

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

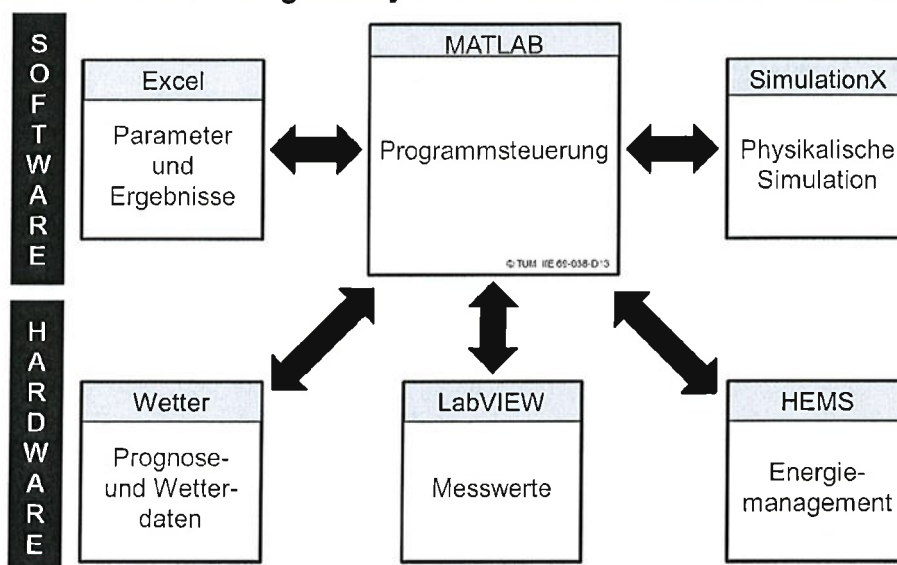
Zusammenfassung

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. Eine ausführlichere Datenanalyse könnte in diesem Zusammenhang mehr Aufschluss geben. Doch war dies nicht der Schwerpunkt dieser Arbeit.

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, den 31.01.2014

.....

Andreas Henneberger

Inhaltsverzeichnis

I	Einleitung	6
II	Hauptteil	8
1	Aufbau der Wetterstation	9
1.1	Aufbau der Datenstruktur	9
1.2	Technischer Aufbau der Station	10
1.2.1	Senderauswahl und Stationsaufbau	10
1.2.2	Registereinteilung und Schnittstellenparametrierung	11
1.2.3	Datenabruf	12
2	Das Modbus-Protokoll	14
2.1	Verbindungstypen	14
2.2	Nachrichtenaufbau	15
2.3	Registertypen	16
2.4	Nachrichtenverarbeitung	16
2.5	Funktions- und Fehlercodes	16
2.6	Cyclic Redundancy Check	18
2.7	Simulation der Wetterstation	20
3	Aufbau und Dokumentation des Programmcodes in MATLAB	21
3.1	Geforderte Funktionseigenschaften	21
3.2	Vorbereitende Maßnahmen	24
3.2.1	Zuweisung variabler Inputparameter und Variableninitialisierung	24
3.2.2	Aufbau von Strukturen	26
3.2.3	Überprüfung der Eingabeparameter	26
3.2.4	Verfügbarkeitsprüfung der seriellen Schnittstelle	31
3.3	Aufbau der seriellen Schnittstelle	31
3.4	Abgleich der Wetterregion im Register	32
3.5	Festlegung der Timerparameter und Starten des Timers	33
3.6	Abschicken der MODBUS-Anfragen	36
3.7	Senden und Empfangen	40
3.8	Rx-Datenverarbeitung	41

3.9	Datenverarbeitung	43
3.9.1	Interpolation	51
4	Datenanalyse	59
4.1	Sonnenscheindauer	59
4.2	Globalstrahlung	61
4.3	Niederschlagsmenge	63
4.4	Windstärke	64
4.5	Mittlere Lufttemperatur	66
III	Schluss	68
5	Zusammenfassung und Ausblick	69
	Anhang	70
	Anhang	71
A	Aufbau der Wetterstation	71
B	Das MODBUS Protokoll	72
C	Hilfsfunktionen	74
C.1	read_com_set	74
C.2	write_com_set	75
C.3	stop_timer	76
C.4	get_reg_address	77
C.5	reg_num	78
C.6	format_modbus_msg	78
C.7	fcode_check	79
C.8	MESZ_calc	79
C.9	res_factor	80
C.10	utc2date	81
C.11	tvector	82
C.12	date2utc	84
C.13	data_mult	84
C.14	diurnal_var	85
C.15	neg_val_corr	86
C.16	crc_check	87
D	Abkürzungsverzeichnis	89

Abbildungsverzeichnis

1.1	Verfuegbare Prognosedaten WS-K RTU485 WPAia	10
1.2	Standorte der Langwellensender und des Empfängers [1]	10
2.1	Skizze der Funktionsweise des MODBUS Protokolls [2, S. 5]	14
2.2	Aufbau einer MODBUS Nachricht	15
3.1	Ablaufplan der Funktion forecast_data	22
3.2	Beispiel für den Funktionsaufruf	23
3.3	Neuberechnung Intervallanzahl und -startwerte	52
3.4	Parameterbestimmung für die slm-Funktion	55
3.5	Vergleich der Interpolationsfunktionen	56
4.1	Ergebnis der Sonnenaufgangs- und Untergangsadaption	59
4.2	Vergleich der interpolierten Sonnenscheindauer mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014	60
4.3	Vergleich der interpolierten Globalstrahlung mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014	62
4.4	Vergleich der interpolierten Niederschlagsmenge mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014	63
4.5	Vergleich der interpolierten Windstaerke mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014	65
4.6	Vergleich der interpolierten mittleren Lufttemperatur mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014	66
B.1	Ablaufdiagramm für die MODBUS Nachrichtenüberprüfung [3, S. 9]	72
B.2	Übersicht der zur Verfügung stehenden Funktionscodes in MODBUS [3, S. 11] . .	73

Tabellenverzeichnis

1.1	Einstellungsparameter für den Kommunikationsaufbau und den Wetterbereich . .	11
1.2	Schnittstellenparameterbelegung	12
1.3	Beispiel für das Unterschiedliche Datenupdate	13
2.1	Übersicht der Registerarten im MODBUS Protokoll	16
2.2	Aufbau einer lesenden Kommunikation mit einem Coil-Register	17
2.3	Aufbau einer schreibenden Kommunikation mit einem Holding-Register	17
2.4	Aufbau einer lesenden Kommunikation mit einem Holding-Register	18
3.1	Inputparameter für die Funktion send_loop	36
4.1	Absolute Abweichung der Sonnenscheindauer von den LMU Wetterdaten in Stunden, Beobachtungsintervall 15.-24.01.2014	61
4.2	RMSE der Globalstrahlung in W/m^2	62
4.3	RMSE der Niederschlagsmenge in l/m^2	64
4.4	RMSE der Windstärke in Bft	65
4.5	RMSE der mittleren Lufttemperatur in $^{\circ}C$	67
A.1	Detaillierte Datenstruktur der Wetterstation[4, S. 17-26]	71

Teil I

Einleitung

Schenkt man der Studie von „Global EV Outlook“ glauben, so wird die Mobilität in Zukunft durch elektrisch angetriebene Fahrzeugen mit geprägt sein [5].

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.[6]

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 Energieversorgung beschäftigt. Ein Aufgabenschwerpunkt hierbei ist es ein integriertes Energiemanagementsystem mittels Aufbau und Betrieb eines Hardware-in-the-Loop Prüfstands zu evaluieren.[7]

Da es sich hierbei um ein Einfamilienhaus handelt spricht man von einem Home Energiemanagementsystem (HEMS). Was ist die Aufgabe eines solchen Systems und was macht ein solches System aus? Eine Antwort auf diese Fragen gibt der Artikel „Link to future“ auf den sich die nachfolgenden Angaben beziehen [8]. Hier ist es Aufgabe eines HEMSs den Nutzer mit umfassenden Funktionen zum internen Informationsaustausch zu versorgen. Diese Informationen dienen letztendlich dazu den täglichen Energieverbrauch zu optimieren, um dadurch bei gleichbleibender Lebensqualität Kosten zu sparen. Drei Bausteine bilden dabei das Grundgerüst für das HEMS.

1. Das „Energy Management Gateway“ übernimmt dabei die Aufgabe des sicheren Datenaustauschs zwischen den hausinternen Gerätschaften sowie über das Versorgungsnetz zu den Energieversorgern.
2. Die „Energy Management Unit (EMU)“ sammelt alle Daten über den Energieverbrauch, die Energieerzeugung sowie -speicherung in einem Haushalt. Abgeleitet aus diesen Informationen und den momentanen Preisen für Energieverbrauch und -erzeugung regelt sie den Einsatz der Geräte.
3. Die Bereitstellung von Informationen für die EMU erledigt ein Netzwerk von Sensoren und Microcontrollern. Hierzu wird häufig ein Home Area Network entweder drahtgebunden oder über Funk installiert.

Diese Arbeit orientiert sich am dritten Baustein und soll Wetterdaten, einer später extern am Haus angebrachten Wetterstation, der EMU zur Verfügung stellen. Die EMU wird hierbei durch ein MATLAB-Programm dargestellt.

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 [4]. Der Hersteller bietet für die Hardware eine Reihe von Lizenzmodelle 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 **Abbildung 1.1** entnommen werden. Für welche geografischen 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 [4, 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.

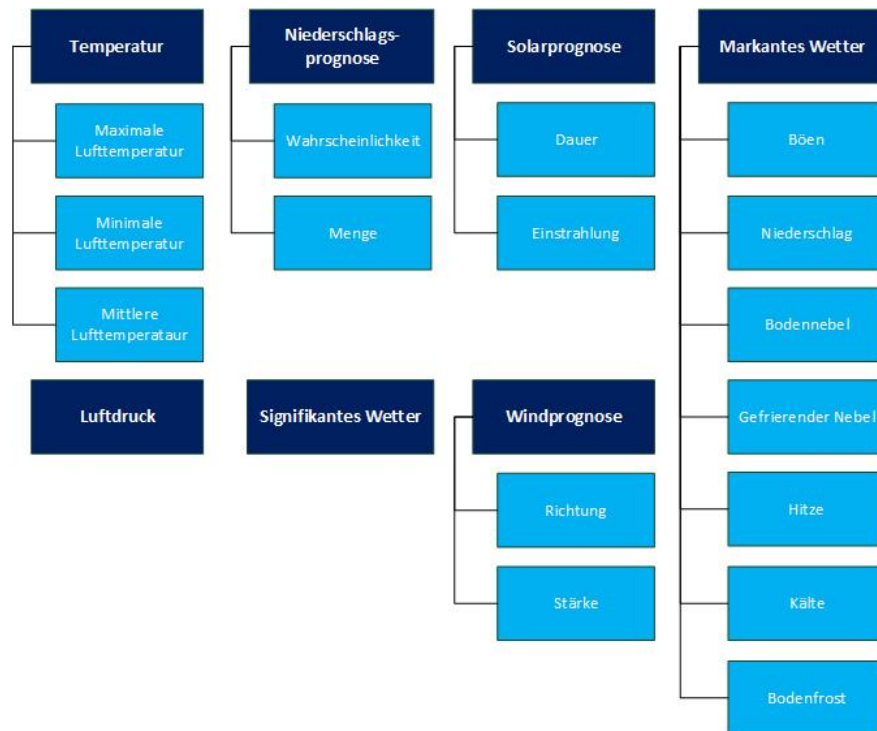


Abbildung 1.1: Verfügbare Prognosedaten WS-K RTU485 WPAia

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)

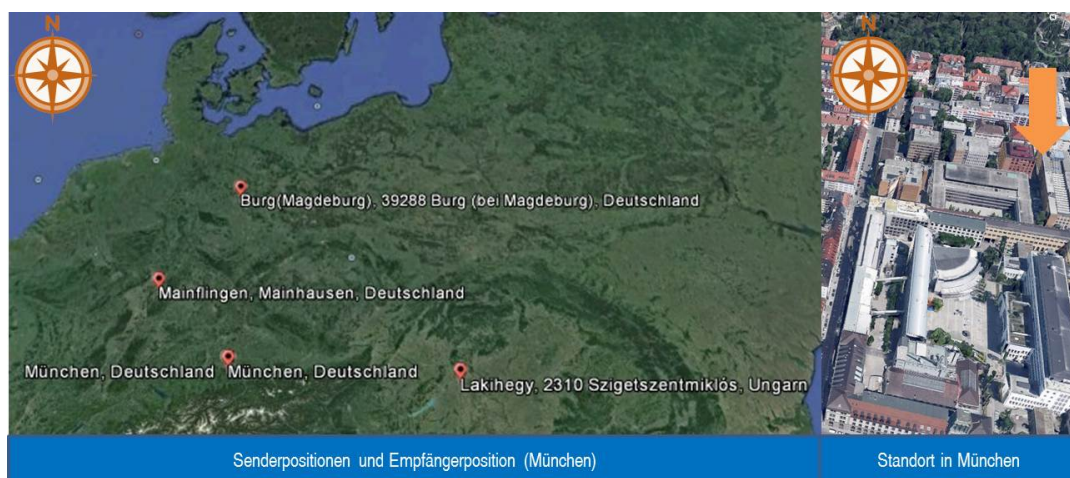


Abbildung 1.2: Standorte der Langwellensender und des Empfängers [1]

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 1 = HGA 22 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 nachfolgenden Kapitel „Das MODBUS-Protokoll“ noch ausführlicher behandelt, können vier Arten von Registern über die MODBUS Kommunikation angesprochen werden. In der Wetterstation werden zwei dieser Register mit Daten gefüllt sein, die von Interesse sind. In dem sogenannten Holdingregister, das später das bedeutenste sein wird, können Einstellungsparameter gesetzt und gelesen werden. Eine Übersicht gibt die oben aufgeführte **Tabelle 1.1**. Außerdem sind sämtliche meteorologischen Daten in diesem Register abgelegt. Eine Auflistung

der den Prognosebereichen zugeordneten Registeradressen und weitere Informationen 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.

Tabelle 1.2: Schnittstellenparameterbelegung

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

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 [9, 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 Kommunikationsprotokoll erfordert in diesem Fall eine erneute Übertragung der Information [9, 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 [9, S. 169-170]. Per Jumperpositionierung kann die Baudrate von 9600 auf 19200 bit/s in der Wetterstation geändert werden. Die Werkseinstellung von 19200 bit/s wurde aber in dieser Arbeit beibehalten. Ebenso einstellbar über Jumper ist die Slave-ID der Wetterstation. Da in einem Bus-Netzwerk mehrere Teilnehmer mit einander verbunden sind, benötigt man einen Identifikationsparameter der Teilnehmer für den Nachrichtenaustausch. Aber auch hier wurde der voreinstellte Wert nicht verändert.

1.2.3 Datenabruf

In der Spezifikation des Herstellers steht wörtlich „Alle Daten werden täglich viermal neu berechnet und übertragen“ [4, S. 4]. Die Frage, die sich hierbei stellt ist, werden im Tagesfortlauf auch Daten neu übermittelt, die zeitlich gesehen schon in der Vergangenheit liegen? Nimmt man an, der Abruf erfolgt um 17 Uhr, dann würden gem. der Spezifikation auch die Werte für die Intervalle 0-6 Uhr und 6-12 Uhr neu berechnet und übermittelt werden. Im Telefonat mit Herrn Volkhardt von der Firma HKW GmbH, stellte sich heraus, dass die maximal 4 Updates pro Tag nicht immer das komplette Register erneuern. Dies gilt nur für die Temperatur und markanten Wetterdaten. Für alle anderen Bereiche werden jeweils nur die aktuellen bzw. in Zukunft liegenden Werte übermittelt. Die **Tabelle 1.3** zeigt dies im Beispiel. Die Funktion in MATLAB wird der Einfachheit halber später keinen Unterschied hierin machen und generell alle Register abfragen, sprich auch Werte, die an sich nicht mehr aktualisiert werden.

Tabelle 1.3: Beispiel für das Unterschiedliche Datenupdate

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

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 [3]. 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.

