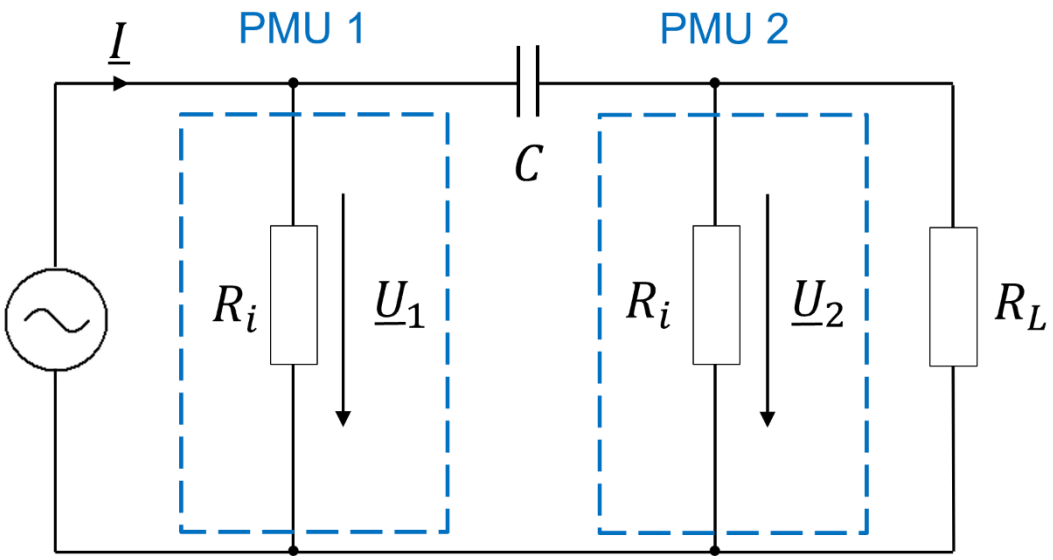

Abbildung 5: Schaltplan zur Spannungsphasenwinkelmessung

Mit einem Lastwiderstand R_L = 750 Ω, einer Kapazität von C = 82 µF und einem Innenwiderstand der PMU am GPIO-Pin von 335,6 kΩ ergibt sich bei einer 50 Hz Wechselspannung ein zu erwartender Spannungsphasenwinkel von φ = 2,963°. In verschiedenen Messreihen wurde die Kapazität teilweise auf 40 µF verringert um die Ergebnisse bei einem Erwartungswert von φ = 6,057° zu beobachten. Die Ergebnisse der Messreihen werden in Kapitel 7 diskutiert.

Die Versorgung der Raspberry Pi Geräte und des GPS-Moduls mit 5 V übernimmt ein externe Spannungsquelle. Der gesamte Messaufbau ist in Abbildung 6 zu sehen.

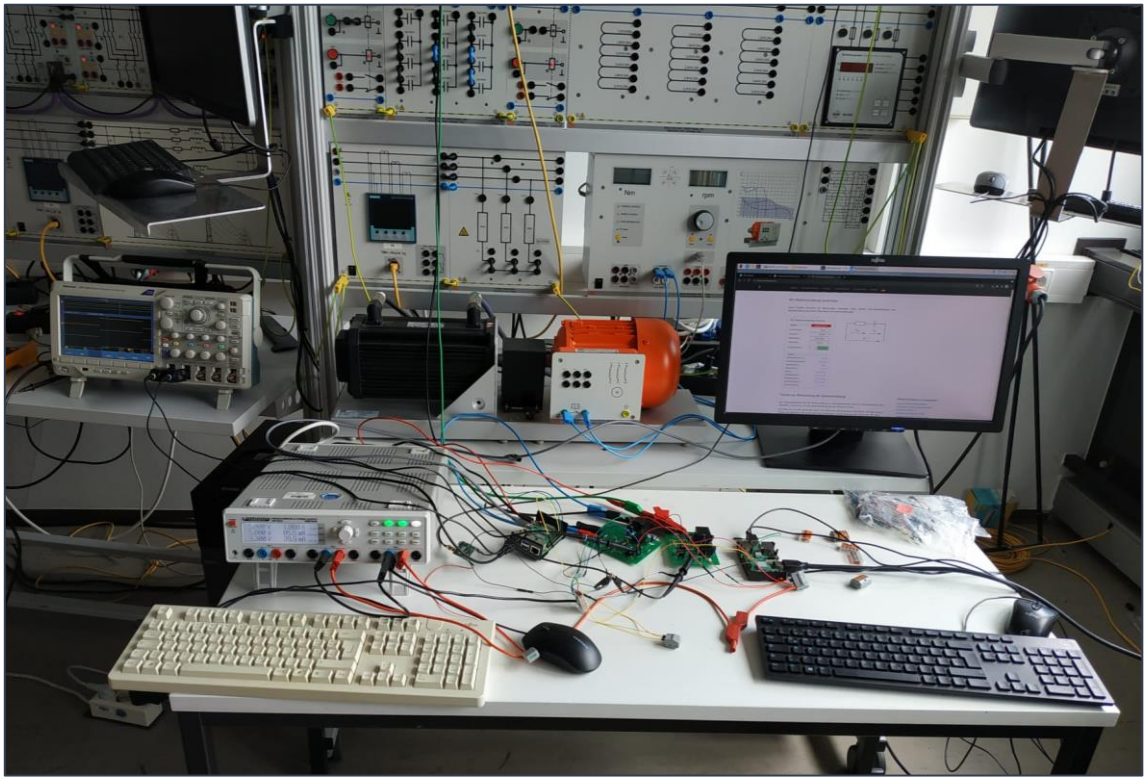

Abbildung 6: Aufbau zur Spannungsphasenwinkelmessung

7 Messergebnisse

Das vorgestellte Messsystem wurde in zwei Konfigurationen getestet. Zum einen wie in Kapitel 5 um die Messgenauigkeit der Hardwareinterrupts zu überprüfen und zum anderen wie in Kapitel 6 beschrieben, um mit zwei Geräten an einer definierten Impedanz den Spannungsphasenwinkel zu ermitteln. Zunächst wird auf die Ergebnisse der Genauigkeitsmessungen am Funktionsgenerator eingegangen.

In einer Messung über mehrere Stunden wurde ein Datensatz aus 4760 Zeitwerte aufgenommen, deren Abweichung vom Erwartungswert in Abbildung 7 in einem Histogramm dargestellt ist.

Auffallend ist, dass die Zeitmessungen nicht um den Nullpunkt schwanken, sondern dass die Zeitmessungen im Durchschnitt 61,99 µs verzögert aufgenommen werden. Ob dieser Offset in unterschiedlichen Szenarien immer konstant ist, muss weiter überprüft werden, um das Messverfahren zuverlässig einsetzen zu können. Eine konstante Abweichung durch eine Verzögerung beim Starten der ersten Messung kann aus den Ergebnissen rausgerechnet werden.

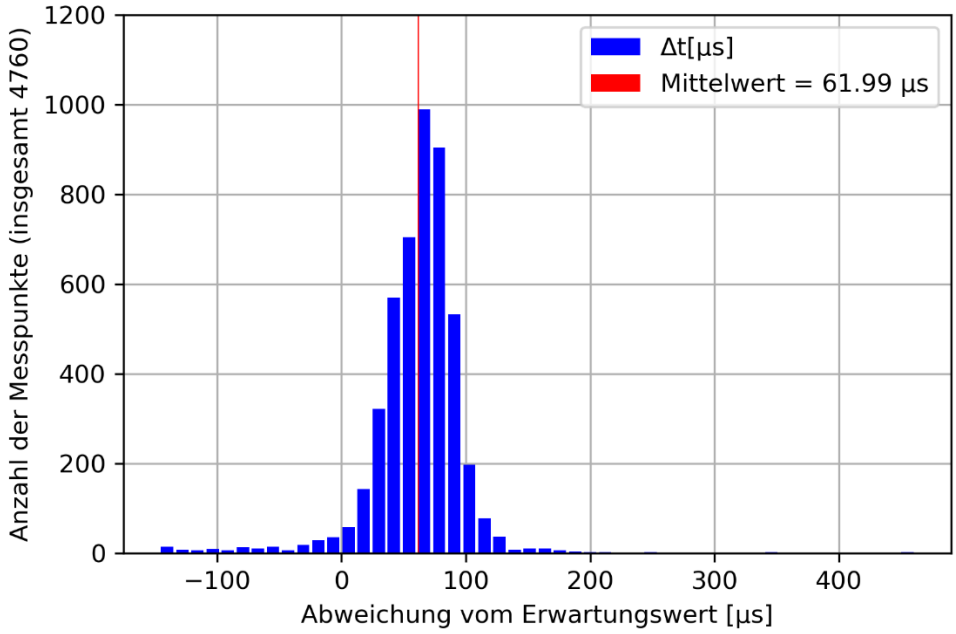

Abbildung 7: Zeitabweichungen der Rohdaten

Die Standardabweichung um den Mittelwert liegt bei σ = 21,39 µs. Wie in Kapitel 5 beschrieben wird eine Genauigkeit von ± 16,67 µs angestrebt, um für den vorgeschlagenen Einsatzzweck zielführende Ergebnisse zu liefern. Da die Abweichungen annähernd symmetrisch in einer Gauß-Verteilung um den Mittelwert streuen, kann bei einer Ausreichenden Menge an Messwerten eine Filterung zur Glättung eingesetzt werden, wie es in Kapitel 4.3 beschrieben ist.

Wie sich eine Filterung mit der Fenstergröße 10 auswirkt, ist in Abbildung 8 zu sehen. Dies bedeutet, dass jeder Messwert durch den Mittelwert der umliegenden 10 Datenpunkte ersetzt

wird. Der erste Zeitstempel jedes Messzyklus wurde hierbei ignoriert, wodurch der eben beschriebene Offset von rund 62 µs nicht mehr ins Gewicht fällt.

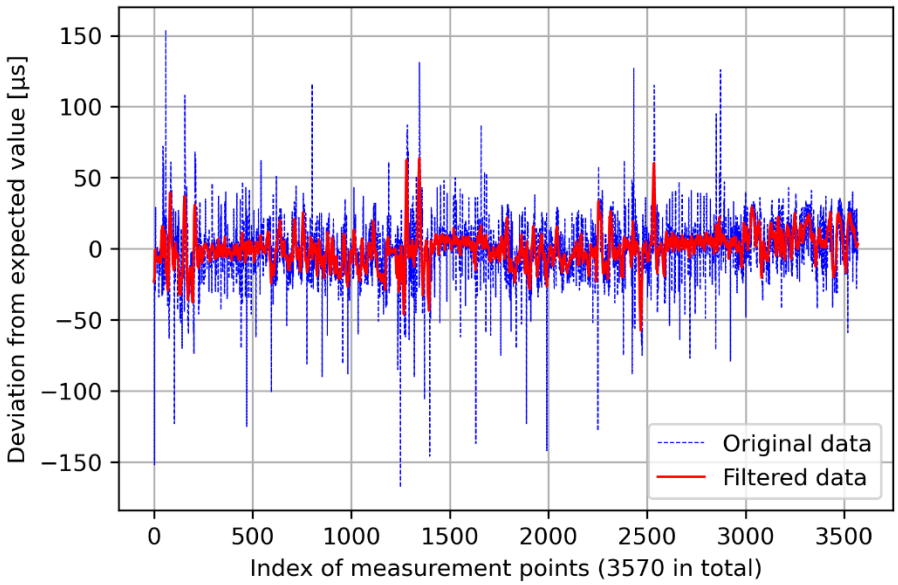

Abbildung 8: Auswirkungen der Filterung mit einer Fenstergröße von 10

Durch diese softwareseitige Rauschunterdrückung werden einzelne Ausreißer kompensiert und die Standardabweichung reduziert sich auf σ = 11,7 µs. Eine größere Fenstergröße führt zwar zu einem gleichmäßigeren Verlauf, jedoch werden dadurch unter Umständen wichtige Netzereignisse weggefiltert. Sie sollte demnach so groß wie nötig und so klein wie möglich gewählt werden, um den Anforderungen an die Genauigkeit zu genügen. Außerdem ist ein ausreichend großer Satz an Messdaten nötig, um mit einer starken Filterung aussagekräftige Ergebnisse zu liefern.

Beim Einsatz des Messsystems bestehend aus zwei Raspberry Pi Computern wie in Kapitel 6 beschrieben konnten die Genauigkeiten nur bedingt reproduziert werden. Es wurden Messungen mit je 78 Datenpunkten aufgenommen, um die Effekte verschiedener Impedanzen zu überprüfen. Wie beschrieben wurde eine Schaltung nach Abbildung 5 mit einem zu erwartenden Spannungsphasenwinkel von φ = 2,963° und eine Schaltung mit φ = 6,057° getestet. Die Abweichungen der Mittelwerte der Gesamtmessungen betrugen im ersten Fall zwischen 0,250° und 0,313° und beim zweiten Aufbau 1,467°.

