FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Maurício Losso Salvador

Development of a Data Acquisition Software for Heidenhain Controllers to Achieve Process Monitoring and Workpiece Quality Prediction

Aachen
2022

Maurício Losso Salvador

Development of a Data Acquisition Software for Heidenhain Controllers to Achieve Process Monitoring and Workpiece Quality Prediction

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Academic Advisor: Prof. Rômulo Silva de Oliveira, Dr.
Local Supervisor: Sven Schiller, M.Sc.

Aachen
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Salvador, Maurício Losso
Development of a data acquisition software for Heidenhain controllers to achieve process monitoring and workpiece quality prediction / Maurício Losso Salvador ; orientador, Rômulo Silva de Oliveira, coorientador, Sven Schiller, 2022.
102 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Aquisição de dados. 3. Controladores Heidenhain. 4. Usinagem. 5. Monitoramento de processo. I. Oliveira, Rômulo Silva de. II. Schiller, Sven. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Maurício Losso Salvador

Development of a Data Acquisition Software for Heidenhain Controllers to Achieve Process Monitoring and Workpiece Quality Prediction

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, December 16, 2022.

ABSTRACT

The demands for large-scale production of complex workpieces is increasing worldwide. These complicated part designs lead manufacturers to employ CNCs in manufacturing in order to attain better efficiency and higher quality. Despite this fact, due to machining factors and the associated technology, it is still difficult to get the end products to the required specifications, as well as to monitor and inspect such workpieces and their production processes. A novel approach to solve these problems rely on technologies from the Industry 4.0, which consist in the implementation of data acquisition software for process-parallel monitoring machine tool operations. This change in the production line can offer more precise process control and quality defect analysis from part digitization. Considering this technological proposal, this thesis, along with gemineers GmbH, aims to develop a data acquisition software for machine tools operated by Heidenhain controllers. To develop the desired application, initial studies were conducted to have its requirements and modeling defined, resulting in a multi-threaded implementation based on four threads that function as state machines. Together, they were built for connecting to machine tools and obtaining data in both high and low frequency that, in turn, are further sent to a web server over the network using a communication protocol. When the development was over, the software was tested with an emulated controller setup for simulating the machining of a part. The data acquired from the operation was dispatched to the company's other services, which transform the information into a digital twin of the machined workpiece, later displayed in the existing Front-End. With digital visualization, the functionality of the data acquisition software was evaluated by comparing the shown results with the initially expected ones. In the end, performance tests were also conducted to analyze the behavior of the developed application and the related services. From the observations made, optimizations were proposed and implemented to achieve overall improvements for the gemineers' product. Visual results and in-depth analyses presented a successful data acquisition that obtains the necessary information to create a digital twin for process monitoring.

Keywords: Data acquisition. Heidenhain controllers. Machining. Process monitoring. Digital twin.

RESUMO

As demandas pela produção em larga escala de peças complexas está aumentando em todo o mundo. Esses modelos complicados de peças levam os fabricantes a empregar CNCs na fabricação a fim de alcançar uma melhor eficiência e maior qualidade. Apesar desse fato, devido a fatores de usinagem e à tecnologia associada, ainda é difícil levar os produtos finais às especificações exigidas, assim como monitorar e inspecionar tais peças e seus processos de produção. Uma nova abordagem para resolver esses problemas reside nas tecnologias da Indústria 4.0, que consistem na implementação de *software* de aquisição de dados para monitoramento paralelo ao processo das máquinas de usinagem em operação. Essa mudança na linha de produção pode oferecer um controle mais preciso do processo e análise de defeitos de qualidade a partir da digitalização de peças. Considerando essa proposta tecnológica, este projeto de fim de curso, juntamente com a gemineers GmbH, visa desenvolver um *software* de aquisição de dados para máquinas de usinagem operadas por controladores Heidenhain. Para desenvolver a aplicação desejada, foram realizados estudos iniciais para ter seus requisitos e modelagem definidos, resultando em uma implementação multi-tarefa baseada em quatro *threads* que funcionam como máquinas de estado. Juntas, elas foram construídas para conexão com máquinas de usinagem e obtenção de dados em alta e baixa frequência que, por sua vez, são posteriormente enviados para um servidor na internet através da rede usando um protocolo de comunicação. Quando o desenvolvimento terminou, o *software* foi testado com a configuração de um controlador emulado para simular a usinagem de uma peça. Os dados adquiridos da operação foram enviados aos outros serviços da empresa, que transformam as informações em um gêmeo digital da peça usinada, e posteriormente exibido no *Front-End* existente. Com a visualização digital, a funcionalidade do *software* de aquisição de dados foi avaliada através da comparação dos resultados mostrados com os inicialmente esperados. No final, também foram realizados testes de desempenho para analisar o comportamento da aplicação desenvolvida e dos serviços relacionados. A partir das observações feitas, foram propostas e implementadas otimizações para alcançar melhorias gerais para o produto da gemineers. Resultados visuais e análises aprofundadas apresentaram uma aquisição de dados bem sucedida que obtém as informações necessárias para criar um gêmeo digital para o monitoramento do processo de usinagem.

