TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Sergei Jegorov 204707IASM

Middleware framework for Digital Twin entities communication

Master's thesis

Supervisor: Anton Rassõlkin
Ph.D.,
Co-Supervisor Eduard Petlenkov
Ph.D.

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sergei Jegorov 204707IASM

Vahevararaamistik digitaalse kaksiku üksuste infovahetuseks

Magistritöö

Juhendaja: Anton Rassõlkin
Ph.D.,
Kaasjuhendaja: Eduard Petlenkov
Ph.D.

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Sergei Jegorov

05.05.2022

Abstract

Digital Twin technology in the automotive industry is a growing trend that promises to bring accurate and cost-effective simulations, testing environments and predictive maintenance platforms. Autonomous vehicles are a special case - the number of sensors such vehicles possess and the amounts of data they generate can help to create precise, sophisticated models and environments for testing and analysis. To make this happen, a research project "Digital Twin for propulsion drive of autonomous electric vehicle" (project number PSG-453) was founded.

In this Master's thesis, a middleware framework for communication of Digital Twin entities is proposed. The framework based on Robot Operating System 2 (ROS2) and micro-ROS frameworks is used to connect two entities of the propulsion drive system Digital Twin. In the end, the latency tests are used to verify the reliability and speed of the framework.

This thesis is written in English language and is 46 pages long, including 6 chapters, 15 figures and 5 tables.

Annotatsioon

Digitaalsete kaksikute tehnoloogia on autotööstuses kiiresti kasvav arengusuund, mis lubab luua täpse ja tootliku keskonda simulatsiooni, testimise ja ennustava hoolduse jaoks. Autonoomsed sõidukid on erijuhtumid - suur arv andureid võimaldab genereerida piisavalt andmeid selleks, et luua täpne ja keeruline keskkond autonoomsete sõidukite testimiseks ja analüüsiks. Seetõttu oli "Isejuhtiva elektrisõiduki veoajami digitaalne kaksik" (projekti kood PSG-453) loodud.

Antud lõputöö pakub vahevararaamistiku digitaal kaksiku üksuste infovahetuseks ning selgitab, kuidas see seob digikaksiku üksusi kokku. Vahevararaamistik on loodud Roboti Operatsioonsüsteemi 2 (ROS2) ning micro-ROS'i põhjal. Kokkuvõtes on esitatud latentsuse testide tulemused, mis kinnitavad, et vahevararaamistik vastab kiiruse ja töökindluse nõuetele.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 46 leheküljel, 6 peatükki, 15 joonist, 5 tabelit.

List of abbreviations and terms

AI	Artificial Intelligence
API	Application Programming Interface
Black box	Object or system producing useful information without revealing any information about its internal workings
CAD	Computer-Aided Design
CLI	Command Line Interface
DAS	Data Acquisition System
DDS	Data Distribution Service
DT	Digital Twin
Guest machine	An independent instance of an OS and associated software and information
GUI	Graphical User Interface
Host machine	The physical machine that provides the guest VM with computing hardware resources
IM	Induction Motor
MATLAB	Commercial numerical analysis programming platform
MCU	Microcontroller Unit
micro-ROS	Native embedded implementation of ROS2
OS	Operating System
PDS	Propulsion Drive System
PMSM	Permanent Magnet Synchronous Machine
PSG-453	DT for propulsion drive of autonomous electric vehicle project code
QoS	Quality of Service
ROS	Robot Operating System
ROS2	Robot Operating System 2 – the successor of ROS
RTT	Round Trip Time
TB	Test Bench
USB	Universal Serial Bus
VM	Virtual machine

Table of contents

List of figures

List of tables

1 Introduction

With the increasing complexity of modern mechatronic systems, the traditional methods of monitoring and maintaining these systems have become inapplicable. At the same time, their sophisticated design and ability to generate large amounts of data open new ways for analysis and simulations.

One example of such a system is the self-driving vehicle ISEAUTO which is being developed on the premises of Tallinn University of Technology (TalTech) since 2018 [1]. A large number of installed sensors and powerful processing units allow this vehicle to navigate autonomously by processing the surrounding environment and making choices based on the received data. Very little is done towards an in-depth understanding of how these autonomous vehicles are affected during operation, considering how analysis of the vehicle's working systems can improve its overall performance.

To solve this problem, a research project PSG-453 [2] [3] was established, which aims to develop a specialized, unsupervised analysis of a propulsion drive system (PDS) of ISEAUTO based on the technology of digital twins (DT). The outcomes of this project are expected to be: a new educational tool, the discovery of new methods for monitoring and maintenance, and an improved analysis of existing systems.

The task of this thesis is to implement the middleware that connects specifically chosen hardware and software components of the DT and test the latency of implemented middleware solution. The thesis is organized in the following way. Chapter 2 presents the background of digital twin technology, outlines the state of the art in DT technology in the automotive field, and provides background to the PSG-453 project. Chapter 3 describes the selection of middleware framework and provides an overview of the selected framework. Chapter 4 covers the implementation details of the middleware. Chapter 5 describes the results of implementation and provides suggestions for future work. Conclusions are given in the Summary section.

2 Background

This section provides an insight into the current state of the art of DT, the history of the concept, and notable examples of systems deployed with DT principles in the automotive industry. At the end of the chapter, an overview and state of the ongoing project are given with defined goals to be achieved.

2.1 Definition of Digital Twin

There are several definitions of DT that were given over time by various academics and organizations. The first-ever definition originates from Dr. Michael Grieves who introduced this concept in 2002 – DT is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level, as shown in Figure 1 [4].

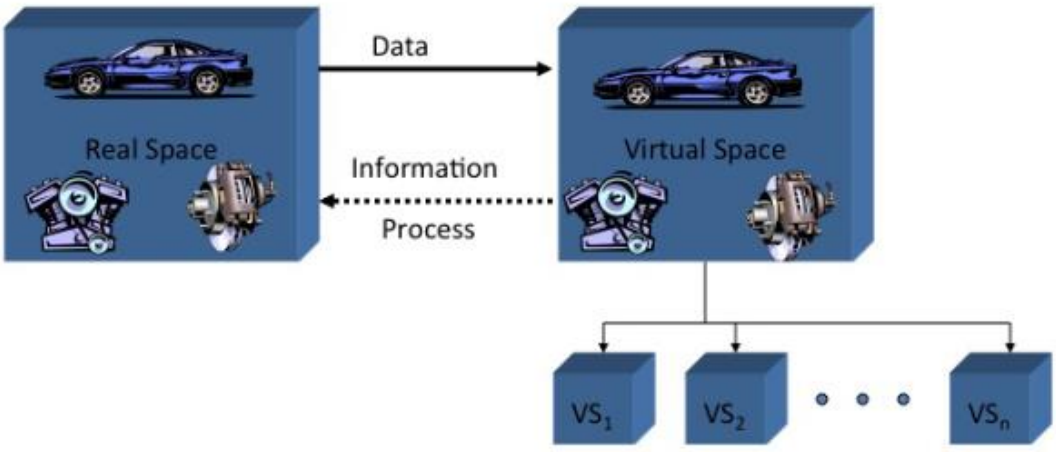


Figure 1. Visual representation of digital twin concept described by M. Grieves [4].

In [5], DT is defined as a software analog of a physical system that mimics the internal processes, technical characteristics, and overall behavior of the system. Lockheed Martin gives the following definition of a DT: “virtual representations of as-built physical assets, processes, and systems that can be used across the product life cycle using real-time data and other sources to provide actual insights” [6]. All in all, most definitions are similar in

describing the idea of the DT – it is a precise virtual clone of a real device or system based on physical properties, gathered or real-time sensor data, intending to simulate its behavior. IBM outlines several key differences that make DT stand ahead of simulations – larger scale (many engineering disciplines studied at the same time) and two-way flow of information (sensor data from the physical device and feedback from the virtual environment of the DT) [7].

2.2 A brief history of Digital Twin technology

The general concept of the DT was first introduced in 2002. Shortly after, it was adopted by the aerospace industry – particularly by NASA and U.S. Air Force. Since 2014, companies such as Lockheed Martin, Boeing, and General Electric were brought together by U.S. Air Force to conduct a series of applied research in the field of DT [8]. The advent of IoT and Big Data has further bridged the gap between physical and virtual worlds and necessitated the development of a sophisticated model to meaningfully process and visualize the physical processes. Altogether, these events have sparked the interest in research of DT technology and, as can be seen in Figure 2, the number of publications has been growing exponentially ever since [9].

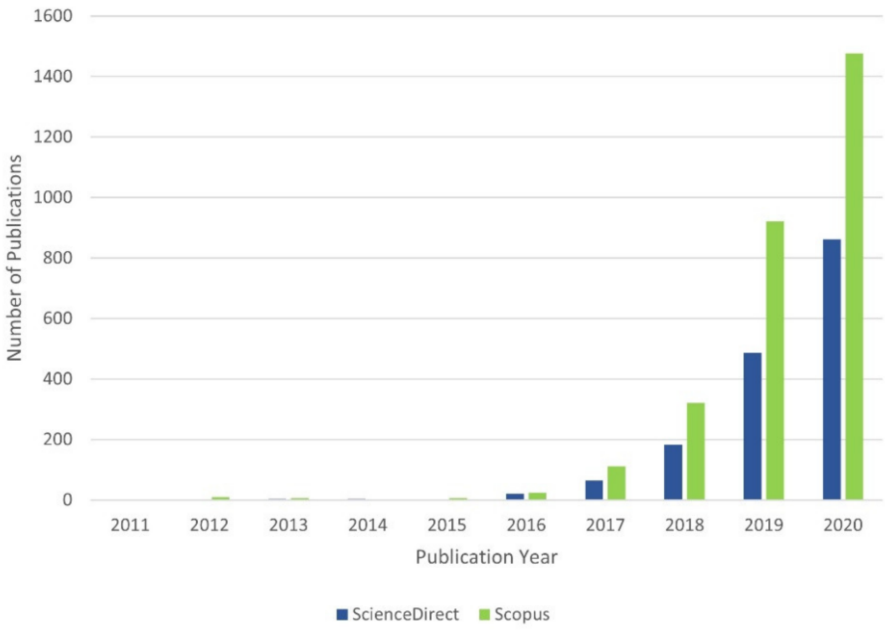


Figure 2. The number of digital twin-related publications by year, from 2011 to 2020 on Scopus and ScienceDirect [9].

The growth and importance of the DT technologies can also be verified by the fact that Gartner has named DT as a strategic technology trend in three consecutive years (2017 - 2019) [10] [11] [12], and Forbes [13] described the DT as one of the defining technologies of next decade.

2.3 Digital Twin technology in the automotive industry

Traditionally, automotive and aerospace systems have been designed with empirical engineering practices [14], but with increasing performance requirements, the necessity for "self-awareness" during operation, and lack of external support, new engineering practices are needed. With the introduction of the DT, new development and testing simulation practices became available to fulfill new requirements, and consequently, the interest in research of these technologies is growing steadily, as can be seen in Figure 3.

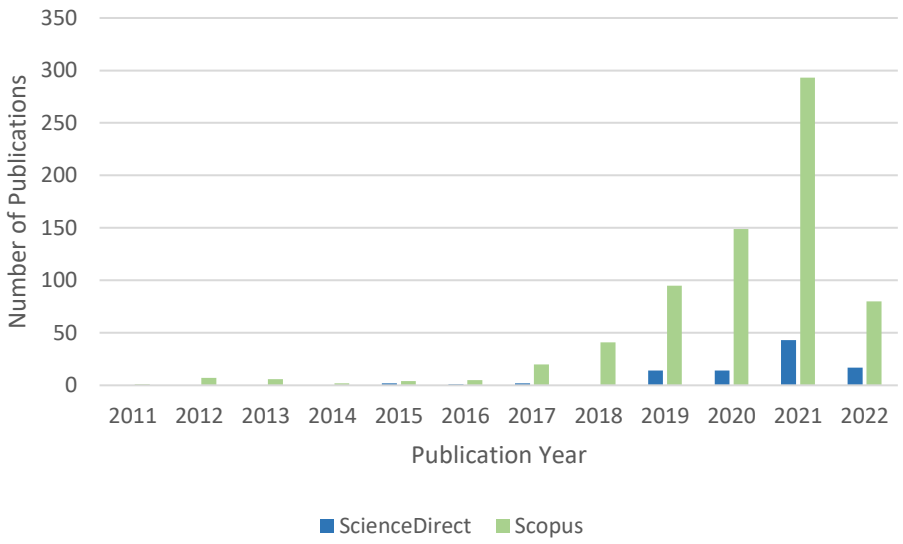


Figure 3. Search[1] results for publications related to digital twins in automotive applications in periods 2011-2022 in ScienceDirect and Scopus.

In [15], Best et al. claim that gained information from vehicle simulations could provide critical training data on algorithmic inefficiencies before actual vehicle testing. As a result, they developed a simulation platform for autonomous driving of a vehicle with the

[1] Search consisted of the following query: (TITLE-ABS-KEY(digital AND twin AND car) OR TITLE-ABS-KEY(digital AND twin AND vehicle) OR TITLE-ABS-KEY(digital AND twin AND automotive)). The last time the search was conducted was on 04.05.2022.

possibility of generating labeled data for machine learning. In their research, simulation covers kinematics and dynamics, traffic rules, path planning, and environmental conditions. However, the research does not cover the simulation of hardware components of the vehicle, solely focusing on the software aspect and AI.

Liu et al. [16] demonstrate two DT models created by two different methods: Gaussian process and convolutional neural networks (CNN). Both DT models were created using the sensor data, collected from a transmission shaft of the vehicle. The simulated measurements were almost identical to those measured on the real vehicle. The conducted study achieved its task of identifying the driving states, but researchers noted that a real-time dual connection between DT and the real vehicle is needed to achieve reliable results.

An improved design was proposed by Chen et al. [17], where scientists developed a hardware-in-the-loop (HiL) simulation platform. The focus was applied on bridging the gap between pure software simulations and hardware simulation, and making the simulation more "online" in nature, by establishing a link between the virtual and real car environments. In their platform, the Electronic Control Unit (ECU) was used for hardware control, with the rest of simulation (such as simulated sensor data, kinematics, and dynamics) occurring in a virtual environment. The simulated data from driving was streamed to the ECU, where the hardware evaluated the state of driving and returned the calculated decisions to the virtual environment.

Using a different approach, Ruba et al. [18] developed a real-time DT implementation using Field-Programmable Gate Array (FPGA) for a propulsion system. In their setup, DT of a propulsion system TB was implemented with two FPGAs: one for mimicking the entire behavior of a Permanent Magnet Synchronous Machine (PMSM) TB, and another FPGA for simulating the control unit. The communication between these two FPGAs was handled by digital and analog IO, utilizing the same interfaces that were used between a PMSM TB and control unit. Therefore, researchers were able to swap the FPGA control unit with the real control unit.

Rassõlkin et al. [3] [19] described a concept of the DT that features three components: a physical entity, a virtual entity, and a service entity. All these entities are interconnected by middleware. The physical entity is represented in a form of the propulsion drive TB, the virtual entity - as the simulated 3D model of TB, and the service entity - as an

integrated service platform responding to the demands of both physical and virtual systems. The study outlines that such DT should provide monitoring capabilities in dynamic regimes.

The abovementioned findings indicate that DT technology is a trending subject of research in the automotive industry and is expected to grow in the upcoming years. Studies are being carried out to make interactions between digital and physical systems more dynamic, occur in real-time, and make simulated operations more identical to those of physical vehicle systems. The benefits of using such systems are reduced cost for carrying out tests and simulations, reduced need for physical testing in the field, and the ability to simulate various scenarios that are difficult to simulate in physical testing.

2.4 Case study - Digital Twin for a propulsion drive system

DT for propulsion drive of autonomous electric vehicle (project number PSG-453) [2] [3] is a research project which aims to develop a specialized unsupervised analysis and prognosis tool of an ISEAUTO PDS, based on DT technology. The design of the proposed DT can be seen in Figure 4.

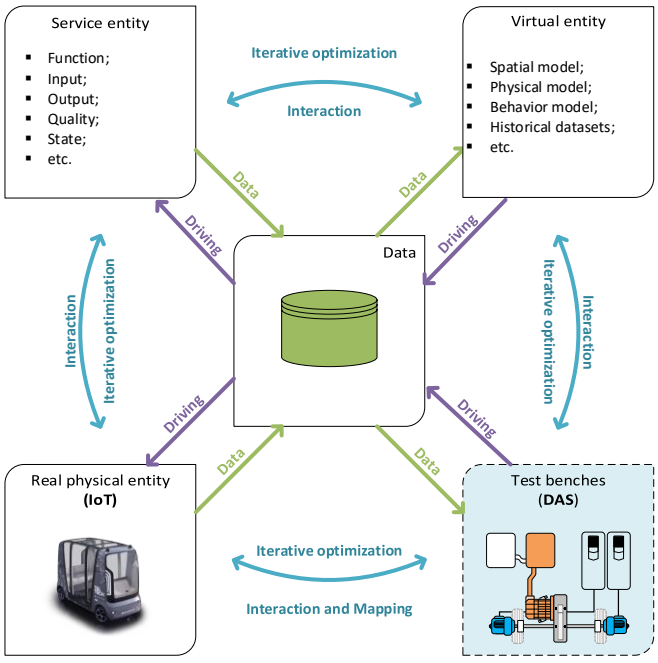


Figure 4. Architecture of the PSG-453 DT showing how the 4 modules are used [20].

The proposed DT consists of four modules: the real vehicle that is supplied with sensors (real physical entity), a test bench (TB) of a vehicle's propulsion drive (designated as Test

benches), 3D models of the TB/real vehicle (virtual entity), the service platform (service entity). All communication is to be handled by a middleware, through which the DT data is going to flow.

The data is being sent and received from all modules to all modules simultaneously, thus enabling the creation of sophisticated controls and analysis of the whole DT in real time. The data generated during the operation of a real vehicle/TB is consumed by the virtual and service entity. Those entities, in response, generate feedback data and other useful parameters that help the analysis of vehicle operation.

To illustrate the process better, assume the following scenario. The PDS TB starts its operation, and the shaft of a motor starts to spin. A sensor installed on the motor records the angular velocity of the shaft and sends it to the virtual entity. The virtual entity calculates the linear velocity of each wheel based on the received shaft angular velocity and forces the 3D model of a vehicle to move. At the same time, friction is exerted on the wheels, causing the vehicle to slow down. The actual recorded linear velocity is recalculated back to the shaft angular velocity and is sent to the PDS to adjust to changes. Meanwhile, the service entity monitors that the data sent by the PDS TB is in the correct range.

2.4.1 Physical entity

The physical entity of the DT is replaced by experimental TB consisting of a PDS identical to the one present inside the ISEAUTO vehicle. The PDS features a Mitsubishi PMSM traction motor Y4F1 (present in i-MiEV car models) which is operated by an ABB HES880 - a frequency converter that transforms the supply power to the motor based on the set parameters. HES880 in its turn is powered by a Cinergia B2C+ battery emulation system. Y4F1 motor’s output is attached to a shaft via a gearbox. The shaft is attached to two ABB IM loading motors (ABB 3GAA132214-ADE) that simulate the loads on the traction motor. Two loading motors are connected to two ABB ACS880 frequency converters that transform the supply power to the motor based on the set parameters. The PDS is attached to a metallic frame which enables the operation of the system and allows the connection of other elements to the system (controllers, converters, sensors, etc). Described TB can be observed in Figure 5 and Figure 6.

Figure 5. TB for PDS in the Electrical Machine Group lab.

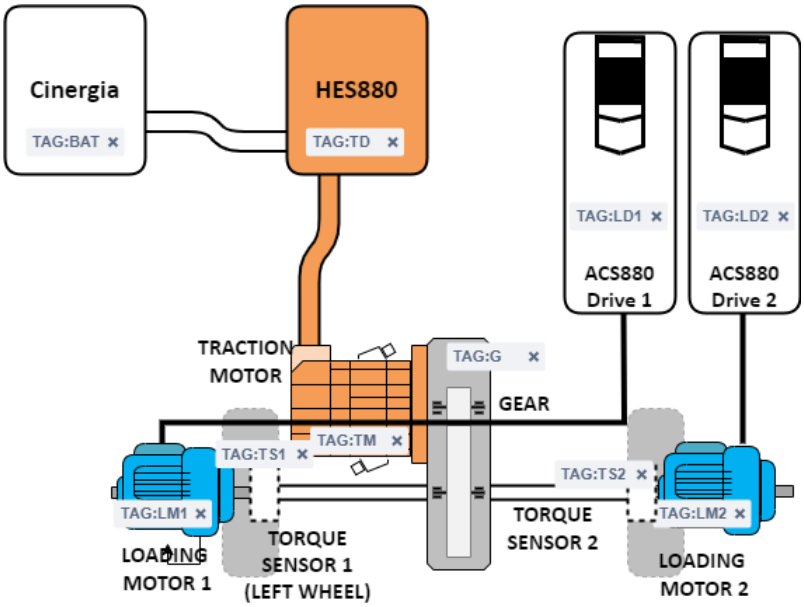


Figure 6. Illustration schematic for TB for PDS.

2.4.2 Virtual entity

As seen in Figure 7, the virtual entity is represented as a 3D model of the TB created in the virtual environment provided by the Unity game engine. The virtual entity is composed of imported CAD geometric models of PDS parts (motors, shafts, bearings, gearbox), thus keeping the real dimensions of the TB. Implemented software in Unity controls the 3D model and can simulate motion and action depending on the provided input. Likewise, the virtual entity can have virtual sensors that record the simulated operation data of the 3D model and stream it back to the physical entity through the middleware.

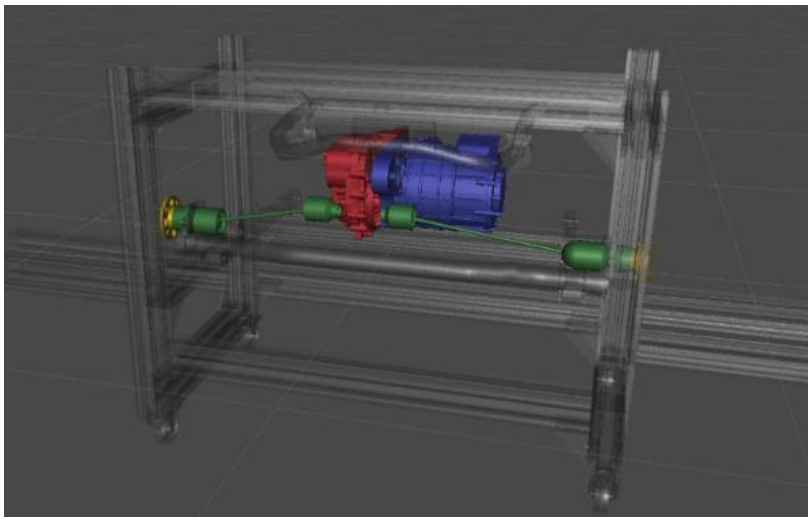


Figure 7. 3D model of the PDS TB in a virtual entity of DT [21].

2.4.3 Service entity

The service system represents an integrated service platform responding to the demands of both physical and virtual systems and acts as a predictive maintenance component [3]. It monitors the operation of TB, analyses any detected abnormalities to find the cause of them, and ultimately warns about problems in the DT. One of the implementations is described in [20] where inter-turn short circuit faults were detected and analyzed in MATLAB software during operation.

2.5 Problem statement

At the time of writing this Master's thesis, the PSG-453 team was in the process of connecting physical entities with their virtual and service entity counterparts. The goal of this Master's thesis is to connect the traction motor of the PDS with a service entity, for the analysis of the traction motor's data.

2.6 Motivation

In any complex system, reliable, scalable, and secure communication between all entities ensures the operation of the system as a whole. In present days, it is observable that many independent technologies that tackle a specific set of problems have begun to be used interchangeably to provide new functionalities. DT technology is one of such fields, and the proposed DT by PSG-453 requires a flexible means to communicate between all the independent technologies.

In this thesis, a middleware framework on a base of ROS2 (Robot Operating System 2) and micro-ROS (micro-ROS) is proposed, through which:

- all entities will be defined in the ROS2-based middleware in a structural manner
- data will be sampled from the physical entity (PDS TB) using micro-ROS
- sampled data will be sent to the service entity
- the service entity will process the data and send it back to the middleware

2.7 Section summary

In this section, the state of the art in DT for automotive applications has been defined. Literature research indicates that interest in DT for automotive applications is increasing every year. Motivations for that are the cost-effectiveness of DTs, advanced maintenance, and analysis of automotive systems. The background for the PSG-453 project and its current state was introduced. Current DT consists of the physical, virtual, and service entities. The problem and motivation for this Master's thesis were outlined.

3 Middleware framework selection

This section provides the definition of middleware, explains the process of selecting the appropriate middleware framework for the needs of described DT, and gives an overview of ROS2 and micro-ROS middleware frameworks.

3.1 Definition of middleware

There is no official definition of the term "middleware", as industry and academics explain this term differently, yet one definition found by the authors explains it the most clearly: middleware platforms are intermediaries between sensors, services, and applications, managing the flow of data and allowing them to interoperate [22]. Middleware handles all the serialization and transfer of information from one platform to another utilizing various applied standards. Middleware has a defined Application Program Interface (API) that allows engineers to bind the middleware software to their parts of the system and allow inter-system communication. Dozens of middleware frameworks are available for use, both proprietary and free of charge. Some of the frameworks are based on standard communication protocols, whereas other frameworks use custom solutions. Different frameworks have different fields of application, ranging from smart homes to aerospace.