Die Ergebnisse scheinen nicht wie zuerst vermutet um einen festen Zeitwert verzögert gemessen zu werden, sondern es ist möglich, dass dieser Offset arbeitspunktabhängig ist. Zudem kann die unterschiedliche Taktrate der verwendeten Single-Board-Computer Raspberry 3B+ und 4B eine unterschiedliche Reaktionszeit auf die Hardwareinterrupts zur Folge haben. Schlussendlich können die Abweichungen auch auf Bauteiltoleranzen und Kabelimpedanzen zurückzuführen sein, da die genaue Impedanz des verwendeten Messaufbau nicht im Vorhinein nachgemessen werden konnte.

8 Fazit

Das vorgestellte Wide Area Measurement System bietet eine kostengünstige Möglichkeit den Spannungsphasenwinkel eines Spannungsnetzes zu messen ohne großen Hardwareaufwand. Voraussetzung hierfür ist, dass jede Phase Measurement Unit eine Verbindung zum Internet hat, damit die Messwerte zur Auswertung ausgetauscht werden können.

Für den geplanten Einsatzzweck reicht die Genauigkeit der Messungen über Interrupt-Routinen eines Raspberry Pi jedoch nur bedingt aus. Verlässliche Messwerte können nur gemittelt über eine Messreihe aufgenommen werden, da ein signifikantes Rauschen herausgefiltert werden muss. Zudem gibt es bei der Aufnahme der Zeitstempel zwischen verschiedenen Interrupts eine Verzögerung, die weiter untersucht werden muss, um zuverlässig kompensiert zu werden. Darüber hinaus ist zu beachten, dass für jede Messung ausschließlich der entsprechende C-Code auf dem Gerät ausgeführt wurde. Wenn jede PMU noch weitere Aufgaben parallel ausführt, kann dies zu Verzögerungen bei der Verarbeitung der Interrupts führen, da die Geräte nicht echtzeitfähig sind. In diesem Fall kann der Linux Befehl „nice“ angewandt werden, um die Ausführung des zeitkritischen Codes zu priorisieren.

Obwohl das Messsystem deutliche Limitierungen besitzt und für den Einsatz im Rahmen des PROGRESSUS Projekts zu unzuverlässige Ergebnisse liefert, kann es ohne viel Aufwand für andere Anwendungsfälle eingesetzt werden. Eine mögliche Applikation ist die Durchführung von dezentralen Messungen des Netzzustands, um Regionen mit Leistungsüberschuss oder -mangel zu identifizieren, bei denen geringere Messgenauigkeiten nötig sind, um verwertbare Aussagen zu treffen [7]. Da die einzelnen PMU ihre Daten über das Internet austauschen sind sie nicht räumlich limitiert und es können Messungen des Spannungsphasenwinkel über große Distanzen ausgeführt werden.

9 Literaturverzeichnis

[1] C. Hotz, S. Baum, E. Waffenschmidt und I. Stadler, „Topology Estimation in Low Voltage Grids Using Wallbox Charging Data Recordings," Porto, 2022.

[2] A. Molina-Cabrera, M. A. Ríos, Y. Besanger, N. Hadjsaid und O. D. Montoya, „Latencies in Power Systems: A Database-Based Time-Delay Compensation for Memory Controllers," *Electronics,* Bd. 10, Nr. 2, 18 01 2021.

[3] T. Schäfer, „Entwurf einer Schaltung zur GPS-synchronisierten Spannungsphasenwinkel-messung in großflächigen, elektrischen Netzen," Technische Hochschule Köln, 2021.

[4] P. Xie, „How Slow is Python Compared to C," 13 07 2020. [Online]. Available: https://peter-jp-xie.medium.com/how-slow-is-python-compared-to-c-3795071ce82a. [Zugriff am 27 Februar 2023].

[5] M. Gut, „Softwareentwicklung eines Messsystems zur Spannungswinkelmessung auf Basis von Span-nungsnulldurchgangsdetektion," Technische Hochschule Köln, 2021.

[6] A. Carvalho, C. Machando und F. Moraes, „Raspberry Pi Performance Analysis in Real-Time Applications with the RT-Preempt Patch," Latin American Robotics Symposium (LARS), Rio Grande, Brasilien, 2019.

[7] E. Waffenschmidt, P. Littau und C. Pelikan, „Evaluation of syncrophasor use in distribution grids to estimate the regional grid state," ETG Congress, Bonn, 2017.

