Bachelor's Thesis

# Aufbau einer Schnittstelle zwischen MATLAB und einer Wetterstation über MODBUS
# Development of a MATLAB gateway to a hardware weather station via MODBUS

verfasst von

Andreas Henneberger

**Matr.Nr. 2647351**

eingereicht am

**Lehrstuhl für Energiewirtschaft und Anwendungstechnik**
**Technische Universität München,**

bei

**Prof. Dr. rer. nat. Thomas Hamacher**

Betreuer: Dipl.-Ing. Christian Kandler und Dipl.-Ing. Patrick Wimmer

**Zusammenfassung**

Ziel dieser Arbeit ist es eine Schnittstelle zwischen MATLAB und einer Wetterstation aufzubauen, um darüber Prognosedaten für ein integriertes Energiemanagementsystem bereitzustellen. Diese Informationen sollen dazu dienen die Planungen des Managementsystems im Smart-Micro-Grid hinsichtlich Lastverläufe und Energieerzeugung zu vereinfachen bzw. zu präzisieren. Die Datenbereitstellung erfolgt über einen Langwellenempfänger, dessen Register über eine MODBUS Kommunikation abgerufen werden können. Die meisten gelieferten Werte weisen eine zeitliche Auflösung von 6 Stunden auf. Da das Managementsystem jedoch umso genauer arbeiten kann, je niedriger diese Auflösung ist, ist es mit Aufgabe der Schnittstelle, die Daten in kleineren Zeitintervallen zur Verfügung zu stellen. Der Datenabruf und die Verarbeitung sollen in anderen MATLAB Programmen zum Einsatz kommen. Es ist daher zweckmäßig den Kommunikationsprozess als MATLAB Funktion mit entsprechenden Übergabeparametern zu implementieren. Um die geforderten Aufgabenziele zu erreichen, wurden die Spezifikationen der Wetterstation und des MODBUS-Protokolls analysiert. Mit den aus der Analyse gewonnen Informationen und den in MATLAB zur Verfügung stehenden Methoden, wurde letztlich das Programm umgesetzt. Wie der Leser am Ende der Arbeit feststellen kann, ergibt ein Vergleich der interpolierten Daten mit genauen Wetteraufzeichnungen der LMU ein differenziertes Bild. ...

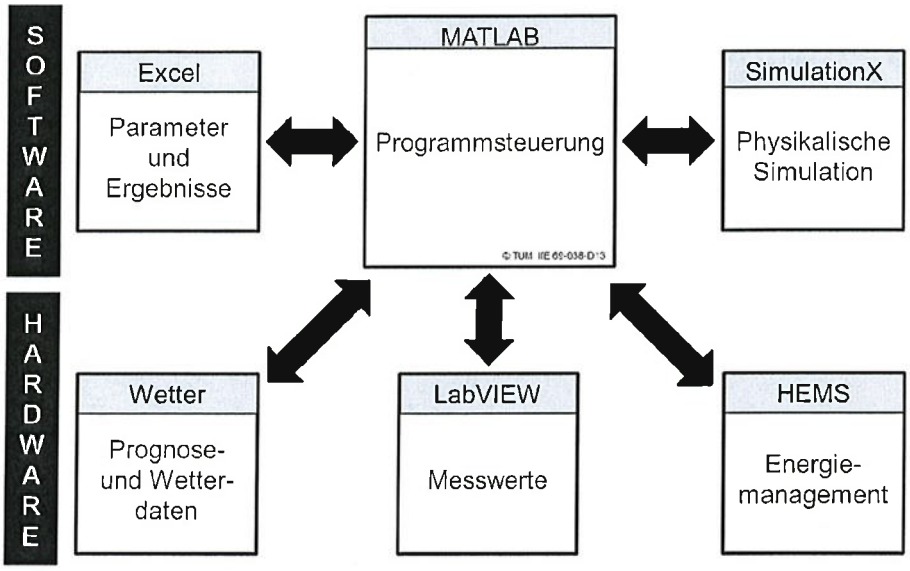
Lehrstuhl für
Energiewirtschaft und Anwendungstechnik
Ordinarius Prof. Dr.-Ing. U. Wagner · Prof. Dr. rer. nat. Th. Hamacher (komm.)

Technische Universität München

**Aufgabenstellung**
**Bachelor´s Thesis**
von
**Herrn HENNEBERGER Andreas**
Matr.-Nr. 2647351

**Thema:**
**Aufbau einer Schnittstelle zwischen MATLAB und einer Wetterstation über MODBUS**

**Development of a MATLAB gateway to a hardware weather station via MODBUS**




Das im Rahmen des Schaufensters Elektromobilität geförderte Forschungsprojekt **e-MOBILie - Energieautarke Elektromobilität im Smart-Micro-Grid** hat es sich zum Ziel gesetzt, in einem integrativen Ansatz elektrische Mobilität mit lokaler regenerativer Stromerzeugung zu verknüpfen.
Um ein handlungsfähiges Energiemanagement im Smart-Home gewährleisten zu können, müssen vorab Prognosen für verschiedene physikalische Größen (Temperatur, Einstrahlung, PV-Erzeugung, Lastverlauf, Preise…) erstellt bzw. von extern bezogen und bewertet werden.

Im Rahmen dieser Arbeit soll dazu eine Kommunikationsschnittstelle in MATLAB erstellt werden, welche via MODBUS die Abfrage einer Hardware-Wettervorhersagestation ermöglicht.

Betreuer: Dipl.-Ing. C. Kandler/ Dipl.-Ing. P. Wimmer Tel.: 089-289-28310

Ausgabedatum: 23.09.2013

Aufgabensteller: Prof. Dr.rer.nat. T. Hamacher

# Rechtserklärung

Hiermit erkläre ich,
Name: Henneberger
Vorname: Andreas Helmut
Mat.Nr.: 2647351

dass ich die beiliegende Bachelor's Thesis zum Thema:

**Aufbau einer Schnittstelle zwischen Matlab und einer Wetterstation über MODBUS**

selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, sowie alle wörtlichen und sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet und die entsprechenden Quellen angegeben habe.
Vom Lehrstuhl und seinen Mitarbeitern zur Verfügung gestellte Hilfsmittel, wie Modelle oder Programme, sind ebenfalls angegeben. Diese Hilfsmittel sind Eigentum des Lehrstuhls bzw. des jeweiligen Mitarbeiters. Ich werde sie nicht über die vorliegende Arbeit hinaus weiter verwenden oder an Dritte weitergeben.

Einer weiteren Nutzung dieser Arbeit und deren Ergebnisse (auch Programme und Methoden) zu Zwecken der Forschung und Lehre, stimme ich zu.

Ich habe diese Arbeit noch nicht zum Erwerb eines anderen Leistungsnachweises eingereicht.

München,

.....
Andreas Henneberger

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Teil I

# Einleitung

In Zukunft wird die Mobilität durch Elektroautos mit geprägt sein. Damit Deutschland auf diesem Technologiefeld eine Spitzenposition einnehmen kann, wurde von der Bundesregierung die Nationale Plattform Elektromobilität initiiert. Ziel dieser Institution ist es Deutschland bis zum Jahr 2020 zum Leitmarkt und Leitanbieter zu entwickeln. Marktvorbereitung, Markthochlauf und der Massenmarkt sind dabei die zu durchlaufenden Phasen. In der Marktvorbereitungsphase, in der wir uns zur Zeit befinden, werden die Ergebnisse aus Forschung und Entwicklung genutzt, um in vier sogenannten Schaufenstern die Modelle und Prognosen für den Markthochlauf zu validieren bzw. bei auftretenden Abweichungen anzupassen.[2]Eines dieser Schaufenster, genannt "Elektromobilität verbindet" wird von den Bundesländern Bayern und Sachsen betreut und finanziert. Das Schaufenster ist aufgegliedert in vier Teilprojekte von denen eines sich den Energiesystemen widmet. Das Themengebiet Energiesysteme ist wiederum in 9 Aufgabengebiete unterteilt, wovon sich eines mit der Integration der Elektromobilität in die dezentrale regenerative Energieversorung beschäftigt. Ein Aufgabenschwerpunkt hierbei ist es ein integriertes Energiemanagementsystem mittels Aufbau und Betrieb eines Hardware-in-the-Loop Prüfstands zu evaluieren. Da das Energiemanagementsystem auch Vorausschautechnologien einbinden soll, ist es erforderlich Prognosedaten zu erheben.[3]

# Teil II

# Hauptteil

# Kapitel 1

# Aufbau der Wetterstation

## 1.1 Aufbau der Datenstruktur

An dieser Stelle der Bachelorarbeit sollen die grundlegenden Eigenschaften der verwendeten Wetterstation dargestellt werden. Eine gute Kenntnis der Datenstruktur sowie der Datenbereitstellung sind eine zwingende Voraussetzung für den späteren Aufbau der MATLAB Funktion. Die nachfolgenden Angaben in diesem Kapitel können der Hardware Spezifikation entnommen werden [**?**] Der Hersteller bietet für die Hardware eine Reihe von Lizenzmodellen an, die den Empfang der Datenmenge bestimmt. Das in dieser Arbeit zum Einsatz kommende Modell nennt sich "WS-K RTU485 WPAia T" und beinhaltet das Prognosepaket "Premium All inclusive advanced", welches es ermöglicht, das komplette Spektrum an Prognosedaten abzurufen. Welche Wetterinformationen genau zur Verfügung stehen, kann der unten aufgeführten Grafik entnommen werden. Für welche Bereiche diese metereologischen Daten zutreffen muss in der Wetterregion spezifiziert werden. Hier besteht die Möglichkeit für über 1000 Städte in fast ganz Europa die Wetterprognosen abzufragen [1, S. 27-38]. Wie schon in der Einleitung erwähnt, wäre eine niedrige zeitliche Auflösung der Daten wünschenswert, damit das Energiemanagementsystem ohne große Verwerfungen planen kann. Jedoch liegen die meisten Daten in einer Auflösung von 6 Stunden vor, d.h. für ein Intervall von morgens, mittags, nachmittags und abends. Lediglich die mittlere Lufttemperatur wird in einer 1 stündigen Auflösung bereitgestellt. Dieser Umstand wird später im Programmablauf gesondert berücksichtigt. Ein Update der Daten erfolgt ebenfalls alle 6 Stunden. Neben der Auflösung unterscheidet sich auch der Prognosehorizont innerhalb der zugänglichen Daten. Die Spanne reicht von einem bis zu drei Folgetagen. Für den aktuellen Tag, liegen für alle Bereiche Daten vor. Welche metereologische Ausprägung welche Eigenschaften besitzt, kann in der Tabelle **A.1** im Anhang nachvollzogen werden.

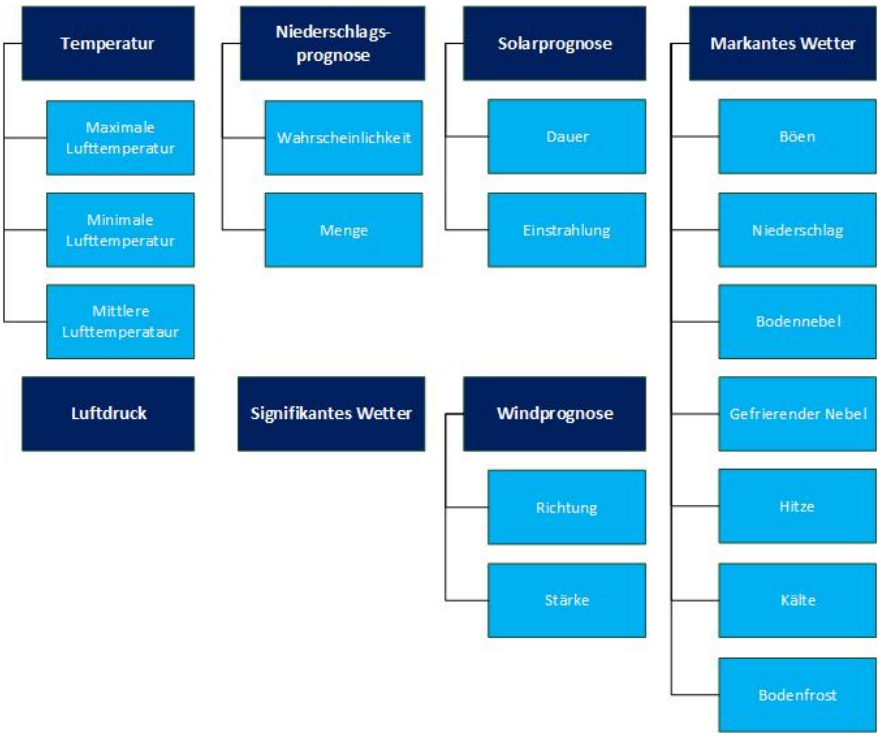



Abbildung 1.1: Verfuegbare Prognosedaten WS-K RTU485 WPAia

## 1.2 Technischer Aufbau der Station

### 1.2.1 Senderauswahl und Stationsaufbau

Die eingesetzte Wetterstation erhält ihre Daten via Langwelle von drei auswählbaren Sendern:

- Sender Mainflingen DCF 49
- Sender Burg DCF 39
- Sender Lakihegy HGA 22 (Ungarn)

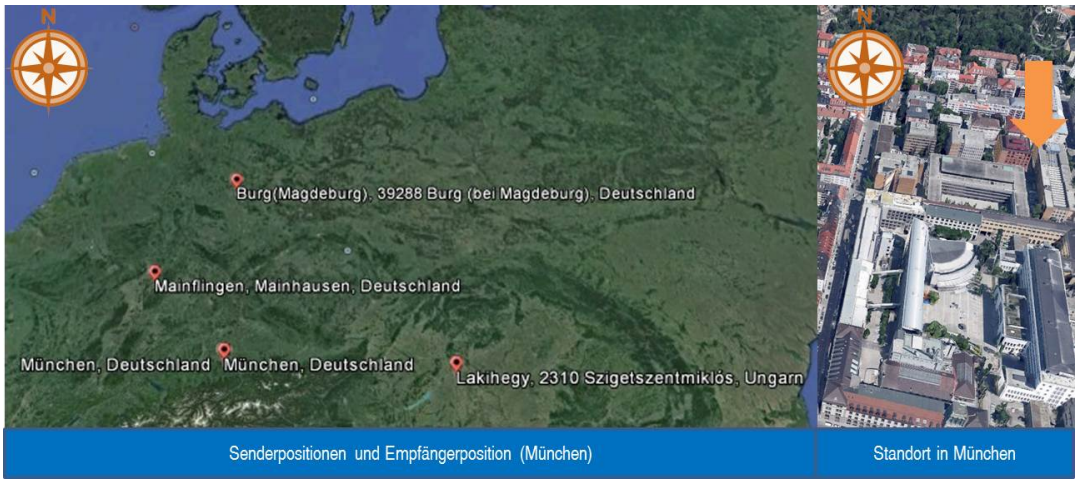



Abbildung 1.2: Standorte der Langwellensender und des Empfängers

Tabelle 1.1: Einstellungsparameter für den Kommunikationsaufbau und den Wetterbereich

| Register-adresse | Bezug | Zugriff | Datentyp | Bereich | Bemerkung |
|---|---|---|---|---|---|
| 110 | Senderstation | Lesen/Schreiben | unsigned | 0,1,2 | 0 = DCF 49<br>1 = HGA 22<br>2 = DCF 39 |
| 111 | Empfangsqualität | Lesen | unsigned | 0...9 | 9 ist höchste Qualität |
| 112 | Stadt ID | Lesen/Schreiben | unsigned | 0...1022 | |
| 100 | Sekunde (Funkuhr) | Lesen | unsigned | | UTC |
| 101 | Minute (Funkuhr) | Lesen | unsigned | | UTC |
| 102 | Stunde (Funkuhr) | Lesen | unsigned | | UTC |
| 103 | Tag (Funkuhr) | Lesen | unsigned | | UTC |
| 104 | Monat (Funkuhr) | Lesen | unsigned | | UTC |
| 105 | Jahr (Funkuhr) | Lesen | unsigned | | UTC |

Die Wetterstation soll in München aufgebaut werden. Um einen guten Empfang gewährleisten zu können, muss sie entsprechend ausgerichtet werden. Der Hersteller gibt hierzu Kriterien vor, die beachtet werden sollten:

- senkrechter Aufbau des Gehäuses mit nach unten austretendem Kabelstrang
- für einen Innenaufbau in der Nähe zum Fenster
- Mindestabstand von 30 cm zu Metallkonstruktionen oder -flächen
- ausreichende Entfernung zu Geräten die elektromagnetisch abstrahlen
- keine direkte Sonnenbestrahlung für das Einbinden der lokalen Temperatur
- ausreichender Bodenabstand, um Einschneien zu vermeiden
- Ausrichtung zum geografisch günstigsten Sender

Unter Berücksichtigung dieser Empfehlungen wurde die Station in Fensternähe in nordwestlicher Richtung aufgebaut und die Sendestation Mainflingen vorgegeben.

