Temperatures Estimation System of Electrical Machines on Wireless Sensor Networks

vorgelegt von
M.Sc.
Yi Huang

an der Fakultät IV - Elektrotechnik und Informatik
der Technische Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
-Dr.-Ing.-

genehmigte Dissertation

Promotionsausschuss

Vorsitzender:	Prof. Dr.-Ing. Reinhold Orglmeister
Gutachter:	Prof. Dr.-Ing. Clemens Gühmann
Gutachter:	Prof. Dr.-Ing. Uwe Schäfer
Gutachter:	Prof. Dr. Ruijuan Chi

Tag der wissenschaftlichen Aussprache: 6. Juli 2020

Berlin 2021

Acknowledgments

First and foremost, I would like to express my deepest appreciation to my doctoral thesis supervisor, professor Dr.-Ing. Clemens Gühmann, head of the chair, who has not only granted me the precious chance to be his doctoral student but also provided me with invaluable guidance as well as steady and strong support. I owe so much to his unwavering encouragement, which has spurred my initiatives into full swing to reach effective fruition and guided me through my doctoral studies.

Additionally, I am very grateful to professor Dr.-Ing. Uwe Schäfer from the Technical University of Berlin, professor Dr.-Ing. Reinhold Orglmeister from the Technical University of Berlin and professor Dr. Ruijuan Chi from China Agriculture University for their very supportive guidance. Their advice and their participation in the committee of examiners must be duly acknowledged.

Next, I owe my speedy adaptation to the environment of my laboratory and extensive comprehension of the outcome of my research to the support of all my colleagues, especially Dr.-Ing Jürgen Funck, who provided me with so much help during my research.

I must also express my appreciation to my students, Mr. Thilo Geismar, Mr. Yuan Gao, Ms. Wenjun Zhu, Mr. Zhecheng Jin, just to name a few. They have helped me to bring my research into a deeper level.

Last but not least, I am most thankful for the support of my family. Hearty thanks to my parents for their unlimited spiritual support, and for bringing me up. I must also thanks to my wife Jing Yuan, whose constant encouragement provided a source of motivation throughout my studies. I also acknowledge Chinese Scholarship Council (CSC) for granting me financial support during my Ph.D. studies.

Abstract

This dissertation proposes a model-based software method to develop a temperatures estimation system for an asynchronous machine, which is implemented in wireless sensor networks (WSN). The system can estimate the temperatures of the stator winding, the rotor cage and the stator core.

Firstly, a physical model of an asynchronous machine is built and validated in Dymola. The electrical, mechanical and the thermal behaviors performed well in the Dymola simulation model. Based on the physical model, an efficient and reliable thermal model for tracking the temperatures of the stator winding, the rotor cage and the stator core is built using Dymola. All the thermal parameters of the asynchronous machine are identified. One of the most difficult tasks is to identify the reference stator core losses and reference friction losses, which can be determined by a no-load test and load test on the test bench. The conductance values are calculated by the losses and temperatures at the steady state of the machine. The best-fit capacitances are found by using Genopt, an optimization program.

Two different algorithms are used for the temperatures estimation. A 4^{th}-order Kalman filter (KF) algorithm and a 9^{th}-order extended Kalman filter (EKF) are first implemented based on the state-space equations in MATLAB/SIMULINK. The Model-in-the-Loop (MiL) method is used to verify the algorithms. The physical model in Dymola and the algorithms are connected together in the simulation using SIMULINK. After the verification of the algorithm, both are implemented in a wireless sensor network (WSN), which is based on the IEEE1451 standard using Contiki OS. To estimate the respective temperatures of the stator winding, the rotor cage and the stator core of an asynchronous machine, KF and EKF algorithms are implemented into the resource restricted embedded system.

Finally, under different experiment conditions, the temperatures estimation system in WSN are tested on the test bench. The real-time WSN temperature estimation system is independent from the control algorithm and functional under any load condition, as long as the current of the stator is a nonzero system and measured with very high accuracy.

Zusammenfassung

Diese Arbeit befasst sich mit einer modellbasierten Software zur Temperaturschätzung für die Statorwicklung, den Rotorkäfig und den Statorkern einer Asynchronmaschine. Die Software kann in einem drahtlosen Sensornetzwerk implementiert werden.

Zunächst wurde ein physikalisches Modell einer Asynchronmaschine in "Dymola" aufgebaut und deren elektrisches, mechanisches und thermisches Verhalten validiert. Danach wurde das physikalische Modell mit einem effizienten und zuverlässigen thermischen Modell mit allen thermischen Parametern erweitert, um die Temperaturen der Statorwicklungen, dem Rotorkäfig und dem Statorkern zu verfolgen. Die Leitwerte wurden durch die Verluste und Temperaturen im stationären Zustand der Maschine berechnet. Die Kapazitäten wurden mit "GenOpt" gefunden.

Für die Temperaturschätzung wurden ein 4^{th}-Order Kalman Filter (KF) und ein 9^{th}-Order Extended Kalman Filter (EKF) basierend auf den Zustandsraumgleichungen in Simulink implementiert. Die beiden Filter-Algorithmen wurden mithilfe der Model-in-the-Loop (MiL) Methode verifiziert. Danach wurden das physikalische Modell in "Dymola" und die Filter-Algorithmen simulativ miteinander in Simulink verbunden und anschliessend in einem Wireless Sensornetzwerk (WSN) implementiert. Das Netzwerk basiert auf dem IEEE1451-Standard unter Contiki OS. Um die jeweiligen Temperaturen der Statorwicklungen, dem Rotorkäfig und dem Statorkern einer Asynchronmaschine zu schätzen, wurden viele Ansätze verwendet, um den Algorithmus in das ressourcenbeschränkte eingebettete System zu integrieren.

Zuletzt wurde das Temperaturschätzsystem im WSN unter verschiedenen Versuchsbedingungen am Prüfstand getestet. Das Schätzsystem in Echtzeit ist unabhängig von dem Steueralgorithmus und unter allen Lastbedingungen funktionsfähig, solange der Strom vom Stator nicht gleich Null ist und mit einer sehr hohe Genauigkeit gemessen wurde.

CONTENTS

List of Figures

List of Tables

LISTINGS

List of Abbreviations and Symbols

Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ADC	Analog-to-Digital Conversion
AIM	Asynchronous Induction Machine
API	Application Programming Interface
ARM	Advanced RISC Machine
CIC	Cascaded integrator-comb
CMSIS	Cortex Micro-controller Software Interface Standard
CoAP	Constrained Application Protocol
CPL	Calculated Power Losses
CPLC	Calculated Power Losses with Compensation
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DAQ	Data Acquisition
DSP	Digital Signal Processing
EKF	extended Kalman Filter
EPL	Export Power Losses
FFT	Fast Fourier Transform
FIFO	First-In-First-Out
FIR	Finite Impulse response
FPU	Float Point Unit
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
HiL	Hardware-in-Loop
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IIR	Infinite Impulse Response
IoT	Internet of Things
KF	Kalman Filter
LED	Light-Emitting Diode

M2M	Machine-to-Machine
MDT	Chair of Electronic Measurement and Diagnostic Technology
MEMS	Micro Electro Mechanical Systems
MiL	Model-in-Loop
MMF	Magneto-motive Force
MSB	Most Significant Bit
MSTL	MDT Smart Transducer Library
NCAP	Network Capable Application Processor
NI	National Instrument
NRMSE	Normalized Root-Mean-Square Error
PI	Proportional Integral
PID	Proportional-Integral-Derivative
PMSM	Permanent Magnet Synchronous Machine
PWM	Pulse Width Modulation
RAM	Random Access Memory
RFID	Radio-Frequency Identification
RMS	Root-Mean-Square
ROM	Read-Only Memory
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks
RPM	Revolutions per Minute
S1	Continuous Full-Load Test
S6	Intermittent-Load Test
TCP	Transmission Control Protocol
TEDS	Transducer Electronic Data Sheet
TIM	Transducer Interface Module
UDP	User Datagram Protocol
WPAN	Wireless Personal Area Networks
WSN	Wireless Sensor Networks
WTIM	Wireless Transducer Interface Module

Symbols

α_{ref}	Temperature coefficient
α_r	Rotor temperature coefficient
α_s	Stator temperature coefficient
ΔT	Temperature drop
$\dot{Q}$	Heat flow
$\lambda_{dr}, \lambda_{qr}$	Rotor flux linkage in d-q axis
$\lambda_{ds}, \lambda_{qs}$	Stator flux linkage in d-q axis
ω	Mechanical angular frequency

ω_r	Electrical angular velocity
ω_s	Stator frequency
ω_{frame}	Reference frame frequency
ω_{ref}	Reference speed
σ	Leakage inductance coefficient
σ_r	Rotor leakage inductance coefficient
σ_s	Stator leakage inductance coefficient
τ	The sampling time; The torque
A	System matrix
B	Input matrix
C	Output matrix
D	Feedthrough matrix
F	Process Jacobians matrix
H	Measurement Jacobians matrix
K	Kalman gain
Q	Process covariance matrix
R	Measurement covariance matrix
u	Control vector
x	State vector
z	Measurement vector
C	Thermal capacitance
f	Input signal frequency
f_r	The frequency of the rotor
f_s	The frequency of the stator
$f_{Nominal}$	Nominal frequency
G	Thermal conductance
I	Effective value of input current
$i(t)$	Instantaneous input current of asynchronous machine
I_L	Line current
$i_{\alpha r}$, $i_{\beta r}$	$\alpha - \beta$ rotor currents
$i_{\alpha s}$, $i_{\beta s}$	$\alpha - \beta$ stator currents
i_{ar}, i_{br}, i_{cr}	Three phase rotor current
i_{as}, i_{bs}, i_{cs}	Three phase stator current
$I_{Nominal}$	Nominal RMS current
i_{qr}, i_{dr}	d-q rotor currents
i_{qs}, i_{ds}	d-q stator currents
I_{ref}	Reference current
J	Rotor inertia

k_{iron}	Iron loss constant
L_m	Main field inductance
L_r	Rotor inductance
L_s	Stator inductance
$L_{r\sigma}$	Rotor leakage inductance
$L_{s\sigma}$	Stator leakage inductance
n_s	Synchronous speed
n_t	Measured speed
p_n	Number of pole pairs
P_U, P_V, P_W	Input power per phase
P_θ	Thermal power
$P_{coreRef}$	Reference stator core losses
$P_{frictionRef}$	Reference friction losses
P_{fw}	Friction losses
P_{grc}	Generator rotor losses
P_{in}	Input power losses
P_{Loss}	Total losses of the machine
P_{mech}	Mechanical output power
P_{mrc}	Motor rotor losses
$P_{Nominal}$	Nominal power output
P_{out}	Output power losses
P_{rc}	Rotor cage losses
P_{ref}	Reference power
P_{sc}	Stator core losses
P_{stray}	Stray load losses
P_{sub}	Remaining power of input power subtracted by stator loss
P_{sw}	Stator winding losses
$Pf_{Nominal}$	Nominal power factor
r_H	The ration of hysteresis losses
R_r	Rotor resistance per phase
R_s	Stator resistance per phase
$R_{Operation}$	Resistance in operation
R_{ref}	Resistance under reference temperature
rpm	Revolutions per minute
s	Slip of asynchronous machine
T_c	Coolant air temperature
T_e	Electromagnetic torque
T_l	Load torque

T_{err}	Temperature error rate
T_{mea}	Measured temperature
T_{ref}	Reference temperature
T_{sim}	Simulated temperature
T_{sw}, T_{sc}, T_{rc}	Temperatures of stator winding, stator core and rotor cage
U	Effective value of input voltage
$u(t)$	Instantaneous input voltage of asynchronous machine
U_L	Line voltage
$v_{\alpha r}$, $v_{\beta r}$	$\alpha - \beta$ rotor voltages
$v_{\alpha s}$, $v_{\beta s}$	$\alpha - \beta$ stator voltages
v_{as}, v_{bs}, v_{cs}	Three phase stator voltage
$V_{Nominal}$	Nominal RMS voltage
v_{qr}, v_{dr}	d-q rotor voltages
v_{qs}, v_{ds}	d-q stator voltages
X_r	Rotor inductive impedance
X_s	Stator inductive impedance

CHAPTER 1

INTRODUCTION

Due to the industrial demands for low cost, robust and low maintenance for machines, asynchronous machines are widely used in industries of pumps, fans, compressors, mills, cranes, hybrid vehicles and electrical vehicles [12]. A condition monitoring system is sometimes implemented to monitor the electrical, mechanical and thermal behaviors of the machine. At the same time, a faults diagnosis system may be applied to detect the failures and to guarantee the safety of such machines [13]. The maximum lifetime of an asynchronous machine, its ability to handle over-load and the level of its precision in a high-performance controller, are variables that largely depends on thermal stress [14]. Exceeding temperatures in different parts may result in insulation deterioration as well as rotor faults [15]. So temperature monitoring of the stator winding, the rotor cage and the stator core, can be used for thermal fault detection and predictive monitoring. It can protect the machine, extend the life span, improve the ability of over-load and contribute to the high performance of the machine. The construction of the machine can also be optimized according to the thermal behavior.

WSNs are networks that consist of many sensor nodes. A sensor node may consist of sensor(s) or actuator(s), a microprocessor, and a wireless transceiver [16]. WSNs have many applications such as the monitoring of the environment, process automation in the industry and remote medical care. With the recent advances in micro electro-mechanical systems (MEMS) technology, wireless communications, and digital electronics, WSNs are more widely used in the Internet of Things (IoT), such as Machine-to-Machine (M2M) and smart city related concepts. On the other hand, more should be considered due to the restrictions of the sensor nodes, such as low cost, low power, weak calculation power and small memory size.

1.1 Motivations and Objective

The heat of an asynchronous machine is generated from the power losses that include the losses from the stator winding, the stator core, the rotor cage, the friction and the stray

load. The losses from the stator winding and the rotor cage are copper losses while the stator core losses are iron losses. All the power losses of an asynchronous machine as an example is shown in figure 1.1:

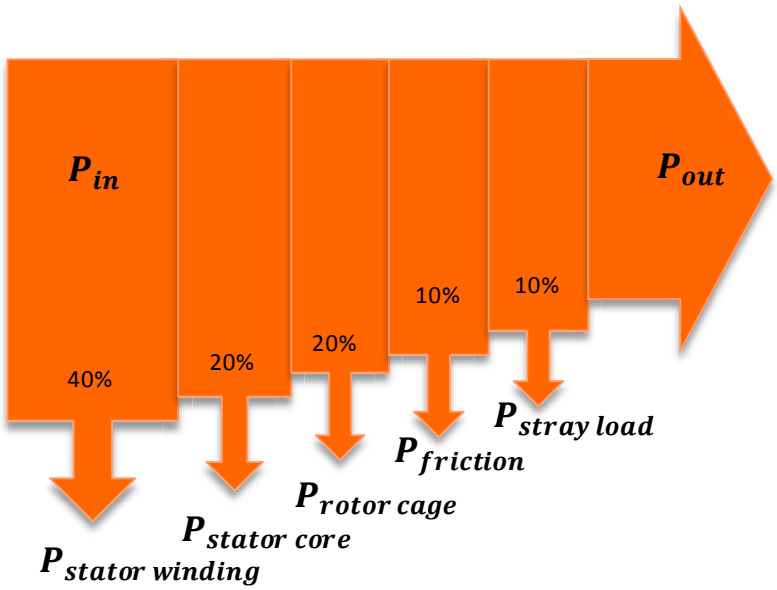


Figure 1.1: The power losses of an asynchronous machine as an example [1]

As thermal behavior is vital to the lifespan and the performance of an asynchronous machine, a temperatures monitoring system that provides high accuracy at low cost is required for the different regions.

Traditionally, the WSN may consist of many different types of sensors such as temperature, pressure, vibration, visual, infrared, (acoustic) noise, which are able to monitor a variety of conditions. The deployed wireless sensor nodes could be used for data acquisition, data processing and wireless transmission. To monitor the temperatures of an asynchronous machine, it is possible to install temperature sensors with wires to measure the temperature of the stator, but it is difficult for the rotor. In accordance with the measurement technology, WSNs can be used for temperature measurement including the rotor cage. However, the high cost and unreliability of the system make this unsuitable.

Apart from the data acquisition and data transmission of the sensor node, the remaining memory space and computation power may be sufficient for the implementation of a basic algorithm. As a result, the model-based method for the implementation of a temperatures monitoring system in WSNs is proposed. However, there are several challenges in the development of the temperatures monitoring system and the algorithm implementation in WSNs:

- Physical models of both an asynchronous machine and thermal networks
- Parameter identification and model validation
- Development of a model-based algorithm for the temperatures estimation
- Algorithm implementation in resource restricted sensor nodes, which is small memory size, weak computation power and without Float-Point-Unit (FPU), etc.

Consequently, the final objective of the research is to develop and implement a temperatures estimation system in WSNs, which could accurately estimate the temperatures of the stator winding, the rotor cage and the stator core.

1.2 State of the Art

As the thesis is a combination topic, the latest research status and the achievements of both WSNs applications and temperatures estimation technology are firstly introduced in the following sections.

1.2.1 Overview of the Applications on Wireless Sensor Networks

The WSNs [17] consist of sensor nodes that include wireless modules for untethered communication. These sensor nodes are small, of low-power, and provide sensory and data processing components. Advances in this field lead not only to less expensive wireless sensor nodes that can be very flexibly operated but also to enhancements of the traditional sensors, which conventionally needed to be connected to a central node for processing. Nodes with smart sensor capabilities are equipped with components for autonomous and unattended operations. As each sensor node generally operates independently of the network besides data transportation, new fields of application are created. As the wireless sensory technology develops, the applications for WSN increase. Based on the specific requirements, WSNs applications can be classified into three groups [18]:

1) Environmental and condition monitoring. This group generally represents the widest field of WSN application nowadays. While environmental monitoring includes the monitoring of air, water, noise as well as the sensing of fire, flood, landslide or other possible disasters, condition monitoring mainly includes the structure of the building, constructions, bridges in the field of architecture, electronic devices in power system, the electronic and mechanical sectors in industry or the health care sector in the medical field [19].

2) Object tracking. This group covers the tracking of mobile object, mainly providing both location information and the route maps. Other information about the object can also

be obtained. The tracking of rare animals by biologist and the tracking of the public transportation in the city are two of the most common applications. WSN is even used in robotic football matches to locate the position of every robot.

3) Process automation. This group provides the applications of the remote automation control via WSN. Smart home and precise agriculture are two such examples. The host sensor node generates the control strategy for the deployed wireless sensor nodes.

WSN nodes are typically organized in one of three types of network topologies, which are shown in figure 1.2. In a star topology, each node connects directly to a gateway. In a cluster tree network, each node connects to a node higher in the tree and then to the gateway, and data is routed from the lowest node on the tree to the gateway. Finally, to offer increased reliability, mesh networks feature nodes that can connect to multiple nodes in the system and pass data through the most reliable path available. This mesh link is often referred to as a router.

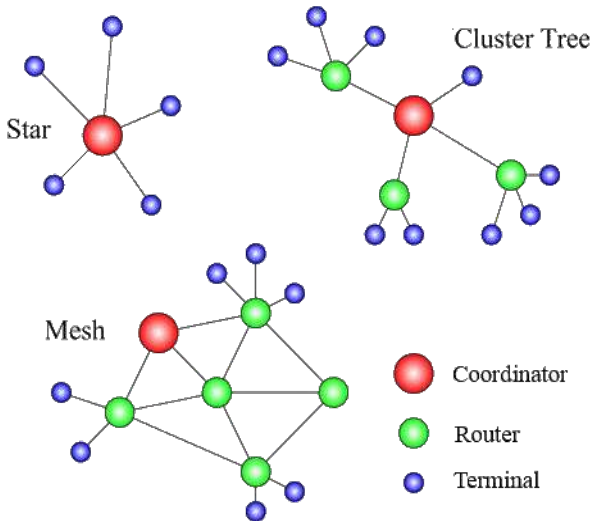


Figure 1.2: Structure of network topologies [2]

In this research, the "Star" topology will be used. There are two main reasons. One is that it is a small system which only needs several sensor nodes. Another reason is that the signals acquired and processed by separated wireless sensor nodes can be directly transmitted to another host sensor node. All the applications are based on the signal processing and wireless communication. However, the focus is on the implementation of monitoring algorithms in WSNs. As there are many restrictions described before on the sensor node, the algorithm should be simple and efficient to be implemented.

Many of the applications in electrical machines based on WSNs are condition moni-

toring systems which acquire information simply via sensors [19]. The temperature monitoring of the rotor shaft based on WSNs is reviewed in [20]. In [21], data measurement systems that measure the temperature, current and voltage of the electric machine are implemented in the wireless sensor nodes. The coordinator device reads different signals through the ZigBee module and monitor the condition of the machine based on WSNs. Wireless sensors that measure temperature and flux in an axial flux machine are described in [22].

In some cases, the measurement and the algorithm are processed in local wireless sensor nodes. Only the results are transmitted to the host sensor node. An embedded system integrated into a WSN for online dynamic torque and efficiency monitoring in induction machines is given in [23]. The values of torque and efficiency of the induction machines are estimated in the local wireless sensor node and then transmitted to the base station through the WSN. The conditions for several induction machines can be monitored with this system. In [24] a flexible filter for a synchronous angular resampling method is proposed and implemented with a wireless sensor network. The methods for nonuniform reconstruction as well as synchronous angular resampling, are implemented in the local wireless sensor node.

For other applications, the data acquisition systems are deployed in the wireless sensor nodes and the monitoring algorithm is implemented in the computer which has a powerful CPU and a large flash memory. In [25], new combined methods based on multiple wireless sensor system for real-time condition monitoring of electric machines are presented. The current and vibration signals of the machine are simultaneously read from multiple machines through wireless nodes and the faults are identified using two combined methods. Stator related faults are diagnosed by analyzing the current signals with fuzzy logic. The vibration signals are analyzed for bearing faults. However, the whole diagnosis system is implemented in the computer. Vibration-based detection by using the accelerometer in the wireless sensor node is used for the health monitoring. All the techniques used here for the analysis and processing of signals have been implemented by MATLAB software [26]. A fault diagnostics and condition monitoring of the machines using wireless technology is discussed in [27].

In conclusion, many of the monitoring applications for the electrical machine based on WSN can be found in the reference. However, none of them has attempted to implement an algorithm for the estimation of temperatures on WSNs, especially in a resource limited wireless sensor node. As such, the temperatures monitoring system of an asynchronous machine on WSN is developed.

1.2.2 Overview of the Wireless Sensor Networks based on IEEE1451

A minimum implementation of the IEEE1451 standard is implemented on a WSN using Contiki OS by the *Chair of Electronic Measurement and Diagnostic Technology* (MDT). The platform is the wireless sensor node Preon32 produced by Virtenio GmbH [11]. The WSN provides standard interfaces between different modules and different sensor nodes for the wireless communication. Data formats for storage and transmission are also defined. However, only parts of the interfaces and commands are implemented as a minimum implementation. Some measurement interfaces which are out of scope for IEE1451 are developed together with the minimum implementation as *the MDT Smart Transducer Library* (MSTL).

1.2.2.1 Overview of the IEEE1451 Standard

The Institute of Electrical and Electronics Engineers (IEEE) Instrumentation and Measurement Society's Technical Committee on Sensor Technology have developed the family of IEEE 1451 standards [3] that defines communication interfaces and protocols for distributed sensor applications. To understand the concept of the single IEEE 1451 standard, it is important to know the terms Network Capable Application Processor (NCAP) and Transducer Interface Module (TIM). The NCAP is a network node which is the master module, and is thus the controlling instance of a whole sensor network that communicates with every sensor node and performs network communication to any arbitrary network. The TIM represents a sensor node that operates a set of sensors and actuators and is responsible for the conditioning and conversion of signals. The communication entirely targets the NCAP. Both terms are described in details in the next subsection.

The family of IEEE 1451 standards [3] currently consists of seven released parts from IEEE 1451.0 to IEEE 1451.5 and IEEE 1451.7. The IEEE 1451.6 document is a proposed standard "A High-speed CANopen-based Transducer Network Interface for Intrinsically Safe and Non-intrinsically Safe Applications". The IEEE 1451.0 standard defines common commands and functionalities for sensor networks that can be applied on each NCAP and on TIMs. The IEEE 1451.1 standard is an older document originally introducing the data model used for communication between local and remote modules, as well as defining interfaces for communication among NCAPs. This interface has been largely replaced by the IEEE 1451.0 standard. The other IEEE 1451.X standards describe the commands and functionalities of each type of communication protocol and media that are applicable as defined by the regarding IEEE 1451.X standard. The IEEE 1451.2 standard dictates interfaces for wired point-to-point communication between NCAP and transducers. A multidrop communication protocol is introduced in IEEE 1451.3 for transducers to communicate with the NCAP over a shared pair of wires. The IEEE 1451.4 standard prescribes

a mixed-mode interface for analog transducers that provides analog and digital operating modes. Wireless communication protocols between NCAP and TIMs are defined in the standard IEEE 1451.5. The radio-specific interfaces defined by this standard encompass wireless standards such as IEEE 802.11 (WiFi), Bluetooth, ZigBee and 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks). The draft of the standard IEEE 1451.6 states an interface for connecting NCAP and transducers via the high-speed CANopen radio-frequency identification (RFID). The relationships among the family members of the various IEEE 1451 standards is shown in figure 1.3:

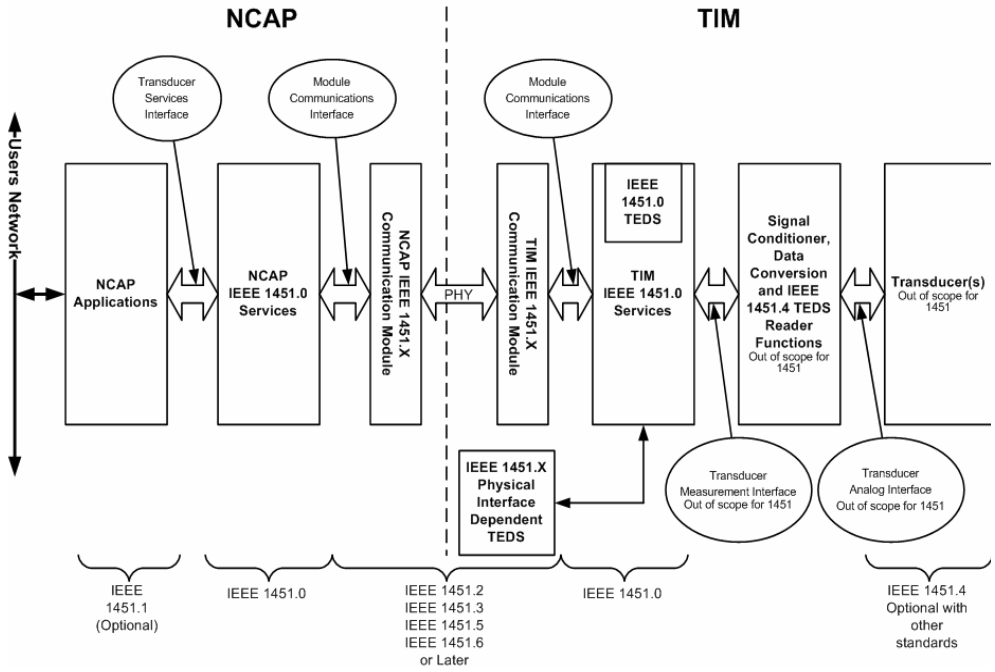


Figure 1.3: Relationships among the IEEE 1451 standard family members [3]

A. IEEE1451.0 Standard

The IEEE 1451.0 standard [3] is the basis of the entire family of IEEE 1451 standards. It defines common commands, functionalities, introduces the Transducer Electronic Data Sheet (TEDS), TIM and NCAP, all of which are introduced briefly below:

a) **TEDS** The IEEE 1451.0 defines a large set of TEDSs for many kind of purposes. The standardized formats cover the storage of information such as the identification of a sensor node, the number and names of available transducers, calibration and correction data, etc. Even if a purpose is not covered by the introduced TEDSs it is considered that manufacturers create their own manufacturer defined TEDSs. In general, TEDSs are provided by the sensor node manufacturer and are stored in the

non-volatile memory of the TIM. It is basically not intended for the WSN operator to change existing TEDSs within the TIMs.

b) **NCAP** As mentioned before, the NCAP is a network node that is the controlling instance of a whole sensor network. It is the intermediary that communicates with every sensor node on the one hand and performs network communication to any arbitrary network on the other hand. Depending on the configuration of the WSN the NCAP application not only handles the communication between the IEEE 1451.0 system and the arbitrary network, it also provides data conversion and processing functions when for instance raw sensor values need to be converted outside of the TIM. Except for locally cached TEDSs that have been received from TIMs the NCAP does not hold or provide own TEDSs.

c) **TIM** The TIM is a transducer node within the WSN which holds TEDSs describing its communication module and capabilities, timing information and all data needed for operating its transducers. Signal conditioning and conversion is part of the TIM application as well as correction and calibration functionalities if applicable. Transducers are described as *TransducerChannels* in the context of TIMs and stand either for sensors or for actuators.

B. IEEE1451.5 Standard

The IEEE 1451.5 standard [28] defines the wireless communication specifications between NCAP and TIMs. In the context of this standard TIMs are stated as Wireless Transducer Interface Modules (WTIM) due to the media of their network communication. The document introduces different communication interfaces and TEDSs that describe the physical layer for each radio type which is supported by the standard. Radio-specific configurations are presented in IEEE 802.11 which are used for WiFi connections. Bluetooth uses the transmission protocol IEEE 802.15.1. ZigBee and LoWPAN are based on the IEEE 802.15.4 communication protocol for Wireless Personal Area Networks (WPAN).

1.2.2.2 Overview of the MSTL

The implementation of data acquisition system is based on the MSTL which provides a universal interface to a variety of sensors and actuators and the implementation follows the IEEE1451 family of standards in many aspects. However, the MSTL is still a work in progress and in some areas, simplifications are made to save resources on the sensor nodes and to ease the program design.

The NCAP acts as a base station and interacts with the different TIMs present in the WSN. The wireless communication protocol between the NCAP and TIM is defined by

the IEEE1451.5 standard. The physical communication and the access control are implemented according to the 802.15.4 wireless communication standard.

A TIM can connect different types of transducers, of which the corresponding TEDS is stored in a non-volatile memory. The analog-to-digital conversion (ADC) and signal processing are performed in TIM, which is out of the scope of the IEEE1451 standard. A request for data acquisition from NCAP will trigger the IEEE 1451.5 layer in TIM, which again calls on the appropriate functions provided by the IEEE1451.0 layer to pass the request to the TIM for data acquisition. After the conversion and correction of the data, acquired data will be passed back to the IEEE 1451.5 layer of TIM, and then wirelessly transmitted to the NCAP. The figure 1.4 gives an overview of the data acquisition system using MSTL.

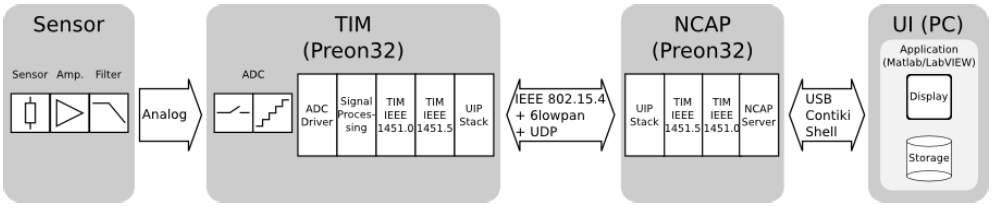


Figure 1.4: Overview of the structure for data acquisition using MSTL

The features of MSTL are listed below:

- available TIMs and transducer channels
- reading information about transducers (TEDs)
- configure properties of transducers
- reading data from transducers (single-shot)
- writing data to transducers
- stream data from transducers
- conversion of signal to SI-units on transducer (linear)
- signal processing on transducer (FIR, CIC-filter)

1.2.3 Model-based Software Development

Model-based software development is used for solving problems which is associated with designing complex control [29], signal processing [30] and communication systems. The physical models can be defined with advanced functional characteristics using continuous time and discrete time building blocks [31]. As is known, the V-model is often

used to develop software in automobile industry [32]. Due to the interface between Modelica and MATLAB/SIMULINK, model-in-loop (MiL) simulation method can be used for the development of the temperatures estimation system. During the early stage, the MiL simulation can verify the algorithm and locate the specification errors [33]. Compared to the traditional software development methodology for embedded system, the model-based method has several advantages:

- It doesn't need complex constructions and extensive software codes.
- It provides a common design environment, which facilitates general communication, data analysis, and system verification between various development groups.
- The errors can be detected and located in the early stage of the design.
- Design reuse, for upgrades and for derivative systems with expanded capabilities, is facilitated.

These models used with simulation tools, can lead to rapid prototyping, software testing, and verification. Not only are the testing and verification processes enhanced, but also, in some cases, the hardware-in-the-loop (HiL) simulation can be used with the new design paradigm to perform testing of dynamic effects on the system more quickly and much more efficiently than with traditional design methodology.

1.2.4 Temperatures Estimation Methods for Asynchronous Machines

The most common method of temperatures measurement is using mounted temperature sensor. It is possible to attach a sensor onto the surface of the stator core or on the insides of the stator winding. Measurement of local temperature, hot-spot and bulk are described in [34]. However, it is difficult to acquire signals from the rotor while it is in operation. Wireless sensor networks can be used to acquire rotor temperatures [35] [36]. In [37], a thermocouple PT1000 mounted in the rotor cage is connected to a microprocessor which is fixed in the shaft of the rotor. The temperature of the rotor cage can be transmitted wirelessly to another sensor node. Some methods using infrared sensors are described in [38] [39]. Fiber optic is used for the on-line monitoring of temperature of the rotor surface in electrical power generators [40]. Some optimized optical fiber sensors are also used for the measurement of the rotor temperature [41] [42]. The design of a rotor temperature monitoring system for contact measurement is proposed [43]. The data transmission between the rotating and stationary part is realized via infrared light. The used temperature sensors are thermocouples of type K. In short, the cost of the measurements using sensors directly is largely increased, and the instability of using these measurement system makes it unsuitable to obtain the temperature directly.

An indirect approach is the calculation of the temperature using an estimation of the resistive parameters. Based on the variation of the stator winding resistance with temperature, a sensor-less internal temperatures monitoring method for induction machine is introduced [44]. The temperature of the stator of a permanent magnet synchronous machine (PMSM) is calculated from the estimation of the stator resistances using EKF [45]. The rotor temperature can also be calculated from the resistance which is identified based on the differential equations in $\alpha\beta$-axis fixed in the stator [46]. In most of the studies [44] [47], resistances of the stator winding and the rotor cage can not be estimated simultaneously. The most frequent estimation techniques rely on a calculation of impedance at a steady state [48] [49] or using an EKF. The estimation of the stator resistance R_s and rotor resistance R_r are presented in [50] and [51]. Other research [52] [53] [54] either uses an open-loop estimator or switches algorithms under different operating conditions to simultaneously estimate the resistances of the stator and the rotor. The temperature dependence and the thermal dynamics of the stator winding are taken into consideration for the simultaneous estimation. However, the resistance of the rotor is estimated to be a constant [55].

Thermal analysis based on lumped-parameter thermal network, finite-element analysis, and computational fluid dynamics are considered in the paper [56]. Heat-transfer analysis is suitable for monitoring the thermal behavior of an asynchronous machine, whose detailed heat-transfer network is described in [57]. Finite-element analysis can only be used to model conduction heat transfer via conduction in solid components while computational fluid dynamic can be used for the prediction of the flow in complex regions [56]. However, the analysis is very demanding in terms of model setup and the time required for computation.

State estimation techniques make it possible to estimate unmeasurable state variables. A method based on the lumped-parameter thermal network is proposed for the estimation of the temperatures of the rotor and the stator using EKF [50]. The algorithm is developed based on the state-space of the asynchronous machine and the simplified thermal model together, in which it is assumed that there is no rise in temperature for the ambient. The thermal modeling of the machine is a complex multi-disciplinary problem, which must also evaluate the main internal losses of the machine [58]. Combining the electrical and mechanical models with a simplified thermal model, an approach to estimate the temperatures of the stator winding, the rotor cage and the stator core is proposed [7].

1.3 Scope and Structure

The final objective of this dissertation is to develop a temperatures estimation system of an asynchronous machine in a wireless sensor network. It also focuses on the development of a temperatures estimation algorithm using the model-based method and implementation

of an algorithm in WSNs, not including the detailed implementation of the IEEE1451 standard in the WSNs. The process of the model-based algorithm development is shown in figure 1.5.

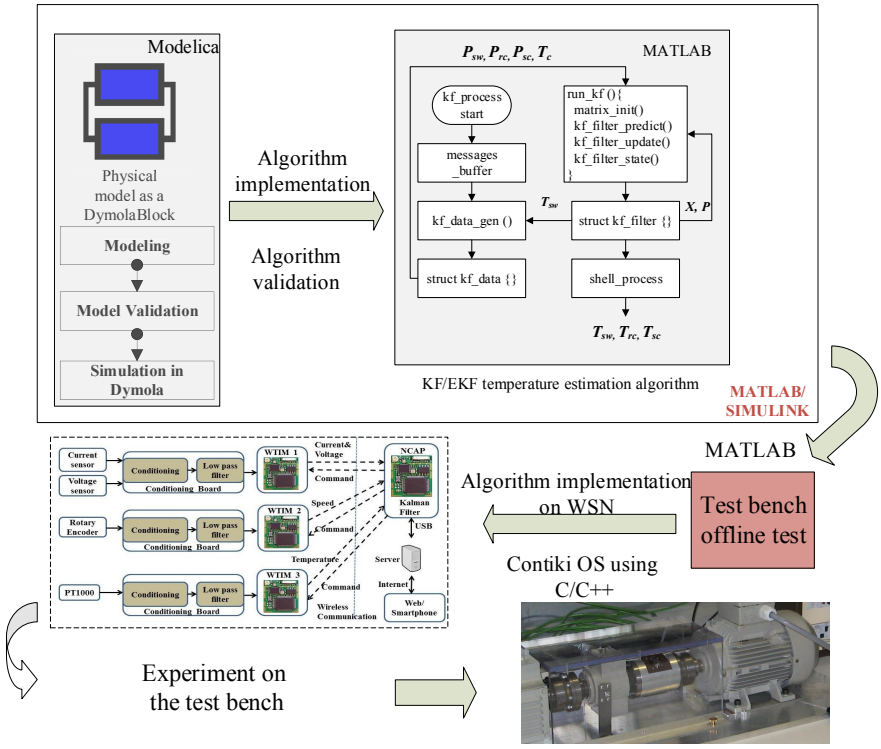


Figure 1.5: Model-based algorithm development process

The main scientific contributions of this dissertation are summarized as follows:

- **The idea of implementing a temperatures monitoring system in WSNs.** Normally the temperatures monitoring system is implemented in a controller which is powerful in computation and has a small memory size. As such, it is not as important to consider the complexity of the algorithm. However, for the implementation in WSNs, both computation complexity and memory consumption should be considered. On the other hand, this system can be used to monitor many electrical machines deployed across a large area, such as the electrical machines in factory.

- **The methods of thermal parameters identification.** The thermal behavior of an asynchronous machine can be simplified as a thermal network, which is similar to an electric circuit. The values of thermal resistance can be calculated based on the equations and some of the variables are measured or calculated from the no-load

and full-load experiments. Thermal capacitances cannot be calculated directly, so the GenOpt software is used to optimize the values of thermal capacitances until the best-fit curves are found.

- **KF and EKF algorithms are used in the estimation of temperatures.** Several methods can be used to estimate temperatures. However, both KF and EKF can accurately estimate the temperatures of the stator winding, the rotor cage and the stator core using less inputs. The KF needs only the sampling rate of 1 Hz, which can be used in many resource limited microprocessors. The EKF can even estimate the speed of the rotor and the load of the machine using only the three-phase currents and voltages, which can reduce the cost and guarantee the accuracy of the estimation. Furthermore, they are a set of mathematical equations which provide an efficient computational (recursive) means to estimate the state of a process. This means that they are the best options for an online temperatures estimator in a resource limited sensor node, because only one sampled data is needed for every computational step, which will largely reduce the consumption of the RAM memory.

- **Simulation of the physical model in Dymola and the temperatures estimation algorithm in Simulink.** Dymola provides a model of asynchronous machine with losses that can be exported to an external thermal model. The whole simulation model for temperatures monitoring in Dymola is compiled as a block in SIMULINK. When it is connected with the block of implemented KF or EKF, the system can be simulated in SIMULINK online.

- **The detailed implementation of KF and EKF in a resource restricted sensor node.** The IEEE1451 standard is implemented in WSNs using Contiki OS. Many processes are created and optimized in order to improve the efficiency of the sensor node. Many Contiki specified functions and macros are used in the implementation. The computation time for every step of the algorithm and the consumption of the memory size are accurately calculated. Fixed-point arithmetic is used as there is no FPU in the sensor node.

This dissertation is organized as follows: chapter 2 shows the construction of an asynchronous machine model with losses and a simplified thermal model based on Modelica. The parameters of the asynchronous machine and the thermal model are identified based on the test bench, and these models are validated in chapter 3. Then in chapter 4, KF and EKF are explored and implemented using MATLAB to estimate the temperatures of the machine in MATLAB/SIMULINK. Simulated temperatures from the physical model and estimated temperatures obtained using the KF or EKF are compared. Next, chapter 5 and chapter 6 elaborate on the implementation of both algorithms in WSNs. Furthermore, the estimation

results from WSNs including temperatures of the stator winding, the rotor cage and the stator core are compared with the measured temperatures in chapter 7. A conclusion and outlook is delivered at the end of this dissertation in chapter 8.

CHAPTER 2

OBJECT-ORIENTED MODELING OF AN ASYNCHRONOUS MACHINE WITH A SIMPLIFIED THERMAL MODEL

The model-based algorithm development method first depends on the physical model of the system, which can be used for the validation of the proposed temperatures estimation algorithm. The physical model is developed to simulate the characteristics of the asynchronous machine, including those of the mechanical, the electrical and the thermodynamic. This chapter will introduce the construction of the simulation model on the basis of papers [7] and [59].

The modeling language Modelica is a non-proprietary, object-oriented language for modeling of large, complex, and heterogeneous physical systems. It is suitable for multi-domain modeling such as mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented sub-components [60]. The Modelica library is very convenient and reliable, and can be used to summarize and maintain Modelica models that are developed and tested [61]. Dymola which is short for Dynamic Modeling Laboratory is one of the most popular commercial modeling and simulation environment that is based on the Modelica modeling language. The extensive Modelica library, meanwhile, contains many reusable components [62].

2.1 Asynchronous Machine Model

Two types of asynchronous machine models are provided by the Modelica library. One is the basic asynchronous machine model from the *BasicMachines* library in the Modelica standard library. The model in a library can simulate mechanical and electrical behaviors without considering the power losses of the machine. The *AdvancedMachines* library takes power losses and the variation of losses with respect to load, speed and temperature into consideration. The power losses generated by the asynchronous machine model serve as heat sources of the thermal model. Power losses can be imported into the thermal model

by an interface and thermal behaviors can hence be simulated. The combination of the two models makes it possible to calculate the temperatures of different parts, like the stator and the rotor of an asynchronous machine under a corresponding load.

2.1.1 Asynchronous Machine Model from Standard Library

The losses in the machine cause its temperature to rise. In [59], the model of a three-phase asynchronous machine with squirrel cage is used to simulate the losses in the machine. The main component of the whole asynchronous machine system is shown in figure 2.1, the module *AIMSquirrelCage* [63], which includes the squirrel cage, the stator and the mechanical parts.

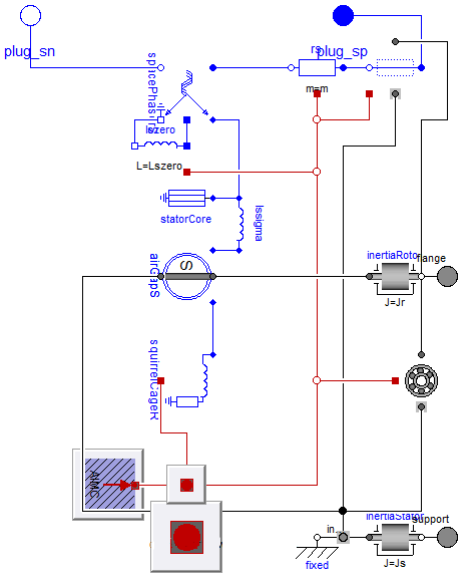


Figure 2.1: Model of an asynchronous machine in Dymola [4]

Dymola provides a library that contains plentiful models. The models of asynchronous machines in the library of *Machines* are built according to the space phasor theory. The detailed description of the standard library model can be referred to section 2.2 of [63].