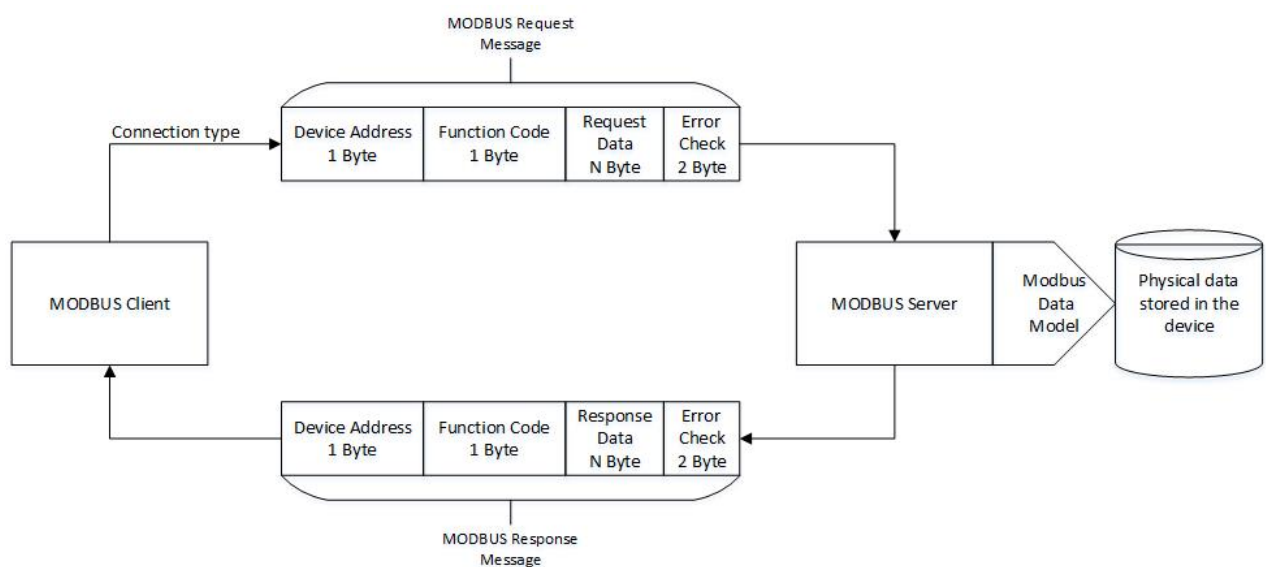


Abbildung 2.1: Skizze der Funktionsweise des MODBUS Protokolls [2, S. 5]

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 1200 m betragen.[10]

Diese Eigenschaften bieten sich an, um die Wetterstation wie in der Einleitung erwähnt, in ein Sensornetzwerk zu integrieren. Das Energiemanagementsystem kann die Daten entsprechend auslesen und Aktoren einstellen.

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 aus

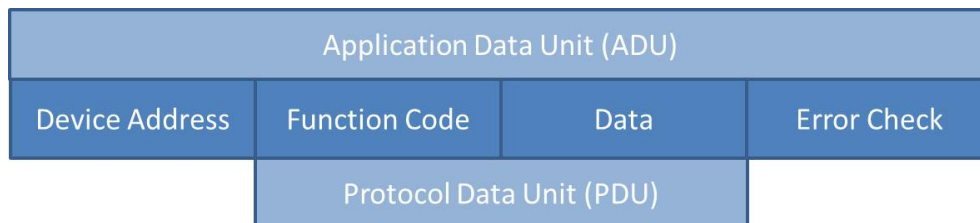


Abbildung 2.2: Aufbau einer MODBUS Nachricht

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. Ein besonderes Augenmerk muss auf die Adressierung im Datenblock gelegt werden. Da hier nur die Startadresse angegeben wird und die Anzahl der nachfolgenden Adressen, ist es nicht möglich mit einer Nachricht nicht konsequente Registeradressen auszulesen oder zu beschreiben. Hierfür wäre jeweils eine eigene Nachricht notwendig. Die Slave-ID 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 hex) und der Fehlermeldung

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

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 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

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

und bei 65535 endet. Zieht man noch eine weitere Quelle heran, so liegen die gültigen Adressbereiche jedoch anders verteilt vor. So besitzen die Coil-Register nur einen gültigen Adressraum von 1-9999, der Diskrete Input einen von 10001-19999, das Input Register einen von 30001-39999 und das Holding Register ab der Adresse 40001-49999 [2].

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. Die Umsetzung dieser Überprüfung erfolgt später in der rx-Datenverarbeitung in Kapitel 3.8 auf Seite 41.

2.5 Funktions- und Fehlercodes

Wie bereits erwähnt, ist der Funktionscode ein entscheidender Baustein in der MODBUS Nachricht. Es ist daher von Vorteil die für die Zwecke dieser Arbeit Wichtigen zu identifizieren, um das Handling des Programms zu vereinfachen. Die im Anhang dargestellte **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. Bei dem

Versuch diese Funktionscodes auszuführen, wurden lediglich Fehlercodes mit der Ausnahme 1 zurückgegeben. Daher ist davon auszugehen, dass die Funktionscodes für den Bereich Services File Record Access, Diagnostics und Other in dieser Hardware nicht gültig sind. Wie schon im Abschnitt 1.2.2 auf Seite 12 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 mit allgemeingültigen Ergänzungen 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

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

Nachrichtentyp	Nachrichtenteil	Wortlänge	Inhalt
Anfrage	Funktionscode	1 Byte	0x01
	Startadresse	2 Bytes	H-Byte 0x00 L-Byte 0x00 (0x0000 bis 0xFFFF möglich)
Antwort	Adressanzahl	2 Bytes	H-Byte 0x00 L-Byte 0x01 (1 bis 2000 (0x7D0) möglich)
	Funktionscode	1 Byte	0x01
	Byteanzahl	1 Byte	0x01 (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	0x01 (8 Coilzustände werden mit einem Byte, hier 00000001 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

Nachrichtentyp	Nachrichtenteil	Wortlänge	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

Funktionscode ist der Code 0x06, mit dem man in das Holding-Register Werte schreiben kann. Wie im Abschnitt 1.2.2 auf Seite 11 in der **Tabelle 1.1** nachzulesen, wird die zu beobachtende

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 0x60 (1 bis 125 (0x7D) möglich)
Antwort	Funktionscode	1 Byte	0x03
	Byteanzahl	2 Byte	2 x N (N = Adressanzahl)
	Registeroutput	N x 2 Bytes	

Wetterregion mit einem Wert im Holdingregister an der Adresse 112 festgelegt. Für die Definition der Wetterregion München (Wert 353 dezimal) ist die hierzu notwendige Kommunikation in der **Tabelle 2.3** als Beispiel skizziert. Der wohl wichtigste und am meisten verwendete Funktionscode in dieser Arbeit ist der Code 0x03 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, 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 01 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.

2.6 Cyclic Redundancy Check

Im MODBUS Protokoll sind zwei Modi definiert, wie die Nachrichten aufgebaut sein können. Da die Wetterstation aber im RTU Modus betrieben wird, muss der ASCII Modus in dieser Arbeit nicht berücksichtigt werden. Die Methode zum Absichern der Datenintegrität im RTU Modus ist der Cyclic Redundancy Check. Der Master sendet die MODBUS Nachricht an den Client, der wiederum den CRC Wert aus der Slave-ID, und der ADU berechnet. Kommt er auf

ein anderes Ergebnis als es im CRC Wert steht, wird er die Anfrage nicht beantworten und es kommt zu einem Fehler, den der Master lösen muss.[2]

Analog dem Algorithmus aus dem Buch „Digitale Schnittstellen und Bussysteme“ wird in MATLAB die CRC Prüfsumme berechnet. Zunächst wird die Nachricht aus Slave-ID und PDU (*modbus_pud_hex*) an die Funktion **crc_calc** übergeben. Die Länge dieser Nachricht geteilt durch zwei ergibt die enthaltene Anzahl an Bytes. Diese müssen im nächsten Schritt auf ein entsprechend breites binäres Bit Wort transformiert werden. Danach erfolgt die Initialisierung des Ausgangsschieberegisters, des Generatorpolynoms und der beiden Bytepositionszeiger *m* und *n*. Die MODBUS Nachricht wird durch die erste for-Schleife byteweise bearbeitet. Dabei werden die einzelnen Bytes zuerst auf 16 Bit breite Worte gebracht indem Nullen der rechten Seite zugewiesen werden. Danach folgt eine Exklusiv-Oder-Verknüpfung mit dem Ausgangsschieberegister 0xFFFF hex. Jetzt erfolgt für jedes Bit im aktuell anstehenden Byte ein Rechtsschieben bis das erste Bit mit einer eins herausgeschoben wurde. Die frei werdenden Bits auf der linken Seite werden mit Nullen aufgefüllt. Im nächsten Schritt wird mit dem Generatorpolynom wieder eine Exklusiv-Oder Operation durchgeführt und die Prozedur wird mit dem Ergebnis fortgesetzt. Am Ende wird das Resultat *crc_erg* dem Schieberegister zugewiesen und die Bytepositionszeiger erhöht. Sind alle Schleifen durchlaufen, wird zum Schluss das Schieberegister in einen 2 Byte hex Wert transformiert und mit der Slave-ID und der PDU zur ADU (*txdata_hex*) verknüpft.[10]

```
function [ txdata_hex ] = crc_calc( modbus_pud_hex )
%This function calculates the cyclic redundancy check value. Its input
%are the message bytes in hexadecimal format, i.e. 01010000000A
% Detailed explanation goes here

%Determine the number of bits the given message has
databits = size(modbus_pud_hex,2)*4;

%Convert the hexadecimal input into a binary format
datastream = hexToBinaryVector(modbus_pud_hex,databits);

%Define the shiftregister
CRCshiftreg_bin = hexToBinaryVector('FFFF',16);

%Define the generator polynom
CRCgenpolynom_bin = hexToBinaryVector('A001',16);

%Extract byte by byte from datastream and represent it as a 16bit word
n = 8;
m = 1;
for s = 1 : ceil(size(modbus_pud_hex,2)/2)
% data_byte will be the first byte for s = 1 in logical format
    data_byte = datastream(1,m:n);
% add zeros to make it a 16 bit word
    data_16bit = logical([zeros(1,8) data_byte]);
% xor the word with the shift register
    crc_erg = xor(data_16bit,CRCshiftreg_bin);
```

```

% for all 8 bits of the data_byte do
% shift word to the left for each left bit = 0 until bit = 1, refill word
% with zeros from the right side. If bit = 1 xor the word with the
% CRCgenpolynom.
    for t = 1 : 8
        if crc_erg(1,end) == 0
            crc_erg = logical([0 crc_erg(1,1:end-1)]);
        else
            crc_erg = xor(logical([0 crc_erg(1,1:end-1)]),CRCgenpolynom_bin);
        end
    end
% assign the result to the shift register and increase the byte counter
    CRCshiftreg_bin = crc_erg;
    m = m + 8;
    n = n + 8;
end
% convert the result in a 2 byte hex word and combine PDU and crc check to
% ADU
crc_16 = dec2hex(bin2dec(sprintf('%i',CRCshiftreg_bin)),4);
txdata_hex = strcat(modbus_pud_hex,crc_16);
end

```

2.7 Simulation der Wetterstation

Nachdem nun die MODBUS und Wetterstationsspezifikation analysiert wurden, kann man mit der Umsetzung des MATLAB Programmes beginnen. Für den Fall, dass man noch nie in MATLAB mit einer seriellen Schnittstelle gearbeitet hat, bietet es sich an, diese zunächst einmal zu simulieren. Hierzu wurden in dieser Arbeit zwei Programme verwendet. Das eine Programm simuliert den COM Port des MODBUS Slave, heißt „Virtual Serial Ports Emulator“ und wird von „eterlogic.com“ zum Download angeboten [11]. Das andere Programm virtualisiert den MODBUS Slave selbst und heißt „PeakHMI MODBUS Serial RTU slave“. Anbieter hierfür ist die Firma Everest Software LLC [12]. Der große Vorteil in der Simulation liegt darin, dass man in das virtualisierte Holdingregister an eine bestimmte Adresse Werte schreiben kann. Mit den Methoden von MATLAB kann man nun versuchen diesen Wert auszulesen. Mit dieser Methodik kann man schnell die Funktionalität des erarbeiteten Codes auch offline erproben.

Kapitel 3

Aufbau und Dokumentation des Programmcodes in MATLAB

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 diesen Voraussetzungen gerecht zu werden, ist es erforderlich vor allem große Datensätze im Code einzubinden. 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

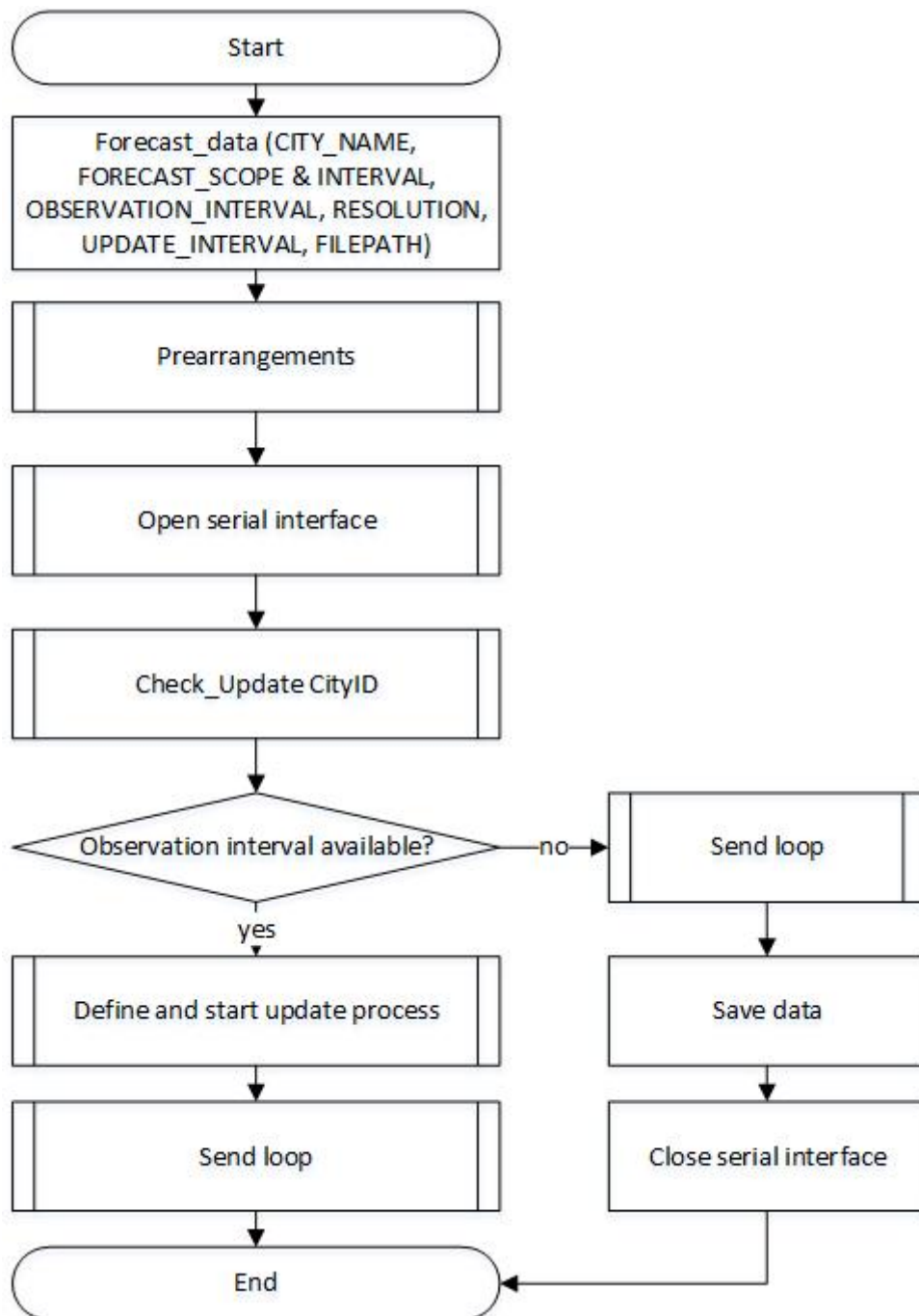
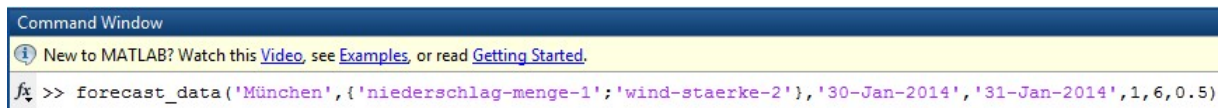