### 1.2.2 Registereinteilung und Schnittstellenparametrierung

Wie im vorigen Kapitel bereits erläutert, können über das MODBUS Protokoll vier Arten von Registern angesprochen werden. In der Wetterstation sind zwei Register für die Kommunikation vorgesehen. Im Holdingregister können Einstellungsparameter gesetzt und gelesen werden. Eine Übersicht gibt die oben aufgeführte Tabelle **1.1**. Außerdem sind sämtliche metereologischen Daten in diesem Register abgelegt. Eine Auflistung der den Prognosebereichen zugeordneten Registeradressen gibt die im Anhang befindliche Tabelle A.1. Es ist dabei zu beachten, dass

die Adressen gegenüber den in der Spezifikation des Herstellers Angegebenen, bereits auf die Struktur des Holdingregisters angepasst wurden. D.h. da das Holdingregister mit einer 0 beginnt, wurde von jeder Adresse eine Position abgezogen. Neben dem Holdingregister gibt es noch das Coilregister, in welchem die Zustände für den externen Temperatursensor und die FSK Qualität vorgehalten werden. Die Adressen hierfür sind 1 bzw. 2 und die zugelassenen Werte 1 und 0 geben jeweils den Zustand an. 1 bedeutet der Sensor sowie die FSK Qualität sind in Ordnung. Um eine funktionierende MODBUS Kommunikation über eine serielle Schnittstelle aufbauen zu können, müssen bestimmte Schnittstellenparameter beim Empfänger sowie beim Sender identisch sein. Die Baudrate gibt dabei an, wieviele Bits pro Zeiteinheit übertragen werden können [4, S .169]. Das Paritätsbit zeigt an, ob im übertragenen Byte ein 1 bit Fehler vorliegt. Dies ist dann der Fall, wenn die Zahl der übertragenen Einsen ungerade ist, jedoch das Paritätsbit (= 0 bei even und 1 bei odd) eine gerade Anzahl anzeigt. Das Kommunikationprotokol erfordert in diesem Fall eine erneute Übertragung der Information [4, S. 25]. Das Databit gibt lediglich die Anzahl der zu übertragenden bits pro Codewort an. Das Stopbit, entweder 1 oder 2 bits, wird zur Synchronisierung von Sender und Empfänger benötigt, damit diese wissen, wann ein Codewort beginnt und endet [4, S. 169-170]. Per Jumperpositionierung kann die Baudrate von 9600 auf 19200 in der Wetterstation geändert werden, die Werkseinstellung von 19200 wurde aber in dieser Arbeit beibelassen. Ebenso einstellbar über Jumper ist die Slave-Adresse der Wetterstation. Aber auch hier wurde der voreinstellte Wert nicht verändert. Nachdem nun das Modbusprotokoll und die Funktionsweise der Wetterstation bekannt sind, kann mit der Umsetzung der MATLAB Funktion begonnen werden.

Tabelle 1.2: Schnittstellenparameterbelegung

| **Parameter** | **Wert** |
|---|---|
| Baudrate | 19600 |
| Parität | even |
| Databit | 8 |
| Stopbit | 1 |
| Slave-Adresse | 03 |

# Kapitel 2

# Das Modbus-Protokoll

Nachdem die Wetterstation über MODBUS kommuniziert, soll in diesem Kapitel das MODBUS Protokoll näher erläutert werden. Dabei wird überwiegend Bezug auf die offizielle MODBUS Spezifikation genommen [5]. Angesiedelt auf der ersten, zweiten und siebten Ebene des OSI Modells und damit einfach zu handhaben, ist MODBUS als Kommunikationsprotokoll in der Industrie weit verbreitet. Die unten stehende Abbildung **2.1** dient zur ersten Orientierung der nachfolgend behandelten Themengebiete.

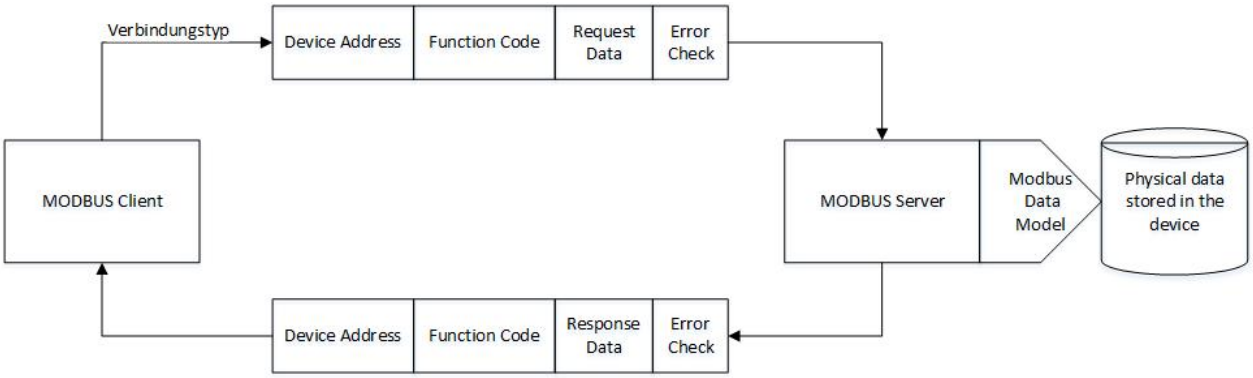



Abbildung 2.1: Skizze der Funktionsweise des MODBUS Protokolls

## 2.1 Verbindungstypen

Es ist möglich das Protokoll auf drei Verbindungstypen zwischen Client und Server einzusetzen. Dazu zählen:

- eine Internetverbindung TCP/IP
- eine asynchrone serielle Verbindung (z.B. RS-232, RS-422, RS-485, etc.)
- eine MODBUS Plus Verbindung

In dieser Arbeit ist die Wetterstation seriell über eine RS-485 Schnittstelle mit dem Rechner verbunden. Die RS-485 Schnittstelle bietet den Vorteil, dass die Verbindung der Netzwerkteilnehmer wie bei einer RS-232 Schnittstelle nur über eine Zweidrahtleitung erfolgen kann. Jedoch

können im Gegensatz zur RS-232 Schnittstelle bis zu 32 Teilnehmer im Netzwerk angeschlossen werden. Die Netzwerklänge kann ohne Verstärker bis zu 1200m betragen [6]. Diese Eigenschaften bieten sich an, um die Wetterstation in ein Netzwerk zu integrieren, welches von einem Energiemanagementsystem gesteuert wird.

## 2.2 Nachrichtenaufbau

Wie eine typische MODBUS Nachricht aufgebaut ist, zeigt die unten stehende Abbildung **2.2**. Die PDU ist unabhängig vom Netzwerk auf dem das Protokoll eingesetzt wird und setzt sich

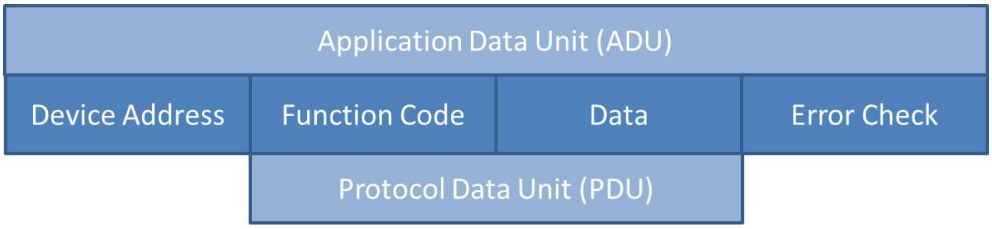



Abbildung 2.2: Aufbau einer MODBUS Nachricht

aus dem Funktionscode und den zu übermittelnden Daten zusammen. Mit einem Byte codiert gibt der Funktionscode an, ob eine schreibende oder lesende Kommunikation an welcher Art Register vorgenommen werden soll. Er kann aber auch einfach nur eine Aktion ausführen. In der Spezifikation werden drei Funktionscodearten genannt, öffentliche, benutzerdefinierte und reservierte Funktionscodes von denen in dieser Arbeit aber nur die Öffentlichen interessieren. Im Falle einer Anfrage des Client, enthält der Datenblock die entsprechenden Informationen über die genaue Adresse und Anzahl der zu lesenden oder beschreibenden Register. Für einen Schreibprozess wird hier auch der notwendige Input angegeben. Die abgefragten Daten des Servers sind ebenfalls im Datenblock untergebracht. Die Device-Adresse und der Error-Check sind Informationen, die für das Netzwerk sprich den Verbindungstyp zwischen den Geräten eine Rolle spielen. Wie eben kurz skizziert, unterscheidet das MODBUS Protokoll zwischen drei Arten von PDUs, die nachfolgend zusammengefasst aufgeführt sind:

- die Anfrage-PDU besteht aus dem Funktionscode und den Anfragedaten
- die Antwort-PDU besteht ebenfalls aus einem Funktionscode und den Antwortdaten
- die Fehler-PDU besteht aus dem Fehlerfunktionscode (Funktionscode + 0x80) und der Fehlermeldung

Die Bytereihenfolge im Datenblock folgt der big-endian Anordnung, d.h. das Most Significant Bit kommt an erster Stelle und das Least Significant Bit an Letzter.

## 2.3 Registertypen

Die Daten, die vom Client abgefragt werden können, müssen physikalisch im Speicher des Servers liegen. Eine Verknüpfung dieses Speichers mit den zur Verfügung stehenden Registern im

Tabelle 2.1: Übersicht der Registerarten im MODBUS Protokoll

| Register | Wortlänge | Zugriff | Info |
|---|---|---|---|
| Diskreter Input | 1 bit | Lesen | Daten werden durch ein I/O System bereitgestellt |
| Coils | 1 bit | Lesen/Schreiben | Daten können über ein Anwendungsprogramm geändert werden |
| Input Register | 16 bit | Lesen | Daten werden durch ein I/O System bereitgestellt |
| Holding Register | 16 bit | Lesen/Schreiben | Daten können über ein Anwendungsprogramm geändert werden |

MODBUS Protokoll ermöglicht den Zugriff. Es werden vier Registerarten unterschieden, die in der Tabelle **2.1** aufgezeigt sind. Jedes dieser Register besitzt einen Adressraum der bei 0 beginnt und bei 65535 endet.

## 2.4 Nachrichtenverarbeitung

Die Abbildung **B.1** im Anhang zeigt den Ablauf einer Nachrichtenüberprüfung und an welcher Position welcher Fehlercode gesendet wird, wenn die Nachricht fehlerhaft ist.

## 2.5 Funktionscodes

Da der Funktionscode ein entscheidender Baustein in der MODBUS Nachricht ist, ist es wichtig die für die Zwecke dieser Arbeit Wichtigen zu identifizieren. Die im Anhang dargestellete Abbildung **B.2** zeigt die zur Verfügung stehenden Funktionscodes. Gelb markiert sind dabei die Codes, die für die Kommunikation zwischen MATLAB und der Wetterstation Bedeutung haben. Wie schon im Abschnitt 1.2.2 auf Seite 13 beschrieben, müssen für den Fall einer Abfrage der Zustände des Temperatursensors oder der FSK Qualität die Coiladressen 0 oder 1 ausgelesen werden. Hierzu reicht es also jeweils eine Adresse in der MODBUS Nachricht anzugeben und die auszulesende Adresszahl auf 1 zu setzen. Die Anfrage- und Antwortnachricht für einen funktionierenden Temperatursensor ist in der Tabelle **2.2** beispielhaft dargestellt. Alle zwei Byte breiten Worte, wie zum Beispiel die Startadresse, setzen sich aus einem sogenannten High-Byte (H-Byte) und einem Low-Byte (L-Byte) zusammen. Ein weiterer wichtiger Funktionscode ist der, mit dem man in das Holding-Register Werte schreiben kann. Wie im Abschnitt 1.2.2 auf Seite 12 in der Tabelle 1.1 nachgelesen werden kann, wird die zu beobachtende Wetterregion mit einem Wert im Holdingregister an der Adresse 112 festgelegt. Die hierzu notwendige Kommunikation ist in der Tabelle 2.3 als Beispiel skizziert. Der wohl wichtigste und am meisten verwendete Funktionscode in dieser Arbeit ist der zum Lesen des Holding-Registers. Über ihn werden sämtliche Prognosedaten ausgelesen. Auch hier soll ein Beispiel in der Tabelle 2.4 den Aufbau verdeutlichen. In dem gezeigten Beispiel werden alle Wetterdaten, insgesamt 96 Werte, für die Mittlere Temperaturprognose abgerufen. Schlägt ein Kommunikationsprozess fehl,

Tabelle 2.2: Aufbau einer lesenden Kommunikation mit einem Coil-Register

| Nachrichten-typ | Nachrichten-teil | Wort-länge | Inhalt |
|---|---|---|---|
| Anfrage | Funktionscode | 1 Byte | 0x01 |
| | Startadresse | 2 Bytes | H-Byte 0x00 L-Byte 0x00 (0x0000 bis 0xFFFF möglich) |
| | Adressanzahl | 2 Bytes | H-Byte 0x00 L-Byte 0x01 (1 bis 2000 (0x7D0) möglich) |
| Antwort | Funktionscode | 1 Byte | 0x01 |
| | Byteanzahl | 1 Byte | 1 (Ist das Ergebnis von Adressanzahl mod 8 = 0, so ergibt sich die Byteanzahl aus dem Ergebnis der Adressanzahl dividiert durch 8, andernfalls wird um ein Byte erhöht.) |
| | Coil Status | n Bytes | 00000001 (8 Coilzustände werden mit einem Byte angezeigt. Das Most Significant Bit im Antwort Byte steht dabei für die höchste Registeradresse.) |

Tabelle 2.3: Aufbau einer schreibenden Kommunikation mit einem Holding-Register

| Nachrichten-typ | Nachrichten-teil | Wort-länge | Inhalt |
|---|---|---|---|
| Anfrage | Funktionscode | 1 Byte | 0x06 |
| | Registeradresse | 2 Bytes | H-Byte 0x00 L-Byte 0x70 (0x0000 bis 0xFFFF möglich) |
| | Registerinput | 2 Bytes | H-Byte 0x01 L-Byte 0x61 (0x0000 bis 0xFFFF möglich) |
| Antwort | Funktionscode | 1 Byte | 0x06 |
| | Registeradresse | 2 Byte | H-Byte 0x00 L-Byte 0x70 |
| | Registerinput | 2 Byte | H-Byte 0x01 L-Byte 0x61 |

Tabelle 2.4: Aufbau einer lesenden Kommunikation mit einem Holding-Register

| Nachrichten-typ | Nachrichten-teil | Wort-länge | Inhalt |
|---|---|---|---|
| Anfrage | Funktionscode | 1 Byte | 0x03 |
| | Startadresse | 2 Bytes | H-Byte 0x00 L-Byte 0x00 (0x0000 bis 0xFFFF möglich) |
| | Adressanzahl | 2 Bytes | H-Byte 0x00 L-Byte 0x5F (1 bis 125 (0x7D) möglich) |
| Antwort | Funktionscode | 1 Byte | 0x03 |
| | Byteanzahl | 2 Byte | 2 x N (N = Adressanzahl) |
| | Registeroutput | N x 2 Bytes | |

so wird vom Server statt der Antwortnachricht eine Fehlernachricht gesendet. Es sind folgende Fehlernachrichten vorgesehen:

- Code 01 Ungültige Funktion
- Code 02 Ungültige Adressdaten
- Code 03 Ungültige Daten
- Code 04 Fehler beim MODBUS Server

Code 1 kann auftreten, wenn die entsprechende Funktion im Gerät nicht implementiert ist oder der Server sich in einem falschen Zustand befindet. Code 02 wird dann gesendet, wenn in der Anfrage mehr Register ausgelesen werden sollen, als zur Verfügung stehen. Code 03 gibt an, dass es sich bei dem im Datenblock befindlichen Wert um einen für den Server nicht Gültigen handelt. Code 04 wird übermittelt, wenn beim Server während der Bearbeitung der Anfrage ein Fehler aufgetreten ist.

# Kapitel 3

# Aufbau und Dokumentation des Funktionscodes

## 3.1 Geforderte Funktionseigenschaften

Die zu schreibende MATLAB Funktion soll nach Fertigstellung in weiteren MATLAB Programmen zum Einsatz kommen. Daher ist es wichtig, dass sämtliche Daten die für das Ausführen erforderlich sind bereits im Code vorliegen und nicht importiert werden müssen. Es sollen auch sonst nach dem Ausführen keine weiteren Maßnahmen oder Eingaben getätigt werden müssen. Um diese Voraussetzungen zu erfüllen müssen vor allem große Datensätze im Code eingebunden werden. Bei dieser Arbeit ist das zum einen die Städteliste mit ihren über 1000 Einträgen und zum anderen das Verzeichnis mit allen Registeradressen, in Summe über 340 Positionen. Daneben gibt es noch ein paar weitere Eigenschaften, die an dieser Stelle kurz aufgeführt werden.