10 Anhang

10.1 C-Code für Hardware Interrupt Routine

```
/*
Zeitstempel.c
Dieses Programm misst die Zeitverschiebung zwischen zwei Flankensignalen
und Speichert das Ergebnis in einer .csv Datei ab.

Version 2.5 Stand 27.01.2022
Autor Marius Kleinbach
Aufruf des python Skripts zum Upload in die Dropbox wird nach jeder Mess-
routine aufgerufen
Messungen können nun im Abstand von time_gap durgeführt werden

VERSIONSGESCHICHTE
-Version 2.4 Stand 26.01.2022
Autor Marius Kleinbach
Programmstruktur angepasst: Pro Messminute wird eine eigene .csv Datei er-
stellt
-Version 2.3 Stand 25.01.2022
Autor Marius Kleinbach
Zeitdeltas abgespeichert anstatt errechneter Frequenz wenn short_data_for-
mat == true
Variabler Dateiname bei unterschiedlichen Devices (IMMER deviceNumber AN-
PASSEN)
-Version 2.2 Stand 20.01.2022
Autor Marius Kleinbach
Bugfix: pps_Cnt vor Messung zurückgesetzt
Bugfix: Flanken_Untersuchen setzt valueX[i] nun auf 1 wenn außerhalb des
Toleranzbereichs anstatt auf 0
Am besten wäre es den Messwert komplett zu ignorieren aber dies dient als
temporärer Fix um
Fehlerhafte Messwerte schnell zu erkennen
valueX[] zählt nun von 0 los, nicht von 1
Zeitstempel Nummerierung angepasst
- Version 2.1 Stand 17.01.2022
Autor Marius Kleinbach
Mess-Zeitstempel wird in direkt in Hardware Interrupt Schleife bestimmt
- Version 2.0 Stand 13.01.2022
Autor Marius Kleinbach
Zeiterfassung direkt in Hardware-Interrupt
Loggen der Zeitstempel um Schwankungen festzustellen
- Version 1.3 Stand 13.01.2022
Autor Marius Kleinbach
UART_Rx_GPS(int, char) angepasst und um Positionserfassung erweitert
Dateiname enthält nun Zeitstempel
- Version 1.2 Stand 13.12.2021
Autor Marius Kleinbach
ttyS0 Port anstatt ttAMA0 hat Einleseproblem gelöst
Programm läuft, wenn regelmäßige Spannungsflanken an EDGE_PIN registriert
werden
- Version 1.1 Stand 08.12.2021
Autor Marius Kleinbach
libwiringPi.so muss in den Debugger Einstellungen verlinkt werden
Hinzufügen von Konsolenkommentaren zum Debugging
Schrittweises deaktivieren von Kommunikationsschnittstellen
um Funktionsweise zu validieren
Zeile 77 liest immer den gleichen Wert ein, was zum Aufhängen des Programms
führt
```

```
- Version 1.0 Martin Gut 12.10.2021
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <wiringSerial.h>
#include <wiringPi.h>
#include <string.h>
#include <assert.h>

#define EDGE_PIN 18      //1
#define PPS 23            //4

volatile int EDGE_Cnt = 0; //evtl überflüssig in neuer Version?
volatile int pps_Cnt = 0;
volatile int meas_Cnt = 0;
char buff[100]= {0}, UTC_Date[11]= {0}, Pos_GPS[25]= {0};
int Zeitstempel = 999999;
double Zeitstempel1=0,Zeitstempel2=0,Zeitstempel3=0,Zeitstempel4=0,Zeit-
stempel5=0;
double valueX[10] = {5,5,5,5,5,5,5,5,5,5}; //Initialisierung mit 5 um
Schreiben von 0 zu beobachten
double result=0;
struct timeval tim;
FILE *fp;

void startCnt(void) // Zeitstempel erfassen
{
    //Zeitpunkt der Zeitmessung wird beendet
    if(meas_Cnt > 0)
    {
        gettimeofday(&tim,(struct timezone*)0); //braucht evtl zu viel Re-
chenzeit? Verschlimmbessert das die Messung?
        meas_Cnt--;
    }
    EDGE_Cnt++;

}


void PPS_Clock(void) // evtl anpassen um Startzeitpunkt präziser zu bestim-
men
{
    //gettimeofday(&tim,(struct timezone*)0); //evtl weniger Schwankungen
im Startzeitpunkt wenn Zeitstempel hier bestimmt wird
    pps_Cnt++;
}

//Auslesen der GNRMC Daten (Global Navigation Satelite System - Recommendet
Minimum Specific GNSS Data)
//UART_Rx_GPS() hat verhältnismäßig lange Laufzeit! nicht zwischen zeitrki-
tischen Operationen durchführen
int UART_Rx_GPS(int serial_port, char GPS) //GPS == D: Datum; GPS == T:
Zeit; GPS == P: Position
{
    char dat,GNRMC_buff[5]= {0};
    unsigned char GNRMC_string = 0;
    unsigned char GNRMC_index = 0;
```

```
    unsigned char GNRMC_Time = 0;
    int Zeitstempel_temp = 0;
    //Hilfsvariablen
    int cnt_i=0;
    int cnt_j=0;
    int j=100000;
    //Initialisiere alle Stellen von buff mit einer 0
    memset(buff,0,strlen(buff));

    Zeitstempel_temp = 0; //Zeitistempel zurücksetzen

    while(GNRMC_Time != 3)
    {
        if(serialDataAvail(serial_port))    // Gibt die Anzahl der zum Le-
sen verfügbaren Zeichen oder -1 für jede Fehlerbedingung
        {
            dat=serialGetchar(serial_port); // Gibt das naechste auf dem
seriellen Gerät verfügbare Zeichen zurück.
            // Dieser Aufruf wird bis zu 10 Sekunden lang blockiert, wenn -
1 zurückgegebn wird
            //Startzeichen der Sequenz
            if(dat == '$')
            {
                GNRMC_string  = 0;
                GNRMC_index = 0;
            }
            else if(GNRMC_string  == 1)
            {
                buff[GNRMC_index++] = dat ;

                //Wenn der komplette Zeitstring (hhmmss.00) gefunden und
abgespeichert im Array "buff[]" liegt, wird die if-Verzweigung ausgeführt
                if(GPS == 'T' && strlen(buff) == 9)
                {

                    GNRMC_Time = 1; //Zeitangabe erfolgreich und vollstän-
dig abgepeichert

                }
                else if(GPS =='D' && strlen(buff)==52)
                {

                    GNRMC_Time = 2;

                }
                else if(GPS == 'P' && strlen(buff)==38) //Positionser-
mittlung
                {
                    printf("%s\n", buff);
                    for(cnt_j = 0; cnt_j <= strlen(Pos_GPS); cnt_j++)
                    {
                        Pos_GPS[cnt_j] = buff[cnt_j + 12]; //Position in
lat und long nach RNC GNNS Standard
                    }
                    GNRMC_Time = 3;
                    return 0;
                }
            }
            else if(GNRMC_buff[0]=='G'&& GNRMC_buff[1]=='N'&&
GNRMC_buff[2]=='R'&& GNRMC_buff[3]=='M'&& GNRMC_buff[4]=='C')
            {
                //Wenn der Datensatz GNRMCvollständig entdeckte wurde
                GNRMC_string =1;
```

```
                GNRMC_buff[0]=0;
                GNRMC_buff[1]=0;
                GNRMC_buff[2]=0;
                GNRMC_buff[3]=0;
                GNRMC_buff[4]=0;

            }
            else if(dat == '$' || dat == 'G' || dat == 'N' || dat == 'R' ||
dat == 'M' || dat == 'C')
            {
                GNRMC_buff[0] =GNRMC_buff[1];
                GNRMC_buff[1] =GNRMC_buff[2];
                GNRMC_buff[2] =GNRMC_buff[3];
                GNRMC_buff[3] =GNRMC_buff[4];
                GNRMC_buff[4] =dat;
            }
        }
        //Umrechung des Datentyps char in den double-Datentypen
        if(GNRMC_Time == 1)     //Falls Zeitangabe vollständig und erfolg-
reich empfangen wurde, wird die if-Verzweigung ausgeführt
        {
            GNRMC_Time = 3;   //damit die while-Schleife nach Daten Umwand-
lung abgebrochen werden kann.

            for (cnt_i=0; cnt_i<6; cnt_i++)
            {
                if(cnt_i==1)
                {
                    Zeitstempel_temp+=(buff[cnt_i]-47)*j; //UTC +1
                    j=j/10;
                }
                else if(cnt_i == 5)
                {
                    Zeitstempel1=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[0]; //warum 49 und nicht 48 zur Umrechnung von ASCII in de-
zimal?
                    Zeitstempel2=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[1];
                    Zeitstempel3=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[2];
                    Zeitstempel4=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[3];
                    Zeitstempel5=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[4];
                    Zeitstempel_temp+=(buff[cnt_i]-48)*j; //Damit Zeitstem-
pel auch die Einerstelle des Sekundenwerts erfasst
                    Zeitstempel = Zeitstempel_temp;
                    return 0;
                }
                else
                {
                    Zeitstempel_temp+=(buff[cnt_i]-48)*j;
                    j=j/10;
                }

            }
        }
        else if(GNRMC_Time == 2)
        {

            GNRMC_Time=3;
            UTC_Date[0]=buff[46];
            UTC_Date[1]=buff[47];
```

```
            UTC_Date[2]='.';
            UTC_Date[3]=buff[48];
            UTC_Date[4]=buff[49];
            UTC_Date[5]='.';
            UTC_Date[6]='2';
            UTC_Date[7]='0';
            UTC_Date[8]=buff[50];
            UTC_Date[9]=buff[51];
            return 0;
        }

    }
}


void Flanken_Untersuchung() //wird nicht benutzt, da Filterung extern
{
    //Überprüfung ob die Frequenz des ersten und zweiten Messwertes in die-
sem Messbereich liegt
    if((1/valueX[1]-valueX[0]) < 49.9 || 1/(1/valueX[1]-valueX[0]) > 50.1)
    {

        //Falls False => erster Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        if(valueX[0] > 0.02)
        {

            valueX[0] = 1; // mit 0 beschreiben ist keine eindeuteige Lö-
sung
            printf("valueX[0] außerhalb der Toleranz\n");

        }
        //Falls False => zweiter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        else if(valueX[1] < 0.02 || valueX[1] > 0.04)
        {

            valueX[1] = 1;
            printf("valueX[1] außerhalb der Toleranz\n");

        }
    }
    //Überprüfung ob die Frequenz des zweiten und dritten Messwertes in
diesem Messbereich liegt
    else if((1/valueX[2]-valueX[1]) < 49.9 || 1/(1/valueX[2]-valueX[1]) >
50.1)
    {

        //Falls False => zweiter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        if(valueX[1] < 0.02 || valueX[1] > 0.04)
        {

            valueX[1] = 1;
            printf("valueX[1] außerhalb der Toleranz\n");

        }
        //Falls False => dritter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        else if(valueX[2] < 0.04 || valueX[2] > 0.06)
        {

            valueX[2] = 1;
```

```
            printf("valueX[2] außerhalb der Toleranz\n");

        }


    }

    //Überprüfung ob die Frequenz des dritten und vierten Messwertes in
diesem Messbereich liegt
    else if((1/valueX[3]-valueX[2]) < 49.9 || 1/(1/valueX[3]-valueX[2]) >
50.1)
    {
        //Falls False => dritter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        if(valueX[2] < 0.04 || valueX[2] > 0.06)
        {

            valueX[2] = 1;
            printf("valueX[2] außerhalb der Toleranz\n");

        }
        //Falls False => vierter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        else if(valueX[3] < 0.06 || valueX[3] > 0.08)
        {

            valueX[3] = 1;
            printf("valueX[3] außerhalb der Toleranz\n");

        }


    }

    //Überprüfung ob die Frequenz des vierten und fuenften Messwertes in
diesem Messbereich liegt
    else if((1/valueX[4]-valueX[3]) < 49.9 || 1/(1/valueX[4]-valueX[3]) >
50.1)
    {
        //Falls False => vierter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        if(valueX[3] < 0.06 || valueX[3] > 0.08)
        {

            valueX[3] = 1;
            printf("valueX[3] außerhalb der Toleranz\n");

        }
        //Falls False => fuenfter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
        else if(valueX[4] < 0.08 || valueX[4] > 0.1)
        {

            valueX[4] = 1;
            printf("valueX[4] außerhalb der Toleranz\n");

        }


    }

}
```

```
int maxValue(int myArray[], size_t size) // zur Optimierung, nicht für
Hauptprogramm relevant
{
    /* enforce the contract */
    assert(myArray && size);
    size_t i;
    int maxValue = myArray[0];

    for (i = 1; i < size; ++i) {
        if ( myArray[i] > maxValue ) {
            maxValue = myArray[i];
        }
    }
    return maxValue;
}

int minValue(int myArray[], size_t size) // zur Optimierung, nicht für
Hauptprogramm relevant
{
    /* enforce the contract */
    assert(myArray && size);
    size_t i;
    int minValue = myArray[0];

    for (i = 1; i < size; ++i) {
        if ( myArray[i] < minValue ) {
            minValue = myArray[i];
        }
    }
    return minValue;
}

int main()
{
    ////////////////////////////////////////////////////////////////////////
    //Setup
    ////////////////////////////////////////////////////////////////////////

    //Benutzereingabe
    ////////////////////////////////////////////////////////////////////////
    char deviceNumber = '1'; //WICHTIG: keine zwei Geräte mit der geichen
deviceNumber, ansonsten können Messdaten nicht zugeordnet werden
    bool isPi3 = true; //auf false setzen, wenn mit RaspberriPi 4 gearbei-
tet wird
    int time_gap = 0; //Anzahl der Minuten, die zwischen zei Messminuten
liegen (0 für minütige Messungen)
    int no_meas = 5; //number of measrements - Anzahl der Minutenmessungen
(auf 0 Stellen für Endlosmessung)
    // Hinweis für Langzeitmessungen: Datumsstempel zwischen 0 und 2 Uhr
morgens evtl. inkorrekt
    ////////////////////////////////////////////////////////////////////////
    time_gap += 1;

    //Variablen zur Analyse der Zeitverzögerung
    int stampDelay[no_meas - 1];
    int maxDelay = 0;
    int minDelay = 0;

    //main Variabelen-Definition
    int serial_port = 0;              // eine ganzzahlige Variable, dient
als Ruckgabewert für Initialieserung der UART-Schnittstelle
```

```
    char Date ='D',Time ='T', Position ='P';         // character-Hilfs Va-
riable: für die Ermittlung des Datenstrings des GPS-Signals
    double t1 = 0;
    int pps_sekunden = 0;
    double mittelwert = 0;
    double cnt_x =0;
    char filepath[50] = "Messwerte/"; //Wird um Zeotstempel und Dateinamen
erweitert
    char timestampString[20];
    char dateString[9] = "XXXXXXXX";
    int meas_per_sec = 5; //Messwerte, die bei Start pro Sekunde aufgenom-
men werden, muss aktuell auf 5 bleiben, da das Programm mit exakt 5 valueX
Werten rechnet
    int curr_meas = 0; //aktuelle Messungsnummer
    int total_meas = 0; //gesamte Messdurchläufe
    struct timeval tim_old;
    unsigned int min_since_start = 0;


    //UART Port festlegen
    char uartPort[20];
    if(isPi3)
    {
        strcpy(uartPort, "/dev/ttyS0");
    }
    else
    {
        strcpy(uartPort, "/dev/ttyAMA0");
    }

    printf("Programmstart\n");

    //Oeffnen der seriellen Schnittstelle fur den GPS-GNSS-5-Click Empfan-
gers
    if((serial_port = serialOpen(uartPort, 9600)) < 0) // bei RP3 ist
"/dev/ttyAMA0" für Bluetooth, ttyS0 für UART
    {
        printf("Fehler bei der Initialisierung der UART-Schnittstelle mit
dem GNSS");
        return -1;
    }

    //Initialisierung der GPIO-Zugriffsbibliothek
    if(wiringPiSetupGpio() == -1)
    {
        printf("Fehler bei der Initialisierung von WiringPiSetup");
        return -1;
    }
    //Output-Interrupt-Konfiguration fur die steigende PPS-Flanke
    if (wiringPiISR (PPS, INT_EDGE_RISING, &PPS_Clock) < 0 )
    {
        printf("Fehler beim setup des Interrupts von low-to-high Pulse Pro
Second Signal transitions");
        return -1;
    }
    //Output-Interrupts-Konfiguration fur die Rechtecksignale
    if (wiringPiISR (EDGE_PIN, INT_EDGE_RISING, &startCnt) < 0 ) //rising
Edge? Fallende Flanken angeblich präziser
    {
        printf("Fehler beim setup des Interrupts von low-to-high Voltage
transitions");
        return -1;
    }
```

```
    UART_Rx_GPS(serial_port, Date);
    //Schleifenbeginn - Programm wird ausgeführt bis no_meas Messungen im
Abstand von time_gap Minuten durchgeführt wurdem
    while(total_meas < no_meas || no_meas == 0)
    {
        UART_Rx_GPS(serial_port, Time);
        printf("%d\n", Zeitstempel);
        if((Zeitstempel % 100) == 0) //Messung immer zur vollen Minute
        {
            printf("Minuten seit Start: %d\n", min_since_start);
            if((min_since_start % time_gap) == 0)
            {
                printf("volle Minute %d\n", Zeitstempel);

////////////////////////////////////////////////////////////////////
                //Datei erzeugen

////////////////////////////////////////////////////////////////////
                //Datumsstempel für Namen der .csv Datei an filepath anfü-
gen
                memcpy(dateString, &UTC_Date[6], 4);
                dateString[4] = UTC_Date[3]; // ein bisschen hässlich, aber
funktioniert
                dateString[5] = UTC_Date[4];
                dateString[6] = UTC_Date[0];
                dateString[7] = UTC_Date[1];
                strcat(filepath, dateString);
                strcat(filepath, "-");

                //Zeitstempel für Namen der .csv Datei generieren und an
filepath anfügen
                snprintf(timestampString, 20, "%d", Zeitstempel);
                strcat(filepath, timestampString);
                strcat(filepath, "_PMU");
                strncat(filepath, &deviceNumber, 1);
                strcat(filepath, "-MW.csv");
                //Dateibenennung evtl. in eigene Funktion ausgliedern

                //Speichermodus festlegen
                fp =fopen(filepath, "w");
                if(fp == NULL)
                {
                    printf("Datei konnte nicht erstellt werden.");
                    return -1;
                }
                //auf volle Sekunde warten
                pps_Cnt = 0;
                while(pps_Cnt == 0) //hässlich aber was solls
                {
                    //printf("Warten auf volle Sekunde für erste Messrou-
tine\n");
                }

////////////////////////////////////////////////////////////////////
                //Messroutine 1 beginnen

////////////////////////////////////////////////////////////////////
                gettimeofday(&tim,(struct timezone*)0); //evtl in void
PPS_Clock() erfassen um näher am vollen Sekundensignal zu sein
                t1 = tim.tv_sec+(tim.tv_usec/1000000.0); //Referenzzeit-
stempel zur vollen Sekunde
                curr_meas = meas_per_sec;
```

```
                // fallende Flanke werden ermittlet
                meas_Cnt = meas_per_sec + 1;
                while (curr_meas > 0) //meas_Cnt wird in startCnt()
herabgezählt
                {
                    if(meas_Cnt == curr_meas && tim.tv_usec !=
tim_old.tv_usec)
                    {
                        valueX[meas_per_sec - meas_Cnt] =
(tim.tv_sec+(tim.tv_usec/1000000.0)-t1); //valueX von 0-4 werden
beschrieben
                        tim_old = tim; //um doppelte Speicherung des glei-
chen Zeitstempels zu verhindern
                        curr_meas --;
                    }
                }
                //Werte abspeichern
                fprintf(fp,"Messwerte von %d steigenden Spannungsflanken-
messungen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeit-
stempel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                printf("Messwerte von %d steigenden Spannungsflankenmessun-
gen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeitstem-
pel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                fprintf(fp, "%lf, %lf, %lf, %lf\n", valueX[0], valueX[1],
valueX[2], valueX[3], valueX[4]);
                printf("%lf, %lf, %lf, %lf\n", valueX[0], valueX[1], val-
ueX[2], valueX[3], valueX[4]);
                //Messwerte mit 5 überschreiben um übersprungene Messpunkte
einfach zu erkennen
                for(int i = 0; i< sizeof(valueX)/sizeof(valueX[0]); i++)
//valueX zurücksetzen
                        valueX[i] = 5;
                //30 Sekunden warten
                while((Zeitstempel % 100) != 30)
                {
                    UART_Rx_GPS(serial_port, Time);
                    printf("%d\n", Zeitstempel);
                }
                printf("halbe Minute %d\n", Zeitstempel);
                //auf volle Sekunde warten
                pps_Cnt = 0;
                while(pps_Cnt == 0) //hässlich aber was solls
                {
                    //printf("Warten auf volle Sekunde für zweite Messrou-
tine\n");
                }

////////////////////////////////////////////////////////////////////
                //Messroutine 2 nach 30 Sekunden beginnen

////////////////////////////////////////////////////////////////////
                gettimeofday(&tim,(struct timezone*)0); //evtl in void
PPS_Clock() erfassen um näher am vollen Sekundensignal zu sein
                t1 = tim.tv_sec+(tim.tv_usec/1000000.0); //Referenzzeit-
stempel zur vollen Sekunde
                curr_meas = meas_per_sec;
                // fallende Flanke werden ermittlet
                meas_Cnt = meas_per_sec + 1;
                while (curr_meas > 0) //meas_Cnt wird in startCnt()
herabgezählt
                {
                    if(meas_Cnt == curr_meas && tim.tv_usec !=
tim_old.tv_usec)
```

```
                    {
                        valueX[meas_per_sec - meas_Cnt] =
(tim.tv_sec+(tim.tv_usec/1000000.0)-t1); //valueX von 0-4 werden
beschrieben
                        tim_old = tim; //um doppelte Speicherung des glei-
chen Zeitstempels zu verhindern
                        curr_meas --;
                    }
                }
                //Werte abspeichern
                fprintf(fp,"Messwerte von %d steigenden Spannungsflanken-
messungen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeit-
stempel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                printf("Messwerte von %d steigenden Spannungsflankenmessun-
gen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeitstem-
pel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                fprintf(fp, "%lf, %lf, %lf, %lf\n", valueX[0], valueX[1],
valueX[2], valueX[3], valueX[4]);
                printf("%lf, %lf, %lf, %lf\n", valueX[0], valueX[1], val-
ueX[2], valueX[3], valueX[4]);

                //Strings zurücksetzen: datestring, filepath, timestamp-
String
                memset(dateString, 0, sizeof(dateString));
                strcat(dateString, "XXXXXXXX"); //Wahrscheinlich unnötig,
aber sicher ist sicher
                memset(timestampString, 0, sizeof(timestampString));
                memset(filepath, 0, sizeof(filepath));
                strcat(filepath, "Messwerte/");
                fclose(fp);

                system("./DropboxUpload.py");
                total_meas += 1;
            }
            min_since_start += 1;
        }
        if(Zeitstempel / 100 == 200) // Um zwei Uhr morgens wird ein neuer
Datumsstempel generiert
            UART_Rx_GPS(serial_port, Date); // Messwerte von 0 bis 2 Uhr
können falschen Datumsstempel haben, da wegen Zeitverschiebung der Stempel
erst um 2 Uhr aufgenommen wird
    }
    return 0;

}
```