Abbildung 3.1: Ablaufplan der Funktion `forecast_data`



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
fx >> forecast_data('München',{ 'niederschlag-menge-1'; 'wind-staerke-2' }, '30-Jan-2014', '31-Jan-2014', 1, 6, 0.5)
```

Abbildung 3.2: Beispiel für den Funktionsaufruf

Programmaufbaus in **Abbildung 3.1** gegeben. Wie in der **Abbildung 3.2** zu erkennen, ist es möglich für die Funktion `forecast_data` acht Eingabeparameter zu definieren.

1. Wetterregion, z.B. **'München'**
2. Prognosebereichsdefinition z.B. **'niederschlag-menge-all'**
3. Start des Beobachtungszeitraums, z.B. **'23-Jan-2014'**
4. Ende des Beobachtungszeitraums, z.B. **'23-Jan-2014'**
5. zeitliche Auflösung, z.B. **1**
6. Updateintervall, z.B. **6**
7. zeitlicher Offset nach dem Ausführen der Funktion, z.B. **3**
8. Pfadangabe des Speicherortes (optional), z.B. **'C:\Test'**

Die Wetterregion wird als Stadtname in Stringformat übergeben. Die Prognosebereichsdefinition besteht aus drei Teilen, die als String in diesem 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. Um mehrere selektive Prognosebereiche abzurufen, gibt es die Möglichkeit diese als Cell-Array zu definieren. Dabei wird das Cell-Array als Zeilenvektor in dieser Form $\{Prognosebereichsdefinition_1; Prognosebereichsdefinition_2\}$ definiert. Möchte man keine gesonderte Auswahl treffen und einfach alle möglichen Wetterdaten 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 6, 1, 0.5, 0.25, 0.08 stellvertretend stehend für eine Interpolation mit einer Auflösung von einer Stunde, einer halben Stunde, einer viertel Stunde und 5 Min., möglich. Wählt man den Wert 6 so erfolgt keine Interpolation der Daten. Das Updateintervall wird ebenfalls als Double definiert. Hier stehen die Werte 6, 12 und 24 zur Verfügung, welches einem vier-, zwei- und einmaligen Update am Tag entspricht. Wann das einmalige Update am Tag genau erfolgen soll, kann man mit dem vorhin erwähnten Offset festgelegt werden. Umfasst der Beobachtungszeitraum den aktuellen Tag der Funktionsausführung, so verschiebt sich der Datenabruf nach Funktionsausführung um den Offsetwert nach hinten.

Liegt der Startpunkt des Beobachtungszeitraums in der Zukunft, erfolgt die Offsetverschiebung ab 0.00 Uhr des Beobachtungsbeginns. Die Pfadangabe wird wiederum als String eingegeben. Die **Abbildung 3.2** gibt hierzu ein Beispiel. Es wird für die Region München die Niederschlagsmenge für den heutigen Tag, die Windstärke für den heutigen und den nachfolgenden Tag, über einen Zeitraum von zwei Tagen abgerufen. Dabei beträgt die zeitliche Auflösung eine Stunde und das Intervall in denen Updates gestartet werden 6 Stunden. Der Datenabruf erfolgt eine halbe Stunde nach Funktionsausführung. 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 Variableninitialisierung

Als erstes wird der Default-Ordner *filepath* für das Abspeichern der Daten festgelegt. Dabei wird nachgesehen, ob lediglich eine Laufwerksangabe für den aktuellen MATLAB Ordner vorliegt oder nicht. 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 ausgegeben und die Funktion beendet. Stimmt die Anzahl, werden die Werte Funktionsvariablen zugeordnet. Möchte man den Zeitpunkt, zu dem der erste Datenabruf nach Ausführen der Funktion erfolgt, verschieben, so kann man einen zeitlichen Offset (Angabe als String in Stunden) als siebtes Inputargument übergeben. Wenn Startzeitpunkt des Beobachtungsintervalls und Funktionsausführung am gleichen Tag liegen, so bewirkt der Offset eine Verschiebung vom Ausführungszeitpunkt der Funktion an. Liegt das Startdatum jedoch in der Zukunft, so findet die Verschiebung von 0.00 Uhr des Startdatums an, statt. Ist zudem noch eine Pfadangabe zu einem Speicherort der Funktion übergeben worden, so wird diese ebenfalls einer Funktionsvariablen zugewiesen. Wurden keine Angaben zu den beiden Werten gemacht, so wird im aktuellen MATLAB Ordner der Default-Ordner „Aufzeichnungen“ als Speicherplatz definiert und der Offset auf Null gesetzt. Existiert der Ordner bereits, erfolgt eine Meldung auf deutsch. Wurden keine variablen Parameter übergeben (entspricht einem einmaligen sofortigen Abruf), so wird die zeitliche Auflösung auf 1 Stunde und der Offset auf Null festgesetzt und ebenfalls der Default-Speicherort vorgegeben.

```
% Assign varargin elements to variables. Create folder to save records.
if size(pwd,2) < 4
    filepath      = [pwd,'Aufzeichnungen'];
else
    filepath      = [pwd,'\Aufzeichnungen'];
end
if ~isempty(varargin)
    if nargin < 4
        fprintf(2,['Bitte geben Sie das Start- und Enddatum des' ...
                  'Beobachtungszeitraums\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 nargin == 7
        start_offset           = varargin{5}*3600;
        [s,mess,messid]        = mkdir(filepath);
        if ~isempty(mess)
            fprintf('Ordner existiert bereits.\n');
        end
    elseif nargin == 8
        start_offset           = varargin{5}*3600;
        filepath                = varargin{6};
        [s,mess,messid]        = mkdir(filepath);
        if ~isempty(mess)
            fprintf('Ordner existiert bereits.\n');
        end
    else
        [s,mess,messid]        = mkdir(filepath);
        if ~isempty(mess)
            fprintf('Ordner existiert bereits.\n');
        end
        start_offset = 0;
    end
end
else
    [s,mess,messid]          = mkdir(filepath);
    if ~isempty(mess)
        fprintf('Ordner existiert bereits.\n');
    end
    resolution = 1;
end
end

```

Nachdem die *forecast_definition* sowohl als String als auch als Cell-Array übergeben werden kann, im späteren Programmablauf aber nur ein Cell-Array erwartet wird, muss im Fall eines Strings eine Konvertierung zum Cell-Array 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** auf Seite 12 gezeigt, lautet die Slave-ID der Wetterstation „03“. Diese wird hier der *device_id* zugeordnet.

```

% 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);

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

% Set device id
device_id            = '03';

```

3.2.2 Aufbau von Strukturen

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 *fc_def* aufzubauen, welches alle Wetterdatenabfragen 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.

```

% 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);

```

3.2.3 Überprüfung der Eingabeparameter

Diese for-Schleife bearbeitet alle übergebenen Wetterdatenabfragen, prüft sie mit der Funktion *input_check* 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.

```

% 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', error_msg{:}, char(10))
        error('Die oben aufgefuehrten Eingabeparameter sind nicht korrekt.');
```

Die Funktion `input_check`, die nachfolgend erläutert wird, liefert als Outputparameter einen Vektor *val_inpt* der angibt, welche Parameter gültig sind. Zusätzlich werden die generierten Fehlermeldungen *err_msg*, die ID der Wetterregion *c_id* und der Breiten- und Längengrad ausgegeben. Im ersten Abschnitt des Input Checks wird die Funktion `get_city_id` aufgerufen, um die ID der Wetterregion, den Längen- und Breitengrad zu ermitteln. Längen- und Breitengrade wurden aus Google Earth übernommen [1]. Ist die ID nicht in der Liste zu finden, wird die entsprechende Vektorposition *val_inpt* auf Null gesetzt .

```

function[ val_inpt, err_msg, c_id, lng, lat ] = input_check( fc_def, city, varargin )
%Checks all input parameter if valid or not and displays wrong parameters
% Detailed explanation goes here
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
```

Betrachtet man die unten aufgeführte Funktion `get_city_id`, so 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, die dem Städtenamen entspricht, zu starten. In der Variable *city_data_set* sind nun alle

Werte dieser Listenzeile, enthalten. Eine einfache Wenn-Dann-Bedingung weist die Daten den Outputparametern zu.

```
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};
    lat = city_id_dataset{:,3};
    lng = city_id_dataset{:,4};
    city_id_correct = 1;
end
end
```

An dieser Stelle des Input-Checks 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.

```
% 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 separte 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

```

Die Überprüfung des Observationszeitraums erfolgt dahingehen, dass der Ausdruck ein Datumsformat darstellen muss, den MATLAB mittels **datetime** 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 nach dem Enddatum liegt. Auch hier werden entsprechende Fehlermeldungen generiert.

```

if ~isempty(varargin)
    try
        a=datetime(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=datetime(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
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.

```

% #### 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.
if true(val_inpt)
    err_msg = NaN;
end

end

```

3.2.4 Verfügbarkeitsprüfung der seriellen Schnittstelle

Existiert bereits eine Variable *serial_interface* im Base-Workspace, so wird diese gelöscht. Danach wird die Verfügbarkeit des COM6 Ports festgestellt. Ist der COM6 Port bereits in Benutzung, so wird dem Nutzer die Möglichkeit gegeben, diese Nutzung zu beenden oder nicht. Dies ist zum Beispiel der Fall, wenn eine serielle Schnittstelle auf dem COM6 Port existiert hat und nicht ordnungsgemäß gelöscht wurde.

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

% Check for available COM Ports, if COM6 is not available print message.
% If COM6 Port is already in use, delete it and make it available for the
% weather station.
av_com_ports = instrhwinfo('serial');
com_port_av = find(ismember(av_com_ports.AvailableSerialPorts, 'COM6'));
serial_ports_in_use = instrfind({'Port'}, {'COM6'});

if ~isempty(serial_ports_in_use)
    prompt = ['Der COM6 Port ist bereits in Benutzung. Wollen Sie ihn loeschen,\n'...
        ' um ihn fuer die Wetterstation frei zu bekommen?.\n'...
        'Moechten Sie fortfahren? Y/N [Y]: '];
    str = input(prompt, 's');
    if isempty(str)
        str = 'Y';
    end
    if strcmp(str, 'Y') == 1
        delete(instrfind({'Port'}, {'COM6'}));
        fprintf('Der COM6 Port steht nun zur Verfuegung.\n');
    else
        fprintf(2, 'Der Funktionsaufruf wurde abgebrochen.\n', char(10));
        return;
    end
elseif isempty(com_port_av)
    fprintf(2, 'COM6 Port ist nicht verfuegbar!\n', char(10));
    return;
end
```

3.3 Aufbau der seriellen Schnittstelle

Die Funktion `open_serial_port` enthält bereits alle in der **Tabelle 1.2** auf Seite 12 festgelegten Schnittstellenparameter.

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

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.

```
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'...
```

3.4 Abgleich der Wetterregion im Register

Da der Wechsel einer Wetterregion unter Umständen mehrere Stunden bis zu drei Tagen dauern kann, bis alle anliegenden Werte Gültigkeit besitzen, wird in diesem Teil des Codes zuerst die bereits eingetragene ID der Wetterregion mit der Funktion **read_com_set** ausgelesen und mit der in dem Funktionsaufruf Angegebenen verglichen. Die Funktionsweise von **read_com_set** kann im Anhang unter C.1 auf S. 74 nachvollzogen werden. Weichen die beiden Werte voneinander ab, so wird der Nutzer per Tastatureingabe aufgefordert dem Wetterregionenwechsel zuzustimmen. Er hat somit die Gelegenheit einen versehentlichen Wechsel abubrechen. Anschließend wird der neue Wert im Register eingetragen. Hierzu wird die Funktion **write_com_set** verwendet (siehe Anhang C.2 auf S. 75).

```
% Read actual city_id value in holding register
city_id_reg = read_com_set(device_id, {'city_id'}, 0);

% 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'}, 0 );
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'}, 0 );
        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
end

```

3.5 Festlegung der Timerparameter und Starten des Timers

Um die Eigenschaft des wiederholten Datenabrufs zu implementieren wurde das Timer-Objekt von MATLAB verwendet. Es bietet den Vorteil im Hintergrund zu laufen ohne dabei MATLAB komplett zu blockieren. Um dieses Objekt sinnvoll einsetzen zu können, müssen zuerst ein paar Parameter ermittelt werden. Insbesondere betrifft dies das Timerintervall, die Anzahl der Timerausführungen und die Timerverzögerung. Das Timerintervall *update_interval_hours* 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 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 plus einem evtl. Offset aufweisen. Bei der Berechnung dieser Angaben sind die MATLAB Funktionen **days365**, **datevec** sowie **etime** äußerst nützlich.

```

% If an observation interval is given as argument in the function, varargin
% is not empty. Otherwise only a single request will be generated.
if ~isempty(varargin)

% Calculate day difference in hours between the observation start date and
% end date. Further determine point in time of starting the function and
% the end 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 interval divided by the update interval. You have to add 1 for the
% first request executed immediately with a 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 plus offset.

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-1)*86400);
    update_cycle_number = floor(diff_days/update_interval);
    assignin('base','update_cycle_number',update_cycle_number);
end

% The waiting period for the timer: interval for an update times 3600 sec
update_interval_hours = update_interval*3600;

```

Mit der Zuweisung „ $t = \text{timer}$ “ 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 $t.StopFcn$ fest, was geschehen soll.

```

% A timer is defined here to control the automatic update cycles.
% Requests start with a specified offset from the moment the function was
% executed. If no offset was defined, it will be zero. For future start dates
% of the observation interval, the start_offset will be the time in seconds
% after midnight. 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 or a failure occurs during the timer function execution.
t = timer;

```

```

if strcmp(start_observation,date) == 1
    t.StartDelay          = start_offset;
else
    t.StartDelay          = start_delay+start_offset;
end
t.TimerFcn               = {@send_loop, size_table_data, fc_def, device_id,...
    filepath, city_name, update_cycle_number, resolution, longitude, latitude};
t.StopFcn                = {@stop_timer, filepath, city_name, resolution};
t.Period                 = update_interval_hours;
t.TasksToExecute         = update_cycle_number;
t.ExecutionMode          = 'fixedRate';
start(t);

else
    send_loop('','', size_table_data, fc_def, device_id, filepath, city_name, '',...
        resolution, longitude, latitude);
    filename          = strcat(filepath, '\weather_data_',date, '.mat');
    weather_data      = evalin('base', 'weather_data');
    save(filename, 'weather_data', '-mat');
    close_serial_port();
end

end

```

Kommt es zu einem Timerabbruch, wird die Funktion **stop_timer**, hier nachfolgend zu sehen, ausgeführt. Dabei wird eine Meldung an den Nutzer ausgegeben, das Timer-Objekt gelöscht und der Datencontainer *weather_data* 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. Der Abbruch erfolgt dabei entweder durch einen Fehler bei der Programmausführung oder durch bewusstes Löschen des Timerobjektes. Dieses Löschen kann durch die Funktion **stop_forecast_data**, die nur den Befehl **delete(timerfindall)** beinhaltet, herbeigeführt werden.

```

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

```

3.6 Abschicken der MODBUS-Anfragen

Tabelle 3.1: Inputparameter für die Funktion send_loop

<i>obj</i>	<i>event</i>	<i>t</i>	<i>fc_def</i>	<i>dev_id</i>	<i>f_path</i>
Timer-Objekt	Timer-Event	Anzahl der Wetterbereichsdefinitionen	Wetterbereichsdefinitionen	Id des MODBUS Slave	Pfad zum Speicherort
<i>c_id</i>	<i>u_c_n</i>	<i>res</i>	<i>lng</i>	<i>lat</i>	
ID der Wetterregion	Anzahl der noch verbleibenden Abrufvorgänge	gewünschte zeitliche Auflösung	Längengrad	Breitengrad	

Die Funktion **send_loop**, die das sequentielle Abschicken und Verarbeiten der einzelnen MODBUS Nachrichten übernimmt wird hier erläutert. Die Definition der Eingabeparameter kann der **Tabelle 3.1** entnommen werden. Nachdem die Eingabeparameter übergeben wurden, werden die im Base-Workspace vorhandenen Variablen *daychange_flag*, *daychange_counter* sowie der Datencontainer *w_dat* für die fortlaufenden Wetterdaten in dieser Funktion bereitgestellt. Eine for-Schleife durchläuft dann alle Wetterdatenabfragen 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 mit dem Befehl **assign** im Base-Workspace aktualisiert.

```

function[ ] = send_loop(obj, event, t, fc_def, dev_id, f_path, c_id, u_c_n, res,...
                        lng, lat )
%Starts the MODBUS message send loop
%   For every forecast definition in fc_def a modbus message is generated.
%   The function send_and_receive_data will transmit those message to the
%   serial interface.

% Initialize waiting bar
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
    cnt = 0;
    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
end

```

Im nächsten Schritt wird das Intervall *fc_int* für die auszulesenden Wetterdaten zusammengestellt. Hierbei muss unterschieden werden zwischen den stündlichen Werten der mittleren Lufttemperatur, 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.

```

% Build the forecast interval
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

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

```

In diesem Abschnitt wird die MODBUS Nachricht erstellt und der Funktion **send_and_receive_data** 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 wird im gleichen Verfahren ermittelt (vgl. C.4 auf S. 77 im Anhang). Die Differenz aus den beiden Adressdaten ergibt die Registeranzahl (vgl. C.5 auf S. 78 im Anhang). Gemäß der MODBUS Spezifikation liegen nun fast 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 (vgl. Kapitel 2.6 auf S. 18). Wie die momentane Verbindungsqualität *con_qual* zum Zeitpunkt der Anfrage ist, wird kurz darauf abgerufen. Ist in der Wetterbereichsdefinition auch die Temperatur mit aufgeführt, kann für einen später zu ermittelnden Korrekturfaktor die lokale Temperatur *lokal_temp* an der Wetterstation abgefragt

werden. Alle für die weitere Verarbeitung erforderlichen Daten werden nun an die Funktion `send_and_receive_data` übergeben.

```
% 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'},cnt);
if strcmp(fc_int{1,1}, 'temperatur') == 1
    lokal_temp = read_com_set('03',{'temperature'},cnt);
else
    lokal_temp = [];
end

% Write message on interface and read and process response
txdata = send_and_receive_data(modbus_pdu, fc_int,...
                               res, con_qual, lng, lat, cnt, lokal_temp);

waitbar(r/t,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. Es folgt darauf das Update der noch verbleibenden Anzahl an Datenabrufen, welche in der Variable `u_c_n` (update cycle number) hinterlegt sind. Zum Schluss wird der gesamte Base-Workspace für den Fall einer unbeabsichtigten Löschung zwischen den Abrufvorgängen gespeichert.

```
% Save requested data in a sepearte file
new_data = evalin('base','new_data');
filename = strcat(f_path,'\',c_id,'-',strrep(num2str(res),'.','_'),'_new_data-',...
                 date,'-',num2str(date2utc(datevec(now),MESZ_calc)),'.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','update_cycle_number');
```

```

        u_c_n = u_c_n-1;
        fprintf('Noch %u ausstehende Abfrage(n).\n',u_c_n)
        assignin('base','update_cycle_number',u_c_n);
    end
    evalin('base',sprintf('save(''%s'')', strcat(f_path,'\workspace')));
    % Close waiting bar
    close(h);
end

```

3.7 Senden und Empfangen

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 mit der Funktion **format_modbus_msg** auf ein geeignetes Format gebracht werden (vgl. C.6 auf S. 78 im Anhang). 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. Jetzt liegt die Antwortnachricht des Servers in der Variablen *rx_data* vor. Die darin enthaltenen Werte wurden als unsigned integer 8 bit empfangen.

```

function [ value ] = send_and_receive_data( modbus_msg, fieldname, res,...
                                           con_qual, lng, lat, cnt, lokal_temp )

%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] = rxdata_processing( rxdata, modbus_msg, fieldname, res,...

```

```
con_qual, lng, lat, cnt, lokal_temp );
```

```
end
```

3.8 Rx-Datenverarbeitung

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

Zu Beginn wird ein Flag für die while-Schleife initialisiert. Diese Schleife wird maximal dreimal durchlaufen. Die Kriterien, die zu einem Abbruch des Schleifendurchlaufs führen werden nachfolgend erklärt.

```
function [ response_data ] = ...
    rxdata_processing( rxdata, modbus_pdu, fc_def, res,...
        con_qual, lng, lat, cnt, lokal_temp )
%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.

% Initial value for the while loop
not_done = 1;
```

Wurde keine Nachricht empfangen, definiert man eine Fehlermeldung und setzt das CRC-Check Flag *crc_check_value* sowie den Funktionsoutput *response_data* auf Null. Diese Werte werden am Ende der Schleife geprüft und stellen das erste Kriterium für einen nochmaligen Durchlauf oder Abbruch dar. Sind Daten empfangen worden, wird der Funktionscode auf einen Fehlercode hin in der Funktion **fcode_check** kontrolliert (vgl. C.7 auf S. 79 im Anhang). Ein Fehlercode wird registriert, wenn der Funktionscode nicht den Wert 1, 3 oder 6 besitzt. Um welchen Ausnahmefall es sich dann handelt, wird in der Switch-Abfrage geklärt. Liegt einer dieser Fehlercodes vor, wird die dazu passende Fehlernachricht erstellt. 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. Liegen dagegen ausgelesene Daten aus dem Holdingregister vor, muss die Funktion **data_processing** aufgerufen werden. Egal welcher Fall auftritt, für jede

Antwortnachricht des Servers wird ein CRC mit der Funktion `crc.check` durchgeführt (vgl. C.16 auf S. 87 im Anhang). Ist die CRC Summe in Ordnung wird das CRC-Check Flag auf 1 gesetzt und keine Fehlermeldung zurückgegeben.

```
% The while loop will run three times in the case of no data was received
% an exception code was thrown or crc check failed.
while not_done == 1
    if isempty(rxdata)
        error_msg = ('No data received! Check if server is available!');
        response_data = [];
        crc_check_value = 0;
    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_msg = ('Exception Code 01 -> Function code not supported');
                case '02'
                    error_msg = ('Exception Code 02 -> Output address not valid');
                case '03'
                    error_msg = (['Exception Code 03 -> Quantity of outputs exceeds'...
                        'range 0x0001 and 0x07D0']);
                case '04'
                    error_msg = (['Exception Code 04 -> Failure during reading discret'...
                        'outputs']);
                otherwise
                    error_msg = ['Unnown exception code: ' exception_code];
            end
            response_data = [];
            crc_check_value = 0;
        else
            error_msg = [];
            switch rxdata(2)
                case 1
                    response_data = rxdata(4);
                    [crc_check_value, error_msg] = crc_check(rxdata);
                case 3
                    response_data = data_processing(rxdata(4:end-2), fc_def,...
                        res, con_qual, lng, lat, lokal_temp);
                    [crc_check_value, error_msg] = crc_check(rxdata);
                case 6
                    response_data = dec2hex(rxdata(5:6),4);
                    [crc_check_value, error_msg] = crc_check(rxdata);
            end
        end
    end
end
```

Die Kriterien für die while-Schleife sind nun komplett. Ein erneuter Schleifendurchlauf wird dann angestoßen, wenn die CRC Summe fehlerhaft war, keine Daten gesendet wurden oder ein Fehlercode vorliegt und eine entsprechende Fehlermeldung generiert wurde.

```

if (isempty(response_data) || isempty(crc_check_value)) && ~isempty(error_msg)
    not_done = 1;
    cnt = cnt + 1;
    if cnt == 3
        error(error_msg);
    else
        send_and_receive_data(modbus_pdu, fc_def, res, con_qual,...
                               lng, lat, cnt, lokal_temp);
    end
else
    not_done = 0;
end

end
end

```

3.9 Datenverarbeitung

Ein weiterer wichtiger Baustein dieses Programms ist die Datenverarbeitung, die mit der Funktion **data_processing** gestartet wird. Neben den schon bekannten Parametern müssen hier noch die empfangenen Datenbytes, im *data_string* hinterlegt, übergeben werden. Da die Funktion auf Variablen aus dem Base-Workspace zurückgreift, muss man diese zuerst in den Funktionsworkspace von **data_processing** laden. Danach liefert die Funktion **MESZ_calc** ein Flag für die Mitteleuropäische Sommerzeit (vgl. C.8 auf S. 79 im Anhang). Im Anschluss werden die Listen für einen späteren Datenabgleich erstellt.

```

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

% Processes the rxdata and allocates it to the data container in a defined
% structure
% Detailed explanation goes here
% Get the new data and weather data container from 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();

% 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'};

```

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.

```

% 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

```

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 und der neue Satz Daten, die Prognosewerte überschreibt. *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, muss mit der Funktion **res_factor** der entsprechende Faktor für die spätere Interpolation bestimmt werden (vgl. C.9 auf S. 80 im Anhang). Hat ein Abruf bereits 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.

```

else
    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
% encounter a daychange.

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

% sindex = starting point of the loop through the observation days
% edindex = end point
[~, sindex] = ismember(fc_def{3}, obs_day);
[~, edindex] = ismember(fc_def{5}, obs_day);

```

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. Liegt nach dem erstmaligen Funktionsaufruf kein Datumswechsel vor aber ein zweites Update steht am selben Tag an, dann müssen diese Variablen noch immer den Wert eins aufweisen. Hierdurch werden die Daten durch „Intraday-Updates“ überschrieben. Die Bedingung für die konstante Position ergibt sich aus der Differenz des letzten Rekordstempel mit dem aktuellen Datum. Hierfür wird der Unix Zeitstempel in das normale Datumsformat „dd-mmm-yyyy“ mit der Funktion **utc2date** und dem Befehl **datestr** konvertiert (vgl. C.10 auf S. 81 im Anhang). Tritt ein Datumswechsel auf, bestimmt die while-Schleife diese Position im Datensatz. Dazu wird die in der Variable *unix_t_strt* gespeicherte Zeitreihe durchlaufen bis das Datum der alten Zeitreihe dem dem Startdatum der neuen Datenreihe 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.

```

% 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

% 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.
    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 position
% 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
    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.
    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
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 Schleifenposition 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.

```

% 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
        end
    end

```

```

        end

    end

    if t == sdindex
        datepart      = str2double(regexpi(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
end

```

Es beginnt nun der Schleifendurchlauf für die Tagessegmente und damit die Auswertung des Datenstrings. Jeder Wert für ein Tagessegment ist dabei aus einem High- und Low-Byte in dezimalen Format aufgebaut und muss deshalb zuerst in ein hex-Format und nach Zusammensetzung wieder in ein Dezimalformat konvertiert werden. Für die Unix-Zeitstempelberechnung wird ein Datumsvektor benötigt, der in der Funktion **tvector** für die Wetterdaten erstellt wird (vgl. C.11 auf S. 82 im Anhang). Der Vektor gibt jeweils den Beginn des Intervalls an, für den der bearbeitete Wert gültig ist. 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 1 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.

```

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});
    end
end

```

```

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 ~= 6
    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

```

In diesem Abschnitt werden die Werte dem Datencontainer zugewiesen. Es sind dabei folgende Variablen zu belegen:

- *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
- *loc_temp* nur für die Daten im Temperaturbereich

Die Transformation in das Unixzeitformat übernimmt die Funktion **date2utc** (vgl. C.12 auf S. 84 im Anhang). 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` (vgl. C.13 auf S. 84 im Anhang). Am Schluss eines Schleifendurchlaufs müssen die Vektorpositionen des Datenstrings und der Zeitreihen angepasst bzw. inkrementiert werden.

```
% Assign received value to continous data container. No interpolation is done.
% Unix time interval start
    w_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(w_dat_r) =...
        date2utc(timevec,MESZ_calc);

% Unix time interval end
    w_dat.(fc_def{1}).(fc_def{2}).unix_t_end(w_dat_r) =...
        date2utc(timevec,MESZ_calc) + timestep;

% Unix time interval mean
    w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(w_dat_r) =...
        date2utc(timevec,MESZ_calc) + floor(timestep/2);

% Unix time record time
    w_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(w_dat_r) =...
        date2utc(datevec(now),MESZ_calc);

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

% Weather data int value (not interpolated at this time) and org value
    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});

% Connection quality
    w_dat.(fc_def{1}).(fc_def{2}).con_qual(w_dat_r_org) = con_qual;

% Local temperature
    if strcmp(fc_def{1},'temperatur') == 1
        w_dat.(fc_def{1}).(fc_def{2}).loc_temp(w_dat_r_org) = lokal_temp/10;
    end

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

% Assign received value to update data container. No interpolation is done.
% Unix time interval start
    n_dat.(fc_def{1}).(fc_def{2}).unix_t_strt(n_dat_r) =...
        date2utc(timevec,MESZ_calc);

% Unix time interval end
    n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(n_dat_r) =...
        date2utc(timevec,MESZ_calc) + timestep;

% Unix time interval mean
    n_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(n_dat_r) =...
```

```

                                date2utc(datevec(now),MESZ_calc);
% Unix time record time
    n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(n_dat_r) =...
                                date2utc(timevec,MESZ_calc) + floor(timestep/2);
% Clear time interval
    n_dat.(fc_def{1}).(fc_def{2}).interval_t_clr{n_dat_r} =...
    {[cell2mat(utc2date(date2utc(timevec,MESZ_calc))), '-',...
    datestr(utc2date(n_dat.(fc_def{1}).(fc_def{2}).unix_t_end(n_dat_r)),13)]};
% Weather data int value (not interpolated at this time) and org value
    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});
% Connection quality
    n_dat.(fc_def{1}).(fc_def{2}).con_qual(n_dat_r) = con_qual;
% Local temperature
    if strcmp(fc_def{1},'temperatur') == 1

```

3.9.1 Interpolation

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.

```

% INTERPOLATION

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

Wird eine andere Auflösung als die Originale gefordert, so können die Werte (Windrichtung,
Markantes Wetter, Signifikantes Wetter und Niederschlagswahrscheinlichkeit), die nicht interpoliert werden gleich im Base-Workspace gespeichert werden.

    else

% 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);

```

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 Engineering“ entnommen (vgl. C.14 auf S. 85 im Anhang). Die Messwerte der Wetterstation werden dann auf vier gleiche Zeitintervalle von Sonnenauf- bis Sonnenuntergang aufgeteilt.

```

else

% 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
    if edindex == 1
        [sun_rise_today, sun_set_today] = diurnal_var(lng, lat, date);
        sun_rise_today = double(date2utc(sun_rise_today, MEZ));
        sun_set_today = double(date2utc(sun_set_today, MEZ));
        diurnal_cont = [sun_rise_today, sun_set_today];
        tmp_dat_x = linspace(sun_rise_today, sun_set_today, 4);
    else
        [sun_rise_today, sun_set_today] = diurnal_var(lng, lat, date);
        [sun_rise_tomorrow, sun_set_tomorrow] = diurnal_var(lng, lat, ...
            datestr(datetime(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));
        diurnal_cont = [sun_rise_today, sun_set_today, sun_rise_tomorrow, ...
            sun_set_tomorrow];
        tmp_dat_x = [linspace(sun_rise_today, sun_set_today, 4), ...
            linspace(sun_rise_tomorrow, sun_set_tomorrow, 4)];
    end
else
    tmp_dat_x = double(n_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,1:end));
    diurnal_cont = [];
end

```

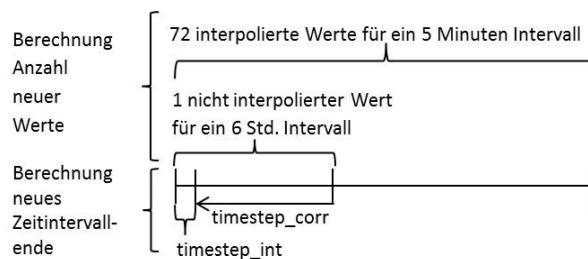


Abbildung 3.3: Neuberechnung Intervallanzahl und -startwerte

Wenn noch kein Datenabruf vorher stattgefunden hat, wird das Ende des Datensatzes durch die Vektorlänge des mittleren Zeitreihenintervalls *unix_t_mean* multipliziert mit dem Faktor für die Auflösung 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 Endzeitpunkte des Gültigkeitsintervalls neu berechnet, indem die alten Intervallgrenzen um die Korrekturfaktoren *timestep_corr* verschoben

werden. Auch die Intervallmitte *unix_t_mean* und die Angabe der Intervallgrenzen im normalen Zeitformat werden neu berechnet. Die **Abbildung 3.3** soll den Vorgang etwas verdeutlichen.

```
% 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.
% Correct interval limit for unix time interval end
    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;
% Correct interval limit for unix time interval mean
    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);
% Correct interval limit for clear time interval
    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)]}];
% Correct interval limit for unix time interval end
    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;
% Correct interval limit for unix time interval mean
    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);
% Correct interval limit for clear time interval
    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)]}];
```


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

```
% Adjust all following intervals

% Build new start interval beginning at the daychange until the last value of the new data
% plus timestep_corr
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);
% Build new end interval beginning at the daychange until the last value of the new data
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);
% Build new mean interval beginning at the daychange until the last value of the new data
% plus timestep_corr/2
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));
% Calculate time record
w_dat.(fc_def{1}).(fc_def{2}).unix_t_rec(i:data_end) =...
date2utc(datevec(now),MESZ_calc);
% Build new clear time interval
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)))';

% Calculate time record
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);
% Build new start interval beginning at the daychange until the last value of the new data
% plus timestep_corr
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);
% Build new end interval beginning at the daychange until the last value of the new data
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);
% Build new mean interval beginning at the daychange until the last value of the new data
% plus timestep_corr/2
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));
% Build new clear time interval
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)))';

```

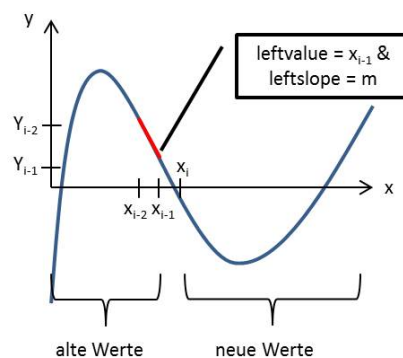


Abbildung 3.4: Parameterbestimmung für die slm-Funktion

In diesem Teil des Programms erfolgt nun die Interpolation der Ausgangswerte. Dabei werden alle für die Interpolation vorgesehenen Bereiche mit dem „shape language modeling“ [13] interpoliert. Es besteht die Möglichkeit in dieser Funktion die Anzahl der Spline-Knoten festzulegen. Für die mittlere Lufttemperatur wurde die Knotenzahl von 96 möglichen (96 Werte liegen vor) auf 48 halbiert. Hierdurch werden zwar nicht alle existierenden Werte korrekt durchlaufen, dafür ist die Kurve im Vergleich zu 96 Knoten wesentlich glatter im Verlauf (siehe **Abbildung 3.5**). Für einen erstmaligen Aufruf wird die linke und rechte Steigung des Graphen mit den Einstellungsparametern *leftslope* und *rightslope* auf 0 festgelegt. Liegen indes schon Daten vor, so wird die linksseitige Steigung der letzten beiden vorangegangenen Graphenpunkte bestimmt

$$m = \frac{y_{i-1} - y_{i-2}}{x_{i-1} - x_{i-2}} \quad (3.1)$$

und als Bedingung der **slm** Funktion für den linksseitigen Steigungswert vorgegeben (vgl. **Abbildung 3.4**). Zusätzlich muss der neue Graph mit dem letzten Graphenwert der vorigen Daten beginnen. So soll ein fließender Übergang der Graphen sichergestellt werden. Die Funktion **neg_val_corr** hat zur Aufgabe, negative Werte für Wetterbereiche, die keine negativen Werte aufweisen können auf Null zu setzen (vgl. C.13 auf S. 84 im Anhang). Hierzu zählen unter anderem die Solarstrahlung und Sonnenscheindauer. Für diese müssen zusätzlich durch die Interpolation außerhalb des Sonnenauf- und -untergangs entstandenen positive Werte ebenfalls auf Null gesetzt werden.

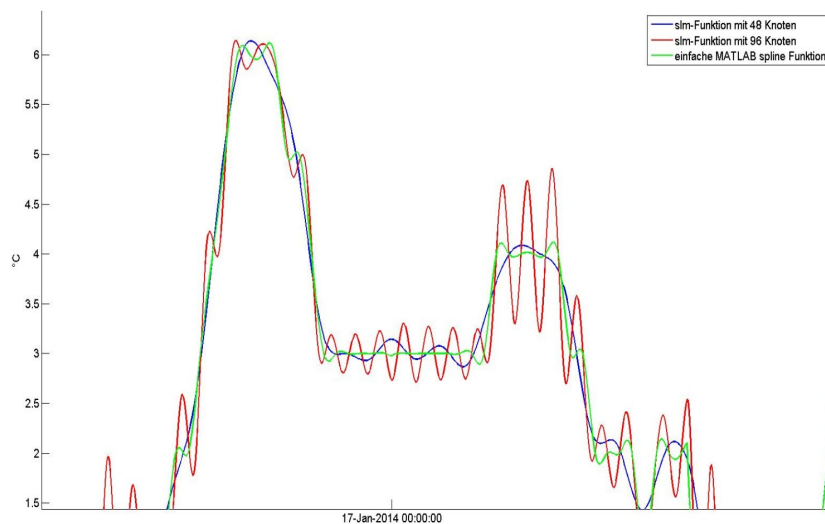


Abbildung 3.5: Vergleich der Interpolationsfunktionen

```
% Perform the slm interpolation for temperatur, staerke and luftdruck
if strcmp(fc_def{2}, 'staerke') == 1 || ...
    strcmp(fc_def{2}, 'menge') == 1 || ...
    strcmp(fc_def{2}, 'min') == 1 || ...
    strcmp(fc_def{2}, 'max') == 1
    knoten = 16;
elseif strcmp(fc_def{2}, 'mittlere.temp.prog') == 1
    knoten = 48;
else
    knoten = 8;
end

if i == 1
    slm = slmengine(tmp_dat.x, tmp_dat.y, 'plot', 'off', 'knots', knoten, ...
        '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);
    if strcmp(fc_def{1}, 'solarleistung') == 1 || ...
        strcmp(fc_def{2}, 'menge') == 1 || ...
        strcmp(fc_def{2}, 'staerke') == 1
        corr_val = neg_val_corr(fc_def{2}, edindex, diurnal_cont, ...
            w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1, i: data_end), ...
            w_dat.(fc_def{1}).(fc_def{2}).int_val(1, i: data_end));
        w_dat.(fc_def{1}).(fc_def{2}).int_val(1, i: data_end) = corr_val;
    end
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', knoten, '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 || ...
    strcmp(fc_def{2}, 'menge') == 1 || ...
    strcmp(fc_def{2}, 'staerke') == 1
    corr_val = neg_val_corr(fc_def{2}, edindex, diurnal_cont, ...
        w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(1,i-1:data_end), ...
        w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1:data_end));
w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i-1:data_end) = corr_val;
end
end

```

Für die Daten, die nicht in einer fortlaufenden Zeitreihe gespeichert werden, sind die linken und rechten Steigungen auf 0 gesetzt.

```

slm_new = slmengine(tmp_dat_x, tmp_dat_y, 'plot', 'off', 'knots', knoten, ...
    '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);
if strcmp(fc_def{1}, 'solarleistung') == 1 || ...
    strcmp(fc_def{2}, 'menge') == 1 || ...
    strcmp(fc_def{2}, 'staerke') == 1
    corr_val = neg_val_corr(fc_def{2}, edindex, diurnal_cont, ...
        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)), ...
            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))));
n_dat.(fc_def{1}).(fc_def{2}).int_val(1,i:data_end) = corr_val;
end

```

Möchte man doch alle originalen Messwerte der Wetterstation im Graphen berücksichtigt haben, so kann hier die normale MATLAB-Funktion **spline** verwendet werden. Hierbei wird immer der letzte Wert des vorigen Abrufs mit in die Berechnung einbezogen. Dazu kommentiert man den Bereich mit der **slm**-Funktion und unkommentiert diesen Bereich. Auch hier wird die Funktion **neg_val_corr** angewandt. Zum Abschluss werden die beiden Datencontainer dem Base-Workspace zugewiesen.