- wiederholte Ausführung des Datenabrufs in bestimmten Zeitabschnitten ohne dabei MATLAB komplett zu blockieren
- Handhabung der kompletten MODBUS Kommunikation, insbesondere des Datenabrufs, der -verarbeitung und der Parametersetzung
- Interpolation der ausgelesenen Werte, um unterschiedliche zeitliche Auflösungen zu erhalten
- Datensicherung derart, dass jeder Datenabruf in einer eigenen Datei und die Summe aller abgerufenen Werte in einer anderen Datei gespeichert werden
- Fehlervermeidung bei der Eingabe von Inputparametern
- Aufbau und Beendigung der seriellen Schnittstelle mit der Wetterstation
- einfache und kurze Inputparameter

Nachdem die Eigenschaften nun bekannt sind, soll in den nachfolgenden Unterkapiteln die genaue Umsetzung im Programmcode erläutert werden. Hierzu wird zunächst eine grobe Struktur des Programmaufbaus in Abbildung **3.1** gegeben. Wie in der Abbildung zu erkennen, ist es möglich

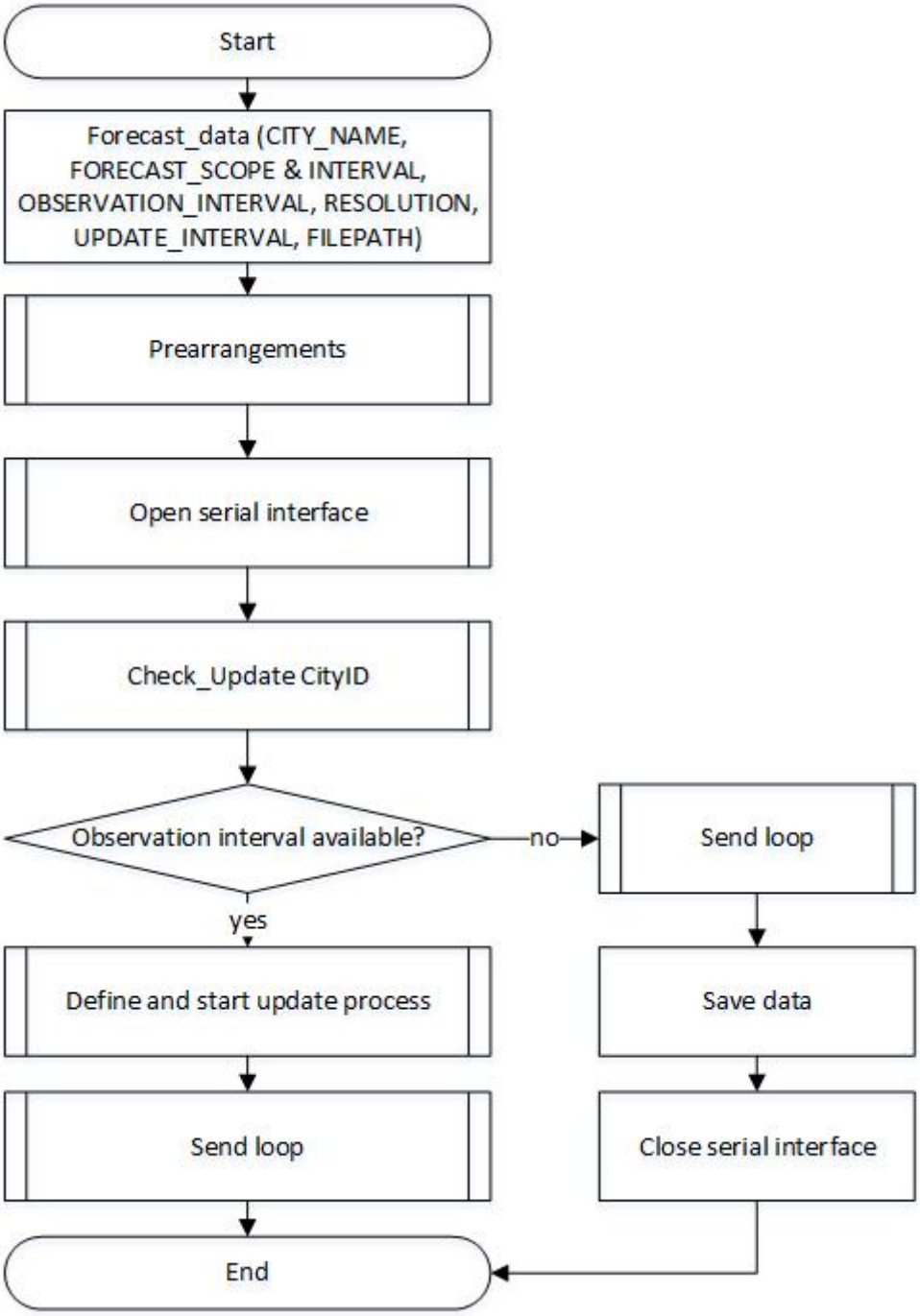



Abbildung 3.1: Ablaufplan der Funktion forecast_data

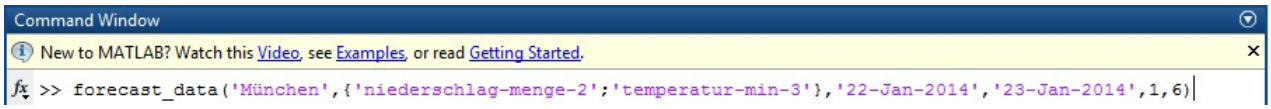



Abbildung 3.2: Beispiel für den Funktionsaufruf

für die Funktion forecast_data sieben Eingabeparameter zu definieren.

1. Wetterregion
2. Prognosebereichsdefinition
3. Start des Beobachtungszeitraums
4. Ende des Beobachtungszeitraums
5. zeitliche Auflösung
6. Updateintervall
7. Pfadangabe des Speicherortes (optional)

Die Wetterregion wird als Stadtname in Stringformat übergeben. Die Prognosebereichsdefinition besteht aus drei Teilen, die als String in dieser Format 'Prognosebereich-Prognosedetail-Prognoseintervall' aufgebaut ist. Der erste Teil gibt den Wetterbereich an. Hier stehen alle Einträge in der Spalte Prognosebereich der Tabelle A.1 im Anhang zur Verfügung. Die Prognosedetaildaten können der zweiten Spalte dieser Tabelle entnommen werden. Der dritte Parameter gibt die Anzahl der auszuwertenden Tage an, dabei steht der Wert 1 für den aktuellen Tag ohne Prognose und die Werte 2, 3, 4 für eine entsprechende Erweiterung des aktuellen Tages um die Prognosetage 1, 2 und 3 sofern vorhanden. Möchte man keine gesonderte Auswahl treffen und einfach alle möglichen Daten abrufen, so kann hier an dieser Stelle der Input 'all' erfolgen. Der Beobachtungszeitraum wird durch zwei Daten begrenzt, die ebenfalls als String in dieser Formation 'dd-mmm-yyyy', wobei mit mmm der englische Monatsname gemeint ist, angegeben werden. Die zeitliche Auflösung wird als Double eingetragen. Hier sind die Werte 1, 0.5, 0.25, 0.08 stellvertretend stehend für 1 Stunde, eine halbe Stunde, eine viertel Stunde und 5 Min., möglich. Ebenso als Double wir das Updateintervall definiert. Hier stehen die Werte 6, 12, 24 zur Verfügung, welches einem vier-, zwei- und einmaligen Update am Tag entspricht. Die Pfadangabe wird wiederum als String eingegeben. Die nächste Abbildung gibt hierzu ein Beispiel. Es wird für die Region München die Niederschlagsmenge für den heutigen und ersten Folgetag, die minimale Temperatur für den heutigen und die beiden nachfolgenden Tage, über einen Zeitraum von zwei Tagen abgerufen. Dabei beträgt die zeitliche Auflösung 1 Stunde und das Intervall in denen Updates gestartet werden 6 Stunden. Nachdem hier kein Speicherpfad angegeben ist, wird ein Ordner mit der Bezeichnung „Aufzeichnungen“ im aktuellen MATLAB Ordner erstellt und als Speicherort ausgewählt.

## 3.2 Vorbereitende Maßnahmen

### 3.2.1 Zuweisung variabler Inputparameter und Variableninitiierung

```
if ~isempty(varargin)
    if size(varargin,2) < 4
        fprintf(2,['Bitte geben Sie das Start- und Enddatum des' ...
            'Boabachtungszeitraums\n sowie die Aufloesung und' ...
            'das Updateintervall an.\n'],char(10));
        error('Zu wenig Inputparameter!');
    else
        start_observation       = varargin{1};
        end_observation         = varargin{2};
        resolution              = varargin{3};
        update_interval         = varargin{4};
    end
    if size(varargin,2) > 4
        filepath            = varargin{5};
    else
        if size(pwd,2) < 4
            filepath        = [pwd,'Aufzeichnungen'];
        else
            filepath        = [pwd,'\Aufzeichnungen'];
        end
        [s,mess,messid]     = mkdir(filepath);
        if ~isempty(mess)
            fprintf('Ordner existiert bereits.\n');
        end
    end
else
    if size(pwd,2) < 4
        filepath            = [pwd,'Aufzeichnungen'];
    else
        filepath            = [pwd,'\Aufzeichnungen'];
    end
        [s,mess,messid]     = mkdir(filepath);
        if ~isempty(mess)
            fprintf('Ordner existiert bereits.\n');
        end
    resolution = 1;
end

% If only one forecast definition is requested, convert char input into
% cell array.
if ~iscell(fc_def)
    fc_def = {fc_def};
end

% Set daychange_flag and daychange_counter to 0 for the first execution
daychange_flag      = 0;
```

```
daychange_counter   = 0;
assignin('base','daychange_flag',daychange_flag);
assignin('base','daychange_counter',daychange_counter);

% Initalize or reset with new function call data container
weather_data    = [];
new_data        = [];

% Set device id
device_id       = '03';
```

Für den Fall, dass variable Inputparameter übergeben wurden, wird zuerst geprüft, ob die Anzahl der geforderten Werte vorhanden ist. Ist das nicht der Fall, so wird eine Meldung an den Nutzer ausgeben und die Funktion beendet. Stimmt die Anzahl, werden die Werte Funktionsvariablen zugeordent. Ist zudem noch eine Pfadangabe zu einem Speicherort der Funktion übergeben worden, so wird diese ebenfalls einer Funktionsvariablen zugewiesen. Wurde keine Angabe hierzu gemacht, so wird im aktuellen MATLAB Ordner ein Ordner „Aufzeichnungen“ als Speicherplatz definiert. Dabei wird nachgesehen, ob lediglich eine Laufwerksangabe vorliegt oder nicht. Existiert der Ordner bereits, erfolgt eine Meldung auf deutsch. Wurden keine variablen Parameter übergeben, so wird die zeitliche Auflösung auf 1 Stunde festgesetzt und ebenfalls ein Speicherort vorgegeben. Nachdem die *forecast_defintion* sowohl als String als auch als Cell-Array übergeben wird, im späteren Programmablauf aber nur ein Cell-Array erwartet wird, muss noch eine Konvertierung stattfinden.Mit dem ersten Funktionsaufruf, an dem noch kein Tageswechsel auftreten kann, werden die Variablen *daychange_flag* und *daychange_counter* auf Null gesetzt und dem Base-Workspace zugewiesen. Ebenso initialisiert werden die späteren Datencontainer *weather_data* und *new_data*. Wie in Tabelle 1.2 gezeigt, lautet die Slave-ID der Wetterstation „03“. Diese wird hier der *device_id* zugeordnet.

### 3.2.2 Aufbau von Strukturen

```
% Create a table with all available forecast definitions.
if strcmp(fc_def,'all') == 1
    fc_def = create_table();
end

% Create the structure with all register addresses
data = create_reg_data();

% Create the list with all available city ids
city_list = create_city_list;

% Assign both created structures to base workspace
assignin('base','register_data_hwk_kompakt',data);
assignin('base','city_list',city_list);
```

Wie bereits im vorigen Kapitel 3.1 angekündigt, müssen große Datensätze in Strukturen gepackt werden um später aus ihnen Daten zu gewinnen. Sollen alle Wetterdaten abgerufen werden, ist es erforderlich ein Cell-Array aufzubauen, welches alle Wetterdatenanfragen beinhaltet. Danach werden Strukturen angelegt, die die Registeradressen und Städtenamen abbilden. Die letzten beiden Strukturen müssen wieder im Base-Workspace verfügbar sein da andere Funktionen auf sie zurückgreifen werden.

### 3.2.3 Überprüfung der Eingabeparameter

```
% Determine the number of requests.
size_table_data         = size(fc_def,1);

% Input check, if all inputs are correct, create data container, else print
% error message.
for z = 1:size_table_data
    [correct_input, error_msg, city_id, longitude, latitude] = ...
        input_check(fc_def{z}, city_name, varargin);
    if true(correct_input)
        weather_data = create_data_struct(fc_def{z}, weather_data, 'weather_data');
        new_data = create_data_struct(fc_def{z}, new_data, 'new_data');
    else
        fprintf(2,'%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n', error_msg{1},...
        error_msg{2}, error_msg{3}, error_msg{4}, error_msg{5},...
        error_msg{6}, error_msg{7}, error_msg{8}, error_msg{9},...
        error_msg{10} , char(10))
        error('Die oben aufgefuehrten Eingabeparameter sind nicht korrekt.');
    end
end

% Assign data container to base workspace. In the base workspace now the
% city_list, the data container and the register
% address structure are available.
assignin('base','weather_data',weather_data);
assignin('base','new_data',new_data);
```

Diese for-Schleife bearbeitet alle übergebenen Wetterdatenanfragen, prüft sie auf Gültigkeit und erstellt zugleich die entsprechenden Datencontainer. Die Datencontainer werden im Base-Workspace eingetragen. Sind ein oder mehrere Eingabeparameter falsch, werden diese dem Nutzer mit einer Nachricht angezeigt und die Funktion beendet.

```
function[ val_inpt, err_msg, c_id, lng, lat ] = input_check( fc_def, city, varargin )
```

Die Funktion input_check liefert als Outputparameter einen Vektor der angibt, welche Parameter gültig sind. Zusätzlich werden die generierten Fehlermeldungen, die ID der Wetterregion, der Breitengrad und Längengrad ausgegeben.

```
varargin = varargin{:};

% #### Check city ####
% Check for the right spelling and availability of city name
% val_inpt(1) will be 1 for existence and c_id contains the numeric
% city id.
[val_inpt(1), c_id, lng, lat] = get_city_id(city);

% Create failure message for a non existent city name
if val_inpt(1) == 0
    err_msg{1} = ['Diese Stadt kann in der CityList nicht gefunden werden: ' ...
           city];
end
```

Im ersten Abschnitt des Input Checks wird die Funktion get_city_id aufgerufen, um die ID der Wetterregion, den Längen- und Breitengrad zu ermitteln. Ist die ID nicht in der Liste zu finden, wird die entsprechende Vektorposition *val_inpt* auf false gesetzt.

```
function [ city_id_correct, c_id, lng, lat ] = get_city_id( cityname )
%Gets the city id from the city list
%   Detailed explanation goes here
citylist = evalin('base','city_list');
citylist.CityName = nominal(citylist.CityName);
city_id_dataset = citylist(citylist.CityName == cityname,:);
if isempty(city_id_dataset)
    city_id_correct = 0;
    c_id = '';
    lng = '';
    lat = '';
else
    c_id = city_id_dataset{:,1};
    lng = city_id_dataset{:,3};
    lat = city_id_dataset{:,4};
    city_id_correct = 1;
end
end
```

In dieser Funktion muss zuerst die im Base-Workspace befindliche Variable *city_list* zugänglich gemacht werden. Danach wird die Spalte mit den Städtenamen mit nominal konvertiert, um im darauffolgenden Schritt eine einfache Suche der Position in der Liste zu starten, die dem Städtenamen entspricht. In der Variable *city_data_set* sind nun alle Werte dieser Liste, die der Position entsprechen, enthalten. Eine einfache Wenn-Dann-Bedingung weist die Daten den Outputparametern zu.

```
% Check for the right definition of forecast, required scheme
% 'forecast_scope-fc_def-interval'
check = strfind(fc_def,'-');
```

```
% Create failure message if less or more then two '-' are existent in the
% forecast definition string
if size(check,2) ~= 2
    val_inpt(2)    = 0;
    err_msg{2}        = ['Die Wetterdatenanfrage wurde nicht korrekt definiert: '...
                          char(fc_def)];

% If no error put val_inpt(2) to 1
else
    val_inpt(2)    = 1;
% Disjoint the forecast definition into seperate parts (1) forecast_scope
% (2) forecast_detail (3) forecast_interval
    fc_def     = regexp(fc_def,'-','split');
% Define the possible values for each part of the forecast definition
    forecast_scope      = {'niederschlag', 'wind', 'temperatur', 'solarleistung', ...
                           'markantes_wetter', 'signifikantes_wetter', 'luftdruck'};
    forecast_details    = {'x', 'richtung', 'staerke', 'min', 'max', ...
                           'mittlere_temp_prog' ...
                           'dauer', 'einstrahlung', 'boeen', 'bodenfrost', ...
                           'gefrierender_regen', 'menge', 'kaelte', 'hitze', ...
                           'bodennebel', 'wahrscheinlichkeit', 'niederschlag'};
    forecast_interval   = {'1','2','3','all'};

% Determine if the input values are member of those lists defined above. If
% this is the case val_inpt values will be 1.
    val_inpt(3)    = ismember(fc_def{1},forecast_scope);
    val_inpt(4)    = ismember(fc_def{2},forecast_details);
    val_inpt(5)    = ismember(fc_def{3},forecast_interval);
% If any input value doesn't exist in the list, create error message.
    if val_inpt(3) == 0
        err_msg{3}    = ['Bitte ueberpruefen Sie den Prognosebereich: ' fc_def{1}];
    end
    if val_inpt(4) == 0
        err_msg{4}    = ['Bitte ueberpruefen Sie das Prognosedetail: ' fc_def{2}];
    end
    if val_inpt(5) == 0
        err_msg{5}    = ['Bitte ueberpruefen Sie das Prognoseintervall: ' fc_def{3}];
    end

end
```