3.2 Overview and selection of available middleware

Considering the complexity of elements that constitute a DT (as described in Section 2.4) and the overall application of a DT, it is important to choose appropriate middleware and define an architecture for DT connections. Based on the needs of the TB DT of PSG-453, a set of qualitative criteria based on [22] was outlined:

1. Area of use suitable for industrial cases.

   Applicability for industrial use-cases guarantees that middleware is reliable, possibly standardized, and is capable to handle desired loads of data flow.

2. Support for desired communication model – *publisher-subscriber*.

As described in Section 2.4, data between components is expected to flow (e.g., continuously streamed). *Publisher-subscriber* model is more appropriate for this reason due to its asynchronous nature (communication speeds may vary for each element), and greater scalability if compared to other communication models.

3. Must support real-time operation.

Data from DT must be coming with real-time precision, thus enabling precise analysis of operating PDS.

4. Availability and clarity of documentation.

Concise documentation that is easily available and covers all the information regarding middleware is required to ensure smooth integration into a system.

5. Quality of the support and livelihood of developer communities.

The livelihood of developer communities guarantees that middleware is being improved continuously, reported bug fixes get resolved, and help will be guaranteed if edge cases are encountered.

In a previously conducted study [23], the author has compared the most common middleware frameworks that apply to the investigated study case. The results of the comparison are presented in Table 1. An initial group of middleware frameworks was selected based on their application cases – industrial, automotive, or robotics. From there, it was important to select those supporting the *publisher-subscriber* model. Then, the advantages and disadvantages of all middleware frameworks were considered, and the choice in favor of ROS2 was made. The native support for real-time operation, availability of extensive documentation, and the use of a standardized (DDS) middleware were the key factors taken into account. Also, the liveliness of ROS2 was considered the best as it is an actively developing platform.

Table 1. Comparison of available middleware frameworks [23].

Framework	Initial Release	Type	Messaging Type	Advantages	Disadvantages
ach	2013	Inter-Process Communication mechanism	Message bus Publish-subscribe	+ Real time support + Solved head-of-line problem for accessing the newest message + Extensive documentation	- Inactive community - Development discontinued - No ready software packages
YARP	2002	Robotics middleware	Publish-subscribe	+ Extensible family of connection types + Extensive documentation + Active community + QoS policies	- Limited real time support - No ready software packages
LCM	2006	Libraries and tools for message passing and data marshaling, targeted at real-time systems	Publish-subscribe	+ Distributed network topology + Low-latency inter-process communication + Large support of programming languages	- No ready software packages - Development stalled - Weak documentation - Inactive community
ROS	2007	Robotics middleware	Publish-subscribe	+ Extensive collection of ready-to-use packages + Extensive documentation + Active community	- Limited real time support - Has a master server through which all connections are handled - Support ends in 2025
ROS2	2017	Robotics middleware	Publish-subscribe	+ Real time support + Distributed network topology + Native embedded support + Based on a standard + Active community + Extensive documentation + QoS policies	- Development is still ongoing - Documentation is aimed more at ROS users - Some of ROS ready packages are still being ported to ROS2

In [23], it was also discovered that the reliability of DDS implementation makes ROS2 better at retaining messages and delivering them without losses – in high-frequency communication, the latency was roughly 25 times less, and the number of lost messages was 32 times less in ROS2 compared to ROS. The discovery was made through latency testing – every message contained a header consisting of a unique ID and a timestamp. All publishers inserted an ID into the message with the time of submission. All subscribers were aware of the message ID that they needed to receive. If the IDs matched, the timestamps were compared, and the difference (indicating the latency) was saved. Else, the message was considered lost, and the subscriber would reset the ID to the next expected one. All data was logged into text files and analyzed separately using Jupyter Notebook.

3.3 ROS2

ROS2 is a state-of-the-art framework for robotics development that consists of a large set of free and open-source tools and libraries for robotic engineering, and a structured communication layer.

The communication in ROS2 is realized via a *publisher-subscriber* messaging pattern. Messaging occurs between ROS Nodes which are defined as "processes that perform computation" [24]. ROS Nodes can advertise (produce and send messages) or subscribe (receive messages) to Topics (name buses over which Nodes exchange Messages) [25]. ROS messages constitute data structures made of typed fields [26] to group all the necessary information collected by ROS Nodes. Messages can be default ones provided by ROS packages or they can be custom-defined. Additionally, ROS2 has a *request-response* messaging pattern in form of ROS Services, which is suitable for cases such as one-time requests to complete some operation.

ROS2 also provides a set of GUI and CLI tools for debugging and monitoring. ROS2 CLI tools typically enable users to get information regarding subscriptions and publishers, frequency of submitted Messages, the Message content, etc. The GUI tools allow data visualization – for example, RQt Plot is used to plot the data on a time graph to visualize how data is changing over time. *Rosbags* [27] allow recording ROS Messages on different topics to play this data back later – a feature particularly useful for offline development, and development based on data gathered during real-life operations.

ROS2's predecessor, ROS, was widely used in academia and research for its rich set of documentation and available ROS packages - already developed software components for complex robotic tasks (navigation, localization, computer vision, etc) that are open-source and available to everybody. ROS was well-perceived by the community - it is widely used in academic institutions for education and research. Furthermore, there were some commercial robot platforms developed [28] as well as the largest framework for autonomous vehicles development – Autoware AI [29]. However, as the use of ROS was growing beyond the academic world, it became apparent that ROS must meet a completely new set of demands than it originally was created for. Therefore, the development of ROS2 began with the aim to create a robust platform suitable to operate in real-time, in non-ideal network conditions, and be possible to use on embedded

devices. ROS2 is developed and managed by Open Robotics, with some parts of the software being co-developed by renowned industry leaders in the automotive and technology sectors (such as Bosch, Sony, AWS, iRobot, etc) [30].

3.3.1 ROS2 architecture

The communication in ROS2 is based on the Data Distribution Service - a middleware protocol and API standard for data-centric connectivity from the Object Management Group (OMG). It provides reliable, low-latency, and real-time communication. Its key feature – dynamic discovery and Quality of Service (QoS) makes it server-free and more extensible [31]. In large systems with multiple communicating elements, DDS solves the problem of scalability and speed, providing a fast network. DDS is well-standardized [32], and has been a part of other time-critical standards used in automotive, aerospace, and defense industries (e.g NATO NGVA, AUTOSAR Adaptive) [33].

ROS2 is structured as follows: the user application layer is used for writing software for ROS Nodes. The user application layer relies on the ROS2 Client layer, which provides users with the language-specific (C++, Python, C) API for ROS2 core libraries and functions. The client layer is connected to the DDS Abstraction layer which binds ROS2 with DDS implementations. The communication is handled entirely by various DDS vendors on the DDS Implementation Layer. ROS2 entirely resides in operating systems. The visual representation of described architecture can be seen in Figure 8.

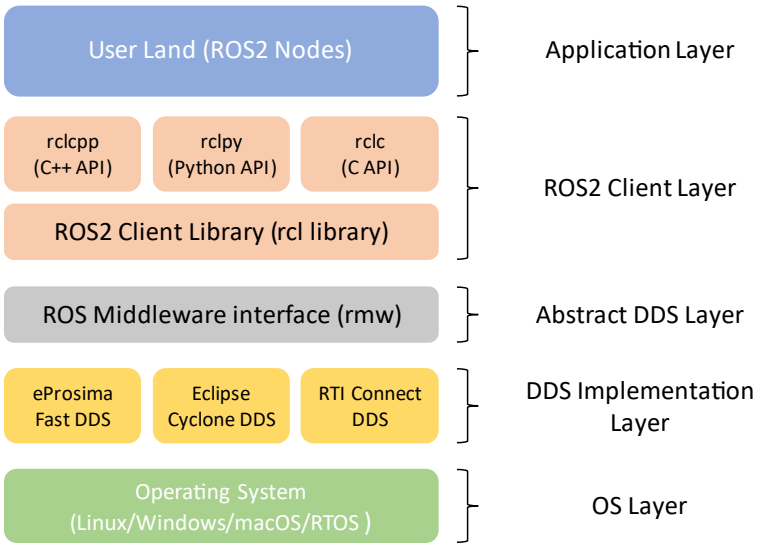


Figure 8. ROS2 Architecture.

User-written software can be grouped into ROS Packages and compiled using *colcon* – a process automation CLI tool for building sets of software packages. Compiled software can be launched by a standalone execution, or it can be executed in series according to specified logic, using *launch* files.

3.3.2 micro-ROS

To gather information from sensors and influence the operation of the TB, hardware interfacing is required. The simplest hardware for this purpose would be microcontroller units (MCUs) – compact integrated circuits designed to control specific operations in embedded systems. Typically, MCUs would be used for low-level operating, control, and data gathering with an interface to a higher-level governing system. For instance, the OS manages the access and use of resources to the user and is interfaced with hardware. Hardware, on the other hand, has its firmware that manages energy, internal sensors, etc. MCUs vary, they have different resources and are equipped with different base software available.

Default distributions of ROS2 are not optimized for use on microcontrollers or real-time OS. Community-developed solutions to tackle these problems exist, namely *rosserial* [34] and *mROS* [35], but they have a very limited set of features, and their development is discontinued. Considering the features available in ROS2 (QoS, security) and ambitions to support real-time operation, a micro-ROS (micro-ROS) project has been established [36].

micro-ROS is a microcontroller-optimized ROS2 distribution that supports all the main features of ROS2 in resource-constrained environments and can be seamlessly integrated with ROS2. It is the de-facto standard ROS2 approach for embedded systems, developed by Bosch GmbH [37]. micro-ROS aims to bring support to a wide set of microcontrollers, but for now, there is a limited set of officially and community-supported MCU platforms. Additionally, Bosch GmbH provides instructions on how to compile micro-ROS on yet unsupported MCU platforms that meet the minimal hardware requirements.

It is important to note, that micro-ROS is still in active development, and it has not yet been officially standardized for production use.

3.3.3 Difference between ROS2 and micro-ROS architectures

As described in [36], the executors present in ROS2 (*rclcpp*) and micro-ROS (*rclc*) are different. *rclcpp* executor requires dynamic memory allocation, which cannot be used on many microcontrollers. Additionally, *the rclcpp* library was not created for resource-constrained environments and thus it is not optimized to fit the small memory of MCUs. Furthermore, the rclc executor features deterministic scheduling and execution and real-time guarantees [38].

If in ROS2 the choice of DDS Implementation is available to a user, in micro-ROS it is fixed to eProsima Micro XRCE-DDS - a software solution that allows communication in extremely resource-constrained environments (in this specific case - MCUs) with an existing DDS network [39].

Contrary to ROS2, the choice of OS (if available) is limited only to RTOS that can operate on MCUs. Currently, supported ones are Zephyr, FreeRTOS, and NuttX [36].

Other features, such as Node discovery and bridge between micro-ROS Nodes and DDS middleware are resource-hungry and are implemented in a ROS 2 Agent – a separate piece of software that is meant to run on the host where ROS2 is running. It supports Serial, UDP, and Bluetooth connection with MCU.

The abovementioned differences can be observed Figure 9 which illustrates the full architecture of micro-ROS.

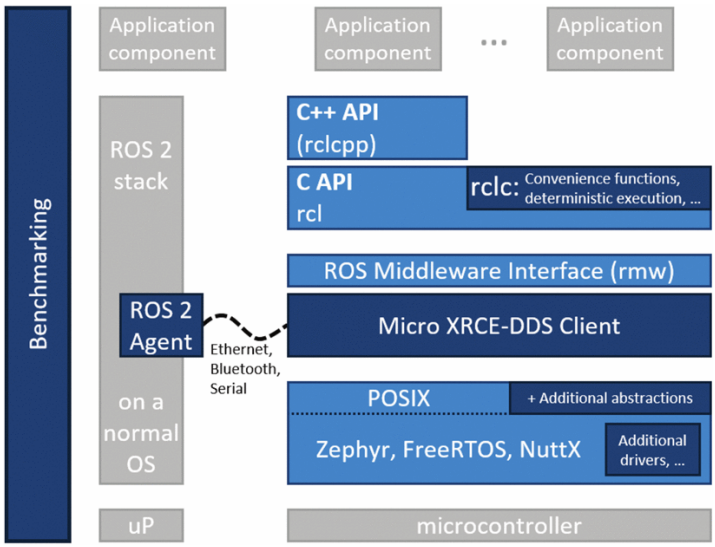


Figure 9. micro-ROS architecture [36].

3.4 Section summary

In this section, the definition of middleware was given. An appropriate middleware framework for purposes of DT was selected. The selection process included a comparison of available middleware frameworks that satisfy the given criteria and a comparison of performance between ROS and ROS2. Definitions and architectures of ROS2 and micro-ROS frameworks were given.

4 Middleware implementation for studied Digital Twin

This section provides the description and requirements of the middleware implementation. The details of the middleware interface to hardware and software are given. The method of gathering motor data using an MCU is explained.

4.1 Desired operation and requirements

To implement the middleware for the DT of PDS, it is required to first determine what it should be interfaced with, and which operation must be performed through it. In the scope of this Master's thesis, only several components of PDS are chosen for detailed study: the HES880 frequency converter and the traction motor. The studied part of PDS TB can be seen illustrated in Figure 10.

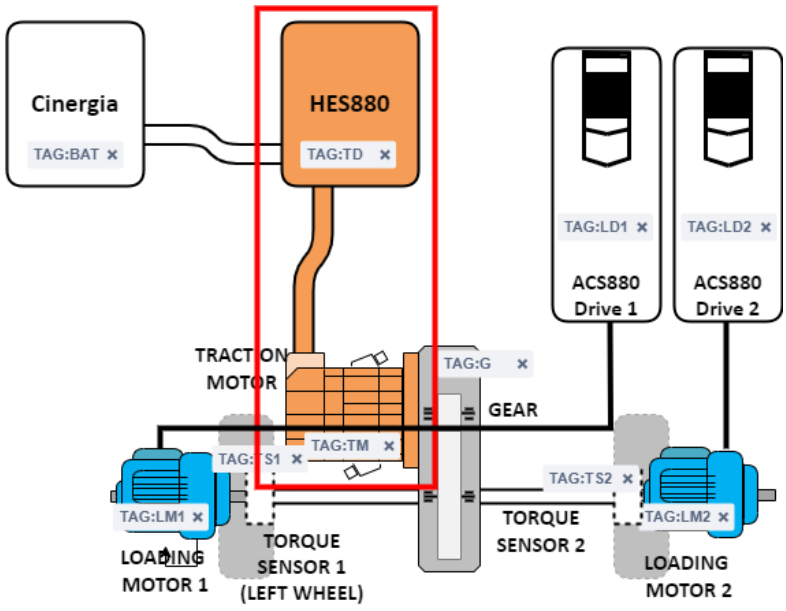


Figure 10. Illustration of studied components in TB for PDS. Studied parts are enclosed by a red frame.

The desired operation to be fulfilled for the abovementioned components of PDS TB is defined as follows:

1. All entities and their subsystems and components must be grouped and structurally represented in the middleware.

2. The DT middleware must receive the data regarding *the supply input power* to the Mitsubishi PMSM traction motor Y4F1.

The supply input power is defined as a 3-phase voltage and current that is generated by a frequency converter attached to the motor. Based on input configurations, the frequency converter modifies the power supply that is then supplied to the motor, causing it to work. The power supply modified by the frequency converter must be sampled and sent into the middleware.

3. This data must be conveyed to the service entity to calculate the motor's output parameters.

The service entity, upon reception of data, must extract the following parameters using analytical model of the traction motor: angular velocity and torque of the traction motor.

4. Calculated torque and angular velocity must be sent into the middleware.

Parameters calculated by the service entity must be present in the middleware for other entities.

4.2 Proposed solution

Considering the described operation and requirements presented in Section 4.1, the following solution is proposed:

1. An MCU with micro-ROS installed will be connected to the output of frequency converter HES880 to sample the data at a 1 kHz frequency.

2. The sampled measurements are serialized into ROS messages and sent to the middleware (via a micro-ROS agent hosted on a separate machine) to the designated topic.

3. The service entity connects to the middleware by subscribing to the designated topic and processes the incoming data to calculate the angular velocity and torque of the traction motor.

4. The service entity publishes the resultant angular velocity and torque to the designated topic, thus sending the data back to the middleware.

The proposed solution is illustrated in Figure 11.

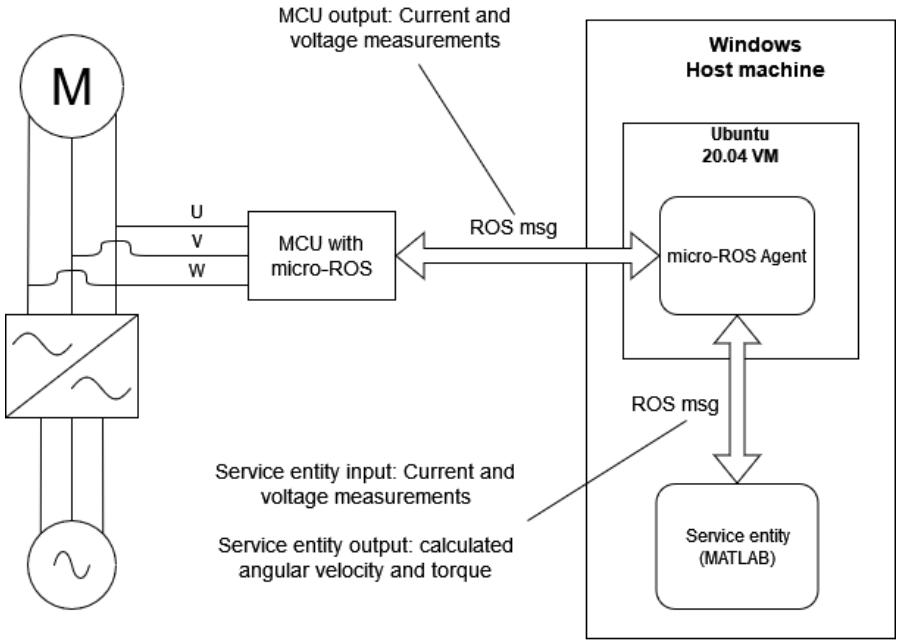


Figure 11. Draft of a proposed solution. U, V, W indicate the 1,2 and 3 phase voltage and current.

In the proposed solution the middleware coverage ranges from a host Windows 10 machine to a microcontroller. Due to the micro-ROS agent requirements, it executes on a guest Ubuntu 20.04 virtual machine (VM). The service entity executes in a MATLAB run-time environment installed in Windows 10. Defined ROS2 messages and interfaces between entities are described in Section 4.3, service entity is described in Section 4.4, and used hardware elements are described in Section 4.5.

4.3 Structure of ROS2 middleware

Inside the DT it is expected that components, parts, and subsystems are going to communicate with each other. The *publisher-subscriber* topology allows for flexible communication between them. But the problem that arises is – how does a component know which information it is supposed to receive? Considering the design of physical entity TB (as described in 2.4.1), we will have a total of three motors – one as part of PDS and two loading motors to simulate the load. These three motors communicate the same

information (torque, angular velocity, power, etc), but they must be differentiated. The same applies to whole subsystems – there may be similar sets of information flows, but they may be required inside the subsystem only, without exposure to other subsystems. To solve this, context is required for every ROS Node, ROS Message, and ROS Topic.

4.3.1 Naming requirements

To differentiate between subsystems in the DT, hierarchical naming and grouping should apply to every component of the DT.

ROS2 provides a flexible naming configuration that helps developers to design modular components of their ROS2 applications and for others to be able to easily integrate them. Although every component is required to have a pre-defined name for a Node or Topic, it can be renamed, mapped, or grouped by any name defined by the user. The names can be of two types: relative and global. A global name would indicate a completely specified name for a Node or Topic, and it cannot be modified. Relative names can be supplied with a namespace during configuration and launch – which makes it possible to have the same Nodes grouped under different names.

In our middleware design, all names are expected to be relative and specified with a namespace indicating a group at a launch time. This will allow modular development and reuse of DT components. Thus, every component will have a default relative Node name (indicating which component it is generally) and Topic names (indicating the generic parameters it communicated with), and upon launch time these components are grouped by a namespace according to the naming requirements of PSG-453 project that can be shown in Table 2.

Table 2. Namespaces used for grouping components of the DT.

Name of the namespace	Components to be used for
/tb_tm	Traction motor components, torque, angular velocity, power calculating nodes, any hardware connected to them.
/tb_lm_left /tb_lm_right	Any loading motor components: torque, angular velocity, power calculating nodes, any hardware connected to them. Left and right specify exactly which loading motor in the physical entity it is.
/tb_service	Service entities used for the analysis of the TB PDS, warning systems.
/tb_virtual	Components of the visual entity that are interfaced to ROS Middleware.
/tb_bat	Components related to the battery that is used to simulate operating battery in ISEAUTO.
/tb_td	Components related to frequency converter (traction drive HES880 used to control traction motor).
/tb_ld_left /tb_ld_right	Components related to frequency converters (ACS850 used to control loading motors). Since each frequency converter can control only one motor, they are designated left and right per loading motor they control.

It is important to note that namespaces are generally applied to Nodes that are associated with the component of DT they represent, and topics they would send the data to would include the Node's namespace. However, it is possible for Nodes of one group to require data from Nodes of other groups.

To better illustrate the latter, assume there are two Nodes: */tb_tm/left_shaft_consumer* and */tb_lm_left/torque_producer*. */tb_tm/left_shaft_consumer* Node is expecting to receive the torque that left loading motor exerts on it. In this case, */tb_tm/left_shaft_consumer* Node would subscribe to a topic published by */tb_lm_left/torque_producer* Node (e.g. */tb_lm_left/torque*). In this case, it is logical to assume that the exerted torque is a part of loading motor, rather than the traction motor's shaft.

4.3.2 ROS2 messages definitions

Section 3.3 described ROS messages as custom or standard data structures made of typed fields to group information provided by Nodes. For DT, custom messages were defined to group several signals and/or parameters together that are related by time and context. Every message contains a header that records the time of submission and a unique ID of the message. All messages were included in a separate ROS Package *digital_twin_msgs* that is required by the middleware to operate the DT. Apart from custom-defined definitions, the middleware uses standard ROS messages (std_msgs) [40] where necessary. Message defined in ROS package *digital_twin_msgs* can be seen in Table 3, and the structure of each message can be observed in Appendix 2.

Table 3. Messages of DT defined in the digital_twin_msgs package.

Name of the message	Description
digital_twin_msgs::Current	A message consisting of 3 phase currents values. Used to store information about AC current.
digital_twin_msgs::Voltage	A message consisting of 3 phase voltages values. Used to store information about AC voltage.
digital_twin_msgs::SupplyInput	Message comprised of Current.msg and Voltage.msg with a timestamp. Used as a container structure to communicate the AC input of the motors.
digital_twin_msgs::Power	A message consisting of power values of a 3-phase AC input at every phase and total mean. Includes a timestamp. Used to store information about AC power.
digital_twin_msgs::Float32Stamped	Generic float data-type message with a timestamp. Can be used for any topic requiring a generic float type data container.

4.4 Used service entity

The service entity in use for the objective is an analytical simulation model of the traction motor which was built in MATLAB/Simulink interfaced with middleware. The simulation model was developed by the author's colleague for a separate research problem, as presented in [41]. Hence, this model is not in the scope of this Master's thesis and will be treated as a black box. The model's purpose is to calculate the traction motor's output torque and angular velocity based on the input voltage of the motor. The

calculation is based on derived analytical equations of the electromagnetic properties of the traction motor.

The input voltage of the model is the input voltage of the traction motor generated by the HES880 frequency converter and is expected to be received by the service entity in real time. A *ROS2 Subscriber* MATLAB block is used to connect the service entity to middleware for voltage data reception, and a *ROS2 Publisher* MATLAB block is used for sending angular velocity and torque data back to middleware. When received, voltage input is deserialized using a *Bus Selector* MATLAB block and is directed into the model. When finished processing, the model outputs angular velocity and torque parameters; in combination with the *Blank Message* MATLAB block, these parameters constitute a new ROS message that is then published via a *ROS2 publisher* block. The described service entity can be seen in Figure 12 and its interface with middleware is shown in Table 4.

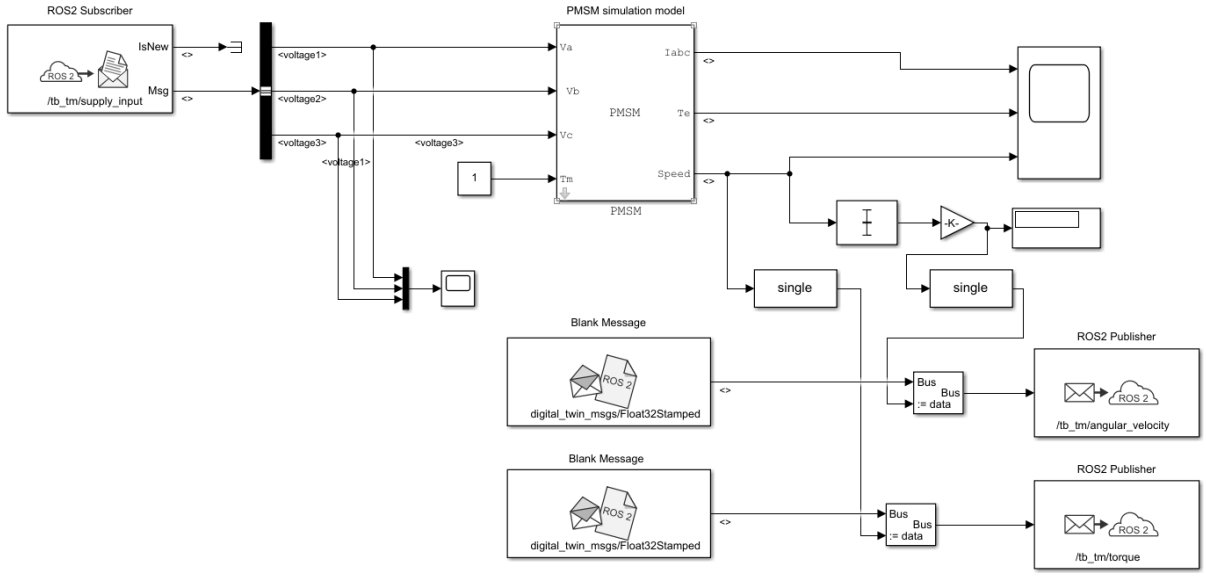


Figure 12. MATLAB/Simulink block diagram of used service entity.