```

%         if i == 1
%             yi = spline(tmp_dat_x,tmp_dat_y,...
%                 w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i:end));
%             if strcmp(fc_def{1},'solarleistung') == 1 || ...
%                 strcmp(fc_def{2},'menge') == 1 || ...
%                 strcmp(fc_def{2},'staerke') == 1
%                 corr_val = neg_val_corr(fc_def{2},edindex,diurnal_cont,...
%                     w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i:end),...
%                     yi);
%                 w_dat.(fc_def{1}).(fc_def{2}).int_val(1,i:end) = corr_val;
%             end
%         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));
%             if strcmp(fc_def{1},'solarleistung') == 1 || ...
%                 strcmp(fc_def{2},'menge') == 1 || ...
%                 strcmp(fc_def{2},'staerke') == 1
%                 corr_val = neg_val_corr(fc_def{2},edindex,diurnal_cont,...
%                     w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i-1:end),...
%                     yi);
%             w_dat.(fc_def{1}).(fc_def{2}).int_val(1,(i-1):(data_end)) = corr_val;
%             end
%         end
%
%     yi_new = spline(tmp_dat_x,tmp_dat_y,...
%         w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i:end));
%
%     if strcmp(fc_def{1},'solarleistung') == 1 || ...
%         strcmp(fc_def{2},'menge') == 1 || ...
%         strcmp(fc_def{2},'staerke') == 1
%         corr_val = neg_val_corr(fc_def{2},edindex,diurnal_cont,...
%             w_dat.(fc_def{1}).(fc_def{2}).unix_t_mean(i:end),...
%             yi_new);
%
%         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)) = corr_val;
%
%     end
%
%     assignin('base','new_data',n_dat);
%     assignin('base','weather_data',w_dat);
%
% end
%
end
%
end

```

Kapitel 4

Datenanalyse

In der Datenanalyse soll nun ein Vergleich der interpolierten Daten aus der Wetterstation mit den Wetterdaten des meteorologischen Instituts der LMU angestellt werden. Hierbei kann auch die Implementierung der Sonnenauf- und Sonnenuntergangsadaptation erläutert werden. Um die Prognosegüte der interpolierten Daten zu bewerten soll der Root Mean Square Error Anwendung finden. Die Farbmarkierung der nachfolgenden Tabellen definiert sich folgendermaßen. Rot steht für den aktuellen Tag, grün für den ersten Folgetag, gelb für den zweiten und weiß für den dritten Folgetag. Die Berechnung der RMSE-Werte erfolgt dabei wie folgt [14].

$$RMSE = \sqrt{\frac{\sum_{t=s}^S (Y_{LMU_t} - Y_{HKW_t})^2}{S}} \quad (4.1)$$

mit:

- S – Anzahl der Messpunkte innerhalb eines Tages,
- s – Prognosezeitpunkt,
- Y_{HKW_t} – HKW Wetterdaten zum Prognosezeitpunkt t ,
- Y_{LMU_t} – LMU Wetterdaten zum Messzeitpunkt t .

4.1 Sonnenscheindauer

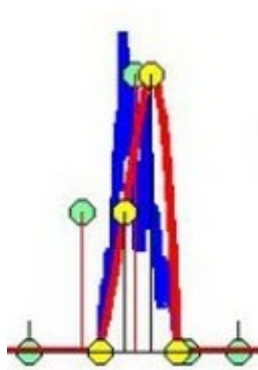


Abbildung 4.1: Ergebnis der Sonnenaufgangs- und Sonnenuntergangsadaptation

An dieser Stelle sieht man zunächst das Ergebnis der Sonnenauf- und Sonnenuntergangsadaptation. Wie in der **Abbildung 4.1** zu erkennen, würde ohne diese Adaption die Spline Kurve durch die grünlich markierten Punkte verlaufen. Wobei der erste und letzte Punkt, den Tagesbeginn 0.00 Uhr bzw. das Tagesende 23.59 Uhr markieren. Das hieße aber, dass es Globalstrahlung den ganzen Tag über geben würde. Die von der Wetterstation gelieferten Werte müssen daher auf den Zeitraum von Sonnenaufgang- bis Sonnenuntergang, hier als gelbe Punkte auf der x-Achse dargestellt, gemappt werden. Das Mapping der Werte geschieht dabei in gleichen

Abständen.

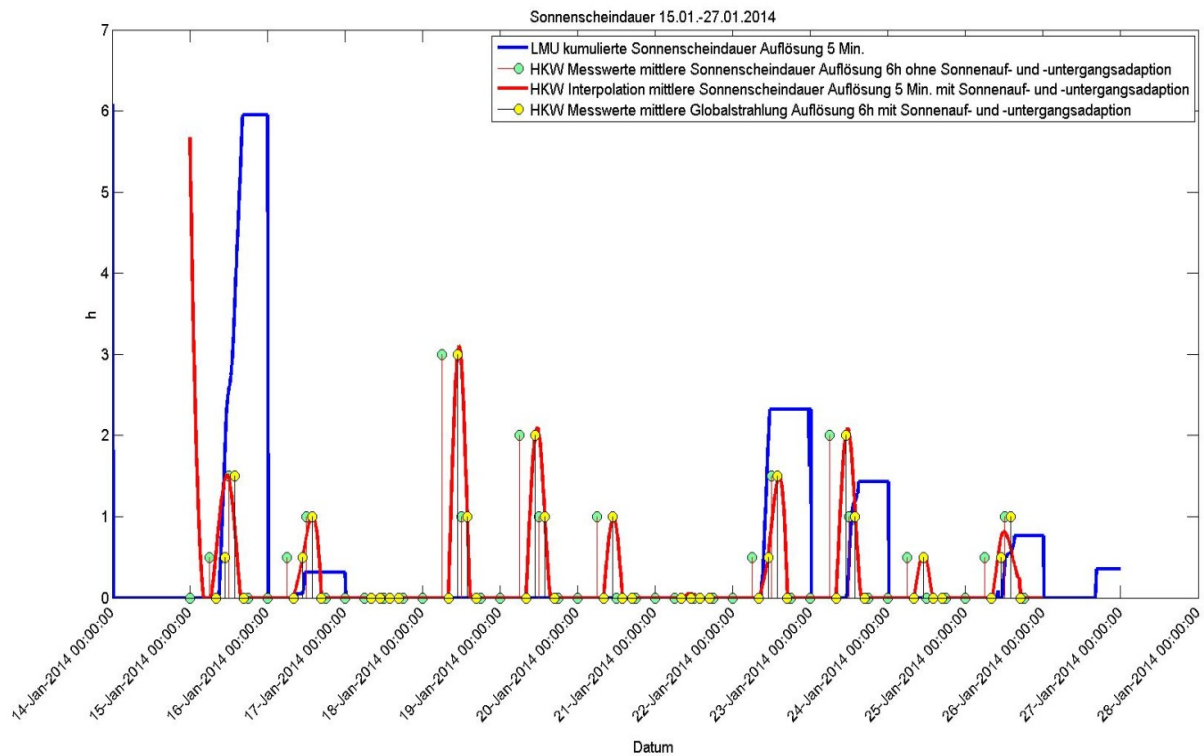


Abbildung 4.2: Vergleich der interpolierten Sonnenscheindauer mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014

Bei der Sonnenscheindauer gibt es Diskrepanzen zwischen den Daten der LMU und denen der Wetterstation. Während die LMU keinen Sonnenschein ausweist, registriert die Wetterstation beispielsweise vom 18.01. bis einschließlich 20.01.2014 positive Werte. Dies kann an der Definition des Sonnenscheins liegen. In der Spezifikation der Station wird dann von Sonnenschein gesprochen, wenn die Globalstrahlung mehr als 120W/m^2 beträgt. Nach Rücksprache mit Herrn Lösslein von der LMU, wird bei den LMU Daten ebenfalls dieses Kriterium angewandt [15]. Wie im Gespräch mit Herrn Volkhardt von der Firma HKW GmbH zu ermitteln war, gibt es die Möglichkeit, solche Abweichungen beim Qualitätsmanagement des Wetterdienstes über die Firma HKW einzureichen [16]. Da die Werte der LMU kumuliert sind, die der Wetterstation jedoch nicht, wird hier zum Vergleich der Prognosegüte nicht der RMSE verwendet. Es wird lediglich über die MATLAB Funktion `trapz` die Fläche unter den Spline-Kurven angenähert und dann mit dem maximalen Wert der LMU Daten verglichen. Hierbei fällt auf, dass die Prognose vom 22.01. näher an den realen Daten lag, als der dann tatsächlich vorliegende Wert der Wetterstation an diesem Tag. Für ein besseres Bild der Prognosegüte müssen mehr Daten ausgewertet werden.

Tabelle 4.1: Absolute Abweichung der Sonnenscheindauer von den LMU Wetterdaten in Stunden, Beobachtungsintervall 15.-24.01.2014

Abrufdatum Intervall 18.00-24.00 Uhr	16.01.	17.01.	18.01.	19.01.	20.01.	21.01.	22.01.	23.01.
16.01.2014	2.66	0.11						
17.01.2014		0.03	8.22					
18.01.2014			7.45	6.04				
19.01.2014				5.49	2.54			
20.01.2014					2.18	0.08		
21.01.2014						0.08	1.12	
22.01.2014							1.83	4.21
prozentuale Abweichung der realen HKW Werte von der Prognose		-73.0	-9.4	-9.1	-14.1	0.0	+63.0	

4.2 Globalstrahlung

Beim Vergleich der Globalstrahlung in **Abbildung 4.3** bietet sich ein geteiltes Bild. Im Bereich niedriger Globalstrahlung ($< 200 \text{ W/m}^2$) befindet sich der RMSE auf einem recht akzeptablen Niveau (siehe **Tabelle 4.2**). Dies ändert sich jedoch bei großen Ausschlägen nach oben, wie sie am 22. und 23.01.2014 zu beobachten sind. Betrachtet man den 23.01. genauer, so lagen die Werte bei 75 und 100 W/m^2 für das Intervall von 6 Uhr bis 12 Uhr bzw. 12 bis 18 Uhr. Der Sonnenaufgang und der -untergang für diese Jahreszeit findet gegen 8 Uhr und 16.45 Uhr statt. D.h. ca. 250 W/m^2 fehlen dem eigentlich relevanten Intervall in dem die Sonne scheint. Verteilt man diese Energie auf die verbleibenden 9 Stunden, würde das die Kurve um ca. 28 W/m^2 nach oben verschieben. Die Daten der LMU erreichen an diesem Tag Spitzenwerte bis über 400 W/m^2 . Die Werte fallen auch nicht nennenswert unter 100 W/m^2 , so dass davon ausgegangen werden kann, dass der mittlere Wert der LMU Daten höher liegt. Auch hier müsste man evtl. die Daten dem Anbieter der Wetterstation übergeben und vom Wetterdienst überprüfen lassen.

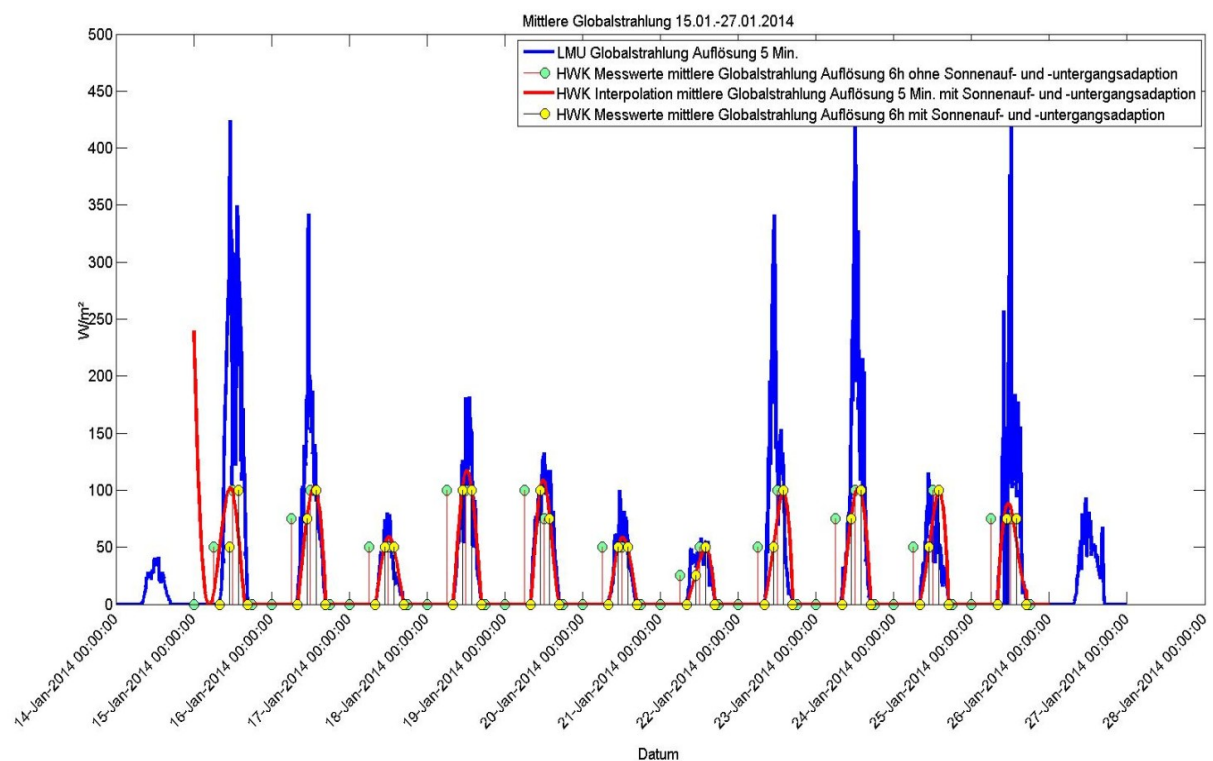


Abbildung 4.3: Vergleich der interpolierten Globalstrahlung mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014

Tabelle 4.2: RMSE der Globalstrahlung in W/m^2

Abrufdatum Intervall 18.00-24.00 Uhr	16.01.	17.01.	18.01.	19.01.	20.01.	21.01.	22.01.	23.01.
16.01.2014	38.11	5.10						
17.01.2014		6.78	22.18					
18.01.2014			16.37	13.22				
19.01.2014				11.27	7.47			
20.01.2014					7.36	12.09		
21.01.2014						7.97	49.76	
22.01.2014							59.81	68.55
prozentuale Abweichung der realen HKW Werte von der Prognose		+32.9	-26.2	-14.8	-1.5	-34.0	+20.2	

4.3 Niederschlagsmenge

Die Niederschlagsmenge wird ebenfalls wie zuvor schon die Solardaten mit einer 6 stündigen Auflösung bereit gestellt. Die Daten der LMU liegen in einer 1 Minuten Auflösung vor. Um die beiden Daten miteinander vergleichbar machen zu können, musste der HKW Wert mit einer Division durch 72 auf ein 5 Minuten Intervall gebracht und die LMU Daten auf 5 Min. gemittelt werden. Im Gegensatz zur Globalstrahlung liegen hier die mittleren Werte der Wetterstation meist über denen der LMU (siehe **Abbildung 4.4**). Betrachtet man den Prognoseverlauf vom 18.01. bis zum aktuellen Wert am 22.01. so erwartet man eine stetige Verbesserung der Werte, da weiter in der Zukunft liegende Prognosen unsicherer sind. Das ist aber nicht der Fall (vgl. **Tabelle 4.3**). Ob sich das auch über einen längeren Zeitraum hinzieht müsste mit einem größeren Datensatz überprüft werden.

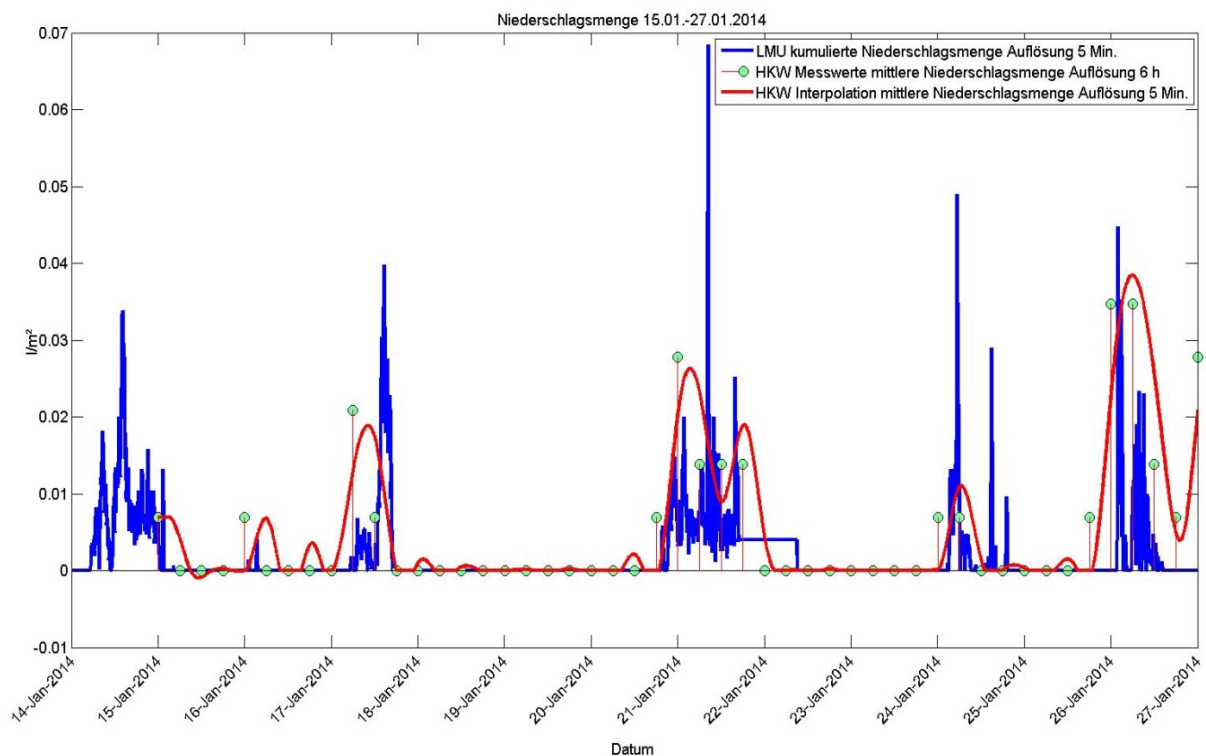


Abbildung 4.4: Vergleich der interpolierten Niederschlagsmenge mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014

Tabelle 4.3: RMSE der Niederschlagsmenge in l/m²

Abrufdatum Intervall 18.00-24.00 Uhr	16.01.	17.01.	18.01.	19.01.	20.01.	21.01.	22.01.	23.01.	24.01.
16.01.2014	0.22	0.43	0.04	0.00					
17.01.2014		0.76	0.04	0.01	0.70				
18.01.2014			0.04	0.03	0.82	1.14			
19.01.2014				0.01	0.27	0.98	0.27		
21.01.2014					0.38	1.40	0.07	0.23	
22.01.2014						1.27	0.05	0.03	0.66
prozentuale Abweichung der realen HKW Werte von der Prognose		+76.7	0.0 0.0	- +300 -66.7	+17.1 -67.1 +40.7	-14.3 +42.9 -9.3	-74.1 -28.6	-87.0	

4.4 Windstärke

Die Windstärkedaten der LMU sind mit einer Auflösung von einer Minute äußerst genau, was sich in den häufigen Ausschlägen widerspiegelt. Dennoch folgt der Graph der interpolierten Daten in der **Abbildung 4.5** relativ gut dem Gesamtverlauf. Da die Windstärke in einer Höhe von 30m an der LMU gemessen wird, lassen sich evtl. die Abweichungen von einem Bft erklären. Die Daten der Wetterstation beziehen sich auf 10m über dem Erdboden.

Geht man zum Beispiel von einer Stärke von 3 Bft auf 10m Höhe aus, dann kann man über das logarithmische Grenzschichtprofil die Windstärke in 30m Höhe abschätzen. Die Formel hierzu lautet: $v(30m) = v(10m) * \frac{\ln(\frac{30m-0}{2})}{\ln(\frac{10m-0}{2})}$. Mit einer Geländeklasse, die einem Stadtkern entspricht, ergibt sich bei einer Geschwindigkeit von 4.4m/s für $v(10m)$ und keinem Versatz der Grenzschicht durch Hindernisse ein Wert von 7.4m/s, der ungefähr einem Bft mehr entspricht. [17]

Wie gut nun die Prognosewerte sind, darüber lässt sich wie vorhin erwähnt eine Aussage mit dem Root Mean Squared Error (RMSE) treffen [18]. Dieser stellt die Standardabweichung der Differenzen zwischen Prognose- und Beobachtungswert dar. Der Deutsche Wetterdienst empfiehlt in seiner Broschüre keine größere Abweichung als 1 Bft für eine gute Prognose [18]. Die nachfolgende **Tabelle 4.4** stellt jeweils den RMSE für unterschiedliche Abrufdaten dar. Wie man in der Tabelle erkennen kann, wird dieses Qualitätsmerkmal weitgehend eingehalten.

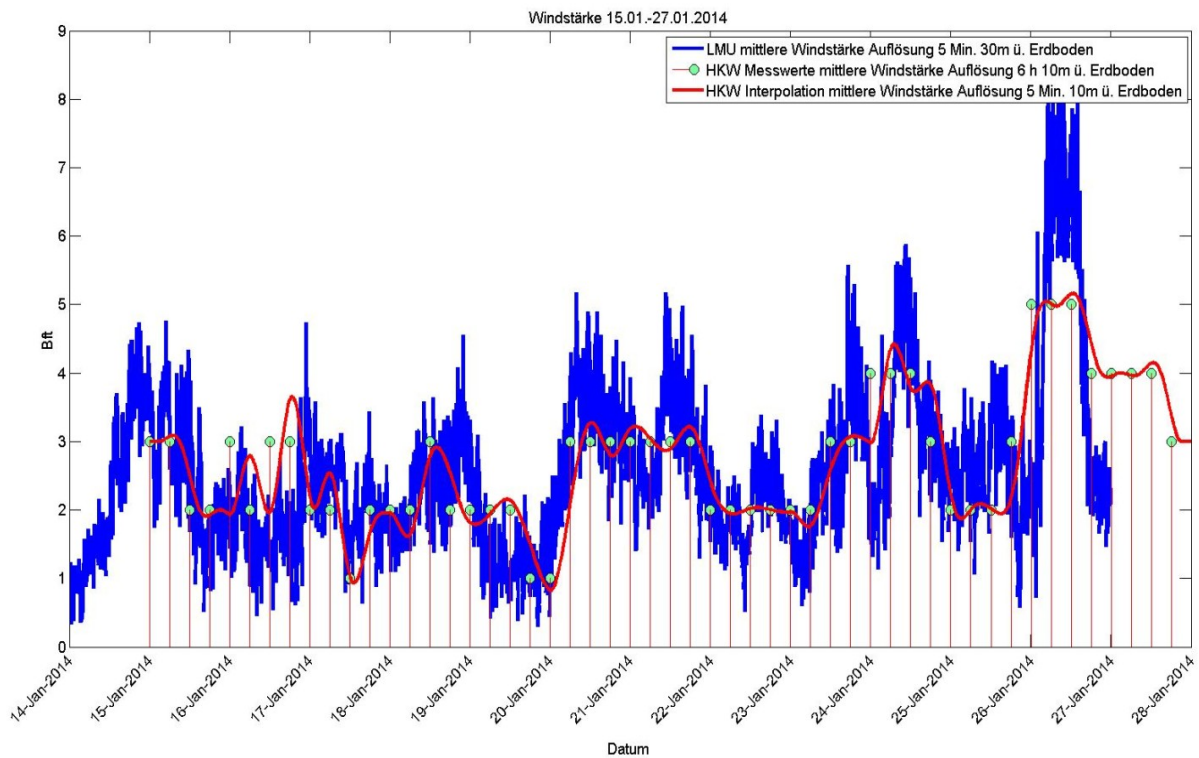


Abbildung 4.5: Vergleich der interpolierten Windstaerke mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014

Tabelle 4.4: RMSE der Windstärke in Bft

Abrufdatum Intervall 18.00-24.00 Uhr	16.01.	17.01.	18.01.	19.01.	20.01.	21.01.	22.01.	23.01.	24.01.
16.01.2014	1.23	0.66	0.81	0.76					
17.01.2014		0.72	0.87	0.55	0.91				
18.01.2014			0.78	0.72	1.32	0.91			
19.01.2014				0.77	1.37	0.89	0.61		
21.01.2014					1.07	0.91	0.65	0.94	
22.01.2014						0.92	0.62	0.94	1.05
prozentuale Abweichung der realen HKW Werte von der Prognose		+9.1	+7.4 -10.3	-27.6 +30.9 +6.9	+45.1 +3.8 -21.9	-2.2 +2.2 +1.1	+6.6 -4.6	0.0	

4.5 Mittlere Lufttemperatur

Die mittlere Lufttemperatur wird mit einer 1 stündigen Auflösung an die Wetterstation übertragen. Es ist deshalb davon auszugehen, dass diese Daten verglichen mit den Daten der LMU weniger große Diskrepanzen aufweisen. Betrachtet man die **Abbildung 4.6** so vermittelt diese, einen die vorige Aussage bestätigenden Eindruck. Auffallend jedoch ist der teilweise Versatz der Datenpunkte nach unten. Ein Blick auf den RMSE in der **Tabelle 4.5** zeigt, dass es sich hierbei meist um 1 bis 2 °C bei den tagesaktuellen Daten und 1 bis 3.5 °C bei den Prognosedaten handelt. Ein Grund für diese eher einseitige Abweichung nach unten könnte ein unterschiedlicher Standort der beiden Wetterstationen sein. Die LMU Wetterstation befindet sich direkt im Stadtzentrum. Bewegt man sich hin zu den Randbezirken Münchens, kann die Temperatur um bis zu 6 °C abfallen [19]. Auch hier ist der Verlauf der Prognoseentwicklung nicht immer der, den man erwarten würde. Nimmt man zum Beispiel den 19. Januar so war die Prognose am 16.01.2014 für diesen Tag besser als die beiden Darauf folgenden, die weniger weit in die Zukunft reichen. Hinsichtlich der Prognosegüte kann man unter Berücksichtigung des vermutlichen Standortunterschieds gem. dem DWD von einer guten Vorhersage ausgehen. Dieser gibt hierfür einen Grenzwert von 2.5 °C (RMSE) vor [18]. Um diesen evtl. standortabhängigen Temperatureffekt zu korrigieren, bietet es sich vielleicht an, den in der Wetterstation integrierten Temperatursensor mitzuverwenden. Eine historische Auswertung der Daten und ein daraus ermittelter Korrekturfaktor könnten den Offset ausgleichen.

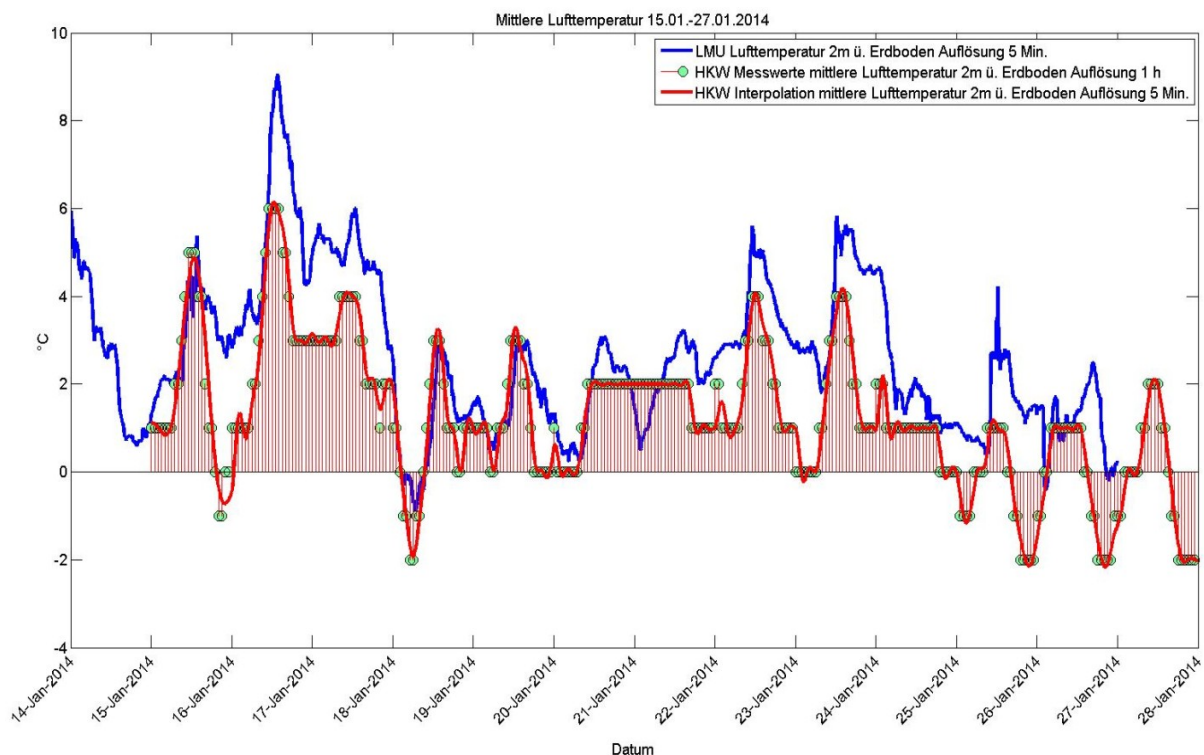


Abbildung 4.6: Vergleich der interpolierten mittleren Lufttemperatur mit den LMU Wetterdaten, Beobachtungsintervall 15.-24.01.2014

Tabelle 4.5: RMSE der mittleren Lufttemperatur in °C

Abrufdatum Intervall 18.00-24.00 Uhr	16.01.	17.01.	18.01.	19.01.	20.01.	21.01.	22.01.	23.01.	24.01.
16.01.2014	2.30	2.65	2.03	1.35					
17.01.2014		2.04	2.81	1.90	0.77				
18.01.2014			0.81	2.07	1.13	1.08			
19.01.2014				0.99	0.58	0.61	2.84		
20.01.2014					0.53	1.08	3.21	3.66	
21.01.2014						1.03	3.01	2.77	2.76
prozentuale Abweichung der realen HKW Werte von der Prognose		-23.0	+38.4 -71.2	+40.7 +9.0 -52.2	+46.8 -48.7 -8.6	-43.5 +77.1 -4.6	+13.0 -6.2	-24.3	

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 meteorologischen 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 -untergangsadaptation ist bis jetzt nur für deutsche Städte vorgesehen. Sollen andere europäische Wetterregionen abgerufen werden, so müssen die Längen- und Breitengrade im Code ergänzt werden. Da die Wetterstation dazu verwendet werden soll dem Energiemanagementsystem Prognosedaten für die Energiegewinnung und den Energieverbrauch zu liefern, ist es angesichts der momentanen Datenlage eine Frage, ob diese hierfür geeignet erscheint. Eine Alternative könnten Daten aus dem Internet darstellen, die ebenfalls kostenlos zur Verfügung gestellt werden. Jedoch liegen diese nicht wie in der Wetterstation als gesammeltes Paket vor und auch die Auflösung ist meist nicht einheitlich. So gesehen ist sie angesichts des Kosten-Leistungs-Verhältnisses eine ganz gute Lösung.

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	Mittlerer Wert
Solarprognose	Sonnenscheindauer	h	0 - 6	1	1 Tag	6h	180 - 187	Mittlerer Wert Globalstrahlung
Solarprognose	Solare Einstrahlung	W/m ²	0 - 1200 / > 1200	25	1 Tag	6h	190 - 197	Mittlerer Wert bezogen auf NN
Luftdruck	Min. Lufttemperatur	hPa	< 938 - > 1063	1	1 Tag	6h	240 - 247	Mittlerer Wert 10m über Erdboden
Windprognose	Stärke	Bft	0 - 12	1	3 Tage	6h	200 - 215	N/NO/O/SO/S/SW/W/NW
Windprognose	Richtung		1 - 8	1	3 Tage	6h	220 - 235	0=keine Böen, 1=45km/h, 2=72km/h, 3=99km/h
Markantes Wetter	Böen		0,1,2,3	1	3 Tage	6h	310 - 325	0=Wahrscheinlichkeit<=50%, 1=>50%
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: Detaillierte Datenstruktur der Wetterstation[4, S. 17-26]

Anhang B

Das MODBUS Protokoll

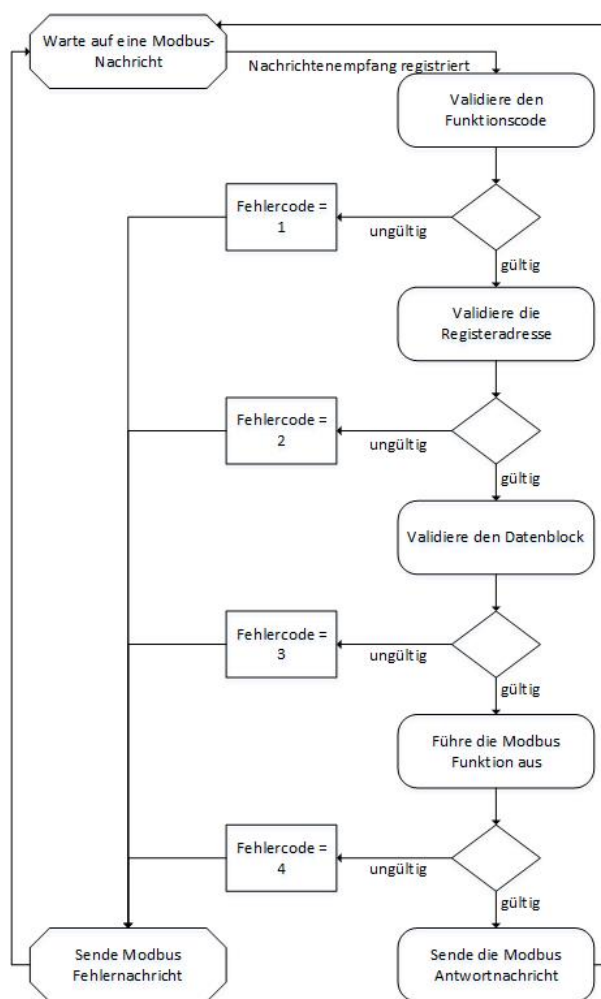


Abbildung B.1: Ablaufdiagramm für die MODBUS Nachrichtenüberprüfung [3, S. 9]

				Function Codes				
				Code	Sub Code	Hex		
Data Access	Bit Access	Physical <u>Discret</u> 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		
			Read File Record	20		14		
	File record access	Write File Record	21		15			
		Diagnostics						
	Read Exception status						07	
Diagnostic	08						00-18,20	08
Get Com Event Counter	11							
Get Com Event Log	12							0C
Report Server ID	17							11
Other		Read Device Identification	43	14	2B			
		Encapsulated Interface Transport	43	13,14	2B			
		<u>CANopen</u> General Reference	43	13	2B			

Abbildung B.2: Übersicht der zur Verfügung stehenden Funktionscodes in MODBUS [3, S. 11]

Anhang C

Hilfsfunktionen

C.1 read_com_set

Die Funktion `read_com_set` ermöglicht es auch außerhalb des Funktionsaufrufs von `forecast_data` Parameter der Wetterstation abzurufen sofern eine serielle Schnittstelle besteht. Neben dem Abruf nur eines Parameters, können auch mehrere Parameter gleichzeitig in einem Cell-Array übergeben werden. Die Ausgabe der abgerufenen Werte erfolgt dabei in der Reihenfolge der angefragten Parameter. Da die Funktion `send_and_receive_data` hier keine Daten interpolieren oder abspeichern muss, werden die Parameter für Längen- und Breitengrad, Verbindungsqualität, lokaler Temperatur und Pfadangabe des Speicherortes nicht benötigt.