An dieser Stelle wird die Wetterdatenanfrage analysiert. Kommt in dem Ausdruck nicht zweimal ein Querstrich vor, ist die Eingabe schon fehlerhaft. Wenn doch, werden die drei einzelnen Bestandteile mit Listen abgeglichen und bei entsprechender Existenz keine Fehlermeldung ausgegeben.

```
if ~isempty(varargin)
    try
        a=datenum(varargin{1});
```

```
        val_inpt(6) = 1;
    catch
        err_msg{6} = ['Bitte ueberpruefen Sie das Startdatum des'...
                      'Beobachtungsintervalls: ' varargin{1}];
        val_inpt(6) = 0;
    end
    try
        a=datenum(varargin{2});
        val_inpt(7) = 1;
    catch
        err_msg{7} = ['Bitte ueberpruefen Sie das Enddatum des'... '
                      'Beobachtungsintervalls: ' varargin{2}];
        val_inpt(7) = 0;
    end

    if val_inpt(6) == 1 && val_inpt(7) == 1
% Calculate the difference of days between the observation start and end
% date.
        diff_days               = days365(varargin{1},varargin{2})*24;

% If the difference is negative, the end date comes previous to the start
% date which is not possible.
        if diff_days < 0
            val_inpt(8)     = 0;
            err_msg{8}        = (['Das Startdatum fuer den Beobachtungszeitraum'...
                                  'muss vor dem Enddatum liegen! Bitte'...
                                  'korrigieren Sie die Datumseingabe.']);
% Same procedure with the start date, which never comes previous to the
% current date.
        elseif days365(date,varargin{1}) < 0
            val_inpt(8)     = 0;
            err_msg{8}        = ('Das Startdatum liegt in der Vergangenheit!');
        else
            val_inpt(8)     = 1;
        end
```

Die Überprüfung des Observationszeitraums erfolgt dahingehen, dass der Ausdruck ein Datumsformat darstellen muss, den MATLAB mittels datenum konvertieren kann. Ist dies nicht möglich, liegt ein Fehler vor. Sind die Datumsformate korrekt, so kann es immer noch der Fall sein, dass das Startdatum in der Vergangenheit oder vor dem Enddatum liegt. Auch hier werden entsprechende Fehlermeldungen generiert.

```
% #### Check resolution and update intervall ####
resolution_values   = {'6','1','0.5','0.25','0.08'};
updateinterval_values = {'6', '12', '24'};

        val_inpt(9)     = ismember(num2str(varargin{3}),resolution_values);

    if val_inpt(9) == 0
        err_msg{9}        = (['Fuer die Aufloesung koennen nur folgende Werte'...
```

```
                                'eingegeben werden: 6, 1, 0.5, 0.25, 0.08!']);
    end

        val_inpt(10)    = ismember(num2str(varargin{4}),updateinterval_values);

    if val_inpt(10) == 0
        err_msg{10}         = (['Fuer das Updateintervall koennen nur folgende'...
                                'Werte eingegeben werden: 6, 12, 24!']);
    end

end
% If no error exists don't return a error msg or if there are less than 6
% error messages, make the last error message empty so it could be
% displayed in the fprintf command.
    if true(val_inpt)
        err_msg           = NaN;
    elseif size(err_msg,2) < 10
        err_msg{10}         = [];
    end
```

Im letzten Teil der Überprüfung werden die Werte der zeitlichen Auflösung und des Updateintervalls wieder mit Listen abgeglichen. Ist der Gültigkeitsvektor in der boolschen Überprüfung wahr, werden keine Fehlermeldungen ausgegeben. Andernfalls wenn weniger als 10 Fehlermeldungen angefallen sind, muss der letzte Vektoreintrag der Fehlermeldungen leer sein. Dies ist erforderlich, um den nachfolgenden Print-Befehl ausführen zu können.

### 3.2.4 Verfügbarkeitsprüfung der seriellen Schnittstelle

```
% Check whether a variable serial interface already exists in the base
% workspace. If true delete this variable and all other available serial
% interfaces to be sure to build up the neccessary serial interface.
if evalin('base', ('exist(''serial_interface'')')) == 1
    evalin('base', ('delete(''serial_interface'')'));
    evalin('base', 'clear serial_interface');
    evalin('base', 'delete(instrfind)');
end

% Check for available COM Ports, if COM6 is not available print message
av_com_ports    = instrhwinfo('serial');
com_port_av     = find(ismember(av_com_ports.AvailableSerialPorts,'COM6'));

if isempty(com_port_av)
    fprintf(2,'COM6 Port ist nicht verfuegbar!\n', char(10));
    return;
end
```

Existiert bereits eine Variable *serial_interface* im Base-Workspace, so wird diese gelöscht. Zudem werden alle anderen seriellen Schnittstellen gelöscht um zu vermeiden, dass bereits eine andere

serielle Schnittstelle den COM6 Port belegt hat. Danach wird die Verfügbarkeit des COM6 Ports festgestellt. Ist dies nicht der Fall, wird eine Fehlermeldung an den Nutzer ausgegeben und die Funktion beendet.

## 3.3 Aufbau der seriellen Schnittstelle

```
% Open serial interface
open_serial_port( 'COM6', 19200, 8, 'even', 1 );
```

Die Funktion open_serial_port enthält bereits alle in der Tabelle 1.2 auf Seite 13 festgelegten Schnittstellenparameter.

```
function [ ] = open_serial_port( com_address, baudrate, databits, parity, stopbit )
%This function establishes the serial interface for the modbus
%communication channel.
%   For the HWK Kompakt COM address has to be COM6, Baudrate = 19200,
%   Databits = 8, Parity = 'even' and Stopbit = 1

% Creates the serial interface
serial_interface = serial(com_address,'BaudRate',baudrate,'DataBits', ...
                          databits,'Parity',parity,'StopBits',stopbit);

% Export variable to base workspace
assignin('base','serial_interface',serial_interface);

% Open the serial interface
fopen(serial_interface);

fprintf(['Serielle Schnittstelle wurde eingerichtet unter der Variable\n'...
```

MATLAB bietet für den Aufbau einer seriellen Schnittstelle eine Funktion namens serial an. Diese wird hier zur Erstellung der Variable *serial_interface* angewandt. Die Variable wird dem Base-Workspace zugewiesen und die serielle Schnittstelle mit dem Befehl fopen geöffnet. Der Nutzer wird über den Schnittstellenaufbau informiert.

## 3.4 Abgleich der Wetterregion im Register

```
city_id_reg                 = read_com_set(device_id, {'city_id'});

% If no value is detected for the city id register, the required city id
% will be written to that register. If the existent register value doesn't
% match the required value it will be overwritten.
if isempty(city_id_reg)
    fprintf('Es befindet sich kein Wert in Register 112!\n');
```

```
    write_com_set( device_id, city_id, {'city_id'} );
    fprintf(['Neue CityID %u wurde in das Register geschrieben.\n'...
        'Es wird ein paar Stunden dauern, bis alle Register aktualisiert wurden.\n\n'],...
        city_id);
elseif city_id ~= city_id_reg
    prompt = ['Die vorhandene City ID entspricht nicht der in der Funktion'...
        'uebergebenen ID.\n Moechten Sie fortfahren? Y/N [Y]: '];
    str = input(prompt,'s');
    if isempty(str)
        str = 'Y';
    end
    if strcmp(str,'Y') == 1
        write_com_set( device_id, city_id, {'city_id'} );
        fprintf(['Neue CityID %u wurde in das Register geschrieben.\n Es wird'...
            'ein paar Stunden dauern, bis alle Register aktualisiert wurden.\n\n'],city_id);
    else
        fprintf(2,'Der Funktionsaufruf wurde abgebrochen.\n', char(10));
        return;
    end
end
```

Da der Wechsel einer Wetterregion unter Umständen bis zu drei Tagen dauern kann, bis alle anliegenden Werte Gültigkeit besitzen, wird in diesem Teil des Codes zuerst die bereits eingetragene Stadt-ID ausgelesen und mit der in dem Funktionsaufruf Angegebenen verglichen. Weichen die beiden Werte voneinander ab, so wird der Nutzer per Tastatureingabe aufgefordert dem Wetterregionenwechsel zuzustimmen. Er hat somit die Gelegenheit einen versehentlichen Wechsel abzubrechen. Anschließend wird der neue Wert im Register eingetragen.

## 3.5 Festlegung der Timerparameter und Starten des Timers

```
if ~isempty(varargin)

% Split the datestring into single elements
    start_observation       = regexp(start_observation,'-','split');
    end_observation         = regexp(end_observation,'-','split');

% Calculate day difference in hours between the observation start date and
% end date. Further determine start and end date of current day.

    diff_days               = days365(start_observation,end_observation)*24;
    end_of_day              = datevec(date)+[0 0 0 24 0 0];
    start_of_day            = datevec(now);

% When the observation start date equals current date, calculate the
% remaining hours from calling the function to the end of current day. The
% number of update cycles results from the sum of remaining hours from the
% current day and hours between the days after current day to end of
% observation divided by the update interval. You have to add 1 for the
```

```
% first request executed immediatly with this function call.
% If the observation start date is in the future, and observation start
% date and end date are equal, 24 hours are available. Update cycle number
% is the result of the division diffdays/update_interval. Start delay is
% calculated from the sum of remaining hours of current date and difference
% of days in hours till start of observation.

    if strcmp(start_observation,date) == 1
        diff_today              = etime(end_of_day,start_of_day)/3600;
        update_cycle_number     = floor((diff_today+diff_days)/update_interval)+1;
        assignin('base','update_cycle_number',update_cycle_number);
    else
        if datenum(start_observation) == datenum(end_observation)
            diff_days           = 24;
        end
        diff_today          = etime(end_of_day,start_of_day);
        diff_days2start     = days365(date,start_observation);
        start_delay         = uint32(diff_today+diff_days2start*86400);
        update_cycle_number = floor(diff_days/update_interval);
        assignin('base','update_cycle_number',update_cycle_number);
```

Um die Eigenschaft des wiederholten Datenabrufs zu implementieren wurde das Timer-Objekt von MATLAB implementiert. Es bietet den Vorteil im Hintergrund zu laufen ohne dabei MATLAB komplett zu blockieren. Für die Initialisierung dieses Objekts müssen zuerst ein paar Parameter ermittelt werden. Insbesondere betrifft dies das Timerintervall, die Anzahl der Timerausführungen und die Timerverzögerung. Das Timerintervall lässt sich einfach dadurch berechnen indem die Variable *update_interval* mit der Anzahl an Sekunden für eine Stunde multipliziert wird. Die Anzahl der Timerausführungen ergibt sich dann durch die zur Verfügung stehenden Stunden im Beobachtungszeitraum dividiert durch die Länge des Timerintervalls. Liegt der Beobachtungszeitraum in der Zukunft, so muss die Timerverzögerung genau die Länge vom Funktionsaufruf bis zum Startdatum aufweisen. Bei der Berechnung dieser Angaben sind die MATLAB Funktionen days365, datevec sowie etime äußerst nützlich.

```
% Requests start with a 3 sec delay. The function to be executed after
% the waiting period is send_loop, which triggers the communication
% between Matlab and the weather station. The stop function deletes the
% timer object after all tasks have been executed.
    t                           = timer;

    if strcmp(start_observation,date) == 1
        t.StartDelay            = 3;
    else
        t.StartDelay            = start_delay;
    end
    t.TimerFcn                  = {@send_loop, size_table_data, fc_def, device_id, filepath, city
    t.StopFcn                   = {@stop_timer, filepath, city_name, resolution};
```

Mit der Zuweisung „t = timer“ wird in der Variablen $t$ das Timer-Objekt erzeugt. Die Eigenschaften können dann ähnlich einer Struktur in MATLAB aufgerufen und bestimmt werden. Zu bestimmen sind die Timerverzögerung ($t.StartDelay$), das Timerintervall ($t.Period$), die Timerausführungen ($t.TasksToExecute$), sowie die Ausführungsmethode ($t.ExecutionMode$). Die Ausführungsmethode „fixedRate“ garantiert, dass in genau gleichen Timerintervallen die in der Timerfunktion ($t.TimerFcn$) definierte Funktion ausgeführt wird. Wird der Timer durch einen Fehler oder durch einen Abbruchbefehl unterbrochen, so legt man in der Timerstopfunktion fest, was geschehen soll.

```
function [  ] = stop_timer(mTimer,~, filepath, city_name, resolution)
%Deletes timer object, serial interface and saves weather_data container to specified folder
%   Detailed explanation goes here

fprintf(['Automatischer Abruf fuer den angegebenen Beobachtungszeitraum\n' ...
      'wurde beendet.\n']);

delete(mTimer)

% Define filename as city_name_weather_data_current_date_current_unix_time
% and save to specified filepath
filename = strcat(filepath,'\',city_name,'_',strrep(num2str(resolution)...
    ,'.','_'),'_weather_data_',date,'_',num2str(date2utc(datevec(now))),'.mat');
weather_data = evalin('base','weather_data');
save(filename,'weather_data','-mat');

close_serial_port();
evalin('base','clear update_cycle_number');
end
```

Kommt es zu einem Timerabbruch, so wird eine Meldung an den Nutzer ausgegeben, das Timer-Objekt gelöscht und der Datencontainer mit den fortlaufenden Werten abgespeichert. Außerdem wird die serielle Schnittstelle beendet. Der Dateiname für den Datencontainer setzt sich zusammen aus der Wetterregion, der angewandten zeitlichen Auflösung, dem aktuellen Datum und dem aktuellen Datum in Unix Zeitformat.

## 3.6 Abschicken der MODBUS-Anfragen

```
function[  ] = send_loop(obj, event, t, fc_def, dev_id, f_path, city, u_c_n, res,...
                         lng, lat )
```

Die Funktion, die das sequentielle Abschicken und Verarbeiten der einzelnen MODBUS Nachrichten übernimmt wird hier erläutert.

```
h = waitbar(0,'Please wait while receiving data...');
```

```
daychange_flag = evalin('base','daychange_flag');
daychange_counter = evalin('base','daychange_counter');
w_dat = evalin('base','weather_data');

% For loop to process every forecast definition(fc_def).
for r = 1:t

        fc_int           = regexp(fc_def{r},'-','split');
% For the first loop determine if the datacontainer weather_data(w_dat) has
% stored previous data by analyzing the last stored recording time stamp.
% If there is such data, compare timestamp with current date. If it is not
% equal, increase daychange_counter and set daychange_flag true. Make both
% variables available in base workspace.

        if r == 1
            if ~isempty(w_dat.(fc_int{1}).(fc_int{2}).unix_t_rec)
                t_rec = w_dat.(fc_int{1}).(fc_int{2}).unix_t_rec(...
                        size(w_dat.(fc_int{1}).(fc_int{2}).unix_t_rec,2));

                if days365(utc2date(t_rec),date) ~= 0
                    daychange_flag = 1;
                    daychange_counter = daychange_counter + 1;
                else
                    daychange_flag = 0;
                end
            assignin('base','daychange_flag',daychange_flag);
            assignin('base','daychange_counter',daychange_counter);
            end
        end
```

Nachdem die Eingabeparameter übergeben wurden, werden die im Base-Workspace vorhandenen Variablen *daychange_flag*, *daychange_counter* sowie der Datencontainer *w_dat* für die Langzeitarchivierung in dieser Funktion bereitgestellt. Eine for-Schleife durchläuft dann alle Wetterdatenanfragen die in *fc_def* definiert wurden. Beim ersten Durchlauf muss geprüft werden, ob bereits zu einem früheren Zeitpunkt Daten ausgelesen wurden damit ein Tageswechsel signalisiert werden kann. Hierzu wird der letzte Zeitstempel des letzten Datenabrufs, sofern vorhanden, mit dem aktuellen Datum verglichen. Ist das Ergebnis ungleich Null, liegt ein Tageswechsel vor und die Variablen werden entsprechend angepasst und im Base-Workspace aktualisiert.

```
        forecast_days               = fc_int{1,3};

            if strcmp(fc_int{1,2},'mittlere_temp_prog') == 1
                switch forecast_days
                    case '1'
                       start_reg    = {'heute' 'am0_00'};
                       end_reg      = {'heute' 'pm11_00'};
                    case '2'
                       start_reg    = {'heute' 'am0_00'};
```

```
                    end_reg      = {'erster_folgetag' 'pm11_00'};
                case '3'
                    start_reg    = {'heute' 'am0_00'};
                    end_reg      = {'zweiter_folgetag' 'pm11_00'};
                case 'all'
                     start_reg   = {'heute' 'am0_00'};
                     end_reg     = {'dritter_folgetag' 'pm11_00'};
            end
        elseif strcmp(fc_int{1,1},'solarleistung') == 1 || ...
               strcmp(fc_int{1,1},'luftdruck') == 1
            if str2double(forecast_days) > 1
                warning(['Fuer die Solarleistungs- und Luftdruckprognose' ...
                        ' werden nur Werte fuer den heutigen Tag und den' ...
                        ' ersten Folgetag bereitgestellt.'])
                forecast_days = 'all';
            end
            switch forecast_days
                case '1'
                    start_reg    = {'heute' 'morgen'};
                    end_reg      = {'heute' 'abend'};
                case 'all'
                    start_reg    = {'heute' 'morgen'};
                    end_reg      = {'erster_folgetag' 'abend'};
            end
        else
            switch forecast_days
                case '1'
                    start_reg    = {'heute' 'morgen'};
                    end_reg      = {'heute' 'abend'};
                case '2'
                    start_reg    = {'heute' 'morgen'};
                    end_reg      = {'erster_folgetag' 'abend'};
                case '3'
                    start_reg    = {'heute' 'morgen'};
                    end_reg      = {'zweiter_folgetag' 'abend'};
                case 'all'
                    start_reg    = {'heute' 'morgen'};
                    end_reg      = {'dritter_folgetag' 'abend'};
            end
        end

%         if size(fc_int,2) < 5
%             end_reg                 = start_reg;
%         else
%             end_reg                 = fc_int(1,5:6);
%         end

        fc_int           = {fc_int{1,1:2}, start_reg{:}, end_reg{:}};
```

