

**EDGE-BASED MACHINE MONITORING ARCHITECTURES
INCORPORATING OPC UA CONTROLLER DATA**

A Thesis
Presented to
The Academic Faculty

by

Marcel F. Neumann

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
George W. Woodruff School of Mechanical Engineering and the University of Stuttgart

Georgia Institute of Technology
August 2020

COPYRIGHT © 2020 BY MARCEL F. NEUMANN

EDGE-BASED MACHINE MONITORING ARCHITECTURES INCORPORATING OPC UA CONTROLLER DATA

Approved by:

Dr. Christopher Saldana, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Oliver Sawodny
Institute of System Dynamics
University of Stuttgart

Dr. Thomas Kurfess
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Cristina Tarín
Institute of System Dynamics
University of Stuttgart

Date Approved: July 01, 2020

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Dr. Christopher Saldana. He provided me with guidance and support during my year at Georgia Tech. He was always able to give good advice in challenging situations.

I would like to thank Andrew Dugenske for helping me during my year at Georgia Tech. He always supported me with needed hardware, and he taught me a lot about developing IoT architectures and edge devices.

I wish to show my gratitude to Dr. Cristina Tarín, Dr. Thomas Kurfess and Dr. Oliver Sawodny who agreed on serving in my committee. I would especially like to thank Dr. Oliver Sawodny for managing the joint degree. His engagement made it possible that I was able to study at Georgia Tech for the last year.

I would also like to thank the Ford Motor Company for funding my research and thanks to the employees who worked together with me on the project during the last year. Thank you for the great discussions and ideas.

I would like to thank my parents for supporting me during the last six years of studying. I would not have been able to successfully manage these years without your help, your advice and your patience.

Last but not least, I would like to pay my special regards to my lab colleagues, especially Fabia Bayer, Pierrick Rauby and Daniel Newman. Thanks for all the good discussions and for all the good times we had.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
LIST OF TABLES	vi
LIST OF FIGURES	vii
LIST OF SYMBOLS AND ABBREVIATIONS	xi
SUMMARY	xv
CHAPTER 1. Introduction	1
1.1 Motivation	1
1.2 Purpose and Research Questions	3
1.3 Structure	3
CHAPTER 2. State-of-the-Art	5
2.1 Maintenance Management Methods	5
2.2 Theoretical Background	6
2.2.1 Communication Protocols	6
2.2.2 Software Implementation Tools	10
2.3 Related Work	12
CHAPTER 3. Design and Implementation of Edge Monitoring Architectures for Machine Tools	17
3.1 IoT Architecture and Edge Device	17
3.2 Edge Monitoring Architectures	21
3.3 Methods	23
3.3.1 Edge Monitoring Architecture Computational Performance	25
3.3.2 Edge Monitoring Architecture Delay Performance	32
3.4 Assumptions	41
3.5 Results	42
3.5.1 Edge Monitoring Architecture Computational Performance	42
3.5.2 Edge Monitoring Architecture Delay Performance	50
3.6 Discussion	58
CHAPTER 4. Health Monitoring of Rolling Element Bearings	60
4.1 Theoretical Background Rolling Element Bearings	60
4.2 Development Edge Monitoring Algorithm	63
4.2.1 Description of the Test Dataset	63
4.2.2 Bearing Fault Analysis Method	64
4.2.3 Bearing Fault Analysis Result	67
4.2.4 Implementation Edge Monitoring Algorithm	71
4.2.5 Assumptions	74
4.3 Experimental Method	75
4.4 Results and Discussion	77

CHAPTER 5. Conclusion	79
5.1 Summary	79
5.2 Contributions	80
5.3 Future Work	81
APPENDIX A. Test Message MQTT Delay	83
APPENDIX B. Equations for Statistical Parameters	84
APPENDIX C. Detection Time for Different Parameters and Bearings	86
APPENDIX D. Average Parameters Over Days	90
References	95

LIST OF TABLES

Table 1 – Technical specifications of the utilized SBCs.	24
Table 2 – Technical specifications EMCO MILL E350 and controller server of SIMUERIK 828D.	24
Table 3 – Technical specifications accelerometer ADXL203EB.	25
Table 4 – Test cases for evaluating the proposed architectures.	26
Table 5 – Adjusted parameter in the streaming sub-test.	28
Table 6 – Cutting parameters of the example part.	30
Table 7 – Adjusted parameter for measuring MQTT delay.	39
Table 8 – Results of analytics sub-test.	43
Table 9 – Detection time for dataset 3 bearing 3.	68
Table 10 – Detection time for dataset 1 bearing 3.	86
Table 11 – Detection time for dataset 1 bearing 4.	87
Table 12 – Detection time for dataset 2 bearing 1.	88
Table 13 – Detection time for dataset 3 bearing 3.	89

LIST OF FIGURES

Figure 1 – Address space of an OPC UA server.....	7
Figure 2 – Read and write method of OPC UA communication.	7
Figure 3 – Publish and subscribe method of OPC UA communication.	8
Figure 4 – Publish and subscribe model of MQTT.....	9
Figure 5 – Simple Node-RED flow to read a file every 30 seconds, parse it into an MQTT message and send it to an MQTT broker.	11
Figure 6 – Architecture for monitoring and teleoperation of a CNC lathe [40].	14
Figure 7 – Architecture of an open CNC system [41]	15
Figure 8 – Architecture for machine monitoring system [42]	16
Figure 9 – Decoupled digital architecture [45].	19
Figure 10 – Structure of the utilized edge device.	20
Figure 11 – Photo of the utilized edge device.....	21
Figure 12 – Direct edge monitoring architecture.	22
Figure 13 – Indirect edge monitoring architecture.	23
Figure 14 – (a) CAD model of the example part and (b) drawing of the example part including the dimensions.	30
Figure 15 – Tool path of the example part.....	30
Figure 16 – Set-up of the use case trajectory monitoring.	31
Figure 17 – Timeline requesting controller data points for direct edge monitoring architecture.....	33

Figure 18 – Timeline requesting controller data points for direct edge monitoring architecture with maximum expected delay.	34
Figure 19 – Measurement principle response time of controller server.	35
Figure 20 – Timeline requesting controller data points for indirect edge monitoring architecture with maximum expected delay.	36
Figure 21 – Measurement principle of MQTT delay.....	38
Figure 22 – Initial situation of the use case cut monitoring.....	40
Figure 23 – Steps during the cutting operation for the use case cut monitoring with (a) the initial situation, (b) the tool enters the part, (c) the tool is in the part without capturing data and (d) the tool is in the part with capturing data.....	41
Figure 24 – Average data point rate streaming sub-test.	44
Figure 25 – Time difference between received controller data points of the streaming sub-test.	45
Figure 26 – Average data point rate overall system test with (a) incrementing algorithm and (b) FFT algorithm.....	46
Figure 27 – Comparison of time difference between received data points for (a) exclusively streaming controller data points, (b) additionally capturing vibration data and (c) additionally executing FFT.....	47
Figure 28 – Timeline of received controller data points for (a) exclusively streaming controller data points, (b) additionally capturing vibration data and (c) additionally executing FFT.	47
Figure 29 – Average data point rate gateway sub-test of (a) the BBB Rev C and (b) the Raspberry Pi 3 Model B+ as controller gateways.....	49

Figure 30 – Result of the use case trajectory monitoring for (a) exclusively streaming controller data points, (b) additionally capturing vibration data every 10 seconds, and (c) additionally executing a FFT algorithm.....	50
Figure 31 – Response time of controller server for different number of requested data points with error bars indicating the width of six standard deviations.	51
Figure 32 – Maximum expected delay for direct edge monitoring architecture.....	52
Figure 33 – Measured roundtrip delay for two Raspberry Pi’s and AWS broker.	53
Figure 34 – Histogram of roundtrip delay for two Raspberry Pi’s and AWS broker.	53
Figure 35 – Comparison average one-way delay for different broker locations.	54
Figure 36 – Comparison confidence level of one-way delay for different broker locations and devices.....	55
Figure 37 – Maximum expected delay indirect edge monitoring architecture for requesting 10 data points at once, Raspberry Pi as controller gateway, confidence level of 99 % and using local MQTT broker hosted on a Raspberry Pi.	56
Figure 38 – Comparison between the two edge monitoring architectures for travel distance without monitoring vibration data depending on the request frequency.	57
Figure 39 – Structure of a rolling element bearing [58].	61
Figure 40 – Photo (a) inner race defect [59] and (b) outer race defect [60].	61
Figure 41 – Frequency spectrum of bearing with an outer race defect, the BPFO frequency and its harmonics are highlighted by red circles.....	63
Figure 42 – Bearing test rig based on [64].....	64
Figure 43 – Control chart for potential outer race defect magnitude of dataset 3 bearing 3.	67

Figure 44 – Overlap of the different defect frequency intervals.....	69
Figure 45 – Control chart for outer race defect dataset 1 bearing 4.	70
Figure 46 – Comparison of the parameters regarding the remaining time before failure of dataset 3 bearing 3.	71
Figure 47 – Flow chart of the edge monitoring algorithm.....	73
Figure 48 – Live dashboard of RMS with measured RMS values in dark blue and the UCL in light blue.	74
Figure 49 – Sealed accelerometer in plastic box.	76
Figure 50 – Accelerometer box screwed to spindle.....	76
Figure 51 – RMS during warmup program at January 30, 2020.	77
Figure 52 – Average RMS over days.....	78
Figure 53 – Average RMS over time.....	90
Figure 54 – Average arithmetic mean over time.	91
Figure 55 – Average geometric mean over time.....	91
Figure 56 – Average potential outer race defect magnitude over time.....	92
Figure 57 – Average potential inner race defect magnitude over time.....	93
Figure 58 – Average potential rolling element defect magnitude over time.	94

LIST OF SYMBOLS AND ABBREVIATIONS

ADC	Analog to Digital Converter
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AWS	Amazon Web Service
BBB	Beaglebone Black
BPFI	Ball Pass Frequency of Inner Race
BPFO	Ball Pass Frequency of Outer Race
BSF	Ball Spin Frequency
C	Crest Factor
CNC	Computerized Numerical Control
D	Tool Diameter
d_B	Rolling Element Diameter
d_p	Pitch Diameter
ERP	Enterprise Resource Planning
FFT	Fast Fourier Transformation
f_{req}	Request Frequency
f_z	Feed
G	Geometric Mean
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
ID	Identifier
IoT	Internet of Things

IT	Information Technology
JSON	JavaScript Object Notation
k	Number of Initial Measurements
Kurt	Kurtosis
L_{capture}	Travel Distance Tool While Capturing Sensor Data
L_{in}	Travel Distance Tool Without Capturing Sensor Data
M	Maximum Value
m	Number of a Specific Node
MC	Microcontroller
MQTT	Message Queuing Telemetry Transport
MTTF	Mean-Time-To-Failure
N	Rotational Speed
n	Number of Data Points
NC	Numerical Control
NCK	Numerical Control Kernel
NEMA	National Electrical Manufacturer Association
n_r	Number of Rolling Elements
N_s	Number of Samples
OPBC	Open Database Connectivity
OPC UA	Open Platform Communication Unified Architecture
QoS	Quality of Service
RAM	Random-Access Memory
REST	Representational State Transfer
RMS	Root Mean Square
ROS	Robot Operating System

rpm	Rounds per Minute
SBC	Single Board Computer
Skew	Skewness
T_1	Time Value of Triggering Data Point Changes
T_2	Time Value Change of Triggering Data Point Recognized on Edge Device
t_{capture}	Time Needed to Capture Sensor Data
TCP	Transmission Control Protocol
T_q	Current Time Data Points are Queried
T_r	Current Time Data Points are Received
T_{r1}	Timestamp Receiving MQTT Message on Device #1
T_{r2}	Timestamp Receiving MQTT Message on Device #2
T_{s1}	Timestamp Sending MQTT Message on Device #1
T_{s2}	Timestamp Sending MQTT Message on Device #2
UART	Universal Asynchronous Receiver-Transmitter
UCL	Upper Control Limit
v_f	Feed Rate
x	Array with Samples
x_{mean}	Arithmetic Mean
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
z	Teeth Number
α	Scaling Factor
ΔT	Delay
ΔT_b	Delay Sending Message from Gateway to Broker
ΔT_c	Communication Time OPC UA between Client and Server

ΔT_e	Delay Sending Message from Broker to Edge Device
ΔT_{\max}	Maximum Expected Delay
ΔT_{mqtt}	Delay due to MQTT Communication
ΔT_{req}	Request Interval
$\Delta T_{\text{oneway1}}$	One-way delay Raspberry Pi or BBB wireless
$\Delta T_{\text{oneway2}}$	One-way delay BBB Rev C
θ	Contact Angle
σ	Standard Deviation
ω_{spindle}	Spindle Rotational Speed

SUMMARY

The usage of external sensors for machine health monitoring is becoming more popular. A variety of methods for monitoring the condition of CNC machines have been developed by researchers. This study focuses on automation principles for these methods connecting an edge device for sensor data acquisition to the OPC UA controller server of a CNC machine. For this purpose, two different architectures are developed and analyzed in experiments to identify the limitations regarding computational power and time delays. Use cases show the impact of these limitations on machine health monitoring. Additionally, an automated bearing health monitoring algorithm is developed to show the benefits of both architectures.

CHAPTER 1. INTRODUCTION

1.1 Motivation

The topic of this research is the integration of machine controller data and edge monitoring solutions to monitor machine health within an Internet of Things (IoT) architecture. It is a well-known fact that cloud computing technologies have significantly advanced during the last 15 years [1]. Cloud services provide on-demand IT-related capabilities and enable high computational power to be used without buying expensive hardware. However, one important limitation regarding ubiquitous use of cloud computing is network bandwidth [2]. In the era of IoT, increasingly more data are generated by billions of devices. In the domain of manufacturing, transferring of all these data to cloud-based storage may be highly inefficient and costly. As data transfer has been becoming the bottleneck of cloud computing, the concept of edge computing has risen as an attractive alternative [3, 4]. The idea of edge computing is to process data at the edge of a network to reduce data transfer and response time while increasing data safety and privacy. In cloud computing, little control over the data is given since the provider of cloud services is in charge of managing the system.

Data processing for machine health monitoring can be accomplished within cloud computing and edge computing environments. Machine health monitoring is often performed by equipping machines with external sensors to monitor certain physical features of the machine [5]. Lee et al. gave an overview of sensors which have been used in recent research to monitor machine health in a smart factory [6]. A few examples are vibration sensors, force sensors, microphones and temperature sensors. It is often important

to contextualize the sensor data with additional process-related information. For instance, the rotational speed of a bearing is crucial for monitoring the bearing health condition based on statistical parameters of captured vibration data [7]. Many modern CNC machines provide controller data via industrial communication protocols [5, 8]. Open platform communication unified architecture (OPC UA) and MTConnect are two popular examples of modern industrial communication protocols. The controller data can be used by IoT devices to obtain information about the condition of the machine. For example, the current rotational speed or the current executed numerical control (NC) program command can be read by an edge device if connected to the machine controller. Using the controller information, captured data of external sensors can be contextualized. Additionally, it is possible to trigger the start of sensor measurements by certain conditions of the machine. For instance, external sensors could measure data during a specific cut of the CNC machine based on the NC program name and the specific G-code line of this cut. In this case, there are various potential architectures for connecting the server of a machine controller (e.g., OPC UA, MTConnect) to an edge device with the purpose of having controller-sourced data points trigger edge-based sensor measurements.

Several researchers have proposed methods to analyze the performance an OPC UA machine server [9, 10]. However, limited research has been done regarding how the coupled performance of machine controllers and other components of the IoT architecture influence the system performance, especially with regard to data synchronization from disparate sources that include machine controllers and edge sensors. Understanding of the system limitations is crucial for planning and designing IoT architectures [11]. The design of such architectures, including positioning of edge processing gateways and data

aggregation methods, can significantly impact system latency and measurement bandwidth. In this regard, there is limited understanding available regarding how edge-based measurement architectures should be implemented for the purpose of streaming analytics. Therefore, investigation of the usage of controller data by edge devices for machine health monitoring purposes is important. The focus of this study is the usage of OPC UA as communication protocol between the machine controller and the edge device to understand the design of edge processing architectures for machine tools.

1.2 Purpose and Research Questions

This study will address decoupled architectures incorporating machine controller gateways and edge devices coupled to a central message query telemetry transport (MQTT) broker. To address the gap in understanding regarding design and configuration of edge-based architectures, the study seeks to address the research question: How does machine controller gateway and edge analytics configurations determine computational and delay performance? The present study will inform how edge architectures should be designed for specific analytics applications. For this purpose, use cases are developed to illustrate the impact of computational and delay performance limitations on machine health monitoring applications. Additionally, a bearing health monitoring algorithm based on capturing of vibration data with an accelerometer is developed to illustrate the benefits of the developed architectures.

1.3 Structure

The remainder of this thesis is organized into the following chapters. The second chapter describes the state-of-the-art. The third chapter discusses the development and the analysis

of two different architectures which use controller data to automatically trigger sensor measurements. The fourth chapter describes the development of a bearing health monitoring algorithm using the benefits of the developed architectures. The last chapter summarizes the results of this study and discusses future work.

CHAPTER 2. STATE-OF-THE-ART

This chapter is divided into three parts. First, different maintenance strategies are described. The second subsection discusses some necessary background for understanding this study. In the last part of this chapter, the work of other researchers in the context of this study is analyzed.

2.1 Maintenance Management Methods

Maintenance cost can account for 15 to 60 percent of the production costs of goods [12]. In literature, the definition of maintenance management methods varies [13]. Mobley defines three different main methods: run-to-failure management, preventive maintenance and predictive maintenance [14]. Within run-to-failure-management approaches, maintenance actions are only taken after a machine fails. Before the failure of a machine, money is not spent on maintenance. This method is a reactive technique and the most expensive maintenance method. In contrast, preventive maintenance approaches are time driven. This means that maintenance tasks are scheduled based on the operation time of a machine. The maintenance tasks are often scheduled based on the mean-time-to-failure (MTTF) statistics. If a machine normally runs for 12 months before it needs to be replaced, replacement of the machine could be scheduled after 11 months, for instance. The drawbacks of using preventive maintenance are possibly wasteful repairs since the machine could get repaired earlier than necessary as well as catastrophic failures of machines earlier than expected since the actual condition of the machines is not monitored. Lastly, predictive maintenance approaches do not rely on time driven statistics. Maintenance tasks are scheduled based on the actual condition of a machine. The condition of the machine is

monitored over time and if the condition of the machine changes, the machine is repaired. The condition of a machine is often monitored using sensors. Predictive maintenance improves productivity and product quality of a manufacturing system and reduces maintenance costs. Improving predictive maintenance methods by using OPC UA controller data is the motivation of this study. The triggering of sensor measurements by specific controller data points can be used to automate predictive maintenance methods. This leads to a more efficient predictive maintenance system.

2.2 Theoretical Background

2.2.1 Communication Protocols

Communication protocols used in many IoT applications are, for instance, websockets, advanced message queuing protocol (AMQP), extensible messaging and presence protocol (XMPP), MTConnect, OPC UA and MQTT [15, 16]. MTConnect is an industrial communication protocol to exchange data between different machines [17–19]. The purpose of MTConnect is to connect different devices in the manufacturing area in a simple and standardized way. The MTConnect agent is the central element of MTConnect. The MTConnect agent collects machine data and provides the data to clients. A client can access data of an MTConnect agent via hypertext transfer protocol (HTTP) requests. The schema of a message is encoded using extensible markup language (XML). MTConnect was not used for experiments in this study. However, the proposed measurement principles for OPC UA communication can be used to analyze the limitations of a MTConnect communication for triggering sensor data collection by controller data.

OPC UA is an industrial communication protocol to transfer data between different machines [20]. OPC UA is based on the server-client-model. An OPC UA server provides its data hierarchically in an address space. Figure 1 illustrates a simple address space. The address space is comparable to a folder structure on a computer. Each data point in the address space is called a node [21]. Each node has a unique node ID which refers to this node. Additionally, each node can have different attributes like a description, the value of the node or a display name.

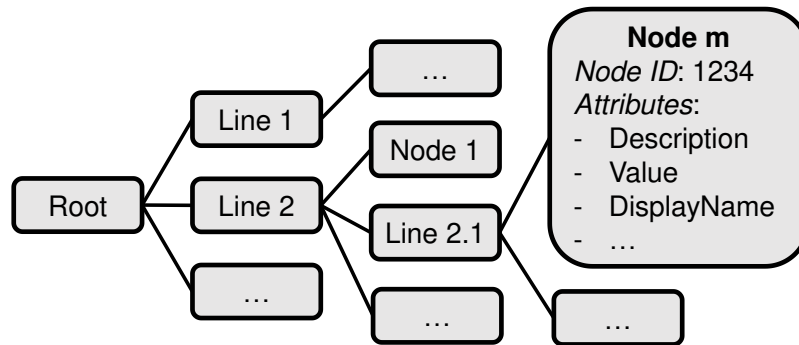


Figure 1 – Address space of an OPC UA server.

An OPC UA client can access the value of a node by different methods [22]. The simplest method is the read and write method. The OPC UA client can read or write one or more attributes of a node. The OPC UA client refers to a specific node by its node ID. The OPC UA client sends a request to the OPC UA server and the OPC UA server sends back a response. This is illustrated in Figure 2.

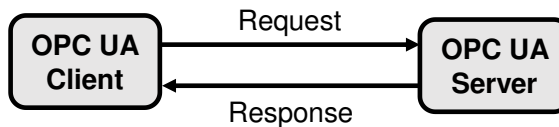


Figure 2 – Read and write method of OPC UA communication.

Another method of accessing data is the subscription mechanism. The OPC UA client sends a subscription request for one or more nodes to the OPC UA server. The principle of the subscription method is shown in Figure 3. The OPC UA server only sends a message to the OPC UA client if the data of the subscribed nodes change. The publishing interval of the OPC UA server can be adjusted.

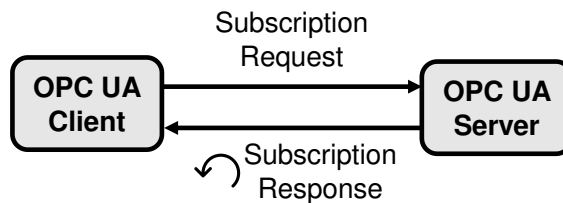


Figure 3 – Publish and subscribe method of OPC UA communication.

The OPC UA server of the Siemens controller SIMUERI 828D [23] was used in this study. The publish and subscribe method was not used in this study. Writing to the OPC UA server was not considered in this study since the purpose of this study is to use OPC UA information to trigger measurements of external sensors. Subsequently, a node of the OPC UA controller server is called a controller data point to avoid confusion with the programming tool Node-RED.

MQTT is a lightweight communication protocol based on a publish and subscribe model [24]. An MQTT message consists of two different main parts, the topic of the message and the message itself (i.e., the info). The publish and subscribe model is shown in Figure 4. The model consists of three different components. The publisher sends messages with a specific topic to a broker. The broker is the middleware of the communication process and responsible for routing the messages between different devices. The subscriber subscribes to messages based on the topic of the message. Thus,

the first step of the subscriber is sending a subscription request for a specific topic to the broker. Whenever the broker receives a message with this topic, it forwards this message to the subscriber. More than one device can subscribe to one message topic and more than one device can publish messages with the same message topic.

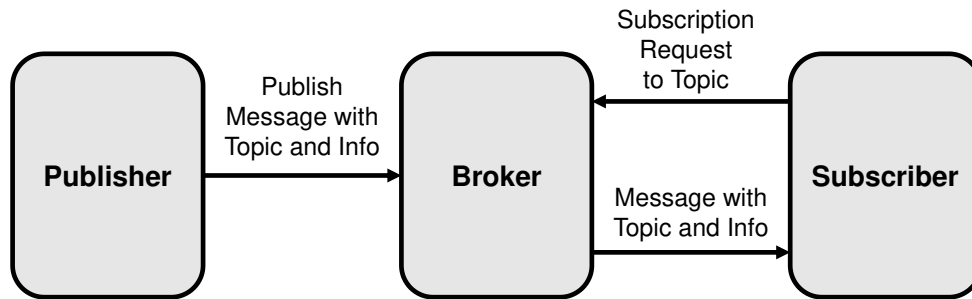


Figure 4 – Publish and subscribe model of MQTT.