2.1.2 Asynchronous Machine Model from Advanced Library

The Modelica Standard Library provides only basic machine models which takes into account the copper losses caused by constant winding resistor models, while an extension machine models with more complex factors is desired [61]. As the definition of IEEE Std-112 part 5, five types of losses are generated during the running of induction machines. These include the stator winding losses P_{sw}, the rotor cage or winding losses P_{rc}, the

frictional and the windage losses P_{fw}, the stator core losses P_{sc}, and the stray load losses P_{stray} [64].

$$P_{Loss} = P_{Input} - P_{Output} \tag{2.1}$$

$$P_{Loss} = P_{sw} + P_{rc} + P_{sc} + P_{fw} + P_{stray} \tag{2.2}$$

where P_{Loss} is the total losses of the machine, P_{Input} and P_{Output} are the input and output power of the machine respectively.

2.1.2.1 Copper Losses

Copper losses of asynchronous machines including the stator and the rotor winding or the rotor cage result from Joule heating. According to Joule's First Law, the losses are calculated by the equation (2.3):

$$P_{Loss} = I^2 \cdot R_{Operation} \tag{2.3}$$

where I is the effective current and $R_{Operation}$ is the resistance in operation. As defined in equation (2.4), it is a temperature dependent value which varies with the temperature change.

$$R_{Operation} = R_{ref} \cdot (1 + \alpha_{ref} \cdot (T_{Operation} - T_{ref})) \tag{2.4}$$

where R_{ref} is the resistance under reference temperature, α_{ref} identifies the linear temperature coefficient of the specific material, with respect to the reference temperature T_{ref}. All the reference values can be referred to Appendix table D.1

2.1.2.2 Core Losses

Core losses P_{sc} are caused by the changes in the magnetic field, which is separated into hysteresis losses and eddy currents losses. Iron sheets is built onto the iron stack to avoid undesired eddy currents [59]. The core losses are strongly related to the frequency of the supply voltage. The frequency of the stator f_s is always supply frequency which is assumed as constant while the frequency of rotor f_r is defined in equation (2.5)

$$f_r = s \cdot f_s \tag{2.5}$$

where s is slip. f_r is always much smaller than f_s. Hence, the losses of the rotor core are very small as compared to the losses of the stator core, and are usually neglected in running

conditions [65]. According to [59], the core losses in Dymola are defined in equation (2.6)

$$P_{sc} = P_{ref} \cdot \left(r_H \cdot \frac{\omega_{ref}}{\omega} + 1 - r_H \right) \cdot \left(\frac{v}{v_{ref}} \right)^2 \tag{2.6}$$

where ω is the angular frequency, v is the voltage and r_H is the ration of hysteresis losses. In the current implementation, the hysteresis losses are neglected i.e. $r_H = 0$

2.1.2.3 Friction Losses

Friction losses P_{fw} [59] are caused by the mechanical rotational losses that include friction caused by the surface of the rotor, the bearings and the windage losses from the cooling fans. All friction losses can be simplified as the production of the friction torque τ_{fw} and the rotor speed ω. As linearization around zero speed is defined by ω_{Linear}. The friction torque τ_{fw} defined in [59] can be calculated by the following equations:
For $|\omega| > \omega_{Linear}$:

$$\tau_{fw} = sign(\omega) \cdot \frac{p_{ref}}{\omega_{ref}} \cdot \left| \frac{\omega}{\omega_{ref}} \right|^{power_\omega - 1} \tag{2.7}$$

For $-\omega_{Linear} \leq \omega \leq \omega_{Linear}$:

$$\tau_{fw} = \frac{p_{ref}}{\omega_{ref}} \cdot \left(\frac{\omega_{Linear}}{\omega_{ref}} \right)^{power_\omega - 1} \cdot \left(\frac{\omega}{\omega_{Linear}} \right) \tag{2.8}$$

The dependency of friction losses on speed is modeled with the exponent $power_w$ which can be referred to [59]. The value can be referred to Appendix table D.1.

2.1.2.4 Stray Load Losses

The stray load losses of the asynchronous machine vary with the load but the values cannot be determined accurately. Such losses are caused by [66]:

- eddy currents in the rotor conductors
- short circuit in the coils under commutation
- eddy currents in the bolts and other solid parts of the rotor
- flux pulsations produced by changes in the reluctance of the magnetic path at teeth and slots, and the flux pulsations produced by currents in coils the under commutation
- distortion of the flux in the rotor produced by the reaction of the rotor

The stray load losses are difficult to be computed or measured, and are originally inspired by the standard 60034-2 [67]. The equation is defined as follows [59]:

$$\tau_{stray} = \frac{p_{ref}}{\omega_{ref}} \cdot \left(\frac{i}{I_{ref}}\right)^2 \cdot \left(\frac{\omega}{\omega_{ref}}\right)^{power_\omega - 1} \tag{2.9}$$

A reference power P_{ref} at reference I_{ref} and ω_{ref} is used to define the losses. The exponent $power_w$ has been described in section 2.1.2.3. The voltage drop is modeled as the torque acting on the shaft [64]. From the above description, it is clear that losses derived by Dymola are computationally efficient but not strictly accurate. This is because some of them are only using empirical relationships. After analyzing the results of the reference [68], a conclusion can be determined that these losses are adequate for the thermal model. However, in case of high demand of these losses, a correction, compensation or a further development of the current model should be taken into consideration.

2.1.3 Asynchronous Machine Model with Losses

To implement the physical model of the asynchronous squirrel cage machine with losses, the following parameters in table 2.1 should be determined:

Table 2.1: Parameters of an asynchronous machine

Name	Symbol
Nominal power output	$P_{Nominal}$
Nominal power factor	$Pf_{Nominal}$
Nominal frequency	$f_{Nominal}$
Nominal RMS voltage	$V_{Nominal}$
Nominal RMS current	$I_{Nominal}$
Stator and rotor resistance	R_s, R_r
Stator and rotor leakage inductance	$L_{s\sigma}, L_{r\sigma}$
Main field inductance	L_m
The number of pole pairs	p
Rotor inertia	J
Reference friction losses	$P_{frictionRef}$
Reference stator core losses	$P_{coreRef}$

Parameters $P_{Nominal}$, $Pf_{Nominal}$, $f_{Nominal}$, $V_{Nominal}$, $I_{Nominal}$ can be read from the nameplate. Others can be identified by experiments, which will be described in chapter 3. Figure 2.2 shows the whole operation system of an asynchronous machine. The AIM SquirrelCage-model connected in star is electrically connected to a three phase sinusoidal voltage source from the *MultiPhase* library. On the other side the machine is connected to

a load consisting of an inertia and a load torque which is quadratically dependent on speed. The inertia is the machine inertia, and the load torque is controlled with a PI controller to achieve the desired mechanical output. Most significantly, the thermal port of the machine can be enabled, and then connected to the thermal port of the thermal model.

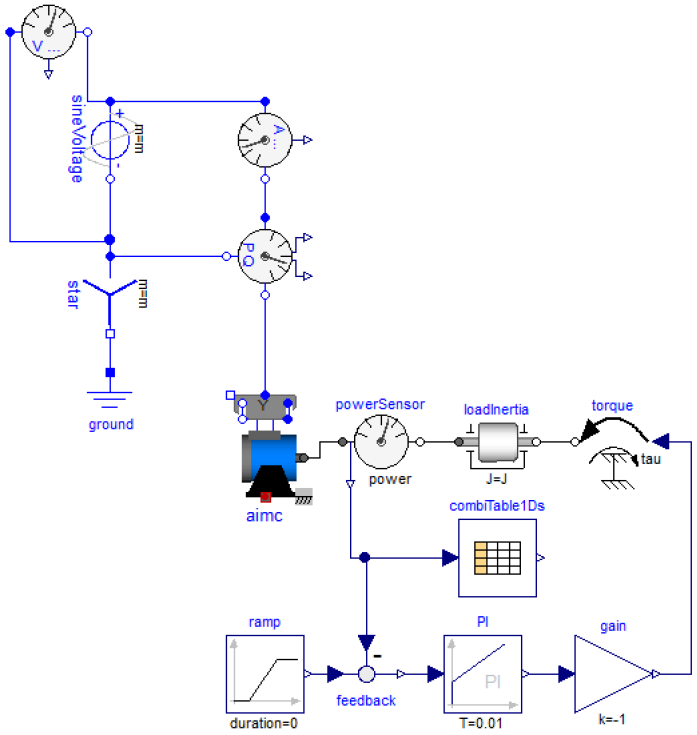


Figure 2.2: Simulation model of AIMC [5]

2.2 Thermal Model of an Asynchronous Machine

The thermal analysis of the electrical machine can be divided into two basic types: analytical lumped-circuit and numerical methods. In its most basic form, the lumped-circuit is analogous to an electrical network, and the analysis consists of the calculation of conduction, convection, and radiation resistances for different parts of the machine construction [56].

2.2.1 Introduction of the Thermal Model

In the thermal network theory, the object is divided into basic thermal elements, which consist of usually one node for every part and several thermal resistances for every surface of the part. These elements are then linked together to form a network of nodes and thermal

resistances. Using the similarity between the electrical and thermal networks, the analysis of the thermal behavior of the object can be simplified.

The thermal system and the electrical system are analogous because they both have similar equation and boundary conditions [69]. The 1th Kirchhoff law of the electrical network is similar to the rule of the thermal network, both of which reveals the conservation of energy. The 2nd Kirchhoff law is derived from the existence of the electrostatic potential, which can be substituted with the temperature potential in the thermal network. The analogous systems are shown in table 2.2.

Table 2.2: Similarity of thermal and electrical system

Thermal System			Electrical System		
Quantity	Symbol	Unit	Quantity	Symbol	Unit
Heat Flow	$\dot{Q}$	W	Current	I	A
Temperature Difference	ΔT	K	Voltage	U	V
Temperature	T	K	Potential	E	V
Thermal Power	P_θ	W	Power	P	W
Thermal Resistance	R_θ	$\frac{K}{W}$	Resistance	R	Ω
Thermal Conductance	G	$\frac{W}{K}$	Conductance	$\frac{1}{R}$	$\frac{1}{\Omega}$
Thermal Capacitance	C_θ	$\frac{J}{K}$	Capacitance	C	$\frac{As}{V}$

The thermal network is analogous to an electrical network, as temperature to voltage, power to current, thermal resistance to electrical resistance and thermal capacitance to electrical capacitance. The heat sources of the machine are the losses of the power, which include the losses of the stator winding, the rotor cage, the stator core, the rotor core, the friction losses and the stray load losses [61]. There are only negligible core losses in a squirrel cage rotor operated at low slip frequency, that can normally be ignored. Friction and windage losses can be seen as a minor additional load. At constant speed, they can be added to the stator core losses. The stray load losses vary with the square of current and thus can be modeled as a small increase proportional to the stator copper losses. Three other losses generate the simplified thermal network of the machine in figure 2.3 [6]:

The whole thermal model is constructed as shown in figure 2.4, which is described in [7]. The detailed geometrical data of the machine is not available from the manufacturer, as such, it is much more proper to apply this simplified thermal model instead of a detailed one. This is connected to the model of the machine by the red part on the top of the whole model, which transmits the power losses inside the thermal model. This model could represent the three relevant temperatures of the asynchronous machine: the temperatures of the stator winding, the rotor cage and the stator core. These three temperatures can be described in details as:

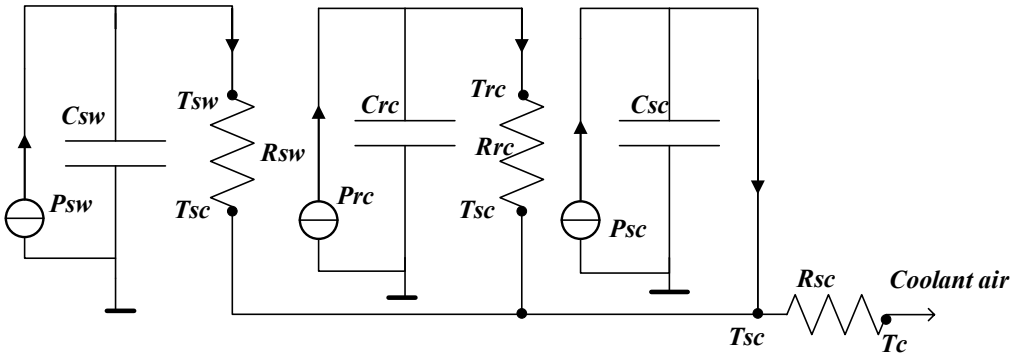


Figure 2.3: Thermal network of an asynchronous machine [6]

- T_{sw}: the temperature of the stator winding, averaged over the slot and end winding region

- T_{rc}: the temperature of the stator core, averaged over the whole cross-section of the stator core

- T_{sc}: the temperature of the rotor cage, averaged over the slot and end ring region

The three regions are represented by the thermal capacitor that is connected to the thermal conductor, which are the most significant components in the model. The heat transfer from Modelica's library, will be introduced in section 2.2.2. A coolant system in section 2.2.3 is used to model the fan of the machine, which takes into account the thermal capacity and cools the whole system.

2.2.2 Heat Transfer

The models of the thermal capacitor and the conductor are shown in figure 2.5. The thermal capacitance is the ability of a material to store heat energy. It is the measure of temperature change in a material based on its volume. The thermal conductance is the reciprocal of thermal resistance which is the ability of a material to resist the flow of heat. The equations (2.10) and (2.11) define the two parameters. Three capacitors and conductors are connected to represent the heat flow among three parts of the machine, i.e. the stator winding, the rotor cage and the stator core.

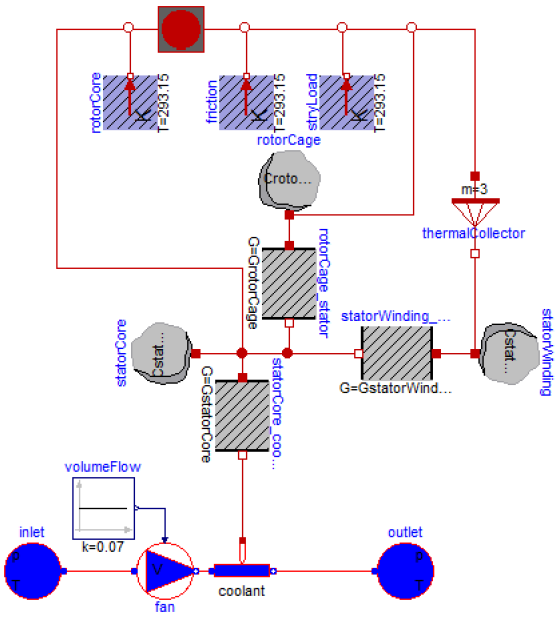


Figure 2.4: Thermal model in Dymola [7]

(a) Thermal capacitor [70]

(b) Thermal conductor [71]

Figure 2.5: Thermal capacitor and conductor in Dymola

$$C = \frac{\Delta Q}{\Delta T} \tag{2.10}$$

$$G = \frac{\dot{Q}}{\Delta T} \tag{2.11}$$

where:

C is the thermal capacitance,

G is the thermal conductance,
ΔQ is the heat difference,
$\dot{Q}$ is the heat flow,
ΔT is the temperature difference.

2.2.3 Coolant System

Besides the thermal model of these three parts, the coolant system is also one part of the model. It is an air cooling system comprised of a fan mounted to the end of the shaft. This is one of the most significant aspects of designing a machine as the cooling of the asynchronous machine can improve the design of the machine, as well as reduce the size and mass of it. This system only models a simple coolant system, which takes into account the basic thermal dynamic effects, i.e. the heat transported by the media's thermal capacity.

The power losses from the asynchronous machine are just like the current source of an electrical circuit. The thermal capacitors represent the ability of a material to store heat energy while the thermal resistor represent the heat conduction.

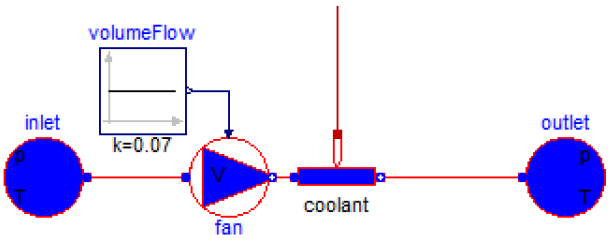


Figure 2.6: Coolant system [7]

The temperature drop is influenced not only by the rise in temperature of the machine but also by the ambient medium. The library supplies three options, air at 30 $°C$, air at 70 $°C$ and water. According to the realistic working environment of the asynchronous machine, air at 30 $°C$ is the most befitting approximation for the ambient medium. The ambient temperature is the initial temperature of the air. Another parameter that must be decided in advance is the volume flow, which represents the wind power generated by the fan. This is determined by the end temperature of the coolant system, which is defined as the temperature of the air inside the coolant, between the fan and the surface of the machine (refer to the arrow in figure 2.7). The final temperature of the coolant air at 35.6 $°C$ is measured in section 3.1.2. From the simulation model, the *volumeFlow* is extrapolated from the experiment as a constant 0.07 m^3/s, and the temperature of the coolant air in the simulation model is simulated.

The parameters are summarized in table 2.3.

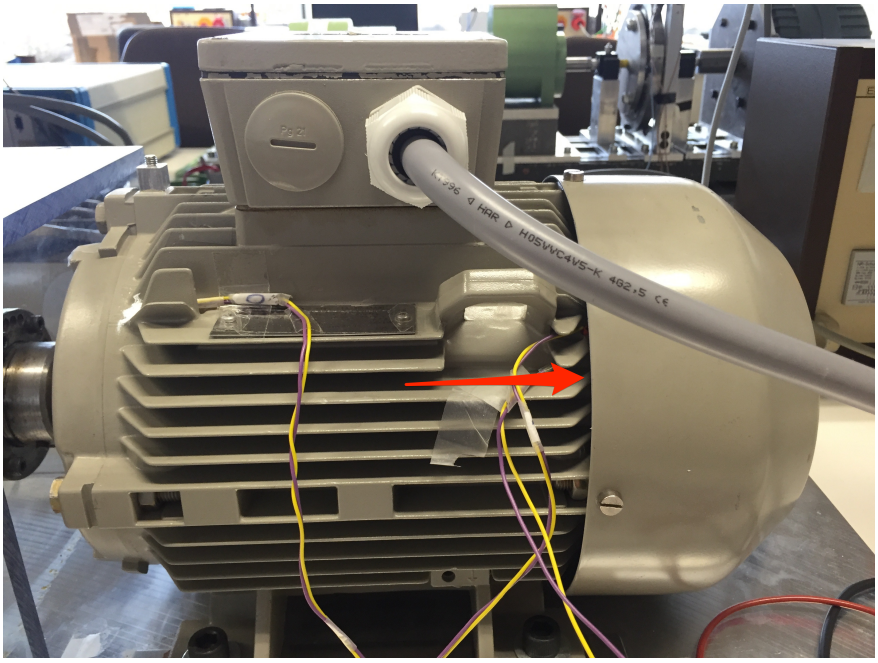


Figure 2.7: Definition of the temperature of the coolant system

Table 2.3: Parameters of coolant system

Description	Value
Ambient medium	Properties of air at 30 $°C$
Ambient temperature	Initial temperature of the air
Volume flow	$0.07\ m^3/s$

2.3 Complete Simulation Model

By enabling *useThermalPort* in **AIMC**, the thermal model is connected to the asynchronous machine model. The **AIMC** module gives the power losses of the asynchronous machine as outputs and they act as the heat sources of the thermal model. The whole simulation system in Dymola is shown in figure 2.8:

2.4 Conclusions

This chapter mainly presents the structure of the simulation model. The first part shows the model of the asynchronous machine in Dymola. This model emit losses from the machine, which are used as heat sources in the thermal model. Then a thermal model consisting of thermal conductors, thermal capacitors and coolant system is constructed.

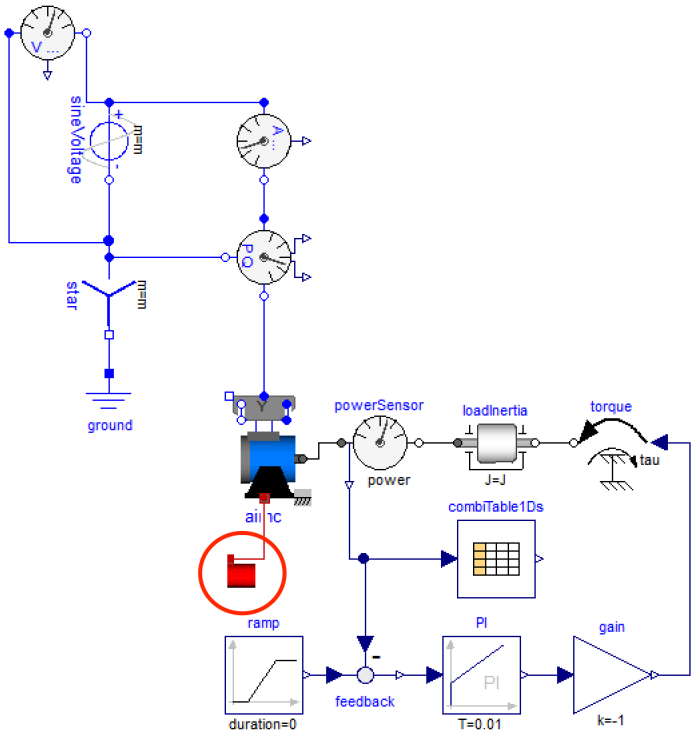


Figure 2.8: The whole simulation system in Dymola

Each individual part is introduced and then elaborated in this chapter. The last part of this chapter introduces the connection between the model of the machine and the thermal model.

CHAPTER 3

PARAMETER IDENTIFICATION OF THE MODEL

To simulate the thermal behavior of the asynchronous machine, parameters of the machine and the thermal model should be identified. This chapter will elaborate on several methods which can be used to identify the parameters.

3.1 Parameter Identification

The parameters of the model should be first identified based on the test bench, so that the electrical, mechanical and thermal behavior can perform properly. Two tests are performed, one is no-load test and the other is load test. The aim of the no-load test is to obtain the friction losses and the stator core losses, which are also necessary for obtaining other kinds of power losses in the load test. The load test provides the end temperature and stable power losses under a full-load condition, which are used for the deduction of the thermal conductances. At the end of this chapter, the method of receiving intrinsic parameters in an asynchronous machine model is discussed and the asynchronous machine model is validated with respect to different characteristic curves.

3.1.1 No-load Test

The temperature rise inside the asynchronous machine is related to the power losses of different components. The machine model takes the following losses into account:

- P_{sw}: heat losses in the temperature dependent resistances of the stator winding
- P_{rc}: heat losses in the temperature dependent resistances of the cage
- P_{sc}: stator core losses
- P_{fw}: friction and windage losses

The rotor core losses are nearly zero during normal operation, and the stray load losses are dissipated for the sake of simplification. The first two losses used for identifying the

thermal model parameter are obtained by the load test, while the friction losses depend on the speed. Therefore, P_{fw} is received by the no-load test, which is also a parameter in the model of the machine. Otherwise, the core losses can also be determined by the no-load test using the segregated loss method described in the IEEE Std. 112 test document [64].

"This test is performed by running the machine as a motor at rated voltage and frequency with no connected load. When separation of no-load losses is to be accomplished, run this test and read temperature, voltage, current, and power input at the rated frequency and at at voltages ranging from 125% of the rated voltage down to the point where further voltage reduction will increase the current." [64]

Due to the restriction of the laboratory's condition, the voltage of the machine cannot be changed. The input frequency can be changed according to a desired speed. One of the methods that determine the no-load power curve is to vary the frequency across a small range, run a test at several points and then plot the resulting power versus voltage curve [64]. At each frequency reading, the value of the instantaneous current and voltage are measured.

The friction losses of some machines may change until the bearings reach stable status. "Stabilization is considered to have occurred whenever the power input at no-load does not vary by more than 3% between two successive readings at the same voltage taken at half-hour intervals." [64] However, the resistance of stator will change correspondingly because of the rise in temperature. Therefore this test is proceeded as quickly as possible so as to maintain the resistor of the stator and the rotor. The machine is assumed to work on a stable condition when the measured speed of the rotor does not change. All the measured and calculated data in the no-load test are summarized in table 3.1.

Table 3.1: Measured and calculated data in no-load test

Measured	**Calculated**
Instantaneous current $i(t)$	Effective current I
Instantaneous voltage $u(t)$	Effective voltage U
Torque τ	Stator core loss P_{sc}
Angular speed w	Friction and windage loss P_{fw}
	Stator winding loss P_{sw}
	Rotor cage loss P_{rc}
	Input power P_{in}

3.1.1.1 Input Power

The input power is derived by measurements of the input current and voltage. In reality the three phases are unsymmetrical, such that the current and voltage from each phase must be measured. The total input power is the sum of the input power from each phase:

$$P_{in} = P_U + P_V + P_W \tag{3.1}$$

where P_U, P_V, P_W is the input power per phase.

The single-phase input power P can be determined by the numerical of instantaneous current and voltage:

$$P = \frac{1}{n}\sum_{k=1}^{n} u_k i_k \tag{3.2}$$

where n is the number of measured samples. As the frequency of i and u is 50 Hz, sampling rate is 2,000 Hz, n is selected to 120, which the time period is three times same as the cycle time of i and u.

3.1.1.2 Stator Losses

For a three-phase machine, the ohmic stator losses P_{sw} is shown in equation (3.3).

$$P_{sw} = 3I^2 R_s \tag{3.3}$$

where I is the value of the measured or calculated effective current per line terminal, in amperes (A). R_s is the per phase dc resistance of the stator, in ohms. By measuring the instantaneous current, the effective value can be calculated as below:

$$I = \sqrt{\frac{1}{n}\sum_{k=1}^{n} i_k^2} \tag{3.4}$$

where n is the number of measured samples. The value is also 120 with the same reason as described in section 3.1.1.1.

3.1.1.3 Friction Losses

The main task of the no-load test is to determine the friction losses. The friction and windage losses P_{fw} are calculated from a linear regression analysis using some lower points of the power versus voltage squared curve [64]. To determine the friction and windage losses, the input power P_{in} is subtracted from the stator I^2R losses P_{sw} at each

of the voltage test points. The remaining power P_{sub} can be defined as:

$$P_{sub} = P_{in} - P_{sw} \tag{3.5}$$

The remaining power P_{sub} versus supply voltage U are plotted and the curve is extrapolated to zero voltage. The intercept of the curve with the zero voltage axis is the value of the friction and windage losses. This intercept may be determined more accurately if the input power minus stator I^2R losses is plotted against the voltage squared for values in the lower voltage range [64]. However, there is no voltage control of the input voltage in the lab, thus an approximation method is employed to alter the desired speed to get a series points of (U,P_{sub}), which can fit a regression curve for deciding P_{fw}. The x axis is not the actual voltage value of the machine. It is the voltage of the controller which can change the frequency of the machine. It requires a constant frequency and variable voltage from an adjustable transformer. Since frequency converter is used for the test bench, we have to set various frequency within small differences. In this way, we can get an approximate value.

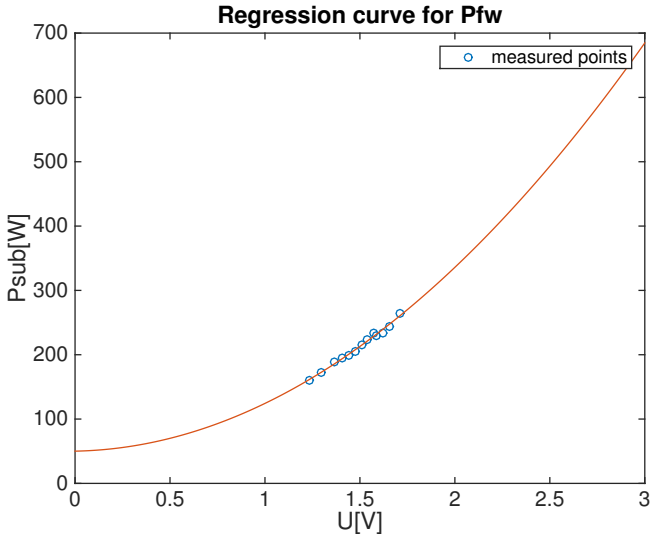


Figure 3.1: Regression curve of P_{sub} and U

From figure 3.1 the value of P_{sw} can be read as the intercept of y-axis. However as a consequence of the approximation method, when values of different data measurements are used to fit the curve, the result is not accurate. To find the best fitting result, a histogram of results obtained by different combinations of measured data is used to determine the friction losses as 50 W.

Another way to determine the friction losses is to use iron losses constant as shown in

the following equation:

$$w^2 \cdot k_{iron} + P_{fw} + P_{sw} = P_{in} \tag{3.6}$$

where k_{iron} is the iron losses constant and w is the measured speed. In this equation there are only two unknown constant parameters, which are the iron losses constant k_{iron} and the friction and windage losses P_{fw}. Only two equations are needed to solve for the unknown parameters, whose coefficients are picked from two different columns of the no-load test table. However, the accuracy of this method is imprecise compared to the first method as only two equations are used.

3.1.1.4 Stator Core Losses

The stator core loss P_{sc} at each test point voltage is obtained by subtracting the value of friction and windage losses P_{fw} which are determined in section 3.1.1.3 from the input power minus stator winding losses which are determined in section 3.1.1.2.

$$P_{sc} = P_{in} - P_{sw} - P_{fw} \tag{3.7}$$

3.1.1.5 Rotor Cage Losses

The rotor cage losses P_{rc} can be determined from the slip using equation (3.8) or equation (3.9). The rotor losses are determined by the equation (3.8).

$$P_{mrc} = (P_{in} - P_{sw} - P_{sc}) \cdot s \tag{3.8}$$

$$P_{grc} = (P_{out} + P_{sw} + P_{sc}) \cdot s \tag{3.9}$$

where P_{mrc} are the machine rotor losses, P_{grc} are the generator rotor losses, s is the slip which is defined in equation (3.11). The slip speed defined in equation (3.10) is the difference between the value of the synchronous and measured speed in r/min.

$$\text{slip speed} = n_s - n_t \tag{3.10}$$

where
n_s is the synchronous speed, in r/min,
n_t is the measured speed, in r/min,

Slip expressed as a per unit quantity is

$$s = \frac{\text{slip speed}}{\text{synchronous speed}} \tag{3.11}$$

The friction, windage losses and the core losses will be determined by using the measured data (section 3.1.1.3 and section 3.1.1.4). The power losses in the no-load test is summarized in table 3.2.

Table 3.2: Parameters of asynchronous machine

Name	Symbol	Value
stator core losses	P_{sc}	$158.1\,W$
input power	P_{in}	$328.7\,W$
friction and windage losses	P_{fw}	$50\,W$
stator winding losses	P_{sw}	$120.6\,W$

3.1.2 Load Test

As mentioned above, this model represents the three relevant temperatures of the asynchronous machine, the temperatures of the stator winding, the rotor cage and the stator core, which are modeled by conductors and capacitors. The aim of the load test is to determine the values of thermal conductances and capacitances in the model. As shown in the paper [7], the equations describing the system in figure 2.3 are:

$$C_{sw}\frac{dT_{sw}}{dt} = P_{sw} - G_{sw}(T_{sw} - T_{sc}) \tag{3.12}$$

$$C_{rc}\frac{dT_{rc}}{dt} = P_{rc} - G_{rc}(T_{rc} - T_{sc}) \tag{3.13}$$

$$C_{sc}\frac{dT_{sc}}{dt} = P_{sc} + G_{sw}(T_{sw} - T_{sc}) + G_{rc}(T_{rc} - T_{sc}) - G_{sc}(T_{sc} - T_c) \tag{3.14}$$

After stabilization of heat exchanges among the air and the components of the machine, the derivatives of temperature with respect to time become zero. By measuring the end temperature of T_{sw}, T_{rc}, T_{sc}, and T_c and calculating of the power losses in the machine the conductances can be derived:

$$P_{sw} = G_{sw}(T_{sw} - T_{sc}) \tag{3.15}$$

$$P_{rc} = G_{rc}(T_{rc} - T_{sc}) \tag{3.16}$$

$$P_{sw} + P_{sc} + P_{rc} = G_{sc}(T_{sc} - T_c) \tag{3.17}$$

The stator core losses have been determined in section 3.1.1.4. The losses of the stator winding and the rotor cage can be calculated by applying equations (3.3) and (3.8). These

are summarized in section 3.1.2.2. All the measured and calculated data in the load test are summarized in table 3.3.

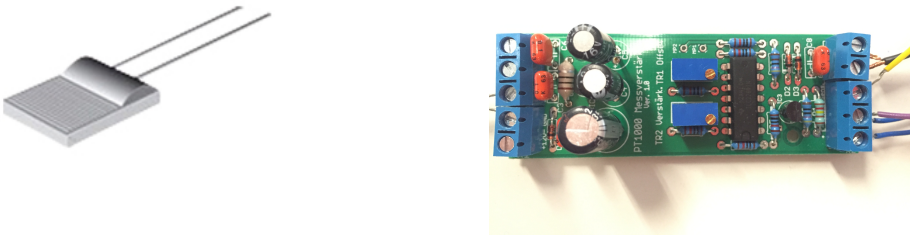
Table 3.3: Measured and calculated data in load test

Measured	**Calculated**
Instantaneous current $i(t)$	Input power P_{in}
Instantaneous voltage $u(t)$	Nominal output power P_{mech}
Torque τ	Stator core losses P_{sc}
Angular speed w	Stator winding losses P_{sw}
Rotor cage temperature T_{rc}	Rotor cage losses P_{rc}
Stator winding temperature T_{sw}	Thermal conductance G_{sw}
Stator core temperature T_{sc}	Thermal conductance G_{sc}
Coolant temperature T_c	Thermal conductance G_{rc}

3.1.2.1 Temperature Test

To solve the equations (3.15) - (3.17), the end temperature of each part of the machine should be measured. Furthermore, the whole temperature rise curve will be used to find the values of the thermal capacitors according to the best fitting curve of temperature. These three temperatures are recorded by MATLAB and the measurement method is described in the paper [72].

The temperatures of the stator winding and the stator core are measured by PT 1000 which is shown in figure 3.2(a). This is a platinum resistance temperature thermometer that can measure temperature between the range of $0\,^{\circ}C$ to $250\,^{\circ}C$ based on the data sheet. The signal is processed by a conditioning board in figure 3.2(b) and acquired by the data acquisition board from National Instruments (NI PCI-6023E).



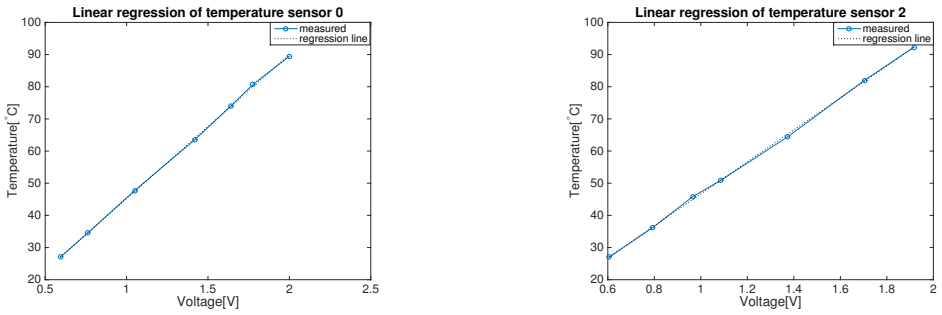
(a) PT1000 (b) Instrument transformer

Figure 3.2: PT1000 instrument transformer

To obtain the equation for the relationship between temperature and voltage, the sensor must first be calibrated by measuring the corresponding output voltage of the transformer under different temperatures. The regression results of the sensors are plotted in figure 3.3 and the regression equations are shown in equations (3.18) - (3.19).

$$T_{sc} = 44.7082°C/V \cdot U + 0.5531°C \tag{3.18}$$

$$T_{sw} = 49.6995°C/V \cdot U - 2.996°C \tag{3.19}$$



(a) Sensor for stator core

(b) Sensor for stator winding

Figure 3.3: Linear regression of temperature sensors

One $PT1000$ sensor for the stator winding temperature T_{sw} is embedded among the winding of the machine, which is shown in figure 3.4. The data can be transmitted via a wire which goes through the hole punched onto the surface of the machine. Another sensor is also embedded into the half drilled-through hole and fixed with white thermal grease, making it closer to the internal temperature of the stator core. The location of the sensor for the temperature of the stator core is shown in figure 3.5.

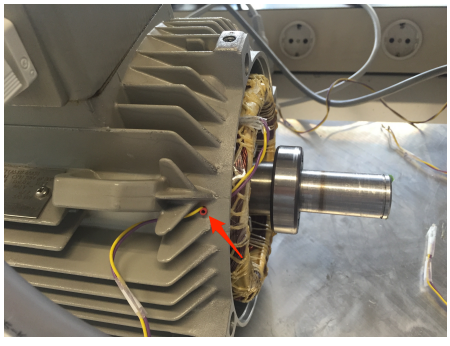


Figure 3.4: Location of stator winding temperature sensor

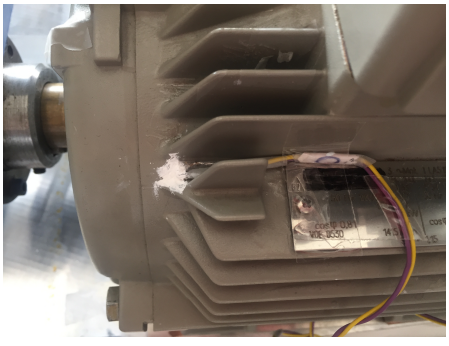


Figure 3.5: Location of stator core temperature sensor

The measurement of the rotor cage temperature is based on a master thesis [37]. As the

rotor is a fast rotation part, the traditional DAQ system that has wires cannot be used. A wireless sensor network is proposed for the measurement of the temperature of the rotor cage. A PT1000 sensor is fixed on the ring of the cage, and a wire is passed through the shaft to the outside of the machine. The internal structure of the rotor cage is shown in figure 3.6:

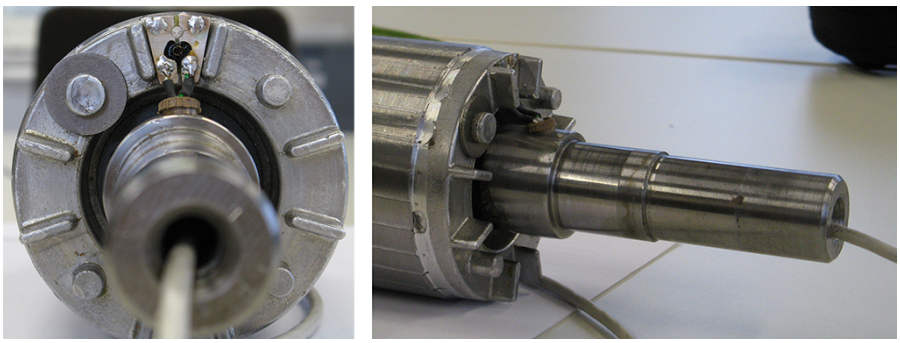


Figure 3.6: The installation of the sensor in the rotor cage

The other end of the PT1000 sensor, which is connected to a conditioning board, is integrated with a wireless sensor node (Preon32). The conditioning board, which is powered by a chargeable battery and mounted inside of the shaft, rotates along with the rotor. The installation of the conditioning board is shown in figure 3.7:

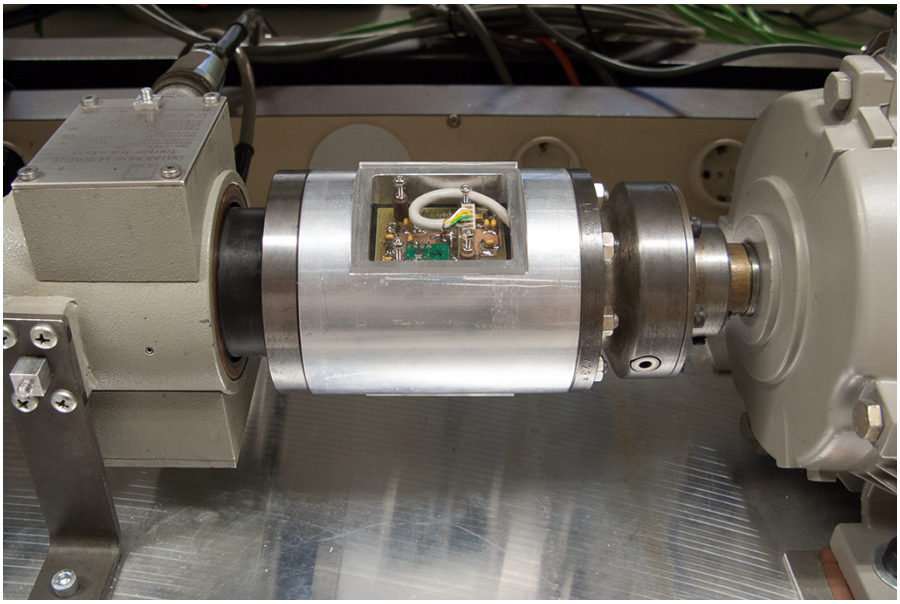


Figure 3.7: The installation of the conditioning board

The acquired temperature signal will be periodically transmitted by the wireless sensor

node to the host sensor node which is connected to a computer. The data will be processed and stored in the computer.

The load test with full load 20 N.m lasts for about three hours so that the heat exchange remains stable. The resulting temperature curves display much disturbances and noises, which should be filtered to smooth the curve so as to facilitate a comparison with the simulated one. A Gaussian window of length 64 is created and the temperature data are combined with this Gaussian function to filter off the noises. The figures on the left side of figure 3.8 are the original temperature curves, which are filtered by FIR filter. Meanwhile, the resulting temperature curves are shown on the right side of figure 3.8. The original temperature curve of rotor cage is already smooth enough, because the data is filtered by an anti-aliasing filter before storage. Thus, the Gaussian window does not exert much effects on the remaining tiny noises of the curve. The end temperatures of the three parts are summarized in table 3.4

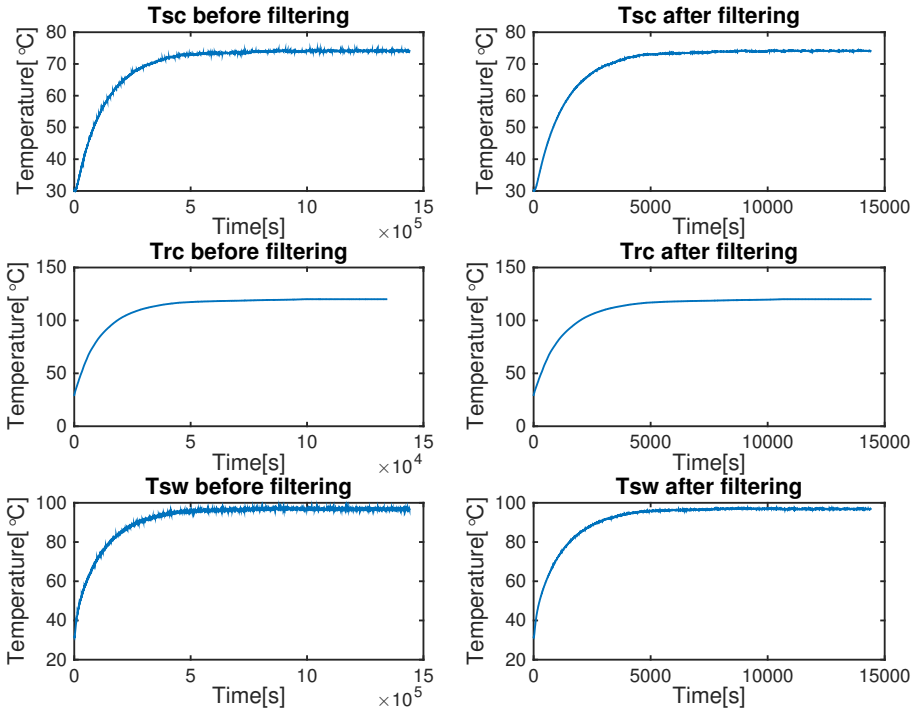


Figure 3.8: Measured temperatures before and after filtering

Table 3.4: The end temperatures of the three parts

Name	Temperature
T_{sc} $[°C]$	74.0872
T_{sw} $[°C]$	96.9135
T_{rc} $[°C]$	120.0537

3.1.2.2 Power of Steady-state

When the machine is stabilized and all the temperatures have been measured, the values of the instantaneous input current and voltage are measured using the same measurement device. The sampling frequency is set to 10000 Hz. The power losses of the load test are calculated in a similar way to the losses of the no-load test introduced in section 3.1.1. The losses of the stator core is derived from the no-load test. The losses of the stator winding and the rotor cage are received according to equations (3.3) and (3.8), and by measuring the values of the instantaneous input current and voltage.

