Entwicklung eines universellen Visualisierungs- und Datenverwaltungstools für asynchrone Messwerte
Ich, Markus Huber, versichere an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel *Entwicklung eines universellen Visualisierungs- und Datenverwaltungstools für asynchrone Messwerte* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, den 1. Oktober 2014

Markus Huber
Inhaltsverzeichnis

1. Einleitung 1

2. Stand der Technik und Grundlagen 2
   2.1. Übliche Messprogramme 2
   2.2. LabVIEW 4
   2.3. TDMS Format 7
   2.4. CSV Format 7

3. Präzisierung der Aufgabenstellung 8

4. Grobkonzept 9
   4.1. Treiber 10
      4.1.1. Sensoren 10
      4.1.2. Ablauf 12
   4.2. Bearbeitung 12
      4.2.1. Bearbeitungsschritte 13
   4.3. Regler 13
      4.3.1. Aktoren 14
      4.3.2. Ablauf 14
   4.4. Visualisierung 14
   4.5. Konfigurierung 15
   4.6. Aufzeichnung 15
   4.7. Wiedergabe 15

5. Feinkonzept 16
   5.1. Treiber 16
      5.1.1. Datumsformat 16
      5.1.2. Signalarten 17
   5.2. Bearbeitung 17
      5.2.1. Parameterinhalt 17
      5.2.2. Benutzerinterface 18
      5.2.3. Ablaufgenerierung 18
      5.2.4. Modulansteuerung 18
      5.2.5. Kontrolle der Bearbeitungsschritte 19
   5.3. Visualisierung 21
      5.3.1. Schnittstelle 21
   5.4. Konfigurierung 21
   5.5. Aufzeichnung und Wiedergabe 22
      5.5.1. Dateiformat 22
Abbildungsverzeichnis

2.1. Tool Einsatzgebiet .............................................. 3
2.2. LabVIEW Beispielprogramm .................................... 5
2.3. XY Diagramm .................................................... 5
2.4. Bargraph ....................................................... 6
2.5. Tacho .......................................................... 6
2.6. TDMS-Dateistruktur ............................................. 7
2.7. CSV-Dateistruktur .............................................. 7
4.1. Grobstruktur .................................................. 10
4.2. Beispielmessung ............................................... 11
4.3. Beispiel Bearbeitungsablauf .................................. 13
4.4. Beispiel Synchronisierung im Bearbeitungsablauf ........... 14
5.1. Datenfluss Beispiel ........................................... 19
5.2. Thread Beispiel ................................................ 20
5.3. GUI Schema ................................................... 21
6.1. Klassenstruktur ............................................... 23
6.2. Operationszustandsautomat .................................... 26
6.3. Operationsarten ............................................... 28
6.4. GUI ........................................................... 31
7.1. Messreihen Geschwindigkeitstest ............................ 33
7.2. Skalierung Geschwindigkeitstest ............................. 34
A.1. UML Struktur .................................................. 37
1. Einleitung

In der Firma WEINMANN Emergency Medical Technology GmbH + Co. KG (Weinmann) wird während der Entwicklung eines Gerätes die Funktionsfähigkeit des Gerätes durch Versuche geprüft und nachgewiesen. Für diese Versuche werden häufig Messungen aufgenommen und ausgewertet.

Auch in der Serienfertigung muss die Funktionstüchtigkeit der einzelnen Geräte in einer Endprüfung nachgewiesen werden. Dazu werden Messungen gemacht, die den Messungen in der Entwicklung sehr ähnlich sind. Sowohl für die Tests in der Entwicklung, als auch für die Endprüfung wird Software geschrieben, die die Messungen steuert. Durch die Ähnlichkeit der Messung ist auch die Software sehr ähnlich. Da die in der Entwicklung geschriebenen Messprogramme jedoch nicht den Qualitätsanforderungen der Endprüfung genügen, müssen dafür bisher komplett neue Programme geschrieben werden.

Durch ein gemeinsam benutztes Softwaretool soll die während der Entwicklung geschriebene Software mit möglichst wenig Anpassungen für die einzelnen Testfälle der Endprüfung verwendet werden können. Dieses Tool soll dem Benutzer automatisch ein Grundgerüst für das Programmieren von Messprogrammen zur Verfügung stellen. So soll das Tool asynchrone Daten unterstützen.
2. Stand der Technik und Grundlagen

2.1. Übliche Messprogramme


**Offset Korrektur** Bereinigen von Daten um einen Offset, in dem ein konstanter Wert auf die Daten addiert wird.

**Skalierung** Skalieren von Daten, in dem sie mit einem konstanten Faktor multipliziert werden.

**Linearisierung** Anpassen von Daten, die von einem Sensor mit einer nichtlinearen Kennlinie aufgenommen wurden. Hierbei wird die inverse Kennlinie auf die Daten angewendet.