The MQTT communication protocol is used in this study since it is lightweight and frequently used in IoT architectures. Additionally, MQTT is an open-source protocol and simple to implement. There are three different Quality of Service (QoS) levels defined for MQTT messages [25]. The QoS levels refer to the guarantee whether an MQTT message was successfully delivered. If the QoS level is set to zero, a message is sent once without ensuring that the message was received by the broker. The QoS level one ensures that the message is received at least one time by the broker. This means that the same message could be delivered more than once. The message is stored in an internal buffer of the publisher until it receives confirmation that the message was delivered successfully. If the QoS level is set to two, the protocol ensures that a message was delivered exactly once. A message cannot be lost or sent twice with this QoS level. For this, a two-step confirmation process is needed. This results in a bigger overhead since sending one MQTT message requires transferring four messages in total. QoS level two was chosen for this study since

it is the most reliable level of MQTT communication. Additionally, this study concentrates on measuring limitations of delays and computational power of edge devices. QoS level two is a worst-case consideration since its overhead is bigger than the overhead of the other QoS levels, resulting in longer communication time.

2.2.2 Software Implementation Tools

Node-RED and Python 3.5.3 are used in this study to execute algorithms on edge devices. Node-RED is a browser-based tool to program event-driven applications [26–28]. In Node-RED, different nodes with certain functionalities are combined to flows by wiring the nodes. Messages are transferred along the wired nodes. Node-RED provides a palette of built-in nodes, which can be used to define an application. Additionally, over 225,000 open-source modules are available, adding nodes to the palette to extend the range of functionalities. Since Node-RED runs based on an operating system, its execution is not deterministic. Consequently, Node-RED cannot be used directly for sensor data acquisition with a defined sampling frequency.

Figure 5 shows a simple Node-RED flow, which reads a certain file on a device every 30 seconds, parses the content of the file to a message and sends the message via MQTT to a broker. This task can be done with four nodes in total. The function node is one of the most important nodes in Node-RED. In Figure 5, it is used to parse the content of the file to a message with a certain structure. The function node provides the possibility to integrate JavaScript code in the Node-RED flow.

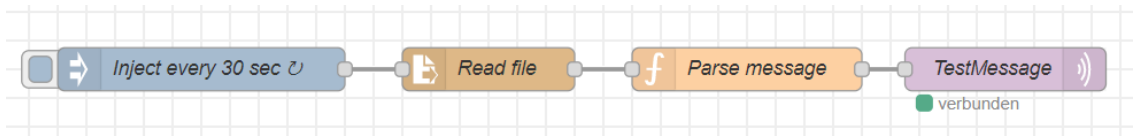


Figure 5 – Simple Node-RED flow to read a file every 30 seconds, parse it into an MQTT message and send it to an MQTT broker.

Node-RED is built on Node.js. Therefore, the nodes are programmed in JavaScript. Node-RED can be run locally on a computer, on an edge-device or in the cloud. The flows are stored in JSON files. The JSON files can easily be exchanged between different devices or developers. Node-RED also provides a live dashboard for visualization purposes. For instance, the dashboard can contain charts, text, buttons or sliders.

Node-RED was chosen in this study for developing several applications due to its simplicity. A great variety of applications can be quickly developed. Node-RED is frequently used in IoT applications since it can be run on a lot of different devices. Additionally, sending and receiving messages is often necessary, which can be easily done in Node-RED. However, Node-RED and JavaScript are not the ideal tools to execute complex data analytics [29, 30]. Executing complex data analytics in a function node in Node-RED slows down other tasks in Node-RED. However, Node-RED provides the opportunity to call other scripts in an execution node. Instead of using a function node to execute data analytics, a script in any programming language can be called to execute this task. Python was chosen to implement data analytic algorithms in this study because of its big community and the wide range of available libraries for data analytics [31, 32].

2.3 Related Work

Several researchers proposed methods to detect anomaly and monitor health condition of manufacturing systems by equipping machines with different sensors. Liu et al. proposed a method to detect anomaly in manufacturing systems by using a structured neural network to analyze the data of 151 sensors [33]. Chen et al. proposed two approaches to detect machine anomaly based on raw energy consumption data from a real machine captured with sensors [34]. Boud and Gindy proposed an application for multi-sensor signals for monitoring the tool and workpiece condition of a broaching machine [35]. Other researchers proposed methods to detect anomaly of manufacturing systems by analyzing controller data. Zhang et al. proposed an anomaly detection method analyzing CNC machine controller data collected with proprietary adapters [36]. Maez et al. proposed a method to detect anomaly of machines based on information of the OPC UA controller server (e.g. current, position and velocity of the spindle and the axes) [37]. In summary, the majority of researcher proposed and developed methods for anomaly detection based on sensor measurements with test datasets collected in experiments. However, the sensor data acquisition in an industrial application needs to be automated to monitor the manufacturing processes permanently. Other researchers used controller data to detect anomaly of manufacturing systems. This study developed and analyzed methods to automate sensor data acquisition based on machine controller data. Edge devices can get information about the status of the machine from the machine controller to start sensor data acquisition at certain states of the machine. This way, machine health condition monitoring can be automated. The proposed methods provide also the possibility to connect sensor and controller data for monitoring machine health and detecting anomalies.

The purpose of this study is the development and analysis of architectures providing the possibility to trigger sensor data acquisition by controller data. Several researchers developed solutions for providing access to sensor data via OPC UA or MTConnect [38, 39]. The sensor data is collected by one device, which hosts an OPC UA server or an MTConnect agent. External applications can then access the data by using OPC UA or MTConnect, respectively. The purpose of this study is not to provide sensor data via OPC UA or MTConnect to different applications. The goal of this study is to use controller data of CNC machines to trigger sensor data acquisition. The data can be used for machine health monitoring processes.

Álvares et al. proposed an architecture for monitoring and teleoperation by connecting a CNC lathe using MTConnect and OPC UA protocols [40]. The proposed architecture is shown in Figure 6. An MTConnect agent and an OPC UA server are hosted in a public cloud and both are connected to the CNC machine via Transmission Control Protocol (TCP). The monitored data items are stored in a database in the cloud. Users can access this data with computers or mobile devices via HTTP requests. The purpose of their architecture is to provide users with machine data. However, using the controller data for contextualizing data acquired by external sensors or to trigger sensor measurements is not considered in their architecture.

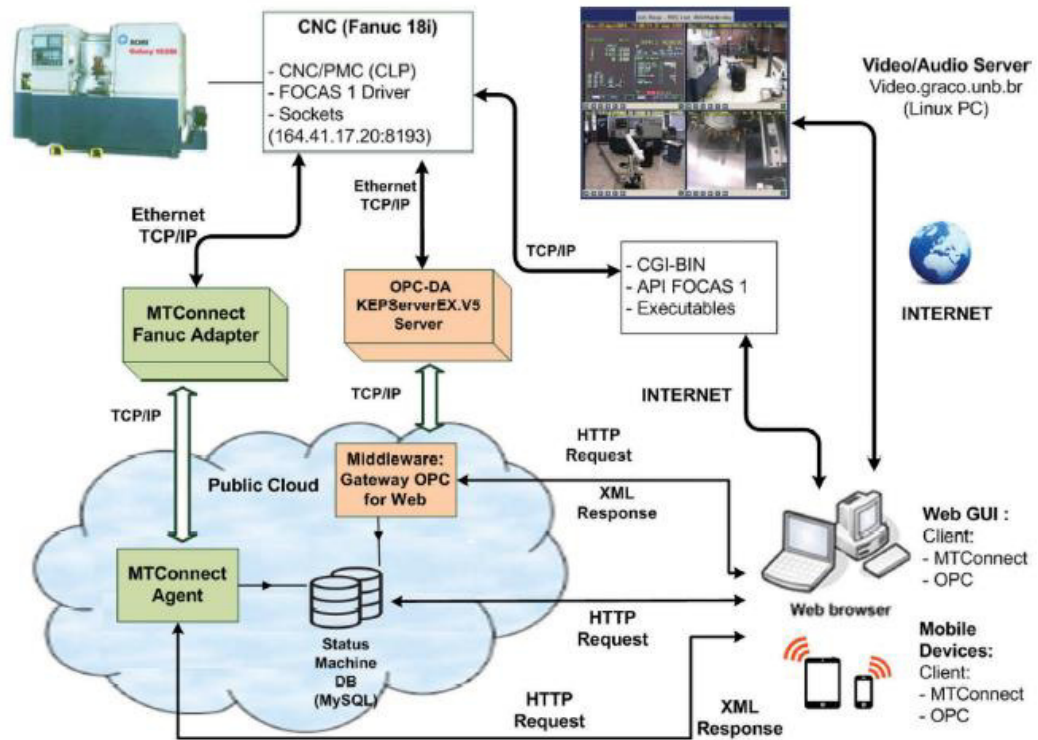


Figure 6 – Architecture for monitoring and teleoperation of a CNC lathe [40].

As shown in Figure 7, Sun et al. introduced an open CNC system design connecting the Numerical Control Kernel (NCK) system and a peripheral perception system directly to a Human Machine Interface (HMI) system [41]. The NCK system provides data of the machine controller to the HMI system. The peripheral perception system includes several sensors which provides sensor data to the HMI system. The HMI System hosts an MTConnect agent. In this way, a manufacturing execution system can collect data from the open CNC system via MTConnect. Sun et al. include external sensors for data acquisition into the CNC system. In comparison, the purpose of the present study is to use existing MTConnect or OPC UA servers hosted by the machine controller to contextualize sensor data or to trigger data acquisition.

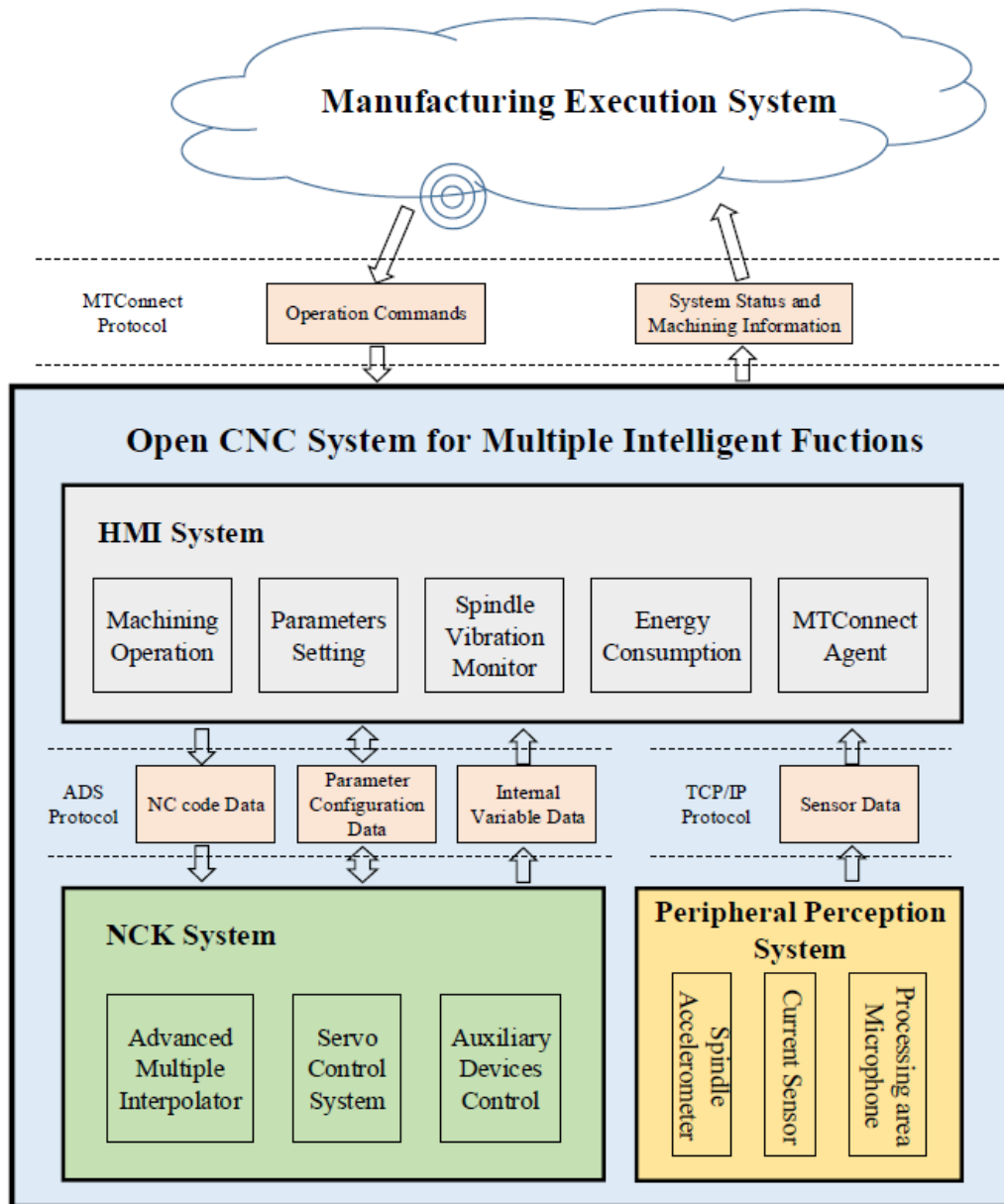


Figure 7 – Architecture of an open CNC system [41]

Chen et al. described the development of a monitoring system for CNC machines using MTConnect [42]. The architecture of the monitoring system is shown in Figure 8. The monitoring system provided the data to other applications via MTConnect. The monitoring system can be connected to OPC UA servers of the machine, to MTConnect agents of the machine, to the machine tool itself and to external sensors. For collecting external sensor

data, a hardware adapter was used which collects and processes the data. The data can be sent to the main monitoring system via socket. Sensor data acquisition can be triggered with this architecture, but this topic was not investigated by Chen et al, particularly in terms of the limitations regarding computational power and delays.

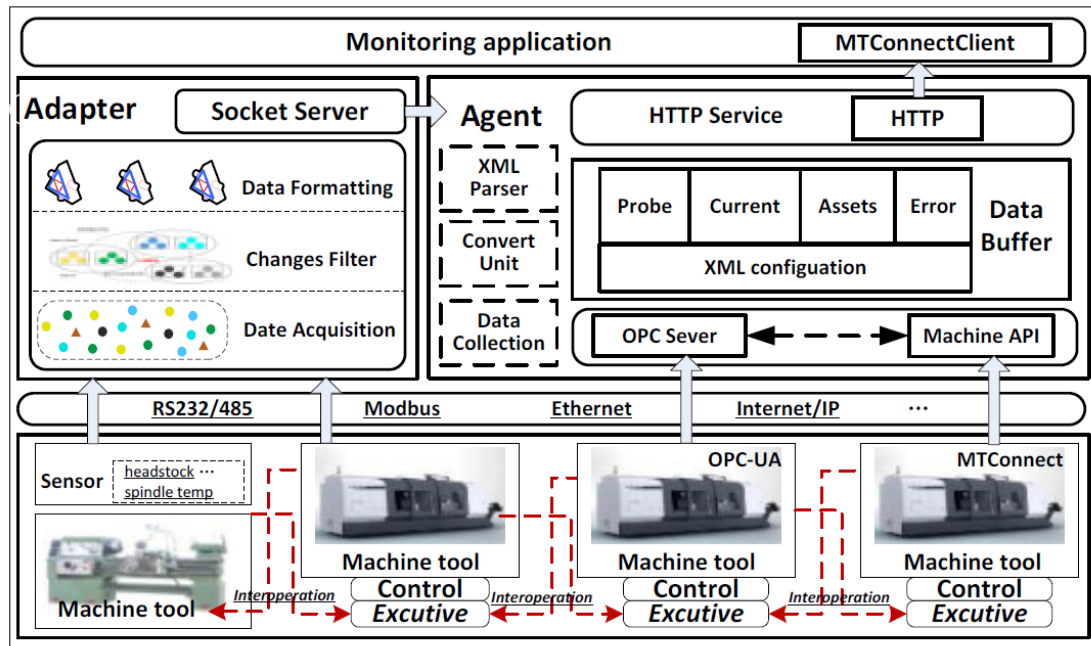


Figure 8 – Architecture for machine monitoring system [42]

In summary, little research has been done regarding how controller data of CNC machines can be made available for edge devices to trigger sensor measurements. The limitations regarding the computational power of edge devices and the time delays were insufficiently analyzed in literature. This study focuses on identifying these limitations in the context of triggering sensor data acquisition by OPC UA controller data points.

CHAPTER 3. DESIGN AND IMPLEMENTATION OF EDGE MONITORING ARCHITECTURES FOR MACHINE TOOLS

CNC machines often provide controller data by various possible communication protocols (e.g., OPC UA, MTConnect). The controller data can be used to trigger different measurements with external sensors during specific states or conditions of the machine. In this study, two different edge monitoring architectures for triggering measurements are proposed and evaluated. Both architectures are analyzed with several experiments to identify their limitations regarding computational power and communication delays. First, the decoupled IoT architecture and the utilized edge device is described. Then, the development of two edge monitoring architectures and the methods to identify their limitations are described. Results of these performance evaluations are then presented and outcomes of the analysis are summarized.

3.1 IoT Architecture and Edge Device

IoT architectures in a manufacturing environment must be able to connect a range of different devices and machines, while also managing different hardware, software and communication protocols. Researchers have proposed several architectures to accomplish these connections. For instance, Raileanu et al. proposed an architecture where aggregation nodes are used to connect multiple device with different communication protocols on the shop floor and the aggregation node sends the aggregated data via HTTP and open database connectivity (ODBC) to the cloud [43]. Wen et al. proposed an architecture where gateways are used to connect devices with several communication protocols to a central

database via representational state transfer (REST) application programming interfaces (APIs) [44]. Newman et al. proposed a digital architecture based on the communication protocol MQTT [45]. Figure 9 shows this proposed digital architecture, which is also used in the present study. The central component of this architecture is the MQTT broker. MQTT is used to transfer messages between components of this architecture. Various manufacturing equipment are shown at the bottom of Figure 9. Depending on the equipment, different communication protocols such as MTConnect, OPC UA or robot operating system (ROS) can be integrated. These communication protocols are translated into standardized MQTT messages by gateways, which are computing devices connected to an MQTT broker. In this way, information can be exchanged between diverse equipment although they might support differing communication protocols. At the top of Figure 9, various components of a manufacturing network are shown. Components like databases, enterprise resource planning (ERP) systems and predictive maintenance applications can be connected to an MQTT broker. A gateway might be used to parse the MQTT message into common format, which can be processed by these applications. Edge devices can be integrated into this architecture to capture sensor data on the machine floor level. The raw data can be sent directly to the MQTT broker or the raw data can be processed locally on the edge device. By processing locally, it is possible to only send the result of this process via MQTT to other components of the architecture. Sending the result of this process instead of the raw data might reduce data traffic. One main advantage of the architecture shown in Figure 9 is its modularity. The components in this architecture are decoupled and can be exchanged without affecting other components.

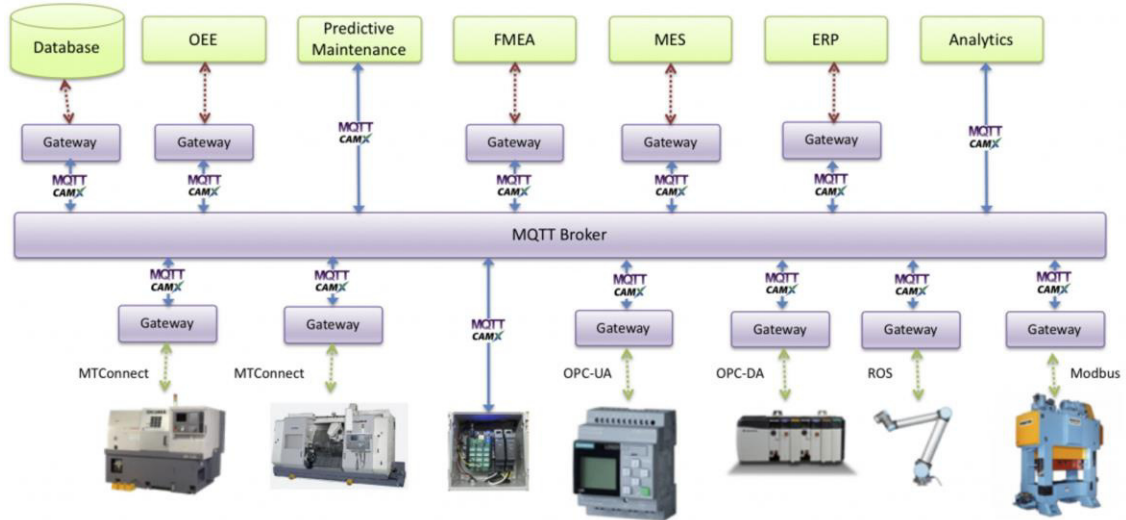


Figure 9 – Decoupled digital architecture [45].

The structure of the edge device used in the present study is shown in Figure 10 [45, 46]. The edge device consists of a single board computer (SBC), the Beaglebone Black (BBB) Rev C and a dedicated microcontroller (MC) Teensy 3.2. The BBB Rev C of the edge device is referred to as the edge device SBC. The Teensy 3.2 is referred to as the edge device MC. The edge device MC connects to edge sensors and transforms the analog signals of these sensors into digital signals using built-in analog to digital converters (ADCs). The data collection code executed by the edge device MC is written in C. The edge device MC is used for real-time, deterministic data collection. The edge device MC and the edge device SBC communicate via universal asynchronous receiver-transmitter (UART). Node-RED is installed on the edge device SBC and used to execute higher order tasks like sending and receiving MQTT messages. The UART communication with the edge device MC is also managed by Node-RED. Since the edge device SBC does not have a built-in Wi-Fi antenna, an external Wi-Fi adapter is attached to the edge device SBC via USB. The edge device is connected to the MQTT broker of the described digital architecture via Wi-Fi. In comparison to the edge device described in [45], the BBB Rev C

is used in this study instead of the BBB wireless as the edge device SBC. The BBB Rev C with an attached Wi-Fi adapter provides a more reliable Wi-Fi connection. The Edimax EW-7811Un Wi-Fi adapter was used in this study [47].

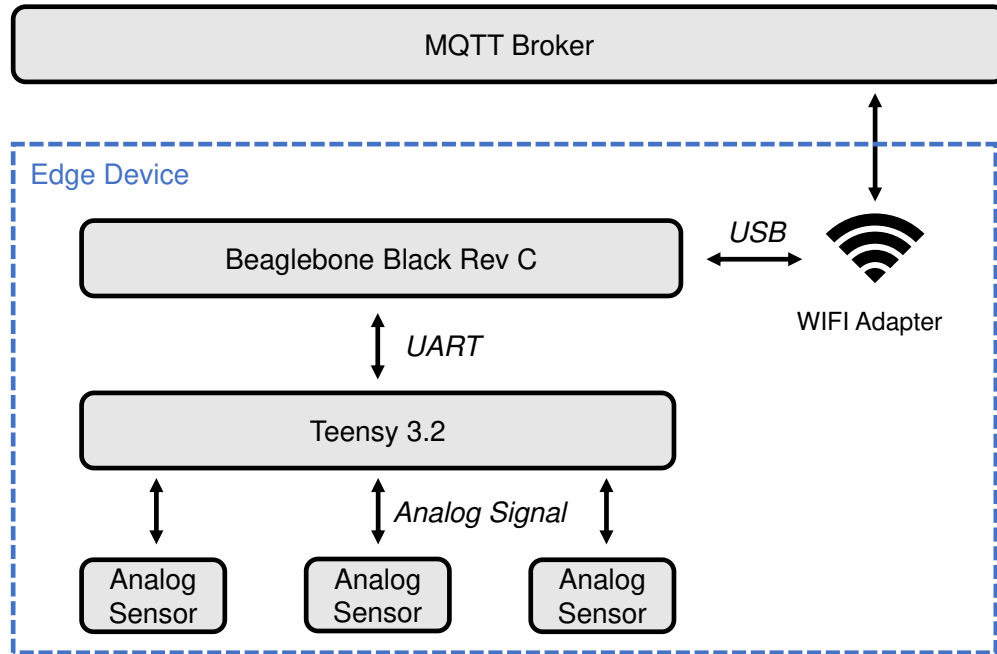


Figure 10 – Structure of the utilized edge device.

Figure 11 depicts the edge device utilized in the present study. The edge device consists of an edge device MC and an SBC placed in a NEMA-rated enclosure with an integrated power supply and M8 analog sensor connections. In this study, an accelerometer is connected to the edge device. The edge device SBC has a built-in Ethernet adapter for direct connection to the machine controller, using an OPC UA server in the present study. Node-RED provides nodes to establish a communication with the OPC UA server. The edge device is consequently able to collect data from external sensors while being connected to an OPC UA server via Ethernet.

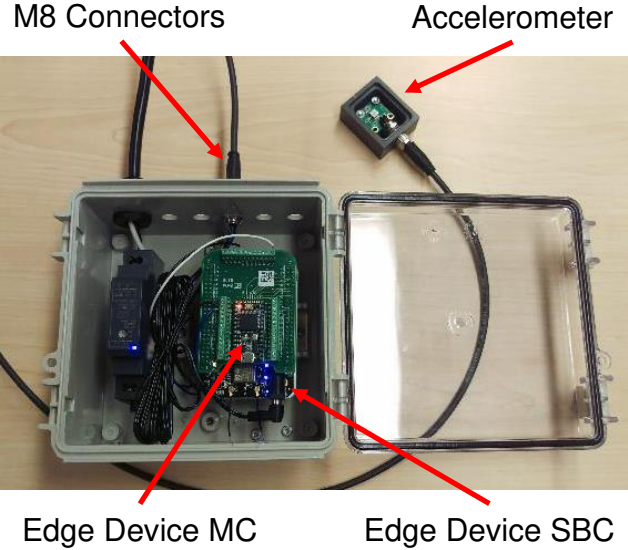


Figure 11 – Photo of the utilized edge device.

3.2 Edge Monitoring Architectures

In the present study, two different edge monitoring architectures are proposed that vary based on the integration of controller gateway functions. The first architecture is referred to as the direct edge monitoring architecture and is shown in Figure 12, this consisting of a direct connection of the OPC UA server of a machine controller to the edge device via Ethernet. In this way, the edge device SBC can request controller data points directly from the OPC UA server. The OPC UA server of the machine controller is referred to as the controller server.

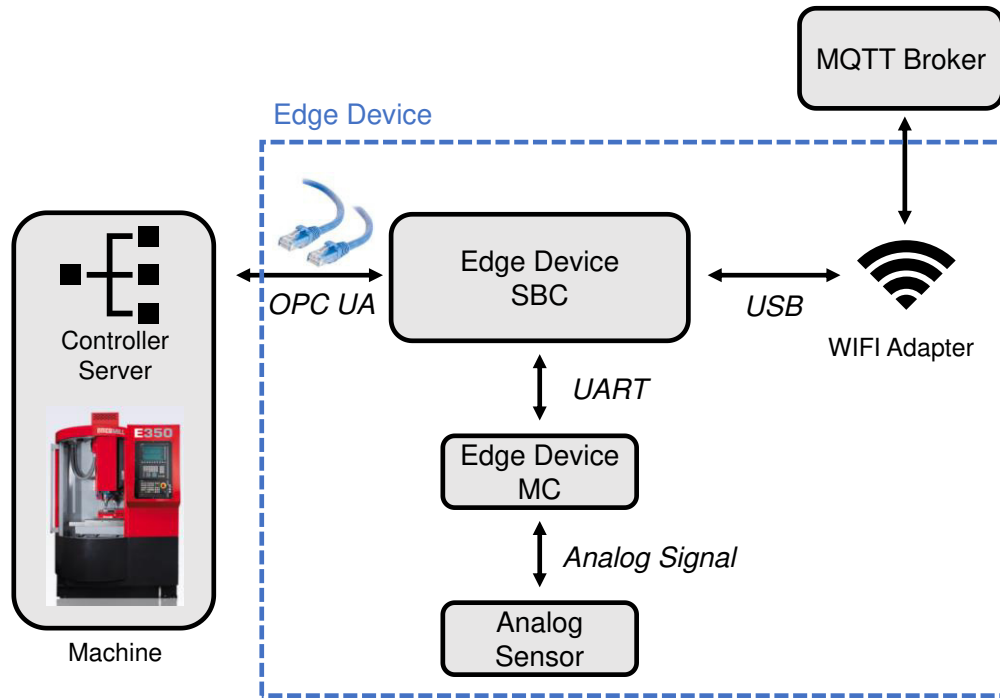


Figure 12 – Direct edge monitoring architecture.

The second proposed architecture is referred to as the indirect edge monitoring architecture and is shown in Figure 13, this consisting of an edge device not directly connected to the machine controller, but instead connected through a central MQTT broker. In this architecture, a secondary device working as an OPC UA gateway is used to send controller data points to the central broker. The OPC UA gateway is referred to as the controller gateway. By using a controller gateway, the edge device can still receive controller data points via MQTT from the broker by subscribing to certain topics of MQTT messages. In this configuration, an additional delay is likely to be introduced by sending the controller data points via MQTT to the edge device instead of directly connecting to the controller server. In the indirect edge monitoring architecture, a second hardware device for the controller gateway is needed, which increases the cost in comparison to the direct connection. However, the workload on the edge device might be reduced since the edge

device is only connected to the MQTT broker. This might improve the performance of the edge device to execute data analytics algorithms.

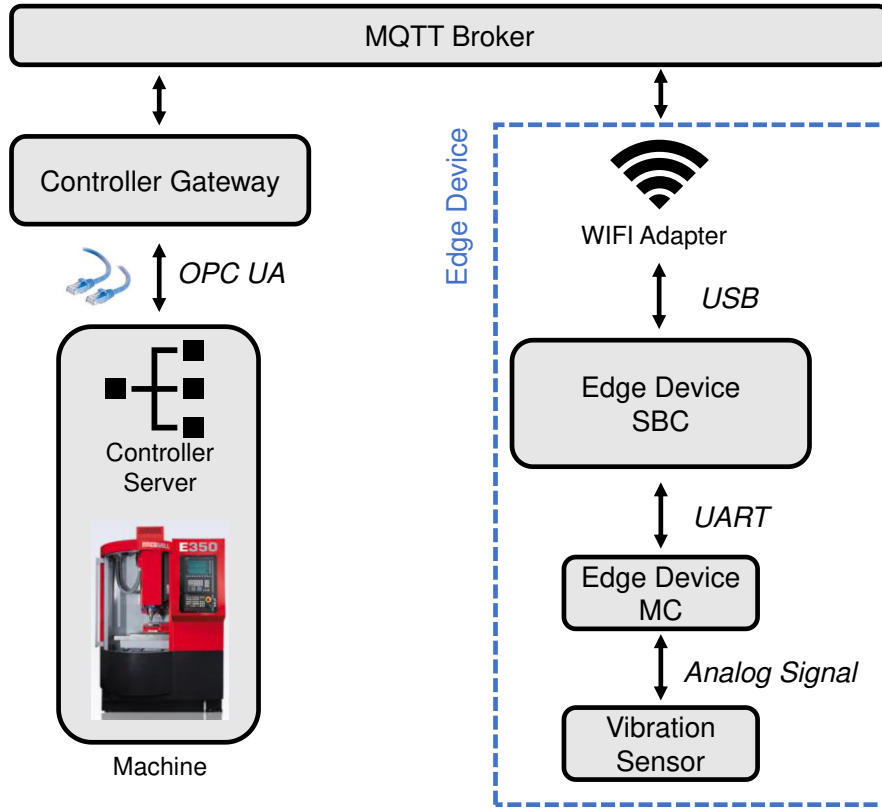


Figure 13 – Indirect edge monitoring architecture.

3.3 Methods

The proposed direct and indirect edge monitoring architectures were analyzed to identify their limitations. Three low-cost single board computers (SBC) working as a controller gateway were tested in this study: BBB Rev C, BBB wireless and Raspberry Pi 3 Model B+. Table 1 summarizes technical specifications of the three SBCs. Similar measurement conditions were used for both architectures. The controller server of an EMCO MILL E350 3-axis mill produced by the EMCO GmbH was used to perform

several experiments. Some technical specifications of the machine and its controller server hosted by the controller SIMUERIK 828D is shown in Table 2 [23, 48].

Table 1 – Technical specifications of the utilized SBCs.

	Beaglebone Black Rev C[49]	Beaglebone Black wireless [50]	Raspberry Pi 3 Model B+[51]
Processor Clock	1 GHz	1 GHz	1.4 GHz
RAM	512 MB	512 MB	1 GB
Connectivity	<ul style="list-style-type: none"> • USB • Ethernet 	<ul style="list-style-type: none"> • USB • Wi-Fi 	<ul style="list-style-type: none"> • USB • Wi-Fi • Ethernet

Table 2 – Technical specifications EMCO MILL E350 and controller server of SIMUERIK 828D.

	Feature	Value
EMCO MILL E350	Travel in X/Y/Z	350 / 250 / 300 mm
	Turing speed range	50 to 10,000 rpm
	Rapid motion speeds in X/Y/Z	24 / 24 / 24 m/min
Controller server of SIMUERIK 828D	Maximum samples per second	500 1/s
	Minimum sampling interval	100 ms
	Number of subscriptions	5
	Number of sessions	5

The communication between the different devices was managed using Node-RED. Data analytics algorithms were performed in Python 3.5.3. All devices with Wi-Fi capability were connected to the network GTother provided by the Georgia Institute of Technology during all experiments. If data acquisition with an external sensor was needed for an experiment, the accelerometer ADXL203EB was attached to the edge device to capture vibration data. The technical specifications of the ADXL203EB are summarized Table 3 [52].

Table 3 – Technical specifications accelerometer ADXL203EB.

Feature	Value
Sensitivity	20 mV/g
Bandwidth	0.5 to 2,500 Hz
Number measured axes	2
Spectral noise	110 $\mu\text{g}/\text{Hz}^{0.5}$

3.3.1 Edge Monitoring Architecture Computational Performance

To determine the limitations with respect to the computing performance, five different test cases were developed. Table 4 summarizes the workload on the edge device SBC and the edge device MC, the adjusted parameters and the metric of interest for the five test cases. For the direct edge monitoring architecture, the workload on the edge device SBC was increased from the measurement sub-test to the overall system test. For the indirect edge monitoring architecture, the performance of different hardware working as a controller gateway was evaluated.

In the measurement sub-test of the direct edge monitoring architecture, the edge device read three controller data points with a specified request frequency of 10 Hz. If the values of the controller data points fell within a certain range of pre-determined values (e.g., trigger), the collection of vibration data is initiated. Specifically, vibration data was captured if the rotational speed of the spindle was above zero. During the collection of analog sensor data, the edge device did not read controller data points. The sensor data was parsed to an MQTT message and the raw vibration data sent to an MQTT broker. The measurement sub-test was used to identify the limits of the edge device MC regarding the sampling rate and the sample size for capturing sensor data. The sample size is the number of measurement points which are sampled during one sample of vibration data. In this test,

the sampling rate and the sample size were increased incrementally to identify their maximum value.

Table 4 – Test cases for evaluating the proposed architectures.

Architecture	Direct				Indirect
Test Cases	Measurement sub-test	Analytics sub-test	Streaming sub-test	Overall system test	Gateway sub-test
Workload on SBC	Reading three controller data points and comparing one data point to a defined value	Reading three controller data points and comparing one data point to a defined value	Streaming controller data points to MQTT broker and comparing one data point to a defined value	Streaming controller data points to MQTT broker and comparing one data point to a defined value	Streaming controller data points to MQTT broker
	Sending raw data to MQTT broker	Executing edge data analytics algorithm Sending raw data to MQTT broker	Sending raw data to MQTT broker	Executing edge data analytics algorithm Sending raw data to MQTT broker	
Workload on MC	Sensor data capture	Sensor data capture	Sensor data capture	Sensor data capture	None
Adjusted Parameters	Sampling frequency	RAM usage	Request frequency	RAM usage	Gateway hardware
	Sample size		Requested number of data points	Request frequency Requested number of data points	Request frequency Requested number of data points
Metric of Interests	Maximum sampling frequency	Computation time	Data point rate	Data point rate	Data point rate
	Maximum sample size				

Performing data analysis on the edge device and sending the result to the broker might reduce network traffic in comparison to sending the raw data to the broker. This was

simulated in analytics sub-test in Table 4. The edge device performed an edge analytics algorithm, this increasing the workload of the edge device SBC in comparison to the measurement sub-test. Two algorithms were implemented to determine effects of random access memory (RAM) usage on computation time in this test. The first algorithm involved simple incrementing of a variable from 0 to 95,000 and generally used a small amount of RAM. The second algorithm executed a fast fourier transformation (FFT) to transform vibration data from the time domain into the frequency domain. The FFT algorithm used a Hann window and Welch's method for de-noising purpose, resulting in significantly more RAM usage compared to the first algorithm. The vibration data was captured every 10 seconds and sent to the MQTT broker after the execution of the edge analytics algorithm. Additionally, three controller data points were read from the controller server with a request frequency of 10 Hz. The controller data point for rotational speed was compared against a defined value (i.e. more than 0 rpm) to simulate the triggering of the vibration measurement by a controller data point in the Node-RED flow. The controller data points were not read during the time the example algorithms were executed and during the time the sensor data was captured. The execution time was measured 140 times for both algorithms.

The controller data points might be needed by other devices or applications in the manufacturing network for monitoring the status of the machine. Therefore, the controller data points need to be streamed to the MQTT broker to make them available to other devices or applications. This was simulated in the streaming sub-test in Table 4. The controller data points were streamed to the MQTT broker in addition to sending the raw data to the MQTT broker. For this experiment, the vibration measurements were taken

every 10 seconds to ensure that vibration data was captured in a certain frequency during the runtime of the experiment. Additionally, the controller data point for rotational speed was compared against a specific value (i.e. more than 0 rpm) to simulate the triggering of the vibration measurement by a controller data point in the Node-RED flow. Controller data points were streamed to the MQTT broker with different conditions corresponding to request frequency f_{req} and the number of data points n for each request. The different parameters used in this study are summarized in Table 5. A request frequency of 2 Hz and a number of data points of 20 indicates that every 500 ms, 20 different controller data points was requested. The metric of interest in this test was the data point rate, which is defined as the ratio of the number of received data points to the number of theoretical requested data points. The theoretical number of data points was calculated by the request frequency f_{req} and the number of data points n for each request, according to Equation 1:

$$data\ point\ rate = \frac{Received\ \# \ data\ points\ in\ 10\ sec}{f_{req} * n * 10\ sec} * 100\% \quad (1)$$

Each measurement interval was set to 10 seconds. The number of received data points in 10 seconds was determined on a second device which was subscribed to the MQTT topic of the controller data point messages. The second device counted the received controller data points in the 10 seconds interval. For each combination of request frequency f_{req} and number of data points n , 13 measurements were taken.

Table 5 – Adjusted parameter in the streaming sub-test.

Parameter	Values
Request frequency f_{req}	1 Hz, 2 Hz, 5Hz, 10 Hz
Number of data points n	10, 20, 30, 40, 50

The overall system test in Table 4 simulated the case of integration on a single edge device the functions of streaming of controller data points and executing edge analytics. The measurement conditions were the same as for the streaming sub-test. Additionally, the incrementing algorithm and the FFT algorithm were executed in parallel. The algorithms were executed constantly during the runtime of the experiment. The metric of interest was the data point rate in the overall system test.

For the indirect edge monitoring architecture, an SBC was used as a controller gateway to stream controller data points to the MQTT broker. The performance of different SBCs working as controller gateways were tested in the gateway sub-test in Table 4. The data point rate of a controller gateway was determined according to Equation 1. The devices streamed the controller data points exclusively to the MQTT broker without additional workload since the devices were used as controller gateways. Among the SBCs considered in the present study, the BBB wireless was not tested to identify its computational performance limits due to the poor performance regarding the delay performance as described in the ensuing.

The impact of the limitations regarding the computational performance of the direct and indirect edge monitoring architectures are illustrated for the use case of trajectory monitoring. The path of the cutting tool was monitored during a cutting operation by the controller data points x-position and y-position. The CAD model of the example part for this use case is shown in Figure 14(a). The structure consists of a linear slot and three equally distanced circles. The dimensions of the example part are shown in the drawing in Figure 14(b). For this simulation, a 10 mm endmill was utilized and the toolpath of the cutting tool is shown in Figure 15. The toolpath consists of in total four linear cuts bridged

by three circular arc toolpaths. Cutting parameters for aluminum were utilized [53] and are shown in Table 6.

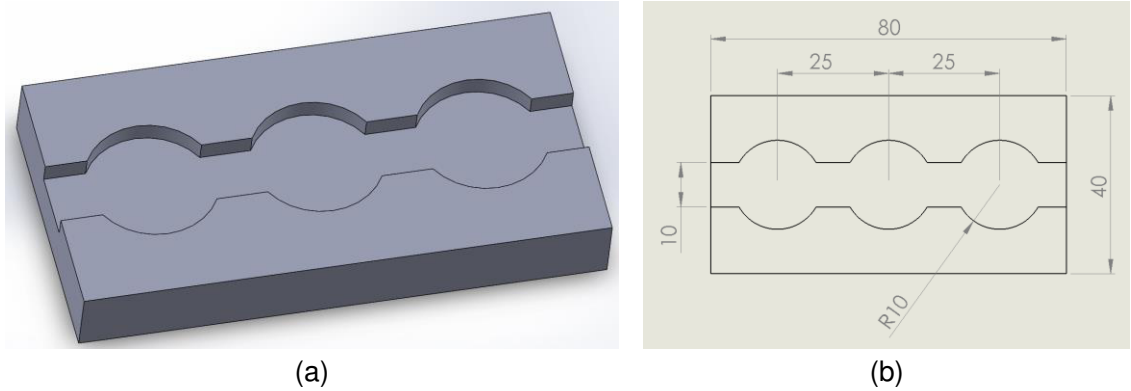


Figure 14 – (a) CAD model of the example part and (b) drawing of the example part including the dimensions.

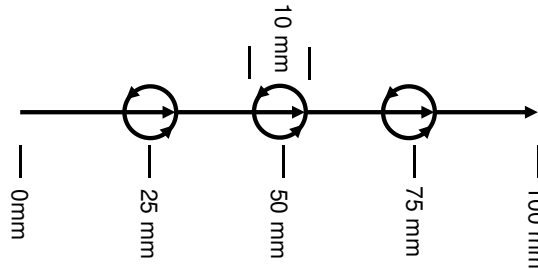


Figure 15 – Tool path of the example part.

Table 6 – Cutting parameters of the example part.

Cutting Parameter	Value
Rotational speed N	6250 1/min
Feed f_z	0.08 mm
Teeth number z	3
Feed rate v_f	25 mm/s

A simulated OPC UA controller server was implemented and hosted on a Raspberry Pi.

The x- and y-position of the tool were simulated on this server and read by OPC UA clients.

The x- and y-position of the tool was updated every 10 ms by the simulated controller server. Further, eight additional controller data points were read to simulate the case of reading ten data points in total. The set-up of the use case is shown in Figure 16. The edge device was connected to the simulated controller server hosted on a Raspberry Pi, this connected by an Ethernet wire. Additionally, a vibration sensor was connected to the edge device with an M8 connector wire. Two out of four Raspberry Pi devices of the cluster were used for this simulation. One Raspberry Pi hosted the controller server, one other Raspberry Pi hosted a local MQTT broker (Eclipse Mosquitto [54]). The edge device was connected to the MQTT broker via Wi-Fi. Different conditions of the edge device were tested. Firstly, the edge device exclusively read the controller data points and streamed them to the MQTT broker. Secondly, vibration data was captured, parsed and sent to the MQTT broker every ten seconds in addition to streaming the controller data points. In the last case, an FFT was executed on the edge device in addition to capturing vibration data every 10 seconds and streaming controller data to the MQTT broker.

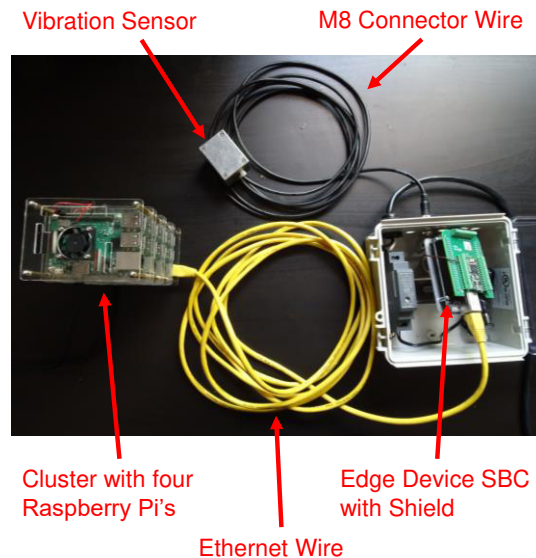


Figure 16 – Set-up of the use case trajectory monitoring.

3.3.2 *Edge Monitoring Architecture Delay Performance*

Measurement delays were evaluated for the direct and indirect edge monitoring architecture. Determining the maximum expected delay is important to design systems with time-critical sensor measurements and shows the limitations of the system. In this study, the delay was the time elapsed from when a controller data point in the controller server changes and this change was recognized on the edge device. The value change of a controller data point was not recognized by the edge device immediately for both architectures. For the direct edge monitoring architecture, Figure 17 shows the timeline of requesting controller data points from an edge device through direct machine connection. The controller data points were queried with a request frequency f_{req} and corresponding request interval ΔT_{req} . The red line on the bottom of Figure 17 illustrates the value of the trigger data point. The trigger data point is the controller data point wherein sensor data is initiated if the value of this data point falls within a certain range. For instance, this could be the G-code line identifier of the executed NC program. The lower level of the red line means that the trigger value has not yet been reached, whereas the higher level means that the trigger value has been reached. The recognized trigger value of the controller data point on the edge device is indicated by the upper red line in Figure 17. The delay ΔT is the time difference between change of the trigger data point, denoted by T_1 , and the recognition of the change on the edge device, denoted by T_2 .

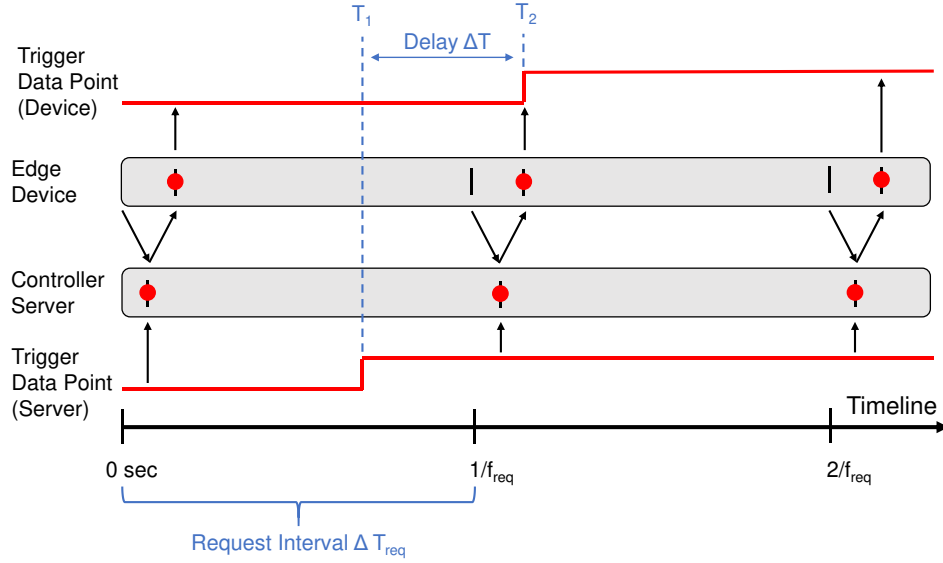


Figure 17 – Timeline requesting controller data points for direct edge monitoring architecture.

Figure 18 shows the timeline of requesting controller data points for the direct edge monitoring architecture with the maximum expected delay. The maximum expected delay ΔT_{max} occurs if the value of the controller data point changes directly after the controller client reads the previous value. Theoretically, the maximum delay is the sum of the request interval ΔT_{req} and the communication time from the controller server to the controller client. However, the last-mentioned component is difficult to measure since the clocks of the controller server and the edge device are not synchronized. For a conservative estimation, the communication time ΔT_c between the client and the controller server is added to the request interval ΔT_{req} to estimate the maximum expected delay ΔT_{max} . Thus, the maximum expected delay for the direct edge monitoring architecture can be calculated by:

$$\Delta T_{max} = \Delta T_{req} + \Delta T_c = \frac{1}{f_{req}} + \Delta T_c \quad (2)$$

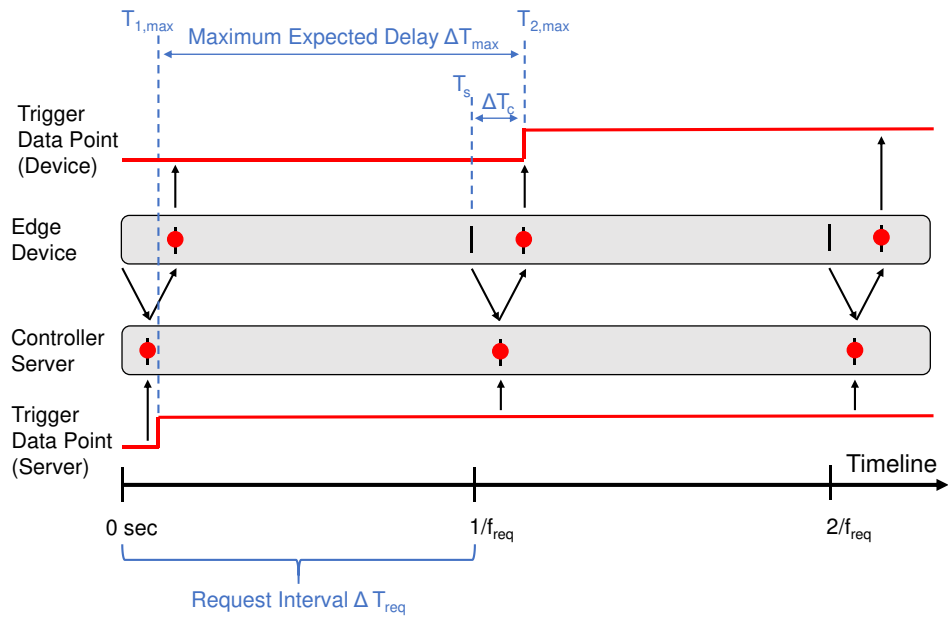


Figure 18 – Timeline requesting controller data points for direct edge monitoring architecture with maximum expected delay.

A Node-RED flow was implemented to measure the response time of the controller server. The measurement principle is shown in Figure 19. First, the current time T_q is stored. Then, n data points are requested. After receiving exactly n data points, the current time T_r is stored again. The communication time for the n data points can be calculated based on the two timestamps. The number of requested data points is increased from 10 to 50 in steps of 10. The response time was measured 200 times in five-second intervals for each step.

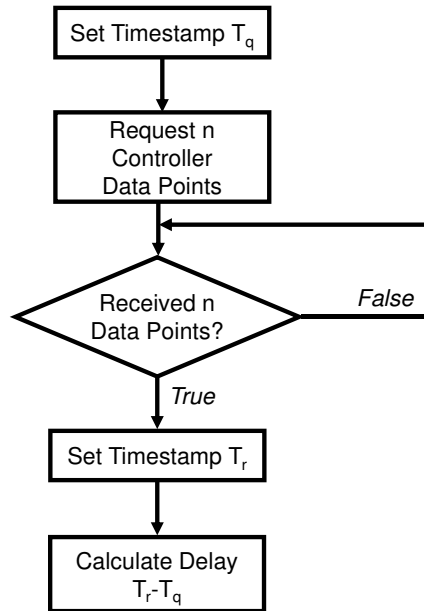


Figure 19 – Measurement principle response time of controller server.

For the indirect edge monitoring architecture, Figure 20 shows the timeline with the maximum expected delay if three different controller data points are streamed to an MQTT broker and one of these three controller data points is used as a trigger to start capturing sensor data with the edge device. The trigger controller data point is marked red in Figure 20. The graph of the trigger controller data point at the bottom of Figure 20 changes its state directly after the trigger controller data point was read by the controller client to determine the maximum expected delay. In comparison to the direct edge monitoring architecture, the controller client is hosted on a controller gateway and not directly on the edge device. After the request interval ΔT_{req} , the data point is read again and the change is recognized by the controller client. Unlike the direct controller connection, the request interval ΔT_{req} might increase due to the workload on the gateway. For high request frequencies and a high number of data points queried at once, the data point rate (DPR) is less than 100 %. The request interval ΔT_{req} is increased in this case. Therefore, the request interval in this case is determined by the reciprocal of the product of request frequency and

data point rate. The communication time ΔT_c is determined in the same way as described for the direct edge monitoring architecture, assuming that the trigger data point is queried before the additional data points. Additionally, sending and receiving data points via MQTT introduces a delay ΔT_{mqtt} . This delay can be split in the delay sending the message from the controller gateway to the broker ΔT_b and the delay sending the message from the broker to the edge device ΔT_e . The maximum expected delay can be determined by:

$$\Delta T_{max} = \Delta T_{req} + \Delta T_c + \Delta T_{mqtt} = \frac{1}{f_{req} * DPR} + \Delta T_c + \Delta T_b + \Delta T_e \quad (3)$$

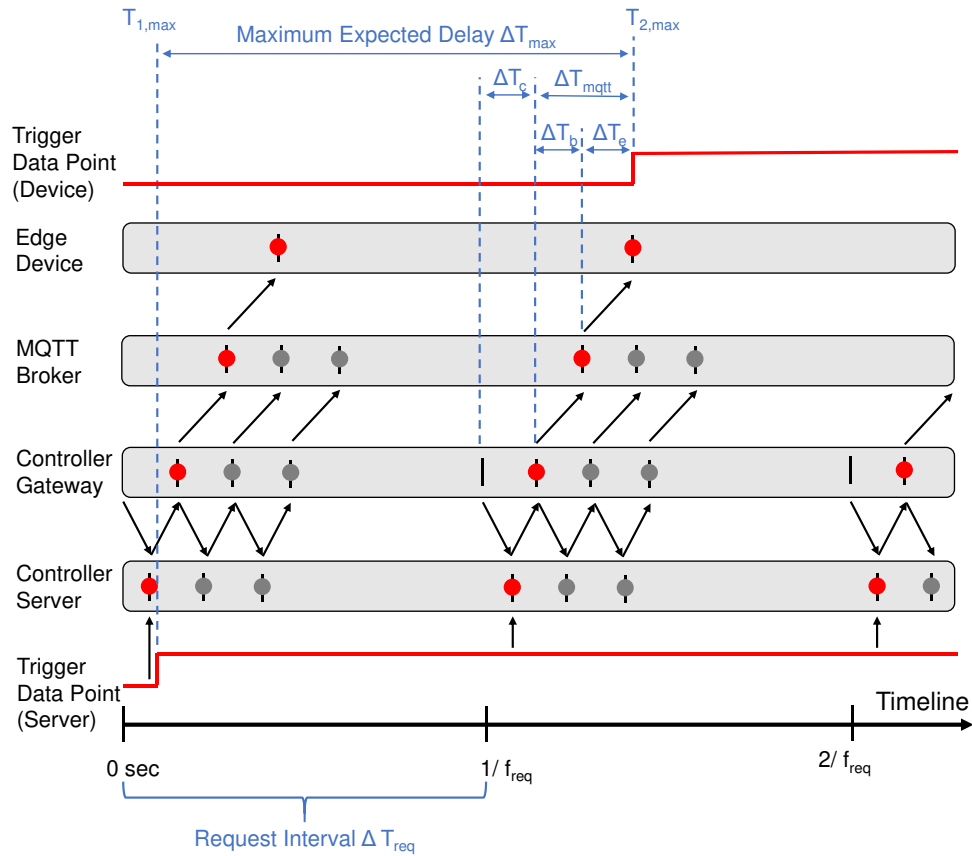


Figure 20 – Timeline requesting controller data points for indirect edge monitoring architecture with maximum expected delay.

Ferrari et al. proposed a method to measure delays of sending and receiving MQTT messages with Node-RED [55, 56], depicted in Figure 21. A test message is sent from one device to an MQTT broker. The sending time T_{s1} is stored. A second device subscribes to this test message and timestamps the time T_{r2} when the message is received. After receiving the message, a delay of five seconds is introduced to make sure that receiving and sending a message do not interfere with each other. After this delay, the test message is published to the MQTT broker again by the second device and the sending time T_{s2} is stored. The first device is subscribed to the test message and the receiving time T_{r1} is stored as well. Five seconds after publishing the test message to the MQTT broker, the second device sends the time difference between receiving the first message and sending the second message to the first device via the MQTT broker. The first device then calculates the delay of the roundtrip of the test message by:

$$\text{Roundtrip Delay} = (T_{r1} - T_{s1}) - (T_{s2} - T_{r2}) \quad (4)$$

The delay of the roundtrip is stored in a file on the first device. After the end of the experiments, the file listing all the measured delays can be pulled from the device. The average one-way delay can be calculated by:

$$\text{Oneway Delay} = \frac{\text{Roundtrip Delay}}{2} = \frac{(T_{r1} - T_{s1}) - (T_{s2} - T_{r2})}{2} \quad (5)$$

The clocks of the two different devices do not have to be synchronized, because only time differences on one device are compared, not absolute time values. The measurement principle shown in Figure 21 is implemented in Node-RED.

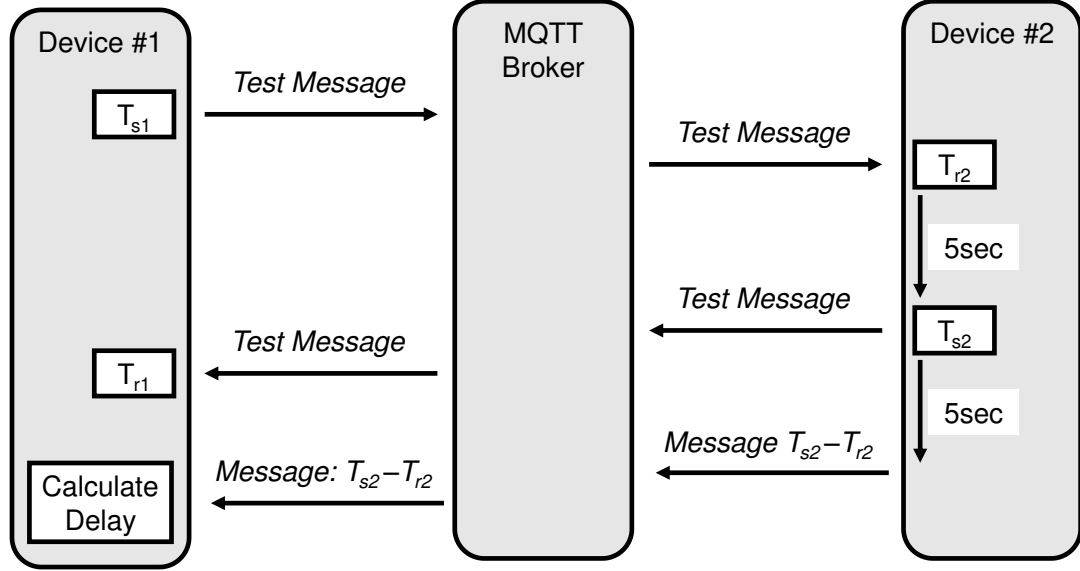


Figure 21 – Measurement principle of MQTT delay.

According to Equation 3, the delay introduced by MQTT ΔT_{mqtt} can be split into the delay sending the message from the controller gateway to the broker ΔT_b and the delay sending the message from the broker to the edge device ΔT_e . If the BBB Rev C is used as a controller gateway, the delay introduced by MQTT ΔT_{mqtt} is simply the one-way delay, since the BBB Rev C is utilized as the edge device SBC. However, if a Raspberry Pi or a BBB wireless is used as a gateway, the two delays ΔT_b and ΔT_e have to be determined separately and added together. Measuring these two components by themselves is difficult since the clocks on the different devices and the MQTT broker are not synchronized. Therefore, the delay of sending one controller data point from the controller gateway to the edge device is estimated by adding half the one-way delay of the Raspberry Pi or the BBB wireless $\Delta T_{oneway1}$ and the BBB Rev C $\Delta T_{oneway2}$ by:

$$\Delta T_{mqtt} = \Delta T_b + \Delta T_e = \frac{\Delta T_{oneway1} + \Delta T_{oneway2}}{2} \quad (6)$$

The adjusted parameters for measuring the MQTT delays are summarized in Table 7. The MQTT delay was measured for the three different SBCs. Two different MQTT broker locations have been tested, the framework Rabbit MQ hosted on an Amazon Web Service (AWS) instance located in North Virginia and the framework Eclipse Mosquitto hosted on a Raspberry Pi 3 Model B+ in the local network. The test message represents sending a single controller data point every 30 seconds. The message size is 187 bytes. The test message is given in Appendix A. To get comparable data, each experiment lasted for one day.

Table 7 – Adjusted parameter for measuring MQTT delay.

Parameter	Types
Controller gateway hardware	BBB Rev C
	BBB wireless
	Raspberry Pi 3 Model B+
Broker location	Broker hosted on an AWS instance
	Broker hosted on a Raspberry Pi 3 Model B+ in the local network

Anomaly detection of a machine is often done by equipping machines with external sensors (e.g. accelerometer, force sensor, temperature sensor) [33–35]. The impact of the maximum expected delays on machine health monitoring is illustrated for the use case of triggering sensor data acquisition by the G-code line number of a specific cut. If the data acquisition is triggered too late, the cut might be already finished. Therefore, the traveling distance of the tool without capturing sensor data was determined by considering the maximum expected delay. If this travel distance of the tool is bigger than the part length, the monitoring system is not reliably able to capture the sensor data during the cut. The

initial situation of this use case is shown in Figure 22. The milling tool cuts into a part on a linear path. The milling tool starts 0.1 times the diameter of the tool away from the part.

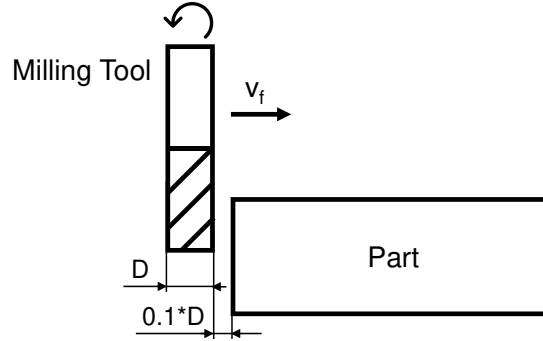


Figure 22 – Initial situation of the use case cut monitoring.

The same milling conditions as in the use case trajectory monitoring are used for this use case. A 10 mm endmill tool is used and the feed v_f is 25 mm/s. The different steps during the cut are shown in Figure 23. The tool starts 1 mm in front of the part. In case the delay of the triggering data point is short enough, the sensor measurement starts already before the complete tool enters the part. In case the delay of the triggering data point is more than the time the tool needs to enter the part, the tool already travels a distance L_{in} in the part without monitored by sensor data. In both cases, the position of the tool in the part for the maximum expected delay can be calculated by:

$$L_{in} = \Delta T_{max} * v_f - 0.1 * D \quad (7)$$

The length of the cut while capturing sensor data depends on the time needed to capture sensor data $t_{capture}$. The travel length of the tool during this time can be calculated by:

$$L_{capture} = t_{capture} * v_f \quad (8)$$

The total needed length for reliable capturing sensor data during the cut can be calculated by:

$$L_{part} = L_{capture} + L_{in} = (\Delta T_{max} + t_{capture}) * v_f - 0.1 * D \quad (9)$$

The part length depends on the maximum expected delay, the time needed to capture data, the feed of the tool and the starting position of the tool. The minimum length of the part can be determined by this analysis if the time for capturing data $t_{capture}$ is known. Since this time is not defined in this general analysis, the travel distance L_{in} considering the maximum expected delay is compared for the direct and indirect architecture in this study.

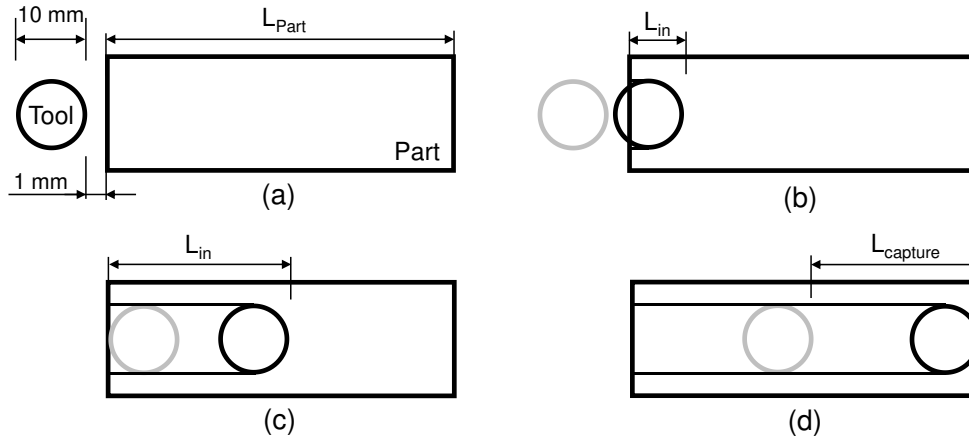


Figure 23 – Steps during the cutting operation for the use case cut monitoring with (a) the initial situation, (b) the tool enters the part, (c) the tool is in the part without capturing data and (d) the tool is in the part with capturing data.

3.4 Assumptions

One central assumption of the present framework is that the performance of the machine controller server is independent of the machining task or the machine status. To be more precise, the communication time between controller server and client is not influenced by

the machine status. Another assumption is that small changes in the request interval are neglected in this study. To determine the maximum expected delay of the direct edge monitoring architecture, it is assumed that the request interval is consistent. Additionally, the request interval can be modified by the data point rate according to Equation 3 for the indirect edge monitoring architecture. According to Figure 18 and Figure 20, the next assumption is that no significant delay is introduced by the controller for updating the controller server. It is assumed that as soon as the value of a data point changes in the controller, the value in the controller server is updated as well. One assumption made to determine the delay introduced by the MQTT communication is that the network traffic during one day does not change significantly at different business days. Therefore, the runtime of the experiments to determine the delay introduced by MQTT communication is 24 hours. The comparison of the MQTT communication time for different devices is possible although the communication time might be determined at different business days. The last assumption is that the one-way delay of MQTT communication can be determined by half of the roundtrip delay. Additionally, the communication time from the publisher to the broker and from the broker to the subscriber is half the one-way delay.

3.5 Results

3.5.1 Edge Monitoring Architecture Computational Performance

Both architectures were analyzed regarding their performance including capturing vibration data, edge analytics and streaming controller data points to an MQTT broker. The measurement sub-test of the direct edge monitoring architecture was used to determine maximum sampling rate and sample size of the edge device MC. The maximum possible

sampling rate of the edge device MC was determined as 27.8 kHz. The maximum sampling rate depends on the ADC speed of the edge device MC. The maximum sample size was determined as 25,239. The maximum sample size depends on the available memory space of the edge device MC and on the implementation of the measurement code loaded on the edge device MC. For the following test cases, the sampling interval was set to 0.00019s (sampling rate 5,263 Hz) and the sample size was set to 4200.

In the analytics sub-test of the direct edge monitoring architecture, the memory usage and the execution time of the incrementing and the FFT algorithm was determined while sending raw vibration data to the MQTT broker and reading three controller data points. The average execution time and the memory usage of the incrementing and the FFT algorithm are shown in Table 8. In the analytics sub-test, limitations of the edge device were not identified. The execution time of an algorithm can be a limit for certain applications. However, time limits were not defined in this general analysis. The memory usage of both algorithms did not reach the limitations of the edge device SBC. The available RAM of the edge device SBC is 512 MB.

Table 8 – Results of analytics sub-test.

	Incrementing Algorithm	FFT Algorithm
Memory (RAM) Usage	666 Bytes	96,336 Bytes
Average Execution Time	4061.6 ms	4251.9 ms

In the streaming sub-test, the edge device SBC streams controller data points to the MQTT broker while sending raw vibration data to the MQTT broker. The data point rate was measured for different numbers of requested data points and request frequencies. The average rates of data points of the streaming sub-test are shown Figure 24. Figure 24 shows

that for all combination of request frequency f_{req} and numbers of data points n , the average data point rate is clearly below 100 percent. The desired data rate was not reached. This means that streaming controller data points at the same time as capturing vibration data does not work reliably. Some controller data points are missed and the effective request frequency is below the required request frequency.

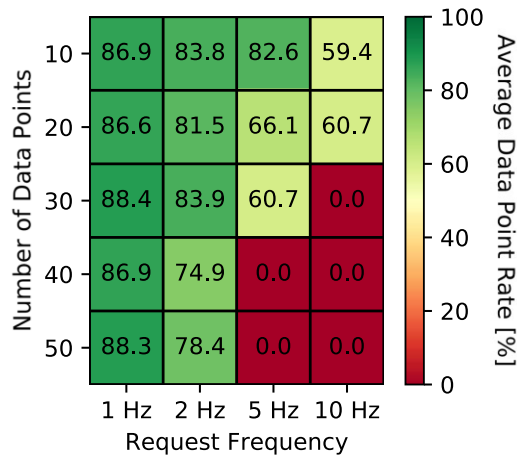


Figure 24 – Average data point rate streaming sub-test.

Figure 25 shows the time differences between received controller data points of the same node ID on the edge device for a request frequency of 1 Hz and the number of different requested data points, i.e., 20. In an optimal case, the time differences should be exactly one second. However, the time difference is approximately three seconds in every eighth data point. This phenomenon corresponds to times in which vibration data is captured, parsed and send to the MQTT broker. The vibration data is captured every ten seconds.

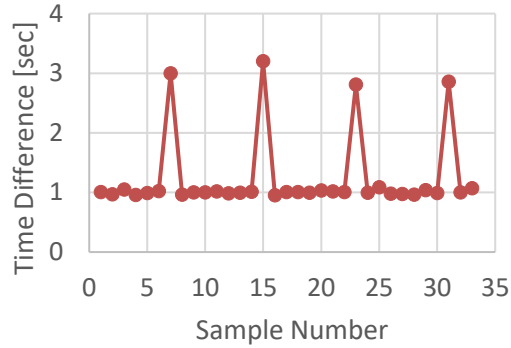


Figure 25 – Time difference between received controller data points of the streaming sub-test.

The streaming sub-test already showed the limitation of streaming controller data points and capturing vibration data at the same time. In the overall system test, edge analytics algorithms are executed in addition to streaming controller data points and vibration data to the MQTT broker. The average rates of data points for using the incrementing algorithm are shown in Figure 26(a) and the average data point rates while using the FFT algorithm are shown in Figure 26(b). The average data point rates are significantly lower in comparison to the streaming sub-test. This means that executing additional data analysis algorithms on the edge device SBC while streaming controller data points to the MQTT broker reduces the average data point rate. Figure 26(a) and Figure 26(b) do not show a clear difference between the two kinds of algorithms. This means that the memory usage of a data analysis algorithm does not influence the average data point rate significantly.

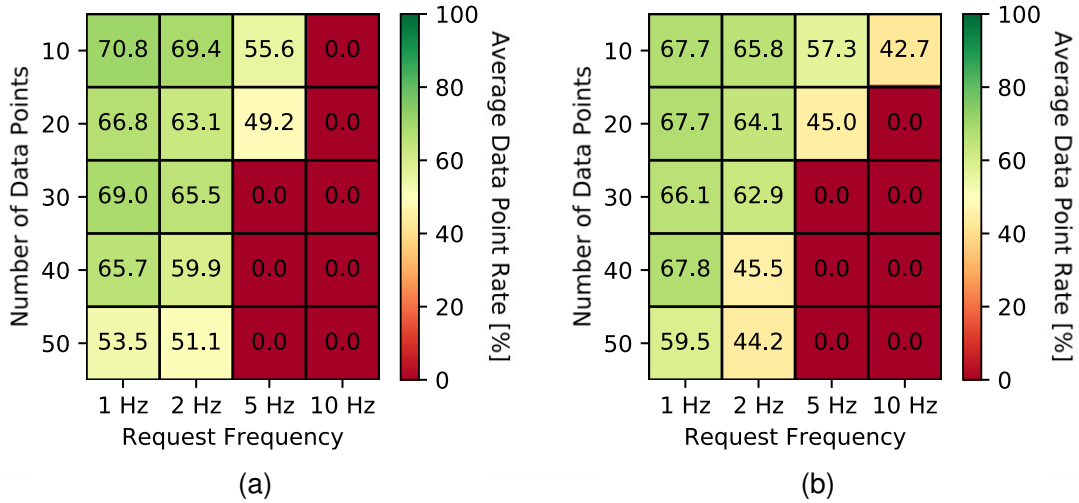


Figure 26 – Average data point rate overall system test with (a) incrementing algorithm and (b) FFT algorithm.

Figure 27 shows the time difference between the received controller data points of one node ID on the edge device for the streaming sub test, the overall system test and if the controller data points are exclusively streamed to the MQTT broker without additional workload on the edge device SBC. The request frequency was set to 1 Hz and the number of data points was 20. The peaks of the time differences for the overall system test are higher than the peaks of the streaming sub-test. Additionally, it can be seen that the time difference is nearly one second if controller data points are streamed exclusively without any data analysis algorithm and without capturing vibration data.

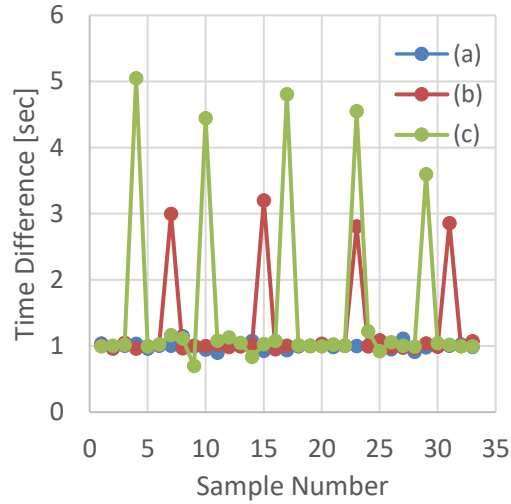


Figure 27 – Comparison of time difference between received data points for (a) exclusively streaming controller data points, (b) additionally capturing vibration data and (c) additionally executing FFT.

Figure 28 shows the timeline of the received controller data points of one node ID. Figure 28 illustrates that the gap of receiving controller data points occurs periodically every 10 seconds. The gap is bigger for the overall system test in comparison to the streaming sub-test.

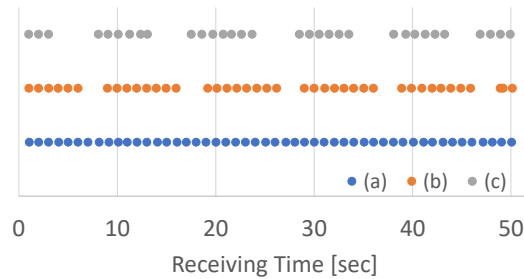


Figure 28 – Timeline of received controller data points for (a) exclusively streaming controller data points, (b) additionally capturing vibration data and (c) additionally executing FFT.

The results of Figure 27 and Figure 28 motivate the usage of the indirect edge monitoring architecture if controller data points are streamed to an MQTT broker. The figures show that the time difference between the received data points is consistent around one second

for exclusively streaming controller data points. Exclusively streaming controller data points to an MQTT broker with a controller gateway seems more reliable and consistent. In the gateway sub-test, the controller data points are exclusively streamed to the MQTT broker by an SBC working as a controller gateway. Figure 29(a) shows the average data point rate of the BBB Rev C working as a controller gateway. The average data point rate is significantly higher than in the previous tests for the direct edge monitoring architecture. The average data point rate for 1 Hz is 100 %, independent of the number of data points n . Closer to the limits, the average data point rate is below 100 %. The limits are marked red in Figure 29(a). However, the rates close to the limits are still significantly higher than the rate of the previous tests. Figure 29(b) shows the average data point rate of the Raspberry Pi. In comparison to the BBB Rev C, the average data point rate is nearly 100 % for all request frequencies and number of data points despite the red area where the pulling of data points was not possible. The reason for the higher rate is most likely the higher computational power of the Raspberry Pi (c.f. Table 1). The area in which pulling data points is not possible is the same as for the BBB Rev C. One of multiple explanations for this is the fact that the Node-RED package for reading controller data points is limited in its speed. Another explanation is that the controller server is limited in its speed. A last explanation could be that the communication time between controller server and client for pulling many data points with a high request frequency is too long and the communication takes longer than the time between the different requests. Regarding the average data point rate, the Raspberry Pi might be the better choice to pull controller data points with a high request frequency and with a high number of data points. If the request frequency is lower and the number of data points is not too high, the BBB Rev C is sufficient as well.

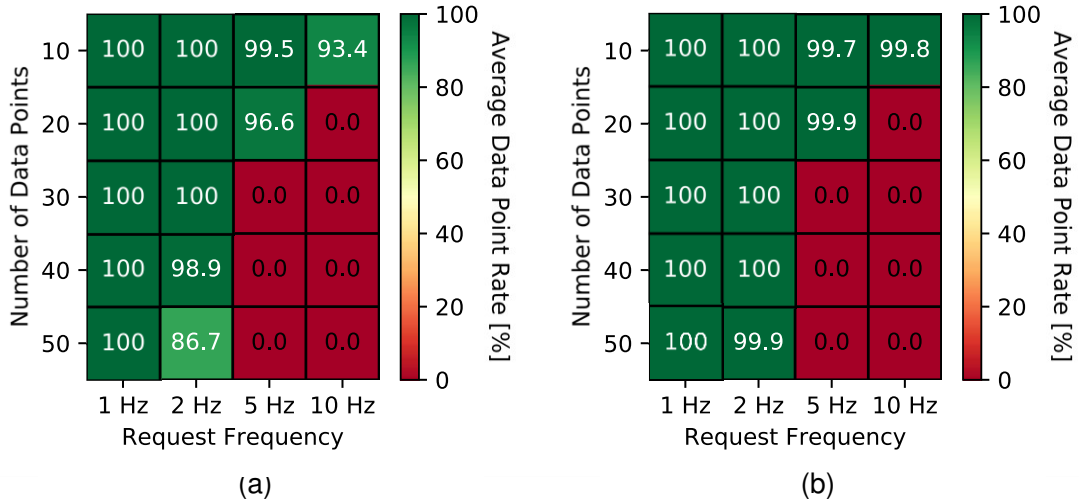


Figure 29 – Average data point rate gateway sub-test of (a) the BBB Rev C and (b) the Raspberry Pi 3 Model B+ as controller gateways.

The impact of the identified limitations regarding the computational performance of the direct and indirect edge monitoring architectures are illustrated for the use case trajectory monitoring of a tool path. The result of the simulation is shown in Figure 30. If the controller data points are streamed exclusively to the broker by the edge device, the shape of the cut is represented by the measured data points. If vibration data is captured additionally every 10 seconds, the prescribed toolpath form is difficult to recognize while the vibration data is captured and parsed. If an FFT is executed in addition to the other tasks, the tool path is not represented by the captured x- and y-position of the tool. For example, in this case, only one data point was read for the middle circular feature in Figure 30 (c). Thus, if controller data points must also be streamed to an MQTT broker, the architecture with an additional controller gateway is the better choice.

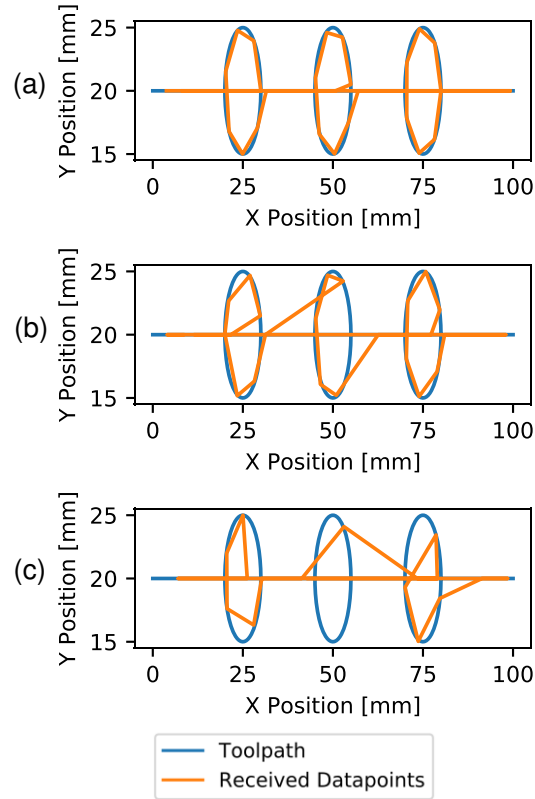


Figure 30 – Result of the use case trajectory monitoring for (a) exclusively streaming controller data points, (b) additionally capturing vibration data every 10 seconds, and (c) additionally executing a FFT algorithm.

3.5.2 Edge Monitoring Architecture Delay Performance

The maximum expected delay for the direct and indirect edge monitoring algorithm was determined with two additional experiments. In the first experiment, the communication time between the SBCs and the controller server was measured. In the second experiment, the delay introduced by MQTT communication was measured. Based on the two experiments, the maximum expected delay for the direct and indirect edge monitoring architecture was determined. A use case illustrates the importance of the maximum expected delay consideration for capturing sensor data during a specific cut of a machine.

The results of measuring the communication time between the controller server and client are shown in Figure 31. Figure 31 shows the response time versus the number of requested data points for the BBB Rev C and the Raspberry Pi. The distribution of the response time is assumed as normal distributed. The error bars represent the width of six standard deviations around the average response time of the controller server. The average response time increased linearly with the number of requested data points. The measured average response time with the Raspberry Pi was around one third smaller than the average response time with the BBB Rev C. The reason is most likely the higher computational power of the Raspberry Pi in comparison to the BBB Rev C (c.f. Table 1). As in Figure 18, the maximum expected delay associated with requesting a single controller data point is needed to calculate the maximum expected delay. This was done by linear regression of upper limit of the error bars which are shown in Figure 31. Based on this regression, the maximum expected communication time for a single controller data point ΔT_c with the BBB Rev C (utilized as edge device SBC) was 79.7 ms. The maximum communication time ΔT_c of the Raspberry Pi was 32.7 ms.

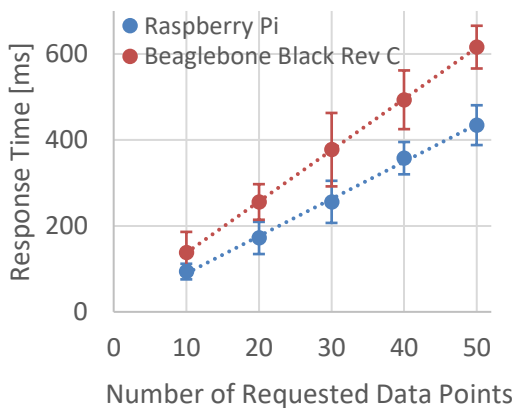


Figure 31 – Response time of controller server for different number of requested data points with error bars indicating the width of six standard deviations.

Figure 32 shows the maximum expected delay for different request frequencies f_{req} using the proposed direct edge monitoring architecture. The maximum expected delay of the direct edge monitoring algorithm in this case was dependent on the request interval ΔT_{req} and the communication time between controller server and client ΔT_c . From these results, the main component of the maximum expected delay was the request interval ΔT_{req} since the communication time ΔT_c was significantly smaller. However, when the request frequency was set to 10 Hz, the communication time increased the maximum expected delay by 80 percent. The maximum expected delay for the direct edge monitoring architecture refers to the measurement sub-test and the analytics sub-test of Table 4, it does not refer to the streaming sub-test and the overall system test, since requesting controller data points was not reliably possible.

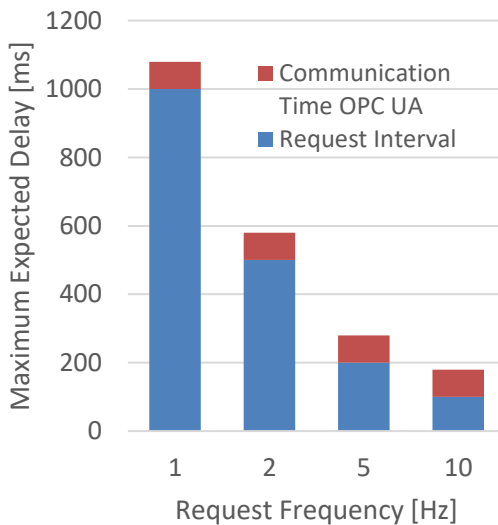


Figure 32 – Maximum expected delay for direct edge monitoring architecture.

For the indirect edge monitoring architecture, a second experiment determined the maximum expected delay by determining that attributed to the MQTT communication. Figure 33 shows the measurement of the roundtrip delay of the test message taken with

two Raspberry Pi's. The AWS broker was used for this experiment. Figure 33 shows that the roundtrip delay varies in a wide range during the runtime of the experiment.

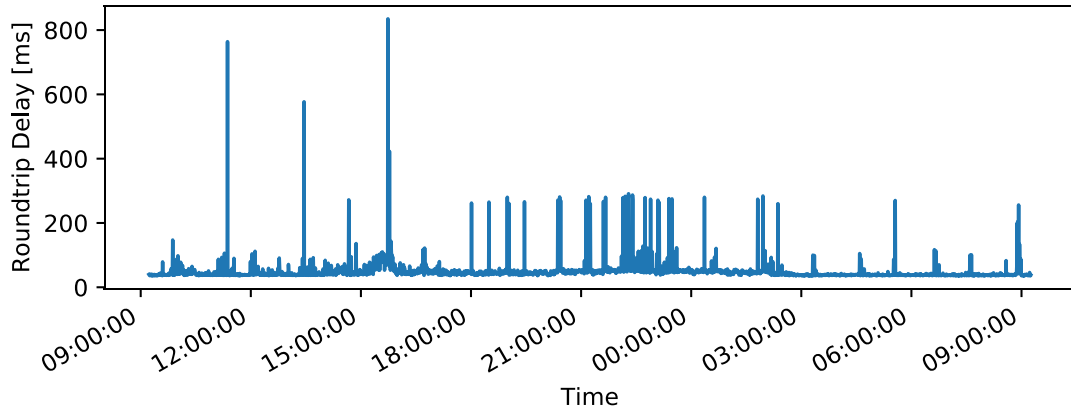


Figure 33 – Measured roundtrip delay for two Raspberry Pi's and AWS broker.

Figure 34 shows the roundtrip delay in a histogram. The roundtrip delay was not normally distributed and the average roundtrip delay was 49.35 ms. Further, the maximum roundtrip delay is about 820 ms and there is clear grouping of delays also around 280 ms. The histogram shows that it is difficult to predict the delay of sending an MQTT message due to the high variation of the values.

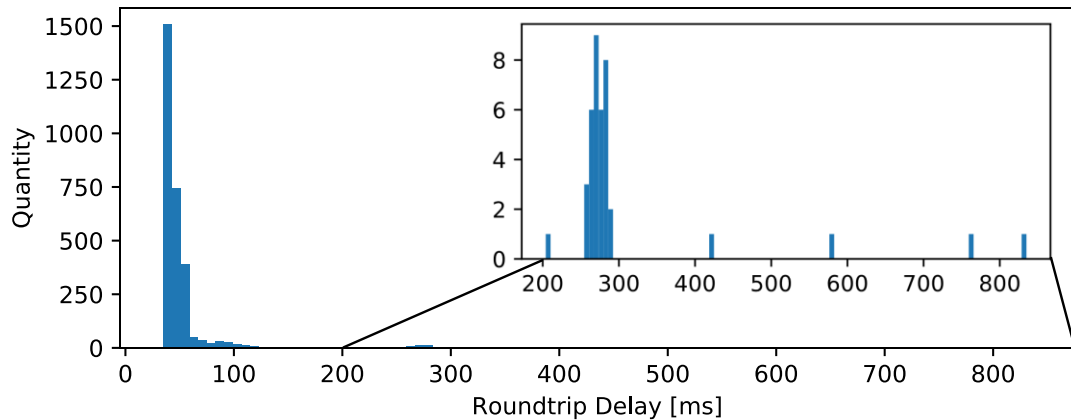


Figure 34 – Histogram of roundtrip delay for two Raspberry Pi's and AWS broker.

The diagram in Figure 35 shows the average one-way delay for different SBCs and broker locations. The average delay of the BBB wireless was about four times higher than the average delay of the BBB Rev C and the Raspberry Pi for both broker locations. The Raspberry Pi experienced the lowest average delay of the three SBCs. In addition, the delay for all three SBCs was lower in the case of a locally implemented local broker instead of the AWS broker. However, the difference is less significant in comparison to changing the hardware.

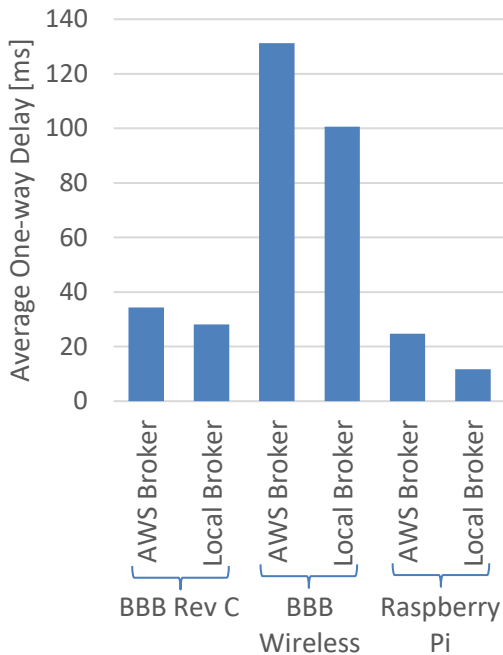


Figure 35 – Comparison average one-way delay for different broker locations.

As Figure 33 and Figure 34 show, the average delay might not be the best parameter to identify the maximum expected delay of the architecture since the delay varies in a wide range. Additionally, the delay is not normally distributed. The confidence level accounts for this issue. The confidence level describes for which delay value a certain percentage of measured values have been below this delay value. Figure 36 shows the one-way delay for

the confidence levels of 80%, 90%, 95%, 99% and 99.5%. The one-way delays of the different SBCs showed the same behavior as the average delay measure. The one-way delay of the BBB wireless was the highest, the one-way delay of the Raspberry Pi was the lowest for all confidence levels of the three SBCs. Further, the one-way delay of the BBB wireless and the Raspberry Pi was lower if the broker was locally-based. In contrast, the BBB Rev C showed a different behavior. For a confidence level of 80%, the one-way delay of the local broker is 26.5 ms and therefore smaller than the delay of the AWS broker which is 30 ms. Starting at 90% confidence level, the one-way delay of the AWS broker was smaller than the delay of the local broker. For a 90% confidence level, the one-way delay of the AWS broker is 34.5 ms and the delay of the local broker is 36.5 ms. This means that although the average delay of the local broker was smaller, the scattering of the high one-way delay values was bigger for the local broker than for the AWS broker.

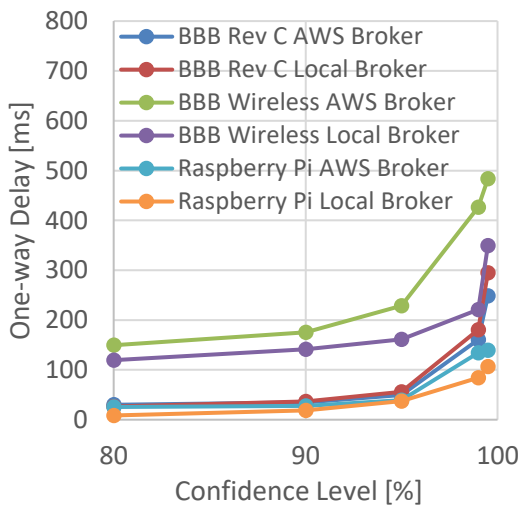


Figure 36 – Comparison confidence level of one-way delay for different broker locations and devices.

The confidence level shown in Figure 36 can be used to identify the limitations for triggering time-critical measurements by using a controller gateway. Depending on the

confidence level, on the SBC and on the broker location, the expected maximum one-way delay of sending an MQTT message can be used to estimate the maximum expected delay. Since the BBB wireless showed a significant higher average one-way delay and one-way delay for different confidence levels than the other two SBCs, the BBB wireless was not used for the other experiments.

Figure 37 shows the dependence of the maximum expected delay on the request frequency in the case of requesting 10 data points at once, using a Raspberry Pi as controller gateway, using a confidence level of 99.5% for MQTT communication and using a local broker hosted on a Raspberry Pi. Figure 37 shows that the main impact of the maximum expected delay for frequencies below 5 Hz is the request interval. Additionally, for increasing request frequency, the relative contribution of the request interval decreased and the relative contributions due to MQTT delay and communication time between the controller server and client increased.

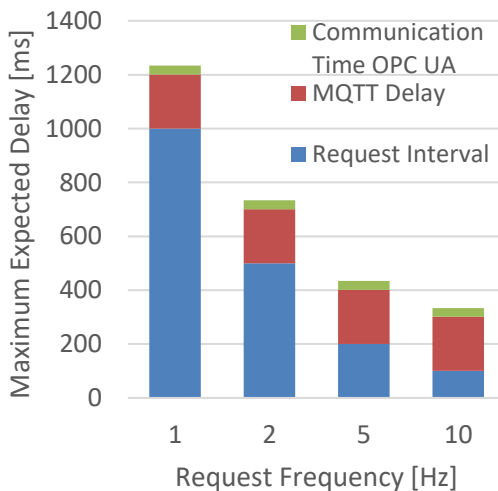


Figure 37 – Maximum expected delay indirect edge monitoring architecture for requesting 10 data points at once, Raspberry Pi as controller gateway, confidence level of 99 % and using local MQTT broker hosted on a Raspberry Pi.

The use case cut monitoring compares the travel distance L_{in} of the tool without capturing sensor data considering the maximum expected delay if the sensor data acquisition is triggered by the G-code line number of the NC program. Figure 38 compares the travel distance L_{in} for different request frequencies f_{req} by considering the maximum expected delay. For the indirect edge monitoring architecture, the conditions are the same as for Figure 37. The travel distance determined by the maximum expected delay of the direct edge monitoring architecture is lower than that using the indirect edge monitoring architecture. For a high request frequency, the difference in travel distance is significant. The part length of the direct edge monitoring architecture for the request frequency of 10 Hz is about one third of the part length of the indirect edge monitoring architecture.

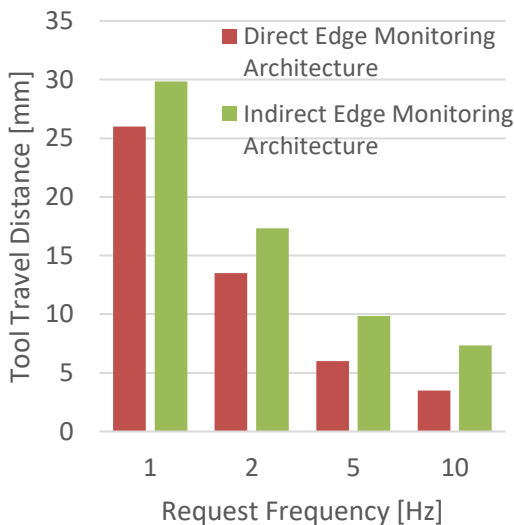


Figure 38 – Comparison between the two edge monitoring architectures for travel distance without monitoring vibration data depending on the request frequency.

The travel distance of the tool without capturing sensor data L_{in} can be reduced with some simple adjustments to the triggering process. One possibility is to move the starting position of the tool further away from the part. This way, the triggering data point is

received on the edge device in an earlier stage of the actual cut in the part. The disadvantage of this method is the loss of productivity since more time is needed for the cut if the tool starts not as close to the part. In addition, the data might need to be captured during a longer time period since the calculations are based on the maximum expected delay. On average, the delay is shorter, and it needs to be ensured that the data is not only captured while the tool has not entered the part yet. The total sampling time needs to be long enough. To further optimize, the capturing of vibration data could be triggered by a G-code line number before the actual cut. This would also increase the time before the cutting starts after triggering without reducing productivity of the machine.

3.6 Discussion

Both proposed architectures can be used in different situations. If streaming of controller data points to a broker is not necessary, connecting the edge device directly to a controller server is the most efficient and simplest solution. This direct edge monitoring architecture provides the smallest delay since no additional delays due to communication via MQTT are present. Additional edge analytics do not influence the performance of the edge device. If the edge analytics needs to be done within time constraints, the execution time of the algorithm needs to be measured to decide whether the computational power of the edge device is high enough to meet the requirements or whether the data needs to be analyzed in the cloud.

If streaming of controller data points to a broker is necessary for other monitoring aspects, a controller gateway should be used. The performance of the direct edge monitoring architecture already decreases when data is captured with external sensors in addition to

streaming the data points. The performance deteriorates further if data analytic algorithms are performed on the edge device additionally. The downside of using a controller gateway is the additional delay introduced by the MQTT connection. Among the three tested devices used as controller gateways, the Raspberry Pi has the best performance regarding the ability to stream a high number of data points with a high frequency to the broker and regarding the additional MQTT delay of sending controller data points via MQTT to the broker.

The request frequency highly influences the maximum expected delay for both proposed edge monitoring architectures. Therefore, the request frequency should be chosen as high as possible resulting in a smaller maximum expected delay. However, the computational performance of the devices needs to be considered to choose the request frequency. The maximum request frequency of the controller server needs to be considered as well. Based on the experiments described in this chapter, the limits of both architectures can be considered for choosing an appropriate architecture for specific use cases. The influence of the different architectures and parameters becomes apparent with the use case of trajectory monitoring and cut monitoring.

CHAPTER 4. HEALTH MONITORING OF ROLLING ELEMENT BEARINGS

In this chapter, an edge monitoring algorithm for the health condition of rolling element bearings is developed. The presented algorithm can be hosted on the edge device which can be integrated in the direct and indirect edge monitoring architectures previously described in chapter 3. The first subsection outlines the theoretical background of rolling element bearings. The second subsection describes the development of an effective edge monitoring algorithm. The third subsection introduces the experimental method testing the developed edge monitoring algorithm in praxis. The fourth part discusses the results of the experiment.

4.1 Theoretical Background Rolling Element Bearings

Failures of rolling element bearings frequently cause downtime of rotating machinery and therefore loss of productivity [57]. Monitoring the health condition of bearings reduces machine downtime since the bearing can be changed before the machine fails. Figure 39 depicts the structure of a typical rolling element bearing [58]. A rolling element bearing consists of four elements: inner race, outer race, cage and rolling elements. The pitch diameter is called d_p . The diameter of the rolling element itself is called d_B . Bearings have a contact angle θ .

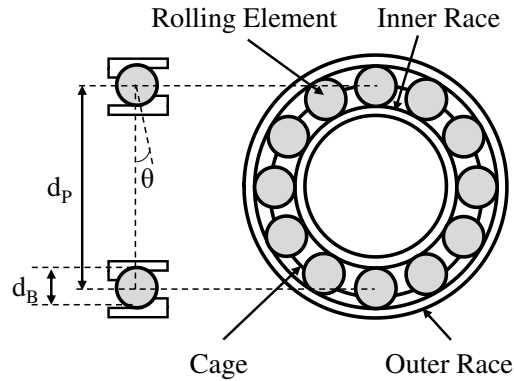


Figure 39 – Structure of a rolling element bearing [58].

Each bearing element can have a localized defect. Three typical defects of bearings are outer race defects, inner race defects and rolling element defects. Figure 40(a) shows an inner race defect, Figure 40(b) shows an outer race defect.



(a)



(b)

Figure 40 – Photo (a) inner race defect [59] and (b) outer race defect [60].

Each localized defect causes a certain vibration signal with peaks at a distinguished defect frequency and its harmonics if one of the races rotates [61]. Harmonics are peaks found at integer multiples of the defect frequency. The frequency of the defect signal can be determined by the characteristics of the bearing [61, 62]. Typically, the outer race of a bearing in machine tools is stationary and the inner race is rotating with the rotational speed

of the shaft. In this case, the generated frequency by an outer race defect can be calculated by:

$$BPFO = \frac{n_r}{2} \left(1 - \frac{d_B}{d_p} * \cos(\theta) \right) * \omega_{Spindle} \quad (10)$$

BPFO is short for ball pass frequency of the outer race. The number of rolling elements is called n_r and the rotational speed of the spindle is $\omega_{Spindle}$. The generated frequency of an inner race defect is called ball pass frequency of the inner race (BPFI) and can be determined by:

$$BPFI = \frac{n_r}{2} \left(1 + \frac{d_B}{d_p} * \cos(\theta) \right) * \omega_{Spindle} \quad (11)$$

The ball spin frequency (BSF) is generated by a rolling element defect and can be calculated by:

$$BSF = \frac{d_p}{2d_B} \left(1 - \left(\frac{d_B}{d_p} \right)^2 * \cos^2(\theta) \right) * \omega_{Spindle} \quad (12)$$

A vibration sensor can capture the vibration signal of a rotating bearing. The vibration signal in the time domain can be converted into the frequency domain by using a FFT [63]. The typical frequency spectrum of a bearing with an outer race defect is shown in Figure 41. The BPFO in Figure 41 is 236 Hz. The frequency spectrum shows peaks at this frequency and at its harmonics.

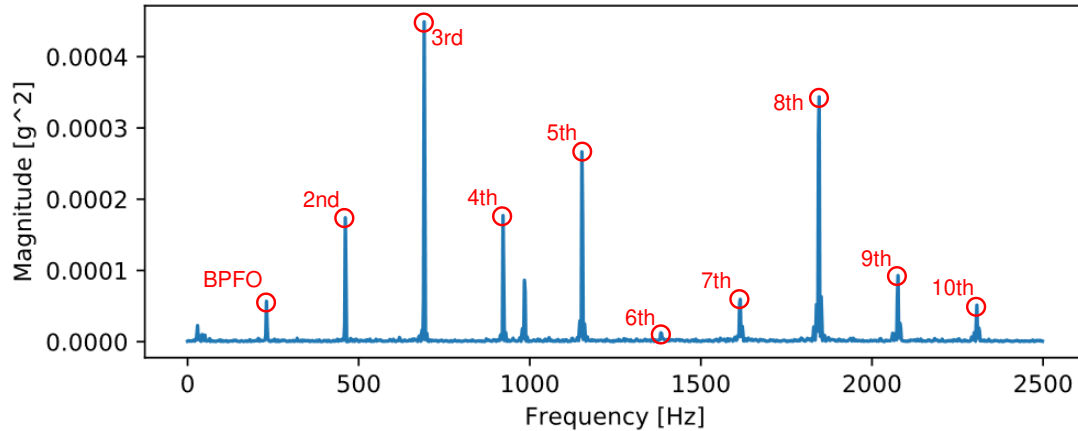


Figure 41 – Frequency spectrum of bearing with an outer race defect, the BPFO frequency and its harmonics are highlighted by red circles.

4.2 Development Edge Monitoring Algorithm

A test dataset with vibration data of test-to-failure experiments of bearings was used to develop a bearing health monitoring algorithm [64]. In the first part of this subchapter, the test dataset is described. In the second part, the developed analysis method is summarized. The third part discusses the results of this analysis. The fourth part explains the actual implementation of the edge monitoring algorithm. Finally, the assumptions of the method are listed.

4.2.1 Description of the Test Dataset

A sketch of the test rig is shown in Figure 42 [64]. Four Rexnord ZA-2115 double row bearings are used for the experiments. They are installed on a rotating shaft. The shaft rotates constantly at 2000 rpm. A radial load of 6000 lbs. is applied on bearing 2 and bearing 3. Accelerometers (PCB 353B33) are installed on each bearing housing to capture the vibration signal of the bearing. The sampling rate is 20 kHz. Each measurement contains 20,480 samples. Liu and Gryllias suspect that the actual sampling rate is

20.48 kHz instead of 20 kHz [65]. However, the potentially different sampling rate only changes slightly the location of the peaks in the frequency domain and does not have an impact on the results of this study. The acceleration was measured every 10 minutes during the runtime of the experiment. The data was collected with a NI DAQ Card 6062E.

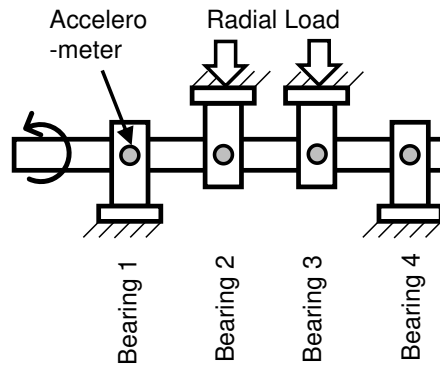


Figure 42 – Bearing test rig based on [64].

The data of three test-to-failure experiments is available. At the end of the first experiment, an inner race defect occurred in bearing 3. In bearing 4, an outer race and a roller element defect occurred. At the end of the second experiment in bearing 1 and at the end of the third experiment in bearing 3, outer race defects occurred.

4.2.2 Bearing Fault Analysis Method

To monitor the health condition of the bearings, different parameters can be used. Some parameters can be calculated from the raw vibration signal in the time domain, whereas others are calculated from the frequency spectrum of the signal, which is obtained by an FFT using Welch's method and a Hann window for de-noising the signal. In this analysis, typical statistical parameters were used. The statistical parameters were calculated in the

time and in the frequency domain. In addition, the magnitude of a potential bearing defect frequency was considered. The used statistical parameters are:

- Arithmetic mean
- Geometric mean
- Standard deviation
- Skewness
- Kurtosis
- Root mean square (RMS)
- Maximum value
- Crest factor

The equations to determine the different statistical parameters can be found in Appendix B. The potential defect magnitudes are determined in four steps. First, the theoretical defect frequency for outer race, inner race and roller element defects are calculated using the spindle rotational speed and the bearing characteristics. Based on the theoretical defect frequencies, the first ten harmonics of each defect frequency are computed by multiplying the defect frequency with integers between two and ten. The third step is to identify the highest peak in a window of $\pm 3.5\%$ around each defect frequency and harmonic. The last step is to calculate the average of the found peaks for each defect respectively.

The parameters are compared in a control chart [66]. The limits of the control chart are identified using the first 100 measurements. First, the arithmetic mean value x_{mean} and the standard deviation σ of the first 100 measurements are calculated. The upper control limit (UCL) depends on the scaling factor α and can be calculated by

$$UCL = x_{mean} + \alpha \cdot \sigma \quad (13)$$

The scaling factor α is typically set to three. For a normal distribution, the values stay within three times the standard deviation with a probability of 99.73% [67]. This is not the case for the measured values, indicating that the parameters of the measured data points do not follow a normal distribution. To reduce the number of false detections, a defect is only flagged if the parameter is above the UCL three times in a row. The different parameters have been compared against the described comparison method with respect to detection time. The detection time is the running time of the test rig up to moment the parameter is above the UCL three times in a row. A comparison of detection times exposes the sensitivity of the different parameters. The result for monitoring the magnitude of a potential outer race defect in the third dataset is shown in Figure 43. The orange horizontal line is the arithmetic mean value of the first 100 measurements. The green horizontal line is the UCL calculated according to Equation 13 with a scaling factor α of three. The purple crosses indicate data points which are above the upper limit. The data points exceed the upper limit several times during the runtime of the bearing before the parameter clearly increases at around 60,000 minutes. Therefore, a defect is only detected if the parameter is above the limits three times in row. The detection time is marked by the red vertical line. In this example, the defect is detected after 59,710 minutes.

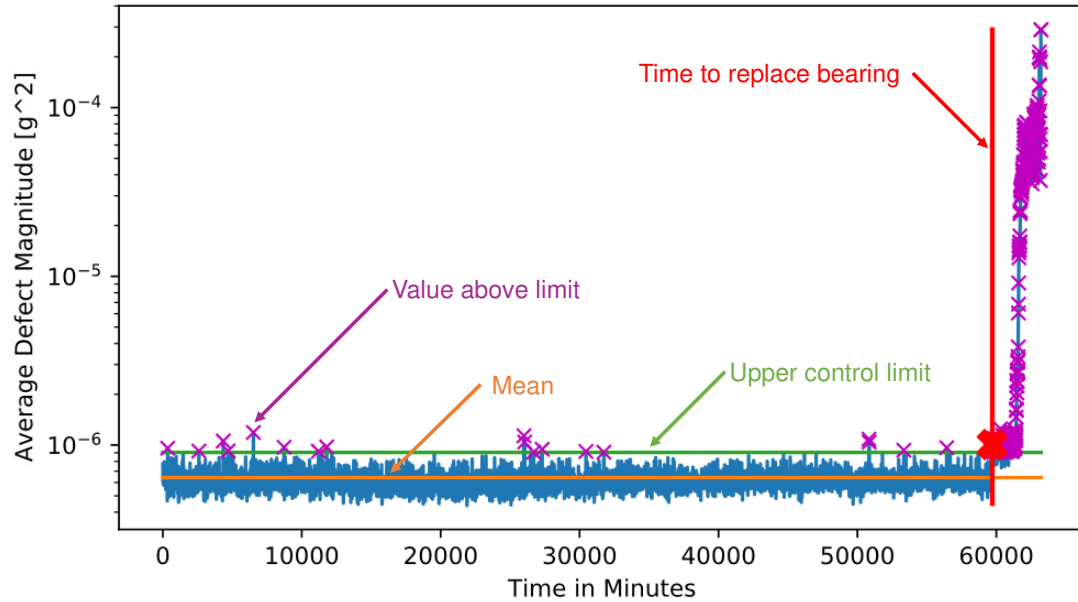


Figure 43 – Control chart for potential outer race defect magnitude of dataset 3 bearing 3.

The goal for this application is to identify parameters which clearly detect a bearing defect before the end of the runtime of the experiment. The detection should be in a reasonable range before the end of the experiment. Early detection is good since more time is available to replace the bearing. However, the detection should not be too early in the runtime of the experiment, since then the bearing is replaced although it is not needed yet.

4.2.3 Bearing Fault Analysis Result

Below, the parameter comparison results for the three datasets are described. For all bearings that exhibited a defect at the end of the experiment runtime, the detection time was determined. This is shown in Table 9 for the example of dataset 3 bearing 3. The runtime of this experiment was 63,240 minutes. The detection times for dataset 1 (bearing 3 and 4) and dataset 2 (bearing 1) can be found in Appendix C.

Table 9 – Detection time for dataset 3 bearing 3.

Parameter	Detection Time in Time Domain [min]	Detection Time in Frequency Domain [min]
Arithmetic Mean	No Detection	59,680
Standard Deviation	59,690	61,640
Skewness	61,610	61,620
Kurtosis	61,720	No Detection
Root Mean Square (RMS)	59,690	61,630
Maximum Value	61,960	45,570
Crest Factor	No Detection	No Detection
Geometric Mean		59,690
Potential Outer Race Defect Magnitude		59,710
Potential Inner Race Defect Magnitude		59,690
Potential Rolling Element Defect Magnitude		61,610

According to Table 9, the detection times for potential outer race defect magnitude and potential inner race defect magnitude are close together for dataset 3 bearing 3. However, only an outer race defect was visible after stopping the experiment and inspecting the bearing. The different defect frequencies overlap, resulting in an increase of potential inner race defect magnitude as well as outer race defect magnitude. The potential defect magnitude is determined by the maximum peak in a window of $\pm 3.5\%$ of the calculated defect frequency or harmonic. These windows can overlap for the different defects, as illustrated in Figure 44. The first three rows show the searching intervals for the maximum peak for BSF, BPFI and BPFO, respectively. In the last row, the overlap between the different intervals is shown. Overlap occurs relatively often. This is the reason for the proximity of the potential inner and outer race defect magnitudes in dataset 3 bearing 3. The same phenomenon can be seen for the other datasets.

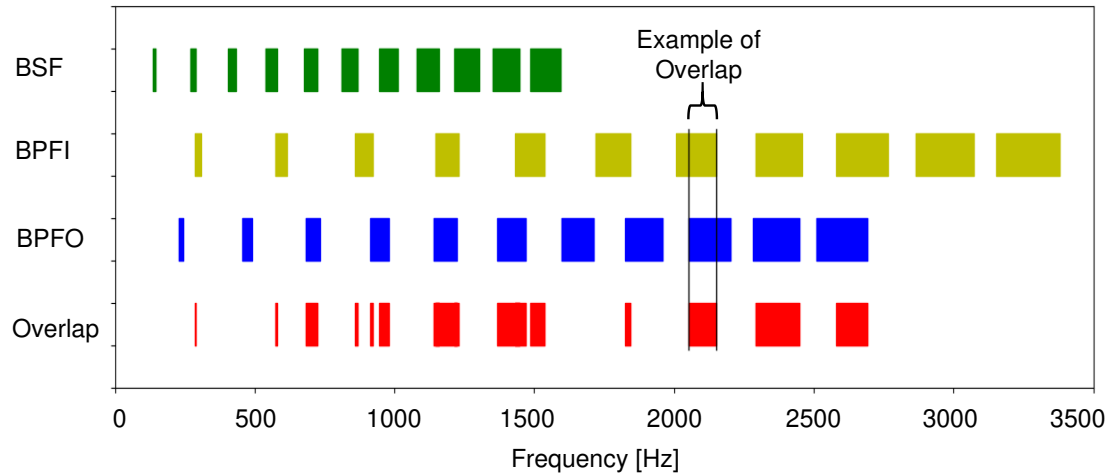


Figure 44 – Overlap of the different defect frequency intervals.

Potential inner and outer race defects are detected relatively early in dataset 1 bearing 4, as indicated in Appendix C in Table 11. Figure 45 shows the control chart for potential outer race defect magnitude of this data. The runtime of the experiment is 21,560 min and the detection time is 5,150 min. Figure 45 shows that the potential defect magnitude increases at around 5,000 min and then decreases again to the beginning level. Starting at around 12,500 min, the magnitude continuously increases again until the end of the experiment. The defect may be smoothed due to the rolling process, explaining the magnitude decrease before the severity increases again.

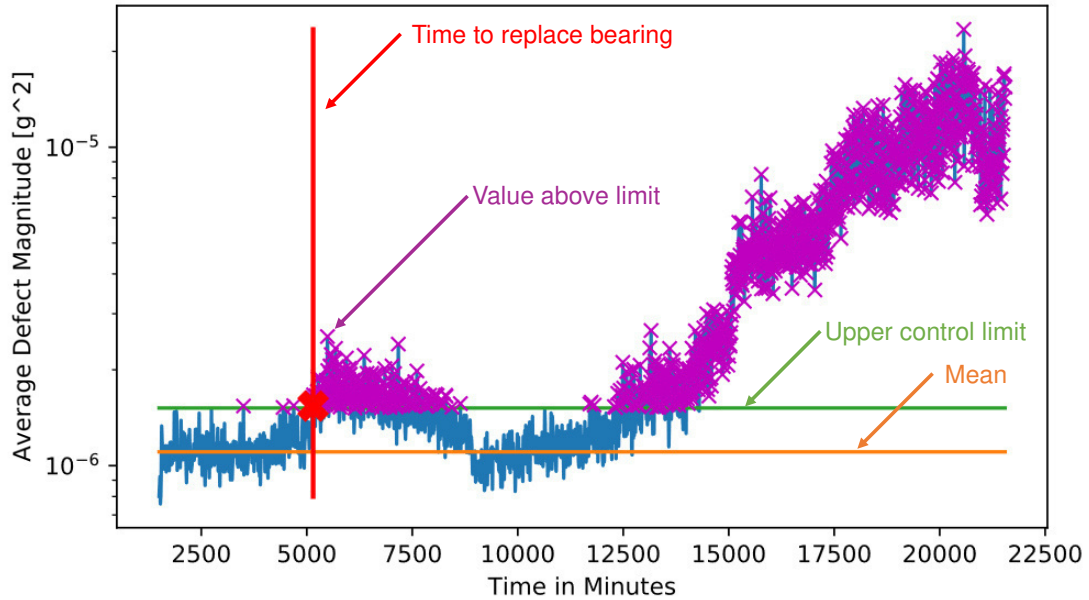


Figure 45 – Control chart for outer race defect dataset 1 bearing 4.

The detection times of the various parameters were compared for all faulty bearings. Figure 46 compares the remaining time before the experiment was stopped for dataset 3 bearing 3. This was calculated by subtracting the detection time shown in Table 9 from the total runtime of the experiment. The parameter maximum value was excluded from the diagram since its detection time was significantly earlier than the other detection times. The parameter crest factor was also excluded since it was not possible to detect failure of the bearing with this parameter. Figure 46 shows that the RMS in the time domain as well as arithmetic and geometric mean in the frequency domain detect the failure of the bearings earlier than the other parameters. The remaining times for the potential defect magnitudes (outer race and inner race) were also relatively high. The same conclusion regarding the sensitive parameters can be made for test dataset 1 and 2. Consequently, these six parameters should be monitored over time to detect a bearing defect. To reduce the false detection of a bearing defect, more than one parameter should be considered for the

decision whether to replace a bearing. For developing a monitoring algorithm, a bearing defect only should be flagged if one of the potential defect magnitudes and one other parameter (RMS in time domain or arithmetic/geometric mean in frequency domain) is above the calculated limit. With this approach, the decrease of the outer race defect magnitude of bearing 4 in dataset 1 does not influence the replacement time since RMS, arithmetic and geometric mean do not cross the UCL as early as the potential defect magnitudes.

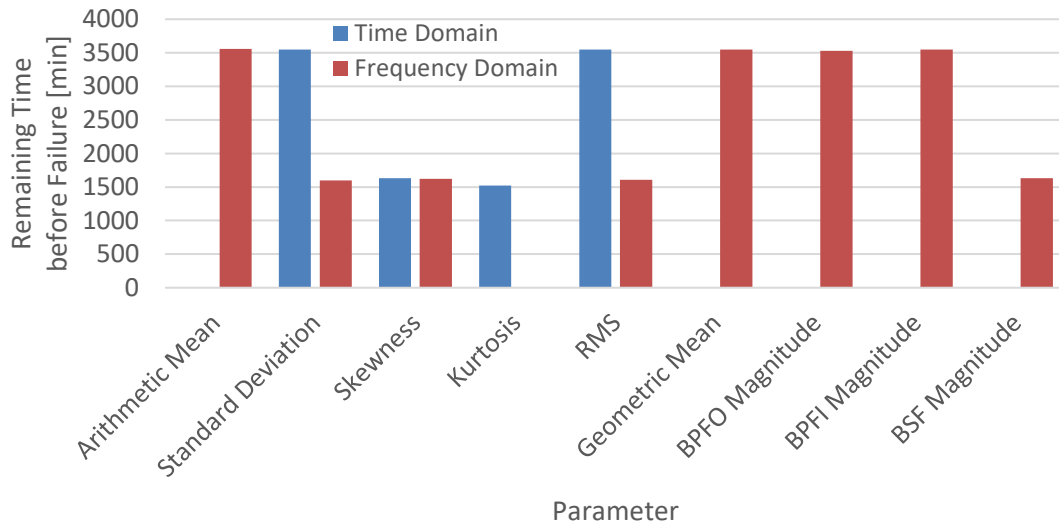


Figure 46 – Comparison of the parameters regarding the remaining time before failure of dataset 3 bearing 3.

4.2.4 Implementation Edge Monitoring Algorithm

Based on the analysis in the previous section, an algorithm was developed to monitor the identified parameters over time. Node-RED is used to implement the edge monitoring algorithm. Additionally, Node-RED calls a data analysis script written in Python to extract the parameters determined in the previous section. A flow chart of the developed edge monitoring algorithm is shown in Figure 47. The algorithm waits until the value of a certain

controller data point is in a specified range. The user can define this range according to the requirements of the specific application. The trigger mechanism was implemented for the direct and the indirect edge monitoring architecture presented in chapter 3. The Node-RED flow can read controller data points directly from the controller server or subscribe to the MQTT messages of the controller gateway. If the chosen controller data points are within the defined range, the vibration data is captured by the edge device MC and the vibration sensor. The vibration data is then sent to the edge device SBC. A Python script is called to execute the FFT and computes the potential defect magnitudes, the RMS in the time domain as well as arithmetic and geometric mean in the frequency domain. The user defines a number k of initial measurements. The default value is set to 100. If less than k measurements were taken, the computed parameters are stored locally on the edge device SBC and if exactly k measurements were taken, the UCLs of the different parameters are calculated according to Equation 13. The scaling factor α can also be adjusted by the user. As default, three is used. The computed UCLs for the different parameters are stored on the edge device SBC. If more than k measurements were taken, the parameters of the current measurement are compared against the calculated UCLs.

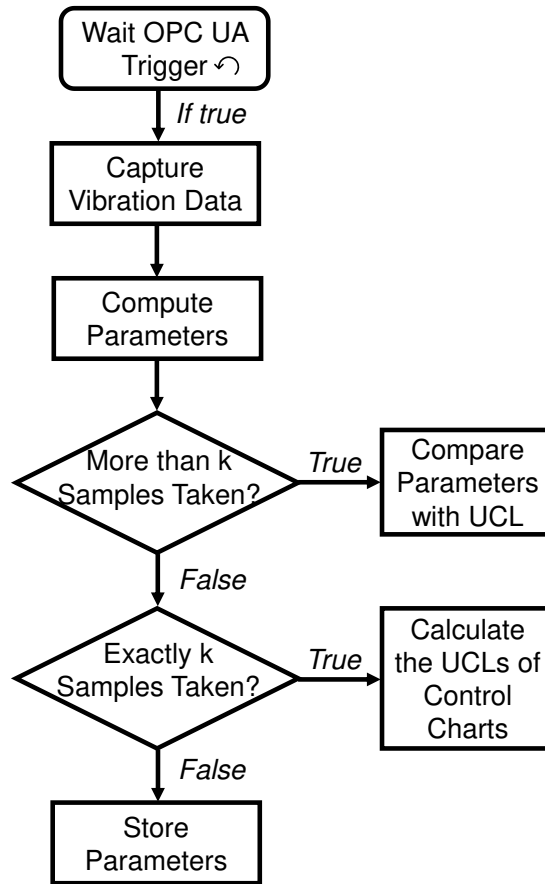


Figure 47 – Flow chart of the edge monitoring algorithm.

The control charts are shown in a dashboard. A picture of the dashboard for RMS is given in Figure 48. The last measured values are indicated in dark blue on the dashboard, and the UCL is indicated in light blue. In addition, a colored LED is displayed. If the LED is green, the most current parameter was below the UCL. If the LED is yellow, the parameter has been above the UCL once or two times in a row. If the LED is red, the parameter has been above the UCL three or more times in a row. Several information can be sent to a broker via MQTT: the raw vibration data, the result of the FFT, the computed parameters and the status of the bearing (based on the control chart).

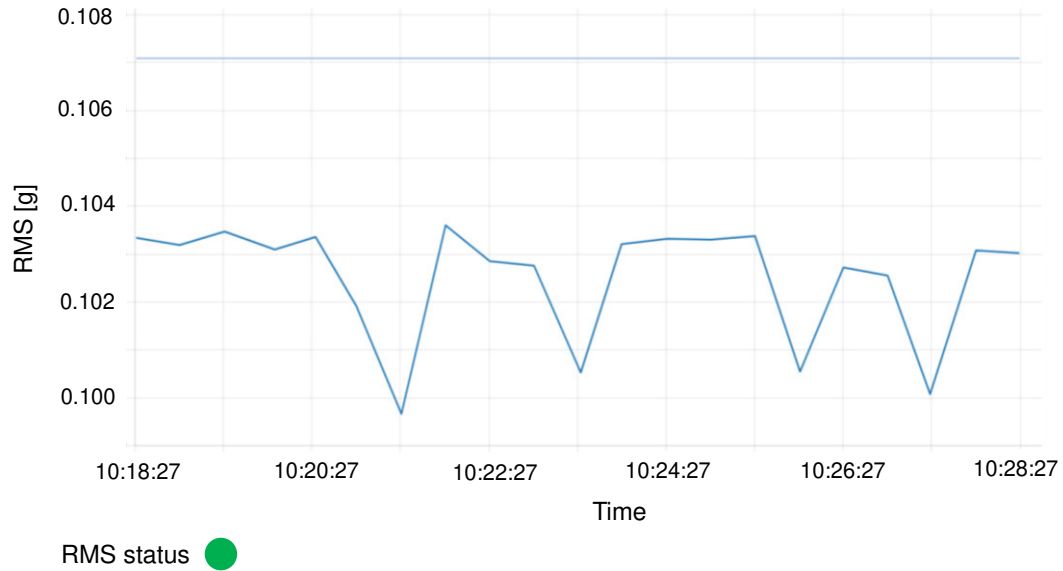


Figure 48 – Live dashboard of RMS with measured RMS values in dark blue and the UCL in light blue.

The bearing health monitoring algorithm was validated with the test dataset [64]. The vibration data of dataset 2 bearing 1 was sent via MQTT to the edge device. The data was analyzed with the developed Python script. The result of the analysis was shown at the dashboard and sent via MQTT to the broker. It was verified that the results of the original data analysis are the same as the results of the developed edge monitoring algorithm. The described monitoring algorithm for bearing health conditions is validated and can be tested in practice.

4.2.5 Assumptions

The bearing health edge monitoring algorithm was developed by analyzing a test dataset. A test rig was used to collect the vibration data. It is assumed that the results of analyzing the test dataset are transferable to other applications. In particular, the measurement results of the test rig in the laboratory environment are comparable to measurements taken from

industrial machines. In comparison to the set-up of the test rig, it might not be possible to mount the accelerometer directly on the bearings. Additionally, other components of machines might influence the measured vibration data. Another assumption is that the rotational speed and the load on the bearings are constant during the runtime of the experiment. The constant conditions are crucial to compare the different statistical parameters over time.

4.3 Experimental Method

The vibration data of a CNC machine has been monitored for approximately one and a half months to test the developed approach of monitoring several vibration parameters over time. The EMCOMILL E350 located in the Montgomery Machining Mall at Georgia Institute of Technology was equipped permanently with an accelerometer of type ADXL203EB. The technical specifications are described in Table 3. The lit-free box with the accelerometer is pictured in Figure 49. The accelerometer was sealed with epoxy to protect the circuits against coolant and fastened into the plastic box with threaded fasteners. A shielded wire was used to reduce the noise of the measured signal due to transmission.

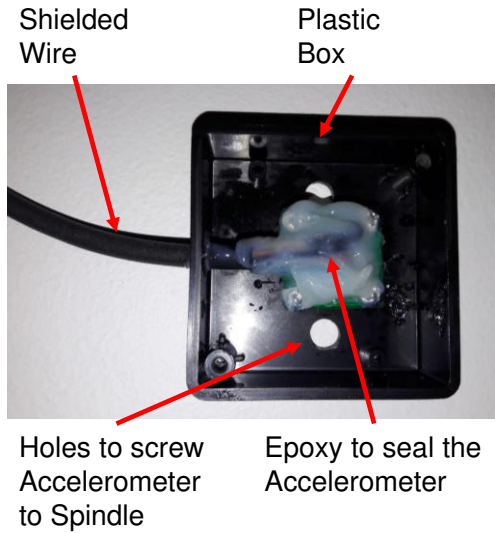


Figure 49 – Sealed accelerometer in plastic box.

Figure 50 shows the accelerometer box attached to the spindle. The box was affixed to the spindle with threaded fasteners. The wire leads out of the machine to the vibration measurement box. In Figure 50, the lid was removed for better illustration.

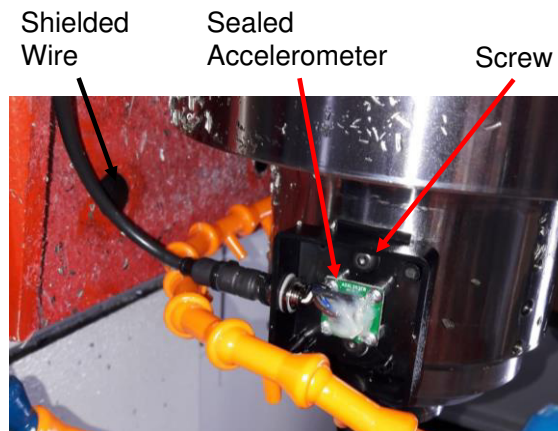


Figure 50 – Accelerometer box screwed to spindle.

The warmup program of the machine was used to monitor the different parameters. This program was executed every day the machine is used. The rotational spindle speed was increased in steps of 1,000 rpm from 1,000 rpm to 10,000 rpm during this program. At each

spindle speed, the machine runs for two minutes without cutting. During the warmup spindle program, the vibration data is measured every five seconds with a sampling frequency of 5,263 Hz and a sample size of 4,200. The recorded vibration data was labeled with the rotational speed during its measurement by using controller data points.

4.4 Results and Discussion

Figure 51 shows the RMS during the execution of the warmup program at 01/30/2020. The signal was relatively noisy after sample 20. The first samples appeared to exhibit less noise. These samples refer to the rotational speed of 1,000 rpm. This characteristic was also present with the other five parameters (arithmetic and geometric mean, potential defect magnitudes). To minimize the effect of noise, the monitoring of the parameters over time was based only on the measurements taken at 1,000 rpm.

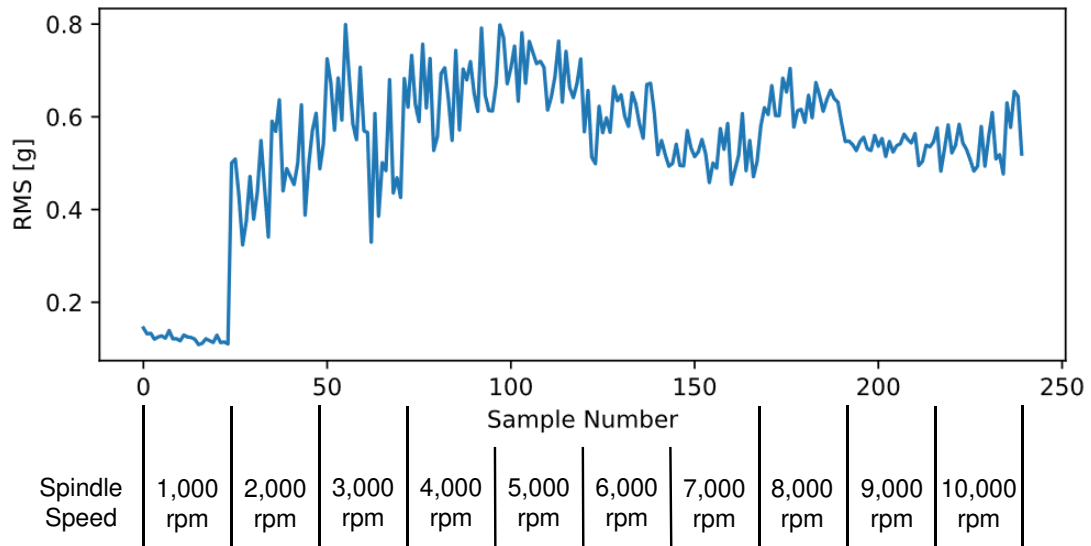


Figure 51 – RMS during warmup program at January 30, 2020.

For each day, all RMS values taken at 1,000 rpm were averaged to yield the average RMS. Figure 52 shows the average RMS over approximately one and a half months. Clearly, the

average RMS increased over time. The average RMS increased significantly between the 17th and the 18th of February.

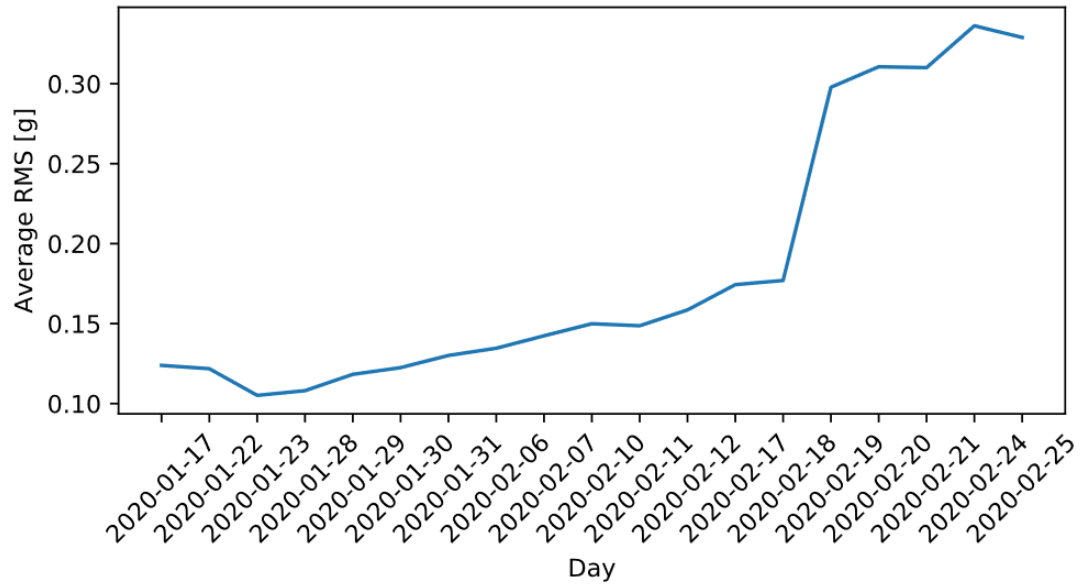


Figure 52 – Average RMS over days.

The other five parameter values also increased (see Appendix D). There are some possible explanations for this phenomenon. It is possible that the accelerometer loosened over time because it was only connected to the spindle by screws. Another possible reason is that the dynamics of the vibration sensor plastic box change over time. It is also possible that the spindle bearing has a defect and the severity of the defect increases over time. Further investigations are needed to identify the reason. One approach is to replace the plastic box with a more robust metal box. The influence of the box dynamics can be reduced in this way.

CHAPTER 5. CONCLUSION

5.1 Summary

The first part of this study analyzed the usage of controller data provided via OPC UA to trigger external sensor measurements. An edge device was used to collect sensor data and sent it via MQTT to a broker. Two different architectures for triggering sensor measurements by controller data points were introduced. In the first architecture, the edge device is directly connected to the controller server. In the second architecture, a controller gateway is used to read controller data points and sent them via MQTT to a broker. The edge device is subscribed to the relevant data points.

The limitations of both architectures regarding computational power and delays were analyzed. To identify the limitations regarding the computational power, the functionality of the system was tested with different workloads on the edge device and by using controller gateways. The use case of monitoring the trajectory of a toolpath based on controller data points shows the impact of the workloads. If streaming of controller data points to an MQTT broker is necessary, using a controller gateway is recommended. If streaming of controller data points is not necessary, the edge device can be directly connected to the controller server.

The delay in this study was defined as the time elapsed from when a controller data point in the controller server changes and this change is recognized on the edge device. The maximum expected delays were determined for both architectures. The use case of cut monitoring illustrates the impact of the measured delays. If a Raspberry Pi is used as a

controller gateway, the smallest maximum expected delay for the indirect edge monitoring architecture was measured. Using a local broker in comparison to a broker in the cloud reduces the maximum expected delay additionally. However, the smallest maximum expected delay was measured for the direct edge monitoring architecture since an additional delay due to MQTT communication is not introduced.

In the second part of this study, a bearing health monitoring algorithm was developed based on vibration data which takes advantage of the two developed architectures. A test dataset was analyzed to identify the relevant parameters which need to be monitored over time to track the bearing health condition. The results of this analysis were used to develop an edge bearing health monitoring algorithm which can be executed on the edge device. The measurement of vibration data is triggered by controller data points. The edge monitoring algorithm can be integrated with both proposed architectures. The developed edge monitoring algorithm was experimentally tested on a machine over one and a half months. The results of this experiment are not conclusive and need more investigation.

5.2 Contributions

In this study, two different architectures connecting a low-cost edge device with an OPC UA controller server to trigger sensor data acquisition by certain controller data points were developed. Both architectures can be integrated in a decoupled IoT architecture where different devices are loosely coupled to a central MQTT broker. A method was introduced to test both architectures regarding their computational power and the maximum expected delay. The results of this method support users to choose the appropriate architecture according to a specific use case. In addition, a bearing health edge monitoring algorithm

based on statistical parameters was developed. The edge monitoring algorithm can be hosted on an edge device and is automated by using controller data points to trigger vibration measurements.

5.3 Future Work

Both proposed architectures were analyzed within certain conditions. The measured delays and the limits of the computational power depend on the utilized Wi-Fi network and on the implementation of the controller server. Further measurements in different Wi-Fi networks and with different controller servers are necessary to get a more conclusive overview. Additionally, the identified limits also depend on the implementations in Node-RED for the edge device and the gateway. Changing the programming language or the utilized packages in Node-RED might improve the performance of the system.

In this study, the controller data points were captured by using the read and write method. However, subscribing to the controller data points could decrease the maximum expected delay between controller server and client since the request interval decreases.

Further investigations could address the challenge predicting the communication time via MQTT. The measurement principle to determine the MQTT delays shown in Figure 21 could be used to ensure that the communication time between the gateway and the edge device was within a certain range. Therefore, additional logic needs to be implemented on the gateway to timestamp the messages on the gateway. The edge device could send the received MQTT message back to the gateway and the roundtrip delay can be calculated on the controller gateway. If the measured roundtrip delay is above a certain defined limit, the system could exclude this sensor measurement. In the use case of cut monitoring, the part

would need to be inspected manually to ensure product quality if the sensor measurement did not start in time. If the sensor measurement did start in time, manual inspection is not needed.

The monitoring algorithm for bearing health was mainly developed and tested with a test dataset. The next step should be a long-term test on different CNC machines. This can either be done using the edge devices of this study or using existing infrastructure to capture vibration data. In addition, the results of the started experiment using the bearing health edge monitoring algorithm are not conclusive. The reason for the increase of the parameters over time needs to be identified. The first step in this investigation is the development of a more sophisticated vibration sensor box fabricated from metal to reduce mechanical compliance.

APPENDIX A. TEST MESSAGE MQTT DELAY

The test message used for determining the MQTT delays is:

```
{  
  "topic": "TestingLatency",  
  "payload":  
    {  
      "dateTime": "2020-01-22T16:25:47.897Z",  
      "assetId": "OPCUAexample",  
      "dataItemId": "rotaryVelocityS",  
      "stamp": 5,  
      "values": 10000},  
  "_msgid": "3b39c5db.c98e0a"  
}
```

APPENDIX B. EQUATIONS FOR STATISTICAL PARAMETERS

In Appendix B, the equations for the used statistical parameters are given. The input to each equation is an array x with the number of samples N_s . The parameters can be calculated from the measured vibration data as well as from the determined magnitudes in the frequency domain.

Arithmetic mean [67]:

$$x_{mean} = \bar{x} = \frac{1}{N_s} \sum_{i=1}^{N_s} x_i \quad (14)$$

Geometric mean [67]:

$$G(x) = \sqrt[N_s]{x_1 * x_2 * \dots * x_{N_s}} \quad (15)$$

Standard deviation [67]:

$$\sigma(x) = \sqrt{\frac{1}{N_s} \sum_{i=1}^{N_s} (x_i - \bar{x})^2} \quad (16)$$

Skewness [68]: (μ_3 : third central moment)

$$Skew(x) = \frac{\mu_3}{\sigma^3} = \frac{\frac{1}{N_s} \sum_{i=1}^{N_s} (x_i - \bar{x})^3}{\sqrt{\frac{1}{N_s} \sum_{i=1}^{N_s} (x_i - \bar{x})^2}} \quad (17)$$

Kurtosis [68]: (μ_4 : fourth central moment)

$$Kurt(x) = \frac{\mu_4}{\sigma^4} - 3 = \frac{\frac{1}{N_s} \sum_{i=1}^{N_s} (x_i - \bar{x})^4}{\left(\frac{1}{N_s} \sum_{i=1}^{N_s} (x_i - \bar{x})^2 \right)^2} - 3 \quad (18)$$

Root mean square (RMS) [68]:

$$RMS(x) = \sqrt{\frac{1}{N_s} \sum_{i=1}^{N_s} x_i^2} \quad (19)$$

Maximum value [67]:

$$M(x) = \max(x) \quad (20)$$

Crest factor [68]:

$$C(x) = \frac{\max(x)}{RMS(x)} \quad (21)$$

APPENDIX C. DETECTION TIME FOR DIFFERENT PARAMETERS AND BEARINGS

In Appendix C, the detection time of the different parameters for the faulty bearings of the test dataset are stated.

Dataset 1 Bearing 3: (Channel 6 of the test dataset)

The runtime of this experiment was 21,560 min.

Table 10 – Detection time for dataset 1 bearing 3.

Parameter	Detection Time in Time Domain [min]	Detection Time in Frequency Domain [min]
Arithmetic Mean	16,920	16,680
Standard Deviation	16,640	20,270
Skewness	18,520	No Detection
Kurtosis	18,270	No Detection
Root Mean Square (RMS)	15,550	19,060
Maximum Value	18,260	21,190
Crest Factor	18,270	No Detection
Geometric Mean		16,640
Potential Outer Race Defect Magnitude		18,070
Potential Inner Race Defect Magnitude		16,460
Potential Rolling Element Defect Magnitude		21,210

For test dataset 1, the first 150 measurements have been ignored since there is an immediate shift in most of the parameters after around 140 measurements which could not be explained by the documentation of the test dataset.

Dataset 1 Bearing 4: (Channel 8 of the test dataset)

The runtime of this experiment was 21,560 min.

Table 11 – Detection time for dataset 1 bearing 4.

Parameter	Detection Time in Time Domain [min]	Detection Time in Frequency Domain [min]
Arithmetic Mean	9,540	14,370
Standard Deviation	14,360	15,090
Skewness	16,700	3,510
Kurtosis	16,160	5,060
Root Mean Square (RMS)	14,420	15,080
Maximum Value	16,160	17,910
Crest Factor	16,160	3,510
Geometric Mean		14,740
Potential Outer Race Defect Magnitude		5,150
Potential Inner Race Defect Magnitude		3,280
Potential Rolling Element Defect Magnitude		14,640

Dataset 2 Bearing 1: (Channel 1 of the test dataset)

The runtime of this experiment was 9,840 min.

Table 12 – Detection time for dataset 2 bearing 1.

Parameter	Detection Time in Time Domain [min]	Detection Time in Frequency Domain [min]
Arithmetic Mean	No Detection	5,340
Standard Deviation	5,340	6,490
Skewness	7,100	5,500
Kurtosis	6,490	6,130
Root Mean Square (RMS)	5,340	6,490
Maximum Value	7,010	7,040
Crest Factor	No Detection	6,410
Geometric Mean		6,090
Potential Outer Race Defect Magnitude		6,110
Potential Inner Race Defect Magnitude		5,470
Potential Rolling Element Defect Magnitude		8,590

Dataset 3 Bearing 3: (Channel 3 of the test dataset)

The runtime of this experiment was 63,240 min.

Table 13 – Detection time for dataset 3 bearing 3.

Parameter	Detection Time in Time Domain [min]	Detection Time in Frequency Domain [min]
Arithmetic Mean	No Detection	59,680
Standard Deviation	59,690	61,640
Skewness	61,610	61,620
Kurtosis	61,720	No Detection
Root Mean Square (RMS)	59,690	61,630
Maximum Value	61,960	45,570
Crest Factor	No Detection	No Detection
Geometric Mean		59,690
Potential Outer Race Defect Magnitude		59,710
Potential Inner Race Defect Magnitude		59,690
Potential Rolling Element Defect Magnitude		61,610

APPENDIX D. AVERAGE PARAMETERS OVER DAYS

In Appendix D, the results of the experiment are shown. The figures show the development of each parameter over time.

Average RMS:

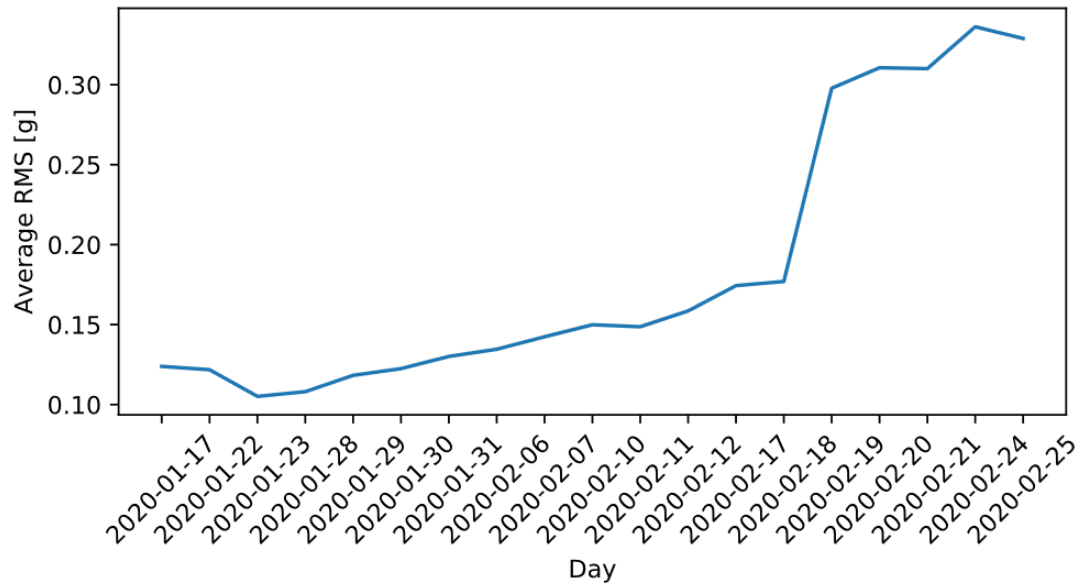


Figure 53 – Average RMS over time.

Average Arithmetic Mean:

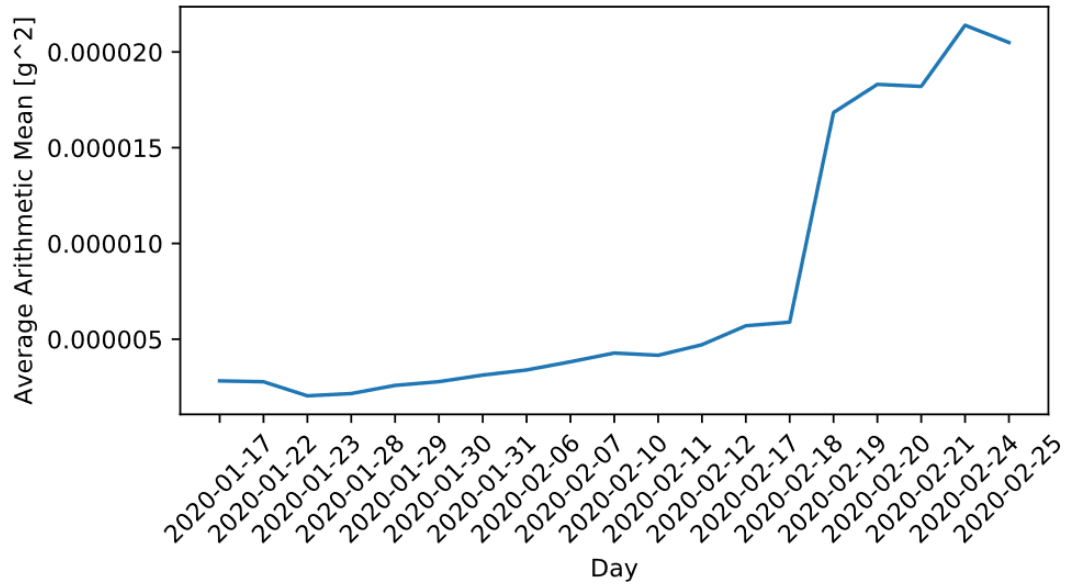


Figure 54 – Average arithmetic mean over time.

Average Geometric Mean:

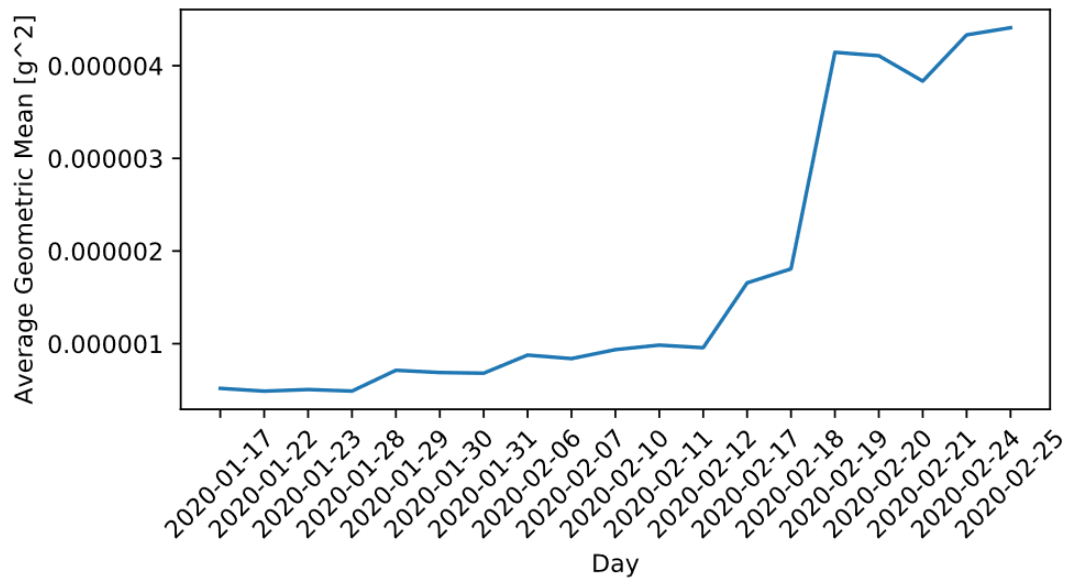


Figure 55 – Average geometric mean over time.

Average Potential Outer Race Defect Magnitude:

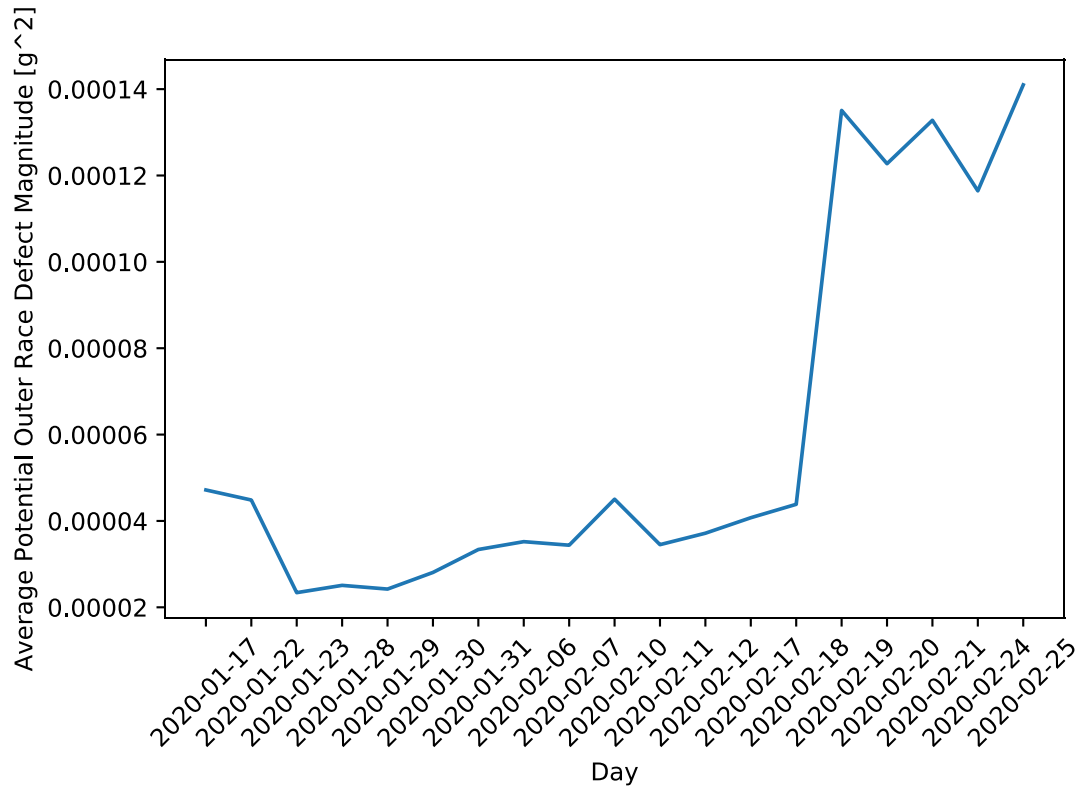


Figure 56 – Average potential outer race defect magnitude over time.

Average Potential Inner Race Defect Magnitude:

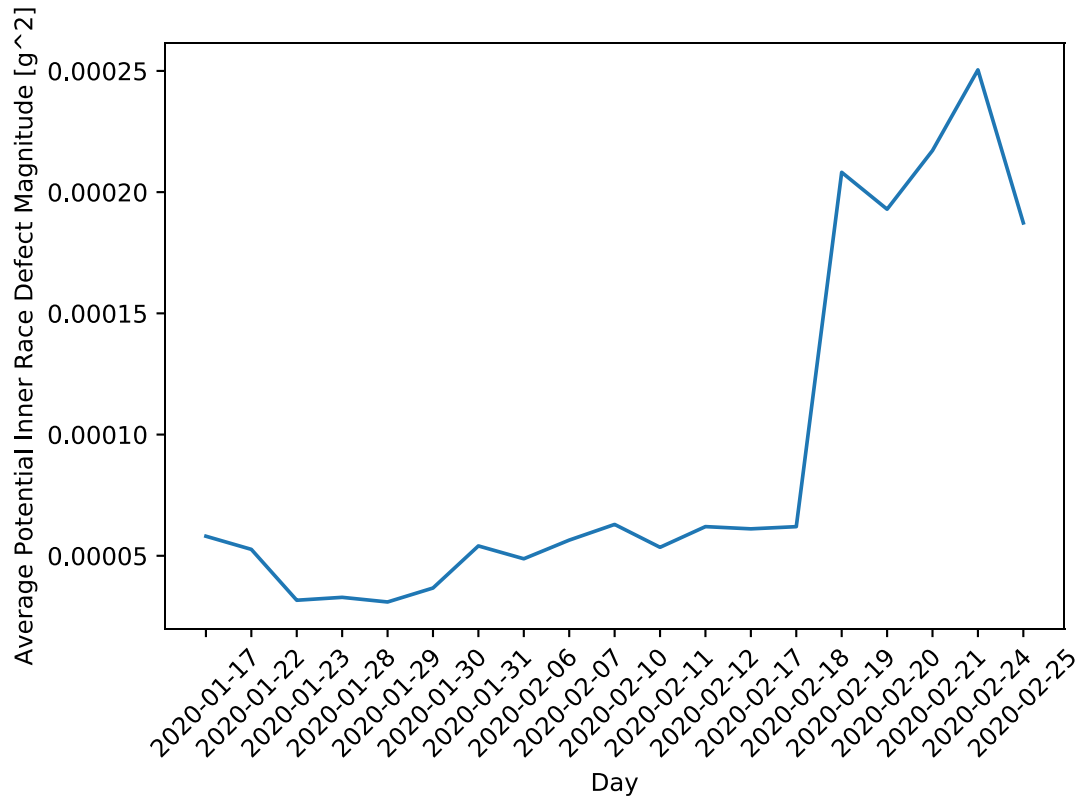


Figure 57 – Average potential inner race defect magnitude over time.

Average Potential Rolling Element Defect Magnitude:

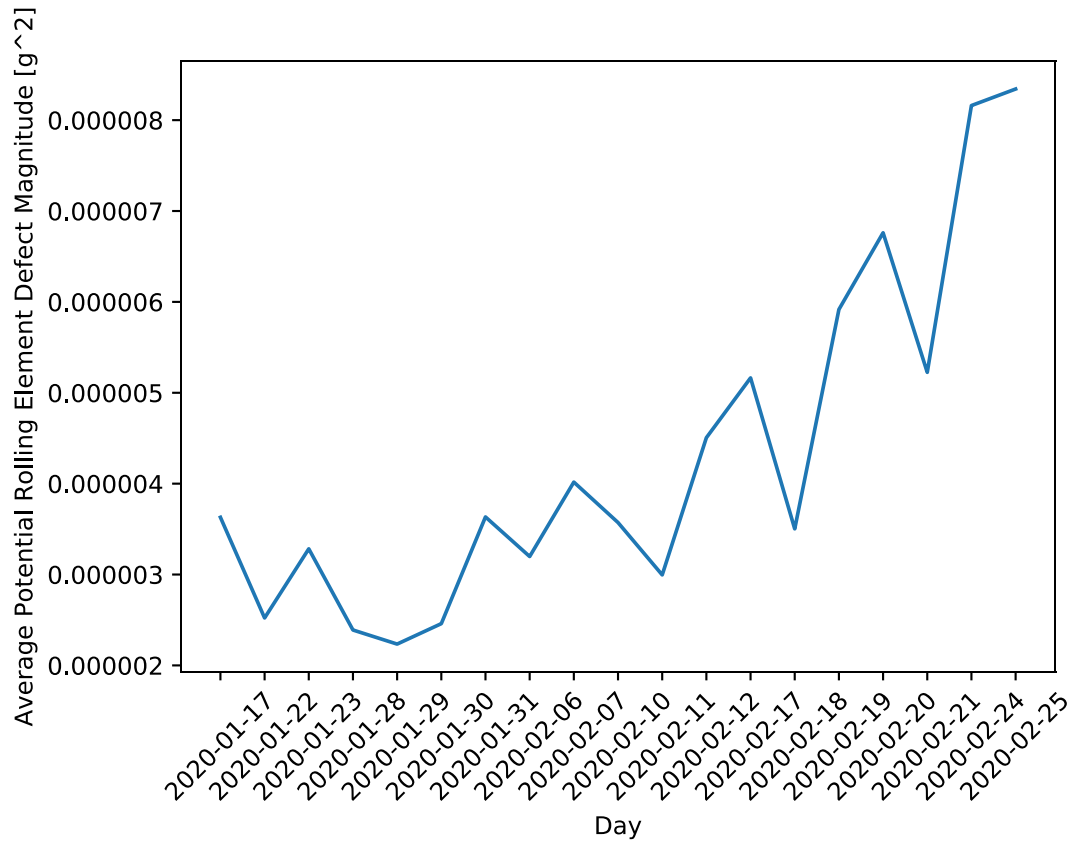


Figure 58 – Average potential rolling element defect magnitude over time.

REFERENCES

- [1] M. T. Tapale, M. N. Birje, P. S. Challagidad, and R. H. Goudar, “Cloud computing review: concepts, technology, challenges and security.,” *IJCC*, vol. 6, no. 1, pp. 32–57, 2017.
- [2] W. Shi, and S. Dustdar, “The Promise of Edge Computing.,” *Computer*, vol. 49, no. 5, pp. 78–81, 2016.
- [3] Y. Ai, M. Peng, and K. Zhang, “Edge computing technologies for Internet of Things: a primer.,” *Digital Communications and Networks*, vol. 4, no. 2, pp. 77–86, 2018.
- [4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges.,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [5] C. Liu, and X. Xu, “Cyber-physical Machine Tool – The Era of Machine Tool 4.0.,” *Procedia CIRP*, vol. 63, pp. 70–75, 2017.
- [6] G.-Y. Lee *et al.*, “Machine health management in smart factory: A review.,” *Journal of Mechanical Science and Technology*, vol. 32, no. 3, pp. 987–1009, 2018.
- [7] D. Dyer, and R. M. Stewart, “Detection of Rolling Element Bearing Damage by Statistical Vibration Analysis.,” *Journal of Mechanical Design*, vol. 100, no. 2, pp. 229–235, 1978.

- [8] C. Liu, H. Vengayil, Y. Lu, and X. Xu, "A Cyber-Physical Machine Tools Platform using OPC UA and MTConnect.," *Journal of Manufacturing Systems*, vol. 51, pp. 61–74, 2019.
- [9] S. Cavalieri, and G. Cutuli, "Performance evaluation of OPC UA.," *2010 IEEE 15th conference on emerging technologies & factory automation (ETFA 2010)*, pp. 1–8, 2010.
- [10] S. Cavalieri, and F. Chiacchio, "Analysis of OPC UA performances," *Computer Standards & Interfaces*, vol. 36, no. 1, pp. 165–177, 2013.
- [11] F. G. Vázquez, "Test Platform for the Performance Evaluation of OPC-UA Servers for Fast Data Transfer Between Intelligent Equipment.," *The Fourth International Conference on Intelligent Systems and Applications*, pp. 179–182, 2015.
- [12] O. Motaghare, A. S. Pillai, and K.I. Ramachandran, "Predictive Maintenance Architecture.," *2018 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–4, 2018.
- [13] A. Garg, and S. G. Deshmukh, "Maintenance management: literature review and directions.," *Journal of Quality in Maintenance Engineering*, vol. 12, no. 3, pp. 205–238, 2006.
- [14] R. K. Mobley, *An introduction to predictive maintenance.*: Elsevier, 2002.

- [15] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, “A survey on application layer protocols for the internet of things.,” *Transaction on IoT and Cloud computing*, vol. 3, no. 1, pp. 11–17, 2015.
- [16] N. Weinert, M. Plank, and A. Ullrich, *Metamorphose zur intelligenten und vernetzten Fabrik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017.
- [17] S. H. Atluru, and A. Deshpande, “Data to information: can MTConnect deliver the promise.,” *Transactions of NAMRI/SME*, vol. 37, pp. 197–204, 2009.
- [18] B. Edrington, B. Zhao, A. Hansel, M. Mori, and M. Fujishima, “Machine Monitoring System Based on MTConnect Technology.,” *Procedia CIRP*, vol. 22, pp. 92–97, 2014.
- [19] A. Vijayaraghavan, W. Sobel, A. Fox, D. Dornfeld, and P. Warndorf, “Improving machine tool interoperability using standardized interface protocols: MT connect.,” 2008.
- [20] V. Plenk, *Angewandte Netzwerktechnik kompakt*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019.
- [21] S.-H. Leitner, and W. Mahnke, “OPC UA – Service-oriented Architecture for Industrial Applications.,” *ABB Corporate Research Center*, vol. 48, pp. 61–66, 2006.
- [22] S. Cavalieri, and F. Chiacchio, “Analysis of OPC UA performances.,” *Computer Standards & Interfaces*, vol. 36, no. 1, pp. 165–177, 2013.

- [23] Siemens AG, “SINUMERIK Access MyMachine / OPC UA.: Configuration Manual,” Dec. 2018.
- [24] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, “Internet of Things: Survey and open issues of MQTT Protocol.,” *2017 International Conference on Engineering & MIS (ICEMIS)*, pp. 1–6, 2017.
- [25] J. E. Luzuriaga, J. C. Cano, C. Calafate, P. Manzoni, M. Perez, and P. Boronat, “Handling mobility in IoT applications using the MQTT protocol.,” *2015 Internet Technologies and Applications (ITA)*, pp. 245–250, 2015.
- [26] *Node-RED*. [Online]. Available: <https://nodered.org/> (accessed: Mar. 27 2020).
- [27] M. Lekic, and G. Gardasevic, “IoT sensor integration to Node-RED platform.,” *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–5, 2018.
- [28] L. Wei, Z. Yunxiao, and L. Weihai, “Streaming Information Transmission Based on OPC UA.,” *Journal of Physics: Conference Series*, vol. 1187, no. 4, 2019.
- [29] C. Shrestha, “Can you use JavaScript for Data Science?,” *JavaScript in Plain English*, 20 Feb., 2019. <https://medium.com/javascript-in-plain-english/how-about-data-science-and-javascript-lets-take-a-look-c123c6981afa> (accessed: Mar. 27 2020).
- [30] Laura, “Python vs. JavaScript: Which One Should You Learn?,” *BitDegree*, 28 May., 2019. <https://www.bitdegree.org/tutorials/python-vs-javascript/> (accessed: Mar. 27 2020).

- [31] J. Hunt, *A Beginners Guide to Python 3 Programming*. Cham: Springer International Publishing, 2019.
- [32] J. Hunt, *Advanced Guide to Python 3 Programming*. Cham: Springer International Publishing, 2019.
- [33] Jie Liu *et al.*, “Anomaly Detection in Manufacturing Systems Using Structured Neural Networks.,” *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, pp. 175–180, 2018.
- [34] H. Chen, X. Fei, S. Wang, X. Lu, G. Jin, and X. Wu, “Energy Consumption Data Based Machine Anomaly Detection.,” *2014 Second International Conference on Advanced Cloud and Big Data*, pp. 136–142, 2014.
- [35] F. Boud, and N. N. Z. Gindy, “Application of multi-sensor signals for monitoring tool/workpiece condition in broaching.,” *International Journal of Computer Integrated Manufacturing*, vol. 21, no. 6, pp. 715–729, 2008.
- [36] L. Zhang, S. Elghazoly, and B. Tweedie, “Introducing AnomDB: An Unsupervised Anomaly Detection Method for CNC Machine Control Data.,” *Proc. Annu. Conf. PHM Soc*, vol. 11, 2019.
- [37] M. A. Saez, F. P. Maturana, K. Barton, and D. M. Tilbury, “Context-Sensitive Modeling and Analysis of Cyber-Physical Manufacturing Systems for Anomaly Detection and Diagnosis.,” *IEEE Trans. Automat. Sci. Eng.*, vol. 17, no. 1, pp. 29–40, 2020.

- [38] D. Hastbacka, L. Barna, M. Karaila, Y. Liang, P. Tuominen, and S. Kuikka, “Device status information service architecture for condition monitoring using OPC UA.,” *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–7, 2014.
- [39] N. K. Verma, R. Dev, N. K. Dhar, D. J. Singh, and A. Salour, “Real-time remote monitoring of an air compressor using MTConnect standard protocol.,” *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pp. 109–116, 2017.
- [40] A. José Álvares, L. E. S. d. Oliveira, and J. C. E. Ferreira, “Development of a Cyber-Physical framework for monitoring and teleoperation of a CNC lathe based on MTconnect and OPC protocols.,” *International Journal of Computer Integrated Manufacturing*, vol. 31, no. 11, pp. 1049–1066, 2018.
- [41] P. Sun, Q. Liu, J. Ding, and S. Pi, “Open CNC System Design for Multiple Intelligent Functions Based on TwinCAT and .NET Framework.,” *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 910–915, 2017.
- [42] S. Chen, C. Yin, and X. Li, “Implementation of MTConnect in Machine Monitoring System for CNCs.,” *2017 5th International Conference on Enterprise Systems (ES)*, pp. 70–75, 2017.
- [43] S. Raileanu, T. Borangiu, O. Morariu, and I. Iacob, “Edge Computing in Industrial IoT Framework for Cloud-based Manufacturing Control.,” *2018 22nd International*

- Conference on System Theory, Control and Computing (ICSTCC)*, pp. 261–266, 2018.
- [44] Z. Wen, X. Liu, Y. Xu, and J. Zou, “A RESTful framework for Internet of things based on software defined network in modern manufacturing.,” *The International Journal of Advanced Manufacturing Technology*, vol. 84, 1-4, pp. 361–369, 2016.
- [45] D. Newman, M. Parto, K. Saleeby, T. Kurfess, and A. Dugenske, “Development of a Digital Architecture for Distributed CNC Machine Health Monitoring.,” *Smart and Sustainable Manufacturing Systems*, vol. 3, no. 2, pp. 68–82, 2019.
- [46] Niklas B. Tritschler, “Developing a Device for Automatic Monitoring of Rolling Element Bearing Conditions.,” Master's Thesis, Georgia Institute of Technology, Atlanta, GA, 2019.
- [47] M. M. Design, *EDIMAX - Wireless Adapters - N150 - 150Mbps Wireless IEEE802.11b/g/n nano USB Adapter*. [Online]. Available: https://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/in/wireless_adapters_n150/ew-7811un/ (accessed: Apr. 15 2020).
- [48] *Emcomill E350: EMCO lathes and milling machines for CNC turning and milling: Milling*. [Online]. Available: <https://www.emco-world.com/en/products/milling/cat/27/d/2/p/1000236%2C27/pr/emcomill-e350.html> (accessed: Mar. 31 2020).
- [49] *BeagleBoard.org - black*. [Online]. Available: <https://beagleboard.org/black> (accessed: Apr. 6 2020).

- [50] *BeagleBoard.org - black-wireless*. [Online]. Available: <https://beagleboard.org/black-wireless> (accessed: Apr. 6 2020).
- [51] *Buy a Raspberry Pi 3 Model B+ – Raspberry Pi*. [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/> (accessed: Apr. 6 2020).
- [52] I. Analog Devices, “ADXL103/ADXL203 (Rev. F): Data Sheet,”
- [53] U. Fischer *et al.*, *Tabellenbuch Metall*, 45th ed. Haan-Gruiten: Verl. Europa-Lehrmittel Nourney Vollmer, 2011.
- [54] *Eclipse Mosquitto*. [Online]. Available: <https://mosquitto.org/> (accessed: May 4 2020).
- [55] P. Ferrari, A. Flammini, E. Sisinni, S. Rinaldi, D. Brandao, and M. S. Rocha, “Delay Estimation of Industrial IoT Applications Based on Messaging Protocols.,” *IEEE Transactions on Instrumentation and Measurement*, vol. 67, no. 9, pp. 2188–2199, 2018.
- [56] P. Ferrari, E. Sisinni, D. Brandão, and M. Rocha, “Evaluation of communication latency in industrial IoT applications.,” *2017 IEEE International Workshop on Measurement and Networking (M&N)*, pp. 1–6, 2017.
- [57] T. Williams, X. Ribadeneira, S. Billington, and T. Kurfess, “Rolling element bearing diagnostics in run-to-failure lifetime testing.,” *Mechanical Systems and Signal Processing*, vol. 15, no. 5, pp. 979–993, 2001.

- [58] G. Gautier, R. Serra, and J.-M. Mencik, “Roller Bearing Monitoring by New Subspace-Based Damage Indicator.,” *Shock and Vibration*, vol. 2015, no. 8, pp. 1–11, 2015.
- [59] *Rolling element bearings - Mobius Institute*. [Online]. Available: <https://www.mobiusinstitute.com/site2/item.asp?LinkID=8011&iVibe=1&sTitle=Rolling%20element%20bearings> (accessed: May 12 2020).
- [60] Y. Wang, P. W. Tse, B. Tang, Y. Qin, L. Deng, and T. Huang, “Kurtogram manifold learning and its application to rolling bearing weak signal detection.,” *Measurement*, vol. 127, pp. 533–545, 2018.
- [61] W. A. Smith, and R. B. Randall, “Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study.,” *Mechanical Systems and Signal Processing*, 64-65, pp. 100–131, 2015.
- [62] J. I. Taylor, “Identification of Bearing Defects by Spectral Analysis.,” *Journal of Mechanical Design*, vol. 102, no. 2, pp. 199–204, 1980.
- [63] R. B. Randall, *Vibration-based condition monitoring: industrial, aerospace and automotive applications.*: John Wiley & Sons, 2011.
- [64] H. Qiu, J. Lee, J. Lin, and G. Yu, “Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics.,” *Journal of Sound and Vibration*, vol. 289, 4-5, pp. 1066–1090, 2006.

- [65] C. Liu, and K. Gryllias, “A semi-supervised Support Vector Data Description- based fault detection method for rolling element bearings based on Cyclic Spectral Coherence.,” *Mechanical Systems and Signal Processing*, vol. 140, pp. 1–13, 2020.
- [66] J. S. Oakland, *Statistical Process Control.*: Butterworth-Heinemann.
- [67] J. Grabmeier, and S. Hagl, *Statistik: Grundwissen und Formeln.*, 216th ed.: Haufe-Lexware, 2010.
- [68] R. B. W. Heng, and M. J. M. Nor, “Statistical analysis of sound and vibration signals for monitoring rolling element bearing condition.,” *Applied Acoustics*, vol. 53, pp. 211–226, 1998.