Duale Hochschule Mannheim
Staatliche Studienakademie

Studiengang Informationstechnik

Bericht zum Modul Praxis I
Praxisphase 1: 05.01. - 29.03.2009
Praxisphase 2: 22.06. - 30.09.2009

Thema 1: Netzwerkbasierter Datensimulator für den Rotorversuchsstand

Thema 2: Datensynchronisator für Wetter- und Telemetriedaten

von

Christian König

Matrikelnr.: 207313

Kurs: TIT08AGR

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Standort Braunschweig

Institut für Flugsystemtechnik
Abteilung Hubschrauber

Betreuer: Martin Gestwa

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass ich
die vorliegende Arbeit selbstständig
und nur unter Verwendung der als Quellen
angegebenen Hilfsmittel angefertigt habe.

Braunschweig, den 30.09.2009 _____
Christian König

Inhaltsverzeichnis

Abkürzungsverzeichnis

ANSI	**A**merican **N**ational **S**tandards **I**nstitute
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
CGI	**C**ommon **G**ateway **I**nterface
DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt
COM	**Com**munication (Port)
DNW	**D**eutsch-**N**iederländische **W**indkanäle
EC	**E**uro**c**opter
FHS	**F**lying **H**elicopter **S**imulator
FTP	**F**ile **T**ransfer **P**rotocol
I/O	**I**nput / **O**utput
I^2C	**I**nter-**I**ntegrated **C**ircuit
IC	**I**ntegrated **C**ircuit
ISS	**I**nternational **S**pace **S**tation
LLF	**L**arge **L**ow-Speed **F**acility
NLR	**N**ationales **L**uft- und **R**aumfahrtlabor der Niederlande
PD	**P**ower **D**own
RAM	**R**andom **A**ccess **M**emory
RTOS	**R**eal-**T**ime **O**perating **S**ystem
Telnet	**Tel**ecommunication **Net**work
UDP	**U**ser **D**atagramm **P**rotocol
URL	**U**niform **R**esource **L**ocator
USB	**U**niversal **S**erial **B**us

Abbildungsverzeichnis

Listings

Vorwort

Das Deutsche Zentrum für Luft- und Raumfahrt ist das Forschungszentrum der Bundesrepublik Deutschland für die Bereiche Luftfahrt, Raumfahrt, Verkehr und Energie. Es fungiert als Raumfahrtagentur der Bundesregierung und kooperiert in diesem Bereich, genau wie in der Forschung, national und international. So betreibt das DLR beispielsweise das Kontrollzentrum des Columbus-Labors der ISS. Die Zielsetzung der Forschung ist in der Internetpräsenz des DLR [3] gut zusammengefasst:

> „Die Mission des DLR umfasst (...)
>
> - die Erforschung von Erde und Sonnensystem,
> - die Forschung für den Erhalt der Umwelt,
> - die Entwicklung umweltverträglicher Technologien zur Steigerung der Mobilität sowie für Kommunikation und Sicherheit."

Das DLR betreibt in Deutschland 29 Institute bzw. Test- und Betriebseinrichtungen, die sich auf 13 Standorte verteilen. International gibt es Büros in Brüssel, Paris und Washington D.C. Insgesamt beschäftigt das DLR etwa 6200 Mitarbeiter.

Die vorliegende Arbeit wurde im Institut Flugsystemtechnik in der Abteilung Hubschrauber erstellt. Die Abteilung befasst sich mit Rotorsystem- und Flugsystemtechnik, mit dem Ziel, das Fliegen sicherer, wirtschaftlicher und umweltfreundlicher zu machen.

Abbildung 1: DLR - Forschung für die Erde [5]

Teil I

Netzwerkbasierter Datensimulator für den Rotorversuchsstand

1 Einleitung

1.1 Aufgabenstellung

Bei den Messkampagnen des Rotorversuchsstandes im niederländischen Windkanal DNW-LLF (*large low-speed facility*) werden Umgebungsdaten wie beispielsweise Temperatur, Luftdruck, Luftfeuchtigkeit und Windgeschwindigkeit per Ethernet an ein Datenerfassungssystem des DLR gesendet. Da die Tests im DNW-LLF zeit- und planungsintensiv sind, werden diese in der Rotorhalle des Instituts Flugsystemtechnik in Braunschweig vorbereitet. Hierbei ist es vorteilhaft, die Umgebungsbedingungen genau wie im niederländischen Windkanal zu messen oder sinnvoll zu simulieren und diese Daten an den Daten-Server zu senden. So ist es möglich, sämtliche Systeme besser auf die gewünschte Funktionsweise zu testen und unnötige Termine im DNW-LLF zu vermeiden.

Das Projekt soll mittels eines Mikrocontrollers bearbeitet werden. Das zur Verfügung stehende Modell ist der SC13-LF von der Firma Beck IPC. Die Kommunikation zwischen dem Daten-Server und dem Mikrocontroller soll so ablaufen, dass der Mikrocontroller wahlweise mit 1 Herz oder mit 0,1 Herz die Daten sendet. Das Timeout soll eine Sekunde betragen.
Außerdem soll ein Web-Interface erstellt werden, über das das System parametrierbar sein soll und in dem auch die aktuellen Messwerte überwacht werden können. Hierfür ist der auf dem Mikrocontroller vorhandene Webserver zur verwenden.
Die Messung des Luftdrucks und der Temperatur soll durch einen entsprechenden Sensor erfolgen, genauer gesagt einem SCP1000 von VTI. Allerdings ist die Installation und Implementierung des Sensors nicht Teil der Aufgabe für dieses Semester. Lediglich ein Einlesen in die Spezifikationen des SCP1000 soll erfolgen.

1.2 Mikrocontroller

Mikrocontroller sind selbstständige Rechnersysteme, die sich komplett auf einem Chip befinden. Neben dem Prozessor sind die integrierten Bauteile zumeist Arbeitsspeicher (RAM), Flashspeicher und einige Peripheriefunktionen. Hier besteht auch der Unterschied zu Mikroprozessoren, die lediglich aus einem Prozessor bestehen und keine weiteren Bauteile enthalten.
Die erwähnten Peripheriefunktionen, die im Mikrocontroller integriert sein können, sind unter anderem verschiedene Schnittstellen, wie beispielsweise USB-, I^2C-, oder Ethernet-Schnittstellen, können aber auch Analog-Digital-Wandler oder LCD-Controller sein. Mikrocontroller gibt es in vielen verschiedenen Kombinationen an Peripheriefunktionen und Prozessor-Architekturen, von 4-Bit bis 32-Bit ist alles gebräuchlich. Mikrocontroller werden als eingebettete Systeme in vielen technischen Geräten verwendet, z.B. in Ladegeräten,

DVD-Playern, Mobiltelefonen, Telefonkarten, Uhren und Autos. [9] [10]
Auf Mikrocontrollern steht nur wenig Speicherplatz zur Verfügung und die Ressourcen sind beschränkt. Eine Anforderung an das fertige Programm wird also sein, dass es wenig Speicherplatz benötigt und ressourcensparend arbeitet. Aus diesem Grund bietet die Programmiersprache C sich zur Lösung der Aufgabe an.