**Addition** Addieren von mehreren Daten miteinander.

**Multiplikation** Multiplizieren mehrerer Daten miteinander.

**Filter** Filtern der Daten. Besonders häufig werden Tiefpassfilter verwendet.


Nach der Verarbeitung werden die Daten in verschiedenen Diagrammen visualisiert. Die Anzeigemöglichkeiten sind vielfältig, nachfolgend sind die häufigsten aufgelistet:

**Kurvendarstellung** Die Daten werden in einem Koordinatensystem angezeigt. In der Anzeige bleibt eine Historie vergangener Daten erhalten, so dass eine Kurve mit den aktuellen Daten entsteht. Hierbei sind zum Beispiel Trends gut ablesbar.
**Text** Der aktuelle Wert wird als Text ausgegeben. Manchmal werden mehrere Textfelder verwendet um eine Historie der letzten Daten anzeigen zu können. Diese Darstellung ähnelt der eines Protokolls.

**Bargraph** Die Länge eines Balkens zeigt das letzte Datum an. Für eine Historie können vergangene Daten transparenter angezeigt werden. Statt eines Balkens kann zum Beispiel auch eine Tachonadel angezeigt werden.

Mit Hilfe der Anzeige der Daten kann der Benutzer direkt in die Messung eingreifen. Um die Daten längerfristig aufzubewahren werden diese üblicherweise in Dateien gespeichert. Die verwendeten Dateiformate sind sehr unterschiedlich. Häufig wird in Comma-separated values (CSV) Dateien geschrieben, die zum Beispiel in Excel bearbeitet und betrachtet werden können. Auch das TDMS-Dateiformat wird verwendet.

Eine Übersicht über das Einsatzgebiet des Tools ist in Abbildung 2.1 gezeigt.

Abbildung 2.1.: Tool Einsatzgebiet
2.2. LabVIEW


Das Blockdiagramm ist in der Abbildung links gezeigt. In ihm ist der eigentliche Programmc ode hinterlegt. Unter Verwendung der Bedienelemente können hier die Anzeigeelemente angesteuert werden. Das Blockdiagramm kann auch weitere VIs enthalten, wie die meisten Programmiersprachen den Aufbau eines Programms aus mehreren Funktionen zulassen. Um auf das Frontpanel der sogenannten Sub-VIs zuzugreifen, muss das Sub-VI die Anzeige- und Bedienelemente an seine Anschlussfläche ausführen. Durch verdrahten dieser Anschlussfläche wird der verdrahtete Wert dem Sub-VI entweder als Eingabe übergeben, oder als Rückgabe gesetzt.

In dem Frontpanel des Beispielprogrammes gibt es zwei Bedienelemente, x und y, sowie ein Anzeigeelement x+y. In dem Blockdiagramm sind die beiden Bedienelemente mit den Eingängen der Sub-VI Addieren verbunden. Die Ausgabe der Sub-VI ist mit dem Anzeigeelement verdrahtet. Werden in die Anzeigeelemente beliebige Werte geschrieben und das VI ausgeführt, erscheint im Anzeigeelement die Summe der beiden Werte.

Zur Visualisierung bietet LabVIEW diverse Anzeigeelemente. Das einfachste, die Textanzeige, wurde im eben erläuterten Beispiel zur Ausgabe der Summe benutzt. Die am häufigsten benutzten Diagramme sind in den Abbildungen 2.3, 2.4 und 2.5 gezeigt.
Abbildung 2.2.: LabVIEW Beispielprogramm. Links ist das Blockdiagramm und rechts das Frontpanel gezeigt.

Abbildung 2.3.: Zeigt den Verlauf von Werten in Form von Kurven an.
2. Stand der Technik und Grundlagen


Um Inhalte im Frontpanel dynamisch zu verändern, gibt es in LabVIEW Subpanels. In sie können Frontpanels anderer VIs geladen werden. So ist es möglich SubVIs in das HauptVI einzubetten.
2.3. TDMS Format


2.4. CSV Format


```
; Spalte1 ; Spalte2
Zeile1 ; Zelle11 ; Zelle12
Zeile2 ; Zelle21 ; Zelle22
```

Abbildung 2.7.: Struktur von CSV Dateien.
3. Präzisierung der Aufgabenstellung

Im Abschnitt Übliche Messprogramme des Kapitels Stand der Technik und Grundlagen wurde der Typische Aufbau der Messprogramme beschrieben. Es soll ein Tool entwickelt werden, das die Programmierung dieser Messprogramme vereinfacht. Eine generische Anbindung für alle Sensoren zu programmieren, ist aufwendig und übersteigt den Rahmen dieser Arbeit. Die Anbindung eines konkreten Sensors gestaltet sich dank der umfassenden Bibliotheken von LabVIEW zur Messdatenerfassung einfach. Damit die Sensoren in das Tool eingebunden werden können, muss also eine Schnittstelle gefunden werden, mit der so angebundene Sensoren ihre Daten an das Tool übergeben können.

Die Sensordaten werden zum Teil nicht regelmäßig empfangen. Dies ist zum Beispiel bei Benutzeroingaben der Fall. Das Tool muss auch für solche Daten eine Schnittstelle anbieten.