10.2 C-Code für zur Überprüfung der Messgenauigkeit

```
/*
Zeitstempel.c
Dieses Programm misst die Zeitverschiebung zwischen zwei Flankensignalen
und Speichert das Ergebnis in einer .csv Datei ab.

Version 2.6 Stand 19.05.2022
Autor Marius Kleinbach
Optimierungen der Testversion in Hauptversion übernommen.

VERSIONSGESCHICHTE
- Version 2.5.3 Stand 05.05.2022
Autor Marius Kleinbach
```

```
Testversion um die Präzision der Interrupt Routine zu überprüfen.
Das Datum für die Vergabe der Dateinamen wird vom System erfasst, damit
auch ohne GNSS5-Click gearbeitet werden kann.
Die Uhrzeit wird ebenfalls lokal efasst. Das Gerät sollte deshalb mit dem
Internet verbunden sein, um eine korrekte Uhrzeit zu gewährleisten.
Die Zeiterfassung der einzelnen Messungen findet nun innerhalb der Inter-
rupt Routinen statt.
- Version 2.5.2 Stand 24.03.2022
Autor Marius Kleinbach
Testversion um die Präzision der Interrupt Routine zu überprüfen.
Messergebnisse werden nun alle 2 Sekunden erfasst.
- Version 2.5.1 Stand 24.03.2022
Autor Marius Kleinbach
Testversion um die Präzision der Interrupt Routine zu überprüfen.
Es wird die Zeitdifferenz zwischen einer steigenden und einer fallenden
Flanke eines Funktionsgenerators gemessen.
Die Ergebnisse werden in minütlichen .csv Dateien abgespeichert und in eine
Dropbox Cloud hochgeladen.
- Version 2.5 Stand 27.01.2022
Autor Marius Kleinbach
Aufruf des python Skripts zum Upload in die Dropbox wird nach jeder Mess-
routine aufgerufen
Messungen können nun im Abstand von time_gap durgeführt werden
-Version 2.4 Stand 26.01.2022
Autor Marius Kleinbach
Programmstruktur angepasst: Pro Messminute wird eine eigene .csv Datei er-
stellt
-Version 2.3 Stand 25.01.2022
Autor Marius Kleinbach
Zeitdeltas abgespeichert anstatt errechneter Frequenz wenn short_data_for-
mat == true
Variabler Dateiname bei unterschiedlichen Devices (IMMER deviceNumber AN-
PASSEN)
-Version 2.2 Stand 20.01.2022
Autor Marius Kleinbach
Bugfix: pps_Cnt vor Messung zurückgesetzt
Bugfix: Flanken_Untersuchen setzt valueX[i] nun auf 1 wenn außerhalb des
Toleranzbereichs anstatt auf 0
Am besten wäre es den Messwert komplett zu ignorieren aber dies dient als
temporärer Fix um
Fehlerhafte Messwerte schnell zu erkennen
valueX[] zählt nun von 0 los, nicht von 1
Zeitstempel Nummerierung angepasst
- Version 2.1 Stand 17.01.2022
Autor Marius Kleinbach
Mess-Zeitstempel wird in direkt in Hardware Interrupt Schleife bestimmt
- Version 2.0 Stand 13.01.2022
Autor Marius Kleinbach
Zeiterfassung direkt in Hardware-Interrupt
Loggen der Zeitstempel um Schwankungen festzustellen
- Version 1.3 Stand 13.01.2022
Autor Marius Kleinbach
UART_Rx_GPS(int, char) angepasst und um Positionserfassung erweitert
Dateiname enthält nun Zeitstempel
- Version 1.2 Stand 13.12.2021
Autor Marius Kleinbach
ttyS0 Port anstatt ttAMA0 hat Einleseproblem gelöst
Programm läuft, wenn regelmäßige Spannungsflanken an EDGE_PIN registriert
werden
- Version 1.1 Stand 08.12.2021
Autor Marius Kleinbach
libwiringPi.so muss in den Debugger Einstellungen verlinkt werden
Hinzufügen von Konsolenkommentaren zum Debugging
```

```
Schrittweises deaktivieren von Kommunikationsschnittstellen
um Funktionsweise zu validieren
Zeile 77 liest immer den gleichen Wert ein, was zum Aufhängen des Programms
führt

- Version 1.0 Martin Gut 12.10.2021
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <wiringSerial.h>
#include <wiringPi.h>
#include <string.h>
#include <assert.h>
#include <time.h>
#include <string.h>
#include <sys/time.h>

#define EDGE_PIN 18       //GPIO 18
#define PPS 23            //GPIO 23

volatile int EDGE_Cnt = 0; //evtl überflüssig in neuer Version?
volatile int pps_Cnt = 0;
volatile int meas_Cnt = 0;
char buff[100]= {0}, UTC_Date[11]= {0}, Pos_GPS[25]= {0};
int Zeitstempel = 999999, Zeitstempel_alt = 999999;
double Zeitstempel1=0,Zeitstempel2=0,Zeitstempel3=0,Zeitstempel4=0,Zeit-
stempel5=0;
double valueX[10] = {5,5,5,5,5,5,5,5,5,5}; //Initialisierung mit 5 um
Schreiben von 0 zu beobachten
double result=0;
struct timeval tim;
FILE *fp;

void startCnt(void) // Zeitstempel erfassen
{
    //Zeitpunkt der Zeitmessung wird beendet
    if(meas_Cnt > 0)
    {
        gettimeofday(&tim,(struct timezone*)0); //braucht evtl zu viel Re-
chenzeit? Verschlimmbessert das die Messung?
        meas_Cnt--;
    }
    EDGE_Cnt++;
}


void PPS_Clock(void) // evtl anpassen um Startzeitpunkt präziser zu bestim-
men
{
    if(pps_Cnt == 0)
        gettimeofday(&tim,(struct timezone*)0); //evtl weniger Schwankungen
im Startzeitpunkt wenn Zeitstempel hier bestimmt wird
    pps_Cnt++;
}

//Auslesen der GNRMC Daten (Global Navigation Satelite System - Recommendet
Minimum Specific GNSS Data)
```

```
//UART_Rx_GPS() hat verhältnismäßig lange Laufzeit! nicht zwischen zeitrki-
tischen Operationen durchführen
//UART_Rx_GPS() wird ab v2.6 nicht mehr verwendet. Anstatt dessen Zeit und
Datum von System bezogen
//int UART_Rx_GPS(int serial_port, char GPS) //GPS == D: Datum; GPS == T:
Zeit; GPS == P: Position
//{
//    char dat,GNRMC_buff[5]= {0};
//    unsigned char GNRMC_string = 0;
//    unsigned char GNRMC_index = 0;
//    unsigned char GNRMC_Time = 0;
//    int Zeitstempel_temp = 0;
//    //Hilfsvariablen
//    int cnt_i=0;
//    int cnt_j=0;
//    int j=100000;
//    //Initialisiere alle Stellen von buff mit einer 0
//    memset(buff,0,strlen(buff));
//
//    Zeitstempel_temp = 0; //Zeitistempel zurücksetzen
//
//    while(GNRMC_Time != 3)
//    {
//        if(serialDataAvail(serial_port))    // Gibt die Anzahl der zum
Lesen verfügbaren Zeichen oder -1 für jede Fehlerbedingung
//        {
//            dat=serialGetchar(serial_port); // Gibt das naechste auf dem
seriellen Gerät verfügbare Zeichen zurück.
//            // Dieser Aufruf wird bis zu 10 Sekunden lang blockiert, wenn
-1 zurückgegebn wird
//            //Startzeichen der Sequenz
//            if(dat == '$')
//            {
//                GNRMC_string  = 0;
//                GNRMC_index = 0;
//            }
//            else if(GNRMC_string  == 1)
//            {
//                buff[GNRMC_index++] = dat ;
//
//                //Wenn der komplette Zeitstring (hhmmss.00) gefunden und
abgespeichert im Array "buff[]" liegt, wird die if-Verzweigung ausgeführt
//                if(GPS == 'T' && strlen(buff) == 9)
//                {
//
//                    GNRMC_Time = 1; //Zeitangabe erfolgreich und voll-
ständig abgepeichert
//
//                }
//                else if(GPS =='D' && strlen(buff)==52)
//                {
//
//                    GNRMC_Time = 2;
//
//                }
//                else if(GPS == 'P' && strlen(buff)==38) //Positionser-
mittlung
//                {
//                    printf("%s\n", buff);
//                    for(cnt_j = 0; cnt_j <= strlen(Pos_GPS); cnt_j++)
//                    {
//                        Pos_GPS[cnt_j] = buff[cnt_j + 12]; //Position in
lat und long nach RNC GNNS Standard
```

```
//                    }
//                    GNRMC_Time = 3;
//                    return 0;
//                }
//            }
//            else if(GNRMC_buff[0]=='G'&& GNRMC_buff[1]=='N'&&
GNRMC_buff[2]=='R'&& GNRMC_buff[3]=='M'&& GNRMC_buff[4]=='C')
//            {
//                //Wenn der Datensatz GNRMCvollständig entdeckte wurde
//                GNRMC_string =1;
//                GNRMC_buff[0]=0;
//                GNRMC_buff[1]=0;
//                GNRMC_buff[2]=0;
//                GNRMC_buff[3]=0;
//                GNRMC_buff[4]=0;
//
//            }
//            else if(dat == '$' || dat == 'G' || dat == 'N' || dat == 'R'
|| dat == 'M' || dat == 'C')
//            {
//                GNRMC_buff[0] =GNRMC_buff[1];
//                GNRMC_buff[1] =GNRMC_buff[2];
//                GNRMC_buff[2] =GNRMC_buff[3];
//                GNRMC_buff[3] =GNRMC_buff[4];
//                GNRMC_buff[4] =dat;
//            }
//        }
//        //Umrechung des Datentyps char in den double-Datentypen
//        if(GNRMC_Time == 1)     //Falls Zeitangabe vollständig und er-
folgreich empfangen wurde, wird die if-Verzweigung ausgeführt
//        {
//            GNRMC_Time = 3;   //damit die while-Schleife nach Daten Um-
wandlung abgebrochen werden kann.
//
//            for (cnt_i=0; cnt_i<6; cnt_i++)
//            {
//                if(cnt_i==1)
//                {
//                    Zeitstempel_temp+=(buff[cnt_i]-47)*j; //UTC +1
//                    j=j/10;
//                }
//                else if(cnt_i == 5)
//                {
//                    Zeitstempel1=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[0]; //warum 49 und nicht 48 zur Umrechnung von ASCII in de-
zimal?
//                    Zeitstempel2=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[1]; //sind diese Zeilen veraltet und überflüssig?
//                    Zeitstempel3=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[2];
//                    Zeitstempel4=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[3];
//                    Zeitstempel5=(Zeitstempel_temp+((buff[cnt_i]-
49)*j))+valueX[4];
//                    Zeitstempel_temp+=(buff[cnt_i]-48)*j; //Damit Zeit-
stempel auch die Einerstelle des Sekundenwerts erfasst
//                    Zeitstempel = Zeitstempel_temp;
//                    return 0;
//                }
//                else
//                {
//                    Zeitstempel_temp+=(buff[cnt_i]-48)*j;
//                    j=j/10;
```

```
//                }
//
//            }
//        }
//        else if(GNRMC_Time == 2)
//        {
//
//            GNRMC_Time=3;
//            UTC_Date[0]=buff[46];
//            UTC_Date[1]=buff[47];
//            UTC_Date[2]='.';
//            UTC_Date[3]=buff[48];
//            UTC_Date[4]=buff[49];
//            UTC_Date[5]='.';
//            UTC_Date[6]='2';
//            UTC_Date[7]='0';
//            UTC_Date[8]=buff[50];
//            UTC_Date[9]=buff[51];
//            return 0;
//        }
//
//    }
//}
//
//void Flanken_Untersuchung() //wird nicht benutzt, da Filerung extern
//{
//    //Überprüfung ob die Frequenz des ersten und zweiten Messwertes in
diesem Messbereich liegt
//    if((1/valueX[1]-valueX[0]) < 49.9 || 1/(1/valueX[1]-valueX[0]) >
50.1)
//    {
//
//        //Falls False => erster Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        if(valueX[0] > 0.02)
//        {
//
//            valueX[0] = 1; // mit 0 beschreiben ist keine eindeuteige Lö-
sung
//            printf("valueX[0] außerhalb der Toleranz\n");
//
//        }
//        //Falls False => zweiter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        else if(valueX[1] < 0.02 || valueX[1] > 0.04)
//        {
//
//            valueX[1] = 1;
//            printf("valueX[1] außerhalb der Toleranz\n");
//
//        }
//    }
//    //Überprüfung ob die Frequenz des zweiten und dritten Messwertes in
diesem Messbereich liegt
//    else if((1/valueX[2]-valueX[1]) < 49.9 || 1/(1/valueX[2]-valueX[1]) >
50.1)
//    {
//
//        //Falls False => zweiter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        if(valueX[1] < 0.02 || valueX[1] > 0.04)
//        {
//
```

```
//            valueX[1] = 1;
//            printf("valueX[1] außerhalb der Toleranz\n");
//
//        }
//        //Falls False => dritter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        else if(valueX[2] < 0.04 || valueX[2] > 0.06)
//        {
//
//            valueX[2] = 1;
//            printf("valueX[2] außerhalb der Toleranz\n");
//
//        }
//
//
//    }
//
//    //Überprüfung ob die Frequenz des dritten und vierten Messwertes in
diesem Messbereich liegt
//    else if((1/valueX[3]-valueX[2]) < 49.9 || 1/(1/valueX[3]-valueX[2]) >
50.1)
//    {
//        //Falls False => dritter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        if(valueX[2] < 0.04 || valueX[2] > 0.06)
//        {
//
//            valueX[2] = 1;
//            printf("valueX[2] außerhalb der Toleranz\n");
//
//        }
//        //Falls False => vierter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        else if(valueX[3] < 0.06 || valueX[3] > 0.08)
//        {
//
//            valueX[3] = 1;
//            printf("valueX[3] außerhalb der Toleranz\n");
//
//        }
//
//
//    }
//
//    //Überprüfung ob die Frequenz des vierten und fuenften Messwertes in
diesem Messbereich liegt
//    else if((1/valueX[4]-valueX[3]) < 49.9 || 1/(1/valueX[4]-valueX[3]) >
50.1)
//    {
//        //Falls False => vierter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        if(valueX[3] < 0.06 || valueX[3] > 0.08)
//        {
//
//            valueX[3] = 1;
//            printf("valueX[3] außerhalb der Toleranz\n");
//
//        }
//        //Falls False => fuenfter Messwert ist ein Ausreißer und wird mit
Null ueberschrieben
//        else if(valueX[4] < 0.08 || valueX[4] > 0.1)
//        {
//
```

```
//            valueX[4] = 1;
//            printf("valueX[4] außerhalb der Toleranz\n");
//
//        }
//
//
//    }
//
//}
//
//
//int maxValue(int myArray[], size_t size) // zur Optimierung, nicht für
Hauptprogramm relevant
//{
//    /* enforce the contract */
//    assert(myArray && size);
//    size_t i;
//    int maxValue = myArray[0];
//
//    for (i = 1; i < size; ++i) {
//        if ( myArray[i] > maxValue ) {
//            maxValue = myArray[i];
//        }
//    }
//    return maxValue;
//}
//
//int minValue(int myArray[], size_t size) // zur Optimierung, nicht für
Hauptprogramm relevant
//{
//    /* enforce the contract */
//    assert(myArray && size);
//    size_t i;
//    int minValue = myArray[0];
//
//    for (i = 1; i < size; ++i) {
//        if ( myArray[i] < minValue ) {
//            minValue = myArray[i];
//        }
//    }
//    return minValue;
//}

int main()
{
    ////////////////////////////////////////////////////////////////////////
    //Setup
    ////////////////////////////////////////////////////////////////////////

    //Benutzereingabe
    ////////////////////////////////////////////////////////////////////////
    char deviceNumber = '1'; //WICHTIG: keine zwei Geräte mit der geichen
deviceNumber, ansonsten können Messdaten nicht zugeordnet werden
    bool isPi3 = false; //auf false setzen, wenn mit RaspberriPi 4 gearbei-
tet wird
    int time_gap = 0; //Anzahl der Minuten, die zwischen zei Messminuten
liegen (0 für minütige Messungen)
    int no_meas = 0; //number of measrements - Anzahl der Minutenmessungen
(auf 0 Stellen für Endlosmessung)
    // Hinweis für Langzeitmessungen: Datumsstempel zwischen 0 und 2 Uhr
morgens evtl. inkorrekt
    ////////////////////////////////////////////////////////////////////////
    time_gap += 1;
```

```
    //Variablen zur Analyse der Zeitverzögerung
    int stampDelay[no_meas - 1];
    int maxDelay = 0;
    int minDelay = 0;

    //main Variabelen-Definition
    int serial_port = 0;              // eine ganzzahlige Variable, dient
als Ruckgabewert für Initialieserung der UART-Schnittstelle
    char Date ='D',Time ='T', Position ='P';         // character-Hilfs Va-
riable: für die Ermittlung des Datenstrings des GPS-Signals
    double t1 = 0;
    int pps_sekunden = 0;
    double mittelwert = 0;
    double cnt_x =0;
    char filepath[50] = "Messwerte/"; //Wird um Zeotstempel und Dateinamen
erweitert
    char timestampString[20];
    char dateString[9] = "XXXXXXXX";
    int meas_per_sec = 5; //Messwerte, die bei Start pro Sekunde aufgenom-
men werden, muss aktuell auf 5 bleiben, da das Programm mit exakt 5 valueX
Werten rechnet
    int curr_meas = 0; //aktuelle Messungsnummer
    int total_meas = 0; //gesamte Messdurchläufe
    struct timeval tim_old;
    unsigned int min_since_start = 0;
    //Für lokale Datumsbestimmung
    time_t t;
    t = time(NULL);
    struct tm tm = *localtime(&t);


    //UART Port festlegen
    char uartPort[20];
    if(isPi3)
    {
        strcpy(uartPort, "/dev/ttyS0");
    }
    else
    {
        strcpy(uartPort, "/dev/ttyAMA0");
    }

    printf("Programmstart\n");

    //Oeffnen der seriellen Schnittstelle fur den GPS-GNSS-5-Click Empfan-
gers
    if((serial_port = serialOpen(uartPort, 9600)) < 0) // bei RP3 ist
"/dev/ttyAMA0" für Bluetooth, ttyS0 für UART
    {
        printf("Fehler bei der Initialisierung der UART-Schnittstelle mit
dem GNSS");
        return -1;
    }

    //Initialisierung der GPIO-Zugriffsbibliothek
    if(wiringPiSetupGpio() == -1)
    {
        printf("Fehler bei der Initialisierung von WiringPiSetup");
        return -1;
    }
    //Output-Interrupt-Konfiguration fur die steigende PPS-Flanke
    if (wiringPiISR (PPS, INT_EDGE_RISING, &PPS_Clock) < 0 )
```

```
    {
        printf("Fehler beim setup des Interrupts von low-to-high Pulse Pro
Second Signal transitions");
        return -1;
    }
    //Output-Interrupts-Konfiguration fur die Rechtecksignale
    if (wiringPiISR (EDGE_PIN, INT_EDGE_RISING, &startCnt) < 0 ) //rising
Edge? Fallende Flanken angeblich präziser
    {
        printf("Fehler beim setup des Interrupts von low-to-high Voltage
transitions");
        return -1;
    }

    //UART_Rx_GPS(serial_port, Date); //Datum von GPS beziehen
    snprintf(UTC_Date, 11, "%02d.%02d.%d", tm.tm_mday, tm.tm_mon+1,
tm.tm_year+1900); //Datum von lokalem Gerät beziehen

    //Schleifenbeginn - Programm wird ausgeführt bis no_meas Messungen im
Abstand von time_gap Minuten durchgeführt wurdem
    while(total_meas < no_meas || no_meas == 0)
    {
        //UART_Rx_GPS(serial_port, Time); //Zeit über GPS erfassen
        t = time(NULL); //Zeit lokal erfassen
        tm = *localtime(&t);
        Zeitstempel = tm.tm_hour * 10000 + tm.tm_min * 100 + tm.tm_sec;
        if(Zeitstempel != Zeitstempel_alt)
            printf("%d\n", Zeitstempel);
        Zeitstempel_alt = Zeitstempel;
        if((Zeitstempel % 100) == 0) //Messung immer zur vollen Minute
        {
            printf("Minuten seit Start: %d\n", min_since_start);
            if((min_since_start % time_gap) == 0)
            {
                printf("volle Minute %d\n", Zeitstempel);

////////////////////////////////////////////////////////////////////
                //Datei erzeugen

////////////////////////////////////////////////////////////////////
                //Datumsstempel für Namen der .csv Datei an filepath anfü-
gen
                memcpy(dateString, &UTC_Date[6], 4);
                dateString[4] = UTC_Date[3]; // ein bisschen hässlich, aber
funktioniert
                dateString[5] = UTC_Date[4];
                dateString[6] = UTC_Date[0];
                dateString[7] = UTC_Date[1];
                strcat(filepath, dateString);
                strcat(filepath, "-");

                //Zeitstempel für Namen der .csv Datei generieren und an
filepath anfügen
                snprintf(timestampString, 20, "%d", Zeitstempel);
                strcat(filepath, timestampString);
                strcat(filepath, "_PMU");
                strncat(filepath, &deviceNumber, 1);
                strcat(filepath, "-MW.csv");
                //Dateibenennung evtl. in eigene Funktion ausgliedern

                //Speichermodus festlegen
                fp =fopen(filepath, "w");
                if(fp == NULL)
```

```
                {
                    printf("Datei konnte nicht erstellt werden.");
                    return -1;
                }
                //auf volle Sekunde warten
                pps_Cnt = 0;
                while(pps_Cnt == 0) //hässlich aber was solls
                {
                    //printf("Warten auf volle Sekunde für erste Messrou-
tine\n");
                }

////////////////////////////////////////////////////////////////////
                //Messroutine 1 beginnen

////////////////////////////////////////////////////////////////////
                //gettimeofday(&tim,(struct timezone*)0); //evtl in void
PPS_Clock() erfassen um näher am vollen Sekundensignal zu sein
                t1 = tim.tv_sec+(tim.tv_usec/1000000.0); //Referenzzeit-
stempel zur vollen Sekunde
                curr_meas = meas_per_sec;
                // fallende Flanke werden ermittlet
                meas_Cnt = meas_per_sec + 1;
                while (curr_meas > 0) //meas_Cnt wird in startCnt()
herabgezählt
                {
                    if(meas_Cnt == curr_meas && tim.tv_usec !=
tim_old.tv_usec)
                    {
                        valueX[meas_per_sec - meas_Cnt] =
(tim.tv_sec+(tim.tv_usec/1000000.0)-t1); //valueX von 0-4 werden
beschrieben
                        tim_old = tim; //um doppelte Speicherung des glei-
chen Zeitstempels zu verhindern
                        curr_meas --;
                    }
                }
                //Werte abspeichern
                fprintf(fp,"Messwerte von %d steigenden Spannungsflanken-
messungen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeit-
stempel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                printf("Messwerte von %d steigenden Spannungsflankenmessun-
gen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeitstem-
pel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                fprintf(fp, "%lf, %lf, %lf, %lf\n", valueX[0], valueX[1],
valueX[2], valueX[3], valueX[4]);
                printf("%lf, %lf, %lf, %lf\n", valueX[0], valueX[1], val-
ueX[2], valueX[3], valueX[4]);
                //Messwerte mit 5 überschreiben um übersprungene Messpunkte
einfach zu erkennen
                for(int i = 0; i< sizeof(valueX)/sizeof(valueX[0]); i++)
//valueX zurücksetzen
                    valueX[i] = 5;
                //25 x 2 Sekunden warten
                for(int i = 1; i <= 25; i++)
                {
                    while((Zeitstempel % 100) != 2 * i)
                    {
                        //UART_Rx_GPS(serial_port, Time); //Zeit von GPS
beziehen
                        t = time(NULL); //Zeit lokal erfassen
                        tm = *localtime(&t);
```

```
                        Zeitstempel = tm.tm_hour * 10000 + tm.tm_min * 100
+ tm.tm_sec;
                        if(Zeitstempel != Zeitstempel_alt)
                            printf("%d\n", Zeitstempel);
                        Zeitstempel_alt = Zeitstempel;
                    }
                    printf("%d Sekunden vergangen %d\n", 2 * i, Zeitstem-
pel);
                    //auf volle Sekunde warten
                    pps_Cnt = 0;
                    while(pps_Cnt == 0) //hässlich aber was solls
                    {
                        //printf("Warten auf volle Sekunde für zweite Mess-
routine\n");
                    }

////////////////////////////////////////////////////////////////////
                    //Messroutine 2 nach 30 Sekunden beginnen

////////////////////////////////////////////////////////////////////
                    //gettimeofday(&tim,(struct timezone*)0); //evtl in
void PPS_Clock() erfassen um näher am vollen Sekundensignal zu sein
                    t1 = tim.tv_sec+(tim.tv_usec/1000000.0); //Referenz-
zeitstempel zur vollen Sekunde
                    curr_meas = meas_per_sec;
                    // fallende Flanke werden ermittlet
                    meas_Cnt = meas_per_sec + 1;
                    while (curr_meas > 0) //meas_Cnt wird in startCnt()
herabgezählt
                    {
                        if(meas_Cnt == curr_meas && tim.tv_usec !=
tim_old.tv_usec)
                        {
                            valueX[meas_per_sec - meas_Cnt] =
(tim.tv_sec+(tim.tv_usec/1000000.0)-t1); //valueX von 0-4 werden
beschrieben
                            tim_old = tim; //um doppelte Speicherung des
gleichen Zeitstempels zu verhindern
                            curr_meas --;
                        }
                    }
                    //Werte abspeichern
                    fprintf(fp,"Messwerte von %d steigenden Spannungsflan-
kenmessungen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeit-
stempel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                    printf("Messwerte von %d steigenden Spannungsflanken-
messungen am %s um %02d:%02d:%02d Uhr:\n", meas_per_sec, UTC_Date, Zeit-
stempel/10000, (Zeitstempel/100) % 100, Zeitstempel % 100);
                    fprintf(fp, "%lf, %lf, %lf, %lf\n", valueX[0], val-
ueX[1], valueX[2], valueX[3], valueX[4]);
                    printf("%lf, %lf, %lf, %lf\n", valueX[0], valueX[1],
valueX[2], valueX[3], valueX[4]);
                }
                //Strings zurücksetzen: datestring, filepath, timestamp-
String
                memset(dateString, 0, sizeof(dateString));
                strcat(dateString, "XXXXXXXX"); //Wahrscheinlich unnötig,
aber sicher ist sicher
                memset(timestampString, 0, sizeof(timestampString));
                memset(filepath, 0, sizeof(filepath));
                strcat(filepath, "Messwerte/");
                fclose(fp);
```

```
                //system("./DropboxUpload.py"); //Upload der Messwerte in
die Dropbox Cloud
                total_meas += 1;
            }
            min_since_start += 1;
        }
        if(Zeitstempel / 100 == 200) // Um zwei Uhr morgens wird ein neuer
Datumsstempel generiert
            //Messwerte von 0 bis 2 Uhr können falschen Datumsstempel ha-
ben, da wegen Zeitverschiebung der Stempel erst um 2 Uhr aufgenommen wird
            //UART_Rx_GPS(serial_port, Date); //Datum von GPS beziehen
            snprintf(UTC_Date, 11, "%02d.%02d.%d", tm.tm_mday, tm.tm_mon+1,
tm.tm_year+1900); //Datum von lokalem Gerät beziehen
    }
    return 0;

}
```