```
function [ request_value ] = read_com_set( device_id, request_list, cnt )
%This function reads the communication settings
%E.g. read_com_set('03',{'city_id'})
%     read_com_set('03',{'city_id','quality'})
%
%     device_id has to be '03'
%
%     To get the index for the actual weather region setup
%     request_list has to be {'city_id'}
%
%     To get the connection quality request_list has to be {'quality'}
%
%     To get the transmitting station id from which data are received set
%     request_list {'transmitting_station'}
%
%     To get more data with a single function call just build a cell array for
%     the request_list with the interested objects like {'city_id','quality'}
%     The results are sorted in the way the request_list was grouped.

% Decide if more than one communication settings are requested
if size(request_list,2) > 1
% Progress bar to show processing time
    wb = waitbar(0,'Please wait while reading registers...');
```

```

% Send request for each list element
    for t = 1:size(request_list,2)
        waitbar(t/size(request_list,2))
% Get register address from struct
        [reg_address, fieldname] = get_reg_address(request_list{t});
% As only one value has to be set, the number of registers is
% restricted to 0x0001
        reg_num = '0001';
% Generate modbus message 'rr' indicates a read register operation
        modbus_msg = gen_msg(device_id, reg_address, reg_num, 'rsr');
% Send message and evaluate the response
        request_value(t) = send_and_receive_data(modbus_msg, fieldname,...
                                                '', '', '', '', cnt, '');
    end
% Close progress bar
    close(wb);
else
% Get register address from struct
    [reg_address, fieldname] = get_reg_address(request_list{1});
% As only one value has to be set, the number of registers is
% restricted to 0x0001
    reg_num = '0001';
% Generate modbus message 'rr' indicates a read register operation
    modbus_msg = gen_msg(device_id, reg_address, reg_num, 'rsr');
% Send message and evaluate the response, write status to protocol
    request_value = send_and_receive_data(modbus_msg, fieldname,...
                                          '', '', '', '', cnt, '');
end

end

```

C.2 write_com_set

Übergabeparameter für diese Funktion sind die Slave ID, der zu schreibende Wert in dezimal Format und die Spezifizierung der Adresse im String Format. Der Zähler *cnt* dient zur Initialisierung des while-Schleifendurchlaufs. Siehe hierzu Kapitel 3.8 auf Seite 41. Der Inputwert wird in hex umgerechnet, danach wird die Registeradresse ermittelt, die PDU gebildet und über die serielle Schnittstelle geschickt.

```

function [ ] = write_com_set( device_id, request_value_list, reg_add_list, cnt )
%Write single value to holding register
% Detailed explanation goes here

% Code for writing a single value to the holding register
fcode_ws_reg = '06';
% Create progress bar
wb = waitbar(0, 'Please wait while writing to registers ...');
% Convert dec value to 2 byte hex value

```

```

request_value_list      = dec2hex(request_value_list,4);
waitbar(1/3,wb)
% Receive register address to write value into
[reg_add, field_name]   = get_reg_address(char(reg_add_list));
% Generate modbus pdu
modbus_msg              = strcat(device_id, fcode_ws_reg, reg_add, request_value_list);
waitbar(2/3,wb)
% Generate modbus adu
msg                     = crc_calc(char(modbus_msg));
% Send message
[rxdata]                = send_and_receive_data(msg, field_name, '', '', '', cnt, '');
waitbar(3/3,wb)
% Close progress bar
close(wb)

end

```

C.3 stop_timer

Die Funktion **stop_timer** meldet dem Nutzer das Ende des Timer Objektes, löscht es, speichert den Datencontainer mit den Langzeitdaten ab und schließt die serielle Schnittstelle. Vorteil dieser Funktion ist es, sobald ein Fehler in der forecast Funktion auftritt, wird zumindest der Datencontainer immer noch gesichert.

```

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

```

C.4 get_reg_address

Die Funktion erhält einen String mit der Information ob es sich um eine Abfrage für die Einstellungsparameter oder Wetterdaten der Wetterstation handelt. Da das Register mit den Adressen in einem struct angelegt wurde, kann man die gesuchte Adresse einfach durch Vorgabe der Zweigpunkte mit dem Befehl **getfield** in dieser Struktur abrufen. Man hätte die Registeradressdaten auch in einer Database anlegen können, aber auch hier hat ein Test gezeigt, dass der Abruf aus der Struktur ein bisschen schneller ist. Schließlich muss nicht immer wieder die komplette Database durchsucht werden.

```
function [ reg_address, field_name ] = get_reg_address( varargin )
%Gets the register number as hex value
% Inputparameter could be forecast scope, forecast detail, {day of
% observation and daysegment}. E.g. 'niederschlag','menge',{'heute',
% 'morgen'} or a member from the coillist 'fsk_qualitaet',
% 'status_ext_temp_sensor', or a request for the radio clock
% 'radio_clock.sec',..., 'radio_clock.year', or a communication setting
% parameter like 'city_id', 'transmitting_station', 'quality',
% 'temperature', 'temperatre_offset'

register_data_hwk_kompakt = evalin('base', 'register_data_hwk_kompakt');

%
if nargin == 1

    coil_list = {'fsk_qualitaet' 'status_ext_temp_sensor'...
                 'reserve1' 'reserve2' 'reserve3'};

    if strmatch(char(varargin), coil_list) ~= 0

        reg_address = getfield(...
            register_data_hwk_kompakt.communication_settings.coil.status,char(varargin));
        field_name = {'register_data_hwk_kompakt.communication_settings'};

    elseif ~isempty(cell2mat(strfind(varargin,'radio_clock')))

        varargin = regexp(char(varargin), '[.]', 'split');

        reg_address = getfield(...
            register_data_hwk_kompakt.communication_settings.register.radio_clock,...
            char(varargin(2)));
        field_name = {'register_data_hwk_kompakt.communication_settings'};

    else

        reg_address = getfield(...
            register_data_hwk_kompakt.communication_settings.register,char(varargin));
        field_name = {'register_data_hwk_kompakt.communication_settings'};
```



```

end

else

% read the structure field for forecast details determined by the
% forecast scope (= varargin{1})
    prog_detail_field = getfield(register_data_hwk_kompakt,varargin{1});
% read the structure field for forecast days determined by the
% forecast detail (= varargin{2})
    prog_day_field = getfield(prog_detail_field,varargin{2});
% read the structure field for forecast hours determined by the
% forecast days (= varargin{3}(1))
    prog_hour_field = getfield(prog_day_field, char(varargin{3}(1)));
% get the register address determined by the forecast hour
% (= varargin{3}(2))
    reg_address = getfield(prog_hour_field, char(varargin{3}(2)));
    field_name = {'register.data.hwk.kompakt.',varargin{1},varargin{2}...
        ,char(varargin{3}(1)),char(varargin{3}(2))};

end

```

C.5 reg_num

Es erfolgt die Berechnung der Anzahl an Registeradressen. Sind die Adressen gleich würde die Differenz 0 ergeben. Daher wird zum Ergebnis der Subtraktion eine 1 addiert.

```

function [ num_reg ] = reg_num( start_address, end_address)
%Calculates the difference between start and end register number
% Detailed explanation goes here
if hex2dec(start_address) > hex2dec(end_address)
    error('Startadresse des Registers ist groesser als Endadresse!','Eingabefehler');
else
    num_reg = dec2hex(hex2dec(end_address)-hex2dec(start_address)+1,4);
end
end

```

C.6 format_modbus_msg

Die Funktion `format_modbus_msg` transformiert den n-Byte großen String der MODBUS-Nachricht in einen n-zeiligen Vektor mit dezimalen Werten.

```

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

```

C.7 fcode_check

Es wird lediglich überprüft, ob der Funktionscode den Wert 1, 3 oder 6 hat. Ist dies nicht der Fall, wird das Flag *fcode_error* auf 1 gesetzt.

```

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

```

C.8 MESZ_calc

Die Berücksichtigung der MESZ spielt vor allem für die Berechnung des Sonnenauf- und Sonnenuntergangs eine Rolle. Aber auch bei der Erstellung des Zeitstempels für den Zeitpunkt des Datenabrufs. Hierbei wird einfach überprüft ob das aktuelle Datum im Intervall für die Sommerzeit liegt oder nicht. Entsprechend wird das Flag ausgegeben.

```

function [ MEZ ] = MESZ_calc( )
%Determines a flag to consider central european summer time or not
% Detailed explanation goes here
years = {'2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020'};

t_ttbl_f_MESZ_change = {'31-Mar-2013 02:00:00' , '30-Mar-2014 02:00:00' , ...
                        '29-Mar-2015 02:00:00' , '27-Mar-2016 02:00:00' , ...

```

```

        '26-Mar-2017 02:00:00' , '25-Mar-2018 02:00:00', ...
        '31-Mar-2019 02:00:00' , '29-Mar-2020 02:00:00'};

t_tbl_f.MEZ_change = { '27-Oct-2013 03:00:00', '26-Oct-2014 03:00:00', ...
        '25-Oct-2015 03:00:00', '30-Oct-2016 03:00:00', ...
        '29-Oct-2017 03:00:00', '28-Oct-2018 03:00:00', ...
        '27-Oct-2019 03:00:00', '25-Oct-2020 03:00:00'};

% Determine position of the actual year from years list
[a,b] = ismember(datestr(date,10),years);

% If there is an entry in the list compare the current date with list
% entries. If it is greater than t_tbl_f.MESZ_change and less then
% t_tbl_f.MEZ_change central european summer time has to be set and so
% central european time is set to 0.
if a == 1
    if now > datenum(t_tbl_f.MESZ_change{b}) && now < datenum(t_tbl_f.MEZ_change{b})
        MEZ = 0;
    else
        MEZ = 1;
    end
else
    fprintf(2,'Liste mit Zeitumstellung muss aktualisiert werden!!!!\n', char(10))
    error('Problem mit Zeitumstellung');
end
end

```

C.9 res_factor

Da die Ausgangsdaten selbst in unterschiedlicher Auflösung vorliegen, muss zur Berechnung der Anzahl interpolierter Werte ein Faktor verwendet werden. Dieser ergibt sich aus der zu Grunde liegenden zeitlichen Auflösung geteilt durch die gewünschte zeitliche Auflösung.