Daten werden von den Sensoren teilweise mit einer Geschwindigkeit von bis zu 10 kHz erfasst. Das Tool soll die Datenerfassung nicht ausbremsen, daher muss es eine Verarbeitung der Daten mit diesen Geschwindigkeiten unterstützen.


Es ergeben sich folgende Aufgaben:

- Festlegung eines Grobkonzeptes für die allgemeine Messaufgabe.
- Erarbeiten mehrerer Feinkonzepte und Entscheidung für ein Feinkonzept.
- Implementierung eines Feinkonzeptes.
- Validierung wesentlicher Bestandteile der Implementierung.
4. Grobkonzept

Für die im Abschnitt 2.1 beschriebenen Messprogramme soll eine allgemeine Grobstruktur gefunden werden. Dafür werden voneinander unabhängige Aufgaben identifiziert, die in jedem Messprogramm vorkommen. Diese Aufgaben sind folgend aufgelistet.

- Die Sensoren werden konfiguriert und deren Daten erfasst.
- Die Daten werden verarbeitet.
- Die Verarbeitungsergebnisse werden zum Ansteuern von Aktoren verwendet.
- Die verarbeiteten Daten werden in einem Diagramm visualisiert.
- Benutzereingaben werden zum Konfigurieren des Tools genutzt.
- Die Daten werden in Dateien gespeichert.
- Daten werden aus Dateien gelesen.

Jede der Aufgaben entspricht einem Modul in der Grobstruktur. Die Module sind im Folgenden kurz benannt.

**Treiber** Erfassung von Messdaten, die zum Beispiel ein Sensor liefert. Treiber werden im Zuge der Arbeit nicht realisiert, aber die Schnittstelle definiert und mit einer Demo verifiziert.

**Bearbeitung** Operieren auf den Daten, zum Beispiel Skalieren der Werte um einen Faktor.

**Regler** Steuert die Aktoren in Abhängigkeit von den Eingangsdaten an.

**Visualisierung** Aufbereitung der Daten zur Anzeige in der GUI.

**Konfigurierung** Verarbeiten von Benutzereingaben zum Konfigurieren der Bearbeitung.

**Aufzeichnung** Schreiben von Daten in Dateien.

**Wiedergabe** Lesen von Daten aus Dateien.


Häufig werden genau die Daten angezeigt, die abgespeichert werden. Prinzipiell können die vom Treiber erfassten Rohdaten aber auch direkt in eine Datei gespeichert werden und weiterverarbeitete Daten werden angezeigt. Zum Beispiel könnten die Rohdaten Strom und Spannung in eine Datei geschrieben werden und das Produkt Strom wird angezeigt. Die Visualisierung muss grundsätzlich also unabhängig von der Aufzeichnung sein.

Im Folgenden werden die Module ausführlich beschrieben.
4.1. Treiber

Der Treiber ist für das Tool eine Datenquelle. Er stellt die Verbindung zwischen dem Programm und einem physikalisch vorhandenem Sensor her. Sie übernehmen das Konfigurieren und die Datenerfassung von den Sensoren.

4.1.1. Sensoren

Messung beendet werden soll.

Die Sensoren messen immer numerische Werte, zu einer bestimmten Zeit. Die empfangenen Daten sind also Zeit-Wert Paare. Sensoren erfassen die Daten mit bis zu 10 kHz.


Abbildung 4.2.: Ein Sinussignal wird gleichmäßig abgetastet gemessen. Die Amplitude wird ungleichmäßig geändert
4. Grobkonzept

4.1.2. Ablauf

Da die Treiber die Sensoren steuern, können im Treiber die entsprechenden Gegenstücke zu den Zuständen der Sensoren gefunden werden.


Die im Zustand laufend empfangene Daten sind im Allgemeinen zu unterschiedlichen Zeitpunkten und mit unterschiedlicher Frequenz aufgenommen. Aufgabe der Treiber ist es, die asynchronen Daten an die Bearbeitung weiterzuleiten. Um eine direkte Weiterverarbeitung und Live-Visualisierung zu ermöglichen, werden die Daten, sobald sie im Treiber erfasst wurden, einzeln an die Bearbeitung übergeben. Die Bearbeitung erhält also bereits Daten, obwohl die Treiber noch weitere Daten erfassen.

4.2. Bearbeitung


Die Ausgabe der Bearbeitung kann, je nach Eingangsdaten und Verarbeitung, asynchron sein. Diese asynchronen Daten werden an die Visualisierung und die Aufzeichnung weitergeleitet.

Die Bearbeitung soll die Aufzeichnung und die Visualisierung getrennt voneinander ansprechen können. Das hat den Vorteil, dass eine Messung auch ohne das Speichern in eine Datei angezeigt werden kann. Außerdem kann die Bearbeitung dann Rohdaten
speichern und unabhängig davon die verarbeiteten Daten anzeigen lassen. Der Benutzer hat dann also mehr Möglichkeiten, die Messung an seine Bedürfnisse anzupassen.