10.3 Python-Skript zum Upload in Dropbox Cloud

```
#!/usr/bin/python3

# -*- coding: utf-8 -*-
"""
Created on Fri Jan  21 15:01:17 2022

DropboxUpload.py
Skript zum Upload der Messergebnisse der Spannungsphasenwinkelmessung
in eine Dropbox Cloud zur weiteren Auswertung

Version 1.1 Stand 25.01.2022
Autor: Marius Kleinbach
Sämtliche lokalen Messergebnisse werden hochgeladen und danach aus /Mess-
werte gelöscht

VERSIONSGESCHICHTE
- Version 1.0 Stand 21.01.2022
Autor: Marius Kleinbach
Skript zum Upload der Messergebnisse der Spannungsphasenwinkelmessung
in eine Dropbox Cloud zur weiteren Auswertung

"""

import os
import dropbox
import datetime
db = dropbox.Dropbox('d-jHD8Y1RGMAAAAAAAAAAZ9hmEJ6IT9Nx-Aa_bmmL43zE8jeLuD-
tLHN3SP88D7M4')
filecount = 0

# existierende Dateien hochladen
files = os.listdir("/home/pi/Code/masterprojekt/CodeSpannungswinkel/Zeit-
stempel_v2/Messwerte")

print("scanning files...")
for file in files:
    print(file)
    fname = 'Messwerte/' + file  # local file name
    dname = '/Messwerte/' + file  # name of file in dropbox
    f = open(fname, 'rb')
```

```
    db.files_upload(f.read(), dname)
    f.close()
    os.remove("Messwerte/" + file)
    filecount += 1
print(str(filecount) + " Datei(en) in db hochgeladen")
```

10.4 Python-Skript zur Auswertung der Spannungsphasenwinkel

```
"""
Created on Mon Jan 31 19:08:42 2022

@author: Hotz

Änderungshistorie:

15.11.2022 Version 2.0
Filterung in 4 verschiedenen Stufen, um Ausreißer zu filtern und Messwerte
zu glätten
Marius Kleinbach
"""

import os
import csv
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np

class MeasurementTuple():
    # Touple of currently four measurement values on one PMU during one
second
    def __init__(self,pmu_index,timestamp,values):
        self.pmu_index = pmu_index # index uf PMU
        self.timestamp = timestamp # timestamp not including ms
        self.values    = values    # four µs-values

    def __str__(self):
        # report pmu-indexes, timestamp, values (measured delta_ts)
        print_str  = ''
        print_str += 'MeasurementTuple-Object\n'
        print_str += 'PMU-Index : '+str(self.pmu_index)+str('\n')
        print_str += 'timestamp : '+str(self.timestamp)+str('\n')
        print_str += 'values    : '+str(self.values)+str('\n')
        return print_str

class CombiEvaluation():
    # analysis of data for PMU-pair to get accuracy data and angle for one
    # second
    def __init__(self,tuple1,tuple2):
        self.tuples      = [tuple1, tuple2]
        self.pmu_indexes = [x.pmu_index for x in self.tuples]
        self.delta_t     = [x-y for x,y in zip(tuple1.values,tuple2.val-
ues)]
        self.avg         = sum(self.delta_t)/len(self.delta_t)
        #print("self.avg")
        #print(self.avg)
        self.rms_error   = (sum([(x-self.avg)**2 for x in self.delta_t])/
                            len(self.delta_t))**0.5
        #angles in degree
        self.angle       = 360*self.avg/(20e-3)
```

```
        self.angle_filter= self.angle
        #angles that are shifted by 1 period (360 deg) are corrected
        if self.angle_filter < -300:
            self.angle_filter = (self.angle_filter) * (-1) - 360
        elif self.angle_filter > 300:
            self.angle_filter = (self.angle_filter) - 360

        self.angle_error = 360*self.rms_error/(20e-3)

    def __str__(self):
        # report pmu_indexes, delta-t-values, rms-error, angle, rms-error-
angle
        print_str  = 'PMU-Combination '+str(self.pmu_indexes)+':\n'
        print_str += '   Δt [µs]:        '+str(round(self.avg*1e6,4))+' '
#avg
        print_str += str([round(x*1e6,4) for x in self.delta_t]) +'\n' #all
Δt
        print_str += '   RMS-err [µs]:   '+str(round(self.rms_error*1e6,3))
        print_str += '\n'+'   angle:          '+str(round(self.angle,2))
        print_str += '° ± '+str(round(self.angle_error,2))+'°\n'
        return print_str

class Event():
    # event class - has measurements for all PMUs during one specific sec-
ond
    # conducts statistical analysis of measurements for one timeframe
    # calculate angle
    def __init__(self,timestamp):
        self.timestamp = timestamp
        self.tuples    = []

    def get_pmu_index_combinations(self):
        # get combinations of pmus to evaluate all angles between any two
        # bussies
        combi_list=[]
        for tuple_1 in self.tuples:
            for tuple_2 in self.tuples:
                if tuple_2.pmu_index>tuple_1.pmu_index:
                    combi_list.append([tuple_1,tuple_2])
        return combi_list

    def evaluate_combinations(self):
        # goes through all combinations of busses for second and makes ma-
trix
        # and list of CombiEvaluation-objects
        combinations            = self.get_pmu_index_combinations()
        max_index               = max([x.pmu_index for x in self.tuples])
        self.evaluation_matrix  = [[None for x in range(max_index+1)]
                                   for y in range(max_index+1)]
        self.evaluation_list    = []

        for pmu_combination in combinations:
            combi_eval = CombiEvaluation(pmu_combination[0],
                                         pmu_combination[1])

            index_0=pmu_combination[0].pmu_index
            index_1=pmu_combination[1].pmu_index
            self.evaluation_matrix[index_0][index_1]=combi_eval
            self.evaluation_matrix[index_1][index_0]=combi_eval
            self.evaluation_list.append(combi_eval)

    def __str__(self):
        # report timestamp, pmus, call all evaluation __str__
```

```
        print_str  = 'Event at time '
        print_str += str(self.timestamp) + ', active PMUs: '
        print_str += str([x.pmu_index for x in self.tuples])+'\n'

        for evaluation in self.evaluation_list:
            print_str+=evaluation.__str__()+'\n'

        return print_str

class CsvParser():
    # parses all csv in directory and creates list of tuples, list is used
by
    # MeasurementEvaluation for further processing
    def __init__(self,directory):
        self.directory=directory
        self.file_list=os.listdir(directory)
        self.tuples=[]
        self.parse_directory()

    def parse_directory(self):
        # go through all csv-files w/ measurements in directory
        for file in self.file_list:
            with open(self.directory+'/'+file, newline='') as f:
                self.reader=csv.reader(f)
                measurement_items=[]
                measurement_item=[]
                for row in self.reader:
                # read lines of csv-file
                    measurement_item.append(row)
                    if len(measurement_item)>1:
                    # every other row is pmu-data and measurement, always
                    # combine two rows into one measurement_item then add
it
                    # to list measurement_items
                        measurement_items.append(measurement_item)
                        measurement_item=[]
                for measurement_item in measurement_items:
                # parse measurement_item (it's 2 lines of csv as one
string)

                    csv_str   = measurement_item[0][0]           #
date+time

                    values    = [self.interpret_number_string(x) # measure-
ments
                                 for x in measurement_item[1]]

                    PMU_index = int(file.split('_')[1].split('-')[0][3])

                    dmy       = [self.interpret_number_string(x)
                                 for x in csv_str.split(' ')[6].split('.')]

                    day       = dmy[0]
                    month     = dmy[1]
                    year      = dmy[2]

                    hms       = [self.interpret_number_string(x)
                                 for x in csv_str.split(' ')[8].split(':')]

                    hour      = hms[0]
                    minute    = hms[1]
                    second    = hms[2]
```

```
                    timestamp = datetime(year,month,day,hour,minute,second)

                    self.tuples.append(MeasurementTuple(pmu_index =PMU_in-
dex,
                                                        timestamp = timestamp,
                                                        values    = values))

    def interpret_number_string(self,arg_str):
        # exterminate leading zeros, avoid empty string, convert to number
        if arg_str in ['0','00']:
            return 0
        else:
            return eval(arg_str.lstrip('0'))

class MeasurementEvaluation():
    def __init__(self, directory):
        # top level evaluation class:
        #    - calls CsvParser to read all csvs
        #    - finds tuples with identical timestamp and creates event-ob-
jects
        self.parser=CsvParser(directory)
        self.tuples=self.parser.tuples
        self.get_events()

    def get_events(self):
        # creates event-objects, fills them with life and puts them into
list
        self.events=[]
        timestamp_list=[x.timestamp for x in self.tuples]
        timestamp_list=list(set(timestamp_list))
        timestamp_list.sort()

        for timestamp in timestamp_list:
            event=Event(timestamp)
            for tuple_item in self.tuples:
                if tuple_item.timestamp==timestamp:
                    event.tuples.append(tuple_item)
            event.evaluate_combinations()
            self.events.append(event)

    def __str__(self):
        # report events
        print_str  = 'Measurement Evaluation: '
        print_str += str(len(self.events))+'Events recorded'
        for event in self.events:
            print_str += event.__str__()
        return print_str


#########################################################################
###
# Definitions for plotting and filteringg
#########################################################################
###

def printAngles(evaluationSet):
    results = []
    for x in evaluationSet.events:
        for y in x.evaluation_list:
            if filter_level > 0:
                results.append(y.angle_filter)
                #print(y.angle_filter)
                y.angle_filter = 5
```

```
                #print(type(y))
            else:
                results.append(y.angle)
            #if y.angle < 0:
                #print(y.angle)
    plt.figure(dpi=300)
    #plt.ylim(top = 10, bottom = -)
    if plot_selected:
        plt.plot(results[selection_start:selection_stop])
        #if filter_level > 3:
            #plt.plot
    else:
        plt.plot(results)
        if filter_level > 3:
            print(len(results))
            plt.plot(range(len(results)), np.full(len(results), average(re-
sults)))

    #plt.yaxis.set_major_locator(matplotlib.ticker(integer=True))
    plt.title("Phasenwinkel Messreihe " + str(Messreihe))
    plt.grid()

def average(lst):
    return sum(lst) / len(lst)

def filter_angles(evaluationSet, series_no):

    filtered_angles = []

    for x in evaluationSet.events:
        for y in x.evaluation_list:
            filtered_angles.append(y.angle_filter)

    it_x = 0 # iterator x
    for x in filtered_angles:
        window_avg = 0
        for y in range(filter_window):
            try:
                window_avg += filtered_angles[it_x + y]
            except:
                window_avg += filtered_angles[it_x - y]
        filtered_angles[it_x] = window_avg / filter_window

        # skip first measurement point because it's usually inaccurate
        if filter_level > 2:
            if it_x == 1:
                filtered_angles[it_x-1] = filtered_angles[it_x]
                print (f"Messreihe {series_no} avg angle in {it_x} window
is {window_avg/filter_window}\n\
                Winkel am Messpunkt = {filtered_angles[it_x]}°")

        #print(f"Last filtered angle = {filtered_angles[-filter_window]}°")
        # Irgendwas stimmt nicht mit der Berechnung der (letzten)
window_avg Werte ?


        it_x += 1

    #self.rms_error   = (sum([(x-self.avg)**2 for x in self.delta_t])/
    #                    len(self.delta_t))**0.5

    # overwrite the old angle_filter elements
    it_y = 0 # iterator y
```

```
    for x in evaluationSet.events:
        for y in x.evaluation_list:
            y.angle_filter = filtered_angles[it_y]
            it_y += 1

################################################################################
####
# Parameter Definitions
################################################################################
####

filter_level = 4 # level of filter from 0 to 4
filter_window = 1
plot_selected = False
if plot_selected:
    selection_start = 2
    selection_stop = 70


for i in range (1, 6, 1):

    Messreihe = i

    me=MeasurementEvaluation(directory='Messreihe_'+str(Messreihe))
    if filter_level > 1:
        filter_angles(me, i)

    #print(me)

    printAngles(me)
```