Im nächsten Schritt wird das Intervall für die auszulesenden Wetterdaten zusammengestellt. Hierbei muss unterschieden werden zwischen den stündlichen Werten der mittleren Lufttempe-

ratur, dem begrenzten Prognosehorizont von Luftdruck und Solarleistung und den verbleibenden Daten. Gibt der Nutzer eine zu hohe Zahl für Luftdruck oder Solarleistung ein, so wird auf die möglichen Werte angepasst und eine Warnung ausgegeben.

```
% Determine register addresses and number of registers to be processed
        start_reg_address       = get_reg_address( fc_int{1}, fc_int{2}, start_reg );
        end_reg_address         = get_reg_address( fc_int{1}, fc_int{2}, end_reg );

        quantity_reg_addresses  = reg_num(start_reg_address, end_reg_address);
% Generate modbus message
        modbus_pdu              = gen_msg( dev_id, start_reg_address,...
                                           quantity_reg_addresses, 'rsr' );
% Read the connection quality
        con_qual                = read_com_set('03',{'quality'});
% Write message on interface and read and process response
        txdata                  = send_and_receive_data(modbus_pdu, fc_int,...
                                                        res, con_qual, lng, lat);

        waitbar(r/t,h)

end
```

In diesem Abschnitt wird die MODBUS Nachricht erstellt und einer Funktion übergeben, die das Schreiben und Lesen auf der seriellen Schnittstelle übernimmt. Die Startadresse des Registers ergibt sich aus dem Prognosebereich, dem Prognosedetail und dem Anfangszeitpunkt zu dem die Werte ausgelesen werden sollen. Die Endadresse wir im gleichen Verfahren ermittelt. Die Differenz aus den beiden Adressdaten ergibt die Registeranzahl. Gemäß der MODBUS Spezifikation liegen nun alle Daten vor, die zur Kommunikation notwendig sind. Lediglich der Cyclic Redundancy Check fehlt noch. Dieser wird in der Funktion gen_msg erstellt und der PDU zusammen mit der Slave-ID angeheftet. Wie die momentane Verbindungsqualität zum Zeitpunkt der Anfrage ist, wird kurz darauf abgerufen. Alle für die weitere Verarbeitung erforderlichen Daten werden nun an die Funktion send_and_receive_data übergeben.

```
% Save requested data in a seperate file
new_data = evalin('base','new_data');
filename = strcat(f_path,'\',city,'-',strrep(num2str(res),'.','_'),'_new_data_',...
                  date,'_',num2str(date2utc(datevec(now))),'.mat');
save(filename,'new_data','-mat');
% Reset new_data container
new_data = [];
for z = 1:t
    new_data = create_data_struct(fc_def{z}, new_data, 'new_data');
end
assignin('base','new_data',new_data);
% Update the remaining number of requests
if ~isempty(u_c_n)
    u_c_n = evalin('base','u_c_n');
```

```
    u_c_n = u_c_n-1;
    fprintf('Noch %u ausstehende Abfrage(n).\n',u_c_n)
    assignin('base','u_c_n',u_c_n);
end
close(h);
end
```

Ist die for-Schleife durchlaufen und alle Daten liegen in den Datencontainern, wird der vollzogene Abruf in dem Container *new_data* mit entsprechendem Dateinamen abgespeichert. Der Dateiname setzt sich dabei aus dem Pfad zum Speicherort, dem Stadtnamen, der gewünschten Auflösung, dem aktuellen Datum und einem Datum in Unix Zeitformat zusammen. Ist das Speichern abgeschlossen, wird der Datencontainer geleert und neu initialisiert. Der letzte Schritt in diesem Programmteil ist das Update der noch verbleibenden Anzahl an Datenabrufen, welche in der Variable *u_c_n* (update cycle number) hinterlegt sind.

## 3.7 Senden und Empfangen

```
function [ value ] = send_and_receive_data( modbus_msg, field_name, res,...
                                            con_qual, lng, lat )
%Writes and reads modbus message on serial interface
%   Detailed explanation goes here

% Makes serial interfaces available in local workspace
serial_interface = evalin('base','serial_interface');

% Formats the modbus message into serial interface readable structure
txdata = format_modbus_msg(modbus_msg);

% Write message
fwrite(serial_interface,txdata);

pause(1);

% Check how many bytes have been received on serial interface
if serial_interface.BytesAvailable == 0
    bytes_num = 8;
else
    bytes_num = serial_interface.BytesAvailable;
end

% Read message data as unsigned values
[rxdata] = fread(serial_interface, bytes_num, 'uint8');

% Call rxdata processing
[value, error_msg] = rxdata_processing( rxdata, modbus_msg, field_name, res,...
                                        con_qual, lng, lat );

end
```

In dem Funktionsworkspace der Funktion send_and_receive_data muss die serielle Schnittstelle zugänglich sein. Um die in der Funktion send_loop generierte Nachricht über die Schnittstelle abschicken zu können, muss sie auf ein geeignetes Format gebracht werden. Dann erst kann die Nachricht mit dem Befehl *fwrite* übermittelt werden. Es wird 1 Sekunde auf die Antwort gewartet. Das bestimmen der empfangenen Bytes und die Angabe beim Lesebefehl fread beschleunigt die Kommunikation enorm. Jetzt liegt die Antwortnachricht des Servers in der Variablen *rx_data* vor. Die darin enthaltenen Werte wurden als unsigned integer 8 bit empfangen.

```
function [ txdata ] = format_modbus_msg( modbus_msg_crc )
%Format modbus pdu to 'fwrite' readable structure
%   Detailed explanation goes here
for e = 2:2:(size(modbus_msg_crc,2)-2)
    if e == 2
        temp1 = modbus_msg_crc(1,1:e);
        temp2 = modbus_msg_crc(1,e+1:end);
        string = strcat(temp1,':',temp2);
    else
        temp2 = modbus_msg_crc(1,e+1:end);
        string = strcat(string(1,1:(size(string,2)-size(temp2,2))),':',temp2);
    end
end
txdata = regexp(string,':','split');
txdata = hex2dec(txdata);
end
```

Die Funktion format_modbus_msg transformiert den String der MODBUS-Nachricht in einen 8-zeiligen Vektor mit dezimalen Werten die den hexadezimalen Werten des Strings entsprechen.

## 3.8 Rx-Datenverarbeitung

```
function [ response_data, crc_check_value, response_msg ] = ...
          rxdata_processing( rxdata, modbus_msg, field_name, res, con_qual, lng, lat )
%Processing the received rxdata from serial interface
%   Rxdata contains the response from the MODBUS server, which has to be
%   processed here and in a subsequent function.

not_done = 1;
```

Hauptaufgabe der Rx-Datenverarbeitung ist es den empfangenen Funktionscode auszuwerten, die Daten zu isolieren und an eine weitere Funktion zur Bearbeitung zu übergeben. Der rx-Datenstring ist als Zeilenvektor aus einer Reihe von Dezimalzahlen folgendermaßen aufgebaut:

- an erster Stelle kommt die Slave-ID
- an zweiter Stelle folgt der Funktionscode

- an dritter Stelle steht die Anzahl der übertragenen Bytes
- an den darauffolgenden Positionen befinden sich die Datenbytes
- die beiden letzten Einträge beinhalten den CRC-Wert

```
while not_done == 1
    if isempty(rxdata)
        error('No data received! Check if server is available!');
%         response_data = [];
%         crc_check_value = 0;
%         response_msg = [];
    else
        func_code = dec2hex(rxdata(2),2);
        fcode_error = fcode_check(func_code);

        if fcode_error == 1
            exception_code = dec2hex(rxdata(3),2);
            switch exception_code
                case '01'
                    error('Exception Code 01 -> Function code not supported');
                case '02'
                    error('Exception Code 02 -> Output address not valid');
                case '03'
                    error(['Exception Code 03 -> Quantity of outputs exceeds'...
                           'range 0x0001 and 0x07D0']);
                case '04'
                    error('Exception Code 04 -> Failure during reading discret'...
                          'outputs');
            end
%             response_data = [];
%             crc_check_value = 0;
%             response_msg = [];
        else
```

Zuerst wird der Funktionscode auf einen Fehlercode hin in der Funktion fcod_check kontrolliert. Dies ist dann der Fall, wenn er nicht den Wert 1, 3 oder 6 besitzt. Um welchen Ausnahmefall es sich dann handelt, wird in der Switch-Abfrage geklärt und das Programm stoppt mit einem Fehler.

```
function [ fcode_error ] = fcode_check( fcode )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
switch fcode
    case dec2hex(1,2)
    fcode_error = 0;
    case dec2hex(3,2)
    fcode_error = 0;
    case dec2hex(6,2)
```

```
    fcode_error = 0;
    otherwise
    fcode_error = 1;
end

end
```

Ist der Funktionscode in Ordnung wird entschieden, wie mit den Datenbytes zu verfahren ist. Eine Switch-Abfrage führt zu den entsprechenden Arbeitsschritten. Für den Fall, dass der Funktionscode den Wert 1 oder 6 aufweist, ist die Bearbeitung recht einfach. Es müssen lediglich ein Datenbyte beim Code 1 und zwei Datenbytes beim Code 6 verarbeitet werden.

```
            switch rxdata(2)
                case 1
%                     byte_count = rxdata(3);
                    response_data = rxdata(4);
                    [crc_check_value, response_msg] = crc_check(rxdata);
                case 3
%                     byte_count = rxdata(3);
%                     if byte_count > (size(rxdata,1)-5)
%                     [ value ] = send_and_receive_data( modbus_msg, field_name );
%                     end
                    response_data = data_processing( rxdata(4:end-2), field_name,...
                                                     res, con_qual, lng, lat );
                    [crc_check_value, response_msg] = crc_check(rxdata);
                case 6
                    response_data = dec2hex(rxdata(5:6),4);
                    [crc_check_value, response_msg] = crc_check(rxdata);
            end
        end
    end

    if (isempty(response_data) || isempty(crc_check_value) || isempty(response_msg))
        not_done = 1;
    else
        not_done = 0;
    end

end
end
```

## 3.9 Datenverarbeitung

```
function[ dec_value ] = data_processing(data_string, fc_def, res, con_qual, lng, lat)
```

Das eigentliche Herzstück dieses Programms ist die Datenverarbeitung die mit der Funktion data_processing gestartet wird. Die notwendigen Parameter sind die empfangen Datenbytes im

*data_string* hinterlegt, die Definition der Wetterdatenabfrage als *fc_def*, die zeitliche Auflösung in der Variable *res*, die Verbindungsqualität, übergeben als *con_qual* und die Längen- und Breitengradangabe in Form von *lng* bzw. *lat*.

```
% Get the new data and weather data container form the base workspace
w_dat = evalin('base','weather_data');
n_dat = evalin('base','new_data');
daychange_counter = evalin('base','daychange_counter');

% Decide if MEZ or MESZ is valid
MEZ = MESZ_calc();

% Container building: for the first request session (=modbus-message)
% set the row counter to 1 for both weather and new data container
% extent the weather data container to a 1,6 cell array and define the new
% data container as a 1,2 cell array. After the first request is finished
% both cell arrays will contain the response for the send message. For the
% following requests we have to receive the new data array from the base
% workspace and set the row counter to the next free row.


% Here lists are defined which will be needed to define the loop numbers or
% to find the right register address
obs_day         = {'heute' 'erster_folgetag' 'zweiter_folgetag' 'dritter_folgetag'};
day_segment     = {'morgen' 'vormittag' 'nachmittag' 'abend'};
point_in_time   = {'am0_00' 'am01_00' 'am02_00' 'am03_00' 'am04_00' ...
                   'am05_00' 'am06_00' 'am07_00' 'am08_00' 'am09_00' ...
                   'am10_00' 'am11_00' 'am12_00' 'pm01_00' 'pm02_00' ...
                   'pm03_00' 'pm04_00' 'pm05_00' 'pm06_00' 'pm07_00' ...
                   'pm08_00' 'pm09_00' 'pm10_00' 'pm11_00'};
com_settings    = {'temperature_offset','temperature','city_id', ...
                   'transmitting_station','quality','fsk_qualitaet' ...
                   'status_ext_temp_sensor' 'reserve1' 'reserve2' 'reserve3'};
```

Zunächst werden die Variablen aus dem Base-Workspace geladen, mit denen gearbeitet wird. Danach liefert die Funktion MESZ_calc ein Flag für die Mitteleuropäische Sommerzeit. Im Anschluß werden die Listen für einen späteren Datenabgleich erstellt.

```
% If no weather data are requested, but communication specific values the
% if condition is true. Otherwise weather data will be processed.
if strcmp('register_data_hwk_kompakt.communication_settings',fc_def) == 1
    dec_value = hex2dec(strcat(dec2hex(data_string(1),2),dec2hex(data_string(2),2)));

    % As we have an unsigned value from the message, we have to convert
    % it to a signed value, which means FFFF or 65535 stands for -1
    if dec_value > 32768
        dec_value = dec_value - 65536;
    end
else
```

Erfolgt nur eine Abfrage der Kommunikationsparameter oder bestimmter Zustände der Wetterstation, muss nur ein Wert ausgewertet werden. Dies geschieht, wenn die If-Bedingung wahr ist. Da die Daten der seriellen Schnittstelle als unsigned int8 Werte empfangen wurden, werden negative Werte ab der Zahl 32769 dargestellt. Um sie zu erhalten muss der Wertebereich von 2 unsigned Bytes abgezogen werden.

```
    t_rec       = [];
    i           = 1;
    n_dat_r     = 1;

% Decide which factor to choose for interpolation
    factor = res_factor(res,fc_def{2});

% When the first function call was executed a record timestamp will be
% stored in the data container. The last record date of the last update
% will be assigned to t_rec to compare it with the current update date to
% evaluate if a daychange has been occured.
    if ~isempty(w_dat.(fc_def{1}).(fc_def{2}).unix_t_rec)
        t_rec = w_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(...
                        size(w_dat.(fc_def{1}).(fc_def{2}).int_val,2));
    end

% sdindex = starting point of the loop through the observation days
% edindex = end point
    [~, sdindex] = ismember(fc_def{3}, obs_day);

% If condition is false, number of loops will be determined by the list
% position (obs_day) of the last day of observation.
        [~, edindex] = ismember(fc_def{5}, obs_day);
    end

% When the function is called the first time t_rec will be empty and the
% start position for the interpolated and original data datavector will
% be 1.
    if isempty(t_rec)
        w_dat_r = 1;
        w_dat_r_org = 1;
    else
```

Ist die If-Bedingung falsch, handelt es sich um Wetterdaten. Zunächst werden Variablen initiiert. Dabei ist $i$ eine Laufvariable, die die Position angibt zu dem ein Datumswechsel stattgefunden hat. $n\_dat\_r$ ist die Vektorposition in dem Datencontainer, der jeden einzelnen Datenabruf speichert. Deshalb ist sie immer auf 1 gesetzt. Da die Ausgangsdaten selbst in einer unterschiedlichen Auflösung gegeben sind, müssen mit der Funktion res_factor auch zwei separate Faktoren für die spätere Interpolation bestimmt werden. Hat ein Abruf bereit zu einem früheren Zeitpunkt stattgefunden, so wird dieser Abruf-Zeitstempel der Variable $t\_rec$ zugewiesen. Er bestimmt sich aus

dem letzten aufgezeichneten Wert und dessen Rekordstempel. Anschließend wird die Startvariable *sdindex* der nachfolgenden for-Schleife für die zu bearbeitenden Prognosetage bestimmt. Dabei definiert die Position in der entsprechenden Liste diesen Wert. Der Endwert *edindex* der Schleife wird in gleicher Weise bestimmt.

```
% If t_rec is not empty a previous function call had been executed. If this
% execution was on the same day data vector position has to stay constant.
% In the case a daychange occures daychange will be increased by 1.
        if days365(datestr(utc2date(...
                    w_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(1)),1),date) == 0
            w_dat_r_org = 1;
            w_dat_r = 1;
        else
% Get the data vector position for which the date changes, start at positon
% 1 from recording.
            while strcmp(datestr(utc2date(...
                        w_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(i)),1),date) ~= 1
                i = i + 1;
                if i > size(w_dat.(fc_def{1}).(fc_def{2}).unix_t_strt,2)
                    break;
                end
            end
% i will be the next position in the data vector to write data to
            w_dat_r = i;
% For the data with no interpolation you don't have to change position
% cause they go parallel.
            if strcmp(fc_def{1},'markantes_wetter') == 1 || ...
               strcmp(fc_def{1},'signifikantes_wetter') == 1 || ...
               strcmp(fc_def{2},'richtung') == 1 || ...
               strcmp(fc_def{2},'wahrscheinlichkeit') == 1
                w_dat_r_org = w_dat_r;
            else
% For interpolated data you have to make a difference between original data
% and interpolated data vector position.
                if strcmp(fc_def{2},'mittlere_temp_prog') == 1
                    w_dat_r_org = 24*daychange_counter+1;
                else
                    w_dat_r_org = 4*daychange_counter+1;
                end
            end
        end
    end
```