```

function [ factor ] = res_factor( resolution, f_c_d )
%Determines the factor for interpolation intervall construction
% As there are two different resolutions in the received data we have to
% assign two different factors. E.g. a resolution of 0.25 hours is
% requested, the factor for the mittlere_temp_prog results from 1 hour
% divided by 0.25 = 4.
switch resolution

    case 6
        factor = 1;
    case 1
        if strcmp(f_c_d,'mittlere_temp_prog') == 1
            factor = 1;
        else
            factor = 6;
        end
end

```

```

case 0.5
    if strcmp(f_c_d,'mittlere.temp-prog') == 1
        factor = 2;
    else
        factor = 12;
    end
case 0.25
    if strcmp(f_c_d,'mittlere.temp-prog') == 1
        factor = 4;
    else
        factor = 24;
    end
case 0.08
    if strcmp(f_c_d,'mittlere.temp-prog') == 1
        factor = 12;
    else
        factor = 72;
    end

end

end

```

C.10 utc2date

Diese Funktion wurde so geschrieben, dass sie Eingabevektoren verarbeiten kann. Ohne diese Maßnahme hätte bei der Interpolation eine for-Schleife für die Berechnung der Start- und Stopzeitpunkte des Gültigkeitsintervalles der Daten in normalem Zeitformat verwendet werden müssen. Ein Test der Laufzeitverbesserung ergab 12 Sekunden für eine for-Schleife. Der hier verwendete Algorithmus entspricht dem im Minix Source Code verwendeten Algorithmus.[20]

```

function [ date_str ] = utc2date( utc_time )
%Converts unix time to date
%   Detailed explanation goes here

% Define start year of unix time calculation
year = 1970;
% Calculate how many seconds elapsed since 0.00 o'clock
day_clock = mod(utc_time,86400);
% Calculate how many days elapsed since 01.01.1970
day_no = floor(double(utc_time)./86400);

% Calculate elapsed hours, min, sec since 0.00 o'clock
hour = floor(day_clock./3600);
min = floor(double(mod(day_clock,3600))./60);
sec = mod(day_clock,60);

% wd = mod(day_no + 4,7);

```

```

% Define the number of days in year, considering leap years
ys(1:size(utc_time,2)) = {year_size(year)};
temp = cell2mat(ys);
year_days = temp(1:3:end);

% Reduce day_no by number of days in a year until day_no is less than a year
while day_no > year_days
    day_no = day_no - year_days;
    year = year+1;
    ys(1:size(utc_time,2)) = {year_size(year)};
    temp = cell2mat(ys);
    year_days = temp(1:3:end);
end

y(1:size(utc_time,2)) = year;
m(1:size(utc_time,2)) = 1;

% Get the number of days in a year, the elapsed days since 01.01. of that
% year, and the number of days in a month starting with January
j(1:size(utc_time,2)) = {year_size(year, m)};
temp = cell2mat(j);
remaining_days = temp(2:3:end);
day_a_month = temp(3:3:end);
% Loop until remaining day_no is less than elapsed days since 01.01.
while day_no >= remaining_days;
    if (double(day_no) - day_a_month) <= 0
        break;
    else
% Subtract number of days in a month
        day_no = day_no - day_a_month;
    end
% Increase number of elapsed month since 01.01.
    m = m + 1;
    j(1:size(utc_time,2)) = {year_size(year, m)};
    temp = cell2mat(j);
    remaining_days = temp(2:3:end);
    day_a_month = temp(3:3:end);
end
d = double(day_no)+1;
result = double([y; m; d; hour; min; sec]);
date_str(1:size(utc_time,2)) = cellstr(datestr(result',0))';
end

```

C.11 tvector

```

function [ timevec ] = tvector(f_c_d, d_p, varargin )
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

```

```

if strcmp(f_c_d, 'mittlere_temp_prog') ~= 1
    switch varargin{2}
        case 'morgen'
            timevec = [d_p, 0 0 0];
        case 'vormittag'
            timevec = [d_p, 6 0 0];
        case 'nachmittag'
            timevec = [d_p, 12 0 0];
        case 'abend'
            timevec = [d_p, 18 0 0];
    end
else
    switch varargin{1}
        case 'am0_00'
            timevec = [d_p, 0 0 0];
        case 'am01_00'
            timevec = [d_p, 1 0 0];
        case 'am02_00'
            timevec = [d_p, 2 0 0];
        case 'am03_00'
            timevec = [d_p, 3 0 0];
        case 'am04_00'
            timevec = [d_p, 4 0 0];
        case 'am05_00'
            timevec = [d_p, 5 0 0];
        case 'am06_00'
            timevec = [d_p, 6 0 0];
        case 'am07_00'
            timevec = [d_p, 7 0 0];
        case 'am08_00'
            timevec = [d_p, 8 0 0];
        case 'am09_00'
            timevec = [d_p, 9 0 0];
        case 'am10_00'
            timevec = [d_p, 10 0 0];
        case 'am11_00'
            timevec = [d_p, 11 0 0];
        case 'am12_00'
            timevec = [d_p, 12 0 0];
        case 'pm01_00'
            timevec = [d_p, 13 0 0];
        case 'pm02_00'
            timevec = [d_p, 14 0 0];
        case 'pm03_00'
            timevec = [d_p, 15 0 0];
        case 'pm04_00'
            timevec = [d_p, 16 0 0];
        case 'pm05_00'
            timevec = [d_p, 17 0 0];
        case 'pm06_00'
            timevec = [d_p, 18 0 0];
    end
end

```

```

        case 'pm07_00'
            timevec = [d.p, 19 0 0];
        case 'pm08_00'
            timevec = [d.p, 20 0 0];
        case 'pm09_00'
            timevec = [d.p, 21 0 0];
        case 'pm10_00'
            timevec = [d.p, 22 0 0];
        case 'pm11_00'
            timevec = [d.p, 23 0 0];
    end
end

end

```

C.12 date2utc

Der hier verwendete Algorithmus entspricht der Berechnungsvorschrift des IEEE.[21, S. 113]

```

function [ unix_zeit ] = date2utc( date_vector, varargin )
%Converts a date vector into unix time code, i.e. date2utc(datevec(now))
% Detailed explanation goes here

jahr = uint64(date_vector(1));
monat = uint64(date_vector(2));
tag = uint64(date_vector(3));
std = uint64(date_vector(4));
min = uint64(date_vector(5));
sec = uint64(date_vector(6));
tage_seit_jahresanfang = {0,31,59,90,120,151,181,212,243,273,304,334};
jahre = jahr - 1970;
jahr = jahr-1900;
schaltjahre = floor((jahr-69)/4)-floor((jahr-1)/100)+floor((jahr+299)/400);
unix_zeit=sec+60*min+3600*std+(tage_seit_jahresanfang{monat}+tag-1)*86400+(365*jahre+schaltjahre)*86400;
if isempty(varargin)
    varargin = {0};
end
if cell2mat(varargin) == 0
    %MESZ is valid
    unix_zeit = unix_zeit + 3600;
end
end

```

C.13 data_mult

Laut Spezifikation des Herstellers[4, S. 19-20] müssen die Werte für die Niederschlagsmenge und die Sonnenscheindauer durch 10 geteilt werden, um die Einheiten l/m² und h zu erhalten.

```

function [ proc_value ] = data_mult( dec_value, prog_detail )
%UNTITLED Summary of this function goes here
% Detailed explanation goes here.
if strcmp(prog_detail,'menge') == 1 || strcmp(prog_detail,'dauer') == 1
    proc_value = dec_value/10;
else
    proc_value = dec_value;
end

end

```

C.14 diurnal_var

Der Algorithmus für die Berechnung des Sonnenauf- und Sonnenuntergangzeitpunktes wurde dem Kapitel 2.2 „Astronomische Gegebenheiten“ entnommen. [22]

```

function [ sunrise_vec sunset_vec ] = diurnal_var( longitude, latitude, date_of_day )
%Calculates the time of sunset and sunrise for given date, longitude and
%latitude
% Detailed explanation goes here
phi_horz = pi/4;
if MESZ_calc == 1
    prime_meridian = pi/12;
else
    prime_meridian = pi/6;
end

current_date = regexp(datestr(date_of_day,6), '/', 'split');
current_month = str2double(current_date(1));
current_day = str2double(current_date(2));

day_num = 30.3*(current_month-1)+current_day;

declination = (23.45/360)*2*pi*sin(2*pi*(284+day_num)/365);

w_s = abs(acos(-tan(declination)*tan(latitude/360*2*pi)));

tlt_sunrise = 12 - w_s/((15/360)*2*pi);
tlt_sunset = 12 + w_s/((15/360)*2*pi);

tequation = 0.123*cos(2*pi*(88+day_num)/365)-0.167*sin(4*pi*(10+day_num)/365);

sunset = tlt_sunset-(tequation+((longitude/360)*2*pi-prime_meridian)/((15/360)*2*pi));
sunrise = tlt_sunrise-(tequation+((longitude/360)*2*pi-prime_meridian)/((15/360)*2*pi));

sunset_hour= floor(sunset);
sunset_dec = sunset-sunset_hour;
sunrise_hour = floor(sunrise);
sunrise_dec = sunrise-sunrise_hour;

```



```

sunrise_min = floor(sunrise_dec*60);
sunrise_sec = floor((sunrise_dec*60 - sunrise_min)*60);

sunset_min = floor(sunset_dec*60);
sunset_sec = floor((sunset_dec*60 - sunset_min)*60);

dv = datevec(date_of_day);

sunrise_vec = [dv(1:3) sunrise_hour sunrise_min sunrise_sec];
sunset_vec = [dv(1:3) sunset_hour sunset_min sunset_sec];

end

```

C.15 neg_val_corr

Funktionsinput sind das Prognosedetail oder der Prognosebereich, die interpolierten Werte und die dazugehörigen Messzeitpunkte. Abhängig von den übergebenen Wetterdaten wird das übergebene Zeitintervall der Messpunkte entweder aufgeteilt oder nicht. Für die Daten der Solarleistung, werden Zeitbereiche gebildet, die vor Sonnenauf- und nach Sonnenuntergang liegen, bzw. dazwischen. Werte egal ob positiv oder negativ, die nach oder vor Sonnenauf- bzw. Sonnenuntergang liegen werden zu Null gesetzt. Werte die dazwischen liegen und negativ sind werden zu Null gesetzt. Für die Niederschlagsmenge und Windstärke werden alle negativen Werte zu Null gesetzt.

```

function [ corr_values ] = neg_val_corr( fc_def, edindex, diurnal_cont, unix_t_mean,...
    int_val)
%Corrects values which cannot be negativ due to interpolation
% Detailed explanation goes here

if (strcmp(fc_def,'dauer') == 1 || strcmp(fc_def,'einstrahlung') == 1) && edindex == 1
    sun_rise_today = diurnal_cont(1);
    sun_set_today = diurnal_cont(2);
    temp_x = unix_t_mean;
    temp1 = temp_x <= sun_rise_today;
    temp2 = temp_x > sun_rise_today & temp_x < sun_set_today;
    temp3 = temp_x >= sun_set_today;
    temp_y = int_val;
    tempy1 = temp_y(temp1);
    tempy2 = temp_y(temp2);
    tempy3 = temp_y(temp3);
    tempy1(tempy1<0) = 0;
    tempy1(tempy1>0) = 0;
    tempy2(tempy2<0) = 0;
    tempy3(tempy3<0) = 0;
    tempy3(tempy3>0) = 0;
    corr_values = [tempy1 tempy2 tempy3];
elseif (strcmp(fc_def,'dauer') == 1 || strcmp(fc_def,'einstrahlung') == 1) && edindex == 2

```

```

sun_rise_today = diurnal_cont(1);
sun_set_today = diurnal_cont(2);
sun_rise_tomorrow = diurnal_cont(3);
sun_set_tomorrow = diurnal_cont(4);
temp_x = unix_t_mean;
temp1 = temp_x <= sun_rise_today;
temp2 = temp_x > sun_rise_today & temp_x < sun_set_today;
temp3 = temp_x >= sun_set_today & temp_x <= sun_rise_tomorrow;
temp4 = temp_x > sun_rise_tomorrow & temp_x < sun_set_tomorrow;
temp5 = temp_x >= sun_set_tomorrow;
temp_y = int_val;
temp_y1 = temp_y(temp1);
temp_y2 = temp_y(temp2);
temp_y3 = temp_y(temp3);
temp_y4 = temp_y(temp4);
temp_y5 = temp_y(temp5);
temp_y1(temp_y1<0) = 0;
temp_y1(temp_y1>0) = 0;
temp_y2(temp_y2<0) = 0;
temp_y3(temp_y3<0) = 0;
temp_y3(temp_y3>0) = 0;
temp_y4(temp_y4<0) = 0;
temp_y5(temp_y5<0) = 0;
temp_y5(temp_y5>0) = 0;
corr_values = [temp_y1 temp_y2 temp_y3 temp_y4 temp_y5];
elseif strcmp(fc_def,'menge') == 1 || strcmp(fc_def,'staerke') == 1
    temp = int_val;
    temp(temp < 0) = 0;
    corr_values = temp;
end

end

```

C.16 crc_check

Der CRC Check zerlegt die empfangene Nachricht des Slaves. Von Interesse ist nur der Teil der Nachricht ohne CRC Summe *size(rxdata,1-2)*. Für jedes Byte wird die Dezimalzahl in eine Hexzahl konvertiert und zur MODBUS Nachricht in Hexformat zusammengesetzt. Danach wird die Funktion **crc_calc** verwendet um deren Ergebnis mit dem übermittelten CRC Wert zu vergleichen.

```

function [ check, error_msg ] = crc_check( rxdata )
%Calculates the CRC sum from the slave response
% Detailed explanation goes here
msg = '';
% for each byte in the response message convert dec to hex values and
% concatenate it to the message in hex format
for t = 1:(size(rxdata,1)-2)

```

```

hex_val = dec2hex(rxdata(t),2);
msg = strcat(msg,hex_val);
end
% extract the crc checksum and calculate the crc
% checksum from the response message
crc_checksum = strcat(dec2hex(rxdata(end-1,1),2),dec2hex(rxdata(end,1),2));
response_msg = crc_calc(msg);
% compare calculated checksum and received checksum and assign flag value
if strcmp(response_msg(end-3:end),crc_checksum) == 1
    check = 1;
    error_msg = [];
else
    check = 0;
    error_msg = 'CRC Check war ungueltig';
end
end

```

Anhang D

Abkürzungsverzeichnis

ADU Application Data Unit

Bft Beaufort

C Celsius

COM Communication

CRC Cyclic Redundancy Check

DWD Deutscher Wetterdienst

EMU Energy Management Unit

EV Electric Vehicle

FSK Frequency Shift Keying

GmbH Gesellschaft mit beschränkter Haftung

h Stunde

hPa Hektopascal

HEMS Home Energy Management System

H-Byte High-Byte

ID Identifikationsnummer

int8 8 Bit-Integer

IP Internet Protocol

I/O Input/Output

LMU Ludwig-Maximilians-Universität

L-Byte Low-Byte

N Nord

NN Normal Null Meeresspiegel

NO Nordost

NW Nordwest

O Osten

OSI Open Systems Interconnection

PDU Protocol Data Unit

RMSE Root mean square error

RS Recommended Standard

RTU Remote Terminal Unit

rx Empfänger

TCP Transmission Control Protocol

S Sueden

SO Suedost

SW Suedwest

W Westen

Literaturverzeichnis

- [1] Paul Telleen et al. Tali Trigg. Global EV Outlook. Website, April 2013. Online verfügbar auf http://www.iea.org/publications/globalevoutlook_2013.pdf; zuletzt geprüft am 24.01.2014.
- [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] F. Bouhafs, M. Mackay, and M. Merabti. Links to the future: Communication requirements and challenges in the smart grid. *Power and Energy Magazine, IEEE*, 10(1):24–32, 2012.
- [5] 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.
- [6] Abruf vom 17.01.2014 19.25 Uhr. Google Earth Version 7.1.2.2041, Januar 2014.
- [7] Gerd Küveler and Dietrich Schwoch. *Informatik für Ingenieure und Naturwissenschaftler*. Viewegs Fachbücher der Technik. Vieweg, Braunschweig [u.a.], 5 edition, 2007.
- [8] 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.
- [9] Industrial Automation Systems MODICON, Inc. Modicon Modbus Protocol Reference Guide. Website, Juni 1996. Online verfügbar auf http://modbus.org/docs/PI_MBUS_300.pdf; zuletzt geprüft am 28.01.2014.
- [10] 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.

- [11] Virtual serial ports emulator. Website. Online verfügbar auf <http://www.eterlogic.com/downloads/SetupVSPE.zip>; zuletzt geprüft am 28.01.2014.
- [12] Peakhmi Modbus Serial RTU Slave simulator. Website. Online verfügbar auf <http://www.hmisys.com/downloads/PeakHMISlaveSimulatorInstall.exe>; zuletzt geprüft am 28.01.2014.
- [13] 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.
- [14] J. S. Armstrong and Frederick Lynch Collopy. Error measures for generalizing about forecasting methods: Empirical comparisons. *International journal of forecasting*, 1992.
- [15] Herr Heinz Lösslein. Re: Frage bzgl. wetterdaten. Email. Email Korrespondenz vom 27.01.2014.
- [16] Hr. Volkhardt. Anruf vom 27.01.2014 10.22 Uhr. Telefonanruf, Januar 2014. Email: volkhardt@hkw-elektronik.de.
- [17] Lars Kühl. Windenergie. In Matthias Kramer, editor, *Integratives Umweltmanagement*, pages 611–634. Gabler, 2010.
- [18] Roland Wengenmayr. Wie gut sind Wettervorhersagen? Qualitätsprüfung beim DWD. Website, Oktober 2009. Online verfügbar auf http://www.dwd.de/bvbw/generator/DWDWWW/Content/Presse/Broschueren/Verifikation__PDF,templateId=raw,property=publicationFile.pdf/Verifikation_PDF.pdf; zuletzt geprüft am 24.01.2014.
- [19] Steinicke und Streifeneder. Thermalbefliegung München Tagaufnahme. Website, Mai 1999. Online verfügbar auf http://www.muenchen.de/rathaus/dms/Home/Stadtverwaltung/Referat-fuer-Gesundheit-und-Umwelt/Dokumente/Stadtklima/mue_tag_klein/Thermalbefliegung%20Tagaufnahme.pdf; zuletzt geprüft am 24.01.2014.
- [20] Prof. Beat Hirsbrunner. Minix source code. Website, 2005/2006. Online verfügbar auf http://diuf.unifr.ch/pai/education/2005_2006/courses/os/minix/idx/S/sys%20src%20lib%20ansi%20gmtime.c.html#11; zuletzt geprüft am 24.01.2014.
- [21] Ieee standard for information technology- portable operating system interface (posix) base specifications, issue 7. *IEEE Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004)*, pages c1–3826, 2008.
- [22] Andreas Wagner. *Photovoltaik Engineering: Handbuch für Planung, Entwicklung und Anwendung*. VDI-Buch. Springer-Verlag Berlin Heidelberg, Berlin and Heidelberg, 2 edition, 2006.