Table 4. Middleware interface of service entity.

Topic	Message type	Description
/tb_tm/supply_input	digital_twin_msgs::SupplyInput	Topic to subscribe. Used to receive voltage of traction motor from the middleware.
/tb_tm/torque	digital_twin_msgs::Float32Stamped	Topic for publishing. Used to send the calculated torque to the middleware.
/tb_tm/angular_velocity	digital_twin_msgs::Float32Stamped	Topic for publishing. Used to send the calculated angular velocity to the middleware.

4.5 Hardware interface between TB and middleware

To gather the data from the HES880 frequency converter and direct it to middleware, there must be hardware that serves as an interface between these two entities. In our case, it must be an MCU capable of reading analog data, have a peripheral interface able to communicate via Serial/USB and be possible to run with micro-ROS.

One of the aims of micro-ROS is to provide support for a large number of families of microcontrollers. Although this is a large and complicated task when this Master's thesis was written several microcontroller families were already supported [42]. This meant, that there were tools for compilation of micro-ROS to targeted microcontrollers and official manuals assisting in this matter. Considering the requirements, the officially supported microcontrollers by micro-ROS, and the availability of the latter on the premises of Tallinn University of Technology, a choice was made to proceed with Teensy 4.0.

Teensy 4.0 [43] is a small ARM family microcontroller. It features a 600 MHz ARM Cortex-M7 processor, with 1024kB of RAM and 1984kB of Flash memory with USB peripheral supporting speeds up to 480 Mbit/sec. It features 40 GPIO pins, 14 of which can be configured as analog input pins. Teensy is programmable through Arduino IDE with an installed Teensyduino add-on. Teensy 4.0 is illustrated in Figure 13.

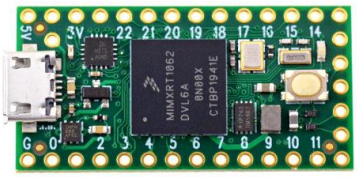


Figure 13. Teensy 4.0 microcontroller unit [43].

To fulfill the operation described in Section 4.1, Teensy 4.0 with micro-ROS software will gather the data from the HES880 frequency converter and send it to the middleware via a micro-ROS agent.

4.5.1 Teensy 4.0 with micro-ROS

To get micro-ROS running on Teensy 4.0, the official tutorial from micro-ROS [44] with some additional steps was followed. In the tutorial, it is suggested to download the already pre-compiled micro-ROS library for microcontrollers and just copy it to the Arduino IDE library folder. For the desired operation, however, support for *digital_twin_msgs* must have been provided, thus additional steps were required. For this to happen, the micro-ROS library was recompiled according to instructions from the official Github repository [45]. The following steps were done:

1. Download and install Arduino IDE and Teensyduino add-on.

2. Download the micro-ROS Arduino source library for ROS2 foxy distribution.

```
git clone git@github.com:micro-ROS/micro_ros_arduino.git
git checkout foxy
```

3. Add d*igital_twin_msgs* package to a folder */extras/library_generation/extra_packages* of micro-ROS Arduino library.

4. Compile the micro-ROS Arduino library for Teensy 4.0.

```
sudo docker pull microros/micro_ros_static_library_builder:foxy
sudo docker run -it --rm -v $(pwd):/project --env
MICROROS_LIBRARY_FOLDER=extras microros/micro_ros_static_library_builder:foxy
-p teensy4
```

5. Copy the contents of */src* into the Arduino IDE's library folder.

Afterward, micro-ROS API becomes available for use in Arduino IDE and enables to write, compile and flash written software with *digital_twin_msgs* messages included in Teensy 4.0. As soon as the software is flashed and is working correctly, Teensy 4.0 must be connected to a computer with a micro-ROS agent via a USB. By default, the connection is plug-and-play and the micro-ROS agent should detect new microcontrollers automatically. However, since the micro-ROS agent is running on a VM, USB support had to be enabled in the settings of a VM. When the micro-ROS agent detects a new connected microcontroller, it becomes available in the whole middleware and the communication (data sending and reception) starts automatically.

4.5.2 HES880 frequency converter

ABB HES880 [46] is a mobile frequency converter for controlling asynchronous AC induction motors. In the case of TB, HES880 controls the Mitsubishi traction motor. The HES880 consists of 2 parts: the drive module and the control module. Based on the selected parameters in the control module, the HES880 modifies the supply AC voltage and frequency into AC motor input.

The frequency converter directly dictates the operation of an electrical motor it controls by supplying AC voltage to the motor. Knowing this, it is possible to measure the output of the frequency converter directly and then forward it to the middleware.

4.5.3 HES880 output measurement

The output of HES880 is AC current, and appropriate electronics were required to transform the AC current into a positive-only (larger than 0V) periodic voltage signal in a range of 0 – 3.3V, for Teensy 4.0 MCU to sample it. Signal conversion and electronics design were done by the author's colleague who had the required knowledge and skills to solve this problem. Therefore, the electronics and signal conversion will be treated as a black box solution and is out of the scope of this Master's thesis. Nevertheless, a short description will be provided to explain the general idea of how the signal conversion is done.

Three devices, known as current clamps, are attached to the cables that connect HES880 output terminals with the Mitsubishi traction motor's input terminals. Depending on the configuration and wiring, the current clamps can measure voltages and currents and

output both as voltage signals. These devices have conversion ratios (also known as scale): 1mV/A for current (that is, every 1mV clamp output represents 1 measured Amp of input current), and 10mV/V for voltage (that is, every 10mV clamp output represents 1 measured Volt of input voltage). The input signals generated by HES880 were in the range of +350A to -350A for current and +500V to -500V for voltage. However, for the selected operation of HES880, the generated current and voltage would not exceed ranges +200A to -200A and +25V to -25V, respectively. As a result of conversion from current clamps, the input signals of current and voltage are scaled to: +200mV to -200mV and +250mV to -250mV, respectively. Because most ADCs (analog-to-digital converters) present in MCUs (including Teensy 4.0) can only process positive analog signals, the output signals of current clamps must be brought to the positive-only range. For this, a level shifter was used that lifts the signal by 1 V.

Teensy 4.0 MCU features 14 analog input pins that can be used to sample the data. Measurements of 3-phase AC current and voltage would require 6 analog inputs. A0 – A5 were used to sample the data, A0-A2 for current and A3-A5 for voltages. The frequency of AC current is estimated to be around 20 Hz, therefore input AC signal is sampled at 1 kHz frequency, eliminating the possibility of aliasing. Teensy 4.0 MCU has a 10-bit ADC (input range 0 – 1023 bits) that can measure voltages in the range 0 – 3.3V, which means that the resolution of the ADC is approximately 3.22 mV. For conversion of bits to voltage in mV, Equation 1 was used:

$$U_{in} = \frac{N_{bits} \times 3300}{1024} \tag{1}$$

Equation 2 and Equation 3 show the conversion of acquired voltage to real measured voltage and current, respectively (1000 was subtracted to bring the measured voltage back to its original range; vt_scale is the voltage scale factor and is equal to 10 mV; ct_scale is the current scale factor and is equal to 1 A):

$$U_{measured} = \frac{U_{in} - 1000}{vt\_scale} \tag{2}$$

$$I_{measured} = \frac{U_{in} - 1000}{ct\_scale} \tag{3}$$

With everything considered, a script that handles data sampling, serialization of data into ROS messages, and transport to the middleware was written, as presented in Appendix 3. The script was compiled using Teensyduino IDE and flashed onto Teensy 4.0 MCU.

4.6 Section summary

This section covered the details of middleware implementation for a given problem. The desired operation of a DT entity was described, and a possible solution was proposed. All communication details were covered: interfaces between hardware and software components, subscribed and published topics, and used messages with the data they contain. An overview of used hardware components was given and the data gathering method was explained.

5 Results

This section covers the achieved results of the conducted work. The sampled data is shown, along with the results of latency tests. The section explains achieved results and provides suggestions for future work.

5.1 Acquired data

Acquired data by Teensy 4.0 MCU was sampled at 1 kHz frequency and converted to raw voltage, as shown in Equation 1. The raw voltage measurement was then used to calculate the real current and voltage as described in Section 4.5.3 in Equations 2, 3 and the resultant measured current and voltage signals were serialized into ROS2 *digital_twin_msgs/SupplyInput* message. As the last step, the messages were published on ROS2 topic */tb_tm/supply_input*. The operation was recorded by ROS2 as a *rosbag* and analyzed in MATLAB. The results of measured AC voltage and current can be observed in Figure 14.

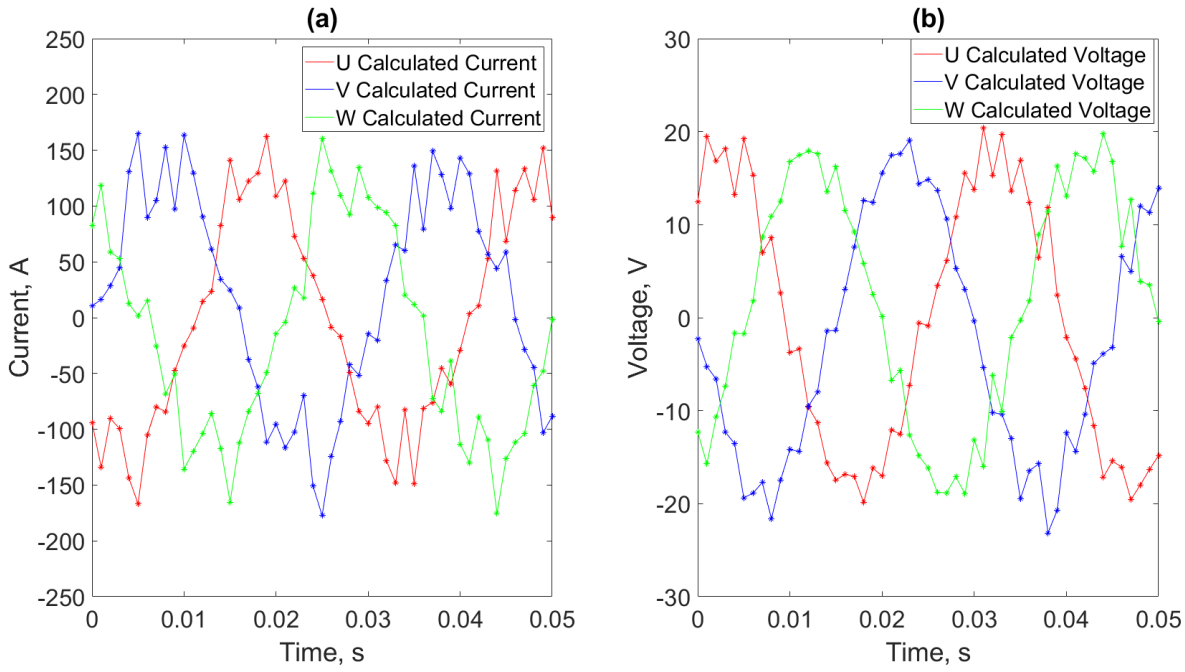
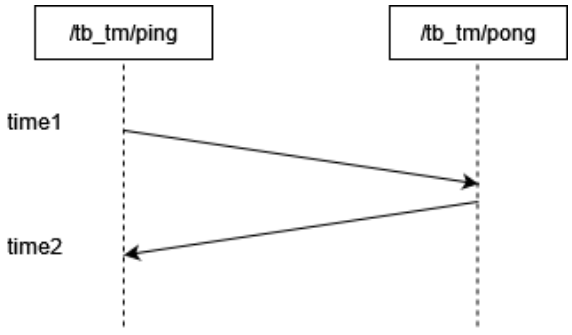


Figure 14. Real measured current and voltage as a result of conversion: a) measured AC current, b) measured AC voltage. U, V, W are designations for every phase in AC current.

Acquired measurements of current and voltage appear to be noisy, but the overall representation of sine waves of voltage and current are kept, thus aliasing was avoided. There is a multitude of factors that can be the cause of the noise: losses in precision from sampling (as the ADC has 10-bit precision), noise from the level shifter caused by oscillation of the shifting signal, the interference from the environment and the signal quality produced by the HES880 frequency converter itself. To smooth out the signal, the service entity uses a second-order filter implementation before feeding the voltage signal to the main model.

5.2 Latency test

To validate that implemented solution can be used in real-time, a latency test was conducted. For this specific case, RTT (round trip time, visual representation can be seen in Figure 15) latency test was chosen, due to MCU and host machine possessing different clocks. Different clocks may not be properly synchronized, leading to false results. Furthermore, virtual machines specifically are subject to an occurrence known as clock drift. Typically, VMs synchronize their clock with the host machine every 60 seconds and therefore may "lag behind" the host system.



$$latency = \frac{time2 - time1}{2}$$

Figure 15. RTT latency test visualization.

To conduct the RTT latency test, a new message type was defined consisting of a message ID and time stamp. Every message was generated by Teensy 4.0 MCU and sent to a ROS2 listener, running in the MATLAB run-time environment. The listener, upon receiving the message, verified that the message was not lost (by comparing the expected message ID

with the received one) and simply sent it back to the Teensy 4.0 MCU. When the MCU received messages back, it calculated the approximate time it took for a message to return. To avoid running out of memory, the Teensy 4.0 did not store the latencies locally and forwarded them to a specially created ROS2 Node on a host machine that later calculated mean, maximum, and minimum latency. The test was conducted for 60000 messages. The latency was measured in microseconds since the clock of Teensy 4.0 is capable of recording time only with microsecond precision. Written scripts for the latency test can be seen in Appendix 4, 5, and 6.

Latency test between Teensy 4.0 MCU and service entity yielded the following results: only 560 messages out of 60000 were received, with mean latency being 350 μs, maximum latency being 7569 μs, and minimum latency being 92 μs. As investigation showed, it was not the fault of middleware, but MATLAB/Simulink software itself. MATLAB/Simulink was unable to receive data at high frequencies and was forced to drop messages, leading to a low rate of successfully delivered messages.

Such operation cannot be considered reliable, and it can be concluded that MATLAB/Simulink solutions must be changed to be capable of receiving high-frequency data.

5.2.1 Suggested improvement to the service entity

Even though MATLAB cannot process software in real time, it has a code generator that can transform various models into lower-level programming languages for target devices. Essentially, MATLAB/Simulink code generator establishes a connection with the target device, transforms the model into a C++ code, and attempts to compile it using the default compiler for ROS2. Therefore, the goal was to use the model of the service entity to generate a ROS2 Node C++ code, with VM being the target device. After following the manual on MATLAB/Simulink code generation [47], the generation succeeded, and the model was available as a ROS2 node in the VM.

The same RTT latency test was conducted to calculate the approximate latency of the solution. The results were indeed better: all 60000 messages were successfully delivered, mean latency was 197 μs, maximum latency was 6594 μs, and minimum latency was 151 μs. The increase in reliability suggests that using lower-level code for processing data is

more preferred in the scope of the proposed DT. For comparison, both latency test results can be observed in Table 5.

Table 5. Results of conducted RTT latency tests

Operation environment	Messages sent (#)	Messages lost (#)	Mean latency (µs)	Maximum latency (µs)	Minimum latency (µs)
MATLAB run-time on Windows host	60.000	59.440	350	7569	92
Compiled C++ program on Ubuntu VM	60.000	0	197	6594	151

5.3 Overview of conducted work and final solution

As a result of implementations described in Section 4, the following tasks were done:

1. Traction motor input data was sampled from the HES880 frequency converter by Teensy 4.0 MCU.

2. The software for Teensy 4.0 MCU was written using the micro-ROS framework. The software handled data sampling, serialization, and transport to the middleware via a micro-ROS agent.

3. The service entity was interfaced with the middleware to receive and send the traction motor data.

4. A latency test was conducted to estimate the reliability of the solution.

5. Conversion of MATLAB/Simulink model to C++ was made as a possible way to fix unreliable data reception by the service entity.

The achieved result satisfies the operational requirements presented in Section 4.1. ROS2 framework proves to be quite flexible for designing systems and implementing the intra-communication between the components of a system. Its internal implementation of the DDS standard provides a reliable means to communicate in a peer-to-peer manner. Custom message definition, contextual grouping using namespaces and provided API are optimal for such fields as DT technology. The micro-ROS framework provides an out-of-the-box approach for connecting microcontrollers to ROS2. However, supported

hardware is still limited and some operational requirements for micro-ROS are yet to be fulfilled. MATLAB/Simulink computational abilities were found to be unreliable and conversion to C++ had to be made to improve the communication between the middleware and the service entity.

5.3.1 Suggestions for future work

To improve the overall design of the DT, the following improvements are suggested:

1. The service entity components that require high-frequency communication and/or real-time operation must be migrated to lower-level implementation, such as C++, Python, or a similar language/platform.

2. For increased precision of the DT, it may be necessary to utilize communication protocols like SPI or I2C between MCU and the middleware. This will increase possible messaging frequency.

3. In the future, TB may have a very large number of connections to the DT, and microcontrollers may not be the optimal way to interface these connections. A larger module/router would be required in this case.

4. Electronics that handle signal processing may need to be of higher precision to eliminate noise.

5.4 Section summary

This section provided an overview of conducted work. Latency tests revealed that the service entity implementation in MATLAB/Simulink was very unreliable when it came to receiving data. A solution to mitigate this problem was provided. Overall, the latency between the MCU and the service entity is low enough to be considered real-time. The final solution was presented, featuring all the interfaces between the components of DT. In the end, the author provided suggestions for future work to improve the state of the DT.

6 Summary

Digital Twin (DT) technology is a trending technology in the automotive field that allows advanced analysis and testing of such complex systems. Autonomous vehicles are a special case – the possession of large amounts of sensors and processing capabilities allows a very in-depth study of the internal workings of the vehicles, but very little is done towards the understanding of how these autonomous vehicles are affected during operation. For this reason, DT for propulsion drive of autonomous electric vehicle (project number PSG-453) was established. The project aims to develop a DT for the propulsion drive system of ISEAUTO – a self-driving vehicle being developed by Tallinn University of Technology since 2018.

In recent years the DT technology in the automotive field has seen a spike in publications and various methods are actively proposed and discussed. The latest developments indicate interest in creating high-precision DTs for hardware components of the vehicles in an attempt to create cost-effective, in-depth analysis systems.

The goal of this thesis was to connect two entities present in DT architecture, proposed by PSG-453: a traction motor from the physical entity with the analytical model of the motor from the service entity. The connection had to follow the implementation of middleware – a special software layer that handles all the communication between all the entities of the DT system. The chosen middleware framework – ROS2, was described in terms of architecture and capabilities.

As a result, the traction motor was interfaced with ROS2 middleware via Teensy 4.0 microcontroller that uses micro-ROS – a ROS2 framework for embedded devices. The analytical model of the motor developed in MATLAB software was interfaced with the middleware using the provided ROS2 API. The acquired results were presented and analyzed. The latency test shows that the implemented solution operates in real time. At the same time, the latency test suggested that the service entity had to be run outside the

MATLAB run-time environment due to low reliability, hence an improvement was made to overcome this issue.

Based on the results of this Master's thesis, a conference paper was written to describe the used approach for connecting DT entities and describing the achieved results.

References

[1] A. Rassõlkin, R. Sell and M. Leier, "Development Case Study of the First Estonian Self-Driving Car, ISEAUTO," *Electrical, Control and Communication Engineering,* vol. 14, pp. 81-88, 07 2018.

[2] Estonian Research Information System, "Digital twin for propulsion drive of autonomous electric vehicle," [Online]. Available: https://www.etis.ee/Portal/Projects/Display/72b66c74-e911-49c3-ac6a-6716f9e72ba5?lang=ENG. [Accessed 27 04 2022].

[3] A. Rassõlkin, T. Vaimann, A. Kallaste and V. Kuts, "Digital twin for propulsion drive of autonomous electric vehicle," in *2019 IEEE 60th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, 2019.

[4] M. Grieves, "Origins of the Digital Twin Concept," August 2016. [Online]. Available: 10.13140/RG.2.2.26367.61609. [Accessed 05 03 2022].

[5] O. G. Brylina, N. N. Kuzmina and K. V. Osintsev, "Modeling as the Foundation of Digital Twins," in *2020 Global Smart Industry Conference (GloSIC)*, 2020.

[6] Lockheed Martin, "Visualizing the digital thread and Digital Twins," Lockheed Martin, October 2021. [Online]. Available: https://www.lockheedmartin.com/en-us/news/features/2021/visualizing-the-digital-thread-and-digital-twins.html. [Accessed 05 03 2022].

[7] IBM, "What is a Digital Twin?," IBM, [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2016-10-18-gartner-identifies-the-top-10-strategic-technology-trends-for-2017. [Accessed 05 03 2022].

[8] J. Wu, Y. Yang, X. U. N. Cheng, H. Zuo and Z. Cheng, "The Development of Digital Twin Technology Review," in *2020 Chinese Automation Congress (CAC)*, 2020.

[9] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray and D. Devine, "Digital Twin: Origin to Future," *Applied System Innovation,* vol. 4, 2021.

[10] Gartner, "Gartner Identifies the Top 10 Strategic Technology Trends for 2017," October 2016. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2016-10-18-gartner-identifies-the-top-10-strategic-technology-trends-for-2017. [Accessed 06 03 2022].

[11] Gartner, "Gartner Identifies the Top 10 Strategic Technology Trends for 2018," October 2017. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-10-04-gartner-identifies-the-top-10-strategic-technology-trends-for-2018. [Accessed 06 03 2022].

[12] Gartner, "Gartner Identifies the Top 10 Strategic Technology Trends for 2019," October 2018. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-10-15-gartner-identifies-the-top-10-strategic-technology-trends-for-2019. [Accessed 06 03 2022].

[13] B. Marr, "These 25 Technology Trends Will Define The Next Decade," Forbes, 20 April 2020. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2020/04/20/these-25-technology-trends-will-define-the-next-decade/. [Accessed 05 03 2022].

[14] E. Glaessgen and D. Stargel, "The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles," in *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*.

[15] A. Best, S. Narang, L. Pasqualin, D. Barber and D. Manocha, "AutonoVi-Sim: Autonomous Vehicle Simulation Platform With Weather, Sensing, and Traffic Control," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Salt Lake City, 2018.

[16] J. Liu, Y. Dong, Y. Liu, P. Li, S. Liu and T. Wang, "Prediction Study of the Heavy Vehicle Driving State Based on Digital Twin Model," in *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*, 2021.

[17] Y. Chen, S. Chen, T. Zhang, S. Zhang and N. Zheng, "Autonomous Vehicle Testing and Validation Platform: Integrated Simulation System with Hardware in the Loop*," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[18] M. Ruba, R. O. Nemes, S. M. Ciornei, C. Martis, A. Bouscayrol and H. Hedesiu, "Digital Twin Real-Time FPGA Implementation for Light Electric Vehicle Propulsion System Using EMR Organization," in *2019 IEEE Vehicle Power and Propulsion Conference (VPPC)*, 2019.

[19] A. Rassõlkin, V. Rjabtšikov, T. Vaimann, A. Kallaste and V. Kuts, "Concept of the Test Bench for Electrical Vehicle Propulsion Drive Data Acquisition," in *2020 XI International Conference on Electrical Power Drive Systems (ICEPDS)*, 2020.

[20] V. Rjabtšikov, A. Rassõlkin, B. Asad, T. Vaimann, A. Kallaste, V. Kuts, S. Jegorov, M. Stępień and M. Krawczyk, "Digital Twin Service Unit for AC Motor Stator Inter-Turn Short Circuit Fault Detection," in *2021 28th International Workshop on Electric Drives: Improving Reliability of Electric Drives (IWED)*, 2021.

[21] V. Kuts, A. Rassõlkin, A. Partyshev, S. Jegorov and V. Rjabtšikov, "ROS middle-layer integration to Unity 3D as an interface option for propulsion drive simulations of autonomous vehicles," *IOP Conference Series: Materials Science and Engineering*, vol. 1140, p. 012008, May 2021.

[22] J. Cardoso, C. Pereira, A. Aguiar and R. Morla, "Benchmarking IoT middleware platforms," pp. 1-7, 2017.

[23] S. Jegorov, A. Rassõlkin, V. Kuts, V. Rjabtšikov and A. Partyshev, "The comparison between ROS and ROS2 based on the propulsion drive of autonomous vehicle," *Array*, 2022 (Submitted).

[24] ROS Wiki, "ROS Nodes," 2017. [Online]. Available: http://wiki.ros.org/Nodes. [Accessed 24 03 2022].

[25] ROS Wiki, "ROS Topics," 2017. [Online]. Available: http://wiki.ros.org/Topics. [Accessed 24 03 2022].

[26] ROS Wiki, "ROS Messages," 2018. [Online]. Available: http://wiki.ros.org/Messages. [Accessed 24 03 2022].

[27] ROS Wiki, "ROS Bags," [Online]. Available: http://wiki.ros.org/Bags. [Accessed 02 05 2022].