Lag noch kein vorheriger Datenabruf vor, so starten die Positionsvektoren des Datencontainers, der die Daten des Beobachtungszeitraums speichert, bei 1. Hier werden zwei Variablen *w_dat_r* und *w_dat_org* benötigt. Während die Erste später mit der interpolierten zeitlichen Auflösung voranschreitet, behält die Zweite die ursprüngliche Schrittweite. Die while-Schleife bestimmt nun die Position bei einem evtl. Datumswechsel. Dazu wird die in der Variable *unix_t_strt* gespeicherte Zeitreihe durchlaufen bis das Datum dem aktuellen Datum entspricht. Der Wert wird

der Vektorposition übergeben. Abhängig ob es sich um Daten handelt die interpoliert werden, bekommt auch die Vektorposition für die originalen Daten einen Wert zugewiesen. Er errechnet sich entsprechend der Auflösung der Daten für die mittlere Lufttemperatur (24 Werte pro Tag) multipliziert mit der Zahl der bis dato vollzogenen Tageswechseln plus eins.

```
% Increment initialization for the data loop
    data_str_hi_byte_pos = 1;
    data_str_lo_byte_pos = 2;

% Loop through response data
    for t = sdindex:edindex

% With the first day of observation, determine the starting point
% of the observation day segment or point in time
% (Mittlere_temp_prog)
        if t == sdindex

% Determine starting point for the first observation day
                if strcmp(fc_def{2},'mittlere_temp_prog') == 1
                    [~, shindex] = ismember(fc_def{4}, point_in_time);
                else
                    [~, shindex] = ismember(fc_def{4}, day_segment);
                end

% End point for the first observation day will either be
% the starting point, when only one value is requested, or
% the entire intervall (24,4), which will be stopped at when
% the data string is completely evaluated.
                if size(fc_def,2) < 5
                    ehindex      = shindex;
                elseif strcmp(fc_def{2},'mittlere_temp_prog') == 1
                    ehindex      = 24;
                else
                    ehindex      = 4;
                end
        elseif t == edindex

% If more then one day is observed we have in any case at
% least a starting index of 1. The ending point is
% determined through the list position in point_in_time or
% day_segment.
                shindex          = 1;

                if strcmp(fc_def{2},'mittlere_temp_prog') == 1
                    [~, ehindex] = ismember(fc_def{6}, point_in_time);
                else
                    [~, ehindex] = ismember(fc_def{6}, day_segment);
                end
        else
```

```
% If the observation intervall exceeds more than two days,
% the starting point and end point are defined over the
% complete forecast intervall for the days between start
% and end day.
                shindex         = 1;
                if strcmp(fc_def{2},'mittlere_temp_prog') == 1
                    ehindex     = 24;
                else
                    ehindex     = 4;
                end
        end

        if t == sdindex
                datepart        = str2double(regexp(...
                                  datestr(date,'yyyy-mm-dd'),'-','split'));
                date_str_num    = datenum(date);
        else
                datepart        = datepart + [0 0 1];
                date_str_num    = date_str_num + 1;
        end
```

Handelt es sich um einen längeren Datenstring, so muss dieser durchlaufen werden. Es erfolgt eine Initialisierung dieser Variablen. Danach beginnt die Schleife für den Tagesdurchlauf. Zu Beginn wird je nach Schleifenpostition der Anfangs- und Endwert der Schleife für die Tagessegmente bestimmt. Der hier verwendete Aufbau scheint ein wenig kompliziert zu sein und stammt aus einer Version in der es möglich war völlig willkürliche Intervalle zu bestimmen und nicht nur tagesweise den Abruf zu durchlaufen. Sind Start- und Endwerte für die Tagessegment-Schleife bestimmt, wird im letzten Schritt das Datum der Schleifenposition ermittelt und in der Variablen *date_part* gespeichert. Diese wird später für die Berechnung der Unix Zeitstempel benötigt.

```
        for s = shindex:ehindex

% Break condition for an completely evaluated data string
            if data_str_lo_byte_pos > size(data_string,1)
                break;
            end

% Evaluation of a 16-bit word, big-Endian
            hi_byte         = dec2hex(data_string(data_str_hi_byte_pos),2);
            lo_byte         = dec2hex(data_string(data_str_lo_byte_pos),2);
            hex_value       = strcat(hi_byte,lo_byte);
            dec_value       = hex2dec(hex_value);

% Receiving uint bytes, signed bytes will be calculated here
            if dec_value > 32768
                dec_value   = dec_value - 65536;
            end

% Set the unix timestamp to the original interval the data are valid for.
```

```
            if s > 4
                timevec = tvector(fc_def{2}, datepart, point_in_time{s});
            else
                timevec = tvector(fc_def{2}, datepart, point_in_time{s},...
                                  day_segment{s});
            end

% If any value received is equal to 10000 the data processing for this
% forecast scope will be canceled.
            if dec_value == 10000
                warning_msg = ['Der Wert fuer den Prognosebereich ',...
                    fc_def{1},'-',fc_def{2},'-',fc_def{3},'-',fc_def{4},...
                    ' ist ungueltig! Der komplette Prognosebereich wurde'...
                    'deshalb nicht gespeichert!'];
                warning(warning_msg);
                return;
            end

% Determine the offset values for interval timestamps in the case a
% different res then the original res was selected.
            if strcmp(fc_def{2},'mittlere_temp_prog') == 1 && factor ~= 1
                timestep = 3600-1;
                timestep_corr = (factor-1)*(3600/factor);
                timestep_int = 3600/factor;
            else
                timestep = 6*3600-1;
                timestep_corr = (factor-1)*(21600/factor);
                timestep_int = 21600/factor;
            end
```

Es beginnt nun der Schleifendurchlauf für die Tagessegmente. Diesen sind die entsprechenden Wetterdaten zugeordnet, die aus einem High- und Low-Byte zusammengesetzt sind und aus dem Hexadezimal- in das Dezimalformat konvertiert werden müssen. Für die Unix-Zeitstempelberechnung wird ein Datumsvektor benötigt, der in der Funktion *tvector* für die Wetterdaten erstellt wird. Liegt ein Wert von 10000 an, wird eine Warnmeldung an den Nutzer ausgegeben und der komplette Prognosebereich nicht weiter bearbeitet. Die Zeitschritte, die die Zeitreihe bestimmen werden in der nachfolgenden If-Bedingung bestimmt. Dabei ist *timestep* der normale Zeitschritt ohne Interpolation. Für die mittlere Lufttemperatur beträgt sie eine Stunde für die anderen Bereiche 6 Stunden. Der Subtrahend entstammt der Intervalleingrenzung, die auf 00:00:00 Uhr bis 00:59:59 festgelegt wurde. Bei einer Interpolation müssen die Zeitschritte angepasst werden. Bei einer Auflösung von 5 Min. muss z.B. das einstündige Intervall um 55 Min. gekürzt werden.

```
% Assign received value to continous data container. No interpolation is done.
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(w_dat_r)    = ...
                    date2utc(timevec);
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(w_dat_r)     = ...
                    date2utc(timevec) + timestep;
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(w_dat_r)    = ...
```

```
                    date2utc(timevec) + floor(timestep/2);
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(w_dat_r)     = ...
                    date2utc(datevec(now),MESZ_calc);

                w_dat.(fc_def{1}).(fc_def{2}).interval_t_clr{w_dat_r} = ...
                    {[cell2mat(utc2date(date2utc(timevec))),'-',...
                    datestr(utc2date(...
                    w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(w_dat_r)),13)]};

                w_dat.(fc_def{1}).(fc_def{2}).int_val(w_dat_r)        = ...
                    data_mult(dec_value,fc_def{2});
                w_dat.(fc_def{1}).(fc_def{2}).org_val(w_dat_r_org)    = ...
                    data_mult(dec_value,fc_def{2});
                w_dat.(fc_def{1}).(fc_def{2}).con_qual(w_dat_r_org)   = ...
                    con_qual;

                fprintf('%s %s - %u %u %u %s %u \n', fc_def{1}, fc_def{2},...
                    date2utc(timevec), date2utc(timevec) + timestep,...
                    date2utc(datevec(now),MESZ_calc),...
                    cell2mat(strcat(utc2date(date2utc(timevec)),...
                    '-',datestr(utc2date(date2utc(timevec) + timestep),13))),...
                    data_mult(dec_value,fc_def{2}));

% Assign received value to update data container. No interpolation is done.
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(n_dat_r)    = ...
                    date2utc(timevec);
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(n_dat_r)     = ...
                    date2utc(timevec) + timestep;
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(n_dat_r)     = ...
                    date2utc(datevec(now),MESZ_calc);
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(n_dat_r)    = ...
                    date2utc(timevec) + floor(timestep/2);

                n_dat.(fc_def{1}).(fc_def{2}).interval_t_clr{n_dat_r} = ...
                    {[cell2mat(utc2date(date2utc(timevec))),'-',...
                    datestr(utc2date(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(n_dat_r)),13)]};

                n_dat.(fc_def{1}).(fc_def{2}).int_val(n_dat_r)        = ...
                    data_mult(dec_value,fc_def{2});
                n_dat.(fc_def{1}).(fc_def{2}).org_val(n_dat_r)        = ...
                    data_mult(dec_value,fc_def{2});
                n_dat.(fc_def{1}).(fc_def{2}).con_qual(n_dat_r)       = ...
                    con_qual;

            % Incrementing data string, and data container row position
            data_str_hi_byte_pos = data_str_hi_byte_pos + 2;
            data_str_lo_byte_pos = data_str_lo_byte_pos + 2;

            w_dat_r     = w_dat_r + 1;
            n_dat_r     = n_dat_r + 1;
```

```
            w_dat_r_org = w_dat_r_org + 1;

        end
    end
```

In diesem Abschnitt werden die Werte dem Datencontainer zugewiesen. Dabei werden folgende Variablen belegt:

- *unix_t_strt* enthält den Startzeitpunkt in Unix Zeitformat des Intervalls für den der interpolierte Wert später vorliegt
- *unix_t_end* enthält den Endzeitpunkt in Unix Zeitformat des Intervalls für den der interpolierte Wert später vorliegt
- *unix_t_mean* enthält den Mittelwert in Unix Zeitformat des Intervalls für den der interpolierte Wert später vorliegt
- *interval_t_clr* enthält Start- und Endzeitpunkt des Intervalls in normalem Zeitformat
- *int_val* enthält den interpolierten Wetterdatenwert
- *org_val* enthält den nicht interpolierten ausgelesenen Wetterdatenwert
- *con_qual* enthält die Verbindungsqualitätsdaten

Die Transformation in das Unixzeitformat übernimmt die Funktion date2utc. Außerdem müssen die Werte der Niederschlagsmenge und der Sonnenscheindauer mit einem eigenen Faktor beaufschlagt werden um die Einheiten $l/m^2$ und h zu bekommen. Dies erledigt die Funktion data_mult. Am Schluss eines Schleifendurchlaufs müssen die Vektorpositionen des Datenstrings und der Zeitreihen angepasst bzw. inkrementiert werden.

```
    if res == 6
% Assign data container to base workspace
        assignin('base','new_data',n_dat);
        assignin('base','weather_data',w_dat);
    else
```

Für den Fall, dass eine Auflösung von 6 Stunden gewünscht ist, wird nicht interpoliert und die Werte können so im Datencontainer in den Base-Workspace abgelegt werden.

```
% For those forecast scopes with no interpolation the data can be assigned
% to the base workspace.
        if strcmp(fc_def{1},'markantes_wetter') == 1 || ...
           strcmp(fc_def{1},'signifikantes_wetter') == 1 || ...
           strcmp(fc_def{2},'richtung') == 1 || ...
           strcmp(fc_def{2},'wahrscheinlichkeit') == 1

            assignin('base','new_data',n_dat);
            assignin('base','weather_data',w_dat);
```

Wird eine andere Auflösung als die originale gefordert, so können die Werte, die nicht interpoliert werden gleich im Base-Workspace gespeichert werden.

```
% Select x and y values for interpolation from new data
            tmp_dat_y = double(n_dat.(fc_def{1}).(fc_def{2}).int_val(1,1:end));

            if strcmp(fc_def{1},'solarleistung') == 1
                [sun_rise_today,sun_set_today] = diurnal_var(lat, lng, date);
                [sun_rise_tomorrow,sun_set_tomorrow] = ...
                    diurnal_var(lat, lng, datestr(datenum(date)+1));
                sun_rise_today = double(date2utc(sun_rise_today, MEZ));
                sun_set_today = double(date2utc(sun_set_today,MEZ));
                sun_rise_tomorrow = double(date2utc(sun_rise_tomorrow,MEZ));
                sun_set_tomorrow = double(date2utc(sun_set_tomorrow,MEZ));
                tmp_dat_x = [linspace(sun_rise_today,sun_set_today,4),...
                             linspace(sun_rise_tomorrow,sun_set_tomorrow,4)];
            else
                tmp_dat_x = ...
                        double(n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,1:end));
            end
```

Für die die Werte der Solarleistung müssen der Sonnenauf- und Sonnenuntergangszeitpunkt bestimmt werden. Tut man dies nicht, so wird später eine Sonnenscheindauer bzw. Globalstrahlung von 0.00 Uhr bis 23.59 Uhr interpoliert. Die Funktion diurnal_var bestimmt diese Werte mit Hilfe der Längen- und Breitengradangabe der Wetterregion sowie dem entsprechenden Datum. Der Algorithmus zur Berechnung dieser Daten wurde aus dem Buch „Photovoltaik Engeneering“ entnommen [7]. Die Messwerte der Wetterstation werden dann auf vier gleiche Intervalle von Sonnenauf- bis Sonnenuntergang aufgeteilt.

```
% Define the end of datavector after interpolation. Take actual size of new
% data i.e. 8 values for 2 days and multiply it with factor will be the
% same as to divide 48h into 5m intervals. 6h have 72 5m intervals. If
% there has been already a interpolation end of data vector will start from
% the new date which was determined in i.
                if i == 1
                    data_end = size(n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean,2)*...
                                    factor;
                else
                    if strcmp(fc_def{2},'mittlere_temp_prog') == 1
                        data_end = i - 1 + edindex*24*factor;
                    else
                        data_end = i - 1 + edindex*4*factor;
                    end
                end

% Adjust start intervals to new res

% Take first interval of daychange 0-6:00 subtract correction to yield
```

```
% 0-0:05 for a res of 5 Min.. Do this for all intervals. To obtain
% the new interval end of continous data, take the first 6h interval of new
% data and subtract timestep_corr. E.g. res is 5m -> new end of
% continous data will be 6h-5h55m.
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(i)      = ...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(1) - timestep_corr;
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i)     = ...
                    floor((w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(i)-...
                           n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1))/2)+...
                           n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1);

                w_dat.(fc_def{1}).(fc_def{2}).interval_t_clr(i)  = ...
                    {[cell2mat(utc2date(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1))),'-',...
                    datestr(utc2date(...
                    w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(i)),13)]};

                n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(1)      = ...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(1) - timestep_corr;
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1)     = ...
                    floor((n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(1)-...
                           n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1))/2)+...
                           n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1);

                n_dat.(fc_def{1}).(fc_def{2}).interval_t_clr{1}  = ...
                    {[cell2mat(utc2date(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1))),'-',...
                    datestr(utc2date(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(1)),13)]};
```

Wenn noch kein Datenabruf vorher stattgefunden hat, wird das Ende des Datensatzes durch die Vektorlänge des mittleren Zeitreihenintervalls multipliziert mit dem Faktor festgelegt. Fanden dagegen schon zuvor Datenabfragen statt, dann berechnet sich das Ende durch Addition der neuen interpolierten Werte auf die Position des Datumwechsels. In den darauffolgenden Schritten werden die Intervallstartwerte neu berechnet, indem die alten Intervallgrenzen um die Korrekturfaktoren verschoben werden.

```
% Adjust all following intervals
% Start intervall will be starting from 0:00-6:00-timest
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(i:data_end)   = ...
                    w_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(i):timestep_int:...
                    (n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(end)+timestep_corr);
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(i:data_end)    = ...
                    w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(i):timestep_int:...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(end);
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i:data_end)   = ...
                    w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i):timestep_int:...
                    (n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(end)+...
                    (timestep_corr/2));
```

```
                w_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(i:data_end)     = ...
                    date2utc(datevec(now),MESZ_calc);

                date_string1 = ...
                    utc2date(w_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(i:data_end));
                date_string2 = ...
                    utc2date(w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(i:data_end));
                w_dat.(fc_def{1}).(fc_def{2}).interval_t_clr(i:data_end) = ...
                    cellstr(strcat(cell2mat(date_string1'),'-',...
                    datestr(cell2mat(date_string2'),13)))';

                n_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(1:size(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_rec,2)*factor)  = ...
                    date2utc(datevec(now),MESZ_calc);
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1:size(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt,2)*factor) = ...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(1):timestep_int:...
                    (n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(end)+timestep_corr);
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(1:size(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end,2)*factor)  = ...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(1):timestep_int:...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(end);
                n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1:size(...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean,2)*factor) = ...
                    n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1):timestep_int:...
                    (n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(end)+...
                    (timestep_corr/2));

                date_string1 = utc2date(n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(...
                    1:size(n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt,2)));
                date_string2 = utc2date(n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(...
                    1:size(n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt,2)));
                n_dat.(fc_def{1}).(fc_def{2}).interval_t_clr(1:...
                    size(n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt,2)) = ...
                    cellstr(strcat(cell2mat(date_string1'),'-',...
                    datestr(cell2mat(date_string2'),13)))';
```