4.2.1. Bearbeitungsschritte


![Diagramm der Bearbeitungsschritte](visualisiertes_Diagramm)


4.3. Regler

Der Regler ist für das Tool eine Datensenke. Er ist die Schnittstelle von dem Tool zu einem Aktor. Der Aktor wird abhängig von dem Eingangssignal angesteuert. Das Eingangssignal des Reglers kann ungleichmäßig sein. Der Regler muss also auch mit ungleichmäßigen Eingangssignalen umgehen können. Um mehrere Akteoren kombinieren zu können müssen diese unabhängig voneinander durch die Bearbeitung ansprechbar sein. Die Signale für die Regler sind im Allgemeinen asynchron zueinander.
4. Grobkonzept


4.3.1. Akteure


4.3.2. Ablauf

Um die Akteure anzusprechen haben die Regler einen entsprechenden Ablauf. Sie fordern bei Start des Programmes Ressourcen an und initialisieren den Akteur. Im Betrieb wird der Akteur basierend auf den Eingangswerten angesteuert. Wie im Treiber ist auch im Akteur ein pausieren sinnvoll, damit der Benutzer Einstellungen ändern kann, wenn der Akteur nicht das gewünschte Verhalten hat. Bei Beendigung des Programmes werden die Akteure in eine Ausgangsstellung gebracht und die angeforderten Ressourcen freigegeben.

4.4. Visualisierung

Um schon während der Aufnahme die Messung beobachten zu können, soll die Visualisierung Daten Live, also während der Erfassung, anzeigen. Das ist besonders nützlich um Fehler, wie falsche Einstellungen, frühzeitig zu erkennen und zu beheben. Vor der eigentlichen Aufnahme wird daher häufig die Messung ohne das Abspeichern in eine Datei gestartet um Filterkoeffizienten einzustellen. Deshalb muss die Visualisierung unabhängig von Dateien funktionieren. Typischerweise werden die eingehenden Daten in ein Koordi-
natensystem als Kurve geplottet, es gibt aber diverse Visualisierungsmöglichkeiten. Die Visualisierung muss also eine Schnittstelle bereitstellen um beliebige Diagramme in der GUI anzuzeigen.

Von der Bearbeitung werden im Allgemeinen asynchrone Daten erhalten. Diese werden in ein Diagramm gelegt, das in der GUI dem Benutzer angezeigt wird. Die asynchronen Daten werden in der GUI synchronisiert angezeigt. Die Synchronisation geschieht im Betriebssystem automatisch und muss nicht gesondert implementiert werden.

4.5. Konfigurierung

Um Einstellungen an der Bearbeitung vornehmen zu können, muss der Benutzer eine Möglichkeit zum Anzeigen und Ändern der aktuellen Einstellungen haben. Hier kann der Benutzer zum Beispiel Filterkoeffizienten ändern, oder die Offsetkorrektur anpassen. Änderungen an diesen Einstellungen sollen zur Laufzeit übernommen werden, so dass der Benutzer die Auswirkung seiner Änderungen direkt beobachten kann.

4.6. Aufzeichnung


4.7. Wiedergabe

Um die gespeicherten Messungen zu betrachten, sollen die in der Aufzeichnung gespeicherten Daten wieder ausgewiesen werden können. Die Daten werden nicht in Echtzeit erfasst, sondern liegen bereits vor und müssen nur nachgeschlagen werden. Wenn der Benutzer zum Beispiel nur einen kleinen Abschnitt der Datei betrachten will, soll dieser in der Datei gezielt gesucht werden können.
5. Feinkonzept

In diesem Kapitel werden Konzepte für die Umsetzung der Bestandteile und Aufgaben der zuvor identifizierten Module vorgestellt. Nach den folgenden Kriterien wird dann das Feinkonzept ausgewählt:

**Umsetzbarkeit** Die Konzepte müssen so gewählt sein, dass sie in der Bachelorarbeit umsetzbar sind.

**Performance** Da die Daten mit bis zu 10 kHz erfasst werden, müssen auch die einzelnen Module diese Geschwindigkeit unterstützen.

**Speichereffizienz** Sowohl die Größe der Dateien auf der Festplatte, als auch die der Daten im Arbeitsspeicher, soll möglichst klein gehalten werden.

**Bedienbarkeit** Das Tool soll möglichst einfach Bedienbar sein. Das ist für die Akzeptanz durch den Anwender besonders wichtig.

5.1. Treiber

5.1.1. Datumsformat


Da alle Messwerte numerisch sind, ist die Verwendung des LabVIEW-Double, eine Gleitkommazahl mit doppelter Präzision gemäß IEEE 754 [1], sinnvoller. Dies ist der am häufigsten vorkommende Messwert. Andere numerische Datentypen, wie Integer, können

5.1.2. Signalarten


Wird nicht zwischen mehreren Signalarten unterschieden, wird nur das allgemeinere ungleichmäßige Signal implementiert. Da es dann nur eine Einzige Datenaustauschmöglichkeit gibt, müssen die Module auch nur diese verstehen. Die Modulschnittstelle und die Module vereinfachen sich dadurch. Zu jedem Wert muss ein Zeitstempel abgespeichert werden, auch wenn die zeitliche Verteilung der Daten durch einen einfachen Zusammenhang gegeben ist.