Palavras-chave: Aquisição de dados. Controladores Heidenhain. Usinagem. Monitoramento de processo. Gêmeo Digital.

LIST OF FIGURES

Figure 1 – Blade Integrated Disk (Blisk). 13
Figure 2 – gemineers logo. 15
Figure 3 – Digital twin example for a blisk. 19
Figure 4 – gemineers product areas. 20
Figure 5 – Structure of the Data Ground area. 21
Figure 6 – Working principle of the selected communication protocol. 22
Figure 7 – Threads from Batcher. 23
Figure 8 – Process Scheduler working principle. 25
Figure 9 – Application Programming Interface (API)s working principle. 26
Figure 10 – Usual coordinate system for a Computer Numerical Control (CNC). . 28
Figure 11 – CNC architecture. 29
Figure 12 – Open- and Closed-Loop systems for CNCs. 30
Figure 13 – Programming Station Software of a Heidenhain iTNC 530 controller. 31
Figure 14 – Overall Heidenhain Data Acquisition (DAQ) operating model. 35
Figure 15 – Activity diagram of Main Thread. 37
Figure 16 – Output data of the Identify Controller step. 38
Figure 17 – Standard procedure to connect and send data to DGServer. 39
Figure 18 – Activity diagram of Low-Frequency (LF) Streamer Thread. 43
Figure 19 – Example of low frequency data output from the LF Streamer 45
Figure 20 – Activity diagram of High-Frequency (HF) Acquisition Thread. 46
Figure 21 – Feed Rate channel metadata from the Scope area. 47
Figure 22 – Scope Information area structure. 48
Figure 23 – Activity diagram of HF Dispatch Thread. 50
Figure 24 – Original house geometry from iTNC 530 Programming Station. . . . 53
Figure 25 – Initial Programming Station setup for simulating a machining operation. 54
Figure 26 – Output data from the Batcher for the house.H. 55
Figure 27 – Resulting house geometry from gemineers' data processing services. 55
Figure 28 – gemineers' Front-End. 56
Figure 29 – Feed Rate data displayed as a point cloud in the Front-End. 57
Figure 30 – Heidenhain DAQ performance for Central Processing Unit (CPU) and memory. 60
Figure 31 – DGServer performance for CPU and memory. 60
Figure 32 – Batcher performance for CPU and memory. 61
Figure 33 – Different ways to handle batch data in the Data Ground services. . . 63
Figure 34 – Result comparison of single and batch data dispatching in the Front-End. 66

Figure 35 – Heidenhain DAQ performance for CPU and memory with two hundred batch data. 67
Figure 36 – DGServer performance for CPU and memory with two hundred batch data. 68
Figure 37 – Batcher performance for CPU and memory with two hundred batch data. 69
Figure 38 – Top view comparison of single and batch data dispatching in the Front-End for Feed Rate signal. 79
Figure 39 – Heidenhain DAQ performance for CPU and memory with twenty batch data. 79
Figure 40 – DGServer performance for CPU and memory with twenty batch data. 80
Figure 41 – Batcher performance for CPU and memory with twenty batch data. . 80
Figure 42 – Heidenhain DAQ performance for CPU and memory with five hundred batch data. 80
Figure 43 – DGServer performance for CPU and memory with five hundred batch data. 81
Figure 44 – Batcher performance for CPU and memory with five hundred batch data. 81
Figure 45 – Performance comparison of Data Ground services according to the batch size. 82
Figure 46 – Performance comparison of batch size according to the Data Ground services. 83

LIST OF TABLES

Table 1 – Data types and the associated frequency of acquisition. 34
Table 2 – Compression level and its respective closing time and final batch size. 71

LIST OF CODES

Code Snippet 1 First Heidenhain controller identification. 38
Code Snippet 2 First DGServer identification. 40
Code Snippet 3 Start of acquisition threads. 41
Code Snippet 4 Monitoring of the acquisition threads. 41
Code Snippet 5 Reset DAQ procedure. 41
Code Snippet 6 Low frequency data streaming. 44
Code Snippet 7 Scope channel selection for high frequency data acquisition. . 48
Code Snippet 8 Receiving high frequency data from the controller. 49
Code Snippet 9 Dispatching of high frequency data to the DGServer. 51
Code Snippet 10 Example of a general batch data function. 64
Code Snippet 11 Example of a general dispatch data function. 64

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
Blisk	Blade Integrated Disk
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CNC	Computer Numerical Control
CPU	Central Processing Unit
DAQ	Data Acquisition
G-Code	Geometric Code
HF	High-Frequency
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IP	Internet Protocol
IPT	Institute for Production Technology
LF	Low-Frequency
M-Code	Miscellaneous Code
MCS	Machine Coordinate System
SLM	Service Lifecycle Management
STL	Standard Triangle Language
TCP	Transmission Control Protocol
UML	Unified Modeling Language
WCS	Workpiece Coordinate System