Sind die Startpositionen bekannt, können jetzt die kompletten Zeitreihen z.B. mit 5 Min. Schritten erstellt werden.

```
% Perform the slm interpolation for temperatur, staerke and luftdruck
                if strcmp(fc_def{1},'temperatur') == 1 || ...
                   strcmp(fc_def{2},'staerke') == 1 || ...
                   strcmp(fc_def{1},'luftdruck') == 1 || ...
                   strcmp(fc_def{2},'menge') == 1

                    if i == 1
                        slm = slmengine(tmp_dat_x,tmp_dat_y,'plot','off',...
                            'knots',16,'increasing','off','leftslope',0,...
                            'rightslope',0);
```

```
                        w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i:data_end) = ...
                            slmeval(w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(...
                            1,i:data_end),slm);
                    else

% Calculate the slope of the end of previous data. The slope of the new
% calculated interpolation values has to be on the left side equal to that
% of the intersecting old data on the right side. Furthermore the values of
% old and new data have to be the same.
                        dy = w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1) -...
                            w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-2);
                        dx = w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,i-1) -...
                            w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,i-2);
                        m = dy/dx;
                        slm = ...
                            slmengine(...
                            [w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,i-1)...
                            tmp_dat_x],...
                            [w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1)...
                            tmp_dat_y],'plot','off','knots',16,'increasing','off',...
                            'leftvalue',...
                            w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1),...
                            'leftslope',m,'rightslope',0);

% Evaluate spline function at timestamps.
                        w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1:data_end) = ...
                            slmeval(w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,...
                            i-1:data_end),slm);
                        if strcmp(fc_def{1},'solarleistung') == 1
                            temp = w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1:...
                                data_end);
                            temp(temp < 0) = 0;
                            w_dat.(fc_def{1}).(fc_def{2}).int_val(...
                                1,i-1:data_end) = temp;
                        end
                    end

                    slm_new = slmengine(tmp_dat_x,tmp_dat_y,'plot','off',...
                        'knots',16,'increasing','off','leftslope',0,'rightslope',0);

                    n_dat.(fc_def{1}).(fc_def{2}).int_val(1,1:size(...
                        n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean,2)) = ...
                        slmeval(n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,1:size(...
                        n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean,2)),slm_new);
```

In diesem Teil des Programms erfolgt nun die Interpolation der Ausgangswerte. Dabei werden nur die Temperatur, der Luftdruck die Windstärke und die Niederschlagsmenge mit dem „shape language modeling“ [8] interpoliert. Für einen erstmaligen Aufruf wird die linke und rechte Steigung des Graphen auf 0 festgelegt. Liegen indes schon Daten vor, so wird die linksseitige

Steigung der letzten beiden vorangegangenen Graphenpunkte bestimmt und als Bedingung der slm Funktion für den linkseitigen Steigungswert vorgegeben. Zusätzlich muss der neue Graph mit dem äußerst rechten Graphenwert der vorigen Daten beginnen. So soll ein fließender Übergang der Graphen sichergestellt werden. Für die Daten, die nicht in einer fortlaufenden Zeitreihe gespeichert werden, sind die linken und rechten Steigungen auf 0 gesetzt.

```
                else
                    if i == 1
                        yi = spline(tmp_dat_x,tmp_dat_y,...
                            w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i:end));
                        yi(yi < 0) = 0;

                        w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i:(data_end)) = yi;
                    else
                        yi = spline(...
                            [w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,i-1)...
                            tmp_dat_x],...
                            [w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1)...
                            tmp_dat_y],w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(...
                            i-1:end));
                        yi(yi < 0) = 0;

                        w_dat.(fc_def{1}).(fc_def{2}).int_val(...
                            1,(i-1):(data_end)) = yi;
                    end

                    yi_new = spline(tmp_dat_x,tmp_dat_y,...
                        w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i:end));
                    yi_new(yi_new < 0) = 0;

                    n_dat.(fc_def{1}).(fc_def{2}).int_val(1,1:size(...
                        n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean,2)) = yi_new;

                end
                assignin('base','new_data',n_dat);
                assignin('base','weather_data',w_dat);
            end
        end
    end
end
end
```

Die übrigen Daten, werden mit einfachen spline Funktionen interpoliert. Dabei wird immer der letzte Wert des vorigen Abrufs mit in die Berechnung einbezogen. Da es bei der Sonnenscheindauer und der Einstrahlung keine negativen Werte geben kann, werden durch die Interpolation generierte negative Werte auf Null gesetzt. Zum Abschluss werden die beiden Datencontainer dem Base-Workspace zugewiesen.

| Prognosebereich | Update-zeitpunkt | Prognoseintervalle | | | |
|---|---|---|---|---|---|
| | | 00:00:00-05:59:59 | 06:00:00-11:59:59 | 12:00:00-17:59:59 | 18:00:00-23:59:59 |
| Temperatur Min. in °C | 17.01.2014 03:26 Uhr | 1 | 3 | 3 | 1 |
| | 17.01.2014 09:26 Uhr | 2 | 3 | 3 | 2 |
| | 17.01.2014 15:26 Uhr | 3 | 3 | 3 | 2 |
| | 17.01.2014 21:26 Uhr | 3 | 3 | 2 | 1 |
| | 18.01.2014 03:26 Uhr | 1 | 1 | 4 | 1 |
| | 18.01.2014 09:26 Uhr | -1 | 0 | 4 | 1 |
| | 18.01.2014 15:26 Uhr | -2 | -2 | 3 | 1 |
| | 18.01.2014 21:26 Uhr | -2 | -2 | 1 | 0 |
| Luftdruck in hPa | 17.01.2014 03:26 Uhr | 1007 | 1005 | 1005 | 1006 |
| | 17.01.2014 09:26 Uhr | 1007 | 1006 | 1004 | 1005 |
| | 17.01.2014 15:26 Uhr | 1007 | 1006 | 1005 | 1006 |
| | 17.01.2014 21:26 Uhr | 1007 | 1006 | 1005 | 1007 |
| | 18.01.2014 03:26 Uhr | 1007 | 1004 | 1001 | 1001 |
| | 18.01.2014 09:26 Uhr | 1007 | 1004 | 1000 | 1000 |
| | 18.01.2014 15:26 Uhr | 1007 | 1004 | 1001 | 1001 |
| | 18.01.2014 21:26 Uhr | 1007 | 1004 | 1001 | 1001 |

Tabelle 3.1: Datenverlust bei kleinem Updateintervall am Beispiel minimaler Temperatur und Luftdruck

## 3.10 Funktionsaufbau

# Kapitel 4

# Datenanalyse

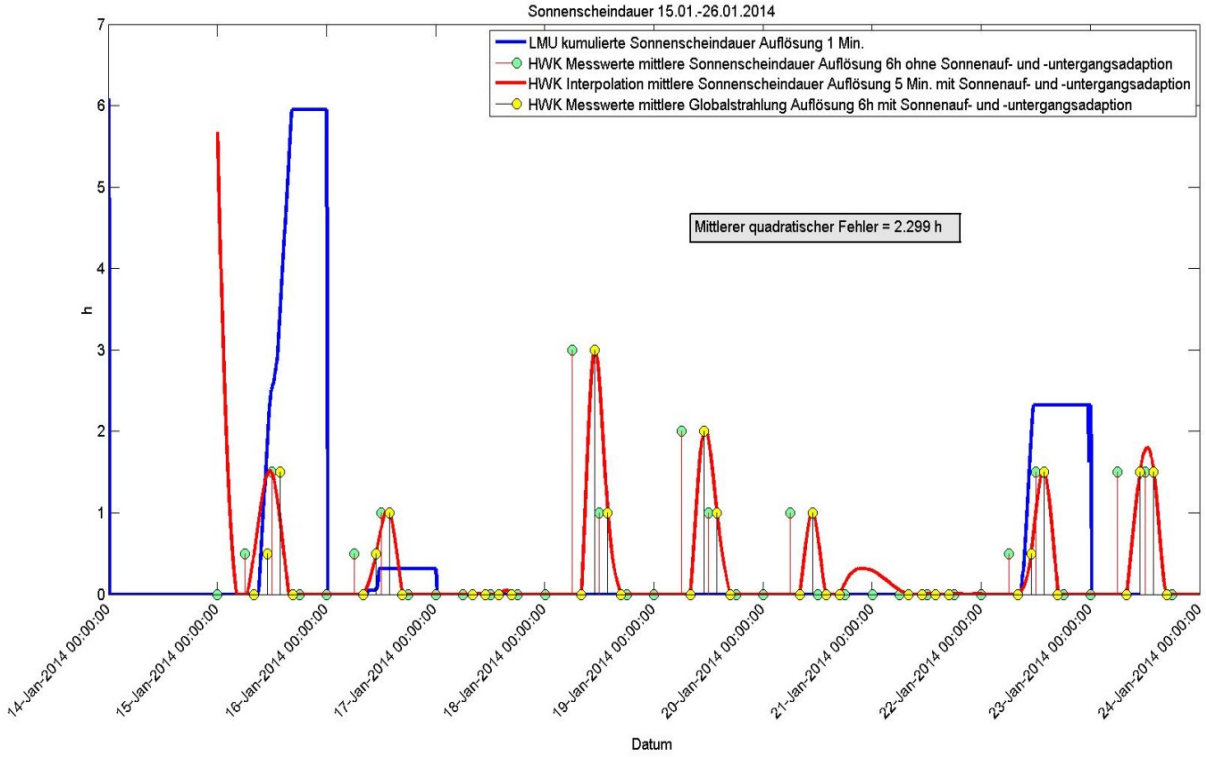



Abbildung 4.1: Vergleich der interpolierten Sonnenscheindauer mit den LMU Wetterdaten

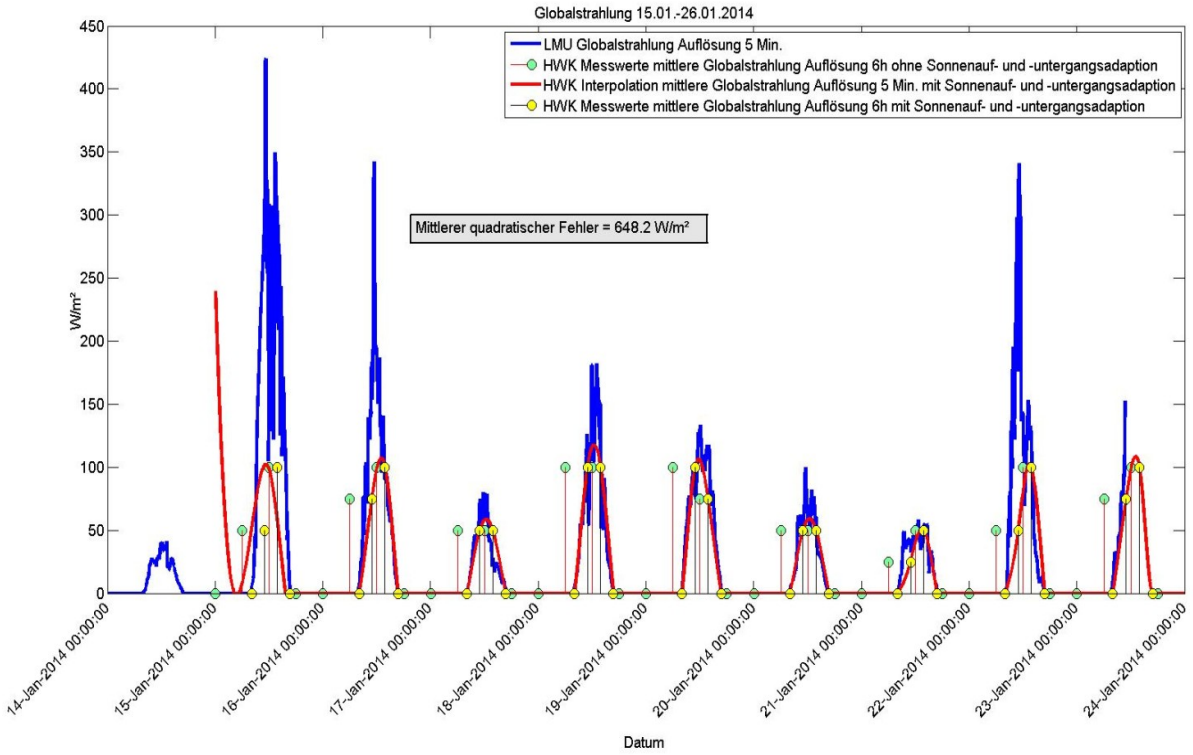



Abbildung 4.2: Vergleich der interpolierten Globalstrahlung mit den LMU Wetterdaten

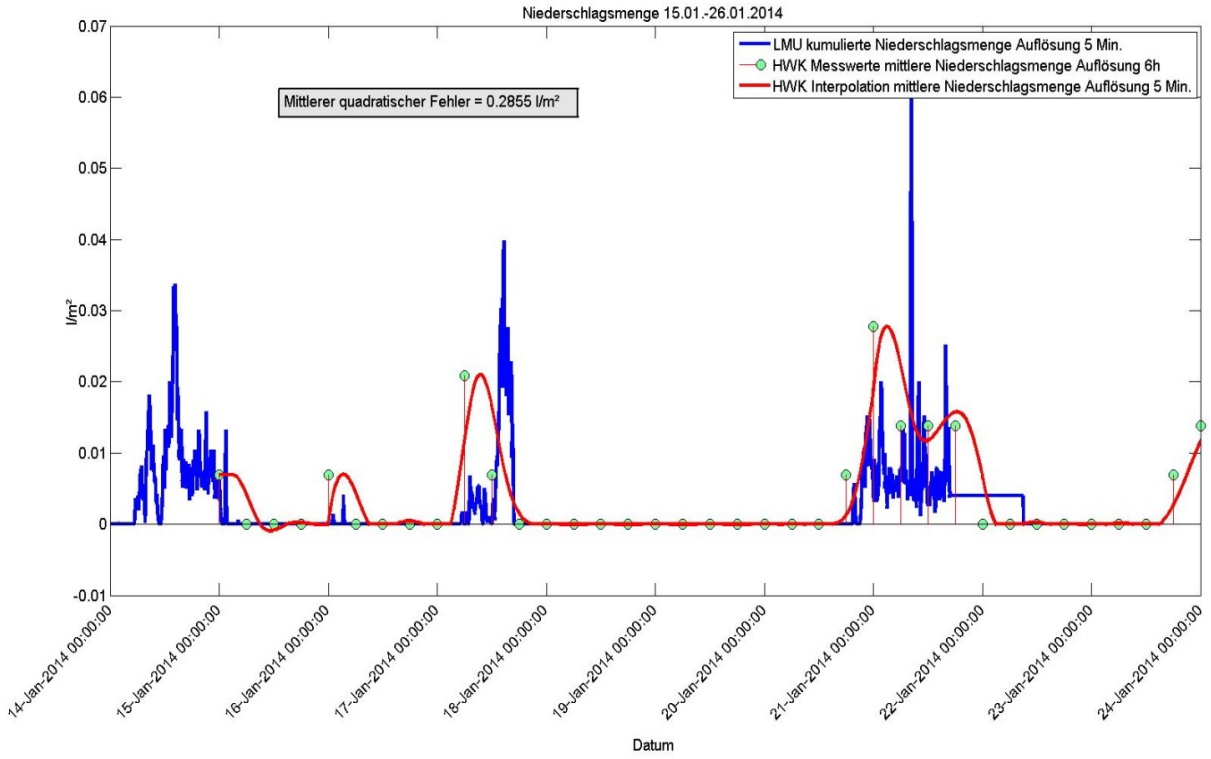



Abbildung 4.3: Vergleich der interpolierten Niederschlagsmenge mit den LMU Wetterdaten