5.2. Bearbeitung

5.2.1. Parameterinhalt


Wenn der Parameter nur aus den Daten bestehen, verringert sich der Programmieraufwand. Allerdings muss durch die fehlende Kommunikationsmöglichkeit auch bei be trachtchen eines kleinen Ausschnitts einer Datei die gesamte Datei geladen werden.

5.2.2. Benutzerinterface

Um möglichst einfach neue Bearbeitungsschritte implementieren zu können und den Bearbeitungsschritten eine klare Struktur zu geben, soll dem Benutzer ein Interface geboten werden. In dem Interface soll Funktionalitäten, die in Bearbeitungsschritten häufig vorkommt zusammengefasst werden. Durch erstellen eines solchen Interfaces kann der Benutzer auf ein Grundgerüst gewarteter Komponenten zurückgreifen. Um die Erweiterung noch einfacher zu gestalten, sollen auch weitere Interfaces bereitgestellt werden, die zum Beispiel die Implementierung von Bearbeitungsschritten mit genau einem Eingangssignal und einem Ausgangssignal vereinfachen. Häufig wiederkehrende Funktionalität sind:

- Konfigurieren durch den Benutzer,
- Zugriff auf Ein- und Ausgabesignale und
- Ablaufkontrolle.

Mit der Ablaufkontrolle ist gemeint, dass die Bearbeitungsschritte in Phasen, wie die Allokation von Ressourcen, aufgeteilt werden können, deren Ablauf durch das Interface gesteuert wird.

5.2.3. Ablaufgenerierung

Der Benutzer muss dem Bearbeitungsmodul die auszuführenden Bearbeitungsschritte und ihre Reihenfolge vorgeben können.


Da das Ziel dieser Arbeit das Hinzufügen von Funktionen zur Messtechnik zu LabVIEW und nicht das Programmieren einer eigenen Oberfläche ist, wird der Bearbeitungsablauf zur Kompilationszeit von LabVIEW erzeugt. Der Benutzer gibt den Bearbeitungsablauf vor, in dem er VIs in einem Blockdiagramm miteinander verdrahtet. Das entspricht der üblichen LabVIEW Programmierung. Der Benutzer muss sich also nicht um gewöhnen. Die Funktionalität zur Beschreibung eines Datenflusses wird von LabVIEW übernommen und muss nicht neu implementiert werden.

5.2.4. Modulansteuerung

Der Benutzer muss im Bearbeitungsablauf auf die Eingaben, wie Treiber und Wiedergabe zugreifen können. Außerdem muss der Benutzer die Übergabe von Daten an einen Regler, die Visualisierung oder die Aufzeichnung anordnen können.

Dazu werden spezielle Bearbeitungsschritte implementiert, die diese Module ansprechen. Das ist möglich, da sich die Modulansteuerung und die Bearbeitungsschritte stark
5. Feinkonzept


5.2.5. Kontrolle der Bearbeitungsschritte


Gerade um die Performance der gesamten Bearbeitungskette zu erhöhen, läuft jeder
5. Feinkonzepte

5.3. Visualisierung

5.3.1. Schnittstelle

Um beliebige Inhalte in einem Frontpanel dynamisch anzuzeigen, bietet LabVIEW die Subpanels an. In sie kann das Frontpanel eines anderen VIs zur Laufzeit eingebettet werden. Diese Funktionalität wird als Schnittstelle benutzt, mit der ein Visualisierungsbearbeitungsschritt der Visualisierung ein Diagramm zur Anzeige übergeben kann. Mit dieser Schnittstelle kann der Programmierer des Visualisierungsbearbeitungsschrittes das Diagramm wie ein Frontpanel zusammenstellen und LabVIEW Bibliotheken, wie den XY-Graph verwenden. Die Schnittstelle ist vom anzuzeigenden Inhalt völlig unabhängig.

Die Visualisierung verwaltet die Subpanels und arrangiert sie untereinander in einem Bereich der GUI. In Abbildung 5.3 ist rechts dieser Bereich mit zwei Diagrammen schematisch dargestellt.

5.4. Konfigurierung

Da die zu konfigurierende Bearbeitung aus mehreren voneinander unabhängigen Bearbeitungsschritten besteht, muss auch die Konfiguration für jeden Bearbeitungsschritt einzeln einstellbar sein. Dazu wird dem Benutzer eine Liste der Bearbeitungsschritte in der GUI präsentiert. In Abbildung 5.3 ist die Liste links mit den Beispielbearbeitungsschritten Operation1, Operation2 und Operation3 abgebildet.
5.5. Aufzeichnung und Wiedergabe

5.5.1. Dateiformat

Die Messungen sollen in Dateien gespeichert werden. Dazu ist ein Dateiformat notwendig, das auch ungleichmäßig abgetastete Signale unterstützt. Außerdem soll die Datei möglichst schnell schreib- und lesbar sein.