10.5 Python-Skript zur Überprüfung der Messgenauigkeit

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 31 19:08:42 2022

@author: Hotz

Änderungshistorie:

23.01.2023 Version 2.1
Acht verschiedene analyze_setting Einstellungen um unterschiedliche Filter-
stufen zu plotten
Marius Kleinbach
24.12.2022 Version 2.0
Änderung der plot Darstellung und Filterung um Standardabweichung zu be-
stimmen
Marius Kleinbach
"""

import os
import csv
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np


class MeasurementTuple():
```

```
    # Touple of currently four measurement values on one PMU during one
second
    def __init__(self,timestamp,values):
        self.timestamp = timestamp # timestamp not including ms
        self.values    = values    # four µs-values
        self.evaluate()

    def evaluate(self):
        self.absolute_delta_t_µ       = [round((val-0.01-n*0.02)*1e6) for
n,val in enumerate(self.values)]
        self.absolute_delta_delta_deg = [x*360/20000 for x in self.abso-
lute_delta_t_µ]

        x=self.values
        self.relative_delta_t_µ       = [round(y*1e6) for y in [x[0]-
0.01,x[1]-x[0]-0.02,x[2]-x[1]-0.02,x[3]-x[2]-0.02]]
        self.relative_delta_delta_deg = [x*360/20000 for x in self.rela-
tive_delta_t_µ]


    def __str__(self):
        # report pmu-indexes, timestamp, values (measured delta_ts)
        print_str  = ''
        print_str += 'MeasurementTuple-Object\n\n'
        print_str += 'timestamp   : '+str(self.timestamp)+str('\n')
        print_str += 'values      : '+str(self.values)+str('\n\n')
        print_str += 'Δt[µs] rel  : '+str(self.rela-
tive_delta_t_µ)+str('\n')
        print_str += 'Δδ[°]  rel  : '+str(self.rela-
tive_delta_delta_deg)+str('\n\n')

        print_str += 'Δt[µs] abs  : '+str(self.abso-
lute_delta_t_µ)+str('\n')
        print_str += 'Δδ[°]  abs  : '+str(self.abso-
lute_delta_delta_deg)+str('\n')

        return print_str

class CsvParser():
    # parses all csv in directory and creates list of tuples, list is used
by
    # MeasurementEvaluation for further processing
    def __init__(self,directory):
        self.directory=directory
        self.file_list=os.listdir(directory)
        self.tuples=[]
        self.parse_directory()

    def parse_directory(self):
        # go through all csv-files w/ measurements in directory
        for file in self.file_list:
            with open(self.directory+'/'+file, newline='') as f:
                self.reader=csv.reader(f)
                measurement_items=[]
                measurement_item=[]
                for row in self.reader:
                # read lines of csv-file
                    measurement_item.append(row)
                    if len(measurement_item)>1:
                    # every other row is pmu-data and measurement, always
                    # combine two rows into one measurement_item then add
it
                    # to list measurement_items
```

```
                        measurement_items.append(measurement_item)
                        measurement_item=[]
                for measurement_item in measurement_items:
                # parse measurement_item (it's 2 lines of csv as one
string)

                    csv_str   = measurement_item[0][0]           #
date+time

                    values    = [self.interpret_number_string(x) # measure-
ments
                                 for x in measurement_item[1]]

                    dmy       = [self.interpret_number_string(x)
                                 for x in csv_str.split(' ')[6].split('.')]

                    day       = dmy[0]
                    month     = dmy[1]
                    year      = dmy[2]

                    hms       = [self.interpret_number_string(x)
                                 for x in csv_str.split(' ')[8].split(':')]

                    hour      = hms[0]
                    minute    = hms[1]
                    second    = hms[2]

                    timestamp = datetime(year,month,day,hour,minute,second)

                    self.tuples.append(MeasurementTuple(timestamp =
timestamp,
                                                        values    = values))

    def interpret_number_string(self,arg_str):
        # exterminate leading zeros, avoid empty string, convert to number
        if arg_str in ['0','00']:
            return 0
        else:
            return eval(arg_str.lstrip('0'))

class MeasurementEvaluation():
    def __init__(self, directory):
        # top level evaluation class:
        #    - calls CsvParser to read all csvs
        #    - finds tuples with identical timestamp and creates event-ob-
jects
        self.parser=CsvParser(directory)
        self.tuples=self.parser.tuples

    def __str__(self):
        # report events
        print_str  = 'Measurement Evaluation: '
        for item in self.tuples:
            print_str+=str(item)+'\n'
        return print_str

def plot_list(lst, title="default"):  # plots list with title
    plt.figure(dpi=300)
    plt.plot(lst)
    plt.plot(range(len(lst)), np.full(len(lst), average(lst))) # mean line
    plt.plot(range(len(lst)), np.full(len(lst), average(lst) +
                                      rmsd_of_list(lst)), "g")# high rmsd
line
```

```
    plt.plot(range(len(lst)), np.full(len(lst), average(lst) -
                                      rmsd_of_list(lst)), "g")# low rmsd
line
    plt.title(str(title))
    plt.grid()
    #print(average(lst)) # for Debugging
    #plt.show() # causes errors when run on raspberry
    if safe_plots_flag:
        if title.split()[6] == "absolute":
            i = 0
            while os.path.exists("plots/angle_deviation_abs_" + str(i) +
".png"):
                i += 1
            plt.savefig(fname = "plots/angle_deviation_abs_" + str(i) +
".png")
        elif title.split()[6] == "relative":
            i = 0
            while os.path.exists("plots/angle_deviation_rel_" + str(i) +
".png"):
                i += 1
            plt.savefig(fname = "plots/angle_deviation_rel_" + str(i) +
".png")
        else:
            print("Plot wasn't saved because of wrong title.")

def plot_lists(lst1, lst2, title="default"):  # plots list with title
    plt.figure(dpi=300)
    plt.plot(lst1, "b--", lw = 0.5, label = "Original data")
    #plt.plot(range(len(lst1)), np.full(len(lst1), average(lst1)), "g") #
mean line # Mittelwert Reihe 1
    plt.plot(lst2, "r", lw = 1.1, label = "Filtered data")
    #plt.plot(range(len(lst2)), np.full(len(lst2), average(lst2)), "y") #
mean line # Mittelwert Reihe 2
    plt.title(str(title))
    plt.grid()
    #print(f"Average lst1 {average(lst1)}, lst2 {average(lst2)}") # for De-
bugging
    #plt.show() # causes errors when run on raspberry
    if safe_plots_flag:
        if title.split()[6] == "absolute":
            i = 0
            while os.path.exists("plots/angle_deviation_abs_" + str(i) +
".png"):
                i += 1
            plt.savefig(fname = "plots/angle_deviation_abs_" + str(i) +
".png")
        elif title.split()[6] == "relative":
            i = 0
            while os.path.exists("plots/angle_deviation_rel_" + str(i) +
".png"):
                i += 1
            plt.savefig(fname = "plots/angle_deviation_rel_" + str(i) +
".png")
        else:
            print("Plot wasn't saved because of wrong title.")

def average(lst):
    return sum(lst) / len(lst)

def mean_of_set(evaluationSet): # mean value of set (absolute and relative
angle)
    mean_val = [0] * 8
    for set_no in range(4):
```

```
        it_x = 0 # iterator x
        for x in evaluationSet.tuples:
            if analyze_setting > 0: #1st Filter level
                if abs(x.absolute_delta_delta_deg[0])<10:
                    mean_val[set_no] += x.absolute_delta_delta_deg[set_no]
                    mean_val[set_no + 4] += x.rela-
tive_delta_delta_deg[set_no]
                    it_x += 1
            else:
                mean_val[set_no] += x.absolute_delta_delta_deg[set_no]
                mean_val[set_no + 4] += x.relative_delta_delta_deg[set_no]
                it_x += 1
        mean_val[set_no] = mean_val[set_no] / (it_x)
        mean_val[set_no + 4] = mean_val[set_no + 4] / (it_x)
        #print(f"{it_x} Werte")
    return mean_val

def rmsd_of_set(evaluationSet, avg_vals): # root mean square deviation
    rmsd_val = [0] * 8
    for set_no in range(4):
        it_x = 0 # iterator x
        for x in evaluationSet.tuples:
            if analyze_setting > 0: #1st Filter level
                if abs(x.absolute_delta_delta_deg[0])<10:
                    rmsd_val[set_no] += (x.abso-
lute_delta_delta_deg[set_no]-avg_vals[set_no])**2
                    rmsd_val[set_no + 4] += (x.rela-
tive_delta_delta_deg[set_no]-avg_vals[set_no + 4])**2
                    it_x += 1
            else:
                rmsd_val[set_no] += (x.absolute_delta_delta_deg[set_no]-
avg_vals[set_no])**2
                rmsd_val[set_no + 4] += (x.rela-
tive_delta_delta_deg[set_no]-avg_vals[set_no + 4])**2
                it_x += 1
        rmsd_val[set_no] = (rmsd_val[set_no] / it_x)**0.5
        rmsd_val[set_no + 4] = (rmsd_val[set_no + 4] / it_x)**0.5
        #print(f"{it_x} Werte")
    return rmsd_val

def rmsd_of_list(lst):
    avg_val = average(lst)
    rmsd_val = 0
    it_x = 0 # iterator x
    for x in lst:
        rmsd_val += (x - avg_val)**2
        it_x += 1
    rmsd_val = (rmsd_val / it_x)**0.5
    return rmsd_val

def filter_values(lst, wndw):
    lst_filtered = lst.copy()
    it_x = 0 # iterator x
    for x in lst_filtered:
        window_avg = 0
        for y in range(wndw):
            try:
                window_avg += lst_filtered[it_x + y]
            except:
                window_avg += lst_filtered[it_x - y]
                #print(f"Filter over at position {it_x + y}")
        lst_filtered[it_x] = window_avg / wndw
        it_x += 1
```

```
    return lst_filtered

def check_values_in_tolerance(lst, tol):
    outliers = 0
    for x in lst:
        if abs(x) > tol:
            outliers += 1
    print(f"Von {len(lst)} Werten liegen {outliers} über der Grenze von ±
{tol}")

me=MeasurementEvaluation(directory='csv_oszi')

##########################################################################
####
# Definitionen für Verion 2.0
if False: # übersichtlicher beim debuggen, nur Ausschnitt wird geplottet
    me.tuples=me.tuples[0:5]

safe_plots_flag = False # Option zum abspeichern der plots
check_window_size = False # Option zum Überprüfen der idealen Filergröße

analyze_setting = 4 # level of filter from 0 to 7
# Setting 1: Zeitreihe: Alle Werte für Winkel > 10° gelöscht
# Setting 2: Zeitreihe: Glättung über Fenstergröße
# Setting 3: Zeitreihe: Darstellung in selben Diagramm
# Setting 4: Zeitreihe: Betrachtung der Zeit delta t, anstatt dem Winkel
# Setting 5: Histogramm: Relative Zeitabweichung
# Setting 6: Histogramm: Absolute Abweichung vom Erwartungswert bestimmen
# Setting 7: Histogramm: Absolute Werte abzüglich Offset
# Setting 8: Zeitreihe: Datstellung im selben Diagramm
filter_window = 10 # Filter Fenstergröße. 1: keine Filterung

hist_height = 1200 # ylim of histogram
const_width = 1 # width of constant lines in histogram