[28] R. Tellez, "Top 10 ROS based robotics companies," The Robot Report, 22 July 2019. [Online]. Available: https://www.therobotreport.com/top-10-ros-based-robotics-companies-2019/. [Accessed 24 03 2022].

[29] The Autoware Foundation, "Autoware Overview," The Autoware Foundation, [Online]. Available: https://www.autoware.org/autoware. [Accessed 24 03 2022].

[30] Open Robotics, "ROS2 Roadmap," [Online]. Available: https://docs.ros.org/en/foxy/Roadmap.html. [Accessed 25 03 2022].

[31] DDS Foundation, "What is DDS?," [Online]. Available: https://www.dds-foundation.org/what-is-dds-3/. [Accessed 25 03 2022].

[32] DDS Foundation, "What is the DDS Standard?," [Online]. Available: https://www.dds-foundation.org/omg-dds-standard//. [Accessed 25 03 2022].

[33] DDS Foundation, "DDS in Other Standards," [Online]. Available: https://www.dds-foundation.org/dds-in-other-standards/. [Accessed 25 03 2022].

[34] P. Bouchier, "Embedded ROS [ROS Topics]," *IEEE Robotics Automation Magazine*, vol. 20, pp. 17-19, 2013.

[35] H. Takase, T. Mori, K. Takagi and N. Takagi, "MROS: A Lightweight Runtime Environment for Robot Software Components onto Embedded Devices," in *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, New York, NY, USA, 2019.

[36] J. Staschulat, I. Lütkebohle and R. Lange, "The rclc Executor: Domain-specific deterministic scheduling mechanisms for ROS applications on microcontrollers: work-in-progress," in *2020 International Conference on Embedded Software (EMSOFT)*, 2020.

[37] R. Lange, "Micro-ROS – bringing the most popular robotics middleware onto tiny microcontrollers," Bosch Research Blog, 19 January 2021. [Online]. Available: https://www.bosch.com/stories/bringing-robotics-middleware-onto-tiny-microcontrollers/. [Accessed 29 03 2022].

[38] micro-ROS, "Execution Management," [Online]. Available: https://micro.ros.org/docs/concepts/client_library/execution_management/. [Accessed 15 04 2022].

[39] eProsima, "eProsima Micro XRCE-DDS," 2018. [Online]. Available: https://micro-xrce-dds.docs.eprosima.com/en/latest/index.html. [Accessed 19 04 2022].

[40] ROS Wiki, "ROS std_msgs message package," 2018. [Online]. Available: http://wiki.ros.org/std_msgs. [Accessed 12 04 2022].

[41] M. Ibrahim, A. Rassõlkin, S. Jegorov, V. Rjabtšikov, T. Vaimann and A. Kallaste, "Conceptual Modelling of an EV-Permanent Magnet Synchronous Motor Digital Twin," 2022 (Submitted).

[42] micro-ROS, "Supported Hardware | micro-ROS," 2022. [Online]. Available: https://micro.ros.org/docs/overview/hardware/. [Accessed 18 04 2022].

[43] PJRC, "Teensy® 4.0 Development Board," PJRC | Electronic Components Available Worldwide, [Online]. Available: https://www.pjrc.com/store/teensy40.html. [Accessed 18 04 2022].

[44] micro-ROS, "Teensy with Arduino | micro-ROS," 2018. [Online]. Available: https://micro.ros.org/docs/tutorials/core/teensy_with_arduino/. [Accessed 12 03 2022].

[45] micro-ROS, "micro-ROS for Arduino," GitHub, [Online]. Available: https://github.com/micro-ROS/micro_ros_arduino#readme. [Accessed 20 04 2022].

[46] ABB, "HES880 drives modules. Mobile drive solution for working machine and marine applications," 2018. [Online]. Available: #https://library.abb.com/d/3AUA0000161471#. [Accessed 20 04 2022].

[47] Mathworks, Inc., "Generate Code to Manually Deploy a ROS 2 Node from Simulink," Mathworks, Inc., [Online]. Available: https://www.mathworks.com/help/ros/ug/generate-code-to-manually-deploy-ros-2-node.html. [Accessed 25 04 2022].

List of publications

S. Jegorov, A. Rassõlkin, V. Kuts, V. Rjabtšikov and A. Partyshev, "The comparison between ROS and ROS2 based on the propulsion drive of autonomous vehicle," *Array*, 2022 (Submitted) – Appendix 7.

V. Rjabtšikov, A. Rassõlkin, B. Asad, T. Vaimann, A. Kallaste, V. Kuts, S. Jegorov, M. Stępień and M. Krawczyk, "Digital Twin Service Unit for AC Motor Stator Inter-Turn Short Circuit Fault Detection," in *2021 28th International Workshop on Electric Drives: Improving Reliability of Electric Drives (IWED)*, 2021. – Appendix 8

V. Kuts, A. Rassõlkin, A. Partyshev, S. Jegorov and V. Rjabtšikov, "ROS middle-layer integration to Unity 3D as an interface option for propulsion drive simulations of autonomous vehicles," *IOP Conference Series: Materials Science and Engineering*, vol. 1140, p. 012008, May 2021. – Appendix 9.

M. Ibrahim, A. Rassõlkin, S. Jegorov, V. Rjabtšikov, T. Vaimann and A. Kallaste, "Conceptual Modelling of an EV-Permanent Magnet Synchronous Motor Digital Twin," 2022 (Submitted). – Appendix 10

S. Jegorov, A. Rassõlkin, V. Rjabtšikov, M. Ibrahim and V. Kuts, " Novel Digital Twin Concept for Industrial Applications. Study Case: Propulsion Drive System," *ASME IMECE Conference 2022*, 2022 (Submitted) – Appendix 11.

Appendix 1 – Non-exclusive license for reproduction and publication of a graduation thesis[1]

I Sergei Jegorov

1. Grant Tallinn University of Technology free license (non-exclusive license) for my thesis “Middleware framework for Digital Twin entities communication”, supervised by Anton Rassõlkin and Eduard Petlenkov.
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive license.
3. I confirm that granting the non-exclusive license does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

05.05.2022

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – digital_twin_msgs ROS2 message definitions

Voltage.msg

```
float32 voltage1
float32 voltage2
float32 voltage3
```

Current.msg

```
float32 current1
float32 current2
float32 current3
```

SupplyInput.msg

```
builtin_interfaces/Time stamp
digital_twin_msgs/Voltage voltages
digital_twin_msgs/Current currents
```

Power.msg

```
builtin_interfaces/Time stamp
float32 phase1
float32 phase2
float32 phase3
float32 total
```

Float32Stamped.msg

```
builtin_interfaces/Time stamp
float32 data
```

Appendix 3 – Embedded software for sampling and transporting current and voltage data

```
/** @file tractionMotorMeasurement.c
 *  @brief Script to handle data sampling, serialization,
 *  and transport to middleware of TB DT.
 *
 *  @author Sergei Jegorov (sejego)
*/

#include <micro_ros_arduino.h>

#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <unistd.h>
#include <time.h>

#include <std_msgs/msg/float32.h>
#include <digital_twin_msgs/msg/supply_input.h>

#define LED_PIN 13
#define RCCHECK(fn, del) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){error_loop(del);}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){}}

rcl_publisher_t publisher;
digital_twin_msgs__msg__SupplyInput msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

unsigned long long time_offset = 0;
const char *node_name = "teensy_mcu";
const char *node_namespace = "tb_tm";
const int VT_SCALE = 10; // scale for voltage measurements 10mV/V;
const int CT_SCALE = 1; // current fir current measurements 1mV/A

typedef struct timespec timespec;
```

```
/** @brief Synchronize time of MCU with uROS Agent time
 *
 *  This function makes a call to uROS agent on the host to
 *  receive the UNIX time in nanoseconds. The current time of the MCU
 *  starts counting from 0 when it launches, thus we can find the time
 *  offset by subtracting MCU time from an actual UNIX time
 *
 *  @param None
 *  @return None
 *
 */
void sync_time(void)
{
    // get the current time from the agent
    unsigned long now = millis();
    RCCHECK(rmw_uros_sync_session(10), 1000);
    unsigned long long ros_time_ms = rmw_uros_epoch_millis();
    // now we can find the difference between ROS time and uC time
    time_offset = ros_time_ms - now;
}

/** @brief Get current UNIX time of the MCU
 *
 *  Takes into account the calculated offset and returns the UNIX time in
seconds and nanoseconds
 *  since seconds to be used as timestamp.
 *  @param None
 *  @return timespec type current time in UNIX seconds and nanoseconds since
seconds
 */
timespec get_time(void)
{
    timespec ts = {0};
    // add time difference between uC time and ROS time to
    // synchronize time with ROS
    unsigned long long now = millis() + time_offset;
    ts.tv_sec = now / 1000;
    ts.tv_nsec = (now % 1000) * 1000000;

    return ts;
}

/** @brief calculate the sample into voltage in mV
 *
 *  Recalculates the input sample bits into voltage. Knowing
 *  that ADC is 10-bit, it gives a precision of ~ 3.22 mV/bit
 *
 *  @param int sample, a value from 0 - 1023
 *  @return float voltage in mV
*/
float sampleToVoltage(int sample)
```

```
{
  return sample*(3300)/((float)1024); // mV
}

/** @brief Computes the real values of traction motor input.
 *
 *  Each pin from A0-A6 is read and first computed to mV value,
 * then is recalculated as follows: first 1000mV is subtracted to bring the
shifted signal down
 * to original one, then it is scaled value to reflect the real value of
sampled current/voltage.
 *
 * @param None
 * @return None
*/
void computeAndPublish(void)
{
    // create a temo variable to store intermediate voltage values
    int adc_in_sample = 0;
    adc_in_sample = analogRead(0);
    msg.currents.current1 = (sampleToVoltage(adc_in_sample) - 1000.0) /
CT_SCALE;
    adc_in_sample = analogRead(1);
    msg.currents.current2 = (sampleToVoltage(adc_in_sample) - 1000.0) /
CT_SCALE;
    adc_in_sample = analogRead(2);
    msg.currents.current3 = (sampleToVoltage(adc_in_sample) - 1000.0) /
CT_SCALE;
    adc_in_sample = analogRead(3);
    msg.voltages.voltage1 = (sampleToVoltage(adc_in_sample) - 1000.0) /
VT_SCALE;
    adc_in_sample = analogRead(4);
    msg.voltages.voltage2 = (sampleToVoltage(adc_in_sample) - 1000.0) /
VT_SCALE;
    adc_in_sample = analogRead(5);
    msg.voltages.voltage3 = (sampleToVoltage(adc_in_sample) - 1000.0) /
VT_SCALE;

    timespec ts = get_time();
    msg.stamp.sec = ts.tv_sec;
    msg.stamp.nanosec = ts.tv_nsec;

    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
}

/** @brief Enter an error state, blinking the LED with a designated frequency
 *  in an infinite loop
 *
 *  @param delay_ms indicating the period of blinking
 *  @return None
*/
void error_loop(int delay_ms)
```

```
{
  while(1){
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(delay_ms);
  }
}

/** @brief Callback function with a wall timer used for publishing ROS
messages periodically
 *
 *  Timer callback is executed everytime a timer fires an interrupt.
 *
 *  @param pointer to timer, int64_t last_call_time
 *  @return None
 */
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    computeAndPublish();
  }
}

/** @brief Setup function to initialize all ROS2 nodes, publishers,
subscribers, timers
 *  and uROS executors
 *
 *
 *  Initializes uROS executors, publishers with designated topics and message
types, timers
 *  and callbacks for publishing and handling subscriptions. In case
something goes wrong,
 *  MCU will enter into an error state with LED blinking
 *
 *  @param None
 *  @return None
 */
void setup() {
  set_microros_transports();

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH);

  delay(1000);

  allocator = rcl_get_default_allocator();

  //create init_options, if fails, will blink every 1s
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator), 1000);

  // create node
```

```
  RCCHECK(rclc_node_init_default(&node, node_name, node_namespace, &support),
1000);

  // create publisher, if fails, the LED blinks every 100ms
  RCCHECK(rclc_publisher_init_default(&publisher, &node,
ROSIDL_GET_MSG_TYPE_SUPPORT(digital_twin_msgs, msg, SupplyInput),
"supply_input"), 1000);

  // create timer,
  const unsigned int timer_timeout = RCL_MS_TO_NS(1);
  RCCHECK(rclc_timer_init_default(&timer, &support, timer_timeout,
timer_callback), 500);

  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator),
500);
  RCCHECK(rclc_executor_add_timer(&executor, &timer), 100);

  sync_time();

}

/** @brief loop function where main code executes
 *
 *  spin the executor forever to run uROS
 *
 *  @param None
 *  @return None
 */
void loop() {
  rclc_executor_spin(&executor);
}
```

Appendix 4 – Latency test software run on Teensy 4.0 MCU

```
/** @file latency_test.c
 *  @brief Script for generating data with time stamps
 *  and measuring RTT latency.
 *
 *  @author Sergei Jegorov (sejego)
 *
 */

#include <micro_ros_arduino.h>

#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <rmw_microros/rmw_microros.h>
#include <unistd.h>

#include <std_msgs/msg/u_int64.h>
#include <digital_twin_msgs/msg/latency_test.h>

#define LED_PIN 13
#define RCCHECK(fn, del) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){error_loop(del);}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){}}


rcl_publisher_t publisher_ping;
rcl_publisher_t publisher_result;
rcl_subscription_t subscription_pong;

digital_twin_msgs__msg__LatencyTest msg_in;
digital_twin_msgs__msg__LatencyTest msg_out;
std_msgs__msg__UInt64 msg_res;

rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

unsigned int msg_id = 0;
unsigned long long time_offset = 0;
```

```
const char *node_name = "teensy_mcu";
const char *node_namespace = "tb_tm";


/** @brief Enter an error state, blinking the LED with a designated frequency
 *  in an infinite loop
 *
 *  @param delay_ms indicating the period of blinking
 *  @return None
*/
void error_loop(int delay_ms)
{
  while(1){
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(delay_ms);
  }
}

/** @brief Synchronize time of MCU with uROS Agent time
 *
 *  This function makes a call to uROS agent on the host to
 *  receive the UNIX time in nanoseconds. The current time of the MCU
 *  starts counting from 0 when it launches, thus we can find the time
 *  offset by subtracting MCU time from an actual UNIX time
 *
 *  @param None
 *  @return None
 *
 */
void sync_time(void)
{
    unsigned long now = micros();
    rmw_uros_sync_session(10);
    unsigned long long ros_time_us = rmw_uros_epoch_nanos() / 1000;
    // now we can find the difference between ROS time and uC time
    time_offset = ros_time_us - now;
}

/** @brief Get current UNIX time of the MCU
 *
 *  Takes into account the calculated offset and returns the UNIX time in
microseconds
 *
 *  @param None
 *  @return uint64_t current time in microseconds
 */
unsigned long long get_time(void)
{
    // add time difference between uC time and ROS time to
    // synchronize time with ROS
```

```
    unsigned long long now = micros() + time_offset;
    return now;
}

/** @brief Publish ping message with the ID and time stamp
 *
 *  @param None
 *  @return None
 */
void publish_ping(void)
{
    unsigned long long stamp = get_time();

    msg_out.seq_id = msg_id;
    msg_out.stamp = stamp;
    RCSOFTCHECK(rcl_publish(&publisher_ping, &msg_out, NULL));
    msg_id += 1;
}

/** @brief Publish latency result message to calculating Node on host
 *
 *  Calculates the difference in received time as a RRT.
 *
 *  @param pointer to message type
 *  @return None
 */
void publish_res(const void * msgin)
{
    unsigned long long time_now = get_time();
    const digital_twin_msgs__msg__LatencyTest * msg = (const
digital_twin_msgs__msg__LatencyTest *)msgin;
    msg_res.data = time_now - msg->stamp;
    RCSOFTCHECK(rcl_publish(&publisher_result, &msg_res, NULL));
}

/** @brief Callback function with a wall timer used for publishing ROS
messages periodically
 *
 *  Timer callback is executed everytime a timer fires an interrupt.
 *
 *  @param pointer to timer, int64_t last_call_time
 *  @return None
 */
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
    RCLC_UNUSED(last_call_time);
    if (timer != NULL){
        publish_ping();
    }
}
```

```
/** @brief Subscriber callback to perform operation when new message is
received
 *
 *  @param pointer to received message
 *  @return None
 */
void subscriber_pong_callback(const void * msgin)
{
    publish_res(msgin);
}

/** @brief Setup function to initialize all ROS2 nodes, publishers,
subscribers, timers
 *  and uROS executors
 *
 *
 *  Initializes uROS executors, publishers with designated topics and message
types, timers
 *  and callbacks for publishing and handling subscriptions. In case
something goes wrong,
 *  MCU will enter into an error state with LED blinking
 *
 *  @param None
 *  @return None
 */
void setup() {
  set_microros_transports();

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH);

  delay(1000);

  allocator = rcl_get_default_allocator();

  //create init_options, if fails, will blink every 1s
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator), 1000);

  // create node
  RCCHECK(rclc_node_init_default(&node, node_name, node_namespace, &support),
1000);

  // create publisher, if fails, the LED blinks every 100ms
  RCCHECK(rclc_publisher_init_default(&publisher_ping, &node,
ROSIDL_GET_MSG_TYPE_SUPPORT(digital_twin_msgs, msg, LatencyTest), "ping"),
2000);
  RCCHECK(rclc_publisher_init_default(&publisher_result, &node,
ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, UInt64), "latency_results"),
2000);

  // create subscriber, if fails, the LED blinks every 100ms
```

```
  RCCHECK(rclc_subscription_init_default(&subscription_pong, &node,
ROSIDL_GET_MSG_TYPE_SUPPORT(digital_twin_msgs, msg, LatencyTest), "pong"),
2000);

  // create timer,
  const unsigned int timer_timeout = RCL_MS_TO_NS(1);
  RCCHECK(rclc_timer_init_default(&timer, &support, timer_timeout,
timer_callback), 1000);

  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 2, &allocator),
500);
  RCCHECK(rclc_executor_add_timer(&executor, &timer), 300);
  RCCHECK(rclc_executor_add_subscription(&executor, &subscription_pong,
&msg_in, &subscriber_pong_callback, ON_NEW_DATA), 3000);

  sync_time();
}

/** @brief loop function where main code executes
 *
 *  spin the executor forever to run uROS
 *
 *  @param None
 *  @return None
 */
void loop() {
  rclc_executor_spin(&executor);
}
```

Appendix 5 – Latency software run on MATLAB

```
latencyTestNode = ros2node("/latencyTestNode");
pause(2);
global next;
global recv;
global lost;
next = 0;
recv = 0;
lost = 0;
pingSubscriber = ros2subscriber(latencyTestNode,"/tb_tm/ping");
pongPublisher =
ros2publisher(latencyTestNode,"/tb_tm/pong","digital_twin_msgs/LatencyTest");

while true
    msg = receive(pingSubscriber,10);
    if next == msg.seq_id
        recv = recv + 1;
        out_msg = ros2message("digital_twin_msgs/LatencyTest");
        out_msg.seq_id = msg.seq_id;
        out_msg.stamp = msg.stamp;
        send(pongPublisher,out_msg);
    else
        lost = lost + msg.seq_id - next;
    end

    next = msg.seq_id + 1;

    if recv >= 60000
        quit();
    end
end
```

Appendix 6 – Latency test software run on Ubuntu VM

```
/**
 * @file latencyTestNode.cpp
 * @author Sergei Jegorov (sejego)
 * @brief This ROS2 Node records latencies, received and lost messages,
calculates
 * min, max and meand latencies in microseconds.
 *
 * @copyright Copyright (c) 2022
 *
 */

#include <iostream>
#include <vector>
#include <chrono>
#include <ratio>
#include <memory>
#include <algorithm>

#include "rclcpp/rclcpp.hpp"
#include "rclcpp/time.hpp"

#include <digital_twin_msgs/msg/latency_test.hpp>
#include "std_msgs/msg/u_int64.hpp"

#include "data_logger/data_logger.hpp"

using namespace DataLogger;
using namespace std::chrono_literals;

class LatencyTestNode : public rclcpp::Node
{
  public:
    std::unique_ptr<SubscriptionLogger> p_input_sub;

    LatencyTestNode() : Node("latency_test_node")
    {
      PongPublisher_ = this-
>create_publisher<digital_twin_msgs::msg::LatencyTest>("/tb_tm/pong", 10);

      PingSubscriber_ = this-
>create_subscription<digital_twin_msgs::msg::LatencyTest>("/tb_tm/ping", 50,

std::bind(&LatencyTestNode::pingCallback, this, std::placeholders::_1));
```

```
      LatencySubscriber_ = this-
>create_subscription<std_msgs::msg::UInt64>("/tb_tm/latency_results", 100,

std::bind(&LatencyTestNode::latencyCallback, this, std::placeholders::_1));
      p_input_sub.reset(new SubscriptionLogger("/tb_tm/ping"));

      RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Subscription logger
initialized");
      RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "LatencyTestNode
initialized");
    }

  private:
    /* Declare all message types, Publishers and Subscribers */

    rclcpp::Publisher<digital_twin_msgs::msg::LatencyTest>::SharedPtr
PongPublisher_;
    rclcpp::Subscription<digital_twin_msgs::msg::LatencyTest>::SharedPtr
PingSubscriber_;
    rclcpp::Subscription<std_msgs::msg::UInt64>::SharedPtr
LatencySubscriber_;
    digital_twin_msgs::msg::LatencyTest msg_to_send;

    /* If the expected 'ping' message is received, it is considered received,
     * and is sent back to the original publisher. Then, it receives the
recorded latencies
     * and stores them in a vector of latencies
    */
    void pingCallback(const digital_twin_msgs::msg::LatencyTest::SharedPtr
msg)
    {
      if(msg->seq_id == p_input_sub->next_id) {
        msg_to_send.seq_id = msg->seq_id;
        msg_to_send.stamp = msg->stamp;
        p_input_sub->recv_counter += 1;
        PongPublisher_->publish(msg_to_send);
      } else {
        p_input_sub->lost_count += 1;
      }
      p_input_sub->next_id = msg->seq_id + 1;
    }

    void latencyCallback(const std_msgs::msg::UInt64::SharedPtr msg){
      uint64_t latency_us = msg->data / 2;
      p_input_sub->time_diffs.push_back(latency_us);
    }
};

int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);
```

```
  auto ptr = std::make_shared<LatencyTestNode>();
  rclcpp::spin(ptr);
  DataLogger::save_logged_data("latency_test_results.csv");
  rclcpp::shutdown();
  return 0;
}
```

Appendix 7 – The comparison between ROS and ROS2 based on the propulsion drive of autonomous vehicle

The comparison between ROS and ROS2 based on the propulsion drive of autonomous vehicle

Sergei Jegorov[1], Vladimir Kuts[1,3*], Anton Rassõlkin[2], Andriy Partyshev[1],Viktor Rjabtšikov[2]

***Abstract*— The Industrial Internet Of Things (IIoT) is a leading trend in systems development and is being applied in various autonomous systems for control and monitoring purposes. Sensors and actuators transmitting data over the network prove valuable for creating models, conducting accurate simulations, verifying and troubleshooting complex systems. Such systems typically consist of several subsystems that are relying on middleware frameworks for intercommunication. Robot Operating System (ROS) framework is commonly used in mobile robots and autonomous vehicles development. Having collection of ready made packages available for use and providing tools for nodes interconnection made ROS famous in academy and industry. Since 2017, ROS2 is continuously being developed and released, and will eventually replace ROS. Main ROS2 targets are to eliminate problems present in ROS and to add new features supporting real-time implementations. In this paper, the authors explain the importance of middleware, compare notable middleware frameworks commonly used in the fields of robotics and autonomous vehicles, and justify why ROS2 could be preferred for the Digital Twin (DT) applications. Moreover, research includes latency performance comparison between ROS and ROS2 on a basis of existing DT system that was migrated from ROS to ROS2.**

***Keywords* – ROS, ROS2, digital twin, autonomous vehicles, middleware, latency, IoT**

I. INTRODUCTION

The advancement of IoT (Internet of Things) has created new opportunities for creating sophisticated systems, such as smart cities, smart gadgets, mobile robots, autonomous vehicles, etc. In mentioned smart systems, the communication between the entities is happening inside a middleware - distributed system services that have standard programming interfaces and protocols [1]. The importance of middleware is critical - without it the system cannot operate as a whole, and this is especially a concern in the complex systems such as autonomous vehicles, robotics, fault-detection etc. The choice of the right middleware becomes therefore crucial, as it often can determine how reliably and fast the system will perform.

In previous research studies conducted by the authors, Digital Twin (DT) for propulsion electrical drive of autonomous vehicle was introduced [2], with a dedicated test bench where a real motor drive was connected with Unity 3D visualized motor[3]. The data exchange and additional computation was happening in Robot Operating System (ROS). Since then, the system defined in middleware was migrated to the successor of ROS - ROS2.

In this article, the choice of middleware will be explained through a comparison of available middleware platforms, and results of performance evaluation of ROS and ROS2 will be discussed.

A. What is middleware?

There is no official definition of the term "middleware" available, as industry and academics explain this term differently, yet one definition found by the authors explains it the most clearly: middleware platforms are intermediaries between sensors, services, and applications, managing the flow of data and allowing them to interoperate [4]. Middleware handles all the serialization and transfer of information from one platform to another utilizing various applied standards. Middleware has a defined Application Program Interface (API) that allows engineers to bind the middleware software to their parts of the system and allow inter-system communication. Dozens of middleware frameworks are available for use, both proprietary and free of charge. Some of the frameworks are based on standard communication protocols (such as DDS), whereas other frameworks use custom solutions.

II. ASSESMENT OF AVAILABLE MIDDLEWARE

There is no systematic way of comparing different middleware frameworks, therefore, in this paper we will define our set of criteria for evaluating and comparing the available middleware. A set of qualitative criteria for IoT frameworks was suggested in [4], what is also appropriate for the current study:

1) area of application (web-development, embedded programming, etc);
2) support for the desired communication model (pub-sub, request-response, etc);
3) availability and clarity of the documentation, as well as available tutorials;
4) quality of the support and livelihood of developer communities;

First and foremost, the area of development of interest of this study is industrial simulations for DTs. Since the main research topic deals with electrical motor-drive system and autonomous vehicle, it was considered beneficial to use technologies that are either used in these fields or utilize standardized protocols suitable for industrial/transportation use cases.

*vladimir.kuts@taltech.ee
[1]Tallinn University of Technology, School of Engineering, Department of Mechanical and Industrial Engineering, Estonia, Harju, Tallinn, Ehitajate str. 5, 19086
[2]Tallinn University of Technology, School of Engineering, Department of Electrical Power Engineering and Mechatronics, Estonia, Harju, Tallinn, Ehitajate str. 5, 19086
[3]University of Limerick, Electronic and Computer Engineering Department, Ireland, Limerick, V94 T9PX

TABLE I
COMPARISON OF AVAILABLE MIDDLEWARE.

Platform	Initial Release	Type	Messaging Type	Advantages	Disadvantages
ach	2013	Inter-Process Communication mechanism	Message bus Publish-subscribe	+ Real-time support + Solved head-of-line problem for accessing the newest message + Extensive documentation	- Inactive community - Development discontinued - No ready software packages
YARP	2002	Robotics middleware	Publish-subscribe	+ Extensible family of connection types + Extensive documentation + Active community + QoS policies	- Limited real time support - No ready software packages
LCM	2006	Libraries and tools for message passing and data marshalling, targeted at real-time systems	Publish-subscribe	+Distributed network topology + Low-latency inter-process communication +Large support of programming languages	- No ready software packages - Development stalled - Weak documentation - Inactive community
ROS	2007	Robotics middleware	Publish-subscribe	+ Extensive collection of ready-to-use packages +Extensive documentation + Active community	- Limited real time support - Has a master server through which all connections are handled - Support ends in 2025
ROS2	2017	Robotics middleware, successor of ROS	Publish-subscribe	+ Real time support + Distributed network topology + Native embedded support + Based on a standard + Active community + Extensive documentation + QoS policies	- Development is still ongoing - Documentation is aimed more on ROS users - Some of ROS ready packages are still being ported

The communication model preferably should be modular and of publisher-subscriber type, since at any point of operation it should be possible to get the data of a single component and study it. This can be helpful in creating systems where elements may need to get the data from another shared element. Likewise, the control of subscriptions and publishers is possible, either by stopping any publishing, or publishing simulated data to observe the reaction of the system. Based on these parameters, the following frameworks can be considered suitable for the needs of the research: ach, LCM, YARP, ROS, ROS2. An overview of the main advantages and disadvantages found in these frameworks is presented in Table I.

Ach [5] provides a message bus or publish-subscribe style of communication between multiple writers and multiple readers. A real-time system has multiple Ach channels across which individual data samples are published. Ach was created with intention to be used for communication in real-time systems that sample data from physical processes.

LCM [6] stands for Lightweight Communications and Marshalling. It is a lightweight library for message passing using publish-subscribe model aimed at assisting the development of low-latency, real-time systems. Apart from its message passing and marshalling, the LCM is also notable for providing real-time deep traffic inspection tool that can decode and display messages with minimal user effort. LCM is a standalone library that can be easily integrated in a variety of systems.

YARP [7] (Yet Another Robot Platform) is a framework developed to support modular development of humanoid robotics that are to be operated in harsh and non-ideal conditions. YARP is highly modular and is compatible with other frameworks, if needed. YARP includes a model of communication that is transport-neutral, so that data flow is decoupled from the details of the underlying networks and protocols in use. Importantly for the long term, YARP is designed to play well with other architectures.[8]

ROS - a framework for robotics development which consists of a pub-sub mechanism that exchanges ROS messages of TCP/UDP network. ROS is widely used in academia and research for its rich set of documentation and available ROS packages - already developed software components for complex tasks (navigation, localization, computer vision etc) that is open-source and available to everybody.

ROS2 is the successor of ROS. ROS2 is currently being actively developed to provide the following features: support of Data Distribution Service (DDS) standard, industry-grade support and the deprecation of original ROS in 2025 [9], transfer of ROS core libraries to use the C++11 standard, real-time operating systems (RTOS) and microcontrollers native support [10]. The interest in development of such system has even brought renown companies in technology such as Apex AI, iRobot and Sony [11] to work on various features of ROS2.

It is understandable that these frameworks can perform better or worse depending on the use cases. However, popularity of ROS and its large support from the community and technical industries promises a robust and reliable system. Additionally, Autoware AI - leading development platform for autonomous vehicles is based on ROS, and is used in

the studied vehicle - ISEAUTO [12]. The test bench that was developed in [2] is largely based on the drive system of ISEAUTO. Likewise, active development of ROS2, official statement that ROS will be supported only until 2025, new features of ROS2 all suggested that migrating existing system in its early development stage would be beneficial for the authors of the research.

III. Comparison of ROS and ROS2 performance

One of the questions that may arise in one's mind when migrating from one technology to another is, why is the new technology better than the previous one? The topic of difference in performance of ROS and ROS2 has already been researched in [13], where dependency of latency and size of the transmitted data was outlined. Since the release of ROS2, researchers have been investigating the performance of ROS2 in order to shed light on possible underdevelopments of ROS2 and help to improve the system design. This way, researchers could actively contribute to the development of ROS2. Notable examples of such kinds are ApexAI's performance test [14] and iRobot's ROS2 benchmark [15]; these companies are directly involved in ROS2 development. However, none of similar frameworks were found for original ROS, hence the performance evaluation between two versions (ROS and ROS2) had to be conducted from scratch.

A. Specimen case study

The system that acts as a specimen in the experiment is a part (refer to it as *loading_motor_dt*) of Loading Motor DT [16] that has been created initially using ROS and then migrated to ROS2. The *loading_motor_dt* is a middleware written in C++ laying between the real test bench and the 3D visualization simulated in Unity3D environment, and is responsible for calculating physical values of the 7,5 kW induction motor. Currently, *loading_motor_dt* uses the data received from the data acquisition system (DAS) and from the efficiency map created during a previous study. DAS reads the 3-phase AC current and voltage magnitudes at a specified frequency and stores them as *ćsv* files. All files are then loaded into the *loading_motor_dt* through ROS launch files, processed and used for further calculation of motor's values. The structure of the current ROS system can be seen in the Fig. 1.

B. Description of evaluation

There are multiple ways of determining the performance of the system. In this research, focus will be on one of the core values of any similar middleware - communication, namely the **latency** of messaging. Latency can be defined as a time delay between initial input and output. In the case of middleware, latency would be the time delay between the moment a ROS message was sent from one node, and the moment it was received by the other node, e.g.:

$$t_{lat} = t_{recv} - t_{sent} \tag{1}$$

where t_{lat} is latency, t_{recv} is the time moment where message was received and t_{sent} is the moment the message was sent.

Every ROS message, apart from the load (or the useful data) consists of two more data fields: the unique ID of the message and the time variable in nanoseconds. ID is required to verify that the received message is indeed the expected message and was not lost. IDs are incremented by 1 every time they are sent. On the receiving node, the received message's ID is checked and then it is incremented by 1 - this way, if a message was lost, the receiving node can still expect correct message during the next cycle. In case the ID matches, the message is counted as received, and the difference in t_{recv} and t_{sent} is calculated; otherwise, this is omitted. Visually, this is represented in the Fig. 2.

In order to keep track of recorded latencies, tracked nodes, received messages count etc, a separate class was defined - *DataLogger*. *DataLogger* is a ROS-independent class (can be instantiated in both ROS and ROS2), whose instances are created for every launched node, and they keep the track on aforementioned data and calculate the maximum, minimum and mean latency of every node. When *loading_motor_dt* stops executing, the following parameters per node are stored in *ćsv* files: *topic_name, sent(#), received(#), mean latency, maximum latency, minimum latency, frequency, message size*. These files are later grouped and analyzed using *Pandas* Python package.

Each experiment is run for 60 seconds. All experiments were carried out on a machine with the following parameters:

- CPU Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz
- RAM 32GB
- x64 POP OS! 20.04 (Ubuntu-based Linux distribution)
- ROS2 Foxy Fitzroy with | ROS Noetic Ninjemys [1]

IV. RESULTS

This section contains the results for latency test obtained through *DataLogger* class and iRobot benchmark. Obtained results were analyzed with *Pandas* in Python.

A. ROS vs. ROS2

Tables II and III compare ROS and ROS2 latency tests. The difference between ROS and ROS2 appear to be significant, with ROS2 being much more robust and reliable. In ROS, loss in messages appears to be higher for topics transmitting messages at very high frequencies, whereas in ROS2 they are somewhat independent of topic frequency. Interestingly, both ROS versions show increased latency for messages published on 'fairly low' frequencies (0.2 Hz). Overall, ROS2 appears to be much more efficient and reliable.

[1]Both versions were present on the machines, but only one at a time was running for latency test. ROS2 uses Default DDS vendor eProsima Fast DDS.

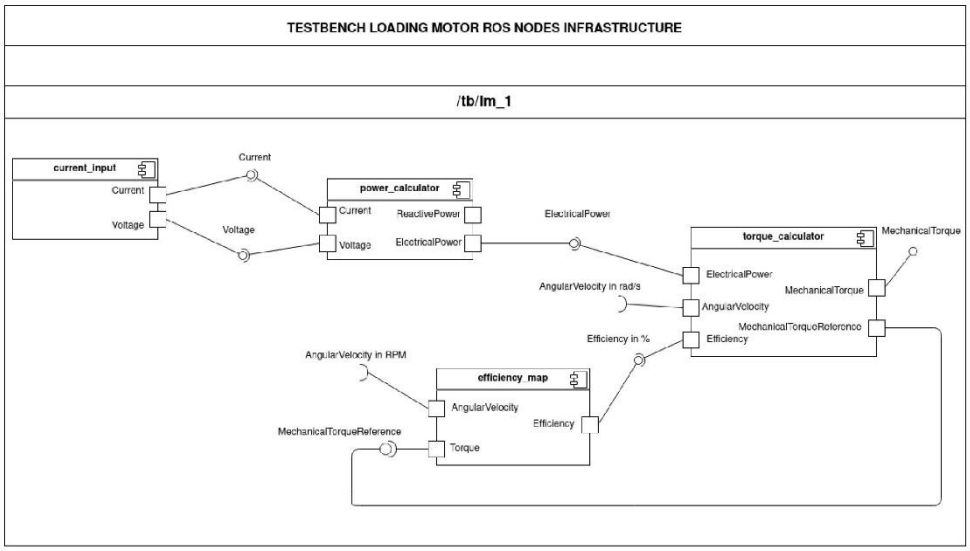


Fig. 1. loading_motor_dt system in ROS2

TABLE II
LATENCY TEST RESULTS FOR ROS.

	topic	received(#)	lost(#)	mean[us]	max_latency[us]	min_latency[us]	message_size[b]	frequency[hz]
1	/input_current	58199	337	8796.0	21603.3	180.4	24	1000
2	/input_voltage	58427	109	8837.3	22004.1	228.0	24	1000
3	/electrical_torque_ref	3532	36	2115737.8	4293061.0	2982.6	16	60
4	/actual_rpm	3532	24	2114560.6	4293068.0	3024.2	16	60
5	/motor_power/electrical_power	11	0	2500060.1	4131624.0	558433.1	28	0.2
6	/efficiency	588	5	15823.0	17038.6	174.3	16	60

TABLE III
LATENCY TEST RESULTS FOR ROS2.

	topic	received(#)	lost(#)	mean[us]	max_latency[us]	min_latency[us]	message_size[b]	frequency[hz]
1	/input_current	58563	3	347.7	18525.2	47.6	24	1000
2	/input_voltage	58563	3	356.5	18489.2	57.8	24	1000
3	/electrical_torque_ref	3557	8	352.6	12211.1	71.6	16	60
4	/actual_rpm	3565	2	364.1	11237.5	91.4	16	60
5	/motor_power/electrical_power	11	0	350.6	496.6	282.6	28	0.2
6	/efficiency	595	0	563.9	110559.0	99.6	16	60

B. ROS vs. ROS2 vs. iRobot benchmark

Another interesting comparison to show would be the difference between the evaluation of ROS, ROS2 and iRobot benchmark. As can be seen in Table IV and V, iRobot benchmark showcases even more superior results of evaluation compared to one conducted in this research. However, it should be noted that iRobot generates a ROS2 system based on a defined topology, where all nodes are supposedly same in their computation, and are optimized to maximum efficiency. Additionally, the results generated by iRobot did not include the actual size of the message transmitted by nodes. Defined message size in topology is recorded as a result, excluding the size of the header each iRobot benchmark message had. For this reason, additional 16 bytes (the size of performance_test/header) were added to the message_size column in Table IV.

V. DISCUSSION AND CONCLUSIONS

Conducted research shows that never mind the system and middleware, there can and will be loss of some data

TABLE IV
LATENCY TEST RESULTS FROM IROBOT BENCHMARK.

	topic	received(#)	lost(#)	mean[us]	max_latency[us]	min_latency[us]	message_size[b]	frequency[hz]
1	/input_current	59849	0	78	902	9	28	1000
2	/input_voltage	59878	0	50	971	10	28	1000
3	/electrical_torque_ref	3595	0	92	750	20	20	60
4	/actual_rpm	3595	0	93	4912	21	20	60
5	/motor_power/electrical_power	12	0	139	233	71	32	0.2
6	/efficiency	600	0	102	796	16	20	60

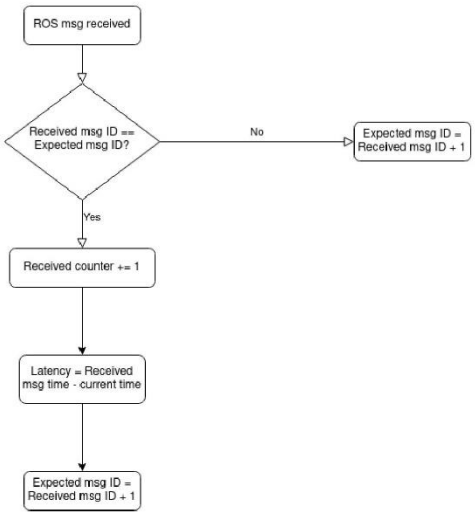


Fig. 2. Visual representation of DataLogger's algorithm.

during transmission, and engineers must be ready to increase redundancy in their systems. Same applies to the maintainers of *loading_motor_dt*.

Communication reliability and speed appears to be significantly larger in ROS2, which can be explained by the lack of *rosmaster*, adopted DDS standard and upgrade of ROS features to accommodate newer C++ features. This suggests that switching to ROS2 in the long-term will improve operations of systems already written and proven to work in original ROS.

Initially, performance evaluation of the ROS2 version of the system was supposed to be conducted using only iRobot's framework, and an attempt was made to create a similar framework for ROS. However, this idea was abandoned due to the fact that original ROS design is not suitable for the creation of a similar framework because of following reasons:

1) Original ROS cannot dynamically generate Nodes[2].

[2]This scenario is possible with use of Nodelets, but it then would assume that Nodelets are running in a single process, which is not true in case of our system in both ROS and ROS2

TABLE V
COMPARISON OF ALL THREE LATENCY TESTS.

	evaluations	average_mean[us]	lost(#)
1	iRobot's ROS2 framework	92.3	0
2	ROS2	389.2	16
3	ROS1	1127302.5	511

2) iRobot's framework design targets latest ROS2 features and API which are totally distinct from original ROS. Reverse engineering in this case is unreasonable.
3) iRobot's framework is well-designed and complex for a reason - the outcome of this work directly contributes to the development of ROS2. There would be minimal contribution from such system to original ROS at this time.

It still remains unclear why iRobot's framework showcased significantly better result and if it is a result of optimizations on iRobot's side or disadvantages of our system, and requires further investigation.

APPENDIX

All experiments with instructions are available on the PSG453 project's GitHub.

ACKNOWLEDGMENTS

The research is supported by the Estonian Research Council under grant PSG453 "Digital Twin for Propulsion Drive of Autonomous Electric Vehicle". In addition, Vladimir Kuts has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 847577; and a research grant from Science Foundation Ireland (SFI) under Grant Number 16/RC/3918 (Ireland's European Structural and Investment Funds Programmes and the European Regional Development Fund 2014-2.

REFERENCES

[1] P. A. Bernstein, "Middleware: A model for distributed system services," *Commun. ACM*, vol. 39, no. 2, pp. 86–98, Feb. 1996, ISSN: 0001-0782. DOI: 10.1145/230798.230809. [Online]. Available: https://doi.org/10.1145/230798.230809.

[2] A. Rassõlkin, T. Vaimann, A. Kallaste, and V. Kuts, "Digital twin for propulsion drive of autonomous electric vehicle," Oct. 2019. DOI: 10.1109/RTUCON48111.2019.8982326.

[3] V. Kuts, A. Rassõlkin, A. Partyshev, S. Jegorov, and V. Rjabtšikov, "ROS middle-layer integration to unity 3d as an interface option for propulsion drive simulations of autonomous vehicles," *IOP Conference Series: Materials Science and Engineering*, vol. 1140, no. 1, p. 012 008, May 2021. DOI: 10.1088/1757-899x/1140/1/012008. [Online]. Available: https://doi.org/10.1088/1757-899x/1140/1/012008.

[4] J. Cardoso, C. Pereira, A. Aguiar, and R. Morla, "Benchmarking iot middleware platforms," pp. 1–7, 2017. DOI: 10.1109/WoWMoM.2017.7974339.

[5] N. T. Dantam, D. M. Lofaro, A. Hereid, P. Y. Oh, A. D. Ames, and M. Stilman, "The ach library: A new framework for real-time communication," *IEEE Robotics Automation Magazine*, vol. 22, no. 1, pp. 76–85, 2015, ISSN: 1070-9932. DOI: 10.1109/MRA.2014.2356937.

[6] A. S. Huang, E. Olson, and D. C. Moore, "Lcm: Lightweight communications and marshalling," pp. 4057–4062, 2010. DOI: 10.1109/IROS.2010.5649358.

[7] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 8, Mar. 2006. DOI: 10.5772/5761. [Online]. Available: https://doi.org/10.5772/5761.

[8] P. Fitzpatrick, G. Metta, and L. Natale, "Towards long-lived robot genes," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, Jan. 2008. DOI: 10.1016/j.robot.2007.09.014. [Online]. Available: https://doi.org/10.1016/j.robot.2007.09.014.

[9] OpenRobotics, *Noetic ninjemys: The last official ros 1 release*. [Online]. Available: http://design.ros2.org/articles/changes.html.

[10] D. Thomas, *Changes between ros 1 and ros 2," ros 2 design*. [Online]. Available: http://design.ros2.org/articles/changes.html.

[11] OpenRobotics, *Ros2 features roadmap*. [Online]. Available: https://docs.ros.org/en/foxy/Roadmap.html.

[12] A. Rassõlkin, R. Sell, and M. Leier, "Development case study of first estonian self-driving car iseauto," 1, vol. 14, 2018, pp. 81–88.

[13] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," Oct. 2016. DOI: 10.1145/2968478.2968502. [Online]. Available: https://doi.org/10.1145/2968478.2968502.

[14] ApexAI, *Apexai ros2 performance test*, https://gitlab.com/ApexAI/performance_test/, 2020.

[15] iRobot, *Irobot ros2 performance evaluation framework*, https://github.com/irobot-ros/ros2-performance, 2020.

[16] A. Rassõlkin, V. Rjabtšikov, T. Vaimann, A. Kallaste, V. Kuts, and A. Partyshev, "Digital twin of an electrical motor based on empirical performance model," in *2020 XI International Conference on Electrical Power Drive Systems (ICEPDS)*, IEEE, 2020, pp. 1–4.

Sergei Jegorov is currently a Computer and Systems Engineering MSc student at Tallinn University of Technology. His primary interest is in software engineering, IoT, and systems development. He participated previously in a Horizon2020 EU project L4MS and is working now in the PSG453 project of Tallinn University of Technology.

Vladimir Kuts received his Ph.D. in Mechanical Engineering from Tallinn University of Technology (TalTech) in 2019. From the Year 2017, Dr. Kuts has been Head of Industrial Virtual and Augmented Reality Laboratory (www.ivar.taltech.ee) in the Department of Mechanical and Industrial Engineering Department of TalTech. Currently, he is a research fellow at the University of Limerick, Ireland. His main research interests include Industrial Digital Twins synchronized with real industrial equipment such as robots and Virtual Reality technologies for human-robot interaction standards validation. Moreover, he is serving as vice-chair of the IEEE Estonian section.

Anton Rassõlkin was born in Tallinn, Estonia, in 1985. He received the BSc, MSc, and Ph.D. degrees in electric drives and power electronics from Tallinn University of Technology (Estonia) in 2008, 2010, and 2014, respectively. In 2010 he received a Dipl.-Ing. degree in automation from the University of Applied Science Giessen-Friedberg (Germany). He has been working in several companies as an electrical engineer and at universities as a lecturer. Internationally, he has been working as a visiting researcher at the Institute for Competence in Auto Mobility (IKAM, Barleben, Germany), a visiting associate at Belarusian State Technological University (Minsk, Belarus). In addition, he serves as a visiting professor at the Faculty of Control Systems and Robotics at ITMO University (St. Petersburg, Russia) and a visiting professor at the Faculty of Electrical Engineering Department of Power Electronics, Electrical Drives and Robotics at Silesian University of Technology (Gliwice, Poland). Presently, he holds the position of professor in Mechatronics at the Department of Electrical Power Engineering and Mechatronics, School of Engineering, Tallinn University of Technology (TalTech). The main research interests are mechatronics and electrical drives, particularly for electric transportation and autonomous vehicles. He is a member of IEEE (S'12-M'16-SM'20) and the Estonian Society of Moritz Hermann Jacobi.

Andriy Partyshev received his BSc in engineering in 2021. He was a part of a competitive robotics team in the US for three years. He is currently a member of the Industrial Virtual and Augmented Reality Laboratory in the Department of Mechanical and Industrial Engineering of TalTech and participated in multiple internships offered by international companies. Andriy's primary research areas are robotization of industry, robot control via Industrial Digital Twin.

Viktor Rjabtšikov received his BSc and MSc degrees in electrical engineering in 2018 and 2020, respectively. He is currently a Ph.D. Student at Department of Electrical Power Engineering and Mechatronics in Tallinn University of Technology. In addition, Viktor has participated in various internships at different Estonian companies. Primary research interest relates to electrical drives and electrical machine control theory. In addition, his research interest includes electric vehicles propulsion systems and Digital Twins implementation to EV systems.

Appendix 8 – Digital Twin Service Unit for AC Motor Stator Inter-Turn Short Circuit Fault Detection

2021 28th International Workshop on Electric Drives: Improving Reliability of Electric Drives (IWED), Moscow, Russia. Jan 27 – 29, 2021

Digital Twin Service Unit for AC Motor Stator Inter-Turn Short Circuit Fault Detection

Viktor Rjabtšikov, Anton Rassõlkin, Bilal Asad, Toomas Vaimann, Ants Kallaste
Department of Electrical Power Engineering and Mechatronics
Tallinn University of Technology
Tallinn, Estonia
viktor.rjabtsikov@taltech.ee

Vladimir Kuts, Sergei Jegorov
Department of Mechanical and Industrial Engineering
Tallinn University of Technology
Tallinn, Estonia

Mariusz Stępień, Mateusz Krawczyk
Department of Power Electronics, Electrical Drives and Robotics
Silesian University of Technology,
Gliwice, Poland

2021 28th International Workshop on Electric Drives: Improving Reliability of Electric Drives (IWED) | 978-1-6654-1456-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/IWED52055.2021.9376328

***Abstract*—A modern trend for industry digitalization brings new demands for the development and application of the modeling and simulation approach. It is already not enough to have only a virtual representation of the object and run it independently from the physical object. The Digital Twin (DT) aspect indicates a connection between the physical object and the corresponding virtual twin established by generating real-time data using sensors. The DT represents physical object operation throughout its life cycle, making it an essential tool for improving that object's reliability. In this paper, an application of the DT service unit for AC motor stator inter-turn short circuit fault detection is presented. According to real-time measurements, Linux Robot Operation System (ROS) simulates AC electrical machine-specific behavior in case of unbalanced stator currents and notify about possible fault appearance and propagation. Fault, such as discussed in the paper (AC machine stator inter-turn) is considered one of the most prevalent possible electrical motor failure.**

Keywords*— *Industry applications, Motor drives, Model-driven development

I. INTRODUCTION

The Digital Twin (DT) is a trending technology. It defined as a virtual representation that collaborates with the physical object throughout its lifecycle and provides services for the evaluation, optimization, prediction, etc. [1] The very first concept of DT presented by M. Grieves [2] in 2014 was composed of three main components – the physical and virtual entities, supported by the data. The main limitation of such a concept highlighted in [3] is the virtual entity's application generally for simulation purposes. More recent works [4] represent DT technology as a five-dimension structure, with services and connections as separate entities. Different test benches, as reduced models of the complex system, may be introduced as an additional entity that may noticeably improve DT development and implementation process, as presented in [5]. Fig. 1 shows the 5-dimensional DT model and the components' synergy as the main component real physical entity provides the DT development basis. The virtual entity represents the physical entity in different approaches and is responsible for the observation, simulation, control, and optimization strategies of the DT system. The data entity acts as an operator for the whole system and develops the DT itself. The service entity will be more detailed discussed in the paper. It represents a platform with a set of services and responds to the demands of either physical and virtual entities. The last entity that completes the five-dimensional DT model is a connection that defines the interaction between other entities. In addition to regulations in the motor drive system [6], service entities may include sub-services, such as path planning, energy optimization, maintenance, diagnostic, etc.

The proper, reliable, accurate, and efficient fault detection DT service units may be a valuable tool throughout the electrical motor-drive life cycle. Literature review shows that preventive maintenance programs reduce total electrical motor rewinds from 85% to 20% of the entire motor repairs [7]. The reasons behind the failures in rotating electrical machines are a combination of one or more issues that have their origin in design, manufacturing tolerance, assembly, installation, working environment, nature of the load, and maintenance schedule. AC machine stator inter-turn fault is considered one of four [8] most prevalent possible electrical motor failures, together with air gap eccentricity, broken rotor bar/end-rings, and bearing failures. Usually, the inter-turn short circuit starts with insulation failure, accompanied by extremely high current flow due to the high voltage potential differences between adjacent coils.

This paper presents the study on inter-turn short circuit fault detection in the induction machine's stator (IM), IM is a popular choice in industrial drive applications. However, IM's stator construction is the same as in the case of other

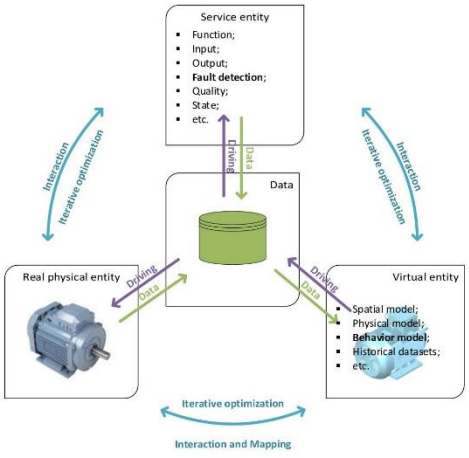


Fig. 1. Five-dimensional Digital Twin model of AC motor.

The research has been supported by the Estonian Research Council under grant PSG453 "Digital twin for propulsion drive of autonomous electric vehicle".

978-0-7381-4443-6/21/$31.00 ©2021 IEEE

AC machines, and fault detection algorithms using specified DT service units may be used for different types of AC electrical machines.

The paper presents the DT service unit application for AC motor stator inter-turn short circuit fault detection and is structured as follows. The nature of the AC electrical motor stator fault is discussed in Section II. The definition of fault imitating measurements for DT development is presented in Section III. The behavior of Linux Robot Operation System (ROS) node according to real-time measurements of a real physical entity (AC electrical motor) and specific behavior model, dedicated to detect unbalanced stator currents and notify about possible fault appearance and propagation, is presented in Section IV.

II. FAULT DIAGNOSTICS AND DETECTION

Unlike preventive and reactive maintenance, predictive maintenance is gaining heightened popularity [7]. The fault detection of electrical machines at the incipient stage for predictive maintenance is essential for a safe and reliable industrial operation. This is also vital for machine life estimation as the faults are degenerative. Any machine under ideal conditions should be perfectly symmetrical for all of its phases. But practically, the asymmetry is investable. The main contributors to those asymmetries are the electrical and mechanical faults. Nearly all faults can be divided into two classes: electrical and mechanical. The most common of them are rotor faults [9], such as bad bearings, broken bars, eccentricity, and winding short circuits. These fault's leading causes may include thermal degradation, hazardous industrial environment, bad foundation, and magnetic stress and vibrations. The winding insulation degradation is slow but a continuous process that can lead to a catastrophic situation. This can lead to the faults such as inter-turn short circuit, phase to phase short circuit, or phase to ground short circuit. Moreover, the increased asymmetry among phase impedances can increase the speed and torque ripples, which can cause other mechanical faults. Almost all faults modulate the supply current with a specific bandwidth of frequencies. Being present in the current, they influence the other parameters such as speed, torque, flux, and voltage, etc.

The detection of those frequency components at the early stage of the fault can avoid significant damage. [10] In induction machines, all fault dependent harmonics are the function of slip. This divides the signal under observation into two categories: the transient and the steady-state. In a steady-state regime, the signal is stationary, and the standard signal processing techniques can be used for fault detection. Among several signal processing techniques, the discrete-time Fourier transform (DTFT) is being used successfully. This is because it can be used on a piece of equipment with low computation power and can give a good insight into the harmonics. Since the fault-based harmonics are dependent on the slip, DTFT fails to provide any meaningful information under no and low load conditions. Another problem of DTFT is the spectral leakage, which can hide all small-amplitude faulty harmonics. In the transient regime, the signal is non-stationary due to varying slip. Hence the time-frequency analysis becomes essential. It may lead to a specific frequency pattern as, during the transient period, the slip changes its value from one to nominal. The signal analysis in the transient interval reduces the problems related to the load dependency of faulty frequency components. The most common time-frequency techniques include short-time Fourier transform (STFT), wavelet transform (WT), and multiple signal classification (MUSIC), etc.

The electrical motors should have minimal speed and torque ripples. In induction machines, the most prominent causes of those ripples are because of the current harmonics. The primary sources of currents harmonics in induction machines are the supply-based, inherit eccentricity, bad bearings, bad foundation, and the presence of any fault. Moreover, the thermal, skinning, and proximity effect also reduces the symmetry of winding electrical parameters such as resistance. The non-symmetrical three-phase impedances produce negative sequence currents in the motor, increasing the speed and torque ripples. These ripples can become a cause for more mechanical faults due to the increase in the vibration. The problem becomes worst with the degrading winding insulation resulting in short circuit failures. Various techniques can be used to detect the short circuit early, such as Park and Clark's vector, extended Park's vector, Park's vector modulus, symmetrical components, pattern recognition-based advanced techniques, etc.

III. EXPERIMENTAL RESULTS

Finding faults in the early stages of the machine work is advantageous when planning and maintaining the machine. Confirmation of a DT fault can be carried out by verifying the mathematical model in which the real physical model's accurate data is continually being sent. A massive amount of data is needed to properly train the mathematical model, so the fault detection's final result will be more accurate.

An imbalance occurs in the stator windings with an inter-turn short circuit, where the resistance decreases in the winding with a turn-to-turn short circuit. For experiments with a smooth decrease in the first phase of the winding resistance, an adjustable resistor was used, connected in parallel to the winding's first phase. By adjusting the parallel-connected resistor, the total resistance of the first phase winding changes and, at the same time decreasing current passing by winding by directing some of the phase current to the resistor. Suppose the resistance of the regulated resistor is equal to the winding resistance of the first phase. In that case, the current passing through the first phase will be divided exactly in half, which gives 50% of the fault, or in other words, an inter-turn short circuit between half of the winding. The illustrative figure is shown in Fig. 2.

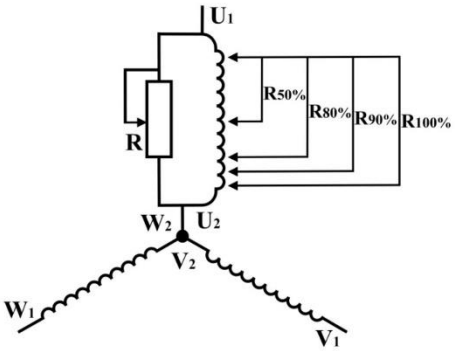


Fig. 2. Stator winding of induction machine with parallel connected regulated power resistor.

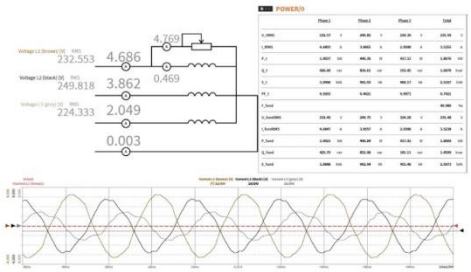


Fig. 3. Example of laboratory simulation and fault data acquisition.

There are several advantages to connecting a resistor in parallel. The first is the possibility of testing the motor without harming it and without changing the winding side of the stator. The second is the ability to measure an error of one percent where the change in current will be minimal, but at the same time, it is essential for the rapid response of the fault detection system. And the third is the ability to adjust the fault percent over a very large interval. This resistance interval depends only on the number of resistors in parallel, where it is possible to achieve any winding resistance.

The test bench contained two 7,5 kW induction motors, where one was used as a driving motor and the other as the loading motor. For data acquisition, current clamps, and DEWETRON data acquisition systems (DAS) were used. Both induction motors were connected to the star connection. The driving motor was connected directly to the grid to eliminate harmonics that can be carried out by a frequency converter. First, the tests were carried out with an intact and faulty motor where the results could be compared. The points of reference for us were faults of 1%, 2%, and 5%; for each fault point, there were four stages of load: no load, 25%, 50%, and 75%. Lastly, two different scenarios were carried out where the neutral point was connected and disconnected from the motor.

Comparing the graphs of the current of different percentages of failure, then a dependence appears that the greater the percentage of failure, the greater the currents' asymmetry. Also, the more load on the motor, the less harmonics in the current are visible, and there is less curvature of the current. And lastly, a disconnected neutral point not only increases the current asymmetry but also affects the voltage shape and amplitude. In Fig. 3. examples of DEWETRON DAS data with faulty currents and with a disconnected neutral point are shown.

Data collected during the experiment were transferred to a ROS-based server, where the studied motor's DT is located. The next section describes the structure of the DT service and presents an example of data processing. To test different file formats performance with ROS in the current setup, measurement data from the real induction motor were saved into files with various extensions (*.mat, *.xlsx, *.csv, *.txt). This data is further fetched into the ROS, transformed to ROS messaged, and is advertised on the topics.

IV. DT Service Unit Description

The developed DT's virtual entity consists of models' set, spatial model, physical model, behavior model, rule model, etc. In the spatial model, the studied electrical motor parts are constructed as a computer-aided geometric model to be assembled in ROS's virtual engine. In the physical model, the performance of separate parts of the physical entity simulated using numerical computing environments, like MATLAB [11], FEMM [12], Agros2D [13], etc. The behavior model is the main focus of the current research paper; it is responsible for transfer data from the real physical entity, calculating motor parameters, and stream to the ROS topics available for models. Rule model covering constraints for road load can be simulated through behavior analysis and data associations that can be observed using virtual sensors.

ROS acts as a physics engine for the current setup of the DT. It simulates the real induction motor's behavior and features and acts as a publisher of data to be used by virtual models. ROS has a publisher/subscriber architecture, it allows models from different environments to publish or subscribe to the ROS topics and interact with them. The communication between various platforms is handled using ROS bridge – a node that converts ROS messages into JavaScript Object Notation (JSON) or Message Queuing Telemetry Transport (MQTT) formatted data. Subscription or publishing to ROS bridge's port allows direct communication with ROS nodes themselves. Due to the JSON standard format, any model in a virtual environment can be programmed to receive or publish messages to ROS topics.

Input signal tracking is performed through the following process: when measurement data from the motor windings come as an input current to ROS, it is being compared against the expected data, and if the received measurement on any of the winding exceeds the margins determined by the admitted error, a fault notification is filed. The measured data is being summed up and compared for a specific amount of time that is necessary to calculate an optimal error, therefore allowing the system to accurately tell if there is a fault in the windings of the motor. After the specific time elapses, the sum value and error are refreshed.

The fault detection algorithm realized for DT is shown in Fig. 4. ROS gets connected to the real motor (for this article, recorded data of the motor was used to simulate the real input from the motor). The current data received from motors is being processed, converted to ROS messages, and is published in the ROS environment, as shown in Fig. 5.a. At the same time, the node listening to the currents topic starts receiving messages. The current values retrieved from these messages are processed as follows: a motor phase is chosen as the base phase. Relatively to this phase, we convert other phase currents to imaginary units. These imaginary units can be used to represent the percentage of load applied on the motor. As a rule of thumb, these values should be very close to one another, e.g., within the allowed margins (due to noises and imperfections). If one of the imaginary units exceeds the set margins, then a warning message is being published by ROS (used to notify the model) and is output in the terminal (shown in Fig. 5b). The published message can be used as an indicator in the 3D model or real test bench of a potential fault in the system.

V. Discussion

Today's technology development level allows even smartphones [14] or any portable recording device to be used for condition monitoring. However, making a service dataset available for condition monitoring and fault diagnosis DTs'

can contribute even more efficiently to electrical energy conversion systems. DT assets enable system users to view the behavior of last in real-time and apply practical knowledge gained with a plant system. DT's application allows using hybrid analyzing methodologies to contribute to computational modeling and simulation of complex problems that appear in numerous multidisciplinary applications. The DT's main principles have an immediate and critical application to fault detection in the energy conversion systems, especially in electric drives. However, there are fundamental risks, like worthless complexity and usefulness of possible DT services. For some applications, DT may be an unconscionable complicated, or expensive or technology.

Due to its nature, DT may be designed so that services give an early indication of the system's possible malfunctions. In that case, maintenance can be scheduled appropriately on an "as needs" [15] basis rather than at fixed time intervals. Nowadays, DT technology is implemented in different industrial applications; some successfully implement various diagnostics services. A literature review of DT's possible implementation for wind turbine condition monitoring and fault diagnosis is given in [16]. The application of DT in vehicles is a popular topic in the recent decade. In 2012 researchers from NASA in [17] suggested using DT to mirror its twin's life and enable unprecedented levels of safety and reliability. DT must integrate high-fidelity simulation with the portable integrated monitoring system, maintenance history, and all available historical and fleet data by the author's approach. A more recent example is given in [18] by K. Shubenkova et al., authors suggest to track data of KAMAZ trucks failures throughout the logistic process and to predict failures of each particular vehicle. Proposed DT compared with the real numbers of failures confirm the forecasts' adequacy at the level of 10%.

VI. Conclusion

In this research work, we have implemented inter-turn short circuit fault detection into the DT of the induction motor. A spatial model developed in Unity 3D was combined with ROS service that allows online condition monitoring. The studied fault was simulated in laboratory conditions to verify the ROS nodes behavior. Basically, the emulator was created from historical data and a physical/mathematical model of the induction motor.

Developed DT can serve as a virtual sensor for the separate electric motor or be a part of complex electrical energy conversion systems. Further investigation is needed for extending the services assets of developed DT. DT services that use signal processing and pattern recognition algorithms for different fault detection are constructed and analyzed. Combining data from plant systems, virtual sensors, and machine learning routines will allow more precise diagnosis and prognosis possibilities for the physical entity and may introduce degradation services (e.g., efficiency loss, demagnetization, etc.) of the physical entity.

References

[1] A. Rassõlkin, V. Rjabtšikov, T. Vaimann, A. Kallaste, V. Kuts, and A. Partyshev, "Digital Twin of an Electrical Motor Based on Empirical Performance Model," 2020.

[2] M. Grieves, "Digital Twin : Manufacturing Excellence through Virtual Factory Replication This paper introduces the concept of a A Whitepaper by Dr . Michael Grieves," *White Pap.*, no.

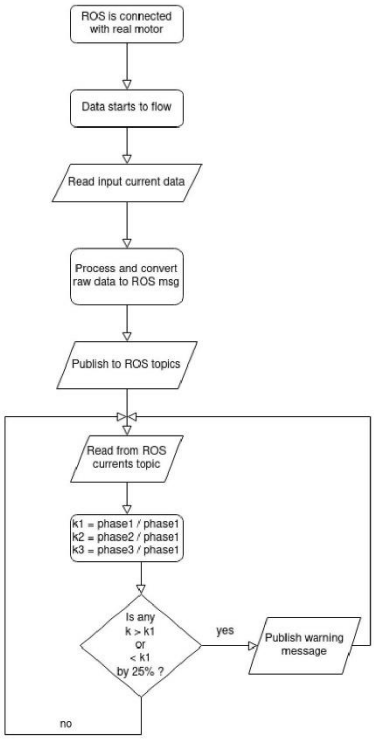


Fig. 4. Simple inter-turn short circuit fault detection algorithm realized for current DT.

a)

```
sejego@Lenovo-IdeaPad-S500:~/catkin_ws$ rostopic echo /tb/loadin
g_motor/motor_status/phase
current1: 100.0
current2: 95.0299987929688
current3: 78.08999633789062
---
current1: 100.0
current2: 95.05999755859375
current3: 78.17999267578125
---
current1: 100.0
current2: 95.0999984741211
current3: 78.29999542236328
---
current1: 100.0
current2: 95.1500015258789
current3: 78.44999694824219
---
current1: 100.0
current2: 95.20999908447266
current3: 78.62999725341797
---
```

b)

```
sejego@Lenovo-IdeaPad-S500:~/catkin_ws$ rosrun tb_digital_twin erro
Check
[ WARN] [1608652023.995218443]: Potential malfunction in windings
[ WARN] [1608652025.995143909]: Potential malfunction in windings
[ WARN] [1608652027.995158397]: Potential malfunction in windings
[ WARN] [1608652029.995140284]: Potential malfunction in windings
[ WARN] [1608652031.995157019]: Potential malfunction in windings
```

Fig. 5. ROS operation terminal at a) data processing and b) warning message when the fault is detected.

March, 2014, [Online]. Available: https://www.researchgate.net/publication/275211047_Digital_Twin_Manufacturing_Excellence_through_Virtual_Factory_Replication.

[3] A. Rassõlkin, T. Vaimann, A. Kallaste, and V. Kuts, "Digital twin for propulsion drive of autonomous electric vehicle," 2019.

[4] F. Tao, M. Zhang, and A. Y. C. Nee, "Five-Dimension Digital Twin Modeling and Its Key Technologies," in *Digital Twin Driven Smart Manufacturing*, 2019, pp. 63–81.

[5] A. Rassõlkin, V. Rjabtšikov, T. Vaimann, A. Kallaste, and V. Kuts, "Concept of the Test Bench for Electrical Vehicle Propulsion Drive Data Acquisition," 2020.

[6] A. Rassõlkin *et al.*, "Digital Twin for Propulsion Drive of Autonomous Electric Vehicle," in *60th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, Oct. 2019, pp. 1–4, doi: 10.1109/RTUCON48111.2019.8982326.

[7] B. Asad, T. Vaimann, A. Rassõlkin, A. Kallaste, and A. Belahcen, "Review of Electrical Machine Diagnostic Methods Applicability in the Perspective of Industry 4.0," *Sci. J. Riga Tech. Univ. - Electr. Control Commun. Eng.*, vol. 14, no. 2, pp. 108–116, 2018, Accessed: Oct. 12, 2019. [Online]. Available: https://ecce-journals.rtu.lv/article/view/2554.

[8] P. Zhang, Y. Du, T. G. Habetler, and B. Lu, "A survey of condition monitoring and protection methods for medium-voltage induction motors," *IEEE Transactions on Industry Applications*, vol. 47, no. 1. pp. 34–46, Jan. 2011, doi: 10.1109/TIA.2010.2090839.

[9] B. Asad, T. Vaimann, A. Rassõlkin, A. Kallaste, and A. Belahcen, "A Survey of Broken Rotor Bar Fault Diagnostic Methods of Induction Motor," *Electr. Control Commun. Eng.*, vol. 14, no. 2, pp. 117–124, 2019, doi: 10.2478/ecce-2018-0014.

[10] B. Asad, T. Vaiman, A. Belahcen, A. Kallaste, A. Rassolkin, and M. N. Iqbal, "Modified Winding Function-based Model of Squirrel Cage Induction Motor for Fault Diagnostics," *IET Electr. Power Appl.*, pp. 1–13, May 2020, doi: 10.1049/iet-epa.2019.1002.

[11] L. Gevorkov, A. Rassolkin, A. Kallaste, and T. Vaimann, "Simulink based model for flow control of a centrifugal pumping system," in *2018 25th International Workshop on Electric Drives: Optimization in Control of Electric Drives, IWED 2018 - Proceedings*, Mar. 2018, vol. 2018-Janua, pp. 1–4, doi: 10.1109/IWED.2018.8321399.

[12] J. Pando-Acedo *et al.*, "Hybrid FEA-Simulink Modelling of Permanent Magnet Assisted Synchronous Reluctance Motor with Unbalanced Magnet Flux," in *Proceedings of the 2019 IEEE 12th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives, SDEMPED 2019*, Aug. 2019, pp. 174–180, doi: 10.1109/DEMPED.2019.8864925.

[13] D. Pánek, T. Orosz, and P. Karban, "Artap: Robust Design Optimization Framework for Engineering Applications," *2019 3rd Int. Conf. Intell. Comput. Data Sci. ICDS 2019*, Dec. 2019, Accessed: Feb. 24, 2020. [Online]. Available: https://arxiv.org/abs/1912.11550.

[14] T. Vaimann, J. Sobra, A. Belahcen, A. Rassõlkin, M. Rolak, and A. Kallaste, "Induction machine fault detection using smartphone recorded audible noise," *IET Sci. Meas. Technol.*, vol. 12, no. 4, pp. 554–560, Jul. 2018, doi: 10.1049/iet-smt.2017.0104.

[15] J. Penman, M. N. Dey, A. J. Tait, and W. E. Bryan, "Condition monitoring of electrical drives," *IEE Proc. B Electr. Power Appl.*, vol. 133, no. 3, pp. 142–148, 1986, doi: 10.1049/ip-b.1986.0019.

[16] A. Rassõlkin *et al.*, "Implementation of Digital Twins for electrical energy conversion systems in selected case studies," *Proc. Est. Acad. Sci.*, vol. 70, no. 1, 2021.

[17] E. H. Glaessgen and D. S. Stargel, "The digital twin paradigm for future NASA and U.S. Air force vehicles," *Collect. Tech. Pap. - AIAA/ASME/ASCE/AHS/ASC Struct. Struct. Dyn. Mater. Conf.*, pp. 1–14, 2012, doi: 10.2514/6.2012-1818.

[18] K. Shubenkova, A. Valiev, E. Mukhametdinov, V. Shepelev, S. Tsiulin, and K. H. Reinau, "Possibility of Digital Twins Technology for Improving Efficiency of the Branded Service System," *Proc. - 2018 Glob. Smart Ind. Conf. GloSIC 2018*, pp. 1–7, 2018, doi: 10.1109/GloSIC.2018.8570075.

Appendix 9 – ROS middle-layer integration to Unity 3D as an interface option for propulsion drive simulations of autonomous vehicles

IOP Conference Series: Materials Science and Engineering

PAPER • OPEN ACCESS

ROS middle-layer integration to Unity 3D as an interface option for propulsion drive simulations of autonomous vehicles

To cite this article: Vladimir Kuts *et al* 2021 *IOP Conf. Ser.: Mater. Sci. Eng.* **1140** 012008

ROS middle-layer integration to Unity3D as an interface option for propulsion drive simulations of autonomous vehicles.

Vladimir Kuts[1*], Anton Rassõlkin[2], Andriy Partyshev[1], Sergei Jegorov[1], Viktor Rjabtšikov[2]

[1]Tallinn University of Technology, School of Engineering, Department of Mechanical and Industrial Engineering, Estonia, Harju, Tallinn, Ehitajate str. 5, 19086
[2]Tallinn University of Technology, School of Engineering, Department of Electrical Power Engineering and Mechatronics, Estonia, Harju, Tallinn, Ehitajate str. 5, 19086

[*]vladimir.kuts@taltech.ee

Abstract. As autonomous vehicle development continues at growing speeds, so does the need for optimization, diagnosis, and testing of various autonomous systems elements, under different conditions. However, since such processes should be carried out in parallel, it may result in bottlenecks in development and increased complexity. The trend for Digital Twins brings a promising option for the diagnosis and testing to be carried out separately from the physical devices, incl. Autonomous vehicles, in the virtual world. The idea of intercommunication between virtual and physical twins provides possibilities to estimate risks, drawbacks, physical damages to the vehicle's drive systems, and the physical one's critical conditions. Although the problem of providing communications between these systems arises, at the speed that will be adequate to represent the physical vehicle in the virtual world correctly, it is still a trending topic. The paper aims to demonstrate a way to solve this problem - by using ROS as a middleware interface between two twining systems on the autonomous vehicle propulsion drive example. Data gathered from the physical and virtual world can be exchanged in the middle to allow continuous training and optimization of the propulsion drive model, leading to more efficient path planning and energy-efficient drive of the autonomous vehicle itself.

1. Introduction

Simulation is an approximate or 1-to-1 imitation of a real process, often taking part in the virtual environment, troubleshooting, researching, testing, training, monitoring, controlling, or educating. In the past decade, simulations have been vital in production and development as they are capable of preventing many problems related to planning and reducing bottlenecks at early stages, also during the real-time maintenance of the process [1]–[5] and, especially with increasing technology complexity and rise in using fully autonomous systems, enforcing and changing work-safety features. One side of the simulation aspect - the concept of Digital Twin (DT) [6], [7] is being exploited in the related research to develop a precise dual-way synchronized simulation interface for the propulsion drives [8], [9] to be ready to be integrated into the electrical vehicles [10].

Physics simulations are very common and critical nowadays. They are used enormously in such applications as MATLAB Simulink, Simscape, CAD design, SolidWorks, etc., and gaming physical

processes simulations. They should be considered in the mechatronic systems' planning stage [11]. Of course, they all have approximation and simplifications, while not all possible physical laws can be yet simulated simultaneously; however, such simulations provide considerable benefit in research and testing.

In a previous study done by authors of the related paper, which was on the on electrical motors simulation under development of DT for propulsion drive of an autonomous electric vehicle [12], Unity3D was used for simulations of DT that was exchanging messages with Robot Operation System (ROS) node through a ROS bridge [13]. However, ROS is not being used only for robots but also for various drones, self-driving vehicles, and autonomous systems. ROS enables inter-process communication; it is believed to be a quality method of interconnecting a digital twin propulsion drive system with its real counterpart. ROS was used for performance calculation using an empirical performance model for induction motor (IM). As a visualization tool in the related research is being used Unity3D which is connected with ROS directly [14]. Even though Unity3D simulated most of the motor's physical behavior (torque and rotation), the response and received numerical values, unfortunately, do not suit the DT development in the long run. The reason for this is the complexity of the overall system of physics of IM. Moreover, to make the system transferable and usable with other models (not the ones present in Unity3D but also in Gazebo or elsewhere) the physics handling has to be close to standalone.

The research's main aim is to develop a framework and a toolkit, including a middle-layer ROS interface connected with the physical propulsion drive workbench and its DT, which can be visualized in various simulation engines. The related paper aims to develop a methodology to connect the interface with Unity3D for the visualization, considering data exchange and feedback.

2. Methodology

2.1. Working principle of a test bench on a digital twin

For the current case study, the DT operates on the simulated data generated based on real data measured and gathered from the 7,5 kW IM (ABB 3GAA132214-ADE). The data was gathered using the data acquisition system (DAS) Dewetron Dewe 2 and saved into files with a different extension (*.mat, *.xlsx, *.csv, *.txt). The measured data can be anything regarding the motor's operation, namely input currents and voltages, consumed and shaft powers, torque and angular velocity on data acquisition, and other side data calculated from them. According to DAS tuning (16Hz - 100kHz), the parameters can be measured with different frequencies, and received data is relative to time. This feature enables to recreate of the motor's behavior precisely as it happened in the real case scenario with the help of ROS Server. An example of such can be seen in Figure 1, where the input current from frequency converter to IM was recorded and now can be simulated in ROS (graph from ROS package *rqt plot* we were not included to the related paper because it could not handle plotting messages at such high frequency).

In the proposed DT system, ROS Server acts as a data server and physics simulator. The idea behind it is the following: the server is a standalone subsystem of a TB DT that is responsible for processing real, measured data of the motor, calculating other motor parameters based on the processed data, and streaming to the ROS topics available for models.

Figure 2 features the architecture of the DT setup for TB. The real data is fetched to appropriate ROS Nodes (components of ROS server that are performing calculations, real data processing, and streaming of data) present in the server, processed and translated into ROS messages, and finally, sent to the DT model over ROS Bridge. The real data can be based on the empirical model/map of the motor (or its part) or the actual raw data.

Upon receiving ROS messages, the model can perform the necessary actions to simulate the mechanical/electrical/thermal behavior. Models can be present in any simulation environment. They are subscribed to ROS Server's topics over API or ROS Bridge and configured to perform the necessary operations based on the subscribed ROS topic (for example, rotation based on received angular speed). Furthermore, the module can feature simulated 'measurement' devices/sensors that can send back the

data over the ROS bridge. In this case, the ROS Nodes can process and calculate other required values, as it would happen in the real TB.

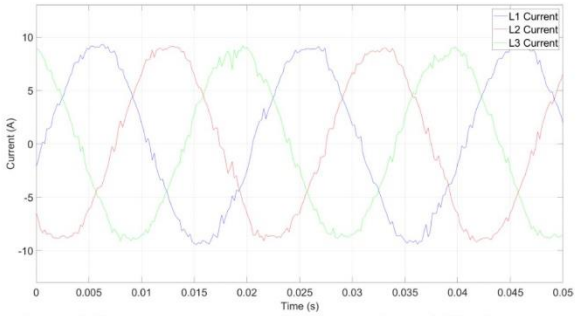


Figure 1. Input current measurements sampled at 5kHz frequency

The current DT of TB consists of the Unity 3D model and ROS Server. ROS Server streams simulated values regarding input power (3-phase current and voltages), efficiency calculated based on measured torque and angular velocity. The torque is calculated by the physics engine of the Unity3D, whereas other values are based on the real ones. This creates a problem of incorrect data calculation because Unity does not focus on calculating correct values on physics laws, as it is more for games, allowing developers to adjust the physics laws to the game setup. This is why the shift from the physics engine of the model environment to ROS was introduced. ROS server would serve physical parameters based on the real TB data and independent of the modeling environment.
Additionally, ROS can record *rosbags* – files with recorded values from topics/servers that can be played back to repeat the behavior. Such a feature would allow us additional analytical features from the DT side.

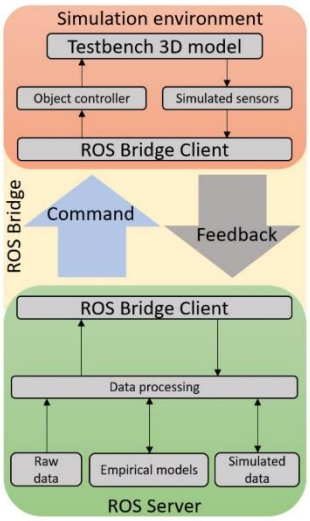


Figure 2. The generic architecture of TB DT

2.1.1. ROS Interfacing

To allow easy interfacing of ROS with other systems, a ROS Bridge node has to be used. It converts ROS communications into a JSON file format and sends them outside of the ROS ecosystem. JSON is used because of its universal format with existing libraries that support its serialization and deserialization in almost every modern programming language. Taking it one step further, ROS Bridge can be used to port specific ROS topics to and out of Message Queuing Telemetry Transport (MQTT) protocol to upscale the system and allow it to run on multiple machines around the world. This so-called MQTT Bridge sends data to the remote server by taking the serialized message on a specified ROS topic and publishes it into a specified MQTT topic. MQTT Bridge is also capable of the inverse - it receives a JSON-serialized message and attempts to deserialize it into a specified ROS topic in a specific message type. Together these systems make interfacing of ROS with any visualization solution much simpler to develop. To further simplify the deserialization process, classes that match ROS message types were created in C# for Unity3D implementation of the ROS interface. This approach can be considered the most efficient because, in this case, a ROS message delivered in the serialized form via MQTT can be directly deserialized into an object of a matching type. This approach can be implemented in similar ways on the majority of existing programming languages, making it the most straightforward and most versatile option.
Visualization is being done in Unity3D (See Fig. 3) engine connected to the physics simulator via ROS Interface, where it is a 1 to 1 scale propulsion drive model with the transmission, wheel parts, and non-visible gears. Model is being assembled as the physical one, and each part is being controlled by a related script, where data is being fed from the middle layer.

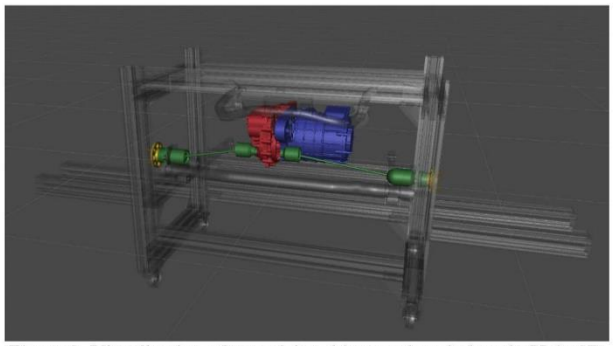


Figure 3. Visualization of propulsion drive test bench done in Unity3D

3. Discussion

The primary outcome of the related part of the more extensive research in developing the fully synchronized DT of the propulsion drive is that the ROS interface was developed. It is possible to feed it with gathered from the physical data and give to the visual simulated, which in related use-case is being Unity3D. The given data simulation runs and gives logged feedback about physical interactions back to the ROS middle layer, where the model is being improved and sent back to the visual side, improving it after each data movement loop. However, some limitations were met during the development of methodology, and more developments go to reach the final aim of the stated research aim (See Table 1).

Table 1. Limitations and further steps

Limitations	Future steps
The model was tested with only one type of visual simulation tool. Possible additional integrations should be done in the middle layer to be suitable for additional software tool packages.	To establish correct torque calculations based on the real values collected from the physical TB.
	To implement a two-way connection between physical TB and its DT.
If DT and TB work simultaneously over the internet, the frequency of data acquisition may be too high to send on time, the possibility of lags	The injection process flow of new components of TB into the DT.
	To create unpredicted behaviors in the system, trigger points, and try to make the system respond to the unpredicted change making it more adaptive to changes

4. Conclusions

The ROS interface connected with the Digital Twin of the propulsion drive workbench visualized in Unity3D was introduced during the related work. Raw and simulated data and empirical models can be post-processed and fed to the visual simulation, where additional data is being logged and given as feedback to the middleware to improve the model and physical simulation itself. The next crucial step is to feed physical simulation directly with data from the physical drive, enabling synchronization between the real and virtual worlds through the developed interface.

5. Acknowledgments

The research has been supported by the Estonian Research Council under grant PSG453 "Digital Twin for Propulsion Drive of Autonomous Electric Vehicle."

References

[1] R. AhmadiAhangar, A. Rosin, A. N. Niaki, I. Palu, and T. Korõtko, "A review on real-time simulation and analysis methods of microgrids," *International Transactions on Electrical Energy Systems*. 2019, doi: 10.1002/2050-7038.12106.

[2] S. Venkatesan, K. Manickavasagam, N. Tengenkai, and N. Vijayalakshmi, "Health monitoring and prognosis of electric vehicle motor using intelligent-digital twin," *IET Electr. Power Appl.*, 2019, doi: 10.1049/iet-epa.2018.5732.

[3] G. Turner, "Soaring through virtual aviation: The role of VR in aerospace manufacturing," *Manufacturing Global*, 2020. .

[4] L. Gevorkov, A. Rassolkin, A. Kallaste, and T. Vaimann, "Simulink based model of electric drive for throttle valve in pumping application," in *2018 19th International Scientific Conference on Electric Power Engineering, EPE 2018 - Proceedings*, Jun. 2018, pp. 1–4, doi: 10.1109/EPE.2018.8395996.

[5] I. Rasheed *et al.*, "Fast Numerical Techniques Based Analysis of Electromagnetic Problems Using MATLAB," in *Proceedings - 12th International Conference on Frontiers of Information Technology, FIT 2014*, Jun. 2015, pp. 115–120, doi: 10.1109/FIT.2014.30.

[6] N. Kousi, C. Gkournelos, S. Aivaliotis, C. Giannoulis, G. Michalos, and S. Makris, "Digital twin for adaptation of robots' behavior in flexible robotic assembly lines," in *Procedia Manufacturing*, 2019, doi: 10.1016/j.promfg.2018.12.020.
[7] V. Kuts, M. Sarkans, T. Otto, T. Tähemaa, and Y. Bondarenko, "Digital Twin: Concept of hybrid programming for industrial robots – Use case," in *ASME International Mechanical Engineering Congress and Exposition, Proceedings (IMECE)*, 2019, vol. 2B-2019, doi: 10.1115/IMECE2019-10583.
[8] V. Kuts *et al.*, "Synchronizing physical factory and its digital twin through an IIoT middleware: a case study," *Proc. Est. Acad. Sci.*, vol. 68, no. 4, pp. 364–370, 2019, doi: 10.3176/proc.2019.4.03.
[9] A. Rassõlkin, T. Vaimann, A. Kallaste, and V. Kuts, "Digital twin for propulsion drive of autonomous electric vehicle," in *60th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, 2019.
[10] N. Khaled, B. Pattel, and A. Siddiqui, "Digital Twin development and cloud deployment for a Hybrid Electric Vehicle," in *Digital Twin Development and Deployment on the Cloud*, 2020.
[11] R. Sell, E. Coatanéa, and F. Christophe, "Important aspects of early design in mechatronic," in *Proceedings of the International Conference of DAAAM Baltic* , 2008.
[12] A. Rassõlkin *et al.*, "Digital Twin for Propulsion Drive of Autonomous Electric Vehicle," in *60th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, Oct. 2019, pp. 1–4, doi: 10.1109/RTUCON48111.2019.8982326.
[13] A. Rassõlkin, V. Rjabtšikov, T. Vaimann, A. Kallaste, V. Kuts, and A. Partyshev, "Digital Twin of an Electrical Motor Based on Empirical Performance Model," in *Proceeding of 2020 XI International Conference on Electrical Power Drive Systems (ICEPDS)*, 2020.
[14] E. Sita, C. M. Horváth, T. Thomessen, P. Korondi, and A. G. Pipe, "ROS-Unity3D based system for monitoring of an industrial robotic process," in *SII 2017 - 2017 IEEE/SICE International Symposium on System Integration*, 2018, doi: 10.1109/SII.2017.8279361.

Appendix 10 – Conceptual Modelling of an EV-Permanent Magnet Synchronous Motor Digital Twin

Conceptual Modelling of an EV-Permanent Magnet Synchronous Motor Digital Twin

Mahmoud Ibrahim
Department of Electrical Power Engineering and Mechatronics
Tallinn University of Technology
Tallinn, Estonia
mahmoh@taltech.ee

Viktor Rjabtšikov
Department of Electrical Power Engineering and Mechatronics
Tallinn University of Technology
Tallinn, Estonia
viktor.rjabtsikov@taltech.ee

Sergei Jegorov
Department of Electrical Power Engineering and Mechatronics
Tallinn University of Technology
Tallinn, Estonia
sejego@taltech.ee

Anton Rassõlkin
Department of Electrical Power Engineering and Mechatronics
Tallinn University of Technology
Tallinn, Estonia
anton.rassolkin@taltech.ee

Toomas Vaimann
Department of Electrical Power Engineering and Mechatronics
Tallinn University of Technology
Tallinn, Estonia
toomas.vaimann@taltech.ee

Ants Kallaste
Department of Electrical Power Engineering and Mechatronics
Tallinn University of Technology
Tallinn, Estonia
ants.kallaste@taltech.ee

***Abstract*— Digital twin (DT) technology has contributed to the development process of many applications, including electric vehicles (EVs). The DT concept is to create a digital representation of the real physical asset and support its performance by utilizing simulation and optimization tools fed with real data. DT technology can be used to solve general problems related to EV motors, such as estimation of the driving torque and the internal rotor temperature. This paper provides the concepts for implementing a DT of an EV permanent magnet synchronous motor (PMSM) based on its analytical performance model. DT architecture comprises two main components: virtual model and real-time data exchange set. The motor physical model (test bench) was provided in detail. An analytical performance q-d mathematical model supported by the motor equivalent circuit was explained. The motor virtual model was built based on the proposed analytical model using MATLAB/Simulink. ROS2 node, implemented on a microcontroller,was used for real-time data exchange between the physical and the virtual motor models. The main target is to monitor the physical motor performance and estimate its torque through its digital twin. The obtained results from the DT showed the effectiveness of the proposed method.**

Keywords—Digital Twin, Modelling, Permanent magnet synchronous motor, Virtual sensor

Nomenclature

u_{sd}, u_{sq}	d- and q-axis stator voltage components
$\lambda_{sd}, \lambda_{sq}$	d- and q-axis flux linkage components
λ_{pm}	Permanent magnet flux linkages
i_{sd}, i_{sq}	d- and q-axis stator current components
ω_e	Electrical angular velocity
p	Number of poles pairs
l_{sd}, l_{sq}	d- and q-axis inductances
r_s	Stator winding resistance
E_{sq}	d- and q-axis induced EMF components
T_e, T_m	Motor electrical torque, mechanical load torque
J	Rotor inertia

The research has been supported by the Estonian Research Council under grant PSG453 "Digital twin for propulsion drive of an autonomous electric vehicle".

I. INTRODUCTION

Digital twin (DT), as a definition, is used to create and maintain a digital representation of a real physical object, asset, process, or service. It's, in essence, a computer program that uses real-world data to create simulations that can predict how a product or process will perform[1].

While simulations and DTs both use digital models to replicate products and processes, the difference between them is that DT creates a virtual environment able to study several simulations, backed up with real-time data and a two-way flow of information between the twin and the sensors that collect this data. This increases the accuracy of predictive analytical models, offering a greater understanding of the management and monitoring of products, policies, and procedures. DT can be applied to an electric vehicle (EV) for different usages such as health monitoring, diagnostics, prognostics, optimization, scenario, and risk assessment [2]. It can be created at the system level, subsystem level, individual component level, and many other assets.

The electric motor is considered the core element of the propulsion system of any EV. It must meet the EV requirements of high performance, high torque/power density, and mainly high operational efficiency [3]. To achieve that, two main conditions must be fulfilled; proper motor design and an efficient control algorithm. Most EV motors depend on torque-based control strategies [4]. Torque estimation is critical for EV manufacturers as it is a vital variable in powertrain management and energy-saving strategies. It is preferable to avoid using torque transducers in EV motors for reasons concerning size, cost, and mechanical positioning challenges.

DT is offering an efficient solution for the above issue. In a connected context, a cross-platform engine Unity 3D was used as a virtual environment for a DT for monitoring an induction motor based on an empirical performance model [1]. Inter-turn short circuit fault detection was implemented into a DT of an AC 3-phase induction motor (IM) in [5]. DT of an EV motor to optimize the motor performance concerning estimating driving torque and cooling control based on a micro lab box was proposed in [6]. Monitoring and conditions analyzing DT of an IM based on a simulation finite element method (FEM) model were addressed in [6]. DT model for fault detection of a 50 MW permanent magnet synchronous motor (PMSM) based on a numerical analysis model was

discussed in [7]. Health monitoring and lifetime prediction of an EV PMSM were done by implementing an intelligent DT model based on MATLAB/Simulink and a mixed fuzzy logic and artificial neural network discussed in [8].

In this research paper, the concept of DT of an EV-PMSM is proposed. The paper is organized as follows; An overview of DT technology and its areas of applications for EV propulsion motors is discussed in section I. Section II illustrates the mathematical dynamic model derivation and equivalent circuits of PMSM. Section III presents the main architecture of the motor DT, which is divided into three subsections. The reduced physical model (test bench) of EV PMSM is presented in section III.A. Data exchange set (communication) model between physical and virtual models is proposed in III.B. Section III.C provides the virtual (Simulation) model of the motor. The concept of DT is achieved by liking the virtual and physical model based on Robot Operating System V.2 (ROS2) framework. Obtianed primilary results are discussed in section VI. Conclusion and future works are addressed in section V.

II. EV-PMSM Analytical Performance Model

The stator flux linkage vector of PMSM can be drawn in the rotor refernce frame (d-q) and, statator reference frame (α-β), as shown in Fig.1.

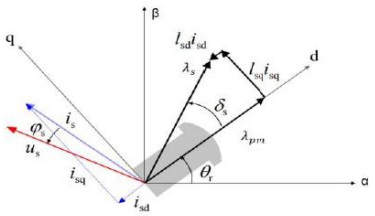


Fig. 1. Vector diagram of a PMSM. Stator reference frame (α-β) and rotor reference frame (d-q).

When the rotor reference frame is considered, the equivalent d- and q-axis stator windings are transformed into the reference frames revolving at rotor speed. The consequence is that there is zero speed differential between the rotor and stator magnetic fields, and the stator d- and q-axis windings have a fixed phase relationship with the rotor magnet axis, which is the d axis in the modeling [9].

Fig. 2 shows the d-q equivalent circuit of PMSM from which the following equations are deduced.

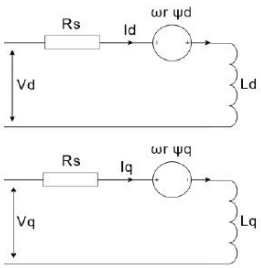


Fig. 2. PMSM d-q equivalent circuit.

The stator voltage equations can be written in synchronous *d–q* reference frame as follows:

$$u_{sq} = R_s i_{s_d} + \frac{d}{dt}\lambda_d - \omega_e \lambda_{sq}, \quad (1)$$

$$u_{sq} = R_s i_{sq} + \frac{d}{dt}\lambda_q + \omega_e \lambda_{sd}, \quad (2)$$

where the flux linkage equation can be given below.

$$\lambda_{sd} = \lambda_{pm} + l_{sd} i_{sd}, \quad (3)$$

$$\lambda_{sq} = l_{sq} i_{sq}. \quad (4)$$

So, the stator voltage equations can be rewritten as follows:

$$u_{sd} = r_s i_{sd} + l_{sd}\frac{d}{dt} i_{sd} + E_{sd}, \quad (5)$$

$$u_{sq} = r_s i_q + l_q \frac{d}{dt} i_q + E_{sq}, \quad (6)$$

where

$$E_{sd} = -\omega_e l_{sq} i_{sq}, \quad (7)$$

$$E_{sq} = \omega_e(\lambda_{pm} + l_{sd} i_{sd}), \quad (8)$$

taking Laplace transformation then $d/dt = s$, then the stator voltage matrix can be expressed as the follow:

$$\begin{bmatrix} u_{sd} \\ u_{sq} \end{bmatrix} = \begin{bmatrix} r_s + s l_{sd} & \omega_e l_{sq} \\ \omega_e l_{sd} & r_s + s l_{sq} \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_e \lambda_{pm} \end{bmatrix} \quad (9)$$

The electromagnetic torque equation can be defined as follows

$$T_e = \frac{3}{2} p\, i_{sq}(\lambda_{pm} + (l_{sd} - l_{sq}) i_{sd}) \quad (10)$$

The mechanical equation of the motor is driven from the general machine swing equation as follows:

$$J\frac{d\omega_e}{dt} = T_e - T_m \quad (11)$$

III. Digital Twin of EV-PMSM

As previously discussed, that DT is a replica of an existing physical model. It consists of three main parts as follows: physical model, Data exchange set (communication model), and virtual simulation model.

A. A.Physical Model (test bench)

The motor physical model was taken from ISEAUTO to be a real representation of an EV motor. ISEAUTO was built on a Mitsubishi i-MiEV trolley based on a Y4F1 PMSM [10]. Table. 1 present ISEAUTO motor parameters.

TABLE I. UNDERSTUDY PMSM PARAMETERS

Parameter	Description	Value	Unit
p	Number of pole pairs	4	-
r_s	Stator resistance	0.07	Ω
l_q	q-axis inductance	$4.4 \cdot 10^{-4}$	H
l_d	d-axis inductance	$5.7 \cdot 10^{-4}$	H
λ_{pm}	Permanent magnet linkage flux	0.279	Web
N_r	Rated speed	3000	rpm
P_r	Rated output power	35	kW

The motor test bench is set as the following. The motor is driven by an EV Inverter (ABB HES880) with direct torque control (DTC) algorithm. It's also equipped with a resolver decoder for speed and position measurement on its shaft. The EV inverter was fed by a battery emulator system (Cinergia B2C+30). The inverter was driven by a visual interface unit.

Voltage and current sensors were placed on the motor input terminals to collect stator voltage and current input signals. Voltage sensors were also connected to the resolver main and auxiliary windings to collect motor speed and position data. Fig.3 shows the experimental motor test bench.

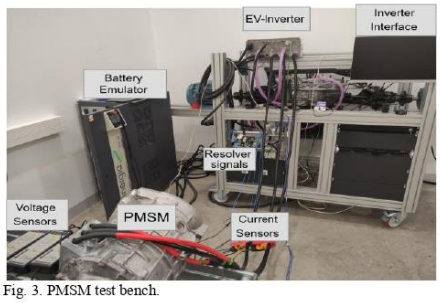


Fig. 3. PMSM test bench.

B. *Data Exchange set (Communication model).*

The outputs of the test bench sensors were connected to ROS2 Foxy node implemented on a (Teensy 4.0) board. Teensy 4.0 is a microcoprocessor development board manufactured by PJRC. It features ARM Cortex-M7 processor, float point math units, 1984 KB of Flash and 1024 KB RAM memory and features a total of 40 GPIO pins. Real time data are received via the subscriber of MATLAB ROS2 Toolbox. The subscriber is a node that subscribes to a topic and processes the received data. The received data are ROS2 messages from the Digital Twin middleware, and each message is a structure consisting of three-phase currents, three-phase voltages, and resolver signals. Upon reception, the stator phase voltage and resolver signals are extracted by the subscriber and are sent to the simulation model for further analysis and processing. Simulated real time data of motor torque is sent to an interface module (Work station) to be analysed. As a future step, the analysed data is transferred into data commends to control the real motor model. Fig. 4 shows the operational architect of DT.

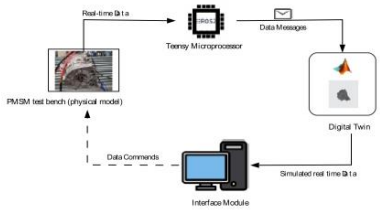


Fig. 4. The operational architecture of Digital Twin

C. *Simulation Model*

The motor simulation model was built using MATLAB/Simulink on the basis of deriviated PMSM analytical d-q equations from section II. Two ROS2 subscribers were implemented in the simulation model to enable data exchange with the physical model.

Subscriber 1 collects real-time data of stator voltage and resolver position coming from the test bench and then uses them as an input for the simulation model. Second, *subscriber 2* is responsible for receiving real-time data of stator current from the physical model to be compared with the result from the simulation. Stator current comparator is used initially for DT tuning in and checking the simulation model accuracy. The resolver decoder block processes the resolver winding signals coming from *subscriber 1*, transferring them into a position value input to the motor block. Measurement block contains scopes to observe motor simulation outputs of electromagnetic torque, angular speed, and the compared stator phase current. Fig.5. shows the simulation model of PMSM with the two ROS2 subscribers.