1.3 Programmiersprache C

Die Programmiersprache C wurde im Jahr 1972 von Dennis Ritchie und Ben Thompson entwickelt und ist eine prozedurale Programmiersprache. C ist nicht objektorientiert, hat aber den Vorteil, dass die erstellten Programme in der Regel sehr schnell und sehr klein sind.

1.3.1 ANSI C

ANSI C ist eine Vereinheitlichung der vorher oft stark abweichenden C-Varianten. Diese Norm wurde im Jahr 1989 entwickelt. ANSI C vereinbart unter anderem die Standardbibliotheken und bietet den Vorteil, dass durch einen einheitlichen Standard die Portabilität von C-Programmen gegeben ist. Ein auf einem Rechner geschriebener C-Code kann ohne Schwierigkeiten auf einem anderen Rechner kompiliert werden. [1]

1.3.2 CLIB von Beck

In der C-Bibliothek CLIB von Beck sind eine ganze Reihe an Funktionen implementiert, die speziell für Mikrocontroller-Steuerung und die Kommunikation mit Mikrocontrollern geschrieben wurden. So enthält diese Bibliothek beispielsweise Kontrollfunktionen und Funktionen für Schnittstellen, die das Senden und Empfangen von Daten betreffen. Viele der Funktionen sind nicht notwendig, da stattdessen auch direkt der dem Maschinencode nahe Assembler-Befehl gegeben werden kann, erleichtern das Arbeiten aber deutlich.

2 Verwendete Hardware

2.1 Mikrocontroller SC13-LF

Der verwendete Mikrocontroller ist der SC13-LF der Firma Beck IPC. Er verfügt über einen 16-Bit Prozessor mit einer eigenen 186er ASIC mit 40 MHz. Er enthält 512 kB RAM sowie einen Flashspeicher mit weiteren 512 kB, der zum Speichern der geschriebenen Programme zur Verfügung steht. Als Betriebssystem fungiert ein RTOS, ein Multitasking-Echtzeitbetriebssystem, das auch ein Flash-Filesystem bietet. Der Mikrocontroller verfügt über I/O-Pins, an die SPI-, I^2C- sowie Ethernet-Schnittstellen angeschlossen werden können. Außerdem ist es möglich, zwei serielle Ports anzusteuern.
Zusätzlich ist ein Watchdog implementiert, der das Funktionieren der Software überwacht. Hierfür meldet sich die Software in regelmäßigen Abständen beim Watchdog, um zu signalisieren, dass es zu keinem Programmabsturz gekommen ist. Reagiert die Software jedoch nicht mehr, kann sie sich auch nicht beim Watchdog melden, der daraufhin die Software in den Ausgangszustand zurückversetzt. So ist gewährleistet, dass die Software stets funktionsfähig ist. [12] [13]

Abbildung 2: Mikrocontroller SC13-LF von Beck [13]

2.2 DK50 Development Board

Der verwendete Mikrocontroller SC13-LF befindet sich auf dem Entwicklungsboard DK50 von BECK. Es verfügt über zwei serielle Schnittstellen, eine Ethernet-Schnittstelle sowie über einen CompactFlash-Kartenleser und dient dem Testen des Chips und der aufgespielten Software. Weiterhin bietet das DK50 acht digitale Input-/Output-Pins, sodass zusätzliche Peripheriegeräte angeschlossen werden können. [14]

2.3 Drucksensor SCP1000

Der Sensor SCP1000 der Firma VTI Technologies ist als Drucksensor konzipiert, bietet aber zusätzlich die Möglichkeit, die Umgebungstemperatur zu messen.
Der Messbereich des Chips liegt zwischen 300 und 1200 hPa für die Druckmessung und zwischen -20 und +70°C für die Temperaturmessung. Die zulässige Versorgungsspannung liegt zwischen +2,4 und +3,3 Volt. Der Chip hat einen Durchmesser von 6,1 mm und eine Höhe von 1,7 mm. Als Schnittstelle bietet der SCP1000 wahlweise entweder I^2C oder SPI an.
Es stehen vier verschiedene Messeinstellungen zur Verfügung, der *High Resolution Mode*, der *High Speed Mode*, der *Ultra Low Power Mode* und der *Low Power Mode*.
Der *High Resolution Mode* bietet eine kontinuierliche Messung der Umgebungsbedingungen. Die Auflösung entspricht 17 bit und die Aktualisierungsrate der Output-Daten beträgt 1,8 Hz. Der Stromverbrauch bei dieser Messeinstellung beträgt 25 μA.
Ebenso wie der *High Resolution Mode* bietet auch der *High Speed Mode* eine kontinuierliche Messung. Allerdings beträgt die Auflösung nur 15 bit, dafür aber die Output-Aktualisierungsrate 9 Hz. Diese Messmethode weist ebenfalls einen Stromverbrauch von 25 μA auf.
In der Einstellung *Ultra Low Power Mode*, in der eine periodische Messung vorgenommen wird und keine kontinuierliche, beträgt die Auflösung ebenfalls 15 bit. Die Aktualisierungsrate des Outputs beträgt in dieser Messmethode etwa 1 Hz. Der Stromverbrauch liegt im *Ultra Low Power Mode* bei 3,5 μA.
Befindet sich der Chip im *Low Power Mode* wird keine automatische Messung vorgenommen, die Messung wird durch einen externen Auslöser manuell eingeleitet. Die Auflösung beträgt wahlweise 15 bit oder 17 bit. Der Stromverbrauch variiert, je nachdem welche Auflösung gewählt ist und wie oft gemessen wird.
Als zusätzlicher Modus steht der *Power Down Mode* zur Verfügung, in dem keine Messungen möglich sind. Dieser Modus kann über den PD-Pin aktiviert beziehungsweise deaktiviert werden. Der Stromverbrauch beträgt hier lediglich 0,2 μA. [16]

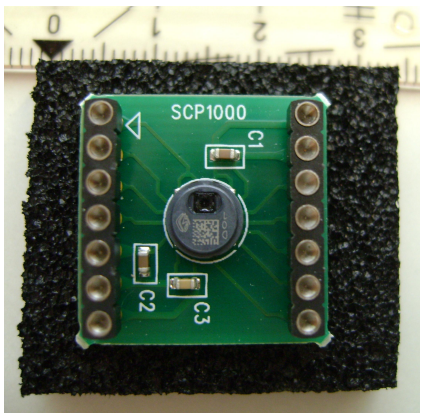

Abbildung 3: Drucksensor SCP1000 von VTI

3 Verwendete Software

3.1 CHIPtool

Die in der Version 5.11.1.0 verwendete Software CHIPtool von BECK dient dazu, die Netzwerk-Konfiguration des Mikrocontrollers vorzunehmen. Weiterhin stellt sie einen FTP- sowie einen Telnet-Client bereit. Mit dem FTP-Client kann die kompilierte Software auf den Chip übertragen und die Konfigurationsdatei des Mikrocontrollers modifiziert werden. Telnet ist ein Netzwerk-Protokoll, das den Zugriff auf die Kommandozeile des auf dem Mikrocontroller vorhandenen Betriebssystems (RTOS) ermöglicht. Weitere Funktionen dieser Software, die aber im Verlauf des Projekts nicht genutzt wurden, sind das Aufspielen einer neuen Version des Betriebssystems sowie das Erstellen eines Software-Images für eine Serienprogrammierung.

3.2 Borland C++ Compiler

Borland C++ ist eine integrierte Entwicklungsumgebung für die Programmiersprachen C und C++. Der verwendete Borland C++ Compiler ist ein Cross-Compiler. Er lässt sich so konfigurieren, dass das durch ihn erstellte Programm auf einem anderen Prozessor und einem anderen Betriebssystem lauffähig ist. Dies ist besonders wichtig, wenn die Zielplattform ein eingebettetes System ist, auf dem die Programm-Entwicklung und Kompilierung aufgrund mangelnder Rechenleistung nicht möglich ist, wie zum Beispiel ein Mikrocontroller. Weiterhin werden Cross-Compiler verwendet, um Zeit zu sparen, indem Programme auf leistungsstarken Systemen entwickelt werden, deren Anwendung auf leistungsschwachen Systemen stattfinden wird. [11]
Borland C++ verfügt über einen Debugger. Es ist also möglich, den Quellcode bei der Fehlersuche schrittweise ablaufen zu lassen und währenddessen die Werte sämtlicher Variablen zu überprüfen, was zu einer massiven Erleichterung der Fehlersuche führt.

3.3 Institutseigene Software

Unter diesem Punkt ist Software zusammengefasst, die von Mitarbeitern des DLR während anderer Projekte erstellt und für dieses Projekt zur Verfügung gestellt wurde.

3.3.1 extserver.exe

Das Programm extserver.exe ist dasselbe, das auch im Netzwerk des Rotorversuchsstand läuft. Es kann jedoch auch auf einem einzelnen Rechner verwendet werden, indem der

LocalHost angesprochen wird. Unter LocalHost versteht man stets den Rechner, auf dem man gerade Arbeitet.
Das Programm dient als Zwischenspeicher und Verteiler von asynchronen Daten, die per TCP/IP von verschiedenen Quellen als ASCII-Strings gesendet werden. Die Quellen werden durch ihre IP-Adresse identifiziert, die Namenskennung der Daten ist in einer Datenbank hinterlegt. Auch Nutzer kommunizieren per TCP/IP mit dem Programm. Sie können aktuelle oder aufgezeichnete Datensätze anfordern.

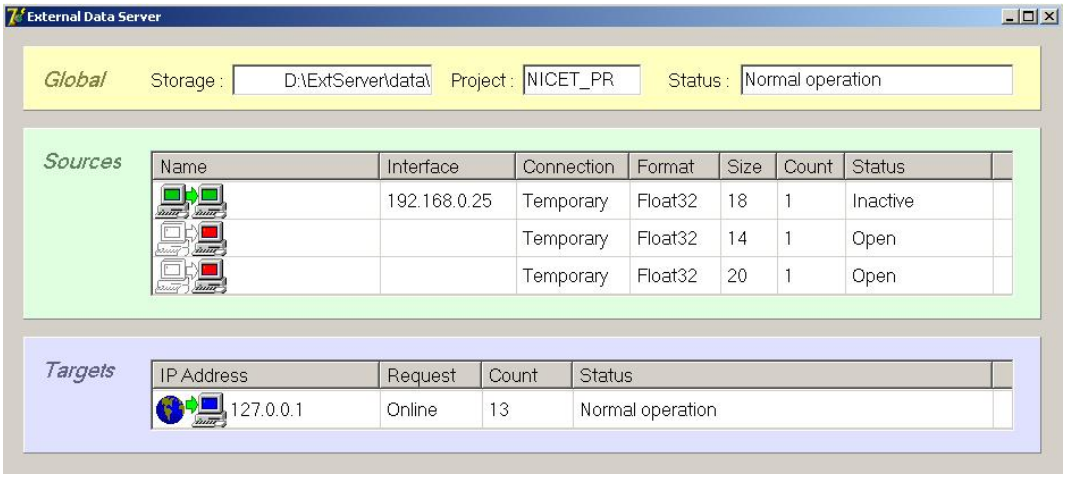

Abbildung 4: Programm extserver.exe

3.3.2 Quickview.exe

Zu den in Abschnitt 3.3.1 erwähnten Nutzer zählt auch das Programm Quickview.exe. Die mit extserver.exe aktualisierten Datensätze können hiermit in Echtzeit ausgelesen werden. Hierfür wird an extserver.exe eine Anfrage vom Client übermittelt, woraufhin dieser die jeweiligen Daten an Quickview.exe sendet. In Quickview.exe kann unter anderem ausgewählt werden, aus welcher Datenbank die Informationen abgefragt und welche Informationen angezeigt werden sollen.

Mit der Kombination aus extserver.exe und Quickview.exe ist es möglich, die Funktionalität des erstellten Programms zu testen, ohne über direkten Zugang zu dem Netzwerk im Rotorversuchsstand zu verfügen.

4 Web-Interface

4.1 Implementierung

Das erstellte Web-Interface basiert auf CGI, einem Verfahren, das im Bereich der Webserver häufig Verwendung findet. CGI ermöglicht es, dynamische Websites zu erzeugen. Es wird unterschieden zwischen zwei Anfrage-Methoden, *get* und *post*. Bei *get* werden die zu übermittelnden Daten, mit deren Hilfe die neue Website erstellt wird, in der URL übertragen, sind also leicht einsehbar. Wird *post* verwendet, so werden diese Informationen an den Server übertragen und dort in einer Umgebungsvariablen zwischengespeichert, ohne dass sie in der URL erscheinen. Ob die übergebenen Parameter in der URL erscheinen ist in diesem Projekt nicht von Bedeutung, weshalb *get* verwendet werden konnte.
Um das CGI-Verfahren zu nutzen, ist es zunächst notwendig, den Webserver so zu konfigurieren, dass er die neu erzeugte Website anzeigt, anstatt sie zum Herunterladen bereitzustellen. Im Falle des Webservers auf dem SC13-LF geschieht dies, indem man der CGI-Tabelle des Mikrocontrollers einen Eintrag hinzufügt, der den URL-Pfad der Seite enthält sowie die Anfrage-Methode und den Namen der Funktion, die aufgerufen werden soll, wenn eine entsprechende Anfrage beim Server eingeht. Dies geschieht mit folgendem Code-Fragment:

```
1  //init cgiFuncSettings
2     //Declarations
3     CGI_Entry  cgi;
4     union REGS inregs;
5
6     //Configure CGI
7     cgi.CgiFuncPtr = cgiFuncSettings;      //CGI-Function
8
9     cgi.PathPtr    = "cgi-bin/settings";   //URL-Path
10    cgi.method     = CgiHttpGet;           //CGI-Method
11
12    //Configure inregs for software interrupt
13    inregs.h.ah = CGI_INSTALL;
14    inregs.x.dx = FP_SEG(&cgi);
15    inregs.x.si = FP_OFF(&cgi);
16
17    //software interrupt to install the CGI-function
18    int86( 0xAB , &inregs , &outregs );
```

Listing 1: Installieren der CGI-Funktion

4.2 Websites

Ist die Website in die CGI-Tabelle des Webservers aufgenommen, muss die Funktion, die aufgerufen wird, zunächst einmal die übermittelten Informationen extrahieren und verarbeiten. Die Informationen befinden sich, wie in Abschnitt 4.1 erwähnt, in einer Umgebungsvariablen und sind dort als Zeichenkette gespeichert. Diese Zeichenkette muss zerlegt werden und die einzelnen Bestandteile werden in Variablen gespeichert. Anschließend wird mit den extrahierten Informationen die neue Seite erstellt, indem der komplette Quellcode der HTML-Seite in einer Zeichenkette gespeichert wird. Zusammen mit der Information, dass es sich um eine HTML-Seite handelt, wird diese Zeichenkette nun an den Webserver übergeben, der die neue Seite anzeigt. Innerhalb der aufgerufenen Funktion sind auch weitere Aktionen möglich, wie zum Beispiel das Übermitteln von Informationen an den Daten-Server.

Das Web-Interface besteht aus zwei HTML-Seiten, die als iFrames in einen Rahmen eingebettet sind, der unter anderem das DLR-Logo enthält.
Eine der Seiten, die Konfigurationsseite, dient dazu, bestimmte Einstellungen vornehmen zu können, wie zum Beispiel die aktuelle Datenpunktnummer, die Windgeschwindigkeit oder den Anstellwinkel der Rotorachse, aber auch, in welchem Intervall die Messwerte an den Server geschickt werden sollen.

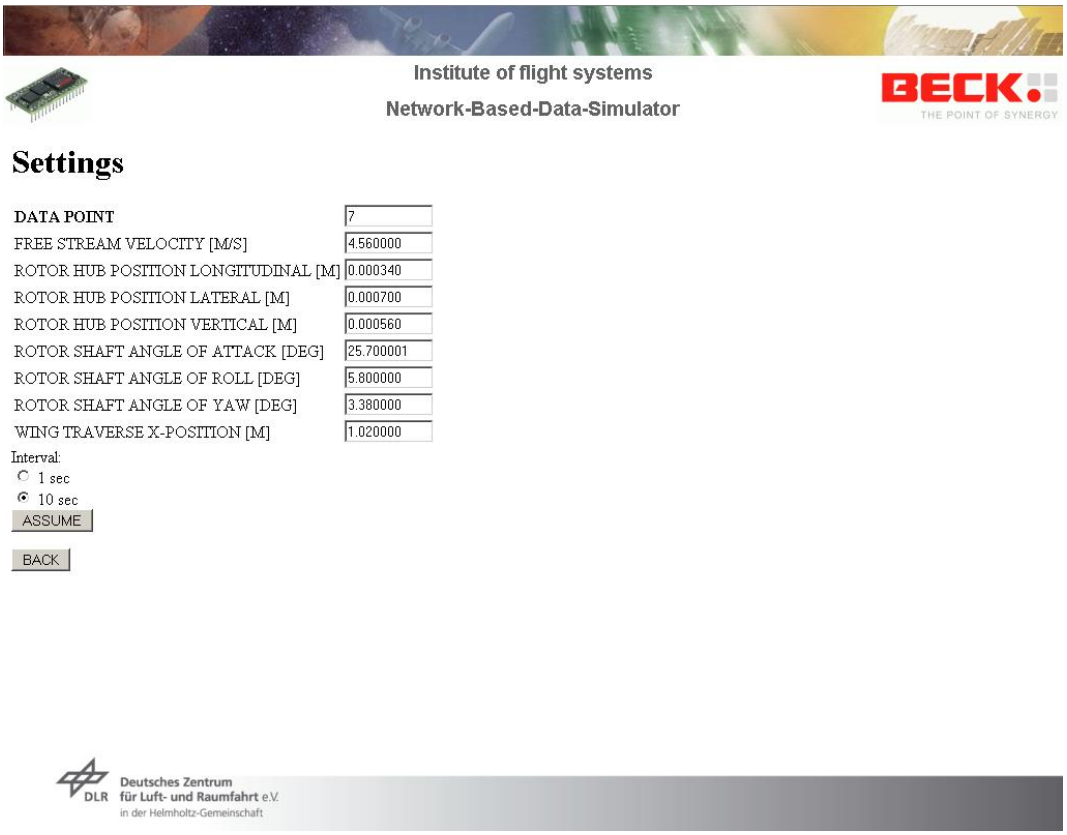

Abbildung 5: Konfigurationsseite des Web-Interfaces

Auf der zweiten Seite werden die Parameter angezeigt, die nicht vom Benutzer kalibriert werden können. Dies sind zum einen die Messdaten vom Sensor, wie zum Beispiel Luftdruck und Temperatur. Zum anderen sind es Werte, die aus den Messdaten errechnet werden, wie die aktuelle Schallgeschwindigkeit oder die Luftdichte.
Zusätzlich bietet diese Seite die Möglichkeit, die Verbindung zum Server zu trennen beziehungsweise herzustellen und das Senden der Daten zu aktivieren beziehungsweise zu deaktivieren. Eine weitere Schaltfläche dient dazu, die Datenpunktnummer um 1 zu erhöhen, ohne dafür die Konfigurationsseite öffnen zu müssen.