The nominal output power P_{mech} is strictly connected to the power losses inside the motor. In the reference nominal power table the nominal output power is 3 kW. However, after comparing the values of the instantaneous input current with the simulated current, it is found that the simulated current is relatively large, which will cause a temperature rise in each component. Therefore, the real output power is measured by a group of measurements (HBM T30FM) with a nominal load on the asynchronous machine. The output mechanical power under nominal condition can be obtained with measured values of speed and torque:

$$P_{out} = \tau \cdot \omega \tag{3.20}$$

where τ is the measured torque, in $N \cdot m$, w is the angular speed, in rad/s. All the power losses of the load test are shown below in table 3.5:

Table 3.5: Power losses of load test

Name	Value
P_{in} $[W]$	3127.2
P_{sc} $[W]$	158.1
P_{sw} $[W]$	263.3
P_{rc} $[W]$	125.8
P_{mech} $[W]$	2580

3.2 Parameters Identification of the Asynchronous Machine

The parameters of the asynchronous machine are identified in [72]. However, resistances are taken to be constants because the influence of temperature changes on the resistors has been neglected. For the asynchronous machine model with losses, the temperature-dependent characteristic of the resistor must be considered. Thus the ambient temperature is supposed to be $25°C$. The values of the stator and rotor leakage inductance are assumed to be the same by providing the same leakage coefficients σ. The inductive related parameters of the asynchronous machine model in Dymola are main field inductance L_m, stator inductance L_s and rotor inductance L_r which can be defined as equations (3.21) - (3.23):

$$1 - \sigma = \frac{L_m^2}{L_s \cdot L_r} \tag{3.21}$$

$$L_s = (1 + \sigma_s) \cdot L_m \tag{3.22}$$

$$L_r = (1 + \sigma_r) \cdot L_m \tag{3.23}$$

where both σ_s and σ_r are equal to σ. The inductive impedances can be computed as shown in the following equations:

$$X_s = 2\pi f L_s \tag{3.24}$$

$$X_r = 2\pi f L_r \tag{3.25}$$

All the parameters are summarized in table 3.6.

Table 3.6: Intrinsic parameters of asynchronous machine

Name	Symbol	Value
stator resistance [Ω]	R_s	1.9693
rotor resistance [Ω]	R_r	1.8081
leakage inductance coefficient	σ	0.0736
main field inductance $[mH]$	L_m	160.26
stator inductance $[mH]$	L_s	172.06
rotor inductance $[mH]$	L_r	172.06
the number of pole pairs	p	2
rotor inertia $[kg \cdot m^2]$	J	0.01654

The main task of this section is to obtain the power losses of the actual machine in the laboratory, losses of the input power, the stator winding, the rotor cage, the stator core,

the friction and windage, under conditions of the no-load and load respectively. Another issue that is addressed in this chapter is the pre-given intrinsic parameters of the models. These parameters are obtained from a student's work which is described in the paper [72]. However, these parameters defined in the model do not coincide with the conventional definitions. As such, the given parameters in [72] should be transformed and calculated to obtain the proper ones. The thermal model can be designed and connected to a good performance model of the asynchronous machine.

3.3 Parameters Identification of the Thermal Model

To simulate the system properly, the thermal parameters of thermal capacitances and conductances must be indentified. These factors affect the end temperature and the rise trend of the temperature curves. Thermal conductances are easily obtained through calculation while the thermal capacitances are identified by GenOpt [7] using the best-fitting curve method.

3.3.1 Thermal Conductances

To determine the thermal conductances in the model, a load test lasting for four hours is conducted and the temperatures of the stator core, the stator winding and the rotor cage are measured in section 3.1.2. The measurements are obtained using PT 1000 temperature sensor, which connected to an instrument transformer. The equation between the temperature and the voltage signal given by transformer are deduced through calibration of the sensor and the DAQ board provided by National Instruments. From equations (3.15) - (3.17) and the results obtained in section 3.1.2, the thermal conductances can be calculated.

Table 3.7: Thermal conductances

Name	Value
$G_{sc}\,[W/K]$	14.0956
$G_{sw}\,[W/K]$	11.6381
$G_{rc}\,[W/K]$	2.7395

3.3.2 Thermal Capacitances

The thermal capacitances cannot be computed directly from the equations (3.12) (3.13) (3.14). As a result, it is possible to utilize optimization methods to determine best-fit thermal capacitances using GenOpt.

3.3.2.1 Interface between Genopt and Dymola

GenOpt is designed for obtaining the values of user-selected parameters that help to minimize an objective function, leading to the best operation of a given system. The objective function is calculated using an external simulation program that is iteratively called by GenOpt, such as Dymola. It can also identify unknown parameters in a data-fitting process, which helps to perform the task. The optimization flowchart is shown in the figure 3.9. The purpose of the red block is to obtain the value of the minimized objective function.

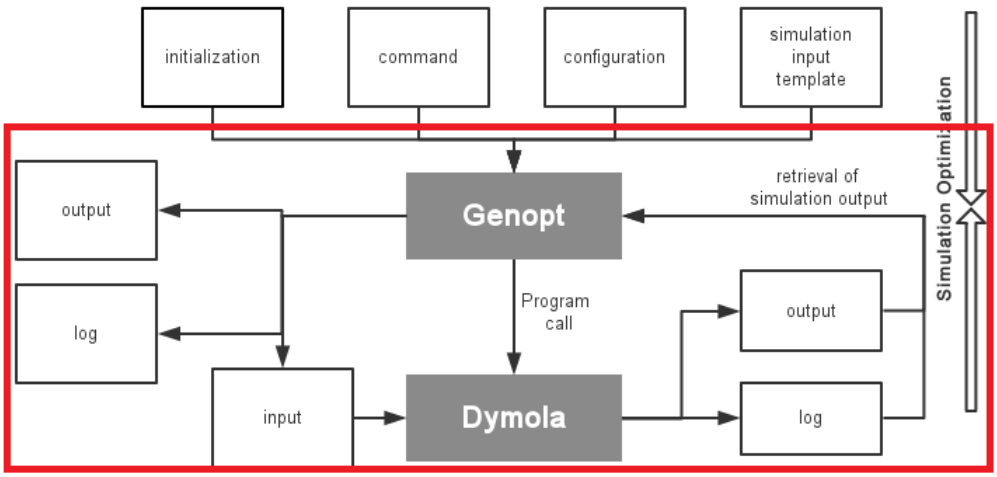


Figure 3.9: Flowchart of running GenOpt with Dymola [8]

In order that Genopt works properly, four input files are required:

- **Initialization file**, specifications of all files' names and locations
- **Configuration file**, the error indicator and the start command
- **Command file**, desired identification parameter name, the varies interval of them and the optimization algorithm
- **Input template file**, names of the identification parameters

These input files are all written by ASCII in *txt* files. The files can be referred to Appendix A.1 A.2 A.3 A.4. All the file names and their corresponding locations in GenOpt are defined in "initialization.txt" and "configuration.txt". By opening this initialization file and starting GenOpt, it automatically calls the other files and the simulation program.

After that, GenOpt will call *ScheduleTemplate.txt*, which is the input template file. The names of the identification parameters as specified by the entry "Name" in the optimization

command file are written in the input template file in the form of %variableName%, which can be replaced by the numerical value of the corresponding variable in each iteration. Meanwhile, the resulting file contents are written as the simulation input file.

Then GenOpt places the initial values of the optimized parameters defined in *commend.txt* into the input template file and calls the program Dymola. After running Dymola, it writes the value of the objective function into the file *result.txt*. GenOpt changes the parameters respectively according to the increase or reduction of the value of the objective function. The file "*commend.txt*" also defines the following parameters used in the optimization process:

- **Ini**, initial value of parameters
- **Min**, lower bound of parameters
- **Max**, upper bound of parameters
- **Step**, step size of optimization process

GenOpt will stop the optimization process once it finds the minimized objective function. There are also some constrains which are pre-set in "*OptimizationSettings*" of the file "*commend.tex*". If these constrains are satisfied, GenOpt would also stop.

- **MaxIte**, maximum number of iteration
- **MaxEqualResults**, the number of times that the cost function value can be equal to a value that has previously been obtained before GenOpt terminates

Algorithm gives the basic setting of the optimization algorithm. GenOpt supplies different algorithms to satisfy different problems. All the algorithms are described in details in the paper [73].

The objective function in this problem are defined as the sum of the root mean square error of the three temperature curves. The task of GenOpt is to find the most suitable thermal capacitances to obtain the best-fit curves of temperature with respect to the value of the minimized objective function. Figure 3.10 shows how to obtain this objective function value using Dymola. The measured temperature data are stored in **TimeTable** and are imported from a "*txt*" file. The simulated temperature curves are directly obtained from the thermal model which should be changed to $^{\circ}C$ at first. A **Feedback** module is used to obtain the error between the simulated temperature and the measured one. At each iteration, Dymola will save the sum of RMS error as a value of the objective function. As in figure 3.11, "*multiSum.y*" is the defined value of the objective function, which is then written into the result file.

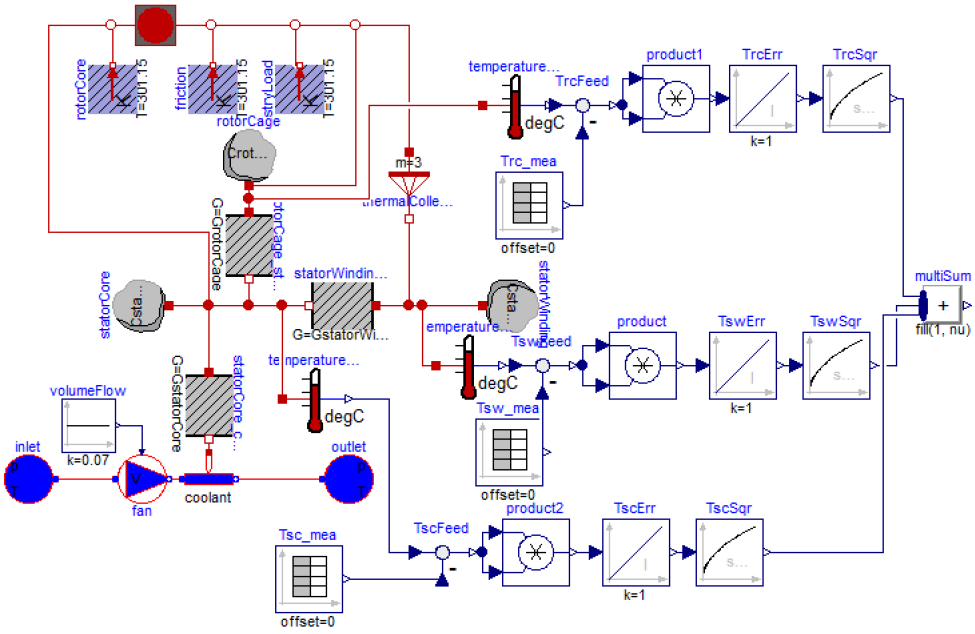


Figure 3.10: Thermal model in Dymola with objective function

```
when terminal() then
Modelica.Utilities.Streams.print("f(x) = " +
realString(number=multiSum.y, minimumWidth=1, precision=16), resultFileName);
end when;
```

Figure 3.11: Code in Dymola of writing objective function to result.txt

3.3.2.2 Optimization Result

Due to the aging of machine and the approximation of leakage inductances, all the simulated end temperatures are higher than the measured ones, which are determined by thermal conductances. Therefore, the optimized values of thermal conductances shown in table 3.8 enable more accurate values of end temperatures to be obtained.

With the optimization of GenOpt, the values of the thermal capacitance are obtained in table 3.9:

3.4 Parameters Identification Related Experiments

Both the parameters of the asynchronous machine and the thermal model have been identified in previous sections. However, whether the parameters of the physical model can perform well must be validated. The simulation of the algorithm in MATLAB and the

Table 3.8: Thermal conductances

Name	Value
$G_{sc}\,[W/K]$	16.1
$G_{sw}\,[W/K]$	14.3
$G_{rc}\,[W/K]$	3.75

Table 3.9: Thermal capacitances

Name	Value
$C_{sc}\,[J/K]$	10580
$C_{sw}\,[J/K]$	1008
$C_{rc}\,[J/K]$	1480

implementation of the algorithm in WSN largely depend on the accuracy of the parameters.

3.4.1 Validation of the Asynchronous Machine Model

The validity of the machine model must first be verified. Reference [59] lists several methods to validate the presented load models, which all rely on the output mechanical power. Generally, the torque-speed (or torque-slip) characteristics of a three phase asynchronous machine can also indicate the performance at any rotor operating speed according to the paper [74]. Therefore, the four different characteristic curves of the measurement and the simulation results shown in figure 3.12 give the validation of the asynchronous machine under various load torques.

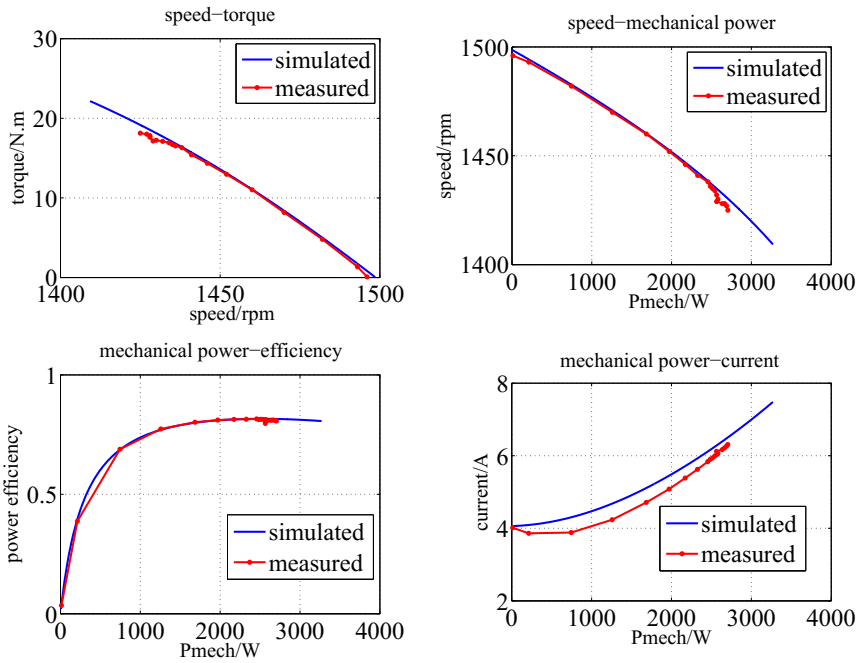


Figure 3.12: Measured and simulated characteristic curves of the asynchronous machine

The first three characteristic curves in figure 3.12 reflect good coincidence of the simu-

lated curve and measured one. For the fourth curve, two issues need to be analyzed. One is that value of the simulated input current is a little bit higher than the measured one by nearly 0.5 A, which may lead to a temperature rise of the thermal model. Another is that the first sampled data is not correct, because a decrease in current with increasing load is physically impossible. It may be caused by uncertain measurement error.

One of the reasons that leads to the differences may be the aging of the asynchronous machine. It's reasonable that the input current reduces with the corresponding load. The error might be also explained by the approximation of two leakage inductances in equations (3.22) and (3.23), which are essentially not same. Another reason is the installation of PT1000 on the rotor cage, which influences the flux density and generates excessive losses [75]. As a result, the measured value of the mechanical power is smaller than the rated value.

3.4.2 Validation of the Complete Model

The thermal model plays an important role in the monitoring of temperatures in machines. To evaluate the thermal model, two tests should be performed. In fact, the aim of the S1 (continues full-load) test is to receive the proper thermal conductances. Nevertheless, it can be also seen as an indicator of the quantity of the thermal model. The error rate is calculated as the numerical evaluation of the thermal model. The S6 (six minutes no-load followed by four minutes full-load) test aims to confirm that the thermal model could also represent of the temperature of asynchronous machine under other load conditions.

3.4.2.1 Continuous Full-load Test

S1 test is performed for about three to four hours. The heat exchange among the air and the components of the machine stabilized after about three hours. Due to the temperature rise, the resistances inside the machine will also increase. This leads to a reduction of the input current and output mechanical power. However, the model of the machine in Dymola does not take into account this change in input current. That is why the thermal conductances should be corrected in order to make the ending temperature lower. As in figure 3.13, the simulated temperature of each part fits well with the measured one. The two curves are nearly the same except for some small periods which reflect only about $1K$ deviation. The error rates are calculated according to equation (3.26), and summarized in table 3.11.

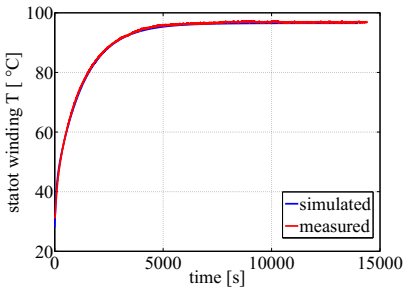
$$T_{err} = \frac{|T_{sim} - T_{mea}|}{T_{mea}} \cdot 100\% \tag{3.26}$$

where

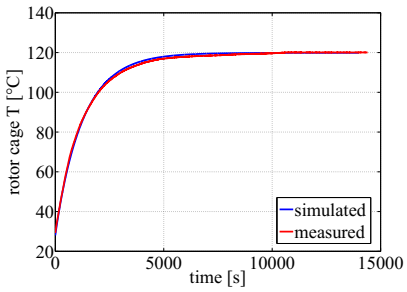
T_{sim} is the simulated temperature,

T_{mea} is the measured temperature,

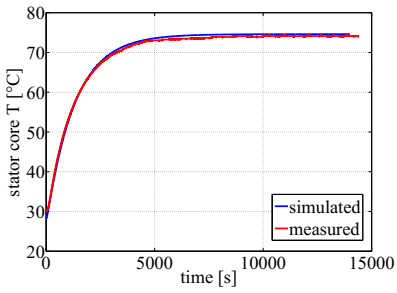
T_{err} is the temperature error rate with respect to time.



(a) Simulated and measured temperatures of the stator winding in S1 test



(b) Simulated and measured temperatures of the rotor cage in S1 test



(c) Simulated and measured temperatures of the stator core in S1 test

Figure 3.13: Simulated and measured temperatures of S1 test

Table 3.10: Temperature error rate under S1

Temperature	Max. T_{err} rate	Average T_{err} rate
T_{sw}	10.87%	0.33%
T_{sc}	11.36%	0.58%
T_{rc}	5.31%	0.71%

From table 3.11, it is shown that the maximum error rate is a little bit higher and the average error rate is relatively smaller. This means that the whole temperature curve fits well during the entire running process. The remaining error may be due to the unstable measurement.

3.4.2.2 Intermittent Load Test

To check whether the model is suitable for all working conditions, an intermittent-load test S6 is performed. The condition of intermittent load test in the thesis is similar to S6 which is defined as 6 minutes with no-load and followed by approximate 4 minutes with full-load for about three hours. No-load condition means that there is no load from the load machine, but the friction load still exists. As a result, S6 condition which is used in the thesis is always similar to the real S6 condition above. Due to the restrict of the test bench, the load machine has to be manually switched on and off. It is the reason that why the measured temperature cannot be synchronized quite well with the simulated temperature.

Although the simulation result in the S1 test is satisfactory, the three curves do not fit well here. As mentioned in section 3.4.1, the input current tends to be lower as the asynchronous machine runs steadily. However, the model does not take this input current into account. When the asynchronous machine is working under the S6 test, there are always excessive losses from the rotor cage (about 55 Watt), which contributes to a higher temperature on the rotor cage. The simulated and measured temperatures of the stator winding and the stator core are much more coincident as in figure 3.14. However, the significant deviation of the rotor cage temperature results from the same reasons as described in section 3.4.1.

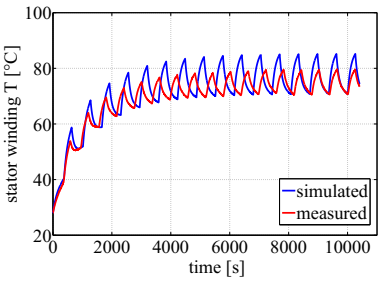
Table 3.11: Temperature error rate under S6

Temperature	Max. T_{err} rate	Average T_{err} rate
T_{sw}	12.55%	2.33%
T_{sc}	18.75%	3.57%
T_{rc}	3.31%	1.61%

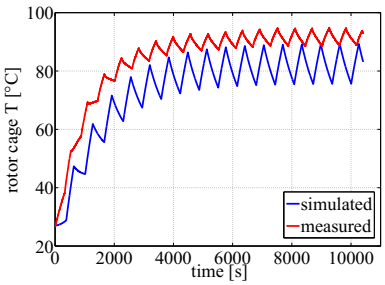
This section compares the temperature curves using identified parameters and analyses the results with respect to the S1 and S6 tests. In the S1 test the numerical indicators are given, which are maximum errors and average errors between simulated and measured temperature. The load switching during simulation can be programmed, but the load switching on the test bench has to be manually turning on and off. As a result, there is time delay when switching test load on the bench. It is difficult to synchronize the switching moment, thus there is no numerical analysis between the simulated and measured temperature under S6 condition.

3.5 Conclusions

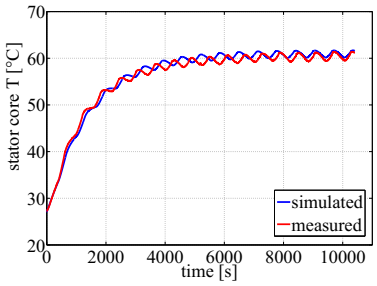
This chapter aims to determine the parameters of the asynchronous machine and thermal model. The thermal conductances are easily obtained with the method introduced in section



(a) Simulated and measured temperatures of the stator winding in S6 test



(b) Simulated and measured temperatures of the rotor cage in S6 test



(c) Simulated and measured temperatures of the stator core in S6 test

Figure 3.14: Simulated and measured temperatures of S6 test with correction of thermal conductances

3.1.2. The thermal capacitances of the model is obtained by using GenOpt which is used to get the best-fit values of the model. S1 and S6 test are performed to verify the thermal conductances and the capacitances of the model, whose values can be referred to the table 3.8 and 3.9.

CHAPTER 4

TEMPERATURES ESTIMATION OF THE ASYNCHRONOUS MACHINE

The mechanical, electrical and thermal behavior of the asynchronous machine can be simulated quite well by using the complete simulation model in chapter 3. The temperatures of the stator winding, the rotor cage and the stator core can be simulate by the simplified thermal model in figure 3.10. As a result, model-based method are used for the algorithm development. KF and EKF estimation algorithm are developed to estimate the temperatures of the stator winding, the rotor cage and the stator core. Both of the algorithms are implemented in MATLAB/SIMULINK. The estimation temperatures of the algorithm can be compared with the simulation temperatures of the model in figure 3.10. These two algorithms are described in detail in the following sections.

4.1 Temperatures Estimation of the Asynchronous Machine using a KF

KF algorithm requires a state-space model of the whole system which consists of the electrical and mechanical models of the asynchronous machine in the reference frame, and also the simplified thermal model of the machine. A 4^{th}-order KF algorithm is proposed for temperatures estimation of the machine.

4.1.1 Thermal Model of the Asynchronous Machine using a KF Algorithm

The thermal model of the asynchronous machine is constructed based on the thermal equivalent network established in [7]. The simplified thermal model is described as the equations (4.1) (4.2) (4.3).

4.1.1.1 Introduction of Thermal Parameters

The thermal network theory has been illustrated in section 2.2.1. Thermal capacity is defined in section 2.2.2. The thermal capacity is generally related to the conditions like

types of the material or pressure. Nevertheless, it can be treated as a constant in solving many technical problems as it is in this thesis.

The thermal resistance is represented as the ability that resists the heat flow between two different temperatures. It is the reciprocal of conductance G which is described in section 2.2.2.

4.1.1.2 Lumped Parameter Thermal Network

An equivalent R-C circuit which is similar to the electrical network can be developed to define the heat-transfer equation. The Dymola thermal model of the machine in figure 2.4 is constructed based on the thermal equivalent network established in [7]. The heat of the machine is generated from the losses of the power, which consists the losses of the stator winding, losses of the stator core, losses of the rotor cage, losses of the rotor core, friction losses and stray load losses [61]. There are almost no rotor core losses due to the low frequency of the rotor field, so the losses of rotor core is taken as zero. Since friction losses are correlated to frequency respectively speed, they can be treated as small additional term to the stator core losses. The stray load losses depend on the square of the stator current and can thus be included in the stator winding term. The simplified thermal model in figure 2.3 can be written as following equations:

$$\frac{dT_{sw}}{dt} = \frac{-G_{sw}T_{sw}}{C_{sw}} + \frac{G_{sw}T_{sc}}{C_{sw}} + \frac{P_{sw}}{C_{sw}} \tag{4.1}$$

$$\frac{dT_{rc}}{dt} = \frac{-G_{rc}T_{rc}}{C_{rc}} + \frac{G_{rc}T_{sc}}{C_{rc}} + \frac{P_{rc}}{C_{rc}} \tag{4.2}$$

$$\frac{dT_{sc}}{dt} = \frac{-G_{sw}T_{sw}}{C_{sc}} + \frac{G_{rc}T_{rc}}{C_{sc}} + \frac{G_{sc}T_{c}}{C_{sc}} + \frac{(G_{sw}+G_{rc}+G_{sc})T_{sc}}{C_{sc}} + \frac{P_{sc}}{C_{sc}} \tag{4.3}$$

where T_{sw}, T_{rc}, T_{sc} and T_c are temperatures above ambient of the stator winding, rotor cage, the stator core and coolant air respectively. G_{sw}, G_{rc} and G_{sc} are thermal conductances. C_{sw}, C_{rc} and C_{sc} are thermal capacitances. P_{sw}, P_{rc} and P_{sc} are the losses respective to the stator winding, the rotor cage and the stator core.

In the simplified thermal model, P_{sw}, P_{rc} are ohmic losses, and P_{sc} is the frequency-dependent iron losses, R_s, R_r are the DC resistance in ohms, between any two line terminals, I_L is the root mean square value of line current, ω is the mechanical angular frequency, k_{iron} is the iron loss constant. The losses can be represented as:

$$P_{sw}(t) = I_L^2 R_s(T_{sw}(t)) \tag{4.4}$$

$$P_{sc}(t) = k_{iron}\omega^2(t) \tag{4.5}$$

As the currents of the rotor cage are not measured or estimated by a simple method, the

rotor cage losses can be calculated as described by IEEE Power Engineering Society [76].

$$P_{rc}(t) = (P_{in}(t) - P_{sw}(t) - P_{sc}(t)) \cdot s(t) \tag{4.6}$$

$$P_{in}(t) = 3U_L I_L \cos(\phi) \tag{4.7}$$

$$s(t) = \frac{w_s - w_r(t)}{w_s} \cdot 100\% \tag{4.8}$$

where P_{in} is the input power of the machine, U_L is the root mean square value of line voltage, ω_s is the synchronous speed, ω_r is the rotor speed, s is the slip of the machine. The stator frequency is stated as constant which has been described in section 2.1.2.2.

The temperatures of the stator winding and rotor cage will increase largely. Normally it will be much higher than the reference temperature. As a result, the actual resistance can be larger than the reference resistance by 40% . The resistances will increase as the machine is running. So the resistance is modeled as temperature dependent according to equation (2.4). All in all, the stator winding losses can be calculated much more accurately. R_s can be replaced by the equation (2.4).

The state-space equations of the system can be acquired by calculating the losses P_{sw}, P_{rc}, P_{sc} defined in equations (4.4) - (4.5), and importing them into equations (4.1) - (4.3). By summarizing the previous equations, the system can be rewritten as a 4$^{\text{th}}$-order linear system in the state space model form:

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{4.9}$$

$$\mathbf{z}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{4.10}$$

where:

$$\mathbf{x} = [T_{sw}, T_{rc}, T_{sc}, T_c]^{\mathrm{T}} \tag{4.11}$$

$$z = cT_c \tag{4.12}$$

$$\mathbf{u} = [P_{sw}, P_{rc}, P_{sc}, 0]^{\mathrm{T}} \tag{4.13}$$

$$\mathbf{A} = \begin{bmatrix} \frac{-G_{sw}}{C_{sw}} & 0 & \frac{G_{sw}}{C_{sw}} & 0 \\ 0 & \frac{-G_{rc}}{C_{rc}} & \frac{G_{rc}}{C_{rc}} & 0 \\ \frac{G_{sw}}{C_{sc}} & \frac{G_{rc}}{C_{sc}} & \frac{-(G_{sw}+G_{rc}+G_{sc})}{C_{sc}} & \frac{G_{sc}}{C_{sc}} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.14}$$

$$\mathbf{B} = \begin{bmatrix} \frac{1}{C_{sw}} & 0 & 0 & 0 \\ 0 & \frac{1}{C_{rc}} & 0 & 0 \\ 0 & 0 & \frac{1}{C_{sc}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.15}$$

$$c = 1 \tag{4.16}$$

$$\mathbf{D} = 0 \tag{4.17}$$

In the state equations, $\mathbf{x}(t)$ is the state vector, $\mathbf{u}(t)$ is the control vector, **A** is the system transition matrix which is a constant matrix, **B** is the input matrix which is also constant matrix. In the measurement equation, **C** is the output matrix which is a constant in this system, **D** is the feedthrough matrix which is zero here. c is a constant. The coolant air temperature T_c is considered a constant due to the slow variation with time.

4.1.2 The Implementation of KF Algorithm

The KF is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error [77]. In general, both the process noise and the measurement noise should be taken into account in the system model and measurement model.

$$\mathbf{x}(k) = \mathbf{A}\mathbf{x}(k-1) + \mathbf{B}\mathbf{u}(k-1) + \mathbf{w}(k-1) \tag{4.18}$$

$$\mathbf{z}(k) = \mathbf{H}\mathbf{x}(k) + \mathbf{v}(k) \tag{4.19}$$

It is necessary to assume that noise of process $\mathbf{w}(k)$ and measurement $\mathbf{v}(k)$ are independent of each other, random white Gaussian noise with zero mean. Their variance can be described by covariance matrix **Q** and **R** respectively. They can be defined as

$$\mathbf{E}[w(i)w^T(j)] = \mathbf{Q}\delta(i,j) \tag{4.20}$$

$$\mathbf{E}[v(i)v^T(j)] = \mathbf{R}\delta(i,j) \tag{4.21}$$

$$\mathbf{E}[w(i)v^T(j)] = 0 \tag{4.22}$$

$\delta(i,j)$ is a Dirac Delta function variation

$$\delta(i,j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{4.23}$$

where **Q** is a 4×4 positive semi-defined constant matrix and **R** is a constant.

4.1.2.1 The Discretization of the KF Model

The 4th-order Kalman filter model is a continuous time system which can not be processed by the computer. Euler's approximation is used to discretize the model, so that the sampled data can be used in the KF algorithm. According to the definition of derivative, equation (4.18) can be rewritten when the sampling time τ is small enough as:

$$\frac{\mathbf{x}(k)-\mathbf{x}(k-1)}{\tau}=\mathbf{A}\mathbf{x}(k-1)+\mathbf{B}\mathbf{u}(k-1) \tag{4.24}$$

By simplifying the equation above, the new equation can be expressed as

$$\mathbf{x}(k)=(\mathbf{E}+\tau\mathbf{A})\mathbf{x}(k-1)+\tau\mathbf{B}\mathbf{u}(k-1) \tag{4.25}$$

As $\boldsymbol{A}$ and $\boldsymbol{B}$ are the matrix, the discrete model is

$$\mathbf{x}(k)=\mathbf{A}_d\mathbf{x}(k-1)+\mathbf{B}_d\mathbf{u}(k-1) \tag{4.26}$$

where $\boldsymbol{A}_d=\boldsymbol{E}+\tau\boldsymbol{A}$ and $\boldsymbol{B}_d=\tau\boldsymbol{B}$, $\boldsymbol{E}$ is 4×4 unit matrix, $\mathbf{C}_d$ is equal to $\mathbf{C}$.

$$\mathbf{A}_d=\begin{bmatrix} 1-\frac{G_{sw}\tau}{C_{sw}} & 0 & \frac{G_{sw}\tau}{C_{sw}} & 0 \\ 0 & 1-\frac{G_{rc}\tau}{C_{rc}} & \frac{G_{rc}\tau}{C_{rc}} & 0 \\ \frac{G_{sw}\tau}{C_{sc}} & \frac{G_{rc}\tau}{C_{sc}} & 1-\frac{(G_{sw}+G_{rc}+G_{sc})\tau}{C_{sc}} & \frac{G_{sc}\tau}{C_{sc}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.27}$$

$$\mathbf{B}_d=\begin{bmatrix} \frac{\tau}{C_{sw}} & 0 & 0 & 0 \\ 0 & \frac{\tau}{C_{rc}} & 0 & 0 \\ 0 & 0 & \frac{\tau}{C_{sc}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.28}$$

4.1.2.2 The Initialization of the KF Model

As the discrete KF is a recursive algorithm starting from sampling time $t=0$, the starting values of the state vector is

$$\hat{\mathbf{x}}(0)=E\left[\mathbf{x}(0)\right] \tag{4.29}$$

where the symbol $\hat{}$ indicates estimated value of a state vector. *var* is variance and a 4x4 covariance matrix is a diagonal matrix as below:

$$\mathbf{P}(0)=var\left[\mathbf{x}(0)\right] \tag{4.30}$$

4.1.2.3 The Prediction Stage of the KF Model

The KF estimates a process by using a feedback control. The filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements [77]. As such, the equations of the KF can be divided into two groups, prediction equations and correction equations.

The equations of prediction stage shown in (4.31) and (4.32) are responsible for projecting forward the current state and error covariance estimates to obtain a prior estimates for the next time step. $\hat{\mathbf{x}}^{-}(k)$ is the predicted value. Equation (4.31) is used for updating the state vector from previous sampling time $k-1$ to current time k. The equation (4.32) is state of updating error covariance matrix.

$$\hat{\mathbf{x}}^{-}(k) = \mathbf{A}\hat{\mathbf{x}}(k-1) + \mathbf{B}\mathbf{u}(k-1) \tag{4.31}$$

$$\hat{\mathbf{P}}^{-}(k) = \mathbf{A}\mathbf{P}(k-1)\mathbf{A}^{T} + \mathbf{Q} \tag{4.32}$$

4.1.2.4 The Correction Stage of the KF Model

The equations of correction stage are responsible for the feedback-i.e. for incorporating a new measurement into a priori estimation to obtain an improved a posteriori estimation [77].

$$\mathbf{K}(k) = \mathbf{P}^{-}(k)\mathbf{H}^{T}(k)(\mathbf{H}(k)\mathbf{P}^{-}(k)\mathbf{H}^{T}(k) + \mathbf{R})^{-1} \tag{4.33}$$

$$\hat{\mathbf{x}}(k) = \hat{\mathbf{x}}^{-}(k) + \mathbf{K}(k)(\mathbf{z}(k) - \mathbf{H}\hat{\mathbf{x}}^{-}(k)) \tag{4.34}$$

$$\mathbf{P}(k+1) = (\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k))\mathbf{P}^{-}(k) \tag{4.35}$$

where $\mathbf{K}(k)$ is Kalman gain, $\mathbf{H}(k)$ is a constant measurement matrix:

$$\mathbf{H} = [0, 0, 0, 1] \tag{4.36}$$

4.2 Temperatures Estimation of the Asynchronous Machine using an EKF

The state-space of the combined model in section 4.2.3 is a non-linear system, so the EKF is used for the estimation of temperatures. This section will describe the implementation of the EKF algorithm in details.

4.2.1 The State-Space Model of the Asynchronous Machines

The asynchronous machine is modeled based on the following hypotheses [13]:

- the space harmonics are ignored. Supposing that the three-phase windings are displaced in symmetrical design with a space angle difference of 120 degrees elect.
- the produced magnetomotive force (MMFs) is distributed sinusoidally along the air-gap.
- ignoring magnetic saturation. Self-inductances and mutual inductances of each winding are constant.
- the hysteresis of the stator core is ignored.

Park's Transformation is used to develop the electrical model of the three-phase asynchronous machine. No matter the rotor of asynchronous machine is wound rotor type or squirrel cage type, assuming it is a balanced symmetrical system, the voltage equations in a dq-axis frame rotating synchronously in arbitrary reference frame are often written as the equations (4.37) (4.38) (4.39) (4.40) below [78]:

$$v_{qs} = R_s i_{qs} + (\omega_s - \omega_{frame})\lambda_{ds} + \frac{d\lambda_{qs}}{dt} \tag{4.37}$$

$$v_{ds} = R_s i_{ds} - (\omega_s - \omega_{frame})\lambda_{qs} + \frac{d\lambda_{ds}}{dt} \tag{4.38}$$

$$v_{qr} = R_r i_{qr} + (\omega_s - p_n\omega - \omega_{frame})\lambda_{dr} + \frac{d\lambda_{qr}}{dt} \tag{4.39}$$

$$v_{dr} = R_r i_{dr} - (\omega_s - p_n\omega - \omega_{frame})\lambda_{qr} + \frac{d\lambda_{dr}}{dt} \tag{4.40}$$

where

$$\lambda_{qs} = L_s i_{qs} + L_m i_{qr} \tag{4.41}$$

$$\lambda_{ds} = L_s i_{ds} + L_m i_{dr} \tag{4.42}$$

$$\lambda_{qr} = L_r i_{qr} + L_m i_{qs} \tag{4.43}$$

$$\lambda_{dr} = L_r i_{dr} + L_m i_{ds} \tag{4.44}$$

where v_{qs}, v_{ds} are dq-axis stator voltages. v_{qr}, v_{dr} are dq-axis rotor voltages. i_{qs}, i_{ds} are dq-axis stator currents. i_{qr}, i_{dr} are dq-axis rotor currents. R_s, R_r are stator and rotor resistance. ω_s is the stator frequency, ω_{frame} is the reference frame frequency, ω is the mechanical angular velocity, ω_r is the electrical angular velocity. λ_{qs}, λ_{ds}, λ_{qr} and λ_{dr} are stator and rotor flux linkages in dq-axis. L_s, L_r are stator and rotor inductances and L_m is mutual inductance.

In order to reduce the calculation, the twin-axis reference frame is fixed on the stator, which is a stationary reference frame. Three-phase voltage and current can be transformed

to $\alpha\beta$-axis by the Clarke transformation, as defined in the equations (4.45) (4.46).

$$\begin{bmatrix} \boldsymbol{i}_{\alpha\mathrm{s}}(t) \\ \boldsymbol{i}_{\beta\mathrm{s}}(t) \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \boldsymbol{i}_{\mathrm{as}}(t) \\ \boldsymbol{i}_{\mathrm{bs}}(t) \\ \boldsymbol{i}_{\mathrm{cs}}(t) \end{bmatrix} \tag{4.45}$$

$$\begin{bmatrix} \boldsymbol{v}_{\alpha\mathrm{s}}(t) \\ \boldsymbol{v}_{\beta\mathrm{s}}(t) \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_{\mathrm{as}}(t) \\ \boldsymbol{v}_{\mathrm{bs}}(t) \\ \boldsymbol{v}_{\mathrm{cs}}(t) \end{bmatrix} \tag{4.46}$$

Stator current $i_{\alpha s}$, $i_{\beta s}$, rotor current $i_{\alpha r}$, $i_{\beta r}$, stator voltage $v_{\alpha s}$, $v_{\beta s}$ in twin-axis stator reference frame is selected. As a result, ω_{frame} is zero. i_{as}, i_{bs} and i_{cs} are the three phase stator current. v_{as}, v_{bs} and v_{cs} are the three phase stator voltage. The value of ω_s is zero, and $\delta = L_s L_r - L_m^2$, which can be referred to [79]. The final equations can be rearranged into following equations:

$$\delta \frac{di_{\alpha s}}{dt} = -R_s L_r i_{\alpha s} + L_m^2 p_n \omega i_{\beta s} + R_r L_m i_{\alpha r} + L_r L_m p_n \omega i_{\beta r} + L_r v_{\alpha s} \tag{4.47}$$

$$\delta \frac{di_{\beta s}}{dt} = -R_s L_r i_{\beta s} - L_m^2 p_n \omega i_{\alpha s} + R_r L_m i_{\beta r} + L_r L_m p_n \omega i_{\beta r} + L_r v_{\beta s} \tag{4.48}$$

$$\delta \frac{di_{\alpha r}}{dt} = R_s L_m i_{\alpha s} - L_s L_m p_n \omega i_{\beta s} - R_r L_s i_{\alpha r} - L_s L_r p_n \omega i_{\beta r} - L_m v_{\alpha s} \tag{4.49}$$

$$\delta \frac{di_{\beta r}}{dt} = L_s L_m p_n \omega i_{\alpha s} + R_s L_m i_{\beta s} + L_s L_r p_n \omega i_{\alpha r} - R_r L_s i_{\beta r} - L_m v_{\beta s} \tag{4.50}$$

The general mechanical model of the system comes from the torque balance equation in [79] which can be expressed as equation (4.51):

$$T_e = F_r \omega + J \frac{d\omega}{dt} + T_l \tag{4.51}$$

where T_e is the electromagnetic torque and T_l is the load torque. F_r is the friction constant and J is total inertia. The total electromagnetic torque of the asynchronous machine T_e can be expressed by the stator and rotor current component in stator twin-axis reference frame:

$$T_e = p_n L_m (i_{\beta s} i_{\alpha r} - i_{\alpha s} i_{\beta r}) \tag{4.52}$$

where p_n is the number of pole pairs. From the equations (4.51) (4.52), the state equation

of the rotor speed is shown below:

$$\frac{d\omega}{dt} = \frac{p_n L_m}{J}(i_{\beta s} i_{\alpha r} - i_{\alpha s} i_{\beta r}) - \frac{F_r \omega}{J} - \frac{T_l}{J} \tag{4.53}$$

4.2.2 The Thermal Model of the Asynchronous Machines using EKF

Thermal behavior of the asynchronous machine is a complex multi-disciplinary problem [80], which largely depends on the structure, materials and the detailed geometric size of the asynchronous machine. The most common method is the network approach derived from energy balance equations. The thermal system and the electrical system are modeled analogously, which has been described in section 2.2.1.

In the simplified thermal model, copper losses are defined in the equations (4.54) (4.55). P_{sw}, P_{rc} are ohmic losses in stator winding and rotor cage. R_s, R_r are the DC resistance, between any two line terminals. They can be calculated by equation (2.4).

$$P_{sw} = \frac{3}{2} R_s (i_{qs}^2 + i_{ds}^2) \tag{4.54}$$

$$P_{rc} = \frac{3}{2} R_r (i_{qr}^2 + i_{dr}^2) \tag{4.55}$$

The core losses in the stator are dependent on the numbers of iron sheets and the magnetic flux and frequency of the magnetic field, which can be separated into hysteresis losses and eddy losses. In order to simplify the model, the hysteresis losses are neglected to zero. P_{sc} can be modeled as a frequency-dependent iron losses in stator core, which is shown in equation (4.5).

4.2.3 The Combined Model of the System

The model of the asynchronous machine and the thermal model have been introduced in the previous sections. There should be some ways to combine these two separate models. In order to combine the model of the asynchronous machine and the thermal model into a series of integrated state-space equations, the temperature dependent characteristics of the resistance is used, which is defined in equation (2.4). The resistance shall vary under different temperatures. As a result, the resistance of the stator and the rotor should be calculated based on different temperatures.

Both R_s and R_r can be replaced the definition equation (2.4). Temperature coefficient of the stator winding α_s is usually the value of copper, and the temperature coefficient the rotor cage α_r is usually the value aluminum.

In the state-space equations, $\mathbf{x}(t)$ is the state vector, $\mathbf{u}(t)$ is the control vector, $\mathbf{A}(\mathbf{x}(t))$ is the system matrix which is variable with time, $\mathbf{B}$ is the input matrix which is constant matrix. In the measurement equations, $\mathbf{C}(t)$ is the output matrix, $\mathbf{D}$ is the feedthrough

matrix which is zero here. The load torque T_l is considered constant due to the slow variation with time.