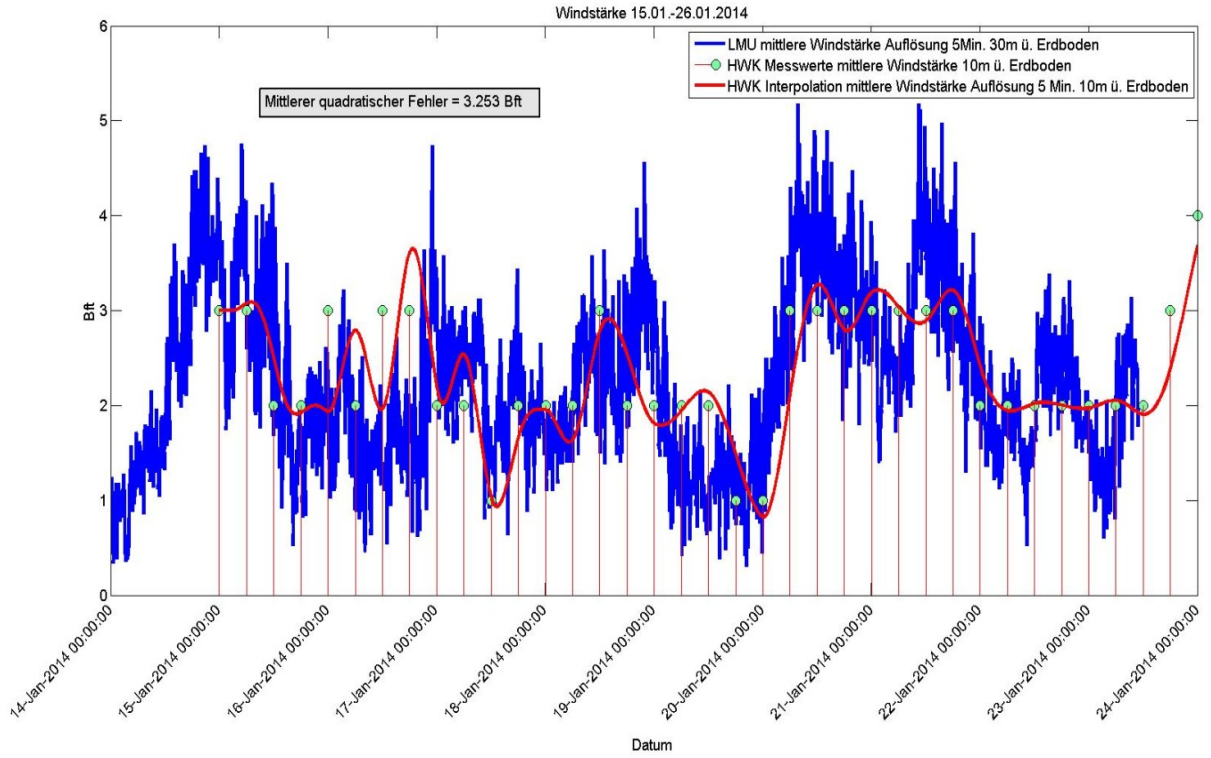



Abbildung 4.4: Vergleich der interpolierten Windstaerke mit den LMU Wetterdaten

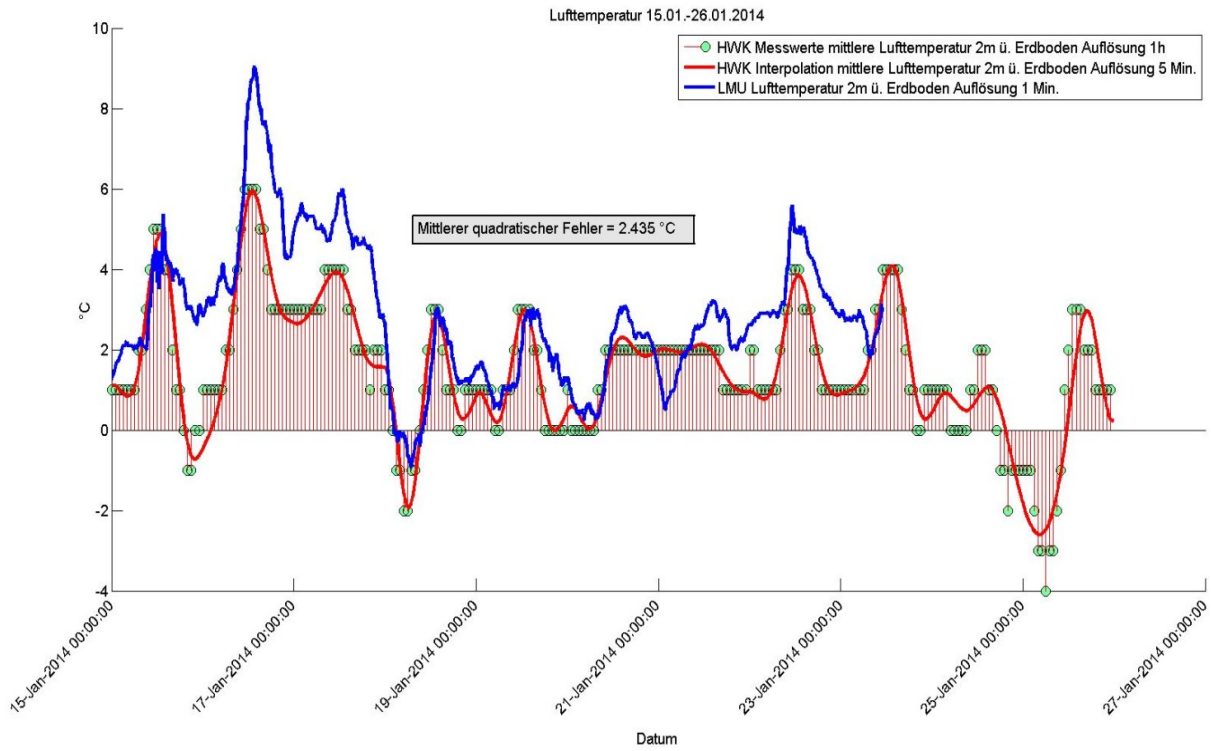



Abbildung 4.5: Vergleich der interpolierten mittleren Lufttemperatur mit den LMU Wetterdaten

# Teil III

# Schluss

# Kapitel 5

# Zusammenfassung und Ausblick

Aufgabe war es die Daten einer Wetterstation wiederholt auszulesen und für die weitere Verarbeitung zu interpolieren bzw. abzuspeichern. Für die Entwicklung des Programmcodes waren das Wissen um die MODBUS- und Wetterstationsspezifikation von entscheidender Bedeutung. Ausgehend von diesen Informationen konnte in einem ersten Schritt in einer virtuellen Umgebung die Kommunikation über die serielle Schnittstelle getestet werden. Mit den nun bekannten Funktionalitäten konnte der weitere Programmaufbau für das mehrmalige Senden und Auslesen von MODBUS-Nachrichten implementiert werden. Als kleine Schwierigkeit gestaltete sich der Aufbau der kontinuierlichen Zeitreihe, da die Grunddaten selbst in einer unterschiedlichen Auflösung vorliegen. Aber auch dieses Problem konnte letztendlich gelöst werden. Nachdem die ersten Ergebnisse vorlagen und mit den Daten des metereologischen Instituts der LMU verglichen wurden, ergaben sich neue Verbesserungsmöglichkeiten. Im Bereich der Solarleistung wurden daraufhin die Daten der Wetterstation auf den Tagesgang der Sonne gemappt. Ein nachträglicher Vergleich bestätigte den Erfolg dieser Maßnahme deutlich. Trotz der guten Datenaufbereitung gibt es noch Verbesserungspotential. Bezogen auf die Temperaturen kann aufbauend auf historische Daten, die vom lokalen Temperatursensor stammen, ein Ausgleichsfaktor bestimmt werden, welcher die Prognosedaten an die tatsächlich vor Ort vorherrschenden Temperaturen anpasst. Für einen Test über einen längeren Zeitraum als einer Woche war in dieser Arbeit leider keine Zeit. Daher wäre es zu empfehlen diesen noch durchzuführen. Der in dieser Arbeit ausgegebene Luftdruck bezieht sich auf das Niveau des Meeresspiegels. Soll ein ortsabhängiger Luftdruck in den Daten gespeichert werden, so müssen die jeweiligen Höhen der Wetterregionen und die barometrische Höhenformel mit in den Programmcode integriert werden. Die Sonnenauf- und -untergangsadaption ist bis jetzt nur für deutsche Städte vorgesehen. Sollen andere europäische Wetterregionen abgerufen werden, so müssen die Längen- und Breitengrade im Code ergänzt werden.

# Anhang

# Anhang A

# Aufbau der Wetterstation

| Prognose-Bereich | Prognosedetail | Einheit | Wertebereich | Auflösung | Prognoseintervall | zeitliche Auflösung | Registeradresse | Info |
|---|---|---|---|---|---|---|---|---|
| Temperatur | Min. Lufttemperatur | °C | < −60 bis > 65 | 1 | 3 Tage | 6h | 420 - 435 | Mittlerer Wert 2m über Erdboden |
| Temperatur | Max. Lufttemperatur | °C | < −60 bis > 65 | 1 | 3 Tage | 6h | 400 - 415 | Mittlerer Wert 2m über Erdboden |
| Temperatur | Mittlere Lufttemperatur | °C | < −60 bis > 65 | 1 | 3 Tage | 1h | 000 - 095 | Mittlerer Wert 2m über Erdboden |
| Temperatur | Lokale Lufttemperatur | °C | < −60 bis > 65 | 1 | Aktuell | 1s | 097 | Standortabhängig |
| Niederschlag | Menge | $l/m^2$ | 0-60 | dyn. | 3 Tage | 6h | 140 - 155 | Mittlerer Wert |
| Niederschlag | Wahrscheinlichkeit | % | 0 - 100 | 10 | 3 Tage | 6h | 160 - 175 | |
| Solarprognose | Sonnenscheindauer | h | 0 - 6 | 1 | 1 Tag | 6h | 180 - 187 | Mittlerer Wert |
| Solarprognose | Solare Einstrahlung | $W/m^2$ | 0 – 1200/> 1200 | 25 | 1 Tag | 6h | 190 - 197 | Mittlerer Wert Globalstrahlung |
| Luftdruck | Min. Lufttemperatur | hPa | < 938 – > 1063 | 1 | 1 Tag | 6h | 240 - 247 | Mittlerer Wert bezogen auf NN |
| Windprognose | Stärke | Bft | 0 - 12 | 1 | 3 Tage | 6h | 200 - 215 | Mittlerer Wert 10m über Erdboden |
| Windprognose | Richtung | | 1 - 8 | 1 | 3 Tage | 6h | 220 - 235 | N/NO/O/SO/S/SW/W/NW |
| Markantes Wetter | Böen | | 0,1,2,3 | 1 | 3 Tage | 6h | 310 - 325 | 0=keine Böen, 1=45km/h, 2=72km/h, 3=99km/h |
| Markantes Wetter | Bodennebel | | 0,1 | 1 | 3 Tage | 6h | 250 - 265 | 0=Wahrscheinlichkeit<=50%, 1=>50% |
| Markantes Wetter | Gefrierender Regen | | 0,1 | 1 | 3 Tage | 6h | 270 - 285 | 0=Wahrscheinlichkeit<=50%,1=>50% |
| Markantes Wetter | Hitze | °C | 0,1,2,3,4 | 1 | 3 Tage | 6h | 350 - 365 | 0=<27°C, 1=27–31°C, 2=32–40°C, 3=41–53°C, 4=>54°C |
| Markantes Wetter | Kälte | °C | 0,1,2,3,4 | 1 | 3 Tage | 6h | 370 - 385 | 0=keine Info, 1=<-15°C, 2=<-20°C, 3=<-25°C, 4=<-30°C |
| Markantes Wetter | Bodenfrost | | 0,1 | 1 | 3 Tage | 6h | 290 - 305 | 0=Wahrscheinlichkeit<=50%, 1=>50% |
| Markantes Wetter | Niederschlag | | 0,1,2,3 | 1 | 3 Tage | 6h | 330 - 345 | 0=keine Info, 1=10mm, 2=50mm, 3=keine Info |
| Signifikantes Wetter | | | 1 - 15 | 1 | 3 Tage | 6h | 120 - 135 | 1=sonnig/klar, 2=leicht bewölkt, 3=vorwiegend bewölkt, 4=bedeckt, 5=Wärmegewitter, 6=starker Regen, 7=Schneefall, 8=Nebel, 9=Schneeregen, 10=Regenschauer, 11=leichter Regen, 12=Schneeschauer, 13=Frontengewitter, 14=Hochnebel, 15=Schneeregenschauer |

Tabelle A.1: Detailierte Datenstruktur der Wetterstation[1, S. 17-26]

# Anhang B

# Das MODBUS Protokoll

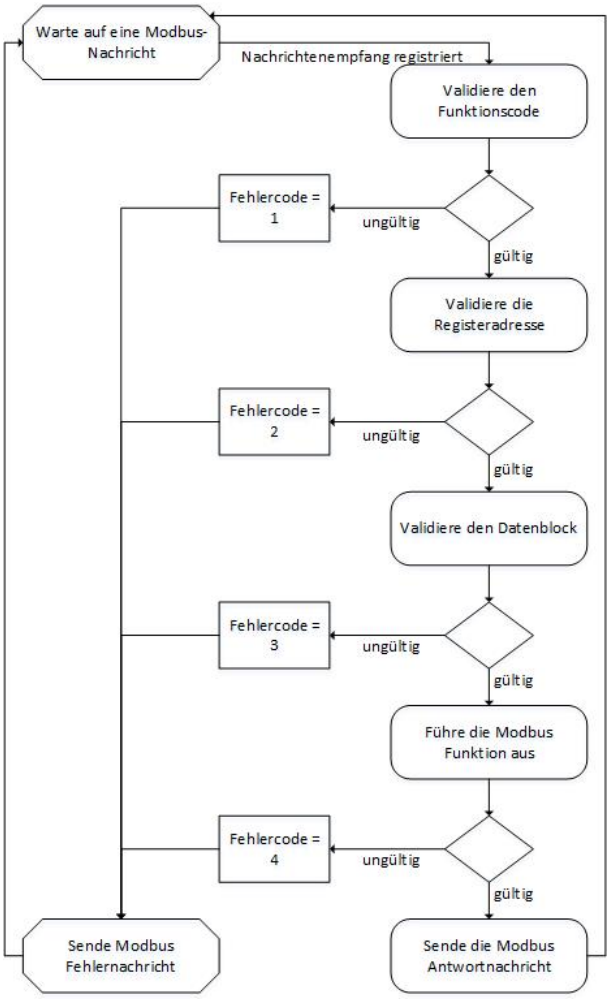



Abbildung B.1: Ablaufdiagramm für die MODBUS Nachrichtenüberprüfung

| | | | | Function Codes | | |
|---|---|---|---|---|---|---|
| | | | | Code | Sub Code | Hex |
| Data Access | Bit Access | Physical Discret Inputs | Read Discrete Inputs | 02 | | 02 |
| | | Internal Bits or Physical coils | Read Coils | 01 | | 01 |
| | | | Write Single Coil | 05 | | 05 |
| | | | Write Multiple Coil | 15 | | 0F |
| | 16 bit access | Physical Input Registers | Read Input Register | 04 | | 04 |
| | | Internal Registers or Physical Output Registers | Read Holding Registers | 03 | | 03 |
| | | | Write Single Registers | 06 | | 06 |
| | | | Write Multiple Registers | 16 | | 10 |
| | | | Read/Write Multiple Registers | 23 | | 17 |
| | | | Mask Write Register | 22 | | 16 |
| | | | Read FIFO queue | 24 | | 18 |
| | File record access | | Read File Record | 20 | | 14 |
| | | | Write File Record | 21 | | 15 |
| Diagnostics | | | Read Exception status | 07 | | 07 |
| | | | Diagnostic | 08 | 00-18,20 | 08 |
| | | | Get Com Event Counter | 11 | | |
| | | | Get Com Event Log | 12 | | 0C |
| | | | Report Server ID | 17 | | 11 |
| | | | Read Device Identification | 43 | 14 | 2B |
| Other | | | Encapsulated Interface Transport | 43 | 13,14 | 2B |
| | | | CANopen General Reference | 43 | 13 | 2B |

Abbildung B.2: Übersicht der zur Verfügung stehenden Funktionscodes in MODBUS

# Anhang C

# Hilfsprogramme

# Anhang D

# Abkürzungsverzeichnis

**NN** Normal Null Meeresspiegel

**hPa** Hektopascal

**Bft** Beaufort

**mm** Millimeter

**km** Kilometer

**h** Stunde

**C** Celsius

**N** Nord

**NO** Nordost

**O** Osten

**SO** Suedost

**S** Sueden

**SW** Suedwest

**W** Westen

**NW** Nordwest

# Literaturverzeichnis


[1] HKW-Elektronik GmbH. *Wetterprognose-Station Kompakt WS-K xx Modbus*. HKW-Elektronik GmbH, Industriestraße 12, D-99846 Seebach/Thur, November 2012. Stand 23.11.2012.

[2] Nationale Plattform Elektromobilität (NPE). Fortschrittsbericht der nationalen plattform elektromobilität. Website, 2012. Online verfügbar auf `http://www.bmu.de/fileadmin/bmu-import/files/pdfs/allgemein/application/pdf/bericht_emob_3_bf.pdf`; zuletzt geprüft am 24.01.2014.

[3] Bayern Innovativ Gesellschaft für Innovation und Wissenstransfer mbH. http://www.elektromobilitaet-verbindet.de/. Website, Januar 2014. Online verfügbar auf `http://www.elektromobilitaet-verbindet.de/projekte/energieautarke-elektromobilitaet.html`; zuletzt geprüft am 24.01.2014.

[4] Gerd Küveler and Dietrich Schwoch. *Informatik für Ingenieure und Naturwissenschaftler*. Viewegs Fachbücher der Technik. Vieweg, Braunschweig [u.a.], 5 edition, 2007.

[5] Modbus Organization, Inc. Modbus application protocol specification. Website, April 2012. Online verfügbar auf `http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf`; zuletzt geprüft am 24.01.2014.

[6] Manfred Schleicher. *Digitale Schnittstellen und Bussysteme: Grundlagen und praktische Hinweise zur Anbindung von Feldgeräten an MOD-Bus, PROFIBUS-DP, ETHERNET, CANopen und HART*. JUMO, Fulda, 5 edition, 2008.

[7] Andreas Wagner. *Photovoltaik Engineering: Handbuch für Planung, Entwicklung und Anwendung*. VDI-Buch. Springer-Verlag Berlin Heidelberg, Berlin and Heidelberg, 2 edition, 2006.

[8] John D'Errico. Slm - shape language modeling. Website, Juni 2009. Online verfügbar auf `http://www.mathworks.com/matlabcentral/fileexchange/24443-slm-shape-language-modeling`; zuletzt geprüft am 06.01.2014.