6. Umsetzung

In den vorgeschlagenen Kapiteln wurden die einzelnen Module identifiziert und Konzepte für diese gefunden. In der Umsetzung werden die Module mit den gefundenen Konzepten implementiert.


Abbildung 6.1.: Klassenstruktur

6.1. Treiber

6.1.1. Datum


6.1.2. Signal

Um mehrere Daten zwischen zwei Bearbeitungsschritten auszutauschen, wird die Klasse MeasurementSeries implementiert. Sie ist ein Interface mit den beiden Methoden Datum receiveDatum() und putDatum(Datum datum). Zur Vereinfachung der Programmierung der Bearbeitungsschritte, das die empfangenen Daten streng monoton wachsend sind, also ∀i, j ∈ N : i > j ⇒ Zeit(i) > Zeit(j). Werden demnach zwei Daten empfangen, wurde das zuerst empfangene Datum auch früher erfasst.

Für beliebig abgetastete Signale ist die Klasse UnevenlySpacedSeries als Realisierung der MeasurementSeries gedacht. In der Methode putDatum wird das übergebene Datum in eine Queue gelegt. Bei Aufruf der Methode receiveDatum wird das am längsten in der
Queue liegende Datum aus ihr entfernt und zurückgegeben. Um die Monotonie sicherzustellen, wird beim legen von Daten in die `UnevenlySpacedSeries` der Zeitstempel des Datums auf Monotonie geprüft. Wenn das Kriterium nicht erfüllt ist, wird das Datum nicht in die Queue gelegt. In dem Fall wird ein Fehler zurückgegeben.

```java
public class UnevenlySpacedSeries extends MeasurementSeries{
    private Queue<Datum> queue=new Queue<Datum>();
    private Timestamp lastTime=null;
    public void putDatum(Datum datum)throws Exception{
        if(lastTime==null || lastTime.isBefore(datum.getTime())){
            queue.put(datum);
            lastTime=datum.getTime();
        } else {
            //Die Zeit ist nicht streng monoton wachsend
            throw new Exception("The time of the datum is invalid");
        }
    }
}
```


6.2. Bearbeitung


6.2.1. Parameter

Der `IOParameter` bietet die Möglichkeit Daten auszutauschen, Daten anzufragen und den Zustand der erzeugenden Operation zu erfahren. Er bündelt also eine `MeasurementSeries`, einen Zustand und einen `CommunicationChannel`.

Mit dem `CommunicationChannel` kann eine Operation den am Eingang verfügbaren Zeitbereich sehen. Daten in diesem Bereich können dann über ihn angefragt werden. Der `CommunicationChannel` benötigt daher zwei Variablen, in denen die untere und obere Grenze des Zeitbereichs vermerkt sind. Ist die obere Grenze kleiner als die untere Grenze

6.2.2. Operation


Die Zustände werden dazu auf den folgenden gemeinsamen Nenner gebracht, die Events ergeben sich daraus:

Zustände = {Beendet, Laufend, Pausiert}, Events = {Start, Stopp, Pausieren, Fortsetzen}


Die überschreibbaren Methoden, die im Zustandsautomat aufgerufen werden, können Fehler zurückgeben. In diesem Fall wird der Zustandsautomat heruntergefahren, also das herunterfahren Event ausgelöst. Außerdem wird der Fehler dem Benutzer angezeigt.

```java
public abstract class Operation{
    private enum state;
    protected void executeState(enum state);
    public void operate(){
        boolean shutdown=false;
        while(!shutdown){
            try{
                executeState(state);
            } catch(Exception e) {
                enum decision=GUI.reportError(e);
                if(decision==shutdown){
                    shutdown=true;
                }
            }
        }
    }
}
```
Abbildung 6.2.: Zustandsautomat der Operationen
6. Umsetzung

6.2.3. Benutzerinterface

Damit möglichst einfach neue Bearbeitungsschritte hinzugefügt werden können, werden von der Operation speziellere Klassen abgeleitet, die für häufig vorkommende Operationssarten Vereinfachungen haben. Abbildung 6.3 zeigt ein Vererbungsdiagramm dieser Operationen. Im Folgenden wird die Funktionalität erklärt.

**Driver** Eine Operation, die keinen Eingang hat. Geeignet um einen Sensor in das Tool einzubinden.

**SIZOOperation** (Single Input Zero Output) Hat genau eine Eingabe und keine Ausgabe. Stellt eine Methode bereit um ohne Arrayindizierung auf die Eingabe zuzugreifen. Eignet sich zum Beispiel für Regler oder Aufzeichnungen.

**SOOperation** (Single Output) Hat nur einen Ausgang. Gerade Operationen die mit Messreihen rechnen, haben meistens nur einen Ausgang.

**VisualizationOperation** Registriert automatisch das Frontpanel eines VIs für die Anzeige als Diagramm. Zur Verwendung als Visualisierungsmodul gedacht.


**DatumByDatumOperation** Eine Operation, die das Empfangen und Legen von Daten in die Ein und Ausgabe Messreihen implementiert. Für jedes empfangene Datum wird eine überschreibbare Methode aufgerufen, die das Datum modifiziert. Das Resultat dieser Methode wird in die Messreihe gelegt.

**ValueByValueOperation** Im Gegensatz zur DatumByDatumOperation werden hier nicht mehr ganze Daten, sondern nur die Werte der Daten modifiziert. Die Zeiten werden also unverändert übernommen.

6.2.4. Bearbeitungsschritte

Folgend sind die Operationen mit Beschreibung aufgelistet, die im Rahmen der Bachelorarbeit zum Demonstrieren der Funktionalität des Frameworks implementiert wurden:

**ScaleOperation** Multipliziert den Wert jedes empfangenen Datums um einen konfigurierbaren Wert.
6. Umsetzung

![Diagram of operations]

Abbildung 6.3.: Operationsarten

**OffsetOperation**  Addiert auf den Wert jedes empfangenen Datums einen konfigurierbaren Wert.

**StoreOperation**  Speichert die Eingangsmessreihe in eine TDMS-Datei.

**LoadOperation**  Lädt eine Messreihe aus einer TDMS-Datei.

**ForkOperation**  Erzeugt zwei Kopien der Eingangsmessreihe und legt jedes empfangene Datum in die beiden Messreihen.

**XYVisualization**  Visualisiert die empfangenen Daten in einem XY-Diagramm.

**SineGenerator**  Ein Treiber, der keinen Sensor anspricht. Stattdessen wird ein Sinussignal als gleichmäßig abgetastete Messreihe ausgegeben.

**UnevenSineGenerator**  Wie der SineGenerator, gibt jedoch eine ungleichmäßig abgetastete Messreihe aus.

6.3. Visualisierung

Für die Visualisierung wird die Klasse `DiagramPane` erstellt. Immer, wenn ein Diagramm visualisiert werden soll, wird dem `DiagramPane` eine Referenz auf das VI übergeben, das das Diagramm im Frontpanel enthält.

6.3.1. Schnittstelle

Für die Übergabe der Referenz an das `DiagramPane` wird die Klasse `Embedable` implementiert. Ihr kann im Konstruktor eine Referenz auf ein VI übergeben werden, deren Frontpanel bei Aufruf der Methode `embed(Subpanel destination)` in das spezifizierte Subpanel eingebettet wird. Ein `Embedable` kann dem `DiagramPane` mit der Methode `register(Embedable embedable)` übergeben werden.
6. Umsetzung

6.3.2. Diagramm Bereich

Das DiagramPane muss bei Aufruf der register(Embedable embedable) Methode ein neues Subpanel erstellen und das Embedable dort einbetten. Für das dynamische erzeugen von Subpanels, oder das führen einer Liste mit beliebigem Inhalt stellt LabVIEW keine Funktionen zur Verfügung.

6.4. Konfigurierung

Die Klasse Controller kontrolliert die Zustandsänderung und die Einstellungen einer Operation.

6.4.1. Zustandsänderung

Die Konfiguration öffnet bei Rechtsklick ein Kontextmenü mit den Events

- starten,
- stoppen,
- pausieren,
- fortsetzen und
- resetten.

Bei Auswahl eines Punktes wird die Methode raiseEvent(Enum event) mit dem ausgewählten Event als Parameter ausgeführt.

6.4.2. Einstellungen


```java
public class DivideOperation extends Operation{
    private double divisor=1;
    protected void configure(double input) throws Exception{
        if(input==0){
            //0 ist eine ungültige Eingabe
            throw new Exception("0 is not allowed");
        } else {
            divisor=input;
        }
    }
```

29
public abstract class Operation{
    protected void configure(double input) throws Exception;
    public void callConfig() {
        boolean validInput=false;
        while(!validInput) {
            try {
                configure(GUI.getUserInput());
                validInput=true;
            } catch (Exception e) {
                GUI.reportError(e);
            }
        }
    }
}


6.5. Aufzeichnung und Wiedergabe

6.5.1. Messdatei

7. Validierung

Um die Korrektheit der Implementierung zu zeigen und die Performance des Tools zu zeigen, werden die wichtigen Module des Tools getestet. Folgend sind die Tests für die Module beschrieben.

7.1. Zustandsautomat

Beschreibung Der Zustandsautomat muss als zentrale Steuereinheit von Operationen richtig implementiert sein (siehe Abbildung 6.2).


Um alle Zweige aus Abbildung 6.2 zu durchlaufen und so eine möglichst große Abdeckung sicherzustellen, werden die folgenden Events nacheinander ausgeführt: Start, Pausieren, Start, Pausieren, Fortsetzen, Restart, Start, Fortsetzen, Stopp, Pausieren, Fortsetzen, Stopp, Start, Pausieren, Restart, Pausieren, Stopp.

Die erwartete Ausgabe nach Abbildung 6.2 ist: onStart(), onPause(), -, -, onResume(), onStop(), onStart(), -, -, onStop(), -, -, -, onStart(), onPause(), onStop(), onStart(), onPause(), onStop().

Ergebnis Die Reihenfolge der geloggten Zustände stimmt überein. Der Zustandsautomat ist korrekt implementiert.

7.2. Dateieingabe

Beschreibung Die Dateioperationen sollen die Dateien am Anfang auf Richtigkeit prüfen. Wenn die Datei ungültig ist, soll schon dann ein Fehler geworfen werden. Dadurch sollen Fehler während des Lesens vermieden werden. Es muss geprüft werden, ob Fehler der Datei erkannt werden.

Testdurchführung Für das Lesen von Dateien muss folgendes gelten:

- Die Datei muss existieren.
- Der gewählte Kanal muss existieren und den Datentyp Double haben.
- Die Datei hat entweder Start und Inkrement, oder einen weiteren Kanal für die Zeiten.
- Wenn sie einen Zeitkanal hat, müssen Zeitkanal und Wertkanal gleich viele Einträge enthalten.
7. Validierung

Abbildung 7.1.: Messreihen Geschwindigkeitstest

- Wenn vorhanden muss Start den Typ Zeitstempel und Inkrement den Typ Double haben.
- Sind Start und Inkrement nicht vorhanden, muss der Zeitkanal den Typ Zeitstempel haben.


**Ergebnis** Alle aufgelisteten Fehler wurden erkannt. Die Datei wird richtig überprüft.

7.3. Messreihenperformance

**Beschreibung** Die Messreihe soll Daten von einer Operation an die nächste weiterleiten. Für die Performance des Tools ist es wichtig, dass die Verzögerung, die durch die Datenweiterleitung entsteht, gering ist.

7. Validierung

Ergebnis  In dem durchgeführten Test wurden 1.000.000 Daten gelegt und empfangen. Für die EvenlySpacedSeries betrug die Verzögerung 7,573 Sekunden. Das ergibt eine durchschnittliche Verzögerung pro Datum von 7,573 Mikrosekunden.

Die UnevenlySpacedSeries hatte eine Verzögerung von 4,537 Sekunden. Das ergibt eine durchschnittliche Verzögerung pro Datum von 4,537 Mikrosekunden.

Das die EvenlySpacedSeries länger braucht, liegt daran, dass die beim empfangen von Daten die Zeit erst berechnet werden muss. Werden der EvenlySpacedSeries statt der Daten nur die Werte übergeben und empfangen, beträgt die Verzögerung nur 3,517 Sekunden, also 3,517 Mikrosekunden pro Datum.

7.4. Operationsperformance

Beschreibung  Um die Performance der Bearbeitung zu bestimmen, wird die Verzögerung von Daten durch die Skalierung bestimmt. Da die Daten mit bis zu 10 kHz ankommen, sollte die Verzögerung deutlich unter 100 Mikrosekunden liegen.

**Ergebnis** In dem durchgeführten Test wurden 1.000.000 Daten gelegt und empfangen. Die Verzögerung betrug 9,483 Sekunden. Das ergibt eine durchschnittliche Verzögerung pro Datum von 9,483 Mikrosekunden. Das ist ein Zehntel der Periodendauer und somit für die Anwendung bis 10 kHz ausreichend.
8. Zusammenfassung und Ausblick

8.1. Zusammenfassung

In der Arbeit ist es gelungen Grobkonzepte zu finden, die die allgemeine Messaufgabe modular zusammenfassen. Das Grobkonzept ermöglicht es, die typischen Messaufgaben in mehreren Modulen unabhängig voneinander zu betrachten. Die Schnittstellen der Module und ihre Vernetzung wurden bestimmt und so ausgelegt, dass auch asynchrone Signale verarbeitet werden können.


Das Tool wurde gemäß den Konzepten in LabVIEW objektorientiert implementiert. Bei der Implementierung wurden LabVIEW möglichst viele LabVIEW Bibliotheken benutzt, um eine gute Integration in LabVIEW zu erhalten. Die Schnittstellen wurden wenn möglich an LabVIEW Schnittstellen angelehnt. Das Tool lässt sich so einfach um LabVIEW Funktionen erweitern.

Die Validierung zeigt, dass grundlegende Bestandteile des Tools für Abtastraten um 10 kHz geeignet sind. Außerdem weist die Implementierung keine Fehler in den wesentlichen Bestandteilen der Implementierung auf.

8.2. Ausblick


A. UML-Struktur

Abbildung A.1.: Von GOOP erzeugte UML-Struktur des Tools
Begriffe

**Weinmann**  WEINMANN Emergency Medical Technology GmbH + Co. KG

**LabVIEW**  NI LabVIEW

**VI**  Virtuelles Instrument

**GUI**  Grafische Benutzeroberfläche

**GOOP**  NI GOOP Development Suite

**UML**  Unified Modelling Language

**CSV**  Comma-separated values
Literatur