Abbildung 6: Darstellung der Messwerte im Web-Interface

5 Server-Kommunikation

Um mit dem Daten-Server zu kommunizieren, muss zunächst die Verbindung hergestellt werden. Um sicherzugehen, dass die Verbindung auch wirklich hergestellt wurde und der Server reagiert, wird ein Ping, also die Aufforderung zu einer Rückmeldung, an den Server gesendet. Erst wenn die Antwort auf diese Anfrage vom Server eingegangen ist, werden die Daten tatsächlich gesendet. So ist sichergestellt, dass die Übertragung auch wirklich empfangen wird. Antwortet der Server jedoch nicht innerhalb einer angemessenen Zeitspanne auf den Ping, können die Daten auch nicht gesendet werden. Stattdessen wird über die Konsole eine Fehlermeldung an den Benutzer ausgegeben, damit dieser darüber informiert wird und das Problem beheben kann. Eine mögliche Ursache für einen Verbindungsabbruch könnte beispielsweise ein Absturz des Servers oder ein nicht richtig eingestecktes Netzwerkkabel sein.
Es werden nicht dauerhaft Daten an den Server gesendet, sondern nur in bestimmten Intervallen. Je nach Konfiguration geschieht dies nur alle zehn Sekunden, weswegen es nicht nötig ist, dass die Verbindung dauerhaft aufrechterhalten wird. Deshalb wird nach jedem Senden eines Datenpakets an den Server die Verbindung wieder getrennt.
Der Ablauf der Serverkommunikation beim Senden von Daten sieht in einem Programmablaufplan also folgendermaßen aus:

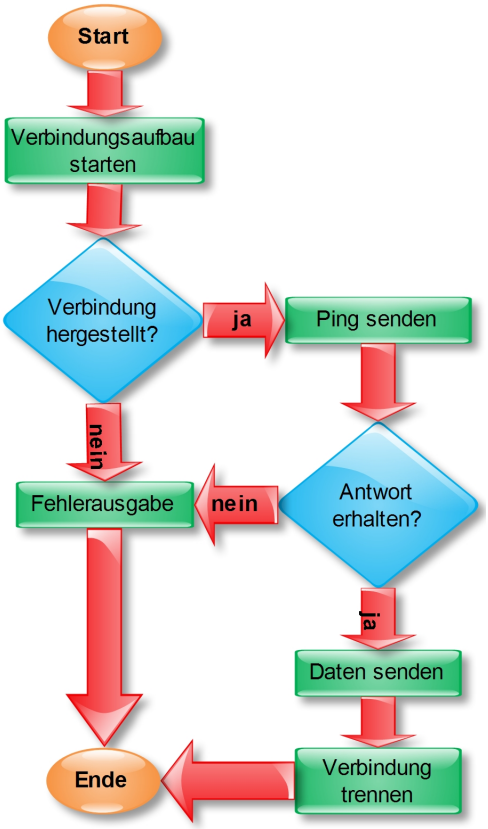

Abbildung 7: Programmablaufplan der Serverkommunikation

Die Messdaten beziehungsweise die auf der Konfigurationsseite vorgenommenen Einstellungen müssen, bevor sie an den Daten-Server gesendet werden, zunächst in einer Zeichenkette gespeichert werden. Hierbei ist es nicht von Bedeutung, ob ein einzelner Messwert oder eine Reihe von Messwerten übermittelt wird, lediglich auf die Syntax muss geachtet werden. Diese schreibt vor, dass der Zeichenkette zunächst die Kennung für den jeweiligen Wert enthalten muss, also zum Beispiel `DNW_VS` für die Schallgeschwindigkeit. Der Kennung folgt ein Gleichheitszeichen, auf das wiederum der jeweilige Wert folgt. Werden mehrere Messwerte auf einmal übertragen, können diese mit Semikola voneinander getrennt werden. Abgeschlossen wird die Zeichenkette von einem *Carriage Return* (Wagenrücklauf) und einem *Line Feed* (Zeilenvorschub). Ein möglicher String könnte folgendermaßen aussehen:

```
DNW_DPT=4;DNW_KINPRS=154126;DNW_TMP=285.6;DNW_PRS=100832.8\r\n
```

Empfängt der Daten-Server diesen String, werden in der Datenbankdatei die Werte des Datenpunktes, des kinetischen Drucks, der Temperatur sowie des Umgebunsdrucks aktualisiert. Der neue Wert des Datenpunktes beträgt beispielsweise 4, der Wert der Temperatur 285.6 Kelvin.

6 Fazit und Ausblick

Der Teil des Projekts, der Bestandteil der Tätigkeiten dieses Semesters war, wurde insgesamt erfolgreich fertiggestellt. Hierzu zählen die Implementierung des Web-Interfaces und die Einrichtung des Datenaustausches mit dem Daten-Server. Weiterhin umfasst dies auch die Anlegung des Grundgerüstes des Programms, das bereits die Schnittstellen zur Anbindung der weiteren Programmteile enthält. Ebenfalls erfolgreich war die Informationsbeschaffung zum Temperatur- und Luftdrucksensor SCP1000.
Bedarf zur Nachbesserung besteht lediglich beim Komfort für den Benutzer. Im Falle eines Fehlers, beispielsweise bei einem Verbindungsabbruch, wird derzeit die Fehlermeldung lediglich in der Telnet-Konsole angezeigt. Es ist also notwendig, diese Konsole zusätzlich zum Web-Interface geöffnet zu haben, um über einen möglichen Fehler informiert zu werden. Dies soll vermieden werden, indem eine solche Fehlermeldung direkt im Web-Interface erscheint. Somit ist es dann auch nicht mehr notwendig, dass CHIPtool, die Software, die auch die Telnet-Konsole enthält, auf dem Rechner installiert ist, auf dem die Anwendung laufen soll. Eine weitere Möglichkeit wäre es, die Fehlermeldung zusätzlich über das Volltext-LCD-Display auszugeben, das noch installiert werden soll.

Zu den weiteren Aufgaben zählt, wie gerade erwähnt, die Einrichtung eines Volltext-LCD-Displays, auf dem die wichtigsten Messwerte und Fehlermeldungen angezeigt werden. Das Display soll ein einfaches Ablesen der Werte ermöglichen, ohne dass das Web-Interface gestarten werden muss.
Um den Datensimulator effektiv nutzen zu können, ist es selbstverständlich nicht ausreichend, die über das Web-Interface eingegebenen Daten an den Daten-Server zu übermitteln. Eine sinnvolle Verwendung wird erst möglich sein, wenn der Druck- und Temperatursensor sowie der Sensor für die Luftfeuchtigkeit, die die Umgebungsbedingungen in der Rotorversuchshalle bestimmen, installiert wurden. Der Sensor für die Messung der Luftfeuchtigkeit muss zunächst noch ausgewählt werden. Um die Sensoren nutzen zu können, muss neben der Installation der Hardware auch die Software implementiert werden, sodass die ermittelten Informationen über die Umgebungsbedinungen an den Mikrocontroller übergeben werden, der sie wiederum an den Daten-Server weiterleitet.
Die gesamte Apparatur soll in einem Gehäuse (siehe Abbildung [8]) eingebaut und dann in der Rotorversuchshalle an der Wand befestigt werden. Hierfür muss allerdings noch entschieden werden, ob tatsächlich alle Bauteile im Gehäuse installiert werden können, ohne dass die Messungen verfälscht würden.
Zum einen ist nicht sicher, ob innerhalb des Gehäuses dieselben Verhältnisse herrschen wie außerhalb. Es wäre möglich, dass es sich mit der Zeit aufheizt, zum Beispiel durch Lichteinstrahlung. Eine mögliche Lösung wäre es, Öffnungen im Gehäuse vorzusehen, in die die Sensoren eingebaut werden. So könnten die Sensoren die Umgebungsbedingungen direkt in der Rotorversuchshalle messen.
Zum anderen muss die Wärmeentwicklung des Mikrocontrollers selbst beachtet werden.