# mean_value = mean_of_set(me) # outdated and inflexible
# rmsd_value = rmsd_of_set(me, mean_value) # outdated and inflexible

##########################################################################
####
# rmsd in Abhängigkeit der Filter Fenstergröße

if check_window_size:
    window_dependency = [[["window size", "mean value 1", "RMSD 1"]],
                         [["window size", "mean value 2", "RMSD 2"]],
                         [["window size", "mean value 3", "RMSD 3"]]]
    for index in range(1, 4):
        data=[x.relative_delta_delta_deg[index] for x in me.tuples if
abs(x.absolute_delta_delta_deg[0])<10]
        for window in range(1, 21): # alternate window size to get best re-
sults
            data = filter_values(data, window)
            window_dependency[index-1].append([window, average(data),
rmsd_of_list(data)])

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

#print(me)

#plot angles
mean_value_abs = [0] * 4
mean_value_rel = [0] * 4
```

```
rmsd_value_abs = [0] * 4
rmsd_value_rel = [0] * 4

if analyze_setting == 0:
    for index in range(4):
        data=[x.absolute_delta_delta_deg[index] for x in me.tuples]
        mean_value_abs[index] = average(data)
        rmsd_value_abs[index] = rmsd_of_list(data)
        plot_list(data, title='Nr '+str(index)+
                  ' Measurement of each tuple absolute differences degree')

    for index in range(4):
        data=[x.relative_delta_delta_deg[index] for x in me.tuples]
        mean_value_rel[index] = average(data)
        rmsd_value_rel[index] = rmsd_of_list(data)
        plot_list(data, title='Nr '+str(index)+
                  ' Measurement of each tuple relative differences degree')

elif analyze_setting == 1:
    for index in range(4):
        data=[x.absolute_delta_delta_deg[index] for x in me.tuples if
abs(x.absolute_delta_delta_deg[0])<10]
        mean_value_abs[index] = average(data)
        rmsd_value_abs[index] = rmsd_of_list(data)
        plot_list(data, title='Nr '+str(index)+
                  ' Measurement of each tuple absolute differences degree')

    for index in range(4):
        data=[x.relative_delta_delta_deg[index] for x in me.tuples if
abs(x.relative_delta_delta_deg[0])<10]
        mean_value_rel[index] = average(data)
        rmsd_value_rel[index] = rmsd_of_list(data)
        plot_list(data, title='Nr '+str(index)+
                  ' Measurement of each tuple relative differences degree')

elif analyze_setting == 2:
    for index in range(4):
        data=[x.absolute_delta_delta_deg[index] for x in me.tuples if
abs(x.absolute_delta_delta_deg[0])<10]
        data_filtered = filter_values(data, filter_window)
        mean_value_abs[index] = average(data_filtered)
        rmsd_value_abs[index] = rmsd_of_list(data_filtered)
        plot_list(data_filtered, title='Nr '+str(index)+
                  ' Measurement of each tuple absolute differences degree')

    for index in range(4):
        data=[x.relative_delta_delta_deg[index] for x in me.tuples if
abs(x.relative_delta_delta_deg[0])<10]
        data_filtered = filter_values(data, filter_window)
        mean_value_rel[index] = average(data_filtered)
        rmsd_value_rel[index] = rmsd_of_list(data_filtered)
        plot_list(data_filtered, title='Nr '+str(index)+
                  ' Measurement of each tuple relative differences degree')

elif analyze_setting == 3:
    for index in range(4):
        data=[x.absolute_delta_delta_deg[index] for x in me.tuples if
abs(x.absolute_delta_delta_deg[0])<10]
        data_filtered = filter_values(data, filter_window)
        mean_value_abs[index] = average(data_filtered)
        rmsd_value_abs[index] = rmsd_of_list(data_filtered)
        plot_lists(data, data_filtered, title='Nr '+str(index)+
                  ' Measurement of each tuple absolute differences degree')
```

```
    for index in range(4):
        data=[x.relative_delta_delta_deg[index] for x in me.tuples if
abs(x.relative_delta_delta_deg[0])<10]
        data_filtered = filter_values(data, filter_window)
        mean_value_rel[index] = average(data_filtered)
        rmsd_value_rel[index] = rmsd_of_list(data_filtered)
        plot_lists(data, data_filtered, title='Nr '+str(index)+
                   ' Measurement of each tuple relative differences degree')

elif analyze_setting == 4:
    for index in range(4):
        data=[x.relative_delta_t_µ[index] for x in me.tuples if abs(x.rela-
tive_delta_t_µ[0])<556]
        data_filtered = filter_values(data, filter_window)
        plot_list(data_filtered, title='Nr '+str(index)+
                   ' Measurement of each tuple relative differences delta
t')

elif analyze_setting == 5:
    if filter_window < 5:
        hist_height = 700
    elif filter_window < 15:
        hist_height = 500
        const_width = 0.8
    elif filter_window < 30:
        hist_height = 350
        const_width = 0.5
    elif filter_window < 50:
        hist_height = 300
        const_width = 0.35
    else:
        hist_height = 250
        const_width = 0.25
    data = []
    data_filtered = [0] * 3
    mean_total = [0] * 3
    rmsd_total = [0] * 3
    for index in range(1, 4):
        data.append([x.relative_delta_t_µ[index] for x in me.tuples if
abs(x.relative_delta_t_µ[0])<556])
        data_filtered[index-1] = filter_values(data[index-1], filter_win-
dow)
        mean_total[index-1] = average(data_filtered[index-1])
        rmsd_total[index-1] = rmsd_of_list(data_filtered[index-1])
    avg_data = [average(mean_total)] * 700
    rmsd_low = [average(mean_total) - average(rmsd_total)] * 700
    rmsd_high = [average(mean_total) + average(rmsd_total)] * 700
    plt.figure(dpi=300)
    plt.hist(data_filtered, 50, histtype = "barstacked", rwidth = 0.82,
color = ["b","b","b"],
             label="Δt[µs]")
    plt.hist(avg_data, 1, rwidth = const_width, color = "r", label =
             f"Average = {'{:.2f}'.format(round(avg_data[0], 2))} µs") #
avg line
    plt.hist(rmsd_low, 1, rwidth = const_width, color = "g", label =
             f"Δσ = {'{:.2f}'.format(round(average(rmsd_total), 2))} µs") #
rmsd line
    plt.hist(rmsd_high, 1, rwidth =const_width, color = "g")
    plt.hist([16.67] * 700, 1, rwidth = const_width, color = "m", label =
"16,67 µs ≙ 0,3°")
    plt.hist([-16.67] * 700, 1, rwidth = const_width, color = "m")
```

```
    if hist_height:
        plt.ylim(0, hist_height)
    #plt.title(f"Messabweichung mit Filter-Fenstergröße = {filter_window}")
    plt.title("Measurement of interrupt relative time difference")
    plt.legend()
    plt.rc('axes', axisbelow=True)
    plt.grid()
    plt.xlabel("Deviation from expected value [µs]")
    plt.ylabel(f"Number of measurement points
({len(data[0]+data[1]+data[2])} in total)")
    if safe_plots_flag:
        try:
            i = 0
            while os.path.exists(f"plots/time_deviation_filter_size{fil-
ter_window}_{i}.png"):
                i += 1
            plt.savefig(fname = f"plots/time_deviation_filter_size{fil-
ter_window}_{i}.png")
        except:
            print("Plot wasn't saved because of wrong title.")
    all_values = []
    for index in range(3):
        all_values.extend(data_filtered[index])
    check_values_in_tolerance(all_values, 50)

elif analyze_setting == 6:
    if filter_window < 6:
        hist_height = 1200
    elif filter_window < 11:
        hist_height = 1000
    elif filter_window < 20:
        hist_height = 800
    elif filter_window < 30:
        hist_height = 600
    elif filter_window < 50:
        hist_height = 350
        const_width = 0.5
    else:
        hist_height = 250
        const_width = 0.25
    data = []
    data_filtered = [0] * 4
    mean_total = [0] * 4
    rmsd_total = [0] * 4
    for index in range(4):
        data.append([x.absolute_delta_t_µ[index] for x in me.tuples if
abs(x.absolute_delta_t_µ[0])<556])
        data_filtered[index] = filter_values(data[index], filter_window)
        mean_total[index] = average(data_filtered[index])
        rmsd_total[index] = rmsd_of_list(data_filtered[index])
    avg_data = [average(mean_total)] * 1200
    rmsd_low = [average(mean_total) - average(rmsd_total)] * 1200
    rmsd_high = [average(mean_total) + average(rmsd_total)] * 1200
    plt.figure(dpi=300)
    plt.hist(data_filtered, 50, histtype = "barstacked", rwidth = 0.82,
color = ["b","b","b","b"],
             label="Δt[µs]")
    plt.hist(avg_data, 1, rwidth = const_width, color = "r", label =
             f"Average = {'{:.2f}'.format(round(avg_data[0], 2))} µs") #
avg line
    if hist_height:
        plt.ylim(0, hist_height)
```

```
    #plt.title(f"Absolute Messabweichung mit Filter-Fenstergröße = {fil-
ter_window}")
    plt.title("Measurement of interrupt absolute time difference")
    plt.legend()
    plt.rc('axes', axisbelow=True)
    plt.grid()
    plt.xlabel("Deviation from expected value [µs]")
    plt.ylabel(f"Number of measurement points
({len(data[0]+data[1]+data[2]+data[3])} in total)")
    if safe_plots_flag:
        try:
            i = 0
            while os.path.exists(f"plots/time_deviation_filter_size{fil-
ter_window}_{i}.png"):
                i += 1
            plt.savefig(fname = f"plots/time_deviation_abs_filter_size{fil-
ter_window}_{i}.png")
        except:
            print("Plot wasn't saved because of wrong title.")

elif analyze_setting == 7:
    data = []
    data_filtered = [0] * 4
    mean_total = [0] * 4
    rmsd_total = [0] * 4
    mean_with_offset = [0] * 4
    for index in range(4):
        data.append([x.absolute_delta_t_µ[index] for x in me.tuples if
abs(x.absolute_delta_t_µ[0])<556])
        data_filtered[index] = filter_values(data[index], filter_window)
        mean_with_offset[index] = average(data_filtered[index])
    offset = average(mean_with_offset)
    data_no_offset = []
    for index in range(4):
        data_no_offset.append([x-offset for x in data_filtered[index]])
        mean_total[index] = average(data_no_offset[index])
        rmsd_total[index] = rmsd_of_list(data_no_offset[index])
    avg_data = [average(mean_total)] * 1200
    rmsd_low = [average(mean_total) - average(rmsd_total)] * 1200
    rmsd_high = [average(mean_total) + average(rmsd_total)] * 1200
    plt.figure(dpi=300)
    plt.hist(data_no_offset, 50, histtype = "barstacked", rwidth = 0.82,
color = ["b","b","b","b"],
             label="Δt[µs]")
    plt.hist(avg_data, 1, rwidth = const_width, color = "r", label =
             f"Mittelwert = {'{:.2f}'.format(round(avg_data[0], 2))} µs") #
avg line
    plt.hist(rmsd_low, 1, rwidth = const_width, color = "g", label =
             f"Δσ = {'{:.2f}'.format(round(average(rmsd_total), 2))} µs") #
rmsd line
    plt.hist(rmsd_high, 1, rwidth =const_width, color = "g")
    plt.hist([16.67] * 1200, 1, rwidth = const_width, color = "m", label =
"16,67 µs ≙ 0,3°")
    plt.hist([-16.67] * 1200, 1, rwidth = const_width, color = "m")
    if hist_height:
        plt.ylim(0, hist_height)
    plt.title(f"Messabweichung ohne Offset mit Filter-Fenstergröße = {fil-
ter_window}")
    plt.legend()
    plt.rc('axes', axisbelow=True)
    plt.grid()
    plt.xlabel("Abweichung vom Erwartungswert [µs]")
```

```
    plt.ylabel(f"Anzahl der Messpunkte (insgesamt
{len(data[0]+data[1]+data[2]+data[3])})")
    if safe_plots_flag:
        try:
            i = 0
            while os.path.exists(f"plots/time_deviation_filter_size{fil-
ter_window}_{i}.png"):
                i += 1
            plt.savefig(fname = f"plots/time_deviation_filter_size{fil-
ter_window}_{i}.png")
        except:
            print("Plot wasn't saved because of wrong title.")

elif analyze_setting == 8:
    data = []
    data_filtered = [0] * 3
    for index in range(1, 4):
        data.append([x.relative_delta_t_µ[index] for x in me.tuples if
abs(x.relative_delta_t_µ[0])<556])
        data_filtered[index-1] = filter_values(data[index-1], filter_win-
dow)
    data = [item for sublist in data for item in sublist]
    data_filtered = [item for sublist in data_filtered for item in sublist]
    plot_lists(data, data_filtered, title='Measurement of each data point
relative differences Δt')
    plt.xlabel(f"Index of measurement points ({len(data_filtered)} in to-
tal)")
    plt.ylabel("Deviation from expected value [µs]")
    plt.legend()


else:
    print(f"Filter Level {analyze_setting} not defined yet")
```