From the equations (4.47) - (4.50), (4.53), (4.1) - (4.3), the final state-space of the system can be acquired by substituting P_{sw}, P_{rc}, P_{sc} expressed in (4.5) (4.54) and (4.55) into (4.1) - (4.3), and substituting R_s, R_r into equation (4.47) - (4.50), (4.1) - (4.3). By summarizing the previous equations, The system can be rewritten as a 9$^{\text{th}}$-order nonlinear continuous system in the state-space model form:

$$\mathbf{x}'(t) = \mathbf{A}(\mathbf{x}(t))\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{4.56}$$

$$\mathbf{z}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{4.57}$$

where:

$$\mathbf{x}(t) = [i_{\alpha s}, i_{\beta s}, i_{\alpha r}, i_{\beta r}, \omega, T_l, T_{sw}, T_{rc}, T_{sc}]^{\mathrm{T}} \tag{4.58}$$

$$\mathbf{z}(t) = [i_{\alpha s}, i_{\beta s}]^{\mathrm{T}} \tag{4.59}$$

$$\mathbf{u}(t) = [v_{\alpha s}, v_{\beta s}, T_c]^{\mathrm{T}} \tag{4.60}$$

$$\mathbf{A}(\mathbf{x}(t)) = \begin{bmatrix}
\frac{-a(t)L_r}{\delta} & \frac{L_m^2 p_n \omega(t)}{\delta} & \frac{b(t)L_m}{\delta} & \frac{L_r L_m p_n \omega(t)}{\delta} & 0 & 0 & 0 & 0 & 0 \\
\frac{-L_m^2 p_n \omega(t)}{\delta} & \frac{-a(t)L_r}{\delta} & \frac{-L_r L_m p_n \omega(t)}{\delta} & \frac{b(t)L_m p_n \omega(t)}{\delta} & 0 & 0 & 0 & 0 & 0 \\
\frac{a(t)L_m}{\delta} & \frac{-L_s L_m p_n \omega(t)}{\delta} & \frac{-b(t)L_s}{\delta} & \frac{-L_s L_r p_n \omega(t)}{\delta} & 0 & 0 & 0 & 0 & 0 \\
\frac{L_s L_m p_n \omega(t)}{\delta} & \frac{a(t)L_m}{\delta} & \frac{L_s L_r p_n \omega(t)}{\delta} & \frac{-b(t)L_s}{\delta} & 0 & 0 & 0 & 0 & 0 \\
\frac{-p_n L_m i_{\beta r}(t)}{J} & \frac{p_n L_m i_{\alpha r}(t)}{J} & 0 & 0 & \frac{-F_r}{J} & \frac{-1}{J} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{3a(t)i_{\alpha s}^2(t)}{2C_{sw}} & \frac{3b(t)i_{\beta s}^2(t)}{2C_{sw}} & 0 & 0 & 0 & 0 & \frac{G_{sw}}{C_{sw}} & 0 & \frac{G_{sw}}{C_{sw}} \\
0 & 0 & \frac{3b(t)i_{\alpha r}^2(t)}{2C_{rc}} & \frac{3b(t)i_{\alpha r}^2(t)}{2C_{rc}} & 0 & 0 & 0 & \frac{-G_{rc}}{C_{rc}} & \frac{G_{rc}}{C_{rc}} \\
0 & 0 & 0 & 0 & a_{95} & 0 & \frac{G_{sw}}{C_{sc}} & \frac{G_{rc}}{C_{sc}} & a_{99}
\end{bmatrix} \tag{4.61}$$

$$\mathbf{B} = \begin{bmatrix} \frac{L_r}{\delta} & 0 & 0 \\ 0 & \frac{L_r}{\delta} & 0 \\ \frac{-L_m}{\delta} & 0 & 0 \\ 0 & \frac{-L_r}{\delta} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{G_{sc}}{\delta} \end{bmatrix} \tag{4.62}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.63}$$

$$\delta = L_s L_r - L_m^2 \tag{4.64}$$

$$a(t) = (R_s(1 + \alpha_s T_{sw}(t))) \tag{4.65}$$

$$b(t) = (R_r(1 + \alpha_r T_{rc}(t))) \tag{4.66}$$

$$a_{95} = \frac{k_{iron} p_n \omega(t)}{4C_{sc}} \tag{4.67}$$

$$a_{99} = \frac{G_{sw} + G_{rc} + G_{sc}}{C_{sc}} \tag{4.68}$$

4.2.4 The Implementation of EKF Algorithm

The extended Kalman filter is a nonlinear version of the Kalman filter which linearizes about an estimation of the current mean and covariance. In general, both the process noise and the measurement noise should be taken into account in the nonlinear system model and measurement model.

$$\mathbf{x}(k) = f(\mathbf{x}(k-1), \mathbf{u}(k-1)) + \mathbf{w}(k-1) \tag{4.69}$$

$$\mathbf{z}(k) = h(\mathbf{x}(k)) + \mathbf{v}(k) \tag{4.70}$$

It is necessary to assume that the process $w(k)$ and the measurement noise $v(k)$ are random white Gaussian noise with zero mean and their variance can be described by covariance matrix Q and R respectively. The definitions of them can be referred to the section 4.1.2. **Q** of EKF is a 9x9 positive semi-defined matrix and **R** is 2x2 positive semi-defined matrix. Both of them are constant matrix. The state space of the system is nonlinear.

4.2.4.1 The Discretization of the EKF Model

The discretization process has been described in the equations (4.24) (4.25) (4.26). The discrete model is below:

$$\mathbf{x}(k) = \mathbf{A}_d\mathbf{x}(k-1) + \mathbf{B}_d\mathbf{u}(k-1) \tag{4.71}$$

where $\boldsymbol{A}_d = \boldsymbol{E} + \tau\boldsymbol{A}$ and $\boldsymbol{B}_d = \tau\boldsymbol{B}$, $\boldsymbol{E}$ is 9x9 unit matrix, $\mathbf{C}_d$ is equal to $\mathbf{C}$.

4.2.4.2 The Linearization of the EKF Model

The linearization of the non-linear model plays crucial role in the the implementation of EKF. The linearization is based on the assumption, that the state variables are constant in one step of the computation. The linearized state equation can be rewritten in a new form:

$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial \mathbf{A}_d(k)\mathbf{x}(k-1) + \mathbf{B}_d(k)\mathbf{u}(k-1)}{\partial \mathbf{x}} \tag{4.72}$$

And the output equation:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \frac{\partial \mathbf{h}(\mathbf{x}(k-1))}{\partial \mathbf{x}} \tag{4.73}$$

The Jacobians matrix $\mathbf{F}(k)$ is defined in equation (4.74), where the coefficients:

$$\mathbf{F}(k) = \begin{bmatrix}
1-\frac{a(k)L_r\tau}{\delta} & \frac{L_m^2 p_n\omega(k)\tau}{\delta} & \frac{b(k)L_m\tau}{\delta} & \frac{L_rL_mp_n\omega(k)\tau}{\delta} & f_{15} & 0 & f_{17} & f_{18} & 0 \\
\frac{-L_m^2 p_n\omega(k)\tau}{\delta} & 1-\frac{a(k)L_r\tau}{\delta} & \frac{-L_rL_mp_n\omega(k)\tau}{\delta} & \frac{b(k)L_mp_n\omega(k)\tau}{\delta} & f_{25} & 0 & f_{27} & f_{28} & 0 \\
\frac{a(k)L_m\tau}{\delta} & \frac{-L_sL_mp_n\omega(k)\tau}{\delta} & 1-\frac{b(k)L_s\tau}{\delta} & \frac{-L_sL_rp_n\omega(k)\tau}{\delta} & f_{35} & 0 & f_{37} & f_{38} & 0 \\
\frac{L_sL_mp_n\omega(k)\tau}{\delta} & \frac{a(k)L_r\tau}{\delta} & \frac{L_sL_rp_n\omega(k)\tau}{\delta} & 1-\frac{b(k)L_s\tau}{\delta} & f_{45} & 0 & f_{47} & f_{48} & 0 \\
\frac{-p_nL_mi_{\beta r}(k)\tau}{J} & \frac{p_nL_mi_{\alpha r}(k)\tau}{J} & \frac{p_nL_mi_{\beta s}(k)\tau}{J} & \frac{-p_nL_mi_{\alpha s}(k)\tau}{J} & f_{55} & -\frac{\tau}{J} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{3a(k)i_{\alpha s}^2(k)\tau}{C_{sw}} & \frac{3b(k)i_{\beta s}^2(k)\tau}{C_{sw}} & 0 & 0 & 0 & 0 & f_{77} & 0 & \frac{G_{sw}\tau}{C_{sw}} \\
0 & 0 & \frac{3b(k)i_{\alpha r}^2(k)\tau}{C_{rc}} & \frac{3b(k)i_{\beta r}^2(k)\tau}{C_{rc}} & 0 & 0 & 0 & f_{88} & \frac{G_{rc}\tau}{C_{rc}} \\
0 & 0 & 0 & 0 & f_{95} & 0 & \frac{G_{sw}\tau}{C_{sc}} & \frac{G_{rc}\tau}{C_{sc}} & f_{99}
\end{bmatrix} \tag{4.74}$$

$$f_{15} = \frac{L_m^2 i_{\beta s}(k) + L_r L_m i_{\beta r}(k)\tau}{\delta}, f_{17} = -\frac{R_s \alpha_s L_r i_{\alpha s}(k)\tau}{\delta}$$
$$f_{18} = \frac{R_r \alpha_r L_m i_{\alpha r}(k)\tau}{\delta}, f_{25} = -\frac{L_m(L_m i_{\alpha s}(k) + L_r i_{\alpha r}(k))\tau}{\delta}$$
$$f_{27} = -\frac{R_s \alpha_s L_r i_{\alpha s}(k)\tau}{\delta}, f_{28} = \frac{R_r \alpha_r L_m i_{\beta r}(k)\tau}{\delta}$$
$$f_{35} = -\frac{L_m L_s i_{\beta s}(k) + L_s L_r i_{\beta r}(k)\tau}{\delta}, f_{37} = \frac{R_s \alpha_s L_m i_{\alpha s}(k)\tau}{\delta}$$
$$f_{38} = -\frac{R_r \alpha_r L_s i_{\alpha r}(k)\tau}{\delta}, f_{45} = \frac{L_m L_s i_{\alpha s}(k) + L_s L_r i_{\alpha r}(k))\tau}{\delta}$$
$$f_{47} = \frac{R_s \alpha_s L_m i_{\beta s}(k)\tau}{\delta}, f_{48} = -\frac{R_r \alpha_r L_s i_{\beta r}(k)\tau}{\delta}$$
$$f_{55} = 1 - \frac{F_r \tau}{J}, f_{77} = 1 + \frac{3(R_s \alpha_s (i_{\alpha s}^2(k) + i_{\beta s}^2(k)) - G_{sw})\tau}{2C_{sw}}$$
$$f_{88} = 1 + \frac{3(R_r \alpha_r (i_{\alpha r}^2(k) + i_{\beta r}^2(k)) - G_{rc})\tau}{2C_{rc}}$$
$$f_{95} = \frac{k_{iron} p_n \omega(k)\tau}{2C_{sc}}, f_{99} = 1 - \frac{(G_{sw} + G_{rc} + G_{sc})\tau}{C_{sc}}$$

4.2.4.3 The Initialization of the EKF Model

In general, the initialization of the EKF is the same as the description in section 4.1.2 of KF. Both the process noise and the measurement noise should be taken into account in the system model and measurement model. And a 9×9 covariance matrix $\mathbf{P}(0)$ is a diagonal matrix as below:

$$\mathbf{P}(0) = var\left[\mathbf{x}(0)\right] \tag{4.75}$$

4.2.4.4 The Prediction Stage of the EKF Model

The EKF estimates a process by using a feedback control. The filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements [77]. As such, the equations of the Kalman filter can be divided into two groups: prediction equations and correction equations. The prediction stage equations of EKF are shown in equations (4.76) and (4.77). Equation (4.76) is used for updating the state vector from previous sampling time k-1 to current time k. The equation (4.77) is state of updating error covariance matrix.

$$\hat{\mathbf{x}}^{-}(k) = f(\hat{\mathbf{x}}(k-1), \mathbf{u}(k-1), 0) \tag{4.76}$$

$$\hat{\mathbf{P}}^{-}(k) = \mathbf{F}(k)\mathbf{P}(k-1)\mathbf{F}^{T}(k) + \mathbf{Q} \tag{4.77}$$

where $F(k)$ is the system process Jacobians at step k

$$\mathbf{F}(k) = \frac{\partial f}{\partial \mathbf{x}}\bigg|_{x=\hat{x}(k)} \tag{4.78}$$

4.2.4.5 The Correction stage of the EKF Model

$$\mathbf{K}(k) = \mathbf{P}^{-}(k)\mathbf{H}^{T}(k)(\mathbf{H}(k)\mathbf{P}^{-}(k)\mathbf{H}^{T}(k) + \mathbf{R})^{-1} \tag{4.79}$$

$$\hat{\mathbf{x}}(k) = \hat{\mathbf{x}}^{-}(k) + \mathbf{K}(k)(\mathbf{z}(k) - \mathbf{h}(\hat{\mathbf{x}}^{-}(k), 0)) \tag{4.80}$$

$$\mathbf{P}(k+1) = (\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k))\mathbf{P}^{-}(k) \tag{4.81}$$

where $\mathbf{H}(k)$ is the measurement Jacobians at step k

$$\mathbf{H}(k) = \frac{\partial h}{\partial \mathbf{x}}\bigg|_{x=\hat{x}(k)}$$

4.3 MiL-Test and Experimental Results

In this section, two MiL-tests are performed based on the Dymola block. One is the EKF algorithm and the other is the KF algorithm, both of which are implemented in the SIMULINK. The model and the simulation environment are described in section 4.3.1.

4.3.1 The MiL-Test of Combined Simulation Models

The model of a squirrel cage asynchronous machine with losses, with which couples the simplified thermal model are built using Dymola. The squirrel cage asynchronous machine with losses is explained in [61] [59] [81]. The model can simulate the transient electronic and magnetic behavior as well as six parts of the machine losses. By connecting an internal thermal port of the machine to the thermal port of the simplified thermal model, all the losses can be passed to the thermal circuit. With the complete model, the temperatures of the stator winding, the rotor cage and the stator core can be calculated. The simplified thermal model is shown in figure 2.4 in section 2.2. Both the simulation model and the experiment are explained in [7].

Based on the parameters of the asynchronous machine in [59] and the thermal parameters in [7]. The complete model is built and tested, which is shown in figure 4.1:

4.3.2 The Test Results for KF Estimator

With the help of Dymola-Simulink interface, the complete model in Dymola can also run in SIMULINK environment. How to configure and implement the Dymola model in

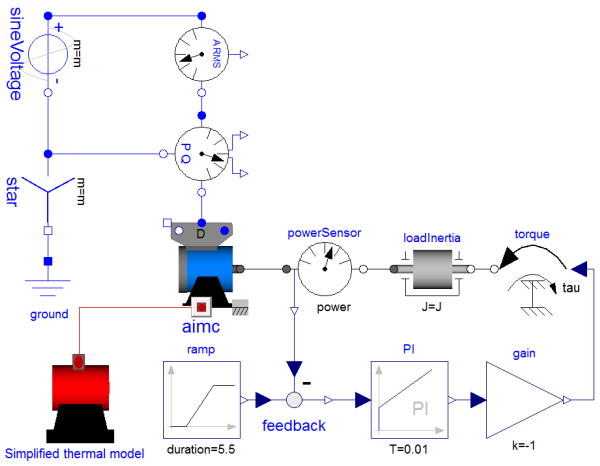


Figure 4.1: The complete model

SIMULINK is described in detail by Garron Fish [82]. The combined physical model in figure 4.1 consists of two parts. One is the squirrel cage asynchronous machine with losses, and the other is the simplified thermal model in figure 2.4. They are compiled as *DymolaBlock* and connected with the Kalman filter algorithm which is implemented as an S-Function with white gaussian noise inside in SIMULINK. In the complete simulation model, all the state variables can be exported from *DymolaBlock* as either the inputs of the algorithm or the compared simulation results. Losses of the stator winding, the rotor cage and the stator core can be calculated from the rotor speed, three-phase stator currents and voltages which are exported from $DymolaBlock$. The estimated temperatures from KF can be compared with the temperatures simulated by $DymolaBlock$. The online KF estimator in MATLAB/SIMULINK is shown in figure 4.2 below:

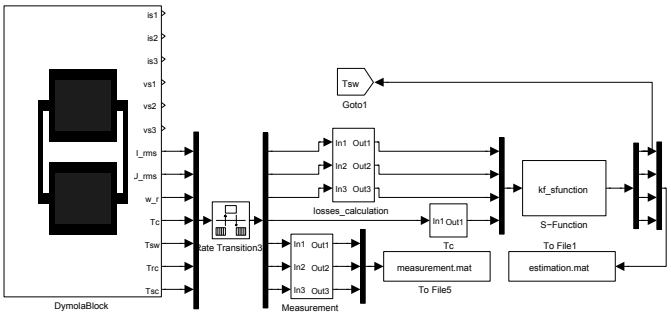


Figure 4.2: KF estimator in SIMULINK

4.3.2.1 Three proposed Methods by KF Estimator

The parameters of the asynchronous machine and the thermal model are based on the papers [7] [59]. The implemented KF algorithm is independent from the control strategy and the running conditions of the machine. It can estimate the temperatures of the stator winding, the rotor cage and the stator core in any operating condition. Full-load test and intermittent-load test have performed in SIMULINK. Because 1 second sampling time is enough for estimation. The sampling time τ set by $RateTransition$ is 1 second. The initial error covariance matrix, process noise covariance matrix and measurement covariance matrix are obtained by trial and error method:

$$\mathbf{P}(0) = diag\begin{bmatrix}20 & 20 & 20 & 20\end{bmatrix}$$

$$\mathbf{Q} = diag\begin{bmatrix}0.001 & 0.001 & 0.001 & 0.1\end{bmatrix}$$

$$\mathbf{R} = 0.1$$

From the state-space equations (4.1) - (4.3), losses of the thermal model largely determine the accuracy of the estimated temperature. In order to prove the assumption, three different implementation methods are proposed. They are all developed based on the same software function. The only difference is how to get the losses as the input. The first method is to export power losses directly from the $DymolaBlock$ to Kalman filter, which is short as EPL. The second is to calculate the power losses based on the equations (4.4) - (4.8), where the stator winding resistance R_{sw} is taken as a constant. It is short as CPL. The third is to calculate the power losses which is the same as the second method, but the difference is that for every simulation step, the estimated temperature of the stator winding T_{sw} would be sent back to $lossescalculation$ block to compensate the ignored stator winding resistance rise defined in equation 2.4. It is short as CPLC. For every method, S1 and S6 experiments are performed.

4.3.2.2 The Estimation Results by using EPL Method

The asynchronous machine model is modeled with losses which is described in [59]. The losses of the stator winding P_{sw}, the rotor cage P_{rc} and the stator core P_{sc} are exported from $Dymola$ and sent to KF algorithm as the input. Both S1 and S6 are performed. Apart from the deviation for every point, the normalized root-mean-square error (NRMSE) e_{NRMS} defined in equation (4.82) is used to evaluate the accuracy of the estimator. y_{mea}

is the measured value and y_{est} is the estimated value.

$$e_{NRMS} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left(\frac{y_{mea}(i) - y_{est}(i)}{max(y_{mea}) - min(y_{mea})}\right)^2} \tag{4.82}$$

Under S1, the maximal error of the stator winding is 0.15 K, of the rotor cage 0.01 K, and of the stator core 0.03 K. The NRMSE of the stator winding is 0.18%, of the rotor cage 0.03% and of the stator core 0.04%. The results under S1 and S6 are listed in tables 4.1 and 4.2:

Table 4.1: The maximum error and NRMSE of EPL under S1

Parameters	Maximum Error	NRMSE
Stator winding	0.15 K	0.18%
Rotor cage	0.01 K	0.03%
Stator core	0.03 K	0.04%

Table 4.2: The maximum error and NRMSE of EPL under S6

Parameters	Maximum Error	NRMSE
Stator winding	0.26 K	0.44%
Rotor cage	0.08 K	0.12%
Stator core	0.02 K	0.08%

4.3.2.3 The Estimation Results by using CPL Method

In the experiment, resistance of the stator winding is a constant. However in the real situation, the resistance will increase as the temperature rises. That is the reason why the estimated temperatures are much lower than the simulated temperatures.

Under S1, the maximal error of the stator winding is 9.5 K, of the rotor cage 4.8 K, and of the stator core 5.2 K. The NRMSE of stator winding is 10.2%, of the rotor cage 5.5% and of the stator core 6.5%. The results are listed in tables 4.3 and 4.4:

Table 4.3: The maximum error and NRMSE of CPL under S1

Parameters	Maximum Error	NRMSE
Stator winding	9.5 K	10.2%
Rotor cage	4.8 K	5.5%
Stator core	5.2 K	6.5%

Table 4.4: The maximum error and NRMSE of CPL under S6

Parameters	Maximum Error	NRMSE
Stator winding	10.2 K	9.3%
Rotor cage	6.3 K	5.4%
Stator core	5.7 K	6.1%

4.3.2.4 The Estimation Results by using CPLC Method

The comparison of simulated and estimated temperatures under S1 is shown in figure 4.3. The maximal error of the stator winding is 2.3 K, of the rotor cage 3.5 K, and of the stator core 2 K. The NRMSE of stator winding is 2.1%, of the rotor cage 2.9% and the stator core 2.2%. The results are listed in tables 4.5 and 4.6:

Table 4.5: The maximum error and NRMSE of CPLC under S1

Parameters	Maximum Error	NRMSE
Stator winding	2.3 K	2.1%
Rotor cage	3.5 K	2.9%
Stator core	2.0 K	2.2%

Table 4.6: The maximum error and NRMSE of CPLC under S6

Parameters	Maximum Error	NRMSE
Stator winding	2.0 K	1.3%
Rotor cage	0.8 K	1.4%
Stator core	1.8 K	1.1%

The comparison of simulated and estimated temperatures under S1 and S6 are shown in figures 4.3 and 4.4.

4.3.3 The Test Results for EKF Estimator

The simulation model has been introduced in section 4.3.2. In the complete model, the three-phase stator current as the measurement, three-phase voltage and the coolant air temperature as the control vector can be exported from $DymolaBlock$ as the input of the EKF. Meanwhile the estimated temperatures from EKF can be compared with the temperatures calculated by the simplified thermal model. The online EKF estimator in MATLAB/SIMULINK is shown in figure 4.5 below:

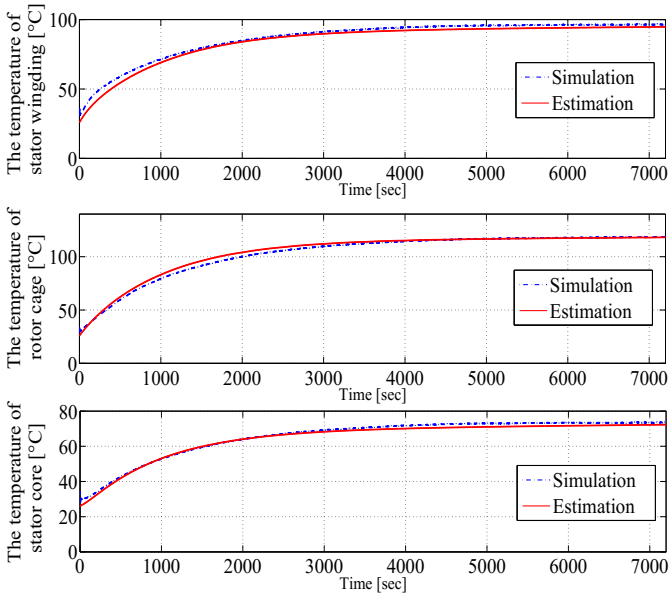


Figure 4.3: CPLC: Comparison of simulated and estimated temperatures under S1

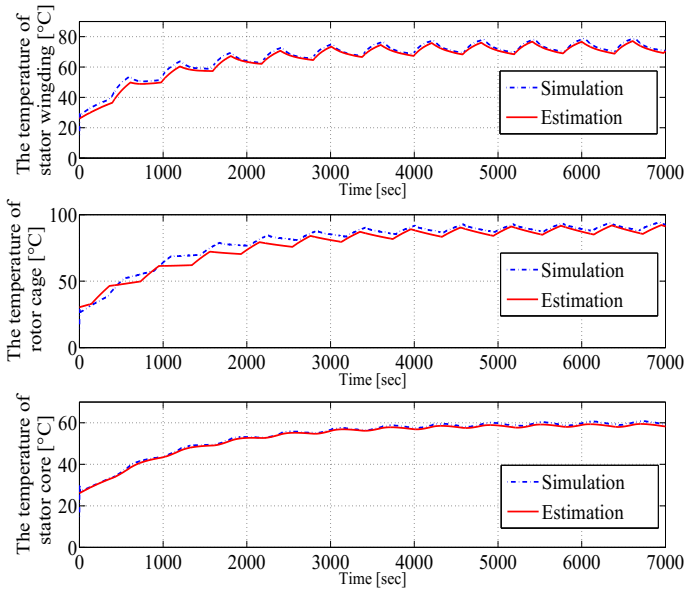


Figure 4.4: CPLC: Comparison of simulated and estimated temperatures under S6

The parameters of the asynchronous machine and the parameters of the thermal model are listed in tables D.2, 3.8 and 3.9 respectively. The implemented EKF algorithm is independent from the control strategy and the running conditions of the machine. It can estimate the temperatures of the stator winding, the rotor cage and the stator core in any operating

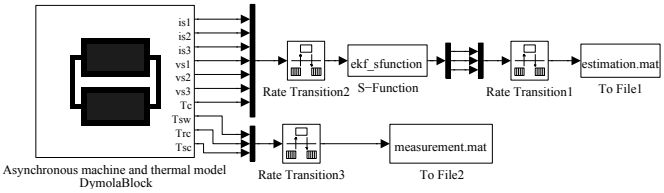


Figure 4.5: EKF estimator in SIMULINK

condition. Full-load test and intermittent-load test have performed in SIMULINK with a sampling rate of 2,000 Hz. This sampling rate is the minimum value which can estimate the temperatures accurately for this system. Trial and error method is used to determine the covariance matrix. The initial error covariance matrix, process noise covariance matrix and measurement covariance matrix are obtained by the defined noise in the simulation:

$$\mathbf{P}(0) = diag\begin{bmatrix}5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5\end{bmatrix}$$

$$\mathbf{Q} = diag\begin{bmatrix}3 & 3 & 0.5 & 0.5 & 0.01 & 0.1 & 10^{-7} & 2\times10^{-7} & 10^{-8}\end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix}0.1 & 0\\ 0 & 0.1\end{bmatrix}$$

Two experiments are performed. One is the continuous full-load **S1** which means the machine runs at the rated condition until the temperatures are stable. The figure 4.6 shows the comparison between the temperatures exported from the physical model in *Dymola* and the temperatures estimate by EKF in SIMULINK. The estimated temperatures follow the simulated reference temperatures quite well. The maximum error and the NRMSE value for each part under S1 condition are summarized in table 4.7. The maximum temperature error of the stator winding is 1.6 K, of the rotor cage3.1 K and of the stator core 1.2 K. The NRMSE of the stator winding is 2.11%, of the rotor cage 2.91% and of the stator core 2.05%.

Table 4.7: The error and NRMSE of the estimated temperatures under S1

Parameters	Maximum Error	NRMSE
Stator winding	1.6 K	2.11%
Rotor cage	3.1 K	2.91%
Stator core	1.2 K	2.05%

S6 experiment is performed with the sampling time 500 μs. The results of the temperatures match the simulated reference temperatures very well. The figure 4.7 shows the

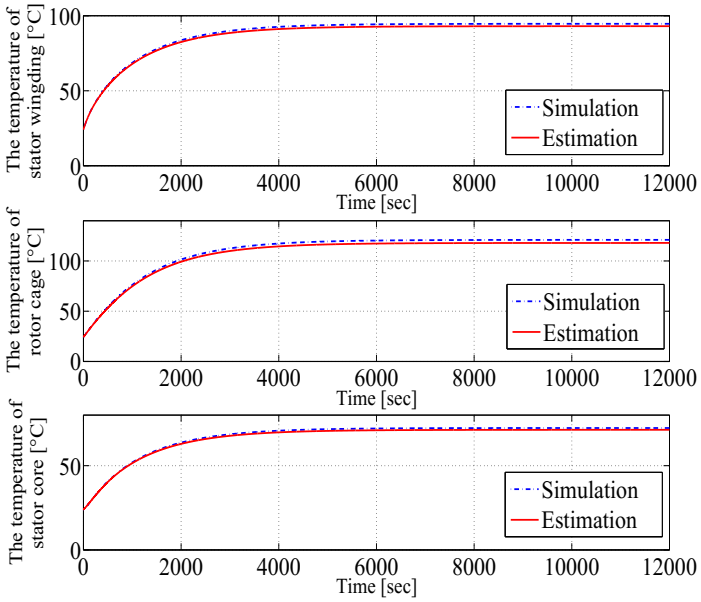


Figure 4.6: EKF simulated and estimated temperatures under continuous duty S1

comparison between the temperatures simulated and the temperatures estimate by EKF under an intermittent load S6. The maximum error and the NRMSE value for each part under S6 condition are summarized in table 4.8.

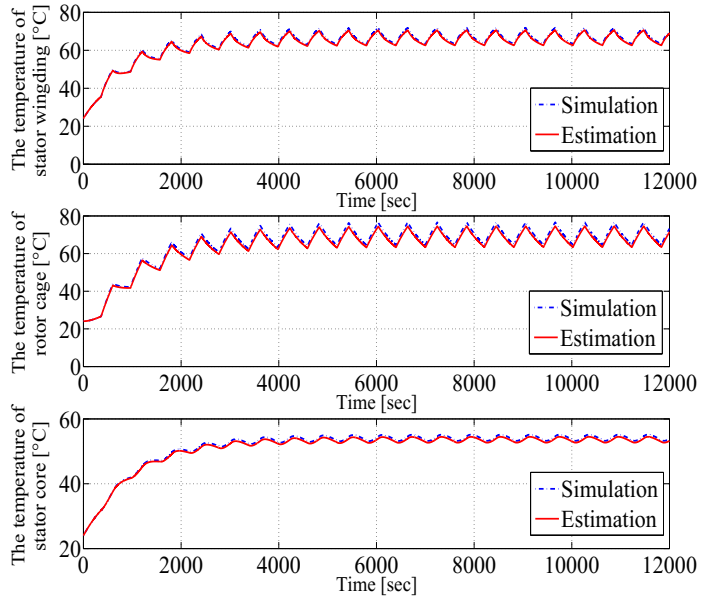


Figure 4.7: EKF simulated and estimated temperatures under intermittent duty S6

Table 4.8: The error and NRMSE of the estimated temperatures under S6

Parameters	Maximum Error	NRMSE
Stator winding	1.6 K	1.88%
Rotor cage	2.1 K	3.01%
Stator core	1.8 K	1.86%

4.3.4 The Results on the Test Bench Machine

In order to be proved that both of the algorithms can be used for the temperatures estimation on the test bench, experiments on the test bench are performed. The temperatures of the stator winding, the stator core and the coolant air are measured by data acquisition card NI PCI-6023E. The temperature of rotor cage is measured by wireless sensor node and transmitted wirelessly to the host sensor node. The test bench is shown in figure 4.8:

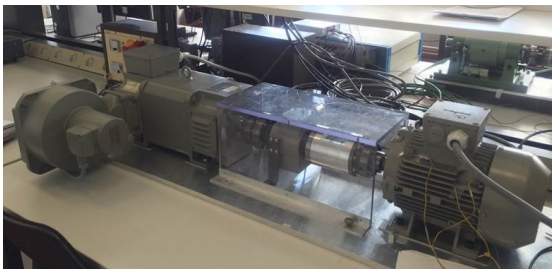


Figure 4.8: The test bench

4.3.4.1 The Estimation Results of KF of the Test Bench

All the signals are processed by the KF algorithm offline. The comparisons between the measured and estimated temperatures are shown in figures 4.9 and 4.10. All the temperatures can be estimated accurately under S1 and S6, except for the rotor cage temperature under S6. It is due to the excessive losses from the rotor cage, the root cause is also described in section 3.4.1. That is why the measured rotor temperature is always higher than the estimated temperature. The maximum error and NRMSE of KF under S1 and S6 are listed in tables 4.9 and 4.10. The error and NRMSE of rotor cage under S6 are not calculated because the time stamp of estimated and measured data is difficult to be synchronized.

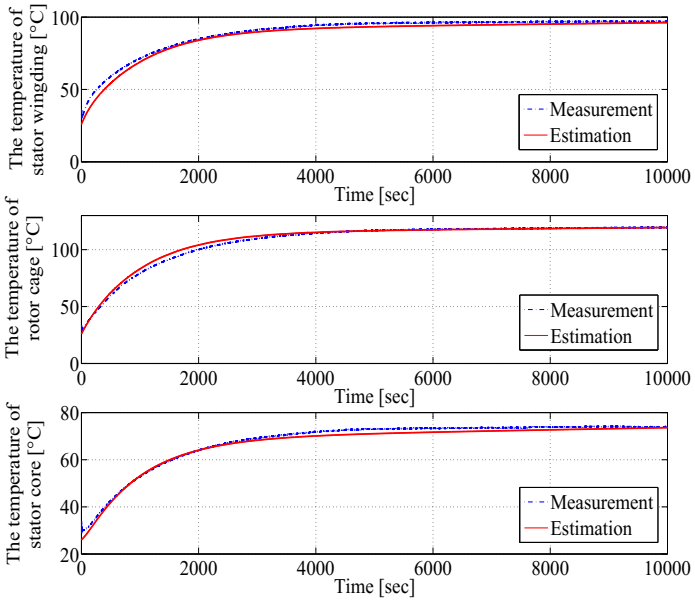


Figure 4.9: KF measured and estimated temperatures under S1

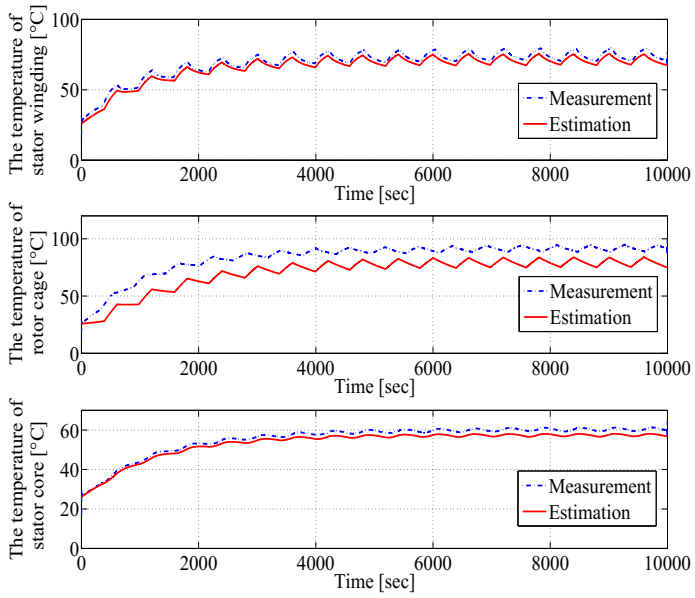


Figure 4.10: KF measured and estimated temperatures under S6

4.3.4.2 The Estimation Results of EKF of the Test Bench

The test bench is running under both S1 and S6 condition for about three hours. The inputs of the EKF including the coolant air temperature, three-phase currents and voltages

Table 4.9: The maximum error and NRMSE of KF under S1

Parameters	Maximum Error	NRMSE
Stator winding	3.2 K	3.1%
Rotor cage	3.8 K	3.55%
Stator core	2.7 K	2.81%

Table 4.10: The maximum error and NRMSE of KF under S6

Parameters	Maximum Error	NRMSE
Stator winding	3.2 K	3.32%
Rotor cage	N/A	N/A
Stator core	2.4 K	2.81%

are acquired by data acquisition card NI PCI-6023E, with the sampling rate of 2,000 Hz. The acquired data will first be stored and processed by the EKF which is implemented in MATLAB/SIMULINK. The comparisons of measured and estimated temperatures on the test bench are shown in figures 4.11 and 4.12. The maximum error and NRMSE of EKF under S1 and S6 are listed in tables 4.11 and 4.12. The reason why the error and NRMSE of rotor cage under S6 is difficult to be calculated has been introduced in section 4.3.4.1.

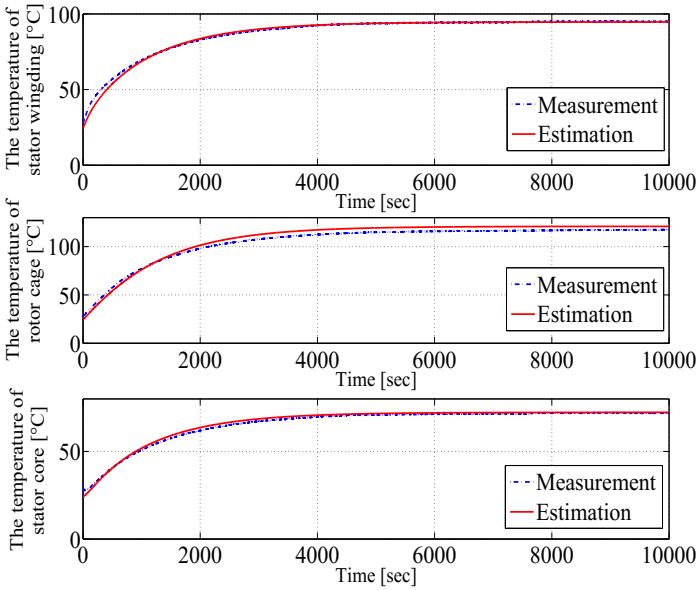


Figure 4.11: EKF measured and estimated temperatures under S1

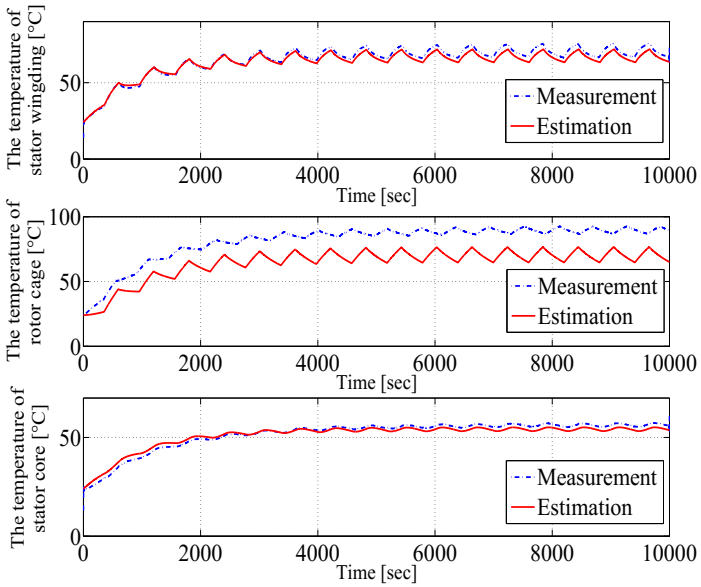


Figure 4.12: EKF measured and estimated temperatures under S6

Table 4.11: The maximum error and NRMSE of EKF under S1

Parameters	Maximum Error	NRMSE
Stator winding	3.5 K	3.32%
Rotor cage	3.2 K	3.35%
Stator core	3.1 K	2.78%

Table 4.12: The maximum error and NRMSE of EKF under S6

Parameters	Maximum Error	NRMSE
Stator winding	3.6 K	2.98%
Rotor cage	N/A	N/A
Stator core	2.8 K	3.12%

4.4 Conclusions

This chapter proposed a method to estimate the temperatures of the stator winding, the rotor cage and the stator core of an asynchronous machine using both KF and EKF. By combining the model of the asynchronous machine in twin-axis stator reference frame and the simplified thermal model, the state-space equations have been defined and implemented as a 4^{th}-order KF and 9^{th}-order EKF. The complete temperatures simulation model based on [7] is modeled by Dymola and runs well. For the three proposed method, EPL can estimate

the temperatures quite accurately, but it is not available in the real machine. Due to the temperature compensation for resistance calculation, CPLC can estimate the temperatures more accurately than CPL. As a result, CPLC is used to calculate the power losses for KF.

The following results are acquired by comparing the estimated temperatures with simulated temperatures. Under continuous full-load S1, the estimated temperatures follow the simulated reference temperatures very well. The maximal error with EKF is 3.1 K on the rotor cage. The maximal error with KF is 0.15 K on the stator winding. Under intermittent duty load S6, the maximal error with EKF is 2.1 K on the rotor cage. The maximal error with KF is 0.26 K on the rotor cage.

The following results are acquired by comparing the estimated temperatures with measured temperatures. The estimated temperatures on the stator winding and the stator core follow the measured temperatures very well. Only the temperatures of the stator winding and the stator core are compared. Under S1, the maximal error with EKF is 3.5 K. The maximal error with KF is 3.8 K. Under S6, the maximal error with EFK is 3.6 K. The maximal error with KF is 3.2 K. The reason why the estimated temperature of rotor cage is not correct is explained in section 3.4.1. If the rotor cage temperature of this specific test bench need to be estimated accurately, the excessive power losses from the rotor should be considered. The advantages of the EKF algorithm are as follows:

- The estimated temperatures can be obtained by acquiring the three phase stator voltage, current and the coolant air temperature. The rotor speed and the torque can also be estimated simultaneously.
- The estimator is independent from the operation conditions. That means no matter what the rotor speed is, and what the mechanical load is, as long as there are currents through the stator winding, the temperature can be estimated correctly.

CHAPTER 5

THE IMPLEMENTATION OF KF IN THE WSN

As is described in chapter 4, a number of methods for temperatures monitoring of the asynchronous machine can be found in the reference. However, none of them has been implemented on a wireless sensor network so far. Some of the methods do not provide satisfactory results or can only estimate the temperatures of the stator winding and the rotor cage without the stator core. Other methods require powerful calculation ability which cannot be run on a resource limited wireless sensor node. Currently, three parts of temperatures estimated by the 4^{th}-order KF algorithm in chapter 4 match the measured temperatures quite accurately, which seems to be the most promising for the implementation in a wireless sensor network.

This chapter is organized as follows: section 1 gives an overview of the proposed system. The implementation of distributed WTIMs and NCAP are given in section 2 and section 3. Section 4 gives the structure of the system and the communication sequence. Finally, the conclusions are given in section 5.

5.1 The proposed System Description

This section generally introduces the target system including the hardware platform, the operating system and the software structure to be used. The structure and the topology of the system is also briefly given.

5.1.1 The Target System

The basis of the platform is the wireless sensor node Preon32 developed by Virtenio GmbH. It contains a 32 bit ARM Cortex-M3 microcontroller with 256 kB flash memory for programming code and 64 kB RAM memory for data. A 2.4 GHz wireless transceiver which is compliant to IEEE 802.15.4 standard can for example be used for ZigBee or 6LoWPAN communication. Two 12-bit analog-to-digital converters (ADC) with maximum sampling rate of 1 M samples/s are provided [9]. The maximum clock frequency is f_{clk} = 72 MHz. However, it is usually preferred to operate at a frequency of f_{clk} = 36 MHz

Figure 5.1: Preon32 [9]

Figure 5.2: Preon32Shuttle [10]

as this can be achieved using the internal RC-oscillator of the processor. Yet, a frequency of f_{clk} = 72 MHz requires an external crystal oscillator with a significantly higher power consumption. The clock for time keeping is generated from a low power watch crystal and has a resolution of 2^{-14} s = 61.035 μs and a width of 32 bit. The sampling period is derived from the CPU-clock and can be set with a resolution of 1 μs [24].

The Preon32Shuttle (dimensions of 35 x 35 x 4.5 mm) from Virtenio is an expansion module for the Preon32 device (dimensions of 27.5 x 19 x 3.3 mm) that provides components for improving overall prototyping and development for the platform. The expansion board has following features: two buttons, four LEDs, voltage regulation with 3.6 to 13.2 V. The serial-to-USB bridge and the USB connector provide a convenient way of programming the Preon32 device via USB and for power supply of the device. The pin headers, buttons and LEDs are extremely useful during prototyping in order to interact with the device and to connect external peripherals easily [10]. In the context of the presented application all used Preon32 devices are already soldered to a Preon32Shuttle. Preon32 and Preon32Shuttle are shown in figure 5.1 and figure 5.2.

As is introduced in the previous chapter, all the sensor nodes are programmed in the C programming language on the Contiki operating system. Contiki is an open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks. Contiki is designed for microcontroller with small amounts of memory. A typical Contiki configuration is 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki supports fully standard IPv6 and IPv4, along with the recent low-power wireless standards: 6LoWPAN, RPL, CoAP. With Contiki's ContikiMAC and sleepy routers, even wireless routers can be battery-operated. Contiki contains two communication stacks: uIP and Rime. uIP is a small RFC compliant TCP/IP stack that makes it possible for Contiki to communicate over the Internet. Rime is a lightweight communication stack designed for low-power radios [83].

Most operating systems for embedded systems require that a complete binary image of the entire system is built and downloaded into each device. The binary includes the operating system, system libraries, and the actual applications running on top of the system. In contrast, Contiki has the ability to load and unload individual applications or services at run-time. In most cases, an individual application is much smaller than the entire system binary and therefore requires less energy when transmitted through a network. Additionally, the transfer time of an application binary is less than that of an entire system image.

The whole software consists of following components: Contiki, ARM CMSIS Library, Preon32 platform, Preon32 firmware and MSTL. Figure 5.3 shows the components of the WSN software. MSTL is implemented by MDT of Technische Universität Berlin. It provides the management of the data acquisition for a variety of sensors and actuators of the wireless sensor nodes. It is inspired by the IEEE1451 family of standards for smart transducers.

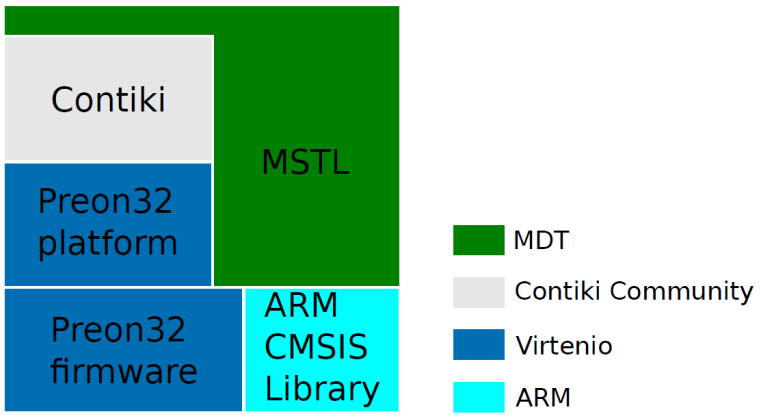


Figure 5.3: Components of the WSN Software [11]

5.1.2 Structure and Topology of the System

As is discussed in section 1.2.2, the "Star" topology is used for this WSN. Based on the proposed KF algorithm, four types of signals are acquired as the inputs of the algorithm. The Root-Mean-Square (RMS) of current and voltage are acquired by the hall sensors and filtered by the low pass filter. PT1000 is used to acquire the coolant air temperature and rotary encoder is used for the rotor speed. All in all, three separate wireless sensor nodes are used for data acquisition, pre-processing and transmission, one wireless sensor node is used for the data receiving and algorithm implementation. The structure of the proposed system is shown in figure 5.4:

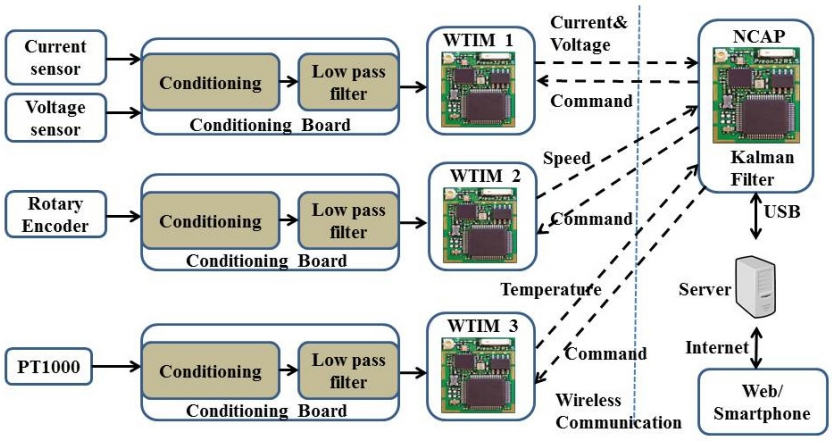


Figure 5.4: Structure of the temperatures estimation system based on WSN [11]

5.2 Implementation of Data Acquisition System in Distributed WTIMs

All the implementation of data acquisition systems is based on the MSTL which has been introduced in section 1.2.2.2. The structure of the MSTL can be referred to figure 1.4.

5.2.1 The Hardware

Preon32 provides multiple I/O interfaces are available for connection to external peripheral digital I/O pins which could be used for the acquisition of rotor speed. Analog signals such as the coolant air temperature, the three-phase currents and voltages can be captured with the integrated ADC. Additional hardware is designed for connecting the sensor with Preon32 sensor node and conditioning the analog signal.

5.2.1.1 The Hardware for Three-phase Currents and Voltages

The whole construction of the DAQ system is shown in figure 5.5. The circuit board is inserted into a housing, which is connected to the standard rail of the control cabinet fits. The design of the board is taken into account of the cable system prevailing in the control cabinet. This provides that all conductors carry the low voltage into the cable channel below the board. The completion of the board and the housing, as well as the installation in the control cabinet was supported by the workshop of the MDT. The preon32 sensor node is used for data acquisition and wireless transmission. The sensor node and other components such as inverter and power supply are all installed in the cabinet.

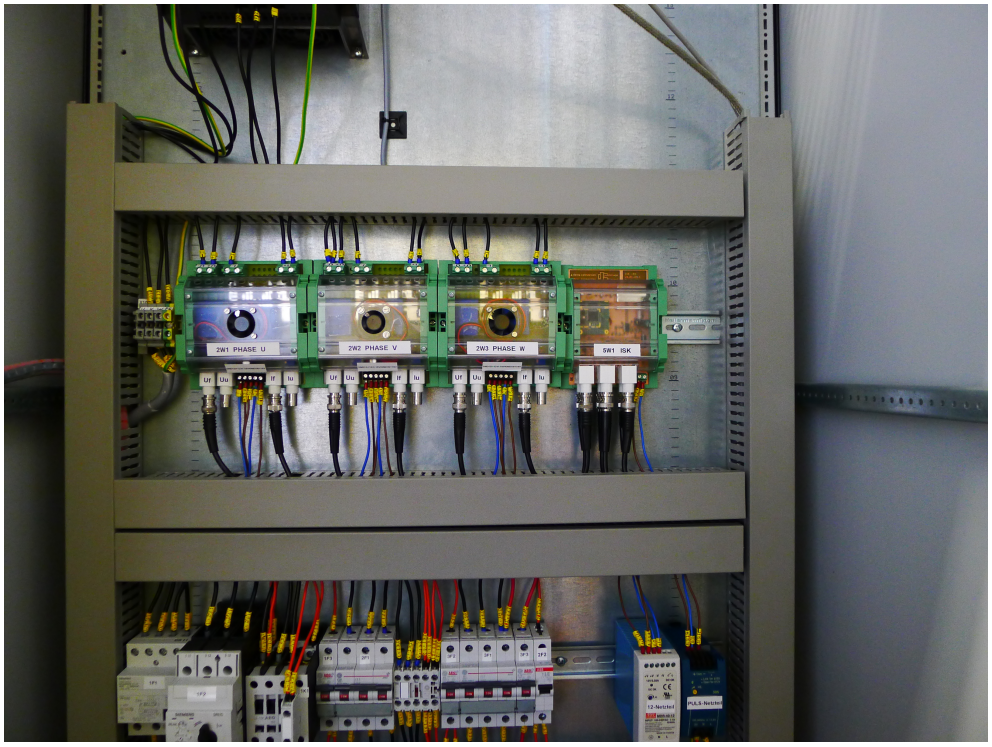


Figure 5.5: The whole construction of the DAQ system

The conditioning boards with six hall sensors were designed for acquiring the three-phase currents and voltages in the project of MDT. The details of the project can be referred to the reference [84]. NI PCI-6023E data acquisition boards are used to acquire the output from the conditioning board, which is in the range of $\pm 10V$. As the analog input port of the Preon32 is in the range of $\pm 3.3V$, other conditioning boards for Preon32 were developed in a master thesis [85], which are shown in figures 5.6 and 5.7.

5.2.1.2 The Conditioning Board for Coolant Air Temperature

The coolant air temperature is one of the inputs which should be measured and transmitted wirelessly by Preon32 sensor node. PT1000 which is the same as in 3.1.2.1 is used for the temperature measurement. The output voltage of the conditioning board provided together with the sensor can be calibrated between 0 V and 3.3 V. Both the PT1000 and the conditioning board are shown in figures 3.2(a) and 3.2(b).

5.2.1.3 The Conditioning Board for Rotor Speed

In order to acquire the speed of the rotor, a rotary encoder "ROD 426 B-6000" from HEIDENHAIN GmbH is used. According to the technical data sheet, a conditioning circuit

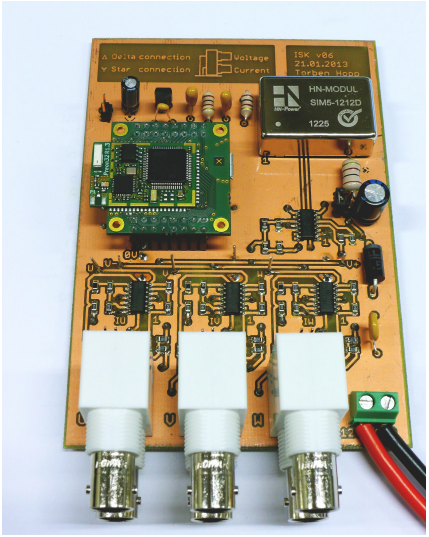


Figure 5.6: Conditioning board without housing [11]

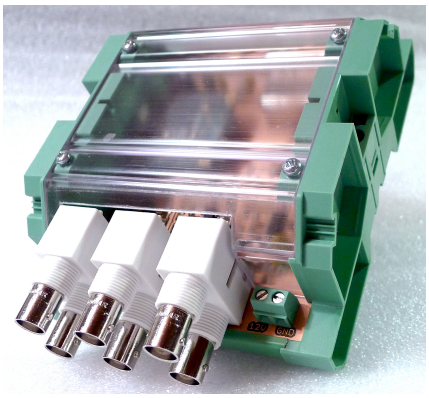
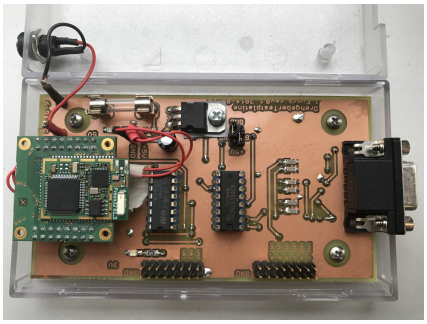
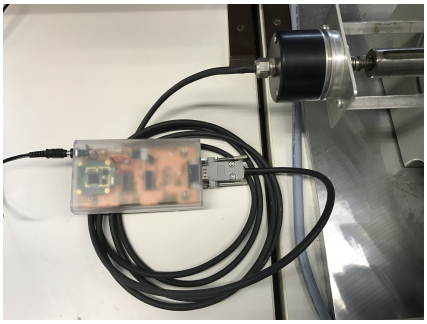


Figure 5.7: Conditioning board with housing [11]

board which is shown in figure 5.8(a), is designed by another project. The construction of the rotor speed acquisition system is shown in figure 5.8(b). A preon32 sensor node is inserted on the board which is powered by 12 V and connected with the rotary encoder via serial port.



(a) Conditioning board for rotary encoder [11]



(b) Construction of the rotor speed acquisition [11]

Figure 5.8: Hardware of the rotor speed acquisition

5.2.2 Analog Sensor Data Acquisition

Preon32shuttle provides two 12-bit ADC with maximum sampling rate of 1 M samples/s. As the data acquisition system is out of the scope for IEEE1451, self-defined measurement interfaces are used. Analog sensor data acquisition system is one part of the

MSTL library. The structure is shown in figure 5.9:

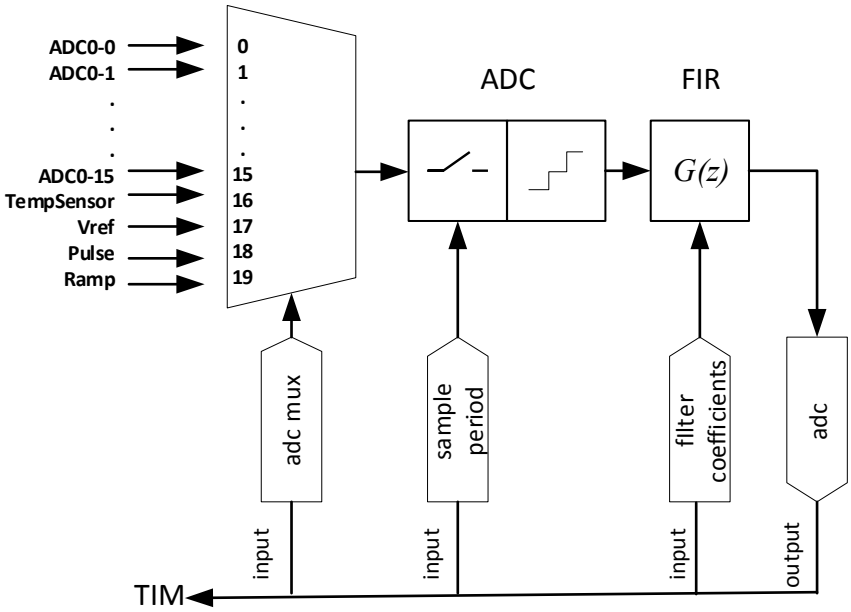


Figure 5.9: Block diagram of the analog sensor data acquisition system

The whole analog sensor measurement is set by six transducer channels, which is the same function as the processor register. An analog multiplexer is used to select the input to the ADC. After that the signal is sampled and converted to digital data. The output noise of the ADC is filtered. The operations of the analog sensor are listed below:

- adc mux: setting the number of input multiplexer
- mode: setting the data acquisition mode (*readData* or *startStreaming*)
- block size: setting the block size in samples
- sample period: setting the sampling period in μs
- filter type: setting the implemented digital filter type
- filter coefficients: setting the coefficients of the FIR filter (Q15-fixed point format)
- reading the output data of the analog-sensor

The table 5.1 below shows the currently supported inputs of the analog acquisition board.

Table 5.1: Supported inputs of the analog acquisition board

Mux. Channel No.	Description
0-15	adc input channels 0-15
16	internal temperature sensor of Preon32 (currently not available)
17	internal voltage reference of Preon32 (currently not available)
18	simulated unity impulse signal
19	simulated ramp signal

The *WriteData* command is called to configure the information of these channels. The *readData* and *startStreaming* can be used to read the data of the channels in one block and periodically. As the three-phase currents and voltages are acquired by one ADC in a single sensor node, the switching time of the analog multiplexer should be considered. An experiment is performed to calculate the switching time which is resulted from the multiplexer. The time differences when three-phase currents or voltages are sampled from different channels at the same time is shown in figure 5.10. $t1$ is the ideal time for the data acquisition. However due to the switching between the channels, phase u is sampled at time $t1$, phase v is sampled at the time $t2$ and w is at $t3$. The sampling time is 500 μs and the switching time Δt is about 5 μs. From the testing, the maximum difference during the switching time is 0.45 V for voltage measurement, 0.012 A for current measurement. The differences can be ignored during the data acquisition.

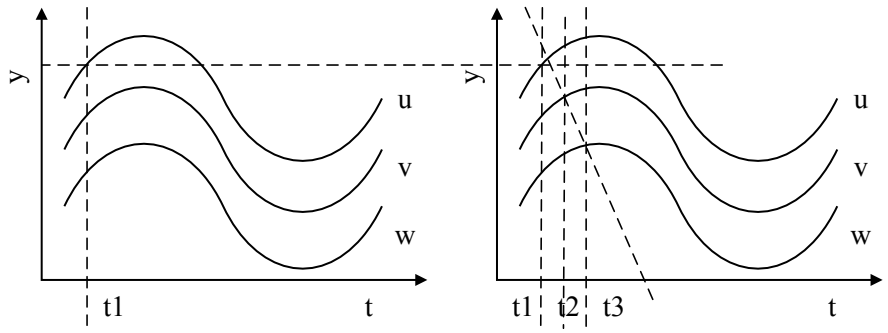


Figure 5.10: Time differences resulted from multiplexer

5.2.2.1 The Transducer and Buffer Management

All acquired data is converted to digital values by ADC and is stored in the *adc_buffer* with maximum array size of 4096 samples. In order to be processed continuously, the acquired data is statically stored in a block buffer which is allocated using Contiki specific allocator MEMB [86]. The block buffers are managed by linked list of Contiki [87]. The original converted data in *adc_buffer* is transmitted to the blocks allocated by MEMB. Three blocks are managed in the queue by using series of list management functions. The mechanism of the data management shown in figure 5.11 is based on the First-In-First-Out (FIFO) principle. Therefore the data will be processed block by block.

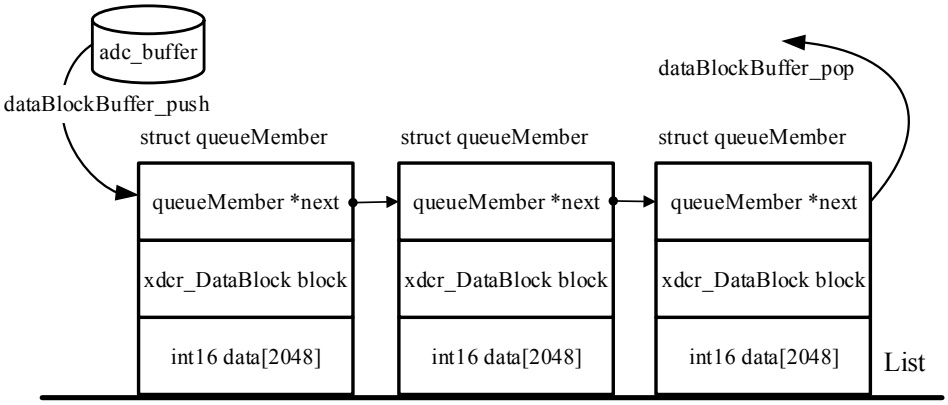


Figure 5.11: The structure of the data queue

If only one channel is used for acquisition, all the data is stored in the data block array, which obey the rule FIFO. If several channels are enabled to acquire the data, data are also stored in the one-dimension array. However it can be taken as a matrix with m rows and n columns. In the application, signals from maximum 8 channels could be acquired simultaneously. When filtering the signal, digital signals are fetched from the data block by the pointer and are filtered channel by channel. The structure of the data block is shown in figure 5.12:

5.2.2.2 Digital Filter Design

As is described in 5.2.2, analog signals are filtered by analog signal filter which implemented on the conditioning board, and converted to digital signals by ADC. There would be still much noise in the digital signals even if they are filtered with analog signal filter. So the digitized signals should be filtered using digital filter again before further processing.

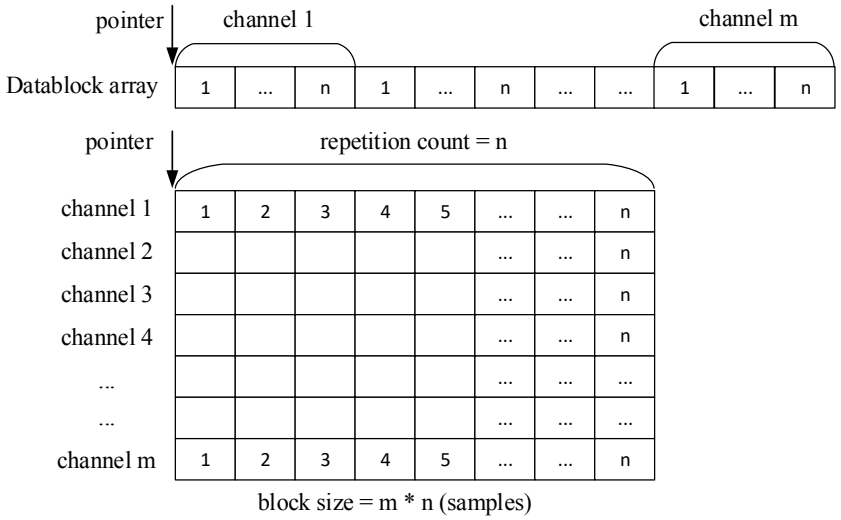


Figure 5.12: The structure of the data block

In a digital filter, the signal is represented by a sequence of binary numbers, rather than a value of voltage or current.

There are several types of digital filter, such as IIR and FIR. Before a digital filter is designed, a set of filter specifications should be defined. As the current, voltage and temperature are low-frequency signals, low-pass or bandpass filter could be selected. In order to define the cut-off frequency, signals are acquired by NI PCI-6023E data acquisition board. The signals are processed both in time domain and frequency domain by FFT (Fast Fourier Transform) in MATLAB. Low-pass filter is designed to filter the signals by using MATLAB.

As there is no FPU in the Preon32, fixed-point arithmetic can be used when implementing the FIR filter. ARM CMSIS contains a DSP (Digital Signal Processor) library which provides different types of digital filters with different format of fixed-point numbers. The resolution and the range of the converter depends on the number of bit of the ADC. The Preon32 is equipped with a 12-bit successive approximation ADC, which converts the acquired analog signal value from 0 to 4095 (i.e. unsigned integer, totally $2^{12} = 4096$) and has an input range of $0 - 3.3$ V. According to the range of the converted value, unsigned int data type with the range $[0, 65535]$ can cover the converted range. FIR filter function *arm_fir_q15()* which is declared in Appendix B.1 is called to filter the signal. The coefficient of the filter can be acquired by filter designer in MATLAB. The coefficient with fixed-point and the declaration of the filter function is listed below in Appendix B.2:

As data will be acquired, filtered and transmitted from TIM to NCAP continuously, the

calculation time of the filter should be considered. For one block measurement, calculation time of the filter is not so important, because only one block would be operated. If it is periodic measurement, data would be processed continuously, so the calculation time of the filter should be shorter than the acquisition time for one filtered block. The detailed signal processing time division is shown in figure 5.13:

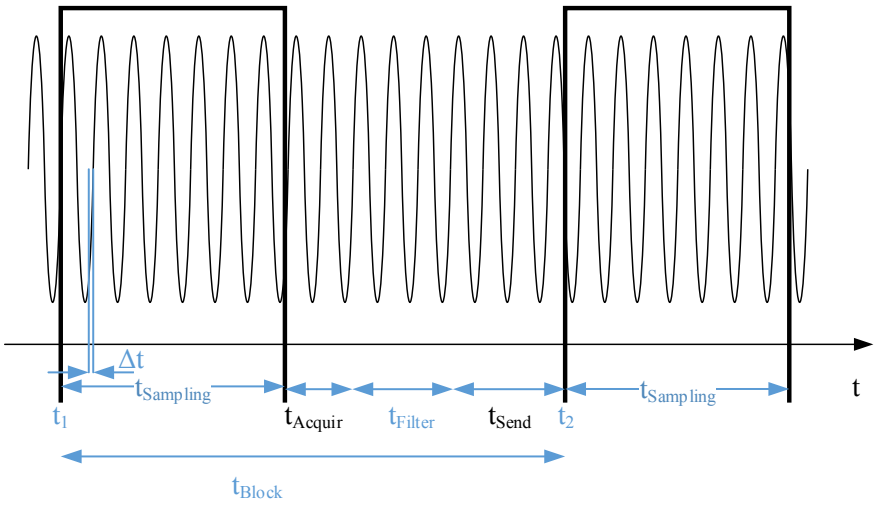


Figure 5.13: Detailed processing time division [11]

where:

- Δt: Sampling time
- $t_{Sampling}$: Sampling period
- t_{Block}: Total time for one block
- $t_{Acquire}$: Time for acquisition and conversion
- t_{Filter}: Time for digital filtering
- t_{Send}: Time for signal packing and sending

By using the function *etimer* which is provided by Contiki OS, the time consumption can be calculated. Taking the following settings as an example, sampling time Δt is 500 μs with 8 channels and 16 repetition counts, totally 128 samples/block. The sampling time $t_{sampling}$ for one block is 8000 μs. The data acquisition and conversion time including the switching time of the ADC multiplexer is 1200 μs. The computation time of the digital filter t_{filter} for one block is 1800 μs. The time for data packing and sending t_{send} is 3090

μs. The sum of $t_{sampling}$ and $t_{acquire}$ is $t_{sampling} + t_{acquire} = 9200\,\mu s$, which means that it will take 9200 μs to put the acquired data in ADC buffer. The sum of t_{filter} and t_{send} is $t_{filter} + t_{send} = 4890\,\mu s$, which is shorter than the total acquisition time. As a result, analog data acquisition system could process and transmit the data periodically.

5.2.2.3 The Chain of the Analog Data Acquisition System

In order to improve the utilization rate and reduce the load of the wireless transmission, several pre-processes are performed in the WTIM. Separate conditioning board is designed for the acquisition of the effective current, voltage and coolant air temperature.

A. The Measurement of Current and Voltage

The complete measurement chain of current and voltage is shown in figure 5.14. Firstly, three-phase analog currents and voltages are filtered by an anti-aliasing filter which is on the conditioning board. Then analog signals are acquired and converted to the digital signals with the sampling rate of 2000 Hz. A low-pass FIR filter is used for filtering the digital signals and passing the filtered signals for the RMS calculation. The *arm_rms_q15* function provided by DSP is used for the effective value calculation every 40 samples. Because 120 samples/block has almost the same value as that of 40 samples/block, 40 samples/block is selected. In this way, the frequency is reduced by 40 times to 50 samples/s. Another decimator is used for further frequency reduction to 10 samples/s. The energy consumption would be largely reduced due to the lower transmission frequency.

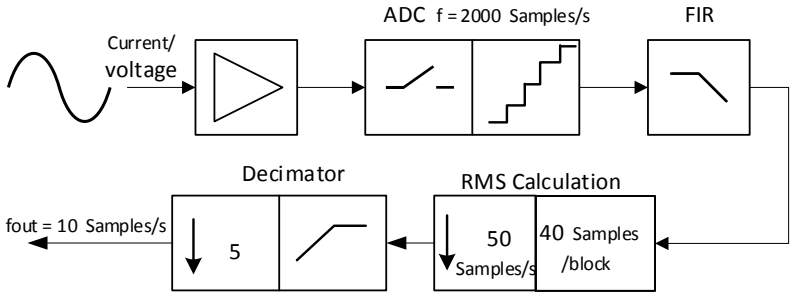


Figure 5.14: Measurement chain of the effective current and voltage

B. The Measurement of Coolant Air Temperature

The measurement chain of the coolant air temperature is shown in figure 5.15. The analog signal is filtered by an Anti-aliasing filter which is on the conditioning board. Then

signals are acquired and converted to the digital signal with the sampling rate of 200 Hz. A low-pass FIR filter (only one is used) is used for filtering the digital signals and pass the filtered signals for the Root-Mean-Square calculation. The *arm_mean_q15* function provided by DSP is used for the mean value calculation every 10 samples. In this way, the frequency is reduced by 10 times to 20 samples/s. Another decimator is used for further frequency reduction to 5 samples/s.

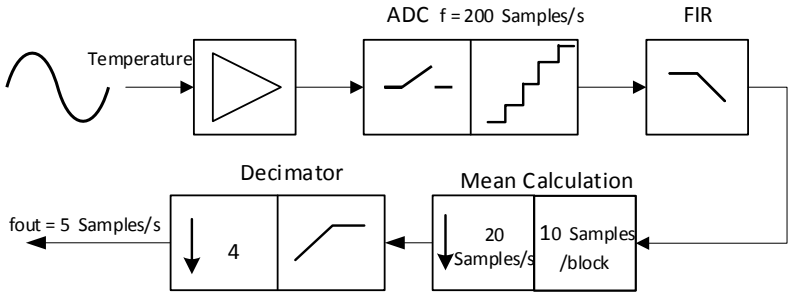


Figure 5.15: Measurement chain of the coolant air temperature

5.2.3 Digital Sensor Data Acquisition

Compared to an analog sensor, a digital sensor is quite different in which data conversion and data transmission takes place digitally. These digital sensors have many advantages: firstly, the sensor has an electronic chip which can directly convert the analog signal into a digital signal. The data transmission is not sensitive to characteristic of the cable, such as the length, the resistance or impedance, which leads that standard cables can be used. Secondly, the connection between sensor and cable can be connected by inductive coupling, so that humidity and related corrosion is no longer an issue. Furthermore, the sensor can be calibrated apart from the system [88].

In our application, an incremental rotary encoder is used to acquire the rotor speed. Incremental encoders generate pulses in a frequency proportional to the rotational speed. Only one of the incremental channels is used to measure the rotational speed n. The period between pulses of incremental channel A can be measured with Contiki *etimer*. The rotary encoder "ROD 426 B-6000" from HEIDENHAIN GmbH is used. For a 6000 rotary incremental encoder we divide 360 by 6000 to get 0.06. This means that there are 0.06 mechanical degrees of rotation for every incremental encoder pulse. The diagram of the generated 6000 pulses in one rotation is shown in figure 5.16:

The formula to calculate the rotation speed in RPM from the pulses can be defined

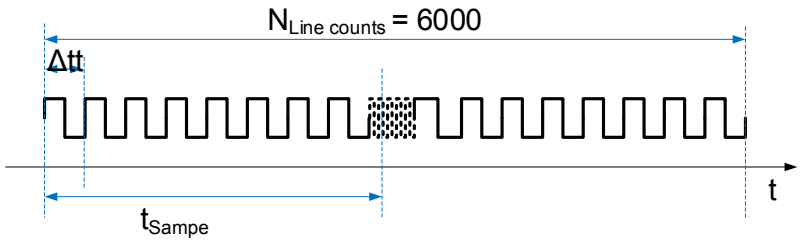


Figure 5.16: The diagram of the generated pulses [11]

in equation (5.1), where Δt is the time between two neighboring pulses, $N_{Line\ counts}$ is the number of encoder lines per revolution, $t_{Sampling}$ is the time period in one session in degree, which is 12 degrees for the encoder.

$$n = \frac{1}{\Delta t \times N_{Line\ counts}} \tag{5.1}$$

5.2.4 The Process of the Data Acquisition in WTIM

The general structure of the WTIM is shown in figure 5.17. *IEEE1451.5* process is to manage the radio module and to handle the communication of WSN. *IEEE1451.0* process represents the interlayer between the application and the *IEEE1451.5* process. It manages both the TEDS information and sampled data of the sensor. Both rotation sensor and analog sensor acquisition system have been implemented. Values in SI unit are sent back to *IEEE1451.0* process periodically as soon as the WTIM receives *startTrigger* or *startStream* commands.

The main functions of the *analog_sensor_continuous_process* can be summarized in the figure 5.18. This process will be started when receiving the *"Continuous"* sampling mode by *process_start* function. *analog_sensor_acquisition_prepare* function is used for initiating and opening the ADC. The process will wait until the polled event happens in a function which finishes transferring the pointer of *adc_buffer*. And then the sampled data will be queued in the linked list, processed and transmitted block by block in the loop. All the setup information of the transducer and the acquired data set are stored and updated in the structure *instance*. The setup information includes the driver of the transducer channel, the configuration data and the calibration coefficients. The data set information consists of type of the sensor, sampling period, timestamps of the first sample, number of repetition counts and the pointer to the sample. The structure instance which is transmitted and updated during the data processing is defined in Appendix B.3:

For the implementation of the digital sensor data acquisition, GPIO (General-purpose input/output) port on the preon32 is connected with the output cable of rotary encoder. The

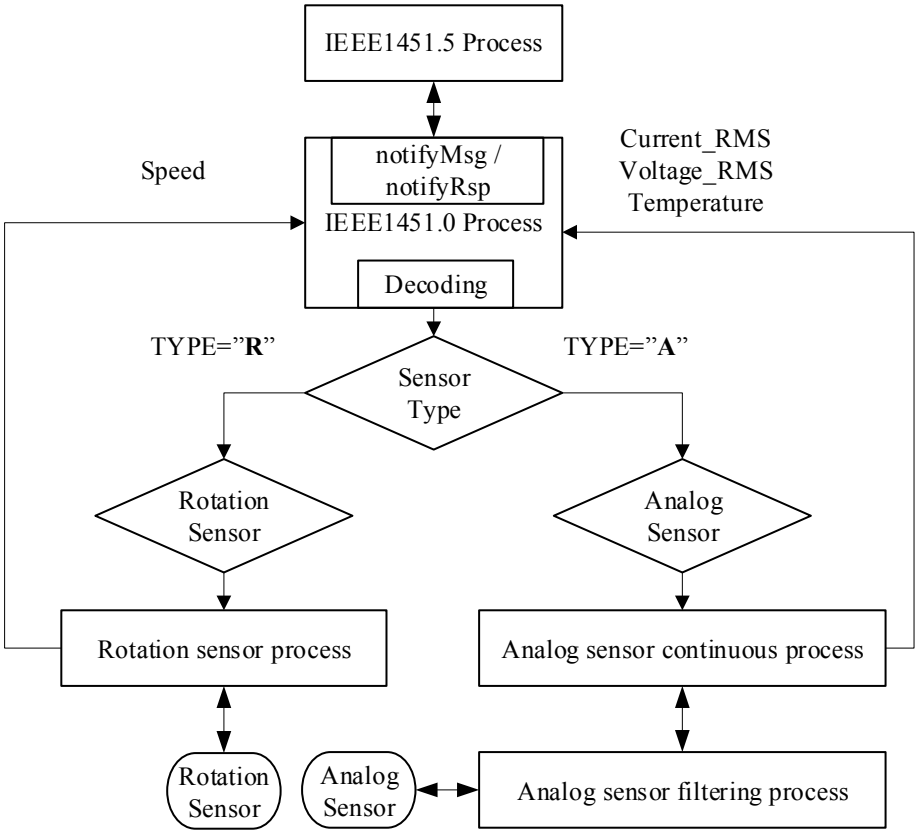


Figure 5.17: Workflow of the data acquisition system in TIM

conditioning electronic circuit has been designed by previous project. The workflow of the rotation sensor acquisition process is shown in figure 5.19, which could be summarized as follows: GPIO ports are first to be set for the acquisition of the digital signal and the interruption. $etimer\_set$ function is used for streaming interval event setting, which will be waited until the interval event finishes. The information of the pulses is stored in the *rotationDataSet* structure, which is defined in Appendix B.4.

The timestamps of pulses for every 12 degrees will be stored in $deltaT[1024]$ and be calculated to the speed in RPM. Then *sendDataSet* will be called to send the calculated mean value of speed to NCAP. After that $etimer\_reset$ is used to reset the timer event for next data streaming operation.

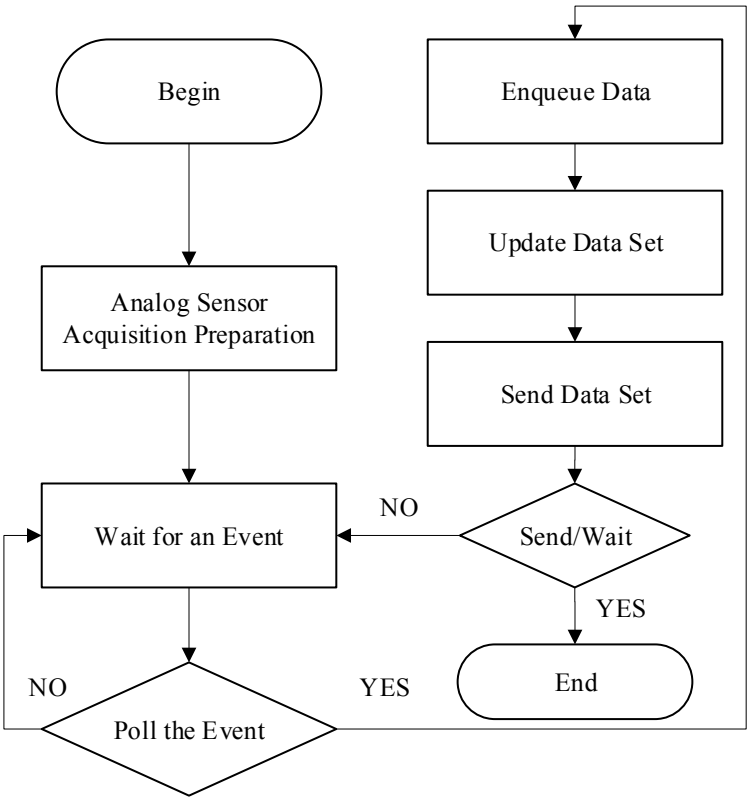


Figure 5.18: Analog sensor continuous process

5.3 Implementation of the KF Algorithm in NCAP

This section provides detailed implementation of the KF algorithm on IEEE1451 standard in NCAP. The minimum implementation of the IEEE1451 standard has been integrated in both in WTIM and NCAP, which consist the generic WSN. Sensors and actuators which are connected to WTIM can be managed by wireless commands from the NCAP. By using the $Request - Response$ process, users can send an HTTP request via internet to the HTTP server on the NCAP and get an HTTP response from the HTTP server [3]. In our application, the KF algorithm is integrated in the NCAP to estimate the temperatures of the stator winding, the rotor cage and the stator core of an asynchronous machine. The Preon32 sensor node is resource restriction such as low-cost, low-power, weak power calculation and small in memory size. In order to be implemented in the NCAP, the algorithm should be simple and efficient. The integration of the KF layer into Contiki system stack is

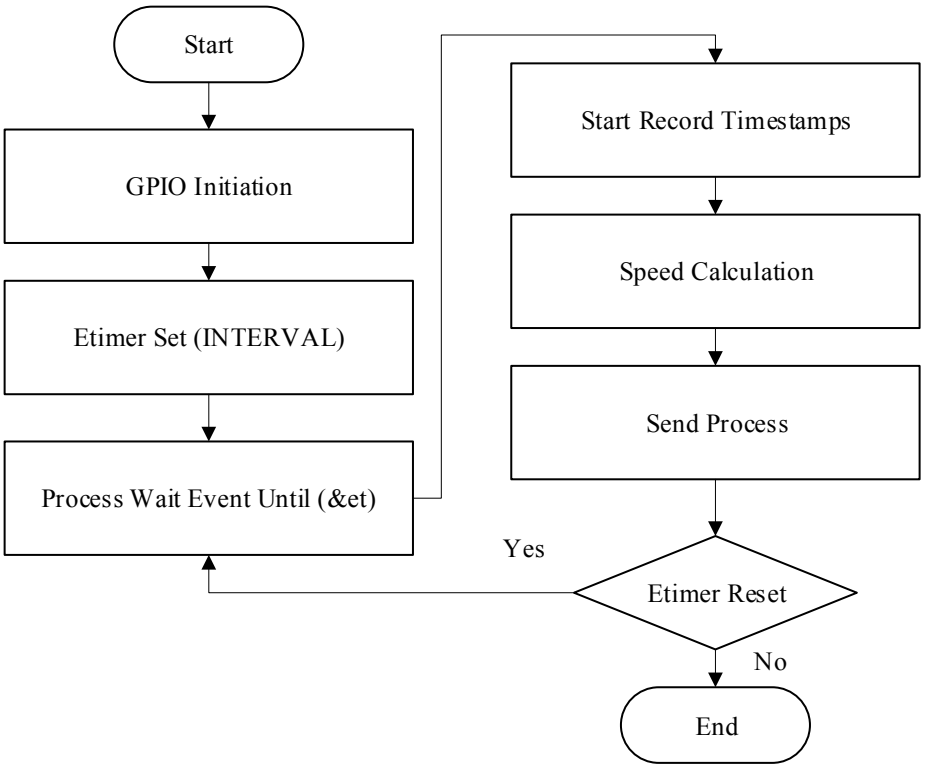


Figure 5.19: Rotation sensor data acquisition process

shown in figure 5.20:

6LoWPAN is defined encapsulation and header compression mechanisms that allow IPv6 packets to be sent to and received from over IEEE802.15.4 links [89]. It is an adaptation layer of IPv6 protocol for WSM. The 6LoWPAN protocol has been implemented together with IEEE802.15.4 Mac layer and IEEE802.15.4 physical layer by Contiki-OS. And the transport layer is responsible for data transmission from application layer between client and server sides. In the application, IEEE1451.0 and IEEE1451.5 standard are implemented which are compatible to the stack. KF algorithm is connected with the transport layer and application layer based on the API of IEEE1451 standard. The efficiency of the messages is largely improved and the overhead of the IP address is reduced by using the header compression in the UDP datagram. The $Request - Response$ model is used to communicate with NCAP and WTIM. Users can manage the WTIM by sending the commands to NCAP via internet, and NCAP will send commands to WTIM for the information. All the API and commands are defined in the standard [3] [28].

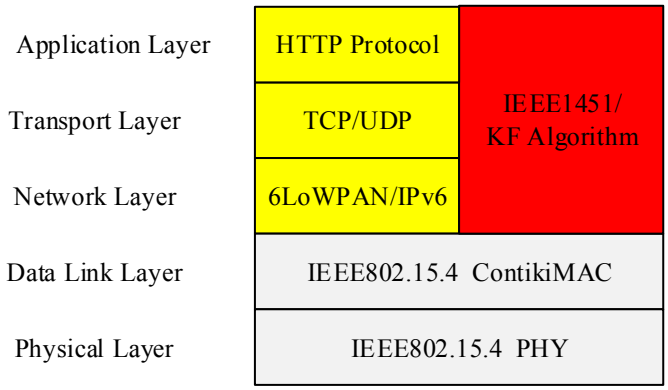


Figure 5.20: The integration of the KF into Contiki system stack of NCAP

5.3.1 The Implementation of Processes in NCAP

This section introduces the operational mechanism of the Contiki OS and the general structure of the processes based on the event-driven system.

5.3.1.1 Operational Mechanism of Contiki OS

In severely memory constrained environments, such as sensor nodes, a multi-threaded model of operation often consumes large parts of the memory resources. Each thread must have its own stack and because it in general is hard to know in advance how much stack space a thread needs, the stack typically has to be over provisioned. Furthermore, the memory for each stack must be allocated when the thread is created. The memory contained in a stack cannot be shared between many concurrent threads, but can only be used by the thread to which is allocated. Moreover, a threaded concurrency model requires locking mechanisms to prevent concurrent threads from modifying shared resources. To provide concurrency without the need for per-thread stacks or locking mechanisms, event-driven systems have been proposed. In event-driven systems, processes are implemented as event handlers that run to completion. Because an event handler cannot block, all processes can use the same stack, effectively sharing the scarce memory resources between all processes. Also, locking mechanisms are generally not needed because two event handlers never run concurrently with respect to each other [90].

In the Contiki operating system, processes are implemented as protothreads running on top of the event-driven Contiki kernel. A protothread is a memory-efficient programming abstraction that shares features of both multithreading and event-driven programming to

attain a low memory overhead of each protothread. The kernel invokes the protothread of a process in response to an internal or external event. The event may be a message from another process, a timer event, a notification of sensor input, or any other type of event in the system. Processes may wait for incoming events using the protothread conditional blocking statements. A protothread is stackless without a history of function invocations. Instead, all protothreads in a system run on the same stack, which is rewound every time a protothread blocks. The comparison between the stack memory requirements for three event handlers with protothreads and the equivalent functions running in three threads is shown in figure 5.21:

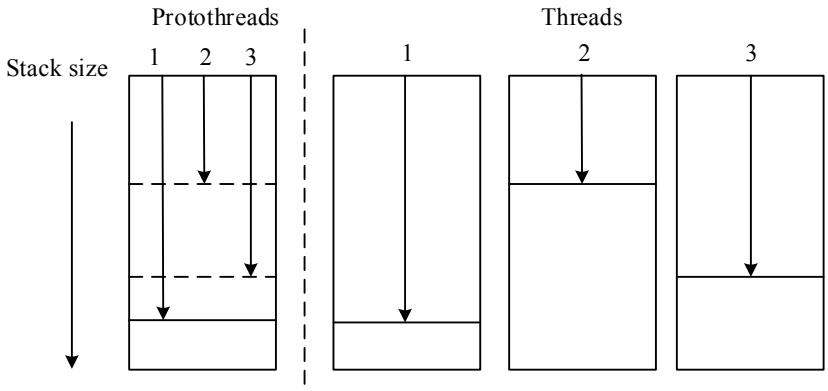


Figure 5.21: Stack comparison between protothreads and threads

All Contiki programs are processes. A process is a piece of code that is executed regularly by the Contiki system. Processes in Contiki are typically started when the system boots, or when a module that contains a process is loaded into the system. A process is run when it receives an event. Figure 5.22 shows the structure of process list. The next field points to the next process structure in the linked list of active processes. The name field points to the textual name of the process. The thread field, which is a function pointer, points to the process thread of the processes. The *pt* field holds the state of the protothread in the process thread. The state and needspoll fields are internal flags that keep the state of the process. The needspoll flag is set by the *process_poll* function when the process is polled and indicates the priority of the process.

A process thread contains the code of the process. The process thread is a single protothread that is invoked from the process scheduler. A protothread is declared using the macro which is listed the Appendix B.5.

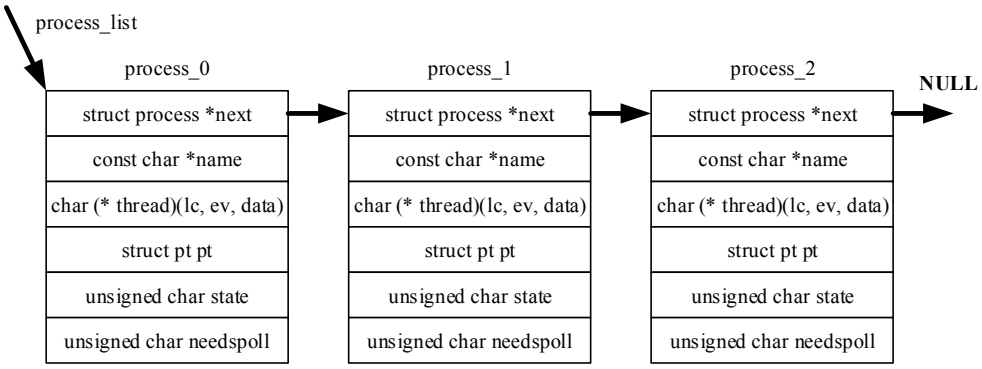


Figure 5.22: Contiki process linked list

The name argument should be replaced by the process **name** specified in the PROCESS macro. **ev** indicates the event generated the process execution and **data** indicates a pointer to the optional data passed as the event is generated. The process function must start with the macro *PROCESS_BEGIN()* and end with *PROCESS_END()*. These two macros work together, as they in fact implement a switch-statement. Between these two macros is the application user code.