Im Betrieb weist dieser eine hohe Wärmeabstrahlung auf, die die Messung verfälschen könnte. Um dies zu verhindern, könnte der Temperatursensor in einem separaten Gehäuse untergebracht werden, mit einem ausreichend großen Abstand zum Mikrocontroller. Zusätzlich sollte überprüft werden, ob die erwähnte Wärmeentwicklung den Mikrocontroller selbst schädigen könnte. Ist dies der Fall, muss zusätzlich ein Kühlkörper für den Mikrocontroller installiert werden.

Wie bereits eingangs erwähnt, wurden die Aufgaben dieses Semesters zufriedenstellend bearbeitet, das Gesamtprojekt ist allerdings längst noch nicht einsatzfähig. Die weiteren Programmbestandteile werden voraussichtlich Teil der Aufgabe des Modul Praxis II sein.

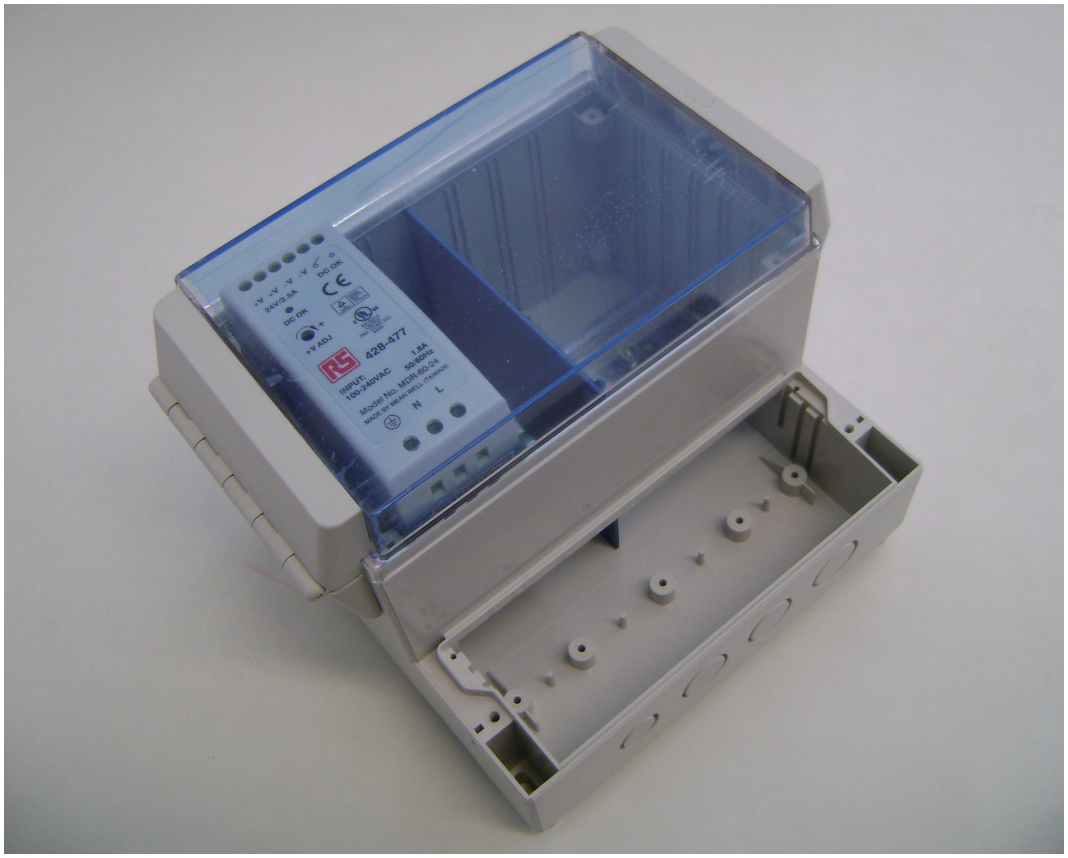

Abbildung 8: Vorgesehenes Gehäuse

Teil II

Datensynchronisator für Wetter- und Telemetriedaten

1 Einleitung

1.1 Aufgabenstellung

Die Testflüge des Forschungshubschraubers FHS werden aus der Telemetrie- und Telekommandoanlage, einer Bodenstation, heraus überwacht. An diese sendet der Hubschrauber einen Datenstrom mit Telemetriedaten, der unter anderem die GPS-Zeit sowie die aktuelle Run-Nummer des Fluges enthält. Der Datenstrom hat eine Aktualisierungsrate von 20 Hz. Parallel werden von einer Wetterstation meteorologische Daten an die Bodenstation übermittelt. Zu diesen Daten zählen Windgeschwindigkeit, Niederschlagsstatus und Temperatur sowie weitere Parameter. Das Ende des Datenstroms bilden Datum und Uhrzeit der Messung. Diese werden mit 1 Hz aktualisiert.
Bisher wurden die beiden Datenströme mit einem zeitaufwengigen Verfahren nachträglich synchronisiert. Die Aufgabe ist, die meteorologischen und die telemetrischen Datenströme zu empfangen und ihren Inhalt zusammen in eine Datei zu schreiben. So entfällt die spätere Synchronisation. Die Output-Dateien sollen in einem Pfad angelegt werden, der aus dem ersten Wetter-Datenstrom ermittelt wird.
Als Programmiersprache soll C++ verwendet werden.

Abbildung 9: Telemetrie- und Telekommandoanlage [7]

1.2 Programmiersprache C++

C++ zählt zu den höheren Programmiersprachen und wurde als Erweiterung der Programmiersprache C entwickelt. Genau wie C unterstützt es eine prozedurale Programmierung. Die größte Veränderung, die C++ gegenüber seinem Vorgänger aufweist, ist, dass es objektorientiert ist.
C++ verfügt zwar über eine Reihe eigener Standard-Bibliotheken, unterstützt allerdings auch sämtliche in ANSI C vereinbarten Standards. So ist es ohne Schwierigkeiten möglich, in C geschriebene Programmteile und Funktionen in ein C++-Projekt einzubinden.

2 Wetterdaten

Die meteorologischen Daten werden vom Wettermast per COM-Datenstrom an die Bodenstation übermittelt. COM ist eine serielle Schnittstelle, zu der jeweils nur ein Programm zurzeit eine Verbindung herstellen kann. Da aber noch ein weiteres Programm Zugriff auf diese Daten benötigt, müssen die meteorologischen Daten an zwei verschiedene, unabhängige Schnittstellen gesendet werden. Hierfür wird ein Y-Adapter verwendet, der den COM-Datenstrom, den er am Eingang erhält, an zwei Ausgängen bereitstellt. Nun liegen die meteorologischen Daten sowohl an COM1 als auch an COM2 an und ein paralleler Zugriff zweier Anwendungen ist möglich.
Der am COM-Port anliegende Datenstrom besteht aus einer Zeichenkette, die, wie bereits in Abschnitt 1 erwähnt, zunächst die meteorologischen Daten enthält und dann den zum Mess-Zeitpunkt gültigen Zeitstempel. Abgeschlossen wird die Zeichenkette durch ein *Carriage Return* (Wagenrücklauf) und ein *Line Feed* (Zeilenvorschub). Ein möglicher Datensatz könnte folgendermaßen aussehen:

```
08.3116   97   3.0   27.8   51.1   16.7   38.5 1008.0    0.0 0   537 1
    208.9 ---.- ----- ---- 20.08.09   8:18:42\r\n
```

Abbildung 10: Wettermast des DLR

3 Telemetriedaten

Die Telemetrie-Daten werden mittels des UDP-Ports von der Bodenstation empfangen. UDP ist ein Netzwerkprotokoll, das auf Verfahren verzichtet, die die Übertragung sicherstellen, wie zum Beispiel einen Hand-Shake. So ist nicht gewährleistet, dass alle Informationen ankommen oder dass sie nicht doppelt empfangen werden. Der Vorteil von UDP ist, dass es, gerade aufgrund der nicht implementierten Sicherungsverfahren, sehr schnell und schlank ist. [18]
Während eines Flugversuches werden die Telemetrie-Daten direkt vom Hubschrauber gesendet. Findet aktuell allerdings kein Flugversuch statt, ist es möglich, diesen zu simulieren. Die Aufzeichnung eines Flugversuches, die sämtliche empfangenen Telemetriedaten enthält, kann abgespielt werden. Hierbei liegen die UDP-Daten ebenfalls am UDP-Port der Bodenstation an. Dies ermöglicht ein einfaches Testen der Software, was während eines normalen Flugversuches nicht in dem Umfang durchzuführen wäre.
Der UDP-Datenstrom besteht aus 36 Zeichen, von denen in diesem Fall lediglich zweimal je 8 Byte relevant sind. Die relevanten Abschnitte enthalten die GPS-Zeit bzw. die Run-Nummer in Form einer Double, einer Gleitkommazahl mit doppelter Genauigkeit.

Abbildung 11: In-Flight-Simulator EC 135 - FHS [6]

4 Implementierung

4.1 Programmablauf

Sämtliche Output-Dateien sollen in dem Verzeichnis `D:\Wetterdaten` gespeichert werden. Der Verzeichnispfad ist in einer Header-Datei des Projekts definiert und ist somit leicht zu ändern, sollte dies einmal nötig sein. Ist dieser Ordner beim Programmstart nicht vorhanden, muss er logischerweise angelegt werden. Die Abfrage, ob der Ordner bereits vorhanden ist und der Vorgang des Erstellens, sollte dies nicht der Fall sein, werden durch folgende Code-Zeilen realisiert:

```
1  //Abfrage, ob der Ordner bereits existiert
2  if(!_access( MET_Params->storageLocation.c_str(),0 ) == 0) {
3
4    //String mit dem Befehl zum Ordneranlegen wird erstellt
5    string storage_folder = "mkdir ";
6    storage_folder.append(MET_Params->storageLocation);
7
8    //Befehl zum Anlegen des Ordners wird gegeben
9    system(storage_folder.c_str());
10
11   cout<<"Subfolder Created: "<<MET_Params->storageLocation.
         c_str()<<"\n";
12 }
```

Listing 2: Bedingtes Ordnererstellen

Innerhalb des Ordners `D:\Wetterdaten` werden die einzelnen Dateien in einen Unterordner gespeichert, der als Namen das aktuelle Datum aufweist. Dieses Verzeichnis wird ebenso angelegt wie der Ordner `D:\Wetterdaten` zuvor, sollte es noch nicht existieren. Der Name der Dateien fängt mit einem `a` an, das anzeigt, dass es sich um meteorologische Daten handelt. Auf dieses folgt der Zeitstempel des ersten Datensatzes und anschließend die Dateiendung `.met`. Ein möglicher Dateiname wäre dementsprechend `a20090829_103405.met`.

Der Telemetrie-Datenstrom wird mit 20 Hz aktualisiert, die meteorologischen Daten hingegen nur mit 1 Hz. Das Programm ist so aufgebaut, dass immer, wenn neue meteorologische Daten empfangen werden, diese zusammen mit den aktuellen Telemetrie-Daten in eine Datei geschrieben werden. Sollten keine aktuellen Telemetrie-Daten verfügbar sein, wird stattdessen der Platzhalter `-----.--- ---` in die Datei geschrieben, sodass das Fehlen auf den ersten Blick ersichtlich ist.
Der am UDP-Port anliegende Datenstrom der Telemetrie-Daten wird nur aktualisiert, wenn

ein neuer Datensatz verfügbar ist, andernfalls liegen noch die alten Informationen am UDP-Port an. So wird zunächst fälschlicherweise der Eindruck vermittelt, es würden weiterhin aktuelle Datensätze empfangen. Wird dies nicht abgefangen, so werden diese zusammen mit den meteorologischen Daten in die Output-Datei geschrieben, was zu falschen Ergebnissen führt. Um diesen Fehler zu beseitigen, wird bei jedem neu eingelesenen UDP-Datensatz die GPS-Zeit mit der des vorhergehenden verglichen. Da das Format der GPS-Zeit eine Angabe der Sekunden nach Mitternacht ist und nie ein Flugversuch über Mitternacht ausgeführt werden wird, ist es hier ausreichend zu überprüfen, ob die neue GPS-Zeit vom Wert her größer ist als die vorhergehende. Ist dies nicht der Fall, so wurde kein neuer Datensatz empfangen und es wird stattdessen der bereits erwähnte Platzhalter in die Datei geschrieben.
Eine vereinfachte Form des Programmablaufplans ist in Abbildung [12] dargestellt.