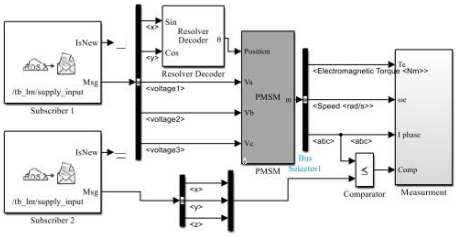
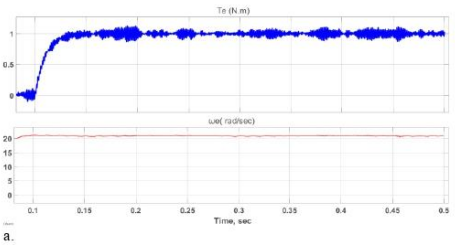


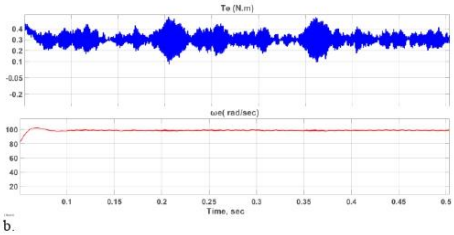
Fig.5. PMSM digital model

IV. Results and Discussion

The objective of this research is to demonstrate the effectiveness and readiness of the DT concept for the EV-motor torque estimation. The physical model was run in parallel with the simulation virtual model of the motor under two operating speeds cases in no loading conditions to validate the simulation model's accuracy in real-time. Stator voltage and rotor position were fed to the simulation model in real-time through the ROS2 node. Stator current from the physical model received in *subscriber 2* was compared with results from simulation. Motor Torque and speed can be observed through the scopes. Fig. 6 a, b shows the resultant motor electromagnetic torque rotor and angular speed obtained from the simulation model for two operating speed cases.



a.



b.

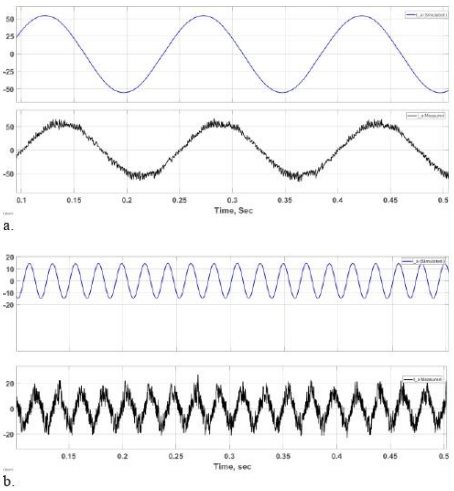
Fig. 6 Estimated Torque – Speed vs time from simulation, a – and b -

From Fig. 6 it can be noticed the reduction in noload torqur (inertia) with the increased speed that explained by the control algorithm of EV-inverter.

Torque in fig.6 a takes longer stability time than that in

fig .6 b as the speed in case a. starts from stand still to low speed but in case b it mutates from low speed to high speed.

Fig.7 shows stator phase current obtained from the simulation model and received from the test bench for the same two operating speed cases.



a.

b.

Fig. 7. Stator current simulated and measured vs time.a – and b -

Fig. 7 a, b shows the highly agreement between measured value of stator phase current from testbench and its counterpart derived from the simulation model is clear, this proves the simulation effectiveness.

From Figure 7 a and b, it can be noticed that the value of the stator current in case a is higher than that in case b, that makes sens with the increased motor speed.

There is some noticeable noise in the phase current waveform coming from the testbench compared to that obtained from the simulation at an acceptable level with no fundamental differences due to the current sensors' accuracy.

V. CONCLUSION

This paper proposes a conceptual methodology for designing a DT of an EV- PMSM. DT is objected to estimating the motors' electromagnetic torque. The MATLAB/Simulink virtual simulation model was built based on the analytical performance model. The concept of the DT of real-time implementation was achieved by linking both virtual and physical models by ROS2 nodes data messages exchange features.

The obtained results of estimated torque from the virtual model are promising and prove the methodology's effectiveness.

What was presented in this research are the basics that can be developed in the future to formulate a complete DT model through which a strategy to control the physical motor torque can be applied.

VI. REFERENCES

[1] A. Rassolkin, V. Rjabtsikov, T. Vaimann, A. Kallaste, V. Kuts, and A. Partyshev, "Digital Twin of an Electrical Motor Based on Empirical Performance Model," Oct. 2020, doi:10.1109/ICEPDS47235.2020.9249366.

[2] M. Ibrahim, A. Rassõlkin, T. Vaimann, and A. Kallaste, "Overview on Digital Twin for Autonomous Electrical Vehicles Propulsion Drive System," Sustain. 2022, Vol. 14, Page 601, vol. 14, no. 2, p. 601, Jan. 2022, doi: 10.3390/SU14020601.

[3] N. A. Elsonbaty, M. A. Enany, and M. I. Hassanin, "An Efficient Vector Control Policy for EV-Hybrid Excited Permanent-Magnet Synchronous Motor," World Electr. Veh. J. 2020, Vol. 11, Page 42, vol. 11, no. 2, p. 42, May 2020, doi: 10.3390/WEVJ11020042.

[4] Y. Guo, Z. Liu, W. Deng, and F. Blaabjerg, "Modeling of electric vehicle driven by PMSM based on torque control," Proc. 2012 2nd Int. Conf. Instrum. Meas. Comput. Commun. Control. IMCCC 2012, pp. 1020–1024, 2012, doi: 10.1109/IMCCC.2012.241.

[5] V. Rjabtsikov et al., "Digital Twin Service Unit for AC Motor Stator Inter-Turn Short Circuit Fault Detection," Jan. 2021, doi: 10.1109/IWED52055.2021.9376328.

[6] F. Toso, R. Torchio, A. Favato, P. G. Carlet, S. Bolognani, and P. Alotto, "Digital twins as electric motor soft-sensors in the automotive industry," in 2021 IEEE International Workshop on Metrology for Automotive, MetroAutomotive 2021 - Proceedings, Jul. 2021, pp. 13–18, doi: 10.1109/MetroAutomotive50197.2021.9502885.

[7] H. Brandtstaedter, L. Hübner, A. Jungiewicz, C. Ludwig, E. Tsouchnika, and U. Wever, "DIGITAL TWINS FOR LARGE ELECTRIC DRIVE TRAINS Copyright Material PCIC Europe Paper No. PCIC Europe EUR18_04 Germany Germany Germany," 2018.

[8] S. Venkatesan, K. Manickavasagam, N. Tengenkai, and N. Vijayalakshmi, "Health monitoring and prognosis of electric vehicle motor using intelligent-digital twin," IET Electr. Power Appl., vol. 13, no. 9, pp. 1328–1335, Sep. 2019, doi: 10.1049/IET-EPA.2018.5732.

[9] D. S. More, H. Kallum, and B. G. Fernandes, "D-q equivalent circuit representation of three-phase flux reversal machine with full pitch winding," PESC Rec. - IEEE Annu. Power Electron. Spec. Conf., pp. 1208–1214, 2008, doi: 10.1109/PESC.2008.4592094.

[10] A. Rassõlkin, R. Sell, and M. Leier, "Development Case Study of the First Estonian Self-Driving Car, ISEAUTO," Electr. Control Commun. Eng., vol. 14, no. 1, pp. 81–88, 2018, doi: 10.2478/ecce-2018-0009.

Appendix 11 – Novel Digital Twin Concept For Industrial Application. Study Case: Propulsion Drive System

Proceedings of the ASME 2022
International Mechanical Engineering Congress and Exposition
IMECE2022
October 30-November 3, 2022, Columbus, Ohio

IMECE2022-97243

NOVEL DIGITAL TWIN CONCEPT FOR INDUSTRIAL APPLICATION. STUDY CASE: PROPULSION DRIVE SYSTEM

Sergei Jegorov, Anton Rassõlkin, Viktor Rjabtšikov, Mahmoud Ibrahim
Tallinn University of Technology, School of Engineering, Department of Electrical Power Engineering and Mechatronics
Tallinn, Estonia

Vladimir Kuts
University of Limerick, Electronic and Computer Engineering Department
Ireland, Limerick

ABSTRACT

This paper introduces a novel concept of interconnecting components of Digital Twins using ROS2 and micro-ROS frameworks. Authors of the research argue that middleware implementation plays the most important role in performance of Digital Twins. Propulsion Drive System Digital Twin is presented where method of connecting hardware and software components is explained through a case study. Interface between traction motor of the Propulsion Drive System and software service entity is implemented on a basis of a microcontroller running micro-ROS. The suggested method of connecting components of Digital Twins is tested through round-trip time latency test. Results of the test indicate that the proposed concept is suitable for Digital Twin technologies in industrial field.

Keywords: Digital Twins, ROS2, micro-ROS, autonomous vehicles, MATLAB

1. INTRODUCTION

Global trending in sustainable and smart manufacturing is pushing technological industries such as automotive, aerospace, marine to develop new means of developing,
manufacturing and maintaining the systems they create. One of technologies that promises to bring manufacturing to the next level are Digital Twins (DT) [1] - cyber-physical systems meant to duplicate the behaviour of real systems, objects, facilities and environment. Such technology provides possibilities to industries to control every step of their manufacturing - starting with facility planning and ending with digitalization of the manufacturing itself for the purpose of optimizing processes. In this paper, we would like to focus on a very important field of manufacturing of our age - automotive. Since the term DT was introduced in 2002, the DT technology has been slowly but surely getting more and more attention from large automotive industries. Just by looking at the results of the search presented in Figure 1, we can imagine how important and widespread the DT trend has become.

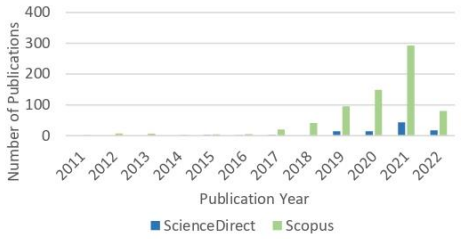


FIGURE 1: RESULTS OF SEARCH[1] FOR PUBLICATIONS RELATED TO DT IN AUTOMOTIVE INDUSTRY.

With growth of interest in DT technology, there has also been an increase in available frameworks and platforms for development of such. To name a few: Azure Digital Twins [2],Google Supply Chain Twin [3] , AWS IoT TwinMaker [4], etc. These platforms use middleware frameworks extensively under the hood to perform communication between various components of DT

[1] Search query for title, abstract, keywords containing "Digital Twin vehicle", "Digital Twin car" or "Digital Twin automotive" terms.

systems. Often times, the right choice of the platform and/or framework is going to determine how well DT is going to perform. In previously conducted research, Jegorov et al. [5] make a comparison of available middleware frameworks that are suitable for industrial and automotive applications. The choice ultimately went to ROS2 [6] – the successor of ROS (Robot Operating System). In this paper, we would like to extend the reach of ROS2 by adding its low-level counterpart – micro-ROS [7] and present a part of DT that gathers low-level motor data to calculate physical parameters of the motor that are difficult to measure in real life.

2. BACKGROUND

Digital Twin for propulsion drive of autonomous electric vehicle (PSG453) [8] is a research project which aims to develop a specialized, unsupervised analysis and prognosis tool for autonomous vehicles. The concept of this project consists of four modules (can be seen in Figure 2): real vehicle that is supplied with sensors (real physical entity), Test Bench (TB) of a propulsion drive that is present in the vehicle (designated as testbenches – laboratory setups that represent some part of a real system), 3D models of the TB/real vehicle (virtual entity), the monitoring system (service entity).

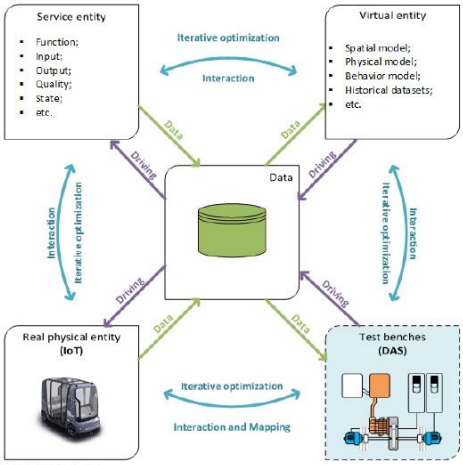


FIGURE 2: ARCHITECTURE OF DIGITAL TWIN FOR PROPULSION DRIVE OF AUTONOMOUS ELECTRIC VEHICLE *[9]*

All communication is handled via the middleware, through which DT data is going to flow. The data is being sent and received from all modules to all modules simultaneously, thus enabling to create sophisticated controls, analysis of the whole DT in real time. Data is being generated by real physical vehicle (either recorded data or if operated online), by TB (during operation), virtual entity (generates data mainly as feedback to reflect changes on real vehicle/TB but can be used as the main data driving entity) and service entity, mainly warnings, alerts and in-depth analytical data of some gathered operation data from TB.

2.1 Physical entity

The physical entity of the DT is presented as a TB consisting of a propulsion drive system (PDS) identical to the one present inside the ISEAUTO. The PDS features a Mitsubishi PMSM traction motor Y4F1 (present in i-MiEV car models, used for operation of the car) which is operated by an ABB HES880 - frequency converter that transforms the supply power to the motor based on the set parameters. HES880 in its turn, is powered by a Cinergia B2C+ battery system. Y4F1 motor's output is attached to a shaft via a gearbox. The shaft is fixed to two ABB Induction loading motors (ABB 3GAA132214-ADE) that simulate the loads the system is subject to. Two loading motors are connected to two ABB ACS880 frequency converters that transform the supply power to the motor based on the set parameters. The PDS is attached to a metallic frame which enables operation of the system and allows connection of other elements to the system (controllers, converters, sensors etc.).

2.2 Virtual entity

Virtual entity is represented as a 3D copy of the TB created in a virtual environment of a Unity game engine. The virtual entity is composed of imported CAD geometric models of all the parts of PDS (motors, shafts, bearings, gearbox), thus keeping real dimensions and parameters of the TB. Implemented software in Unity controls parts of the 3D model and can simulate motion and action depending on the provided input. Likewise, the virtual entity can have virtual sensors that record the simulated data from operation of the 3D model of the PDS in a virtual environment and is able to stream it back to the physical entity via middleware.

2.3 Service entity

Service system represents an integrated service platform responding to the demands of both physical and virtual systems and acts as a predictive maintenance component [8]. It monitors the operation of TB, analyses any found abnormalities in operation to find the cause of them and ultimately warns about abnormalities of the system. One of the implementations is described in [9], where inter-turn short circuit faults were detected and analyzed in MATLAB/Simulink software during operation. It is important to note, that MATLAB/Simulink cannot be used in situations where receiving high-frequency data is required, as it does not have enough processing power to do that effectively. In such cases, MATLAB Code generator is used to migrate the software made in MATLAB/Simulink on a lower-level implementation to ensure efficient handling of high-frequency data.

2.4 Interconnecting middleware

The element that ties everything together is the middleware of the DT – it is responsible for transport of data between all the entities. It is implemented using the ROS2 framework. ROS2

provides a reliable, decentralized publisher-subscribe communication mechanism which is based on a DDS standard. Entities themselves consist of a multitude of components that can communicate data, and they are defined as ROS2 Nodes. In ROS2, when communicating, these Nodes publish and/or subscribe to topics to receive data and perform some actions based on this data. All entities of the DT, when transporting data, use custom-defined messages that are available on ROS topics, prefixed with the group name, indicating the entity that the data belongs to. The custom-defined messages are all part of the collection (or ROS package) named *digital_twin_msgs*.

2.5 Study case – traction motor with service entity integration

In this paper, we will investigate the integration of traction motor with service entity. The goal is to establish the connection between a PMSM analytical model of the Y4F1 traction motor and the motor itself, namely with the supply input power of the motor. The supply input power of the motor is the output of the ABB HES880 frequency converter. Based on the supplied voltage, for an electrical motor that is not subjected to any load, it is possible to compute angular velocity and torque of the motor as described in [10]. The corresponding service entity with interfaces to middleware can be seen in Figure 3. It is important to note, that this service entity model was generated into a C++ ROS2 node using Simulink Code Generator.

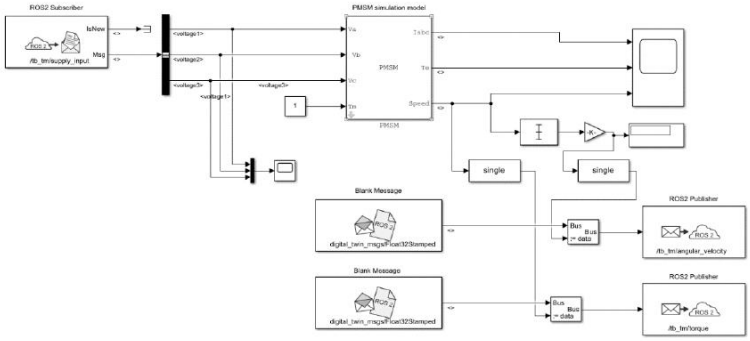


FIGURE 3: SERVICE ENTITY WITH PMSM ANALYTICAL MODEL.

3. DEVELOPMENT OF MIDDLEWARE BETWEEN TRACTION MOTOR AND SERVICE ENTITY

Development of middleware for the mentioned problem consists of 2 subtasks:

1. Defining the structure of data that is involved in the process.
2. Implementing the hardware/software interface between traction motor and the service entity.

3.1 Defined data message structures

The supply input power consists of mainly two parameters that are of highest importance – the input AC current and input AC voltage. As these parameters are interrelated, they should be coupled together. For this reason, custom ROS2 message was defined (Figure 4). As can be seen, the structure of this message consists of 3-phase voltage and 3-phase current containers and a variable to store the time (typically time when data was recorded). The voltage and current containers are defined in their respective messages as seen in Figure 5.

Defined message structure will be used for serialization by a Node responsible for interfacing with the hardware (i.e., traction motor), and by the service entity for receiving the data.

```
builtin_interfaces/time stamp
digital_twin_msgs/Current currents
digital_twin_msgs/Voltage voltages
```

FIGURE 4: DEFINED MESSAGE STRUCTURE FOR SUPPLY INPUT

```
float32 current1        float32 voltage1
float32 current2        float32 voltage2
float32 current3        float32 voltage3
```

FIGURE 5: DEFINED MESSAGE STRUCTURE FOR AC CURRENT AND VOLTAGE CONTAINERS

3.2 Hardware-software interface

The supply input of the traction motor is generated by ABB HES880 frequency converter. The generated current and voltages are known to be periodic signals alternating at a frequency ~20 Hz. In order to record these signals, Teensy 4.0 MCU [11] was used to sample the data at 1 kHz frequency (to get precise signal sampling and avoid aliasing).

To get the signal into a range that Teensy 4.0 MCU can sample, three devices, known as current clamps, were attached to the cables that connect HES880 output terminals with Y4F1 traction motor's input terminals. The current clamps have a specific conversion ratio and transform the AC current and voltage signals to the following signals: -400mV to +400mV periodic wave that represents AC current in range -40A to +40A; and -1.4V to +1.4V periodic wave that represents Ac voltage in range -1400 mV to +1400 mV. Because most ADCs (analog-to-digital converter) present in MCUs (including Teensy 4.0) can only process positive analog signals, the output signals of current clamps must be brought to only positive range. For this, a level shifter was used. To serialize the sampled data into ROS messages and sent it to ROS2 middleware, micro-ROS framework was used for writing the software on Teensy 4.0. micro-ROS connected to the micro-ROS agent running on the same machine where service entity was and did all the serialization and message transport. Figure 6 illustrates the described hardware/software interface, including gathering of data, serialization and transport of messages to the service entity.

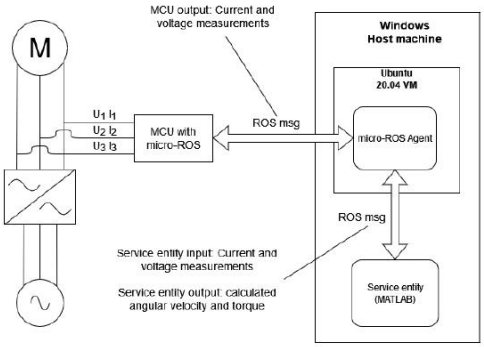


FIGURE 6: HARDWARE/SOFTWARE INTERFACE BETWEEN TRACTION MOTOR AND SERVICE ENTITY

4. RESULTS AND ANALYSIS

In order to validate that implemented solution can be used in real-time, a latency test was conducted. For this specific case, RTT (round trip time, visual representation can be seen in Figure 8) latency test was chosen, due to MCU and host machine possessing different clocks. Different clocks may not be properly synchronized, leading to false results.

To conduct the RRT latency test, a new message type was defined consisting of a message ID and time stamp. Every message was generated by Teensy 4.0 MCU and sent to a ROS2 listener operating in the same environment as service entity. The listener, upon receiving the message, verified that the message was not lost (by comparing the expected message ID with the received one) and simply sent it back to the Teensy 4.0 MCU. When the MCU received its messages back, it calculated the approximate time it took for a message to return. To avoid running out of memory, the Teensy 4.0 did not store the latencies locally, and forwarded them to a specially created ROS2 Node on a host machine that later calculated mean, maximum and minimum latency. The test was conducted for 60000 messages. The latency was measured in microseconds, since the clock of Teensy 4.0 is capable of recording time with microsecond precision.

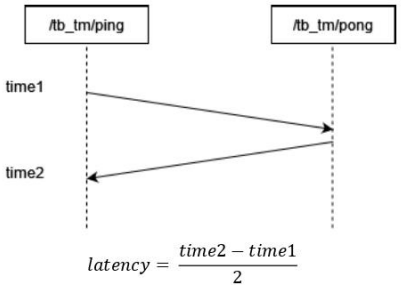


FIGURE 7: ILLUSTRATION OF RTT LATENCY TEST

Latency test between Teensy 4.0 MCU and service entity in MATLAB/Simulink yielded the following results: 60000 out of 60000 messages were successfully delivered, mean latency was 197 us, maximum latency was 6594 us, min latency being 151 us. Such reliable and low-latency communication suggests that the implemented feature can operate in real-time, which is a key feature of DT technology.

5. CONCLUSION

In this research paper the authors investigated a novel method of interfacing elements of DT. ROS2 and micro-ROS frameworks were used as middleware for transport and serialization of data. Hardware component of DT – traction motor, was connected to another component of DT – service entity. The conducted latency test suggests that reliability and low latency of the method is suitable for use in DT technologies.

ACKNOWLEDGEMENTS

The research is supported by the Estonian Research Council under grant PSG453 "Digital Twin for Propulsion Drive of Autonomous Electric Vehicle". In addition, Vladimir Kuts has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 847577; and a research grant from Science Foundation Ireland (SFI) under Grant Number 16/RC/3918 (Ireland's European Structural and Investment Funds Programmes and the European Regional Development Fund 2014-2.

REFERENCES

[1] M. Grieves, "Origins of the Digital Twin Concept," August 2016. [Online]. Available: 10.13140/RG.2.2.26367.61609. [Accessed 05 03 2022].

[2] Microsoft, "Azure Digital Twins," [Online]. Available: https://azure.microsoft.com/en-us/services/digital-twins/. [Accessed 26 04 2022].

[3] Google, "Google Supply Chain Twin," [Online]. Available: https://cloud.google.com/solutions/supply-chain-twin. [Accessed 26 04 2022].

[4] Amazon, "AWS IoT TwinMaker," [Online]. Available: https://aws.amazon.com/iot-twinmaker/. [Accessed 26 04 2022].

[5] S. Jegorov, A. Rassõlkin, V. Kuts, V. Rjabtšikov and A. Partyshev, "The comparison between ROS and ROS2 based on the propulsion drive of autonomous vehicle," 2022.

[6] Open Robotics, "ROS," [Online]. Available: https://www.ros.org/. [Accessed 26 04 2022].

[7] J. Staschulat, I. Lütkebohle and R. Lange, "The rclc Executor: Domain-specific deterministic scheduling mechanisms for ROS applications on microcontrollers: work-in-progress," in *2020 International Conference on Embedded Software (EMSOFT)*, 2020.

[8] A. Rassõlkin, T. Vaimann, A. Kallaste and V. Kuts, "Digital twin for propulsion drive of autonomous electric vehicle," in *2019 IEEE 60th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, 2019.

[9] V. Rjabtšikov, A. Rassõlkin, B. Asad, T. Vaimann, A. Kallaste, V. Kuts, S. Jegorov, M. Stępień and M. Krawczyk, "Digital Twin Service Unit for AC Motor Stator Inter-Turn Short Circuit Fault Detection," in *2021 28th International Workshop on Electric Drives: Improving Reliability of Electric Drives (IWED)*, 2021.

[10] M. Ibrahim, A. Rassõlkin, S. Jegorov, V. Rjabtšikov, T. Vaimann and A. Kallaste, "Conceptual Modelling of an EV-Permanent Magnet Synchronous Motor Digital Twin," 2022.

[11] PJRC, "Teensy® 4.0 Development Board," PJRC | Electronic Components Available Worldwide, [Online]. Available: https://www.pjrc.com/store/teensy40.html. [Accessed 18 04 2022].

[12] Gartner, "Gartner Identifies the Top 10 Strategic Technology Trends for 2019," October 2018. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-10-15-gartner-identifies-the-top-10-strategic-technology-trends-for-2019. [Accessed 06 03 2022].

[13] ROS Wiki, "ROS Topics," 2017. [Online]. Available: http://wiki.ros.org/Topics. [Accessed 24 03 2022].

[14] A. Rassõlkin, V. Rjabtšikov, T. Vaimann, A. Kallaste and V. Kuts, "Concept of the Test Bench for Electrical Vehicle Propulsion Drive Data Acquisition," in *2020 XI International Conference on Electrical Power Drive Systems (ICEPDS)*, 2020.

[15] V. Kuts, A. Rassõlkin, A. Partyshev, S. Jegorov and V. Rjabtšikov, "ROS middle-layer integration to Unity 3D as an interface option for propulsion drive simulations of autonomous vehicles," *IOP Conference Series: Materials Science and Engineering,* vol. 1140, p. 012008, May 2021.