The process code once executed will run until a specific macro is reached indicating a wait for an event. Some process-specific protothread macros are presented in table 5.2.

Table 5.2: Some process specific protothread macros

Protothread Macros	Description
PROCESS_WAIT_EVENT();	Wait for any event.
PROCESS_WAIT_EVENT_UNTIL();	Wait for an event, but with a condition.
PROCESS_YIELD();	Wait for any event.
PROCESS_WAIT_UNTIL();	Wait for a given condition.
PROCESS_PAUSE();	Temporarily yield the process.
PROCESS_EXIT();	Exit the process.

5.3.1.2 The Structure of the Processes

As is described in previous section, Contiki OS is an event-driven system which is managed by protothreads. In order to operate different WTIMs, manage the message transmission and process the KF algorithm, several functional processes are implemented in the

NCAP. The structure of the implemented processes are shown in figure 5.23:

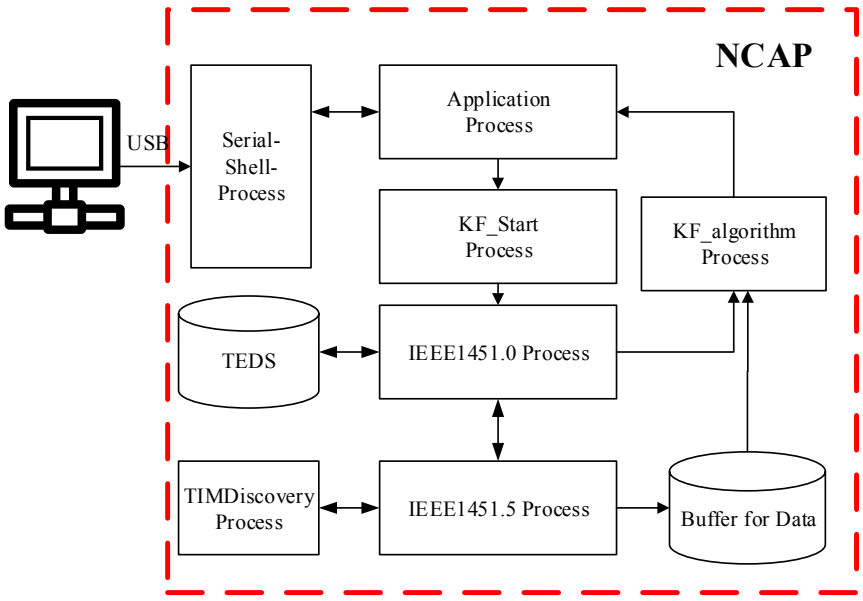


Figure 5.23: The structure of implemented NCAP [11]

The shell process has been implemented for connecting the WSN to an arbitrary network. A PC connected with NCAP is implemented as a server of the network. Users can manage the WSN by using a web based application or an App on the smart phone [24]. *TIMDiscovery* process is firstly used for discovering WTIMs before every command from application process. *KF_Start* process is used for configuring and initiating. *IEEE1451.5* process is implemented to manage the radio module and handle the data wireless transmission. The *IEEE1451.0* process represents the interlayer between *IEEE1451.5* process and *KF_Start* process. The buffer for storing data from different WTIMs is allocated in this process. The received I_{rms}, V_{rms}, ω_r, T_c will be passed to the *KF_algorithm* process for the temperatures estimation and the results could be sent out through serial shell process.

5.3.2 KF Algorithm Implementation in NCAP using Fixed-Point Arithmetic

The KF algorithm for temperatures estimation has been first implemented in MATLAB. It is proved that the temperatures can be accurately estimated both in simulation and off line experiment from test bench. In order to implement the KF algorithm on the resource restriction sensor node, the same algorithm is implemented in C programming language using floating point arithmetic on Eclipse IDE platform. The flowchart of the *KF_algorithm*

process is shown in figure 5.24 which can be summarized as follows: when the KF process starts, the system will retrieve and decode the messages from the messages buffer where different messages from different WTIMs are stored as inputs. The function *kf_data_gen* is called to calculate the losses P_{sw}, P_{rc}, P_{sc} from the measured current, voltage and rotor speed, and generate the inputs with T_c. The inputs are stored in the structure *kf_data* which is listed in Appendix B.6.

Input data is passed into the *run_kf* function where the main KF process is performed. The DSP library is used for the fixed-point matrix calculation. The state vector **x** and error covariance matrix **P** are set as the globe static array to store the values of previous calculation step. **x** and **P** are sent back to the next recursion. The estimation temperatures $\hat{T}_{sw}$, $\hat{T}_{rc}$, $\hat{T}_{sc}$ could be sent out for storage and display. $\hat{T}_{sw}$ is sent back to calculate the losses of the stator winding so that the resistance rise due to temperature could be compensated. The structure *kf_filter* in Appendix B.7 is an very important structure which holds all the KF filter related constants, variables and temporary values.

Compared to the implementation in MATLAB and on Eclipse in C language, implementation on the Preon32 sensor node using Contiki OS has several challenges which are described in the following subsection.

5.3.2.1 Memory Allocation for Messages

Firstly, the methods to allocate and free memory spaces are different between the standard C library and the Contiki OS. The standard C library allocates heap memory using *malloc* function. However, Contiki platforms specify a small area of their memory spaces for the heap because of the resource restriction [86]. If *malloc* function is used for memory allocation, the heap would easily be overflowed. The memb memory block allocator is used to allocate a block of static memory for struct *kf_data*{}, in which contains P_{sw}, P_{rc}, P_{sc}, T_c as the input of the algorithm, and another struct *kf_filter*{}, in which holds all the variables and matrices which would be used during the prediction stage and updated stage of the KF algorithm.

5.3.2.2 Fixed-Point Arithmetic

The difference between fixed-point numbers and floating-point numbers is that the decimal point is fixed or not. Many low-cost microprocessors, such as the ARM-Cortex M3 in Preon32 sensor node, do not have Floating Point Unit (FPU) and only give hardware support for integer numbers, and floating point operations have to be realized by the compiler which will largely reduce the computation speed [91]. In our application, Cortex-M3 processor is specially designed for WSN as it has outstanding control abilities. However, it is reasonable that this processor has fair computational ability and no hardware support for

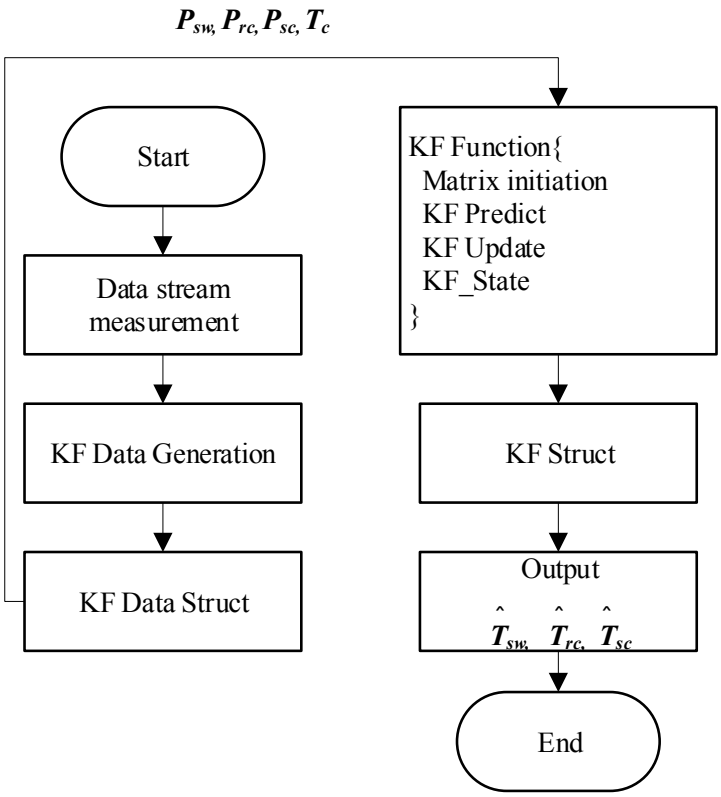


Figure 5.24: The flow chart of the KF algorithm implementation process

floating-point operations, because the applications in these aspects of control are usually light and rarely challenge its computation. The way to implementing floating-point operations on Cortex-M3 is to execute *"software floating point calculus"*, or short *"soft float"*, which fully depends on a compiler and software floating point library. Consequently, the efficiency of the soft float technique in floating-point operations is certainly lower than the efficiency of integer operations.

In order to implement the KF algorithm with so many floating point calculations in the sensor node, fixed-point arithmetic is a feasible way for the implementation [92]. The detailed description and implementation of the fixed-point arithmetic can be referred to Appendix B.4.

5.3.3 Fixed-Point Arithmetic Implementation

Based on the definition of the KF algorithm introduced in section 4.1, the algorithm has been successfully implemented in MATLAB. Matrix operations in MATLAB or a microprocessor with DSP library can largely improve the computation speed. The ARM Cortex-M3 processor provides the CMSIS DSP library, which contains matrix functions in fixed-point [93]. The matrix operation functions which would be used are listed in table 5.3:

Table 5.3: The matrix operations function

Functions	Description
$arm\_mat\_init\_q31()$	matrix initialization
$arm\_mat\_add\_q31()$	matrix addition
$arm\_mat\_sub\_q31()$	matrix subtraction
$arm\_mat\_mult\_q31()$	matrix multiplication
$arm\_mat\_trans\_q31()$	matrix transpose
$mat\_inv\_q31()$	matrix inverse

Every shift left, add/subtract or multiply can produce an overflow and lead to a nonsensical answer if the exponents are not carefully chosen [91]. Both the range and the resolution of the data are the key factors for choosing the type of Q-format. The system can avoid computation overflow by the saturation modes provided by CPUs, or by designing the arithmetic operations. The number of overflow checks is minimized by the division of variables by scaling factor SF, which scaled all the variables and auxiliaries to $[-1, 1-2^n]$. By checking the computation in MATLAB step by step, the maximum and minimum values of all the related variables in section 4.1 are listed in table 5.4:

Table 5.4: The data range of the variables

Constant	Max	Min	Variable	Max	Min
A_d	1	3.8857×10^{-4}	X	119.8216	0.0385
B_d	0.4995	5×10^{-5}	$\hat{X}$	119.8212	0.0228
K	0.2702	8.4325×10^{-6}	P	0.0270	8.4325×10^{-7}
Q	0.01	1×10^{-5}	$\hat{P}$	0.0370	1.1554×10^{-6}
R	0.1	0	U	318.2536	0
H	1	0	Z	35.3651	12.9985
τ	1	0			

Thus the Q0.31 format is used for the arithmetic with a resolution of 2^{-31} and a range of [-1, 0.999999999534]. This means that one bit is used to represent the sign within a

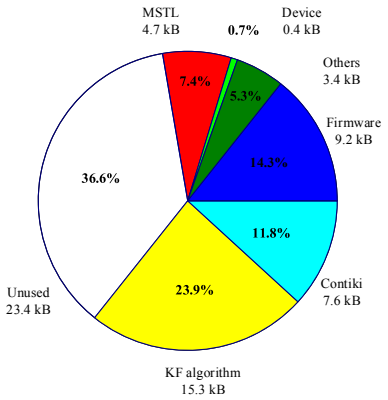


Figure 5.25: The usage of the RAM on the NCAP sensor node (total memory: 64 kB)

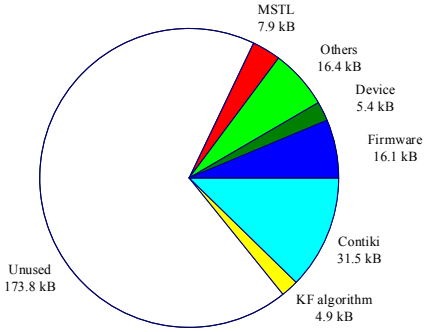


Figure 5.26: The usage of flash memory on the NCAP sensor node (total memory: 256 kB)

two's complement, no bits represent the integer portion and the remaining 31 represent the fractional part of the number [22]. By analyzing the structure of the KF equations in section 4.1, all the variables $X, \hat{X}, P, \hat{P}, U, Z$ and Q, R can be divided by 1000. In this case, all the values are in the range of [0, 1], and results of the matrix operations will also in the range [0,1]. The minimum value of number is 8.4325×10^{-10}, which is larger than the Q0.31 format resolution 4×10^{-10}. As a result, by scaling every equation with 1000, Q0.31 format can be used for all the constants and variables, and the matrix operation functions in table 5.3 can be directly used. The estimated values of temperatures can be transferred by timing the scaling factor $SF = 1000$.

5.3.4 Memory Usage and Calculation Time

In the implementation of the KF algorithm in NCAP, all the memory blocks are allocated statically so that fragmentation can be avoided. By using this way, it is easy to analyze the memory usage of both RAM and Flash. The usage of RAM on the NCAP sensor node is shown in figure 5.25. The buffers of the KF algorithm take up about 24% of the total memory space. The basic system, which consists of the Contiki OS, the firmware provided by Virtenio, and other parts from the standard C library, consumes about 32%. The MSTL library takes up 7.4%. About 37% of the space is unused.

The usage of the flash memory for programming on the NCAP is shown in figure 5.26. Only about 5% of the memory is used for the KF algorithm and the MSTL library. The system takes up most of the used memory. The rest of about 62% of the total memory is not used.

The system gets the data from different buffers to generate the input, which costs 120 μs and the computation time of the KF algorithm for one step is about 600 μs. The total time of data generation and KF computation is much shorter than the calculation interval 1 s.

5.4 The Communication between NCAP and WTIMs

The sequence on the NCAP side is shown in figure 5.27. The *TIMDiscovery* command which is defined in IEEE1451 standard is first used to discover the available WTIMs in the network. All the WTIMs will be registered with the *WTIM_IDs* which include the ID of the sensor node and the ID of the channel after being discovered. The *start_KF* function is called, the message is encoded and passed from the IEEE1451.0 layer to the IEEE1451.5 layer and then broadcasted to the WTIMs. Acquired data from different WTIMs is first stored in a queue in different buffers which is identified by the WTIM_ID. The data from different buffers will be fetched by data generation function according to the timestamps. Preprocessed data with the same or nearest timestamps will be passed and processed by the KF algorithm. Finally the temperatures are estimated and sent to the GUI.

The sequence on the WTIM side is shown in figure 5.28. The request message which is packed based on the standard is first received by WTIM. Then the message is notified and decoded by *IEEE1451.0* process. The data acquisition system can be triggered by the *startStream* command for continuous data acquisition. After the system is triggered, the signal will be acquired and converted from analog signal to digital signal by ADC. The low-pass FIR filter is used for filtering and other pre-processes are performed. For the pre-process of current and voltage, the RMS is calculated and the frequency is decimated to the estimation rate of the KF algorithm. The rotor speed is calculated to the unit in Revolution Per Minute (RPM) from the counting of the pulse in fixed time. The coolant air temperature is also calculated to the physical unit in °C. In this way, more useful information can be transmitted with less load for the WSN. The pre-processed data is sent back to the NCAP for the KF algorithm.

5.5 Conclusions

This chapter describes the implementation of the temperatures estimation system of induction machines on a WSN. The 4^{th}-order KF with fixed-point arithmetic is implemented in NCAP. Three WTIMs are implemented as the data acquisition systems. The fixed-point and floating-point KF algorithm is implemented. The experiments prove that the KF implementation is suitable for real-time temperatures estimation on a resource limited wireless sensor node. The KF temperatures estimation in WSN experiments will be described in

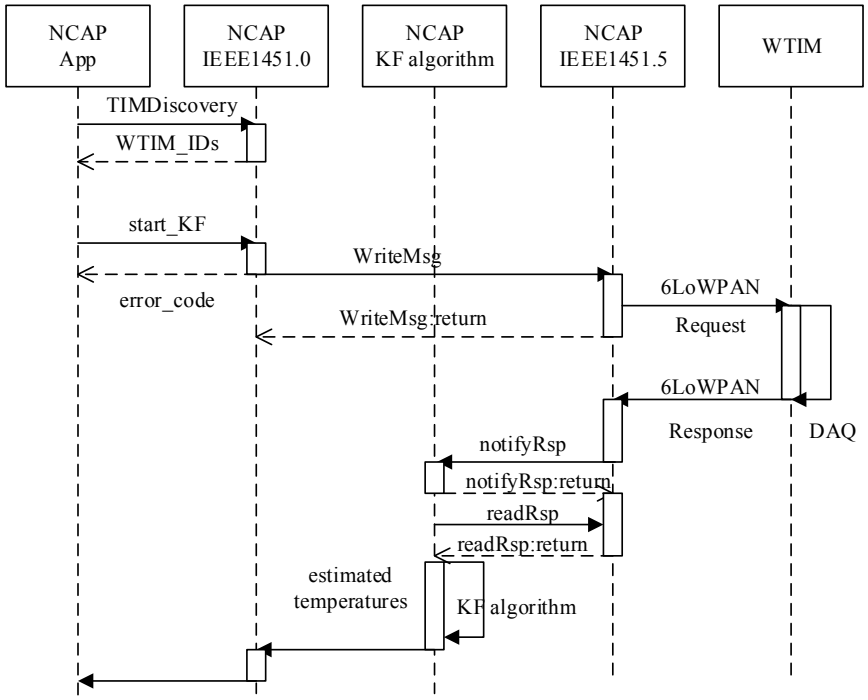


Figure 5.27: The sequence on the NCAP side

chapter 7.

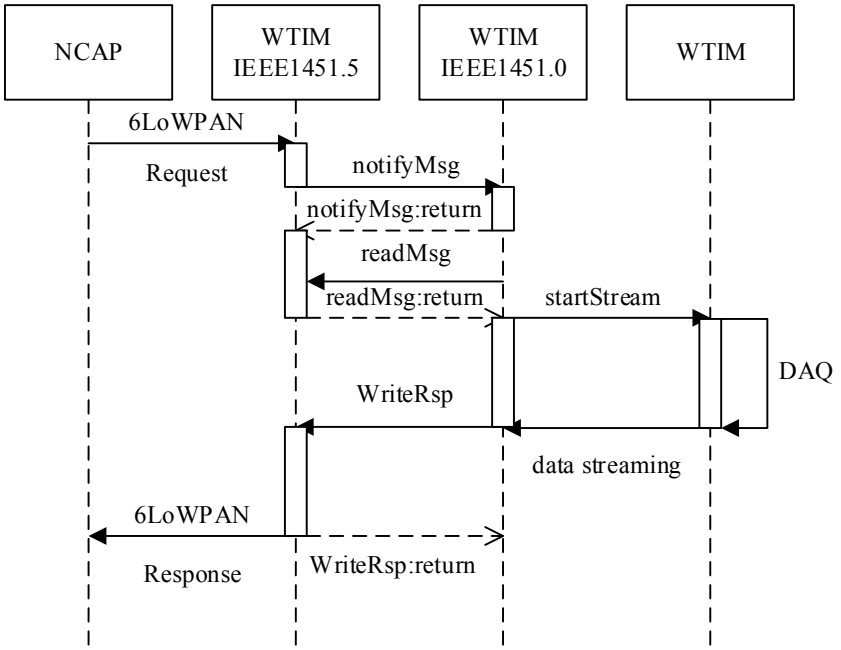


Figure 5.28: The sequence on the WTIM side

CHAPTER 6

THE IMPLEMENTATION OF THE EKF IN THE WSN

The 9^{th}-order non-linear EKF algorithm can be easily processed on a computer. However, due to certain resource limitations, such as low power computation and small memory size, a huge discrepancy exists between the high computing demand of EKF and the limited computational resource provided by Preon32. The implementation of the EKF in a sensor node means that the algorithm should be small and efficient enough in terms of complexity and memory consumption, so as to fit into the hardware platform. To implement EKF algorithm in the resource restricted WSN, optimization of the EKF in Contiki OS is introduced in section 1. Section 2 describes the detailed implementation of EKF algorithm in WSN. In order to make the system more stable and more accurate, faults handling and compensation mechanism are proposed in section 3. The conclusions are given in section 4.

6.1 Implementation and Optimization of EKF Algorithm in Contiki OS

The equations of the EKF for prediction and update have been introduced in section 4.2.1. Throughout this section, the implementation of the EKF in NCAP in C will be described in details. The numbers and matrices in EKF function are represented in fixed-point arithmetic, details of which are explained in section 6.1.1. The variables state of the system are stored in a structure *ekf_info*, which is introduced and illustrated in Appendix C.1. It contains all the information about the EKF, including the state vector $\boldsymbol{X}$, the control vector $\boldsymbol{U}$ and other filter relevant matrices. Meanwhile, two function pointers of linear filter function and linear measurement function are also included. In a specific application, the state dimension is 9 and measurement dimension is 2. All the matrices of EKF are defined in *MatFix32*, which is a customary structure for matrices in 32 bit fixed-point arithmetic.

There are two unique matrices, process covariance noise $\boldsymbol{Q}$ and measurement covariance noise $\boldsymbol{R}$ that are related to the convergence of EKF and which, to a certain degree, influence the accuracy of the result and convergence rate. According to the experiments, many factors can influence the convergence of the state variables, such as the type of the FIR filter,

the sampling rate. After many experiments and adjustments, the values of $\boldsymbol{Q}$ and $\boldsymbol{R}$ are determined as following:

$$\boldsymbol{Q} = diag \left[\begin{array}{cccccccc} 0.1 & 0.1 & 0.8 & 0.8 & 0.01 & 0.01 & 10^{-6} & 2 \times 10^{-6} & 10^{-6} \end{array} \right]$$

$$\boldsymbol{R} = \left[\begin{array}{cc} 0.01 & 0 \\ 0 & 0.01 \end{array} \right]$$

The function pointer *ffun* points to the function, in which matrices $\boldsymbol{A}$ and $\boldsymbol{F}$ are recalculated because of the linearisation of the non-linear dynamics around the prediction of the state $\boldsymbol{X}$. The linear measurement function, which is marked by the pointer *mfun*, calculates the errors between the measured and estimated values. As both of these functions are modified in different applications, they should be implemented in the application layer as *Callback functions*. The function pointers of both are passed as arguments to the operation layer, where several universal EKF functions are operated.

The life cycle of structure *ekf_info* is the same as each *inputDataSet*, which is transmitted from WTIM and received by NCAP. In other words, when a data package is received, NCAP generates a new structure *ekf_info*. When EKF finishes processing all the data in *inputDataSet*, the memory of *ekf_info* will be released. Hence, it is necessary to save some key information, such as the state matrix $\boldsymbol{X}$ and the state covariance matrix $\boldsymbol{P}$, by using the global arrays *X0* and *P0*. The initial state matrix of the previous *inputDataSet* should also be preserved in the global array *X0_1*, the purpose of which is presented in section 6.1.2.2.

After introducing some important structure and variables, the program architecture of the EKF process can be illustrated using a program flowchart that is shown in figure 6.1. When a new data package containing current and voltage arrives at NCAP, the function *run_ekf* (listed in Appendix C.2) will be called after some preparatory work.

step 1 Firstly, the coefficients in matrices $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{F}$ should be initialized based on the input *sample_period*, which, for hardware reason has slight difference between each data package even in the same sampling frequency in WTIM.

step 2 The structure *ekf_info* should be initialized and the memory block should be allocated, such as dimensions, matrices $\boldsymbol{B}$,$\boldsymbol{Q}$,$\boldsymbol{R}$ and function pointers.

step 3 A set of currents and voltages data in one sampling cycle and the current value of T_c should be imported into *ekf_info*.

step 4 Reset some key matrices by copying the values in the global arrays. The implementation of some methods mentioned later are also integrated into this function, e.g. estimation of changes in temperatures of unprocessed blocks, resetting the key

matrices if the rotor speed *n* and T_{sw} are abnormal, the matrices will be reset. The related function *ekf_reset* is listed in Appendix C.7.

step 5 Two equations for prediction would be calculated based on the fixed-point matrix operations library after recalculating matrices $\boldsymbol{A}$ and $\boldsymbol{F}$ by calling on the function pointer *ffun*.

step 6 Four equations for updating would be calculated based on the fixed-point matrix operations library, after calculating errors by calling up the linear measurement function through function pointer *mfun*.

step 7 Save the matrices of *ekf_info* into global arrays.

step 8 Repeat steps 3 - 7 until all the data in *ekf_info* is processed by EKF.

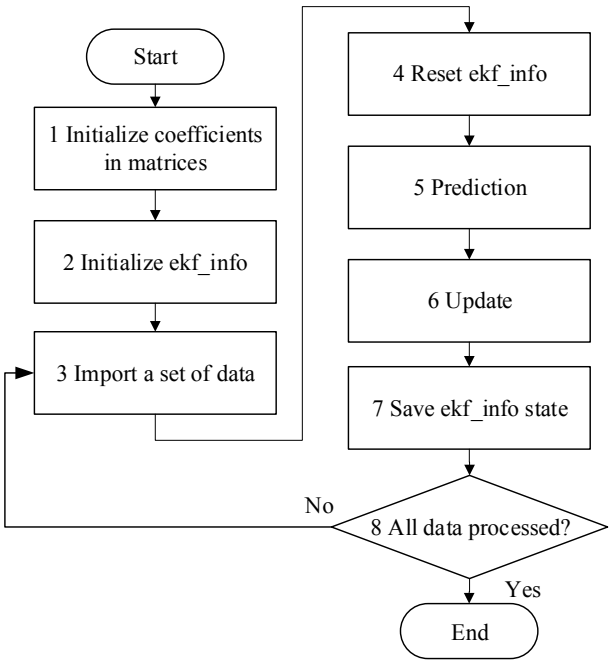


Figure 6.1: Program flowchart of the EKF process

6.1.1 Fixed-point Arithmetic

One of the algorithm optimization methods aimed for better computing performance is the introduction of fixed-point arithmetic into the EKF process. Most of the numbers involved in the EKF process, such as all the numbers in matrices are represented in fixed-point data type instead of floating-point data type.

Practically, it is impossible to implement the 9^{th}-order EKF using floating-point arithmetic in the Preon32 hardware platform for an online estimation. In the arithmetic conversion from floating-point to fixed-point, two main tasks are performed. The first is that some input values should be scaled beforehand and afterward in the interface layer that connects EKF to the external data transmitting process in NCAP. On the other hand, a series of macros and functions have been implemented as a library for conversion from or to fixed-point number and matrix operations.

Hence, in preliminary work some real-time consumption tests with both floating-point and fixed-point arithmetic have been performed based on the 4^{th}-order KF, which is a result of other related research.

6.1.1.1 Fixed-point Number Format and Scaling

The definition of the fixed-point number and the implementation of the fixed-point arithmetic can be referred to Appendix B.4. Unlike floating-point numbers, the resolution of a fixed-point variable in the *Qm.n* format will remain constant over the entire range, and the range is fixed if Q-format of this variable has been determined by the user. This inflexible characteristic of fixed-point numbers comes into conflict with the demand for a wider number range and a higher resolution, especially when the constants or variables of a certain application are widely distributed, as is case of this thesis. The coefficients in matrices $\boldsymbol{A}$ and $\boldsymbol{F}$ require resolution at the level of 10^{-10} (a coefficient in matrix A: $a_{95} = 2.06 \times 10^{-10}$), while the peak values of the input voltages are larger than 300. Thus, there is no 32-bits binary word in a single Q-format which can represent both the maximal and minimal numbers in EKF. One of the solutions is to represent different constants or variables in different Q-format but this compromises the efficiency of the calculation, which is unacceptable in this application. To reduce computational complexity, another feasible solution is to rescale certain matrices of the EKF or specific coefficients in matrices before the main EKF process.

In consideration of the matrix multiplication operations, each element in matrices $\boldsymbol{X}$, $\boldsymbol{U}$, X_{dot} , $\boldsymbol{P}$, $\boldsymbol{Q}$, P_{dot}, $\boldsymbol{E}$, $\boldsymbol{R}$ and $\boldsymbol{err}$ should be scaled down with a *scaling factor* of 1000. This way, all the large numbers, such as voltage and current inputs in matrices $\boldsymbol{U}$, $\boldsymbol{err}$, become smaller 10, or mostly smaller than 1. Meanwhile, some coefficients in matrices $\boldsymbol{A}$ and $\boldsymbol{F}$ should be scaled up in the initialization function to keep the matrices correct, because these two matrices are calculated in each iteration using the linear filter function based on scaled down matrix X. The coefficients of the first order should be multiplied by the scaling factor once, and the coefficients of the second order should be multiplied by the scaling factor twice. Take $A(7,1)$ in matrix $\boldsymbol{A}$ as an example:

$$A(7,1) = a_{71}X(0) + b_{71}X(0)X(6)$$

As $X(0), X(6)$ is scaled down with scaling factor of 1000:

$$X'(0) = X(0)/1000, \quad X'(6) = X(6)/1000$$

To obtain the same correct result of $A(7,1)$, the coefficients a_{71}, b_{71} should be scaled up:

$$a'_{71} = 1000 \cdot a_{71}, \quad b'_{71} = 1000 \cdot 1000 \cdot b_{71}$$

At the same time, the minimal value of the coefficients in matrices $\boldsymbol{A}$, $\boldsymbol{F}$ is increased to a level of 10^{-7}. Out of all constants and variables in the EKF process, the Q4.27 format is the best choice for this application with a range of $[-16, 16)$ and a resolution of 7.45×10^{-9}.

6.1.1.2 Number and Matrix Operations

The numerical relationship between a floating-point number A and its equivalent fixed-point integer B in the Q4.27 format can be easily obtained:

$$B = 2^{27} A \tag{6.1}$$

The number 2^{27} is actually the fixed-point value of float number 1. The macros converting to and from fixed-point numbers can be written in the header file *fix32.h*, which contains some fixed-point related definitions and macros. The result should be rounded to the nearest integer when converting to fixed-point numbers, which is listed in the Appendix C.3.

The addition and subtraction that operate on two fixed-point numbers has been described in Appendix B.4. The multiplication of two fixed-point numbers must take more into consideration. If A_3 is the product of two floating-point numbers A_1, A_2, and their corresponding fixed-point numbers in the Q4.27 format are B_1, B_2, B_3. Then the numeric relations among them, based on equation (6.1) are as following:

$$\begin{aligned} A_3 &= A_1 \cdot A_2 \\ \Rightarrow 2^{-27} B_3 &= 2^{-27} B_1 \cdot 2^{-27} B_2 \\ \Rightarrow B_3 &= 2^{-27} \cdot B_1 B_2 \end{aligned} \tag{6.2}$$

The multiplier 2^{-27} in equation (6.2) can be performed by the arithmetic right-shift operation, that is the result of $B_1 \cdot B_2$ stored as 64-bit integer should be shifted to the right by 27 positions and be inserted a copy of the sign bit (MSB) to the left. For higher computational accuracy the top decimal bit should be added to the lowest bit of the result as rounding operation, which is listed in Appendix C.4.

In this EKF application, fixed-point numbers are in the form of matrix involved in arithmetical operations. Hence, it is necessary to integrate a bunch of self-built relative light

matrix operations into the library to guarantee an efficient way of processing matrices. For this reason, some functions in the DSP library, such as the boundary check, errors output and others, do not take into account this in-depth customized library.

First of all, the matrix structure *MatFix32* in the Q4.27 format should be defined as and include information of the row number, column number and data as in Appendix C.5. Based on this structure a bundle of functions, including *mat_init*, *mat_add*, *mat_sub*, *mat_transpose*, *mat_mult*, *mat_scal_mult*, *mat_inv* are implemented into the matrix operation library, which is listed in Appendix C.6. Function *mat_inv* used in the calculation of equation (6.3) is more complicated due to issues of accuracy and scaling. To improve the accuracy of the inverse matrix, the temporary results should be stored as a *double* data type. Another problem is that numbers in the inverse matrix *E* vary across a wide range and are probably outside the range of the Q4.27 format. Clearly, these numbers should be scaled down to suit the Q4.27 format. However, it is difficult to select a fixed scaling factor when taking into account the accuracy, because numbers are scaled down at a compromise of accuracy of the final inverse matrix. To find each scaling factor, every number in the temporary inverse matrix of *E* should be scaled down in the ratio of 1/10 in a loop until all the numbers are within the required range. The final scaling factor is the production of the scaling factor in one step. The number of loops is a function argument that is transferred to the external EKF processing function. In the end, this scaling factor will be involved in the operation of equation (6.3).

$$K = P_{dot} \cdot H' \cdot inv(E)' \cdot ScalingFactor \tag{6.3}$$

The main operations of the EKF process are implemented based on the mentioned number and matrix operation library. The inputs and outputs of the EKF, usually in floating-point data type, should be converted to and from numbers in the fixed-point arithmetic before transferring into EKF. The floating-point data type is only applied in EKF process as a compromise on accuracy.

6.1.2 Sampling Block Method

The algorithm optimization method that is described previously effectively, reduces the computational complexity in NCAP. The NCAP performs an experimental verification of fixed-point arithmetic on Preon32, and becomes sluggish and then crashes in the tests after running for a quite short time. The time consumed by the EKF process in the error-free running period is recorded in table 6.1 below. It should be explained additionally that the comparative test of the 9^{th}-order EKF in floating-point arithmetic cannot run on Preon32 successfully. However, it proves the effectiveness of the fixed-point optimization method from another perspective.

Table 6.1: Time-consumption of 9^{th}-order EKF and sensor sampling time

	The time of EKF process	Sensor sampling time
Fixed-point	4.87 ms	0.5 ms

The result of the test clearly shows that the EKF process in fixed-point arithmetic takes nearly ten times longer than the sensor sampling data in one cycle. There is a wide gap between the sampling rate of sensors and processing speed of EKF, so that NCAP cannot handle the data processing tasks together with WTIM and crashes due to the time conflict of CPU. Thus, it is obviously not enough to optimize the algorithm in NCAP using only one method.

Another approach to algorithm optimization, otherwise known as *sampling block method* in this thesis, is to reduce the amount of data transmitted to NCAP for processing. In section 6.1.2.1, the theoretical basis of this method is derived from the thermal model while the optimization idea based on it is introduced. The specific implementation in WTIM and NCAP is explained in section 6.1.2.2 and some effects of this method are discussed in section 6.1.2.3 as well.

6.1.2.1 Theoretical Basis and Methods

First of all, the organization, storage, and transmission methods of the sampling data in WTIM are explained briefly. Based on related work, the analog sensor in WTIM is defined as a structure *analog_sensor_instance_t*, in which a sensor data set containing information of sensor channels, repetition count and sample period can be customized by user according to requirements. The sensor data set in WTIM_Motor uses six channels of Preon32 for collecting sampled data of three-phase voltages and currents. The sampling frequency f_s is 2000 Hz according to the requirement of EKF algorithm on input waveforms. To cover one complete period of voltages and currents (f_1 =50 Hz) in one data set, a repetition count N_{rep} is chosen as $N_{rep} = f_s/f_1 = 2000Hz/50Hz = 40$. These 6 channels, 40 repetition count sampling data make up a data block in the sensor data set which is stored in a 6×40 array of 16-byte integer.

The theoretical basis for sampling block method is that, the temperature changes in the asynchronous machine during a short time Δt can be simplified as $\Delta t = kT_1$, where k is a small positive integer and $T_1 = 1/f_1 = 0.02s$. The temperature of the stator winding T_{sw} can be taken as an example. In thermal equations (4.1) to (4.3), the angular velocity of rotation ω_m and effective values of $i_{qs}, i_{ds}, i_{qr}, i_{dr}$ are basically constant if the asynchronous machine operating condition doesn't change in the period of Δt. As a result,

P_{sw}, P_{sc}, P_{rc} are constants in Δt. The equation (4.1) can be defined as follows:

$$\begin{aligned}\frac{T_{sw}(t) - T_{sw}(t+\Delta t)}{\Delta t} &= \frac{1}{C_{sw}\Delta t} \cdot ((P_{sw}(t) - P_{sw}(t+\Delta t)) \\ &\quad - G_{sw}((T_{sw}(t) - T_{sc}(t)) - (T_{sw}(t+\Delta t) - T_{sc}(t+\Delta t)))) \\ \Rightarrow \frac{T_{sw}(t) - T_{sw}(t+\Delta t)}{\Delta t} &= \frac{G_{sw}}{C_{sw}} \cdot \frac{(T_{sw}(t+\Delta t) - T_{sc}(t+\Delta t)) - (T_{sw}(t) - T_{sc}(t))}{\Delta t}\end{aligned}$$

From the measured temperatures of the test bench, the maximal difference between T_{sw} and T_{sc} can be calculated based on the temperature curves:

$$T_{sw} - T_{sc} \leq 0.11(^{\circ}C/s) \tag{6.4}$$

Then the following result can be obtained:

$$\begin{aligned}T_{sw}(t) - T_{sw}(t+\Delta t) &\leq \frac{G_{sw}}{C_{sw}} \cdot |T_{sw} - T_{sc}|_{max} \cdot \Delta t \\ \Rightarrow T_{sw}(t) - T_{sw}(t+\Delta t) &\leq 1.12 \times 10^{-3} \Delta t(^{\circ}C/s)\end{aligned} \tag{6.5}$$

When k in $\Delta t = kT_1$ is a small positive integer in the range of $k \leq 25$, the Δt is within the range:

$$\Delta t = kT_1 \leq 0.5\ s \tag{6.6}$$

Hence, the difference between $T_{sw}(t)$ and $T_{sw}(t+\Delta t)$ will be:

$$T_{sw}(t) - T_{sw}(t+\Delta t) \leq 5.58 \times 10^{-4}(^{\circ}C/s) \tag{6.7}$$

The difference can be ignored in the practical application. The temperatures of other parts, such as T_{rc} and T_{sc} draw the same conclusion. In summary, under the same operating conditions, the rates of change in temperature of the asynchronous machine almost remain the same during a short period. As such, they can be identified as linear in Δt.

Based on the analysis, the temperature changes in the sequential data blocks of number k ($N_{rep} = 40$, sampling duration of one block $T_{dur} = T_1 = 0.02s$, $k \leq 25$) can be treated as the same under the same stable condition. Consequently, only one data block in a group should be processed by EKF and the temperature changes of the remaining data blocks can be estimated based on the outputs of first one. The processed data block chosen from a group of data blocks can be recognised as a *sampled data block*. Therefore, the optimization method based on this idea could be named as *sampling block method*. As illustrated in figure 6.2, the frequency of wireless data transmission from WTIM to NCAP is reduced by k times. Meanwhile, NCAP only undertakes the computational task of one data block in a group of k, and this task takes a relatively long time, based on the time

consumption in table 6.1. The rest $k-1$ data blocks are not transmitted to NCAP and the temperature changes of these abandoned blocks are estimated by NCAP based on the outputs of the last processed block. The specific implementation method in both WTIM and NCAP will be discussed in subsequent sections.

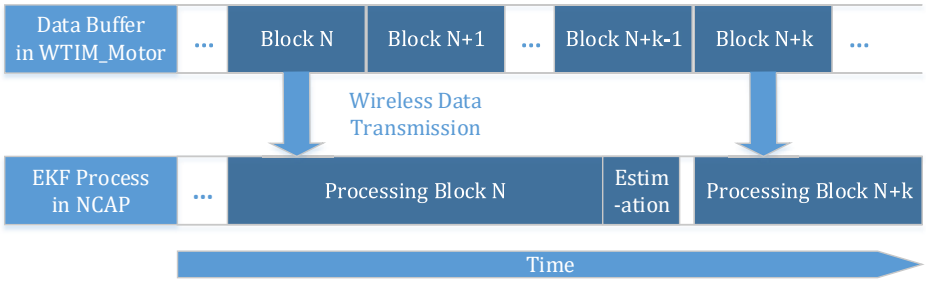


Figure 6.2: Sampling block method

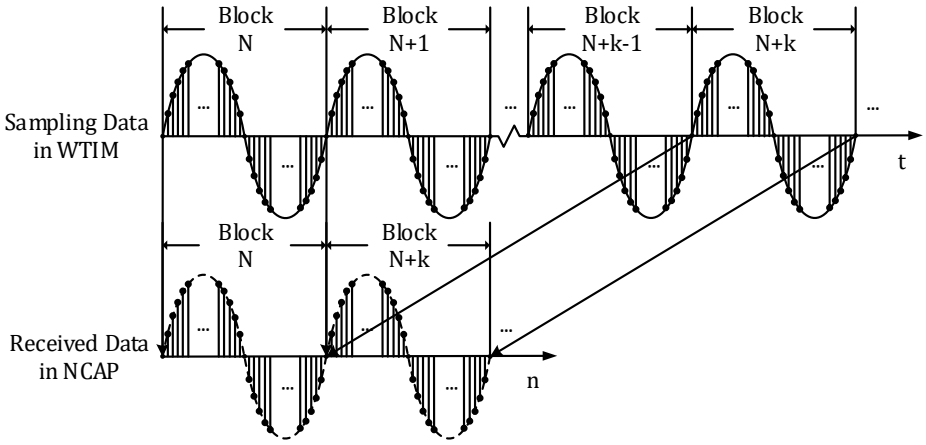


Figure 6.3: Sampling data in WTIM and received data in NCAP

In addition, for a stable iterative operation of EKF, the input data blocks N to $N+k$ should form up a continuous sine curve. This is why the repetition count of a single data block should be set at 40 to include the sampling data of a complete cycle. Figure 6.3 means that the inputs are reconstructed as 50 Hz digital sine waves in NCAP. However, the actual temperatures should be compensated in the discarded data blocks.

6.1.2.2 Implementation in WTIM and NCAP

According to figure 6.2, a frequency-reduced data transmission should be implemented in WTIM. A variable *blocksCounter* continues to count the data blocks pulled from the ADC conversion buffer until it reaches a user-defined constant *sendFreqDown*. Then, WTIM transmits the current data block to NCAP and resets *blocksCounter* to 0 afterwards. Therefore, the frequency of the transmission of the data block is reduced down to $1/sendFreqDown$ of original block sending frequency. The constant *sendFreqDown* should be obtained from real tests under the condition that NCAP runs smoothly over a long period of time.

Meanwhile, the main task for NCAP is to estimate the temperature changes of the unprocessed blocks as a compensation in the function *ekf_reset*. This function is invoked at the beginning of the EKF process, as shown in figure 6.1. For estimation of this group of lost blocks, two main pieces of information are the prerequisites:

- The numbers of the group of discarded data blocks: $N_{unproccedBlock}$
- The temperature changes of the latest processed data block

Under normal circumstances, $N_{unproccedBlock}$ can be easily deduced from *sendFreqDown* f_{sd} based on previous analysis $N_{unproccedBlock} = f_{sd} - 1$. During the real experiments, NCAP might stop working when being affected by the disturbances in the surrounding environment. It may result from the implementation of the relatively unstable transport layer protocol *UDP*, which is essential to this WSN application. The solution to this problem is given in section 6.3.3, in which the NCAP should restart under the condition of failure. Because the numbers of lost data blocks might be uncertain, $N_{unproccedBlock}$ should be calculated in another way rather than be defined as a constant. The EKF data set in the structure *ekfDataSet* is generated from a received data set in *xdcr_AnaDataSet*. Through the transformation of this data format, $N_{unproccedBlock}$ is derived from the timestamp, which is a member of the structure *xdcr_AnaDataSet* and indicates the clock ticks of the first sample in this data block from the beginning:

$$N_{unproccedBlock} = round(\frac{t_{stamp}|_{current\ block} - t_{stamp}|_{previous\ block}}{t_{second} \cdot T_{dur}}) - 1 \qquad (6.8)$$

where t_{second} is clock ticks of one second. The data format transformation and further calculations are discussed in section 6.2.3. As an advantage of this computational path, the changing $N_{unproccedBlock}$ contributes to a better robustness of the EKF as the adverse impact on the estimation caused by transmission failure and restart is basically eliminatedt.

Similar to the timestamp, the initial state matrix of previous processed data block should be saved as a global array *X0_1*. Then, with the current state matrix *X0*, which is the final

state of last sample in previous block saved also in a global array, temperature changes of last block can be easily deduced. For the purpose of improving robustness of EKF against disturbance, these state changes should be monitored within a safe range. This fault handling method will be presented in detail in section 6.3.2. The full code of this part is already listed in Appendix C.7.

6.1.2.3 Side Effects

There are indeed some side effects for disadvantages in the sampling block method when applied in the actual tests. Firstly, the tracking speeds of the rotational speed ω and rotational torque T slow down by k times. Because both ω and T are not increments in temperature, the rates of the rotational speed and torque change are zero when in a stable state:

$$\omega(t) = \omega(t + \Delta t) = 0, \quad T_e(t) = T_e(t + \Delta t) = 0$$

Therefore, there is no need for EKF to compensate for these two outputs of the discarded blocks in a stable state. In the stages of starting a machine or switching the machine load in S6, the lost data blocks affect how fast ω and T reach the steady state. However, they do not affect the steady accuracy of ω and T and temperatures. As there are no exact requirements for ω and T especially in regards to the tracking speed in this application, further optimization of the sampling block method isn't performed in this thesis.

Secondly, very slight temperatures estimation errors occur at the instant when the load in S6 switches. This is due to the lost of the data blocks. The temperature changes of the processed data block in one condition is different from the change of lost blocks in another condition:

$$T(t)|_{full-load(or\ no-load)} \neq T(t + \Delta t)|_{no-load(or\ full-load)}$$

However, these deviations are negligible if this switch time is closed to the next processed block.

6.1.3 Optimization of Memory Usage

The memory space left to EKF is insufficient so that it is necessary to reduce the overhead of large array and increase memory utilization. In this specific application, the structure definition during the matrix operations, which consist of a large array, should be minimized. Thus, the dedicated memory for temporary matrices can be canceled based on the concept *"Memory Pool"* which is shown in figure 6.4. This is also known as fixed-size blocks allocation, which is the use of pools for memory management that allows the allocation of dynamic memory. The application can allocate, access, and free blocks represented by users.

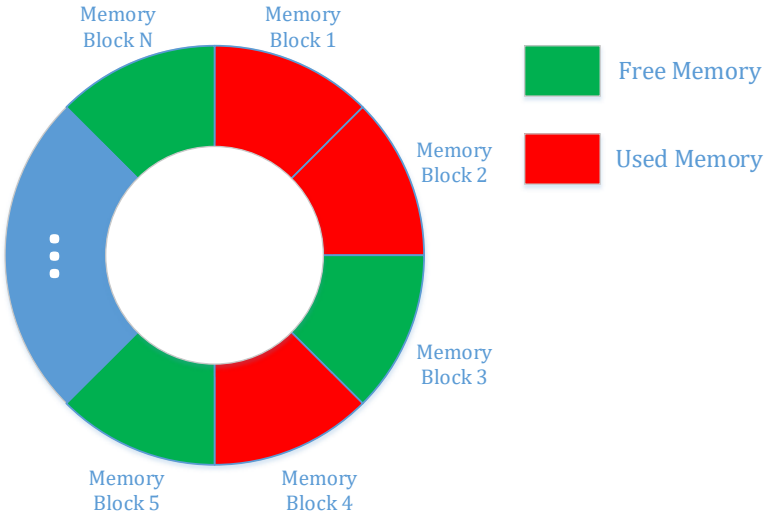


Figure 6.4: Memory pool

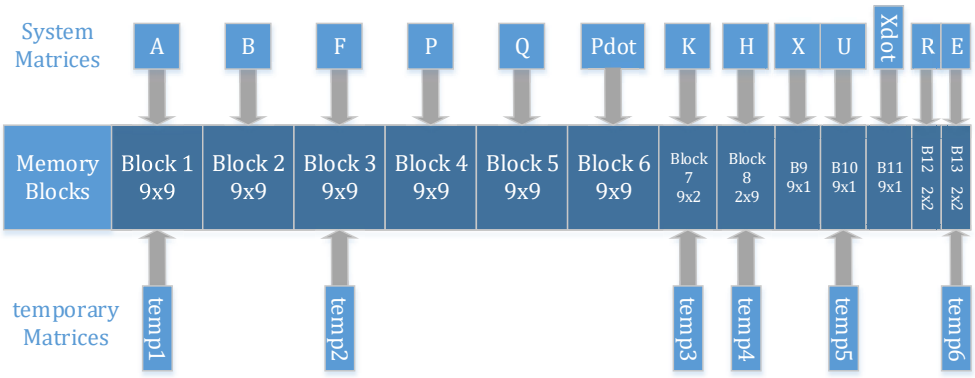


Figure 6.5: Memory mapping table of matrices in EKF

In the EKF process, all the memory requirements are known and operations are iterative at all time. Therefore, the memory allocations are practically statistically. However, the assigned memory of the system matrices in EKF are not always occupied, especially when this matrix related operations have not yet started or have already come to an end in the iteration cycle. Therefore, some ideas in *Memory Pool*, such as *memory block* and its *reusability*, by different variables can be introduced into the EKF application effectively. The temporary matrices can be allocated to system matrices of the same size, which is no conflict in memory usage. The specific memory mapping table of the EKF process is

graphically represented in figure 6.5.

6.2 Implementation of EKF Algorithm in the WSN

This section introduces the processes of the NCAP. The distributed WTIMs are adapted for the data acquisition and data processing. The integration of EKF in NCAP is also described.

6.2.1 The Processes of NCAP

With the except of the EKF process, which has already been discussed in section 6.1, other layers in this figure are explained in the following sections. The architecture of the EKF in NCAP is the same as the KF architecture in section 5.3, which is shown in figure 5.20. Several processes based on the IEEE1451.0 standard are invoked in NCAP:

- *timDiscovery*
- *startStream*
- *startTrigger*
- *measurementUpdate*
- *stopStream*

The time synchronization of the two WTIMs is triggered by the *startStream* process, which is also similar to the KF in NCAP. The flowchart of the processes in NCAP is shown in figure 6.6.

6.2.2 Adaptation for Distributed WTIMs Topology

The original process *MeasurementUpdate* in the Contiki application *shell-ncap* is responsible for retrieving the data from the data buffer of NCAP, where NCAP uninterruptedly stores measured data that is transmitted from WTIM. In the MSTL library, the process *MeasurementUpdate* works for only one WTIM and that is not enough for this application. Thus, process *MeasurementUpdate* should be modified in order to be adapted to the distributed WTIMs topology. Reception of the sampling data from both WTIM_Motor and WTIM_Temp should be organized properly to optimize the EKF operatiom by ensuring sufficient CPU time and memory.

Figure 6.7 illustrates the sequence diagram of the process *MeasurementUpdate*. As the air temperature of the coolant T_c changes slowly and has less impact on the estimation results of EKF, the update frequency could be set to a low value $f_{temp} = 2\ Hz$ In

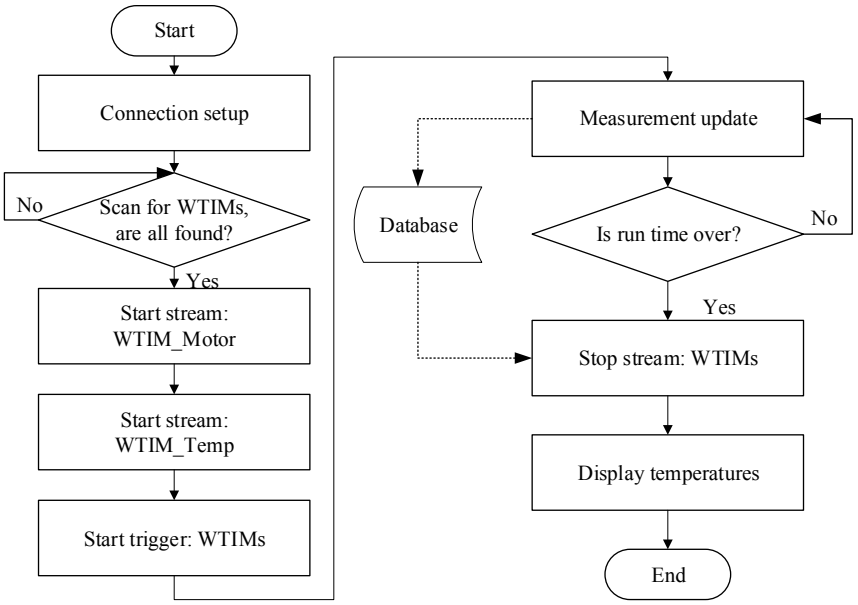


Figure 6.6: Flowchart of the processes in NCAP

addition, a default value T_c is initialized in the initialization process of NCAP. The data buffer, which is the storage spaces for temperatures from WTIM_Temp in data structure of linked list, should be overlooked. The old message will be overwritten by the new one when this data buffer becomes full. If the buffer has a new message or more than one messages, the message on the top would pop out from the linked list and then transferred to *tempDataReceived*, which is an octet array that is based on IEEE1451.0. The structure *tempDataReceived* comprises three members:

- length of the buffer in type of unsigned int (4 octets)
- maximal data buffer size in type of unsigned int (4 octets)
- data buffer (arbitrary number of octets), which casts a complete analog sensor data set of a transducer *xdcr_AnaDataSet* to octet array

T_c should then be updated according to *tempDataReceived*. That is, the data buffer of *tempDataReceived* should be cast from the octet array back to *xdcr_AnaDataSet*, as illustrated in figure 6.8. The structure *xdcr_AnaDataSet* is defined as Appendix C.8.

The value of T_c should then be calculated from the sample in *xdcr_AnaDataSet* and taking into consideration *AccuracyScaleFactor*. However, if the temperature buffer is empty, the current value of T_c won't be changed and can act as an input to the EKF process if necessary.

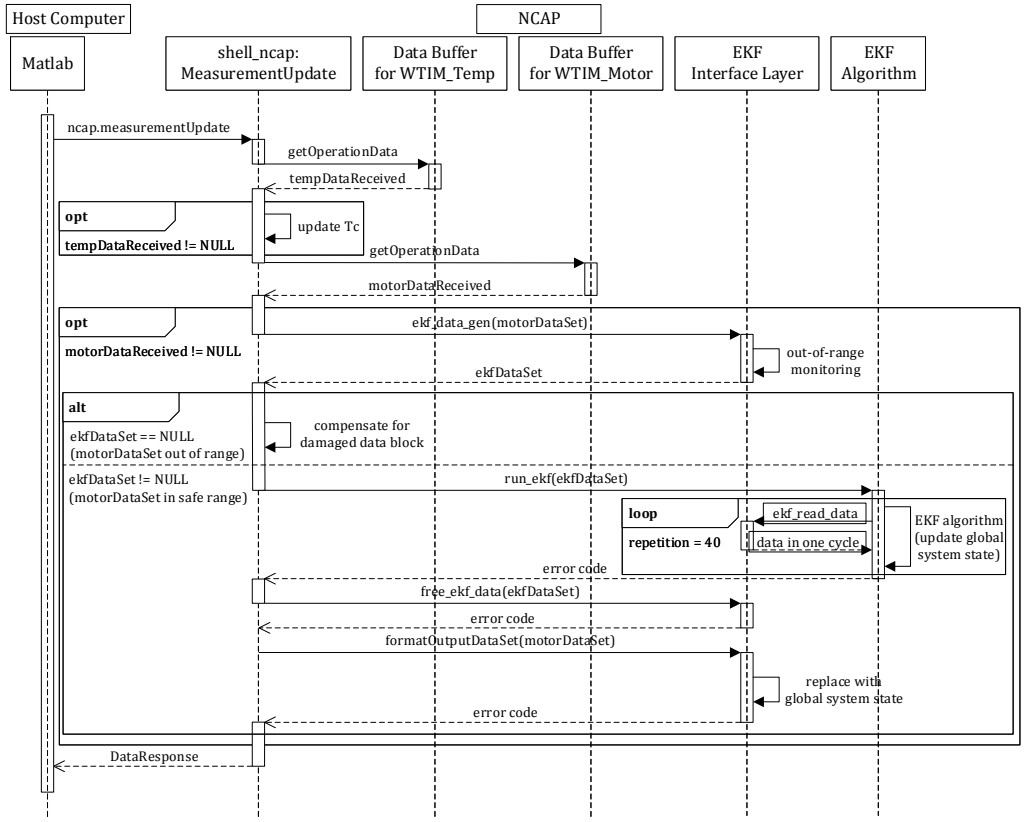


Figure 6.7: Sequence diagram of process MeasurementUpdate

In the next step, the buffer for the asynchronous machine data should be inspected in a similar manner to the temperature buffer. If the buffer length of the returned *motorDataReceived* is zero, this means that no new message arrives at NCAP, and the process *MeasurementUpdate* will send an empty data response back to the host computer. When the data buffer of the returned *motorDataReceived* is not empty, the EKF process should process should be invoked. The detailed integration of the EKF in *MeasurementUpdate* process will be discussed in next section.

To minimize the memory space, the maximal data buffer sizes of *tempDataReceived* and *motorDataReceived* should be set to the exact sizes that the data blocks require. The *xdcr_AnaDataSet* for temperature data block should have only one sample in each channel, while the data block should have 40 samples in each of the 4 channels. The sizes of other members in *xdcr_AnaDataSet* are fixed and can be put together as a constant value *XDCR_ANA_DATA_SET_HEADER_SIZE*.

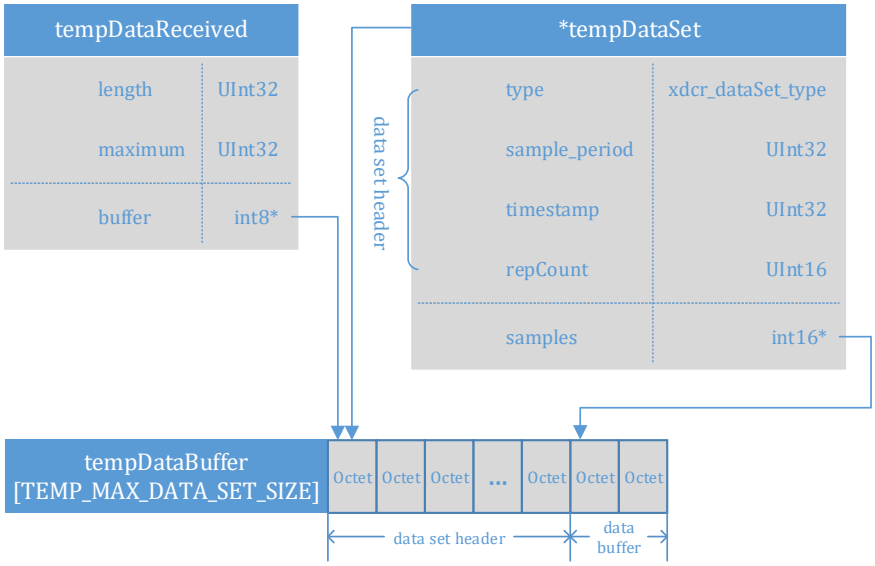


Figure 6.8: Data format transformation in process MeasurementUpdate

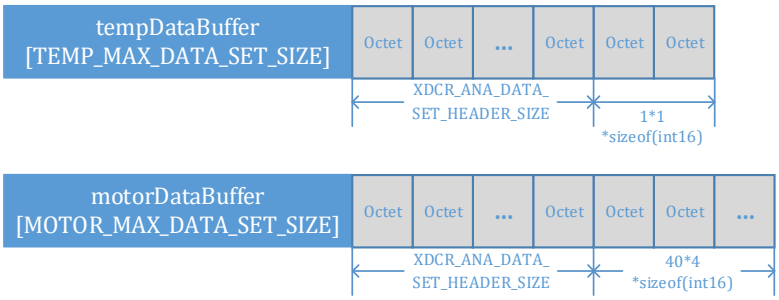


Figure 6.9: Sizes of temperature and data set buffer

6.2.3 Integration of the EKF

If the EKF process is invoked by the returned *motorDataReceived*, the communication between the EKF process and the *MeasurementUpdate* process should be implemented in the interface layer. At the beginning the data buffer of *motorDataReceived* should be casted to *motorDataSet*, which is then as an argument transferred to function *ekf_data_gen*. The return value of *ekf_data_gen* is structure *ekfDataSet* in Appendix C.9.

SamplesPointer points to the data of the first channel and is used to read data from *motorDataBuffer* in each iteration of the EKF process. In general, *ekf_data_gen* should be integrated with following functions:

- monitoring of the range of input data in *motorDataSet*: Once one sample is out of range, the whole data block is treated as damaged and the function *ekf_data_gen* will return "NULL" as a warning sign. The compensation method is presented in section 6.3.1.
- allocation of the memory for *ekfDataSet*
- initialization or calculation of the members in *ekfDataSet*

According to equation (6.8), *unproccedBlockNum* can be calculated based on the time stamps of current and previous *motorDataSet*. If the returned *ekfDataSet* isn't empty, the EKF algorithm *run_ekf* should be invoked with *ekfDataSet* as a function argument. In each iteration of EKF process a set of samples of 4 channels should be retrieved from *motorDataBuffer* using *samplesPointer* and be transformed according to certain scale factors for accuracy and fixed-point arithmetic. Meanwhile, the current value of Tc should be converted into temperature increment. Then two data arrays for system matrices Z and U can be assigned to data sets of both machine and temperature. *SamplesPointer* should eventually move to the next data of the first channel. The listing of data transmission of the measurement is shown in Appendix C.10.

After each iteration of EKF process, the system state will be saved into the global arrays $X0, P0$. The memory for *ekfDataSet* will be released after all the data in the block has been processed by EKF. The structure *motorDataSet* is also recognized as the output data set. Thus, the original data of the asynchronous machine in the data set should be replaced with 6 outputs of EKF (global array $X0[4-9]$).

6.3 Faults Handling and Compensation

Individual nodes are not reliable, as they may be disturbed by unstable environmental conditions. The system may be deployed in harsh electromagnetic environment and sometimes the message received by NCAP may be damaged. The raw data of the asynchronous machine in the real test shown in figure 6.10 reveals that environment may interfere with the received data in NCAP, hence leading to the uselessness of this data block or even worse, failure of the node. Thus, WSN itself must remain operational in these cases. Three methods are implemented in NCAP against the disturbance:

- monitoring of the range of input data in *motorDataSet* and compensate for damaged the data blocks.
- monitoring of the range of estimation changes of EKF and reset system state when outputs are out of range.
- restarting NCAP in case of node failure.

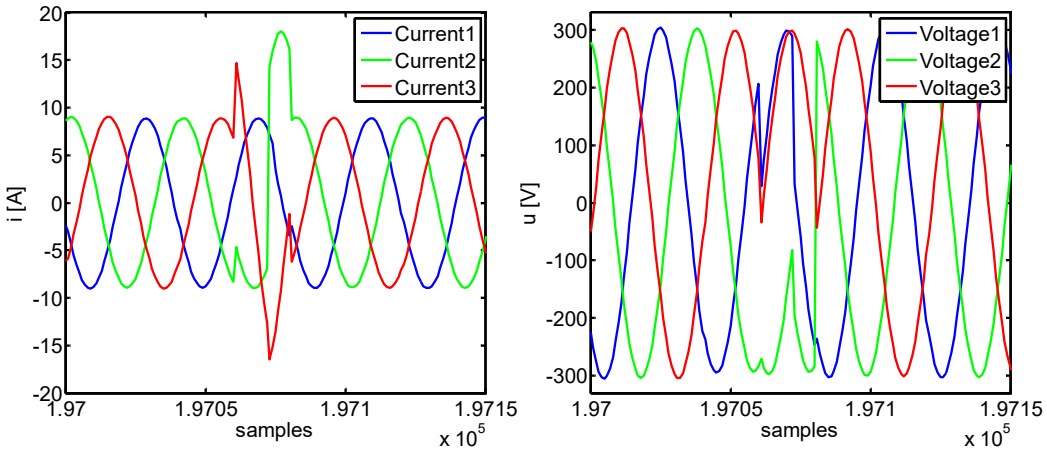


Figure 6.10: Received interfered data

6.3.1 Input Data Range Monitoring Compensation

The asynchronous machine data in *motorDataSet* should be monitored within the safe range:

$$-10\ A \leq i_{dq} \leq 10\ A$$

$$-315\ V \leq u_{dq} \leq 315\ V$$

If the returned *ekfDataSet* is NULL which means the received data is out of the safety range, the process *MeasurementUpdate* should make as much compensations as possible. Regardless of the damaged samples *unproccedBlockNum* can be derived from the time stamp as before. Therefore, the estimation of lost data blocks can still be compensated based on the sampling block method. As a compensation for the damaged block, its temperature changes can be approximately taken to be the same as the previous processed block, which is listed in Appendix C.11.

In most cases, the out-of-range data block has very little impact on the estimation after applying this compensation method.

6.3.2 Output Range Monitoring and Reset

By doing experiments, we found that the output changes of EKF should also be monitored with a safe range:

$$-300\ rad/s \leq \Delta\omega \leq 600\ rad/s$$

$$-0.2^{\circ}C \leq \Delta T_{sw,rc,sc} \leq 0.2^{\circ}C$$

When the changes are out of these ranges, the last processed data block can be regarded as a damaged data block. Thus the estimations of this block should not be accepted, even though the estimations are already delivered to the host computer. As a compensation, the current inaccurate system state should be set back to the initial state before processing the damaged data block by using *X0_1*. Figure 6.11 shows the resetting process of the temperature output.

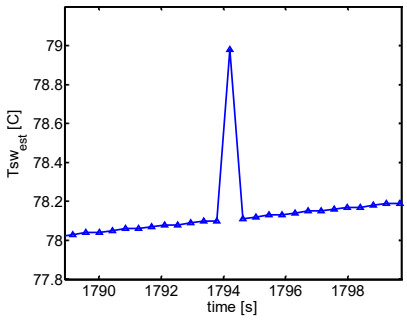


Figure 6.11: Reset system state in case of unacceptable estimation

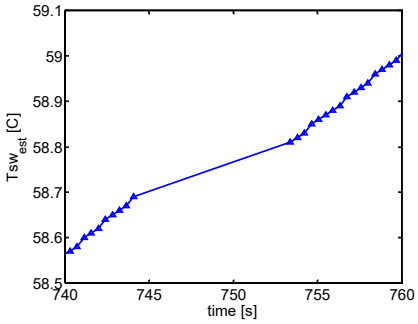


Figure 6.12: Temperature compensation for disconnection

6.3.3 NCAP Restart in the Case of Disconnection

Due to the interference, disconnection or packages lost between NCAP and WTIMs would happen. As a result, there is no estimation output from NCAP. It will be restarted when it reaches timeout. In the loop of invoking process *MeasurementUpdate* the variable *emptyCounter* counts the empty data response. If *emptyCounter* reaches the preset maximal number, NCAP could be considered as losing connections with WTIMs. It should then be restarted.

Because the system state is stored in global arrays and they won't be released during the restart. Therefore, after restarting NCAP the lost data blocks in this period can be compensated as well by calculating *unproccedBlockNum*. Figure 6.12 shows the temperature compensation for disconnections between NCAP and WTIMs in real experiment.

6.4 Conclusions

In this chapter, an EKF algorithm is first implemented in NCAP using C language. To be implemented in the resource restricted Contiki OS, the 9^{th}-order EKF with extensive computation is optimized by the following three methods:

- *Fixed-point arithmetic* contributes to the improvement of computational efficiency and the reduction in time of each operations.

- *Sampling block method* greatly reduces the amount of data block processing task on a basis of sectional linearisation of temperature curves.
- The *memory usage optimization* decreases the number of memory blocks for matrices based on the periodic free memory block in EKF operations.

The optimized EKF algorithm is implemented as a process and integrated into WSN in Contiki OS. A general discussion on the architecture and control program, as well as an elaboration on sequence diagram and some modifications of the most important process *MeasurementUpdate* are presented.

In the experiments, there are some interference during the wireless data transmission, which leads to packages lost. As a result, three mentioned faults handling methods against disturbance and other functions or processes are implemented to improve the robustness of the system and the accuracy of the estimated temperatures. The EKF temperatures estimation in WSN experiments will be described in chapter D.

CHAPTER 7

THE EXPERIMENT RESULTS

All the experiments are performed on the test bench. The structure of the whole system is shown in figure 7.1. The parameters of the asynchronous machine are listed in details in Appendix table D.2. The DC motor is fixed as the load to provide different load condition for the system. An integrated sensor (HBM T30FM) for the output torque and rotor speed are connected to the measurement box which converted the analog signals to digital signals. The value of the torque and the speed are sent to the computer for controlling. Three temperature sensors are mounted onto the machine to measure the temperatures of the stator winding, the stator core and the coolant air. The data acquisition board (NI PCI-6023E) from National Instrument is used for the temperatures measurement which would be compared with the estimated temperatures. Meanwhile, three-phase currents and voltages could also be acquired for the parameters identification. This is discussed in details in the master thesis [85]. The WSN is used to acquire the temperature of the rotor cage, which is designed in the Diploma-thesis [37]. WTIM1 is used to acquire the temperature of the rotor cage, WTIM2 for the rotor speed, WTIM3 for the temperature of the coolant air, and WTIM4 for the acquisition of the effective value of the stator current and voltage.

7.1 The Experiments of the KF Algorithm using WSN

The wireless sensor nodes WTIM2, WTIM3 and WTIM4 are used to acquire the rotor speed, the coolant air temperature, the three-phase currents and voltages. The KF algorithm is implemented in the wireless sensor node as NCAP to estimate the temperatures. The sampling time is 1 s. The sampling period is about 2 hours, after which the temperatures of the estimated parts remain stable. The ambient temperature is 26°C. The comparisons between the estimated and measured temperatures under S1 condition are shown below in figure 7.2. The maximal deviation of the stator winding is 2.3 K, the maximal deviation of the rotor cage is 3.5 K, and that of the stator core is 2.0 K. The maximum error and the NRMSE of the estimator are summarized in table 7.1.

During the experiment under the condition of intermittent-load S6, an unusually exces-

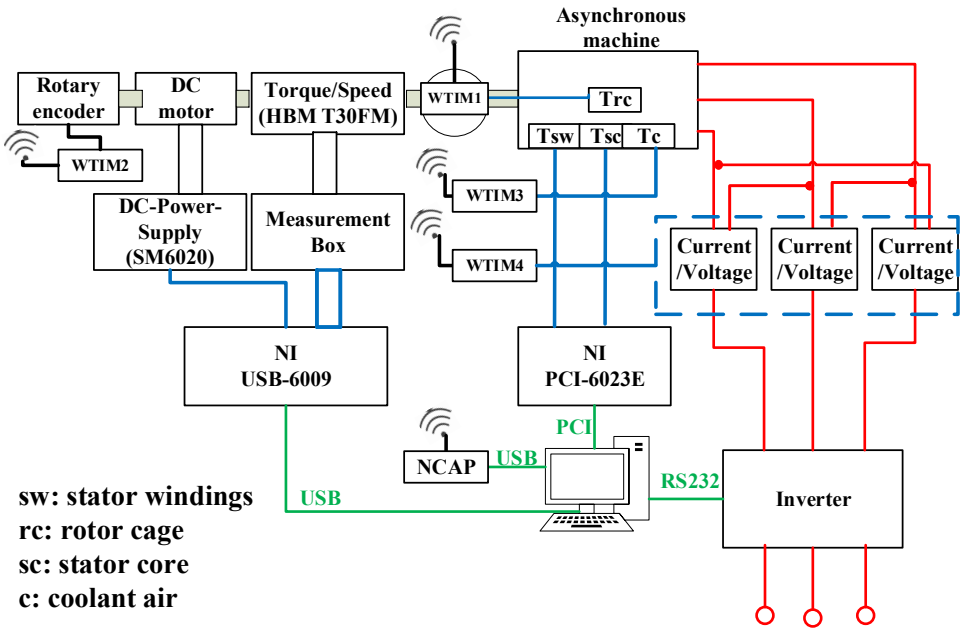


Figure 7.1: The structure of the test bench

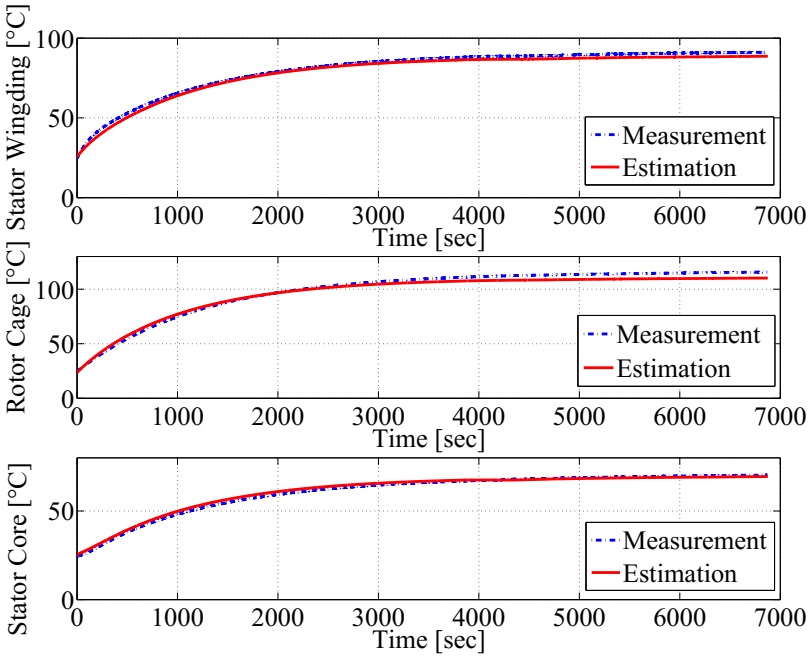


Figure 7.2: Comparison of measured and estimated temperatures under S1 with KF

Table 7.1: The error and NRMSE of the estimated temperatures under S1 with KF

Parameters	Maximum Error	NRMSE
Stator winding	2.3 K	3.2%
Rotor cage	3.5 K	2.08%
Stator core	2.0 K	2.71%

sive heat is discovered to be generated by the maximum deviations of the stator winding and rotor cage at the beginning is about 3.5 K. Shortly after being started, the estimated temperatures follow the measured temperatures quite well with an error under 1 K. The reason for that obvious differences at the beginning stems from the installation of PT1000 on the rotor cage, which influences the flux density and generates excessive losses of about 55 Watt, as compared to a healthy machine [75]. The comparison between measured and estimated temperatures under S6 is shown in figure 7.3. The maximum error and the NRMSE of the estimator are summarized in table 7.2.

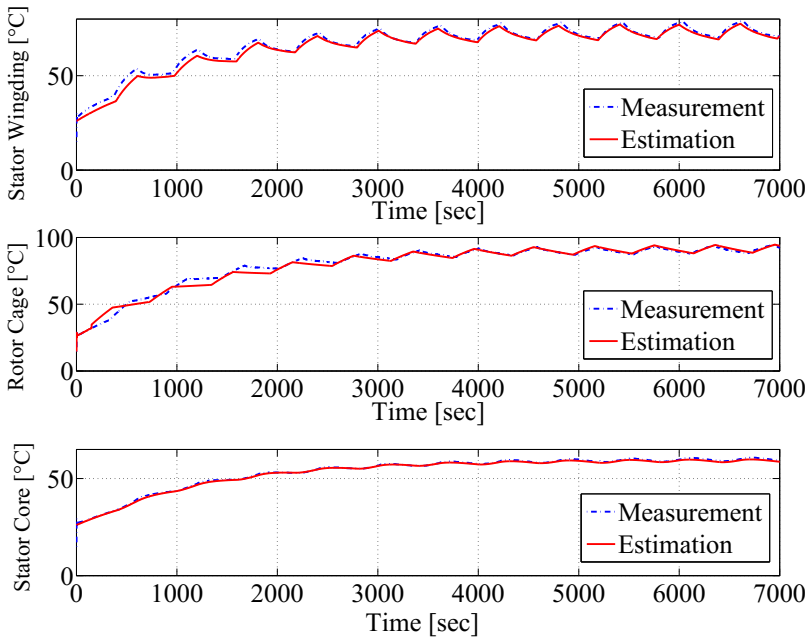


Figure 7.3: Comparison of measured and estimated temperatures under S6 with KF

The temperature differences under both S1 and S6 conditions may result from the aging of the machine. On the other hand, the inaccuracy identification of both machine and thermal parameters would also be the reason.

Table 7.2: The error and NRMSE of the estimated temperatures under S6 with KF

Parameters	Maximum Error	NRMSE
Stator winding	3.5 K	2.69%
Rotor cage	3.5 K	2.45%
Stator core	1.5 K	1.36%

7.2 The Experiments of the EKF Algorithm using WSN

Only two WTIMs are needed for the data acquisition of the EKF temperatures estimation system. WTIM3 and WTIM4 are used to acquire coolant air temperature, the three-phase currents and voltages. As described in section 6.1, the sampling rate for TIM4 is 2000 Hz, and the data will be processed using Park transform. As some blocks are discarded by the dynamic management, the currents and voltages in d-q axis are reconstructed in NCAP. Two experiments are performed in EKF algorithm. The ambient temperature is 26°C. The comparisons of the estimated and measured temperatures under the S1 condition are shown in figure 7.4. The maximal deviation of the stator winding is 2.9 K, the maximal deviation of the rotor cage is 3.8 K, and that of the stator core is 3.7 K. The maximum error and the NRMSE of the EKF estimator are summarized in table 7.3.

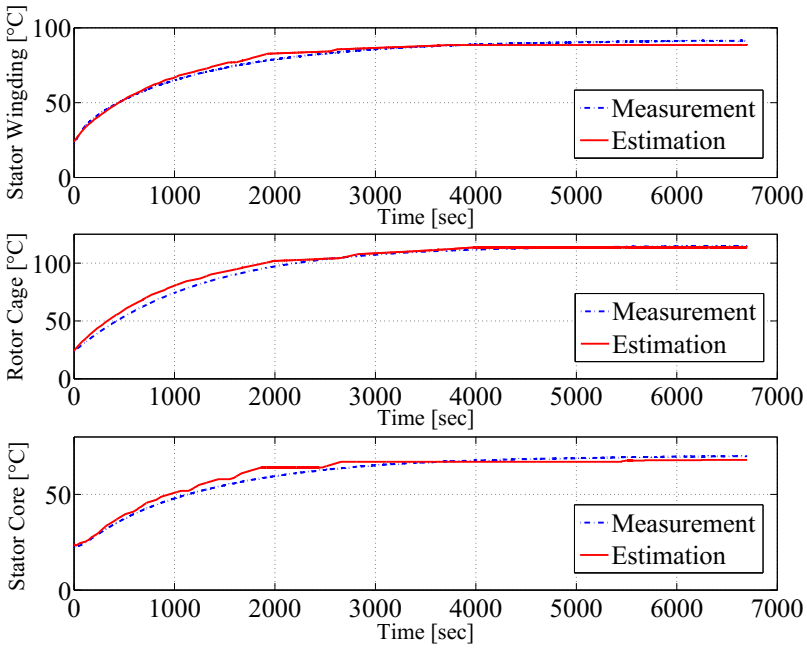


Figure 7.4: Comparison of measured and estimated temperatures under S1 with EKF

The comparisons of the estimated and measured temperatures under the S6 condition

are shown in figure 7.5. The maximal deviation of the stator winding is 2.9 K, the maximal deviation of the rotor cage is 2.3 K, and that of the stator core is 1.8 K. The maximum error and the NRMSE of the EKF estimator are summarized in table 7.4.

Table 7.3: The error and NRMSE of the estimated temperatures under S1 with EKF

Parameters	Maximum Error	NRMSE
Stator winding	2.9 K	2.89%
Rotor cage	3.8 K	3.3%
Stator core	3.7 K	4.86%

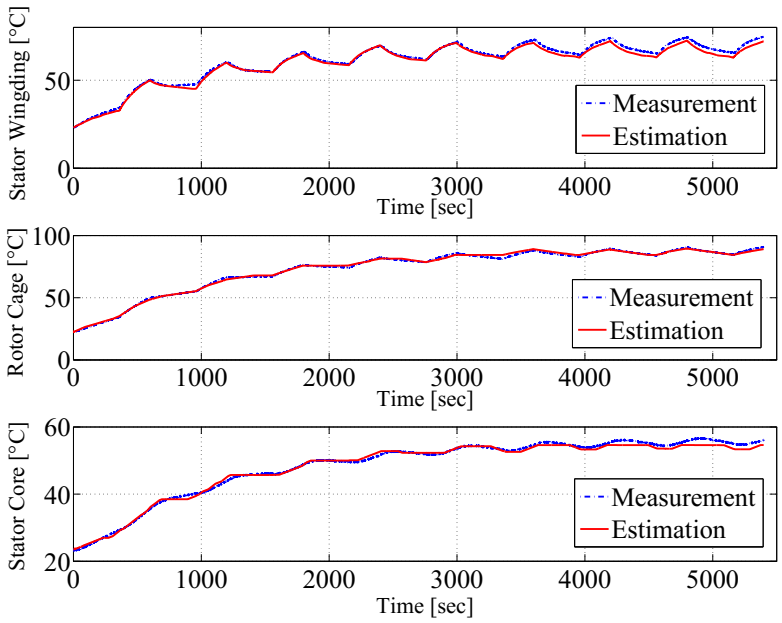


Figure 7.5: Comparison of measured and estimated temperatures under S6 with EKF

Table 7.4: The error and NRMSE of the estimated temperatures under S6 with EKF

Parameters	Maximum Error	NRMSE
Stator winding	2.9 K	2.87%
Rotor cage	2.3 K	1.44%
Stator core	1.8 K	2.33%

The temperature differences under both S1 and S6 conditions are the same reasons as the KF algorithm.

7.3 Conclusions

In this chapter, the experiments of both KF and EKF temperatures estimation system on WSN are performed under two different operating conditions S1 and S6 on the test bench.

For the KF temperatures estimation system, under S1 operating condition, the maximal deviation of the stator winding is 2.3 K, the maximal deviation of the rotor cage is 3.5 K, and that of the stator core is 2 K. Under S6 operating condition, the maximal deviation of the stator winding is 3.5 K, the maximal deviation of the rotor cage is 3.5 K, and that of the stator core is 1.5 K.

For the EKF temperatures estimation system, under S1 operating condition, the maximal deviation of the stator winding is 2.9 K, the maximal deviation of the rotor cage is 3.8 K, and that of the stator core is 3.7 K. Under S6 operating condition, the maximal deviation of the stator winding is 2.9 K, the maximal deviation of the rotor cage is 2.3 K, and that of the stator core is 1.8 K.

Comparing to the accuracy of both KF and EKF system, both of them are successfully implemented in the resource restricted WSN, and perform quite well with the NRMSE values of less than 5%. The KF system needs speed sensor to measure the rotor speed, the EKF doesn't need the speed sensor and the speed can be simultaneously estimated together with the temperatures. As a result, the EKF system has the same performance as KF but requires lower cost.

CHAPTER 8

SUMMARY AND OUTLOOK

The thesis is focused on the development of temperatures estimation algorithm and the implementation of the algorithm on a resource restricted WSN. Model-based software development method is used for the algorithm development and the implementation on a WSN are also detailed described.

To use the model-based method for the algorithm development, a model of the asynchronous machine with power losses is built in Dymola. The experiments are performed for the validation of the electrical, mechanical and the thermal behaviors. Based on the physical model, an efficient and reliable thermal model for tracking the temperatures of the stator winding, the rotor cage and the stator core is also built in Dymola. All the thermal parameters of the asynchronous machine are identified. The conductance values are calculated by the losses and temperatures at the steady state of the machine. The best-fit capacitances are found by using GenOpt which is an optimization tool.

Two different algorithms are developed for the temperatures estimation. 4^{th}-order KF algorithm and 9^{th}-order EKF are first implemented based on the state-space equations in the MATLAB/SIMULINK. The MiL method is used to verify the both algorithms. The physical model in Dymola and the algorithms are connected together in the simulation using SIMULINK. After the verification of the algorithms, both of them are implemented in a WSN, which is based on the IEEE1451 standard using Contiki OS. To estimate the respective temperatures of the stator winding, the rotor cage and the stator core of an asynchronous machine, many approaches are used to integrate the algorithm into the resource restricted embedded system.

For the 4^{th}-order KF, the rotor speed, coolant air temperature, and the effective current and voltage are acquired by WTIM separately and are transmitted to NCAP, where the KF algorithm is implemented. The losses are calculated from the measurement and are processed together with the coolant air temperature by KF algorithm. As the resistance varies with the temperature, the rising resistance is compensated by the estimated stator winding temperature. For the 9^{th}-order EKF, the coolant air temperature, three-phase voltages and currents are acquired and pre-proccesed by two WTIMs, and are transmitted to a NCAP

where the EKF algorithm is implemented.

Finally, under different experiment conditions, both KF and EKF temperatures estimation systems on WSN are tested on the test bench. These two real-time WSN temperatures estimation systems are independent from the control algorithm and functional under any load condition, as long as the current of the stator is a nonzero system with the accuracy of higher than 97%.

The following improvements are expected in future research:

- model-based algorithm development method can be extended to develop the temperatures monitoring system of the permanent magnet synchronous machine (PMSM);
- more temperatures, such as the temperatures for the bearings, can be estimated based on the thermal network;
- based on the implementation of 4^{th}-order KF and the 9^{th}-order EKF, these two algorithms can be compiled as a open source library for further usage;
- the communication between the WTIM and the NCAP should be improved more reliably;
- these two temperatures monitoring systems on WSN can be used in the monitoring of many machines in large areas.

REFERENCES

[1] T. Litman. *Efficient electric motor systems handbook*. Fairmont Press, 1995. (Cited on pages v and 2.)

[2] Network topologies. http://masters.donntu.org/2013/fknt/mironyuk/diss/indexe.htm, 2017. (Cited on pages v and 4.)

[3] IEEE standard for a smart transducer interface for sensors and actuators - common functions, communication protocols, and transducer electronic data sheet (teds) formats. *IEEE Std 1451.0-2007*, pages 1–335, Sept 2007. (Cited on pages v, 6, 7, 90 and 91.)

[4] Asynchronous machines. https://simulationresearch.lbl.gov/modelica/releases/msl/3.2/help/modelica-electrical-machines-basicmachines-asynchronousinduction-machines.html, 2016. (Cited on pages v and 16.)

[5] Asynchronous machine with losses. https://simulationresearch.lbl.gov/modelica/releases/msl/3.2/help/modelica-electrical-machines-examples-asynchronousinduction - machines.html, 2016. (Cited on pages v and 20.)

[6] Yi Huang and Clemens Gühmann. Wireless sensor network for temperature estimations in an asynchronous machine using a kalman filter. *IMEKO TC-10, Technical Diagnostics in Cyber-Physical Era*, 2017. (Cited on pages v, 21 and 22.)

[7] Anton Haumer, Christian Kral, Vladimir Vukovic, Alexander David, Christian Hettfleisch, and Attila Huzsvar. A parametrization scheme for high performance thermal models of electric machines using modelica. *Ifac Proceedings Volumes*, 45(2):1058–1062, 2012. (Cited on pages v, 11, 15, 21, 23, 24, 32, 39, 49, 50, 62, 64 and 73.)

[8] Genopt. https://simulationresearch.lbl.gov/go/overview.html, 2016. (Cited on pages v and 40.)

[9] Introduction of preon32. http://www.virtenio.com/en/products/radio-module.html/, 2015. (Cited on pages vi, 75 and 76.)

[10] Introduction of preon32shuttle. http://www.virtenio.com/en/products/evaluation-module.html/, 2015. (Cited on pages vi and 76.)

[11] Yi Huang and Clemens Gühmann. Temperature estimation of induction machines based on the wireless sensor networks. *AMA Conferences 2017, Nürnberg, Germany*, pages 139 – 144, 2017. (Cited on pages vi, 6, 77, 78, 80, 85, 88 and 95.)

[12] B. Venkataraman, B. Godsey, W. Premerlani, E. Shulman, Thaku M, and R. Midence. Fundamentals of a motor thermal model and its applications in motor protection. In *58th Annual Conference for Protective Relay Engineers, 2005.*, pages 127–144, April 2005. (Cited on page 1.)

[13] J.C. Trigeassou. *Electrical Machines Diagnosis*. ISTE. Wiley, 2013. (Cited on pages 1 and 54.)

[14] M. O. Sonnaillon, G. Bisheimer, C. D. Angelo, and G. O. Garcia. Online sensorless induction motor temperature monitoring. *IEEE Transactions on Energy Conversion*, 25(2):273–280, June 2010. (Cited on page 1.)

[15] R. Beguenane and M. E. H. Benbouzid. Induction motors thermal monitoring by means of rotor resistance identification. *IEEE Transactions on Energy Conversion*, 14(3):566–570, Sep 1999. (Cited on page 1.)

[16] Mehmet Can Vuran Ian F. Akyildiz. *Wireless Sensor Networks*. WILEY, 2010. (Cited on page 1.)

[17] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002. (Cited on page 3.)

[18] Milan Erdelj, Nathalie Mitton, Enrico Natalizio, et al. Applications of industrial wireless sensor networks. *Industrial Wireless Sensor Networks: Applications, Protocols, and Standards*, pages 1–22, 2013. (Cited on page 3.)

[19] Bushra Rashid and Mubashir Husain Rehmani. Applications of wireless sensor networks for urban areas. *J. Netw. Comput. Appl.*, 60(C):192–219, January 2016. (Cited on pages 3 and 5.)

[20] S. Ben Brahim, R. Bouallegue, J. David, T. H. Vuong, and M. David. Design and implementation of wireless sensor network node for rotating electrical machine. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 738–742, June 2016. (Cited on page 5.)

[21] Pravin N. Matte Niteen V. Deshmukh. Design and implementation of embedded remote monitoring system for electric drive. *International Journal of Engineering Research & Technology (IJERT)*, February 2015. (Cited on page 5.)

[22] Andreas Bock, Jürgen Funck, Artur Guidimin, Dian Liu, Richard Burke, and Clemens Gühmann. Wireless sensor for temperature and flux measurements in an axial flux machine. In *17th International Conference on Sensors and Measurement Technology*, March 2015. (Cited on pages 5, 99 and 146.)

[23] A. C. Lima-Filho, R. D. Gomes, M. O. Adissi, T. A. B. da Silva, F. A. Belo, and M. A. Spohn. Embedded system integrated into a wireless sensor network for online dynamic torque and efficiency monitoring in induction motors. *IEEE/ASME Transactions on Mechatronics*, 17(3):404–414, June 2012. (Cited on page 5.)

[24] Jürgen Funck and Clemens Gühmann. A flexible filter for synchronous angular resampling with a wireless sensor network. *Measurement*, 98:393 – 406, 2017. (Cited on pages 5, 76 and 95.)

[25] İlhan Aydın, Mehmet Karaköse, and Erhan Akın. Combined intelligent methods based on wireless sensor networks for condition monitoring and fault diagnosis. *Journal of Intelligent Manufacturing*, 26(4):717–729, 2015. (Cited on page 5.)

[26] R Udayakumar and V Khanaa. Health monitoring system for induction motors. *International Journal of Engineering And Computer Science ISSN*, pages 2319–7242. (Cited on page 5.)

[27] Prateek Saxena and Naresh Tandon. Fault diagnostics and health monitoring of machines using wireless condition monitoring systems. *International Journal of Scientific & Engineering Research*, 6:178–182, 2015. (Cited on page 5.)

[28] IEEE standard for a smart transducer interface for sensors and actuators wireless communication protocols and transducer electronic data sheet (teds) formats. *IEEE Std 1451.5-2007*, pages C1–236, Oct 2007. (Cited on pages 8 and 91.)

[29] John Reedy and Stephen Lunzman. Model based design accelerates the development of mechanical locomotive controls. Technical report, SAE Technical Paper, 2010. (Cited on page 9.)

[30] M. Ahmadian, Z. J. Nazari, N. Nakhaee, and Z. Kostic. Model based design and sdr. In *2005 The 2nd IEE/EURASIP Conference on DSPenabledRadio (Ref. No. 2005/11086)*, pages 8 pp.–, Sept 2005. (Cited on page 9.)

[31] Model based design method. https://en.wikipedia.org/wiki/model-based-design, 2016. (Cited on page 9.)

[32] Hua Huang. *Model-based calibration of automated transmissions*, volume 2. Universitätsverlag der TU Berlin, 2016. (Cited on page 10.)

[33] Clemens Gühmann. Model-based testing of automotive electronic control units. In *3rd International Conference on Materials Testing: Test*, volume 2005, 2005. (Cited on page 10.)

[34] Peter Tavner. *Condition monitoring of rotating electrical machines*, volume 56. IET, 2008. (Cited on page 10.)

[35] Sonia Ben Brahim, Ridha Bouallegue, Jacques David, Tan Hoa Vuong, and Maria David. A wireless on-line temperature monitoring system for rotating electrical machine. *Wireless Personal Communications*, pages 1–21, 2016. (Cited on page 10.)

[36] S. Ben Brahim, R. Bouallegue, J. David, and T. H. Vuong. Modelling and characterization of rotor temperature monitoring system. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 735–740, Sept 2016. (Cited on page 10.)

[37] Andreas Bock. Inbetriebnahme und Plausibilisierung eines Sensors zur Messung der Rotortemperatur eines Asynchronmotors. Studienarbeit, Technische Universität Berlin, 2013. (Cited on pages 10, 34 and 123.)

[38] Rodolfo Ghirlando, Huaying Zhao, Andrea Balbo, Grzegorz Piszczek, Ute Curth, Chad A Brautigam, and Peter Schuck. Measurement of the temperature of the resting rotor in analytical ultracentrifugation. *Analytical biochemistry*, 458:37–39, August 2014. (Cited on page 10.)

[39] K. D. Stephan, J. A. Pearce, Lingyun Wang, and E. Ryza. Prospects for industrial remote temperature sensing using microwave radiometry. In *2004 IEEE MTT-S International Microwave Symposium Digest (IEEE Cat. No.04CH37535)*, volume 2, pages 651–654 Vol.2, June 2004. (Cited on page 10.)

[40] Stewart K. Brown and Len Mannik. Field testing of a fiber optic rotor temperature monitor for power generators. *Proc. SPIE*, 1584:15–22, 1991. (Cited on page 10.)

[41] C. Hudon, C. Guddemi, S. Gingras, R. C. Leite, and L. Mydlarski. Rotor temperature monitoring using fiber bragg gratings. In *2016 IEEE Electrical Insulation Conference (EIC)*, pages 456–459, June 2016. (Cited on page 10.)

[42] L. Wang, X. Yang, and B. Sheng. Distributed optical fiber sensor for virtual monitoring of turbine rotor's temperature. In *2009 International Conference on Measuring Technology and Mechatronics Automation*, volume 1, pages 16–19, April 2009. (Cited on page 10.)

[43] M. Ganchev, B. Kubicek, and H. Kappeler. Rotor temperature monitoring system. In *Electrical Machines (ICEM), 2010 XIX International Conference on*, pages 1–5, Sept 2010. (Cited on page 10.)

[44] M. Sabaghi, H. Feshki Farahani, H. R. Hafezi, P. Kiani, and A. R. Jalilian. Stator winding resistance estimation for temperature monitoring of induction motor under unbalance supplying by dc injection method. In *Universities Power Engineering Conference, 2007. UPEC 2007. 42nd International*, pages 217–222, Sept 2007. (Cited on page 11.)

[45] S. G. Ahn, B. G. Park, R. Y. Kim, and D. S. Hyun. Fault diagnosis for open-phase faults of permanent magnet synchronous motor drives using extended kalman filter. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pages 835–840, Nov 2010. (Cited on page 11.)

[46] R. Beguenane and M. El Hachemi Benbouzid. Induction motors thermal monitoring by means of rotor resistance identification. In *1997 IEEE International Electric Machines and Drives Conference Record*, pages TD2/4.1–TD2/4.3, May 1997. (Cited on page 11.)

[47] J. L. Zamora and A. Garcia-Cerrada. Online estimation of the stator parameters in an induction motor using only voltage and current measurements. *IEEE Transactions on Industry Applications*, 36(3):805–816, May 2000. (Cited on page 11.)

[48] Z. Gao, T. G. Habetler, and R. G. Harley. An online adaptive stator winding temperature estimator based on a hybrid thermal model for induction machines. In *IEEE International Conference on Electric Machines and Drives, 2005.*, pages 754–761, May 2005. (Cited on page 11.)

[49] Z. Gao, T. G. Habetler, R. G. Harley, and R. S. Colby. A novel online rotor temperature estimator for induction machines based on a cascading motor parameter estimation scheme. In *Diagnostics for Electric Machines, Power Electronics and Drives, 2005. SDEMPED 2005. 5th IEEE International Symposium on*, pages 1–6, Sept 2005. (Cited on page 11.)

[50] Ozsoy E E, Gokasan Metin, and Bogosyan Seta. Simultaneous rotor and stator resistance estimation of squirrel cage induction machine with a single extended kalman filter. *Turk. J. Elec. Eng. & Comp. Sic.*, 2010. (Cited on page 11.)

[51] C. Kral, T. G. Habetler, R. G. Harley, F. Pirker, G. Pascoli, H. Oberguggenberger, and C. J. M. Fenz. Rotor temperature estimation of squirrel-cage induction motors by means of a combined scheme of parameter estimation and a thermal equivalent

model. *IEEE Transactions on Industry Applications*, 40(4):1049–1057, July 2004. (Cited on page 11.)

[52] S. Bogosyan, M. Barut, and M. Gokasan. Braided extended kalman filters for sensorless estimation in induction motors at high-low/zero speed. *IET Control Theory Applications*, 1(4):987–998, July 2007. (Cited on page 11.)

[53] H. Tajima and Y. Hori. Speed sensorless field-orientation control of the induction machine. *IEEE Transactions on Industry Applications*, 29(1):175–180, Jan 1993. (Cited on page 11.)

[54] In-Joong Ha and Sang-Hoon Lee. An online identification method for both stator-and rotor resistances of induction motors without rotational transducers. *IEEE Transactions on Industrial Electronics*, 47(4):842–853, Aug 2000. (Cited on page 11.)

[55] C. J. Chiang, Y. K. Wang, and W. T. Cheng. Ekf-based rotor and stator resistance estimation in speed sensorless control of induction motors. In *American Control Conference (ACC), 2012*, pages 1174–1179, June 2012. (Cited on page 11.)

[56] A. Boglietti, A. Cavagnino, D. Staton, M. Shanel, M. Mueller, and C. Mejuto. Evolution and modern approaches for thermal analysis of electrical machines. *IEEE Transactions on Industrial Electronics*, 56(3):871–882, March 2009. (Cited on pages 11 and 20.)

[57] Motor-CAD. https://www.motor-design.com/, 2016. Accessed April 4, 2016. (Cited on page 11.)

[58] Yi Du, T. G. Habetler, and R. Gordon Harley. Methods for thermal protection of medium voltage induction motors - a review. In *2008 International Conference on Condition Monitoring and Diagnosis*, pages 229–233, April 2008. (Cited on page 11.)

[59] Anton Haumer, Christian Kral, Hansjörg Kapeller, Thomas Bäuml, and Johannes V Gragger. The AdvancedMachines Library: Loss Models for Electric Machines. In *The 7 International Modelica Conference, Como, Italy*, pages 847–854. Linköping University Electronic Press, October 2009. (Cited on pages 15, 16, 17, 18, 19, 43, 62 and 64.)

[60] Modelica Association. Tutorial: Modelica - a unified object-oriented language for physical systems modeling. Technical report, 2000. (Cited on page 15.)

[61] Dehuai Zeng. *Advances in Computer Science and Engineering*. Springer Publishing Company, Incorporated, 2012. (Cited on pages 15, 16, 21, 50 and 62.)

[62] Dynasim AB. *Dymola Dynamic Modeling Laboratory User's Manual*. (Cited on page 15.)

[63] Anton Haumer and Christian Kral. Modelica libraries for dc machines, three phase and polyphase machines. *4th International Modelica Conference, March 7-8, 2005*, pages 549–588. (Cited on page 16.)

[64] Electric Machinery Committee of the IEEE Power Engineering Society. IEEE Std 112-2004, IEEE Standard Test Procedure for Polyphase Induction Motors and Generators. pages 1–87, February 2016. (Cited on pages 17, 19, 28, 29 and 30.)

[65] Core losses. http://www.electrical4u.com/losses-and-efficiency-of-induction-motor/, 2016. (Cited on page 18.)

[66] Hansjorg Kofler. Stray load losses in induction machines a review of experimental measuring methods and a critical performance evaluation. *Renewable Energy and Power Quality Journal*, 1:318–323, 04 2003. (Cited on page 18.)

[67] Rotating electrical machines - part 2-1: Standard methods for determining losses and efficiency from tests (excluding machines for traction vehicles), 2007. (Cited on page 19.)

[68] M. Plainer C. Kral, A. Haumer. Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the simpleflow-library. pages 213–218, March 2005. (Cited on page 19.)

[69] K&K Associates. *Thermal Network Modeling Handbook*. 1999-2000. (Cited on page 21.)

[70] Heat capacitor. https://www.maplesoft.com/documentation-center/online-manuals/ modelica/modelica-thermal-heattransfer-components.html/modelica.thermal.heattransfer.components.heatcapacitor., 2016. (Cited on page 23.)

[71] Thermal conductor. https://www.maplesoft.com/documentation-center/online-manuals/modelica/modelica-thermal-heattransfer-components.html/modelica.thermal.heattransfer.components.thermalconductor., 2016. (Cited on page 23.)

[72] Florian Doering. Parameteridentifikation einer Asynchronmaschine am Prüfstand. Diplomarbeit, Technische Universität Berlin, 2009. (Cited on pages 33, 38 and 39.)

[73] Michael Wetter. GenOpt manual. pages 1–109, August 2016. (Cited on page 41.)

[74] Madhava Rao Ashwin Murali, Arushi Gupta. Towards a Novel Direct Online Speed-Torque Curve Plotter for Three Phase Induction Motor. pages 1–6, June 2014. (Cited on page 43.)

[75] J. F. Bangura and N. A. Demerdash. Effects of broken bars/end-ring connectors and airgap eccentricities on ohmic and core losses of induction motors in asds using a coupled finite element-state space method. *IEEE Transactions on Energy Conversion*, 15(1):40–47, Mar 2000. (Cited on pages 44 and 125.)

[76] IEEE Power Engineering Society. Ieee standard test procedure for polyphase induction motors and generators. 2004. (Cited on page 51.)

[77] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995. (Cited on pages 52, 54 and 61.)

[78] Paul C. Krause, Oleg Wasynczuk, and Scott D. Sudhoff. *Analysis of Electric Machinery and Drive Systems*, pages 605–613. Wiley-IEEE Press, 2002. (Cited on page 55.)

[79] J. K. Al-Tayie and P. P. Acarnley. Estimation of speed, stator temperature and rotor temperature in cage induction motor drive using the extended kalman filter algorithm. *IEE Proceedings - Electric Power Applications*, 144(5):301–309, Sep 1997. (Cited on page 56.)

[80] D. Staton, A. Boglietti, and A. Cavagnino. Solving the more difficult aspects of electric motor thermal analysis in small and medium size industrial induction motors. *IEEE Transactions on Energy Conversion*, 20(3):620–628, Sept 2005. (Cited on page 57.)

[81] A. Haumer C. Kral. Modelica libraries for dc machines, three phase and polyphase machines. *4th International Modelica Conference*, pages 549–558, March 2005. (Cited on page 62.)

[82] Garron Fish. Dymola-simulink interface. http://www.claytex.com/blog/dymola-simulink-interface/, 2011. (Cited on page 63.)

[83] The contiki os. http://www.contiki-os.org/, 2014. (Cited on page 76.)

[84] Jürgen Funck and Sebastian Nowoisky. MDT Strom- und Spannungswandler-Modul. Aug 2011. (Cited on page 79.)

[85] Torben Hopp. Intelligenter Sensor zur Leistungsmessung im Dreiphasennetz. Master's thesis, Technische Universität Berlin, 2013. (Cited on pages 79 and 123.)

[86] Memory allocation. https://github.com/contiki-os/contiki/wiki/memory-allocation/, 2016. (Cited on pages 83 and 96.)

[87] Contiki 2.6 linked list library. http://contiki.sourceforge.net/, 2015. (Cited on page 83.)

[88] Digital sensor. https://en.wikipedia.org/wiki/digital-sensors/, 2016. (Cited on page 87.)

[89] Zach Shelby and Carsten Bormann. *6LoWPAN: The Wireless Embedded Internet*. Wiley Publishing, 2010. (Cited on page 91.)

[90] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 29–42, New York, NY, USA, 2006. ACM. (Cited on page 92.)

[91] Advanced RISC Machines Ltd (ARM). *Fixed Point Arithmetic on the ARM*. Application Note 33, September 1996. (Cited on pages 96 and 98.)

[92] Enrico Bocchieri. *Fixed-Point Arithmetic*, pages 255–275. Springer London, London, 2008. (Cited on page 97.)

[93] CMSIS - Cortex Mircrocontroller Software Interface Standard. http://www.arm.com/products/processors/cortex-m/cortexmicrocontroller-software-interface-standard.php/, 2015. (Cited on page 98.)

[94] Stephan Rein. Fixed-point arithmetic in c: A tutorial and an example on wavelet filtering. Technical report, Wavelet Application Group, Technische Universitaet Berlin, July 1 2008. (Cited on page 146.)

CHAPTER A

PARAMETERS IDENTIFICATION OF THE MODEL

A.1 Building Interface between Genopt and Dymola

Listing A.1: initialization.txt

```
Simulation {
  Files {
    Template {
      File1 = ScheduleTemplate.txt;
      File2 = dsin.txt;
    }
    Input {
      File1 = Schedule.txt;
      File2 = dsin.txt;
    }
    Log {
      File1 = dslog.txt;
    }
    Output {
      File1 = result.txt;
    }
    Configuration {
      File1 = DymolaWinXP.cfg;
    }
  }
  ObjectiveFunctionLocation{
        Delimiter1 = "f(x) =" ;
        Name1      = "f(x)";
  }
}
```

A.2 Building Interface between Genopt and Dymola

Listing A.2: configureation.txt

```
Optimization {
  Files {
    Command {
      File1 = command.txt;
    }
  }
}
```

Listing A.3: command.txt

```
* GenOpt command file
Vary{
  Parameter{ Name = CrotorCage; Min = 100; Ini = 1500;
        Max = 10000; Step = 1; }
  Parameter{ Name = CstatorCore; Min = 100; Ini =  10168;
        Max = 20000; Step = 1; }
  Parameter{ Name = CstatorWinding; Min = 100; Ini = 1000;
        Max = 10000; Step = 1; }
}

OptimizationSettings{
  MaxIte = 10000;
  MaxEqualResults = 100;
  WriteStepNumber = false;
}

Algorithm{
 Main = GPSHookeJeeves;
 MeshSizeDivider = 2;
 InitialMeshSizeExponent = 0;
 MeshSizeExponentIncrement = 1;
 NumberOfStepReduction = 4;
}
```

Listing A.4: ScheduleTemplate.txt

```
double tab1(3,1) # comment line
$%CrotorCage%$
$%CstatorCore%$
$%CstatorWinding%$
```

CHAPTER B

THE IMPLEMENTATION OF KF IN THE WSN

B.1 FIR Filter Function

Listing B.1: Declaration of Q15 FIR filter function

```
  /**
   * @brief Processing function for the Q15 FIR filter.
   * @param[in] *S points to an instance of the Q15 FIR.
   * @param[in] *pSrc points to the block of input data.
   * @param[out] *pDst points to the block of output data.
   * @param[in] blockSize number of samples to process.
   * @return none.
   */
  void arm_fir_q15(
  const arm_fir_instance_q15 * S,
  q15_t * pSrc,
  q15_t * pDst,
  uint32_t blockSize);
```

Listing B.2: Coefficient of FIR low-pass filter

```
#define FILTER_TAPS 28
static const int16 filterCoeff [FILTER_TAPS] = {28,58,-3,\
 -187,-275,57,696,823,-312,-2055,-2187,1135,6978,11629,\
 11629,6978,1135,-2187,-2055,-312,823,696,57,-275,-187,\
 -3,58,28};
```

B.2 Structure Instance of the Sensor

Listing B.3: Declaration of the structure instance

```
typedef struct {
        xdcr_driver xdcrChannel;        /* driver of transducer channel */
        analog_sensor_conf_t conf;      /* sensor specific configuration */
        xdcrCalib_linCoeff_t* calib; /* calibration coefficients */
        xdcr_AnaDataSet* dataSet;               /* pointer to data the set */
```

```
        UInt32 dataSetSize;                    /* size of data set in bytes */
        UInt32 messageCounter;          /* counts the number of messages */
} analog_sensor_instance_t;
```

Listing B.4: The structure rotation data set

```
/*structure of rotation data set  */
struct rotationDataSet {
  xdcr_dataSet_type type;       /* type code of data set */
        UInt32 sample_period; /* sampling period in delta angle of pulses */
        UInt32 anglestamp;      /* angle of first encoder pulse */
        UInt16 repCount;        /* number of acquisition steps in dataset */
        UInt32 t0;      /* timestamp of first pulse as 32 Bit integer */
        int16  deltaT[1024];    /* timestamps as 32 Bit integers */
};
```

Listing B.5: The structure of Contiki process

```
PROCESS_THREAD(name, ev, data)
{
  PROCESS_BEGIN();
  /Application code/
  PROCESS_EDN();
}
```

B.3 KF Structure

Listing B.6: The structure of KF data

```
struct kf_data {
  q31_t dt; //sampling time
  int nb_samples; //number of samples per block
  q31_t Psw; //stator winding losses
  q31_t Prc; //rotor cage losses
  q31_t Psc; //stator core losses
  q31_t Tc; //coolant air temperature
};
```

Listing B.7: The structure of Kalman filter

```
typedef struct kf_filter {
  unsigned state_dim;
  unsigned measure_dim;
  /* state                          */
  arm_matrix_instance_q31 *X;//(4,1)
  /* control                          */
  arm_matrix_instance_q31 *U;//(4,1)
  /* state covariance matrix         */
  arm_matrix_instance_q31 *P;//(4,4)
  /* process covariance noise        */
  arm_matrix_instance_q31 *Q;//(4,4)
```

```
  /* measurement covariance noise    */
  arm_matrix_instance_q31 *R;//(1,1)
  /* A matrix                        */
  arm_matrix_instance_q31 *A;//(4,4)
  /* B matrix                       */
  arm_matrix_instance_q31 *B;//(4,4)
  /* jacobian of the measure wrt X   */
  arm_matrix_instance_q31 *H;//(1,4)
  /* error matrix                    */
  arm_matrix_instance_q31 *E;//(1,1)
  /* kalman gain                     */
  arm_matrix_instance_q31 *K;//(4,1)
  q31_t err[1];

  filter_function ffun;
  measure_function mfun;

  /* temps */
  arm_matrix_instance_q31 *Xdot;//(4,1)
  arm_matrix_instance_q31 *Pdot;//(4,4)

  arm_matrix_instance_q31 *tmp1;//(4,4)
  arm_matrix_instance_q31 *tmp2;//(4,4)
  arm_matrix_instance_q31 *tmp3;//(4,4)
  arm_matrix_instance_q31 *tmp4;//(1,4)
  arm_matrix_instance_q31 *tmp5;//(4,1)
  arm_matrix_instance_q31 *tmp6;//(4,1)
  arm_matrix_instance_q31 *tmp7;//(1,1)

} kf_filter;
```

B.4 Fixed-Point Arithmetic

A. Number Representation

A number is represented as a binary word in a microprocessor. A binary word with $N = 8$ bits is taken as an example $b_7 b_6 b_5 b_4 \cdots b_0$. If this word represents an unsigned integer, the numerical value is:

$$B = 2^7 b_7 + 2^6 b_6 + 2^5 b_5 + 2^4 b_4 + \cdots + 2^0 b_0 \tag{B.1}$$

The Most Significant Bit (MSB) is b_7 which is located left. If the 8 bits binary word is 01100011, the unsigned integer value is 99. If it is interpreted as a fixed-point number 011000.11, the value of this word can be calculated as $2^4 + 2^3 + 2^{-1} + 2^{-2} = 24.750$, or generally as:

$$B = \frac{1}{2^{exp(b)}} \sum_{n=0}^{N-1} 2^n b_n \tag{B.2}$$

where $exp(b)$ ($exp(b) = 2$) denotes the number of positions which follow the point. The number range for an unsigned fixed-point number B is given as:

$$[0, \frac{2^N - 1}{2^{exp(b)}}] \tag{B.3}$$

The numerical value A of a fixed-point number a is thus given by the following equation:

$$A = a \cdot 2^{-exp(a)} \tag{B.4}$$

a denotes the integer value of the internal binary word (that excludes the MSB bit) and $exp(a)$ denotes the exponent of a.

The MSB of the binary word is represented for the sign. If the MSB is zero, the number is positive. If the MSB is set, the number is negative. The two's complement is used for signed numbers which represents positive numbers in the same way as in the ordinary unsigned notation with the exception that the MSB must be zero. A negative number is built by inverting all bits and adding one to the result, e.g., $-2 = inv(0010) + 1 = 1101 + 1 = 1110$. The interpreted value A of such a number is given as [94]:

$$A = 2^{-exp(a)}[-2^{N-1} + \sum_{n=0}^{N-2} 2^n b_n] \tag{B.5}$$

In order to transfer the existing KF algorithm in floating point to fixed point representation, proper Q-format(Qm.n) defined in [22] is first to be considered. The Q-format is used to denote the fixed-point number format which assumes the notation of the two's complement. Therefore, an N-binary word always has N-1 bits for the absolute numerical value. A Q(m,n) format denotes that m bits are used to designate the two's complement integer portion of the number, not including the MSB bit, and that n bits are the number of bits to the right of the radix point. Thus, a Q(m,n) number always requires $N = m + n + 1$ bits. Its range is given by $[-2^m, 2^m - 2^{-n}]$ and the resolution is given by 2^{-n}. The value m in Q(m,n) is optional. If the value m in Q(m,n) is left, it is assumed to be zero.

The fractional arithmetic uses a specific case of the Q-format where the number only has fractional portions (Q(m,n) with m=0). For example, the Q15 data range is $[-1, 0.999969482]$. The advantage of this format is that the product of two numbers between $[-1, 1]$ stays in the range of $[-1, 1]$, thus there will be no overflow problem.

B. Definition of Arithmetic

The implementation of fixed point arithmetic using C macros in the Contiki is based on the tutorial [94]. The shift operator, e.g. left shift " << " and right shift " >> " are used for the virtual shift in order to change the exponent of the word. It is better to use the

logical shift than the arithmetic shift, because the computation speed is much faster than the arithmetic shift. The left-shift and the right-shift can be used to change the exponent of numbers, because some operations need the same exponent. An appropriate C macro for changing the exponent can be as in Listing B.8:

Listing B.8: Definition of changing the exponent

```
#define ChangeExp(a,expA,expB)
(((expB)>(expA)) ? (a)<<((expB)-(expA)) : (a)>>((expA)-(expB)))
```

It is required in the arithmetic that floating-point numbers and fixed-point numbers should be converted to or from each other. Equation (B.6) shows a floating-point number $Bfloat$ is converted to a fixed-point number B with the exponent $exp(b)$.This method would reduce the precision because the int cuts the fractional part. Equation (B.7) shows the converting of a fixed-point number B with exponent $exp(b)$ to a floating point number $Bfloat$.

$$B = int(Bfloat \cdot 2^{exp(b)}) \cdot 2^{-exp(b)} \tag{B.6}$$

$$Bfloat = B * 2^{-exp(b)} \tag{B.7}$$

The appropriate macros are defined as in the Listing reflst:FloatToFix:

Listing B.9: Definition of *FloatToFix* and *FixToFloat* Macros

```
#define FloatToFix(Bfloat,expB) ((int)( (Bfloat)*(float)\
        (1<<(expB) ) ))
#define FixToFloat(b,expB) ( (float) (b) / (float)\
        (1<<(expB)) )
```

For addition and subtraction the two operands a and b have to be converted to the exponent of the result number c. It then can be calculated by adding the binary words:

$$c = a \cdot 2^{-exp(c)} + b \cdot 2^{-exp(c)} \tag{B.8}$$

If the exponents of a and b do equal, the macro to return a fixed-point number with the same exponent is given as in the Listing B.10:

Listing B.10: Definition of *AddFix* and *SubFix* Macros

```
#define AddFix(a,b) ((a)+(b))
#define SubFix(a,b) ((a)-(b))
```

Otherwise, the operands first have to be converted. In case of an overflow, the sum of two binary numbers require one more integer bit in the result. If the operands are in the form Qa.b, the result will be in the form Q(a+1).b which is defined as the Listing B.11.

Listing B.11: Definition of *AddFixExp* and *SubFixExp* Macros

```
#define AddFixExp(a,b,expA,expB,expC)
        (ChangeExp(a,expA,expC)+ChangeExp(b,expB,expC))
#define SubFixExp(a,b,expA,expB,expC)
        (ChangeExp(a,expA,expC)-ChangeExp(b,expB,expC))
```

The product C of two binary words a and b can be computed with an integer multiplication as:

$$A \cdot B = a \cdot 2^{-exp(c)} \cdot b \cdot 2^{-exp(c)} \tag{B.9}$$

The exponent of the result is given as the sum of the input exponents. A macro to compute the product with exponent $exp(c)$ of two operands with exponent $exp(c)$ is given as in the Listing B.12:

Listing B.12: Definition of *MulFix* Marco

```
#define MulFix(a,b,expC) (((a) * (b))>>(expC))
```

A macro to compute the product with exponent $exp(c)$ of two operands with exponent $exp(a)$ and $exp(b)$ is defined as in the Listing B.13:

Listing B.13: Definition of *MulFixExp* Marco

```
#define MulFixExp(a,b,expA,expB,expC)
        (ChangeExp((a) * (b),(expA)+(expB),expC))
```

The division $C = A/B$ can be performed with an integer division as:

$$C = \frac{a \cdot 2^{exp(b)-exp(a)+exp(c)}}{b} \cdot 2^{-exp(c)} \tag{B.10}$$

A macro for division of two numbers with exponent expC returns a number with exponent $exp(c)$ which is defined in the Listing B.14:

Listing B.14: Definition of *DivFix* Marco

```
#define DivFix(a,b,expC) (((a)<<(expC)/(b))
```

CHAPTER C

THE IMPLEMENTATION OF EKF IN THE WSN

C.1 EKF Structure

Listing C.1: Declaration of EKF structure in C

```
typedef struct
{
    /* dimensions */
    uint8_t state_dim;
    uint8_t measure_dim;

    /* matrices */
    MatFix32 X;     // state, 9*1
    MatFix32 U;     // control, 9*1
    MatFix32 P;     // state covariance matrix, 9*9
    MatFix32 Q;     // process covariance noise, 9*9
    MatFix32 R;     // measurement covariance noise, 2*2
    MatFix32 A;     // A matrix, 9*9
    MatFix32 B;     // B matrix, 9*9
    MatFix32 F;     // jacobian of Xdot wrt X, 9*9
    MatFix32 H;     // jacobian of the measure wrt X, 2*9
    MatFix32 E;     // error matrix, 2*2
    MatFix32 K;     // kalman gain, 9*2
    MatFix32 Xdot; // temp A matrix, 9*1
    MatFix32 Pdot; // temp P matrix, 9*9

    /* function pointers */
    filter_function ffun;
    measure_function mfun;
} ekf_info;
```

C.2 EKF Function Definition

Listing C.2: Definition of fuction *run_ekf*

```
int run_ekf(ekfDataSet *inputDataSet)
{
        coeff_cal(inputDataSet->sample_period);

        /* initialize matrix B */
    MatFix32 B,Q,R;
fix32t b[81]={bb11,0,0,0,0,0,0,0,0,
               0,bb22,0,0,0,0,0,0,0,
               bb31,0,0,0,0,0,0,0,0,
               0,bb42,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,bb88,0,
               0,0,0,0,0,0,0,0,bb99};
mat_init(&B,9,9,b);

        /* initialize matrix Q */
fix32t q[81]=
{FloatToFix(0.1/ScaleFactor),0,0,0,0,0,0,0,0,
0,FloatToFix(0.1/ScaleFactor),0,0,0,0,0,0,0,
0,0,FloatToFix(0.8/ScaleFactor),0,0,0,0,0,0,
0,0,0,FloatToFix(0.8/ScaleFactor),0,0,0,0,0,
0,0,0,0,FloatToFix(0.01/ScaleFactor),0,0,0,0,
0,0,0,0,0,FloatToFix(0.01/ScaleFactor),0,0,0,
0,0,0,0,0,0,FloatToFix(0.000001/ScaleFactor),0,0,
0,0,0,0,0,0,0,FloatToFix(0.000001/ScaleFactor),0,
0,0,0,0,0,0,0,0,0,FloatToFix(0.000001/ScaleFactor)};
mat_init(&Q,9,9,q);

/* initialize matrix R */
fix32t r[4]={FloatToFix(0.01/ScaleFactor),0,
0,FloatToFix(0.01/ScaleFactor)};

mat_init(&R,2,2,r);
/* measure */
fix32t y[2]={0,0};
/* command */
fix32t u[9]={0,0,0,0,0,0,0,0,0};

/* initialize ekf_info */
ekf_info *filter;

filter = ekf_new(9,2,&B,&Q,&R,linear_filter,linear_measure);
if (filter == NULL)
        return MEMORY;
/* filter run */
for (iter=0; iter<inputDataSet->repCount; iter++)
{
        ekf_read_data(inputDataSet,y,u);
```

```
        ekf_reset(filter,X0,X0_1,P0,iter,\
         inputDataSet->unprocessedBlockNum,\
         inputDataSet->sample_period);
        ekf_predict(filter,u);
        ekf_update(filter,y);
        ekf_get_state(filter,X0,P0);
        }
        ekf_free(filter);

        return NO_ERROR;
}
```

C.3 EKF Matrix Definition

Listing C.3: Basic definition of conversion macros

```
typedef int             fix32t;
#define FixNum_1        134217728   // 2^27
#define FloatToFix(x)   ((fix32t)(((x)>=0)?((x)*FixNum_1+0.5):((x)*FixNum_1-0.5)))
#define FixToFloat(x)   ((float)((float)(x)/FixNum_1))
```

Listing C.4: The macros of multiplication

```
#define DecimalPlacesNum    27
#define GetTopDecimalBit    0x04000000
fix32t MulFix(fix32t Src1, fix32t Src2)
{
        fix64t product = (fix64t)Src1 * Src2;

        if (product < 0)
                product--; // in order to round -1/2 correctly

        fix32t result = product >> DecimalPlacesNum;
        result += (product & GetTopDecimalBit) >> (DecimalPlacesNum-1);

        return result;
}
```

Listing C.5: The definition of a matrix

```
#define FIXMATRIX_MAX_SIZE  9
typedef struct {
    uint8_t rows;
    uint8_t columns;
    fix32t  data[FIXMATRIX_MAX_SIZE][FIXMATRIX_MAX_SIZE];
} MatFix32;
```

Listing C.6: Definition of fuction *matrix_ekf*

```
#include <stdio.h>
#include "matrix_ekf.h"
```

```
fix32t MulFix(fix32t Src1, fix32t Src2)
{
        fix64t product = (fix64t)Src1 * Src2;

        if (product < 0)
                product--; // This adjustment is required in order to round -1/2
                    correctly

        fix32t result = product >> DecimalPlacesNum;
        result += (product & GetTopDecimalBit) >> (DecimalPlacesNum-1);

        return result;
}

void mat_init(MatFix32 *Matrix, uint8_t nRows, uint8_t nColumns, fix32t *DataArray
    )
{
        int row, col;
        Matrix->rows = nRows;
        Matrix->columns = nColumns;
        for (row = 0; row < nRows; row++)
        {
          for (col = 0; col < nColumns; col++)
          {
                  Matrix->data[row][col] = *(DataArray + row * nColumns + col);
          }
        }
}

void mat_add(MatFix32 *Src1, MatFix32 *Src2, MatFix32 *Dst)
{
        int row, col;
        Dst->rows = Src1->rows;
        Dst->columns = Src1->columns;
    for (row = 0; row < Dst->rows; row++)
    {
      for (col = 0; col < Dst->columns; col++)
      {
          Dst->data[row][col] = Src1->data[row][col] + Src2->data[row][col];
      }
    }
}

void mat_sub(MatFix32 *Src1, MatFix32 *Src2, MatFix32 *Dst)
{
        int row, col;
        Dst->rows = Src1->rows;
        Dst->columns = Src1->columns;
    for (row = 0; row < Dst->rows; row++)
    {
      for (col = 0; col < Dst->columns; col++)
      {
          Dst->data[row][col] = Src1->data[row][col] - Src2->data[row][col];
```

```
      }
    }
}

void mat_transpose(MatFix32 *Src, MatFix32 *Dst)
{
        int row, col;
        Dst->rows = Src->columns;
        Dst->columns = Src->rows;
        for (row = 0; row<Dst->rows; row++)
        {
      for (col = 0; col<Dst->columns; col++)
      {
          Dst->data[row][col] = Src->data[col][row];
      }
    }
}

void mat_mult(MatFix32 *Src1, MatFix32 *Src2, MatFix32 *Dst)
{
        int row, col, k;
        Dst->rows = Src1->rows;
        Dst->columns = Src2->columns;
        for (row = 0; row < Dst->rows; row++)
        {
                for (col = 0; col < Dst->columns; col++)
                {
                        Dst->data[row][col] =0;
                        for (k=0; k < Src1->columns; k++)
                        {
                                Dst->data[row][col] += MulFix(Src1->data[row][k],
                                    Src2->data[k][col]);
                        }
                }
        }
}

void mat_scal_mult(MatFix32 *Src, int Scale, MatFix32 *Dst)
{
        int row, col;
        Dst->rows = Src->rows;
        Dst->columns = Src->columns;
        for (row = 0; row < Dst->rows; row++)
        {
                for (col = 0; col < Dst->columns; col++)
                {
                        Dst->data[row][col] = Src->data[row][col] * Scale;
                }
        }
}


void mat_inv(MatFix32 *Src, MatFix32 *Dst, int *Scale)
{
```

```
        fix64t  det_a,temp1,temp2;
        double temp[Src->rows][Src->columns];

        Dst->rows = Src->rows;
        Dst->columns = Src->columns;
        *Scale = 1;
        if ( Src->rows == 1 && Src->columns == 1 )
        {
           temp[0][0] = (double) FixNum_1 / Src->data[0][0];
           while ( temp[0][0] >= MaxFloatNum || temp[0][0] < MinFloatNum)
           {
                   temp[0][0] = temp[0][0] / 10.;
                   *Scale = *Scale * 10;
           }
           Dst->data[0][0] = FloatToFix(temp[0][0]);
        }
        else if( Src->rows == 2 && Src->columns == 2 )
        {
                temp1 = MulFix(Src->data[0][0],Src->data[1][1]);
                temp2 = MulFix(Src->data[0][1],Src->data[1][0]);
                det_a = temp1 - temp2;
                temp[0][0] = (double) MulFix(Src->data[1][1],FixNum_1) / det_a;
                temp[0][1] = (double) MulFix(Src->data[0][1],FixNum_1) / (-det_a);
                temp[1][0] = (double) MulFix(Src->data[1][0],FixNum_1) / (-det_a);
                temp[1][1] = (double) MulFix(Src->data[0][0],FixNum_1) / det_a;
                while ( temp[0][0] >= MaxFloatNum || temp[0][0] < MinFloatNum ||
                    temp[0][1] >= MaxFloatNum || temp[0][1] < MinFloatNum ||
                           temp[1][0] >= MaxFloatNum || temp[1][0] < MinFloatNum
                    || temp[1][1] >= MaxFloatNum || temp[1][1] < MinFloatNum )
                {
                        temp[0][0] = temp[0][0] / 10.;
                        temp[0][1] = temp[0][1] / 10.;
                        temp[1][0] = temp[1][0] / 10.;
                        temp[1][1] = temp[1][1] / 10.;
                        *Scale = *Scale * 10;
                }
                Dst->data[0][0] = FloatToFix(temp[0][0]);
                Dst->data[0][1] = FloatToFix(temp[0][1]);
                Dst->data[1][0] = FloatToFix(temp[1][0]);
                Dst->data[1][1] = FloatToFix(temp[1][1]);
        }
        else
                exit(0);
}

void mat_print(MatFix32 *Src)
{
        int row, col;
        for (row = 0; row < Src->rows; row++)
        {
                for (col = 0; col < Src->columns; col++)
                {
                        printf("%d ",Src->data[row][col]);
//                      printf("%.9f ",FixToFloat(Src->data[row][col]));
```

```
                }
                printf("\n");
        }
        printf("\n");
}
```

Listing C.7: Definition of fuction *ekf_reset*

```
void ekf_reset(ekf_info *filter,fix32t *X0,fix32t *X0_1,\
fix32t *P0,int idx,uint8_t unprocessedBlockNum,double tau)
{
        uint8_t i;

        /* the motor load status and set the flag */
        if ((X0[5]>M_lower) && (X0[5]>X_1[5]))
                noLoadSign=0;
        else if ((X0[5]<M_upper) && (X0[5]<X0_1[5]))
                noLoadSign=1;

        /* simulation for abnormal motor core losses */
    bb88=FloatToFix(Prc_s*tau*noLoadSign/C2/ScaleFactor);
    filter->B.data[7][7] = bb88;

        /* predict the unprocessed blocks */
if (idx == 0)
{
        /* safe range of rotor speed change:(-300,600) */
        /* safe range of Tsw change:(-0.2,0.2) */
if ((X0[4]-X0_1[4]>n_min) && (X0[4]-X0_1[4]<n_max)
         && (X0[6]-X0_1[6]>T_min) && (X0[6]-X0_1[6]<T_max))
{
        X0[6] = X0[6] + unprocessedBlockNum*(X0[6]-X0_1[6]);
        X0[7] = X0[7] + unprocessedBlockNum*(X0[7]-X0_1[7]);
        X0[8] = X0[8] + unprocessedBlockNum*(X0[8]-X0_1[8]);
                /* reset X0_1 with X0 */
                for (i = 0; i < 9; i++)
                        X0_1[i] = X0[i];
}
        else
         for (i = 0; i < 9; i++)
         X0[i] = X0_1[i];
}
mat_init(&(filter->X),filter->state_dim,1,X0);
mat_init(&(filter->P),filter->state_dim,filter->state_dim,P0);
}
```

Listing C.8: Declaration of the analog data set*xdcr_AnaDataSet* structure

```
typedef struct {
        xdcr_dataSet_type type; /* type code of data set */
        UInt32 sample_period;   /* sampling period in microsecond */
        UInt32 timestamp;       /* timestamp of first sample */
        UInt16 repCount;        /* number of acquisition steps in data set */
        int16  samples[1];      /* samples as 16 Bit integers, array of dummy
```

```
                                size, transducers can implement bigger size */
} xdcr_AnaDataSet;
```

Listing C.9: The definition of *ekfDataSet*

```
typedef struct
{
        UInt16  repCount;           /* number of samples in one channel */
        double  sample_period;      /* sampling period in microsecond */
        uint8_t unproccedBlockNum;  /* number of lost blocks */
        int16*  samplesPointer;     /* points to samples in first channel */
} ekfDataSet;
```

Listing C.10: The data transmission of the measurement

```
UInt16 nRep = ekfDataSet->repCount;
int16* op = ekfDataSet->samplesPointer;

/* y[2] = {ids; iqs} */
y[0] = FloatToFix(*op / AccuracyScaleFactor / ScaleFactor);
y[1] = FloatToFix(*(op + nRep) / AccuracyScaleFactor / ScaleFactor);

/* u[9] = {vds; vqs; 0; 0; 0; 0; 0; 1; Tc-T0} */
u[0] = FloatToFix(*(op + 2 * nRep) / AccuracyScaleFactor / ScaleFactor);
u[1] = FloatToFix(*(op + 3 * nRep) / AccuracyScaleFactor / ScaleFactor);
u[7] = FixNum_1;
u[8] = FloatToFix((TC - T0) / ScaleFactor);

ekfDataSet->samplesPointer ++;
```

C.4 Compensation of the EKF Estimation

Listing C.11: Compensation of the estimation due to lost blocks

```
for (int i = 6; i < 9; i++)
{
    /* compensate the estimation of lost blocks */
    X0_increment[i-6] = X0[i] - X0_1[i];
    X0[i] = X0[i] + unproccedBlockNum * X0_increment[i-6];
    X0_1[i] = X0[i];
    /* estimate the temperature changes of damaged block */
    X0[i] = X0[i] + X0_increment[i-6];
}
```

CHAPTER D

THE EXPERIMENT RESULTS

Table D.1: Reference parameters of asynchronous machine

Parameters	Symbols	Values
Reference power	P_{ref}	3 kW
Reference voltage	v_{ref}	380 V
Reference current	v_{ref}	6.8 A
Reference temperature	T_{ref}	26°C
Reference speed	ω_{ref}	1415 r/min
Stator temperature coefficient	α_s	0.0039 /°C
Rotor temperature coefficient	α_r	0.0040 /°C
Reference exponent	$power_w$	1.5

Table D.2: Parameters of asynchronous machine

Parameters	Symbols	Values
Nominal output	P_m	3 kW
Nominal voltage	V	380 V
Nominal frequency	f	50 Hz
Nominal torque	T	20 N.m
Connection		delta
Pole pair	p_n	2
Stator resistance	R_s	1.9693 Ω
Rotor resistance	R_r	1.8081 Ω
Main reactance	X_m	52.025 Ω
Stator leakage reactance	X_s	2.02 Ω
Rotor leakage reactance	X_r	2.02 Ω
Rotor's moment of inertia	J_r	0.01654 $kg.m^2$
Iron loss constant	k_{iron}	0.00664 $W/(rad/s)^2$