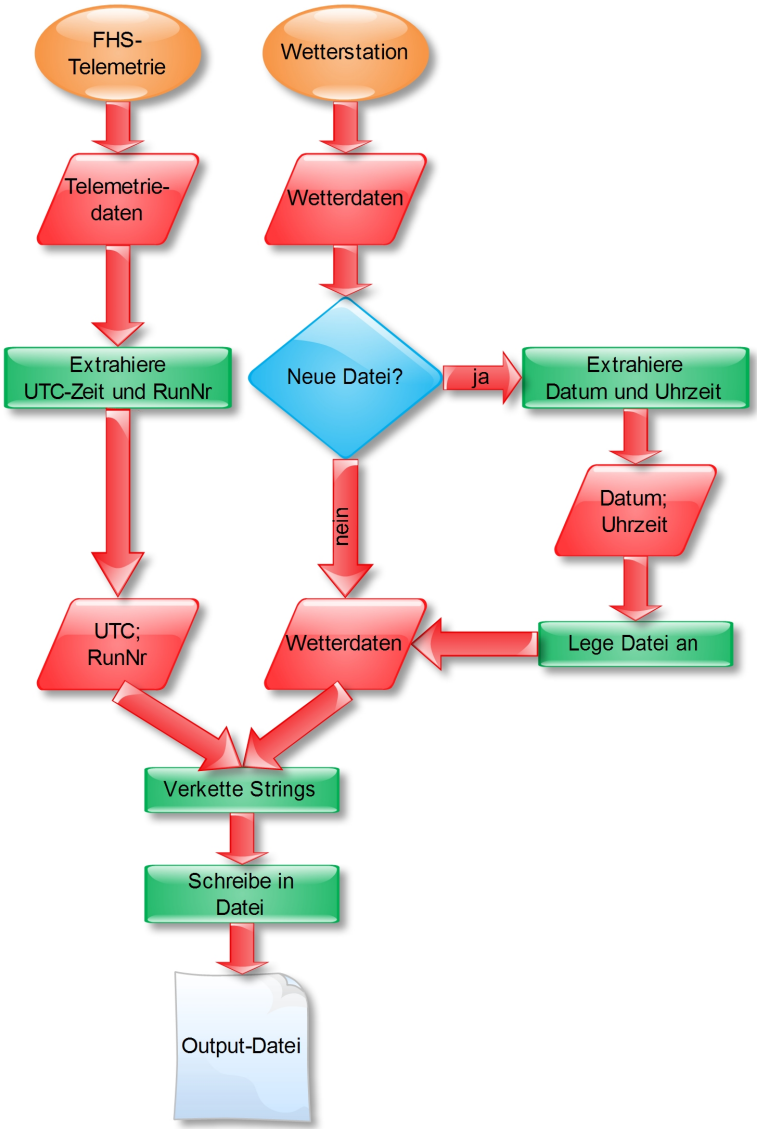

Abbildung 12: Vereinfachter Programmablaufplan des Datensynchronisators

4.2 Programmaufruf

Das erstellte Programm ist eine Win32-Konsolen-Anwendung. Beim Programmaufruf können die Nummer des COM-Ports, die Anzahl der Datensätze pro Datei sowie die UDP-Port-Nummer übergeben werden. Sollten keine Parameter beim Programmaufruf übergeben werden, wird das Programm mit den Standardeinstellungen gestartet. Diese sind COM-Port 1, 600 Datensätze pro Datei und UDP-Port 4488.

4.3 Konsolenausgabe

Während das Programm läuft wird in der Windows-Konsole eine Ausgabe produziert. An dieser Ausgabe ist zunächst einmal zu erkennen, ob meteorologische Daten empfangen werden. Wenn dies der Fall ist, wird es durch eine animierte Ausgabe angzeigt. Bricht der Datenstrom ab, friert die Animation ein. Die Animation besteht aus einem Strich, der sich um sich selbst dreht.
Weiterhin informiert die Ausgabe darüber, ob aktuell ein UDP-Datenstrom anliegt. Sollten keine Datenpakete empfangen werden, wird der Text `No UDP Data` angezeigt. Andernfalls sind die GPS-Zeit und die Run-Nummer zu sehen, die der UPD-Datensatz enthält.
Die Ausgabe könnte folgendermaßen aussehen:

```
/ UDP GPS Time 27134.198 RunNo 004
```

5 Fazit und Ausblick

Das erstellte Programm erfüllt sämtliche an es gestellten Anforderungen. Das Verhalten des Programms in den getesteten Situationen ist definiert und es kommt zu keinem Programmabsturz. Zu den erwähnten Situationen gehören beispielsweise ein Sendungsabbruch durch ein loses Kabel oder eine falsche Benutzereingabe.
Der erstellte Datensynchronisator soll bereits beim nächsten Flugversuch eingesetzt werden. Zur Absicherung wird vorerst zusätzlich das alte System als Backup mitlaufen. So ist sichergestellt, dass sämtliche Daten gespeichert werden, auch wenn der erstellte Datensynchronisator abstürzen oder sich unerwartet verhalten sollte.

Eine mögliche Erweiterung des Programms ist eine grafische Oberfläche. Damit verbunden wäre eine Erhöhung des Nutzerkomforts, da die Übersichtlichkeit erhöht werden würde. Außerdem wäre es so einfacher umzusetzen, dass bestimmte Parameter während der Laufzeit verändert werden. So wäre es zum Beispiel auch nach dem Programmstart möglich, die Anzahl der Datensätze zu ändern, die in eine Datei geschrieben werden.

Literatur

[1] Kernighan, Brian W.; Ritchie, Dennis M.: „Programmieren in C“, 2. Ausgabe, ANSI C, Carl Hanser Verlag München, ISBN 987 3 446 15497 1

[2] C++ Buch

[3] http://www.dlr.de/desktopdefault.aspx/tabid-636/1065_read-1465/
16.03.09

[4] http://www.dlr.de/ft/desktopdefault.aspx/tabid-1387/1915_read-3372/
10.03.09

[5] http://www.dlr.de/Portaldata/1/Resources/ueber_dlr/erde_dlr.jpg
16.09.09

[6] http://www.dlr.de/Portaldata/1/Resources/veranstaltungen/fhs.jpg
16.09.09

[7] http://www.dlr.de/ft/Portaldata/18/Resources/images/
flugversuchstechnik/mtta.jpg
16.09.09

[8] http://www.dlr.de/ft/desktopdefault.aspx/tabid-1359/1862_read-3347/
16.03.09

[9] http://www.mikrocontroller.net/articles/Mikrocontroller
11.03.09

[10] http://de.wikipedia.org/wiki/Mikrocontroller
11.03.09

[11] http://de.wikipedia.org/wiki/Cross-Compiler
16.09.09

[12] http://www.beck-ipc.com/files/infosheet/is_sc11_sc13_de_v4.pdf
11.03.09

[13] http://www.beck-ipc.com/files/datasheet/DS_SC13_V5.pdf
16.09.09

[14] http://www.beck-ipc.com/files/manual/IPCatCHIP_Gettingstart_V23.pdf
06.03.09

[15] http://www.beck-ipc.com/files/manual/IPCatCHIP_RTOS_DOCUMENTATION_
Build_090216.pdf
25.02.09

[16] http://www.vti.fi/midcom-serveattachmentguid-9cbae6a382efd245cb62354a54ff62c7/scp1000-d01_-d11_pressure_sensor_datasheet_28-08-2007.pdf
16.03.09

[17] http://www.vti.fi/midcom-serveattachmentguid-30de88ec8eb89a476cb8c8e70b969dc3/scp1000_product_family_specification_rev_0.08.pdf
06.07.09

[18] http://tools.ietf.org/html/rfc768
16.09.09