CONTENTS

1 INTRODUCTION . . . 13
1.1 MOTIVATION . . . 13
1.2 GEMINEERS GMBH . . . 15
1.3 OBJECTIVES . . . 16
1.4 DOCUMENT STRUCTURE . . . 17
2 STATE OF THE ART . . . 18
2.1 GEMINEERS PRODUCT . . . 18
2.1.1 Digital Twin . . . 18
2.1.2 Microservices Architecture . . . 19
2.1.2.1 Data Ground Area . . . 20
2.1.2.2 Information Management Area . . . 24
2.1.2.3 Data Processing Area . . . 25
2.1.2.4 Interfaces Area . . . 26
2.2 HEIDENHAIN CONTROLLERS SCOPE . . . 27
2.2.1 Working Principle . . . 27
2.2.2 Auxiliary Technologies . . . 31
2.2.2.1 Programming Station . . . 31
2.2.2.2 TNCscope . . . 32
2.2.2.3 TNCremo . . . 32
3 HEIDENHAIN DATA ACQUISITION IMPLEMENTATION . . . 33
3.1 REQUIREMENTS . . . 33
3.2 SOLUTION MODELING . . . 35
3.3 DEVELOPMENT . . . 36
3.3.1 Main Thread . . . 36
3.3.1.1 Identify Controller . . . 37
3.3.1.2 Identify DGServer . . . 39
3.3.1.3 Start Acquisition Threads . . . 40
3.3.1.4 Monitor Acquisition Threads . . . 41
3.3.1.5 Reset DAQ . . . 41
3.3.2 Low Frequency Streamer Thread . . . 42
3.3.2.1 Stream Data . . . 44
3.3.3 High Frequency Acquisition Thread . . . 46
3.3.3.1 Select Channels From Controller . . . 47
3.3.3.2 Receive Data From Controller . . . 49
3.3.4 High Frequency Dispatch Thread . . . 49
3.3.4.1 Dispatch Acquired Data . . . 51
4 TESTS AND RESULT ANALYSIS . . . 52

4.1 DATA ACQUISITION FUNCTIONALITY 52
4.2 COMPARISON TO SIEMENS DATA ACQUISITION 58
4.3 DATA GROUND SERVICES PERFORMANCE 59
5 SERVICE OPTIMIZATIONS 62
5.1 DATA GROUND IN BATCHES 62
5.1.1 Implementation to Data Ground Services 62
5.1.2 Optimization Tests and Results 65
5.2 BALANCED CLOSING OF BATCHER FILES 70
6 CONCLUSION AND FUTURE WORK 72
6.1 FUTURE WORK 73
6.2 PERSONAL OBSERVATIONS 73
References 75
APPENDIX A – ADDITIONAL RESULTS FOR BATCH DATA OPTIMIZATION 79
A.1 FUNCTIONALITY COMPARISON IN THE FRONT-END 79
A.2 OPTIMIZED SERVICE PERFORMANCE 79

1 INTRODUCTION

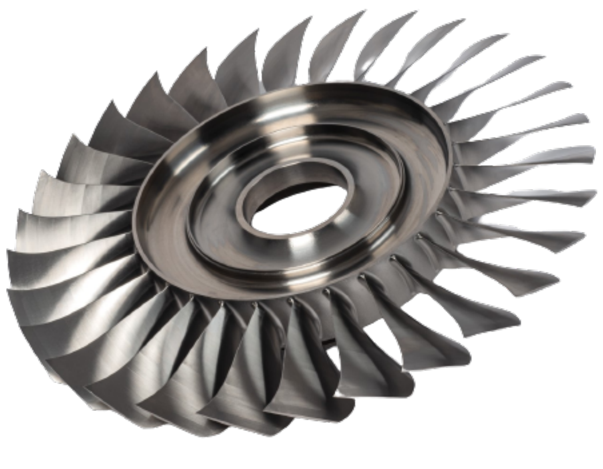
Over time, the demands of large-scale industrial production of parts and tools have become more and more stringent. To meet these requirements, new different Industry 4.0 technologies are being developed day by day. These technological improvements are taken into consideration for the development of this project, which aims to assist companies in achieving higher quality workpieces and more efficient production lines.

This first chapter presents the scope of the work developed. Firstly, the motivation is introduced in Section 1.1, where the general knowledge about the problem and the current plausible solutions addressed are mentioned. The company where the work was carried out and the associated project are then described in Section 1.2, along with the specific objectives of the work in question in Section 1.3. Finally, the structure of the document is discussed in Section 1.4, situating the reader on the content of each chapter of this report.

1.1 MOTIVATION

The demand for high quality, sustainable workpieces is increasing for many different types of industries around the world. For the turbomachinery and aircraft industry, a key technology part that has been used in recent years is the Blisk - see Figure 1 - a turbine component designed for higher fuel efficiency and reduced CO_2 emissions [1]. These elements with better aerodynamics offer lower weight and drag, being 8% more efficient when compared to conventional blade and disk assembly [2].

Figure 1 – Blisk.



Source: Fraunhofer IPT.

However, new demanding requirements, such as improved efficiency, tend to lead to more complicated part designs. Consequently, to achieve large-scale quality production of these complex workpieces, manufacturers in the metal industry have been driven to widely employ CNCs, since 1989 [3], in the machining processes. Despite this fact, due to various mechanical factors related to the machining itself, it is still difficult to get the desired end products to the required specifications. Blisks, in particular, still represent a machining challenge in an economic and technological way [4]. These complex thin-walled components with tight tolerances take days to be produced in strong and expensive alloys such as titanium [2] [4]. Moreover, their requirements are crucial for safety reasons, since the failure of blisks has already led to serious accidents. One example was the Pensacola aircraft crash in 1996, when a tiny quality deficiency prevented the cooling fluid from having the proper effect [5].

In addition to the challenging new specifications for part manufacturing, the monitoring and inspection of the machining processes and the finished workpieces is also an arduous task that few companies are able to perform [5]. To ensure the high quality of these complex components in the end, time and money are invested in conventional measuring methods such as optical and tactile metrology. Depending on the industry, these procedures can take up to 25% of production costs and 20% of lead times [6], besides the possibility of distorting or marking the part [2]. In the current case of blisks, this uncertainty of piece quality during the machining process generates about 25% rework at the end, since the result can only be seen when the part is definitely ready [5].

Nowadays, a trendy suitable way to solve these multiple issues is to implement a data acquisition software to process-parallel monitor machine tool operations [5]. Therefore, conventional measuring methods are replaced by workpiece digitization, which can bring more detail into process at a much faster rate, leading to a more reliable, faster, and accurate process control [2]. As an example, errors and quality defects can be detected at an early stage of production, preventing the reworking of parts and enabling instantaneous optimizations even before milling is finished. This improved quality prediction and assurance also has an economic and sustainable advantage, as resources invested in less efficient machining and metrology are reduced [6]. The digitization also impacts in the final blisk aspect, and even the slightest improvement in quality means safer components, better fuel efficiency and, in turn, lower CO_2 emissions. It has been estimated, for example, that for a 2% improvement in the quality of blisks, 16 million metric tons of CO_2 could be saved per year [5].

Bearing this in mind, gemineers GmbH proposes a similar approach for efficient workpiece measurements and quality assurance. The company's main idea is to develop a software capable of acquiring machine tool data for further development of a digital twin (a virtual representation that serves as the real-time digital counterpart of the finished component). This digitized part will be then virtually analyzed and compared with the Computer Aided Design (CAD) model of the element, providing a precise evaluation of the process and the final piece [6].

1.2 GEMINEERS GMBH

Envisioned in April 2019 and officially founded in July 2021, the gemineers GmbH - logo displayed in Figure 2 - is a spin-off startup associated with the High Performance Cutting department of Fraunhofer Institute for Production Technology (IPT). This subdivision of the Process Technology area is known for solutions and technologies for machining operations along the entire process chain, mainly in simultaneous multi-axis milling of thin-walled components for turbomachinery and aircraft applications [7] [8].

Figure 2 – gemineers logo.

Source: [9].

Due to a great motivation of the department to develop better blisks for the market, the main company's objective is to build a software that goes all the way into production, from machining data acquisition to information processing and extraction. The application, in this way, provides the user with easy access to previously undiscovered manufacturing data [6]. Hence, benefits that could be gained by the proposed solution are the reduction of conventional inspection efforts and the costs of post machining operations, time savings through targeted optimization, simple quality assessment, and a complete digital documentation of processes and products based on digital twins [9].

To achieve the goal of a functional and useful digital twin for the user, the technology of gemineers is divided into three different core elements [9]:

- **Machine and sensor data acquisition** for all common machine controls and sensor systems;
- **Fully automated Digital Twinnig** using high performance technology models;
- **User-friendly quality dashboard** browser-based.

This thesis is related to the first mentioned area, where a data acquisition system for CNCs is needed. The information acquired from the machine tool will be used in the entire product's process chain, as this is the basis of the digital twins presented to the user at the end.

1.3 OBJECTIVES

The main objective of this project is to develop a data acquisition software to process-parallel monitor machine tool operations. The application must be able to acquire low and high frequency information from the controllers to ensure that all details of a process are obtained. These data will further serve as an input to other services of the company that must handle storage and processing.

Considering the CNC producers worldwide, it is notable that Heidenhain controllers are one of the leading and most important when it comes to machining [10]. These CNCs reach a large part of the current global market and, according to gemineers' experience, tend to be challenging in the matter of data acquisition. Consequently, an application capable of safely and accurately acquiring data for Heidenhain controllers is of extreme interest to the company and is the ultimate goal of the work presented.

Based on this stipulated objective, specific tasks have been defined in order to achieve a functional software that is well suited to the gemineers' project as whole:

- Gather the application requirements and build its workflow;
- Define data types and their respective frequency to be acquired from the machine tool;
- Research the possibility of sending high frequency messages to a web server;
- Search libraries and auxiliary technologies to access Heidenhain controllers;
- Implement a secure connection between the application and the CNC;
- Determine and implement a mechanism to access controller areas that store data;
- Acquire and handle high and low frequency data from the machine tool;
- Connect and dispatch the obtained data to a web server;
- Perform functionality and performance testing for the data acquisition software and related applications using machine tool simulations;
- Propose and develop optimizations for these services.

1.4 DOCUMENT STRUCTURE

This document is divided into six chapters to further explain the development of the proposed solution. In Chapter 2, the state of the art is introduced, where the basic concepts and subjects are shown for a complete understanding of the subsequent activities. The company's microservices structure, technologies used and communication protocols necessary for the final application are explained in depth.

The implementation activities of the data acquisition software for Heidenhain controllers are discussed in Chapter 3. The requirements established for the desired application, the modeling of the solution, and the development process for each expected service step are explained in more detail.

Chapter 4 presents testing and analysis of results for services associated with the data acquisition area. The tests performed for software functionality and performance are shown along with their respective overall results obtained.

From information acquired within the testing phase, service optimizations are proposed in Chapter 5 for the data acquisition area. In addition, each reported optimization is here implemented and tested once more. As soon as the new results are achieved, a comparison is performed between the two software versions.

Finally, in Chapter 6 the conclusion takes place, where the finished work is summarized, as well as ideas for next steps related to the scope of this thesis are exposed.

2 STATE OF THE ART

This chapter is intended to further explain the scope in which this data acquisition software falls. Firstly, Section 2.1 extensively discusses gemineers' current product, presenting in detail the internal composition of the entire application. Subsequently, in Section 2.2, the operation of Heindenhain controllers are described together with possible technologies associated with them.

2.1 GEMINEERS PRODUCT

Aiming to assist and improve production lines, gemineers develops a software capable of acquiring, analyzing, and evaluating data from manufacturing processes. In order to achieve these features, a digital twin is created by the final program for each machined component. The Subsection 2.1.1 further explains how this concept works within and outside the company's scope.

Furthermore, since gemineers product is a complex software that handles multiple different functionalities at the same time, a microservices architecture has been implemented. Each of these framework applications is responsible for a certain step of the project as a whole. The connections between these services are further explored in Subsection 2.1.2.

2.1.1 Digital Twin

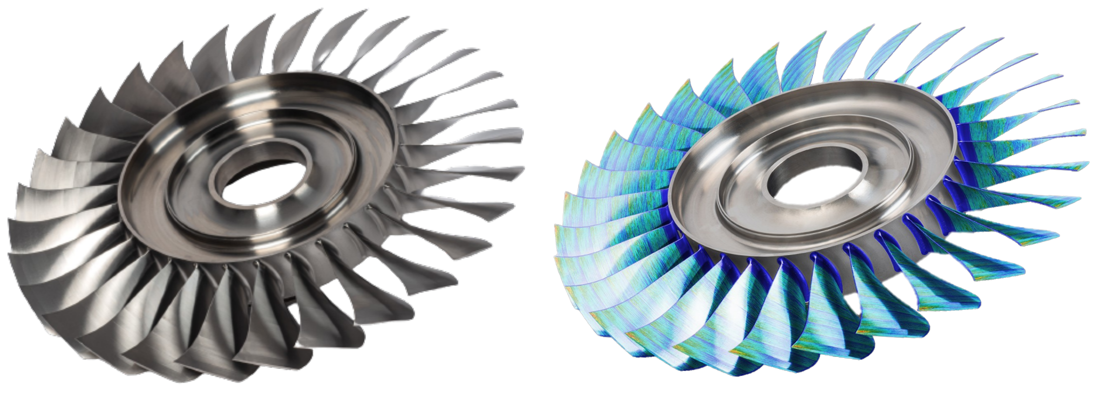
First introduced into manufacturing in 2002 by Dr. Michael Grieves, a digital twin is a digitized representation of a physical object, process, or service. The development of this concept consists of extensive real-time data acquisition from sensors attached to the original workpiece and the application of mathematical models to replicate and simulate an element from the real world in the virtual environment. Once developed, digital twins are commonly used to run simulations and tests in place of the original object to study and predict the properties and performance of the part before the processes associated with it are even initiated [11] [12].

Being an up-to-date and accurate copy of a physical counterpart, digital twins present numerous additional benefits to large-scale manufacturing. Some of them are related to monitoring and analyzing products and processes during the production line, leading to early detection of flaws, instant insights for improvements, and better final quality [11] [12]. The studied concept has also proven useful for improving efficiency in manufacturing, such as with IBM and Siemens' Service Lifecycle Management (SLM) project [13] that minimizes downtime in production [14].

Due to their countless benefits and with today's technological advances, digital twins have been a key development for automation in Industry 4.0. Their common applications can range from automotive and aircraft industries to healthcare services and urban planning. Furthermore, according to International Business Machines (IBM), the future of this concept is nearly limitless, due to the fact that increasing amounts of cognitive power are constantly being devoted to their use [11]. It is estimated that by 2027, the digital twin market will be valued at USD 73.5 billion [15].

Following this same path, gemineers and Fraunhofer IPT aim to implement an application for production lines based heavily on the concept of digital twins. From this technology, the company pursue to monitor machining processes on 5-axis machines in order to achieve more efficient and less expensive production and higher quality and safer products [6]. An example of the desired future implementation of digital twins by gemineers is shown in Figure 3, in which a blisk is again used as a sample. The colored lines on each blade essentially represent the data acquired in each machine cutting point.

Figure 3 – Digital twin example for a blisk.

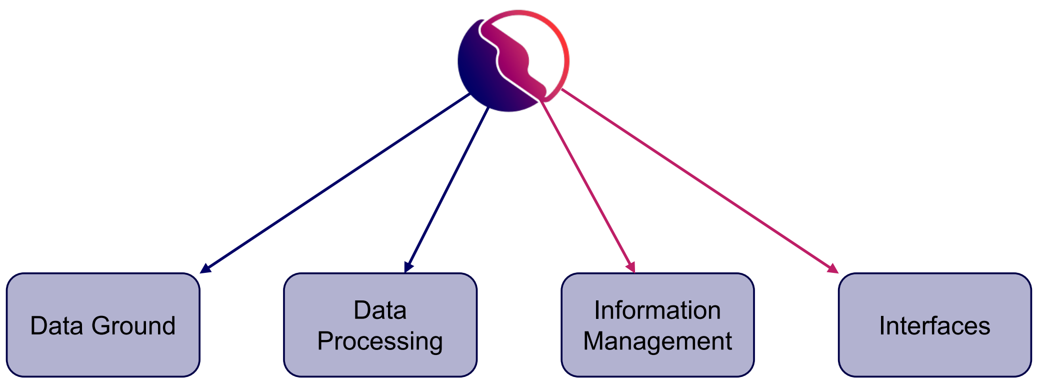


Source: Fraunhofer IPT and gemineers GmbH.

2.1.2 Microservices Architecture

A microservices architecture defines a framework for an application that brings together a set of smaller services. Within this structure, each microservice is a minor software built to handle specific features and tasks from the whole application that can communicate with each other through simple interfaces [16] [17]. The gemineers' architecture was designed to separate the developed services based on their tasks into four different operational areas: Data Ground, Data Processing, Information Management, and Interfaces. Figure 4 illustrates the proposed subdivision.

Figure 4 – gemineers product areas.



Source: Author.

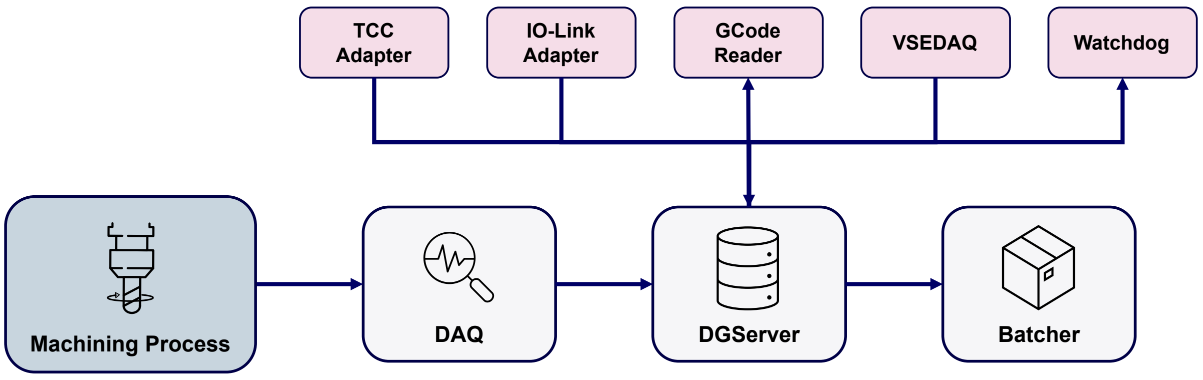
An in-depth explanation of each area and how they are correlated is shown in the following subsections, focusing on the Data Ground, where the Data Acquisition software are placed.

2.1.2.1 Data Ground Area

The Data Ground is the first area of the entire gemineers' software to be triggered when a machining operation is taking place on the machine tool. The Data Acquisition (DAQ) software in here developed are the main responsible for starting the whole process chain that will be followed by the next areas of the application.

The objective of this area is to connect to operating machine tools to acquire necessary data, to standardize this information and to send a raw data batch to the next services. This procedure is heavily built on the Data Acquisition software, the Data Ground Server (commonly called DGServer in the company's scope), and the Batcher, which together handle most of the required processes of this stage. The secondary applications within the Data Ground are not fully needed to the data workflow to function, even though they do provide additional features for the mentioned area. Figure 5 presents the structure of the Data Ground services and how they communicate to each other.

Figure 5 – Structure of the Data Ground area.



Source: Author.

The five smaller components connected to the main server at the top of the image represent the supplementary applications that will not be covered in this report. DAQ, DGServer, and Batcher, in the center, are the primary services in this area and will be explained in sequence for a complete understanding of the data flow. The leftmost block characterizes a machining operation on the machine tool from which the data will be acquired.

Data Acquisition Software

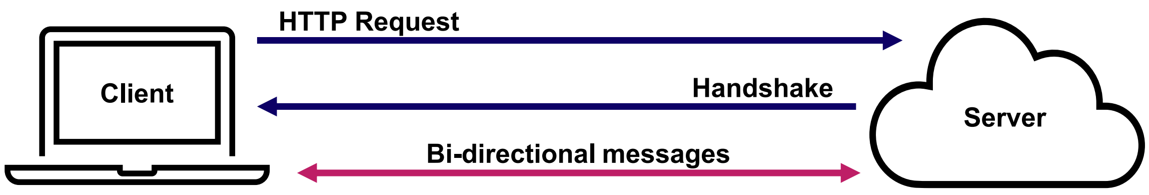
DAQ applications are the first step in getting the gemineers product started, as they are directly connected to machine tools that are performing 5-axis machining processes. Once the hardware and software are coupled, these high-performance services are able to acquire the desired information from the machine in both low and high frequency manner for further dispatching to the Data Ground Server over the network. At the current state of the company, there is only one Data Acquisition service running, and it is designed for Siemens controllers.

This existing DAQ follows the framework just mentioned. Its workflow begins by establishing a connection to the operating machine tool. Once connected, the software specifies the controller variables that store the desired data type (such as for example, the coordinates) and correctly calls the functions that are able to read this information. These methods are, in this Siemens case, available based on specific libraries designed for these controls.

Each single value acquired by this program is standardized in programming objects and translated into a message to be sent to the DGServer via the network. The selected network communication protocol allows developers to build faster real-time applications for sending message-based data. It starts in the same way as an Hypertext Transfer Protocol (HTTP), based on a request of a resource. When acknowledged by the server, it returns a handshake signal that keeps the Transmission Control Protocol

(TCP) connections alive even after the response is received. This behavior provides a full-duplex communication channel that permits messages to be passed back and forth (bi-directional) between the client and the server. Figure 6 represents the working principle just mentioned.

Figure 6 – Working principle of the selected communication protocol.



Source: Author.

This protocol was selected for the data acquisition scope especially because of the high frequency data acquired and dispatched to the DGServer. With the TCP connections still available, the DAQ can stream plenty of information without exceeding the network load due to multiple requests necessary for other communication protocols, such as the HTTP, for example.

Data Ground Server

As the core service of the Data Ground, the DGServer's job is to receive data coming from the DAQ and redirect it to the interested services, maintaining a common and structured interface for different types of data. The server is developed on top of a query language that turns possible to define the exact data desired, replacing numerous requests from any other API with a single call.

The language is built on three main operations: **queries**, **mutations**, and **subscriptions** that interact with a **schema**, a predefined data structure that defines types, and a list of associated fields.

Bearing this in mind, it is time to better comprehend the workflow from the gemineers' service that hosts this query language. The Data Ground Server starts its execution when connections to Data Acquisition are established. When coupled, the DAQ is able to transmit real-time data that simulates mutations to the server. Thus, for each message, mutation behaviors are performed, updating the schema with the most recent obtained values. As soon as data are modified within the DGServer, the publishers notify and trigger the subscribed services and transfer the acquired data to them. This process is repeated over and over again in a millisecond range.

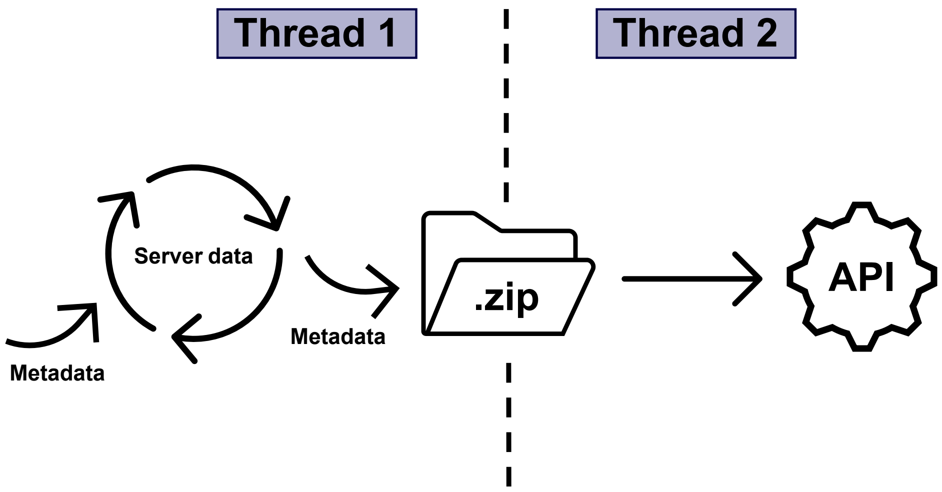
Currently, the Data Ground Server only works with one data point at a time for each type, which means that when new values arrive for the same type, the old information is forgotten within the terms of the server.

Batcher

The Batcher, also known as the Batching Service, is responsible for organizing and agglomerating data that are passing through the server in batches. This information is obtained based on queries and subscription calls to the Data Ground Server that transfer its temporarily stored data to the service in question. For each type, an individual file is created that stores all data acquired during a machining operation. In the end, these files are placed together in a zip folder, the so called batch, to be further sent to the next areas of the entire application.

A normal Batcher run begins by creating two complementary threads designated for different purposes. The first one connects to the Data Ground Server via the selected protocol mentioned before and acquires all its transient data to create a batch of information. The second is responsible for establishing connections over HTTP with the gemineers API (better explained in Subsubsection 2.1.2.4) to send the batches created to the next services. A simplified illustration of how the threads work and are correlated is presented in Figure 7.

Figure 7 – Threads from Batcher.



Source: Author.

When a query or subscription retrieves information from the metadata type, a flag is set inside the Batcher to start a new batch. If the program has just been started, the acquisition of data begins sequentially, based on returned messages coming from queries and subscriptions. If an existing batch is already running in the service, this older batch is closed and a new one is instantiated to continue the acquisition. A metadata is directly associated with an operation running on the machine tool, which means that when this information is read, a new operation has started and, consequently, a new batch should also be.

In the meantime of the reading of the two metadata signals, all other data of various types are aggregated and appended to the computer's local buffer, being associated in this way to the last metadata/operation acquired. Once the batch has to be closed, the buffered data are transformed into files and then zipped together.

The second mentioned thread operates once the batching is done and the zip folder is well prepared. A request to the gemineers' API is sent, attaching the batch of raw data to the message. With the data sent to the next areas of the application, the entire workflow of the Data Ground area is finished.

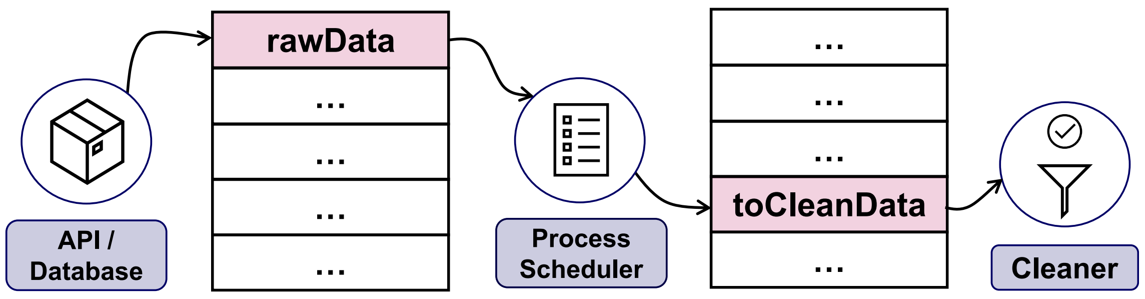
2.1.2.2 Information Management Area

Consisting basically of a software called Process Scheduler, the Information Management area is mainly responsible for being the brain of the whole application when it comes to communication between services. It mainly benefits from two important technologies: the gemineers' database; and a queue-structured message broker, which allows multiple information to be rerouted to different services.

The Process Scheduler works on scheduling tasks for different software based on inputs and outputs coming from other applications across the process chain. This service receives an output data posted to a specific queue, and redirects it to the correct queue that is consumed as input by the next application, creating a continuous data flow among services. Still using the output information, the Process Scheduler is able to update the data objects within the database, providing always up-to-date instances of the "geminis", the way digital twins are called in the company's scope.

For a better understanding, the Figure 8 presents an example with the raw data outputted by the Data Ground area. The API sends the data-based message containing the zip folder to the broker's *rawData* queue, which is then redirected to the *toCleanData* by the Process Scheduler. The Cleaner, a service from the Data Processing area, then consumes the latter queue and uses the resulting data as an input for its activities.

Figure 8 – Process Scheduler working principle.



Source: Author.

2.1.2.3 Data Processing Area

To produce meaningful information from the raw data acquired, a series of applied mathematical calculations and computations must be performed by the Data Processing area. The main goal of the services associated here is to transform all the data from the machining process into understandable information for the end user of the software.

The Cleaner is the primary application in this area and is responsible for removing useless data and outliers, a data point that deviates significantly from similar information in a sample [18], thus creating cleaned data. This service and all others in the Data Processing area work in sequence, one building on the output of the other. Consequently, with each execution of each software, more relevant calculated information are added to the geminies that will be presented at the end.

The basic operational process of getting and sending data to other services is common to all applications within this area. As seen in Figure 8 with the Cleaner example, they generally receive their inputs based on a queued consumption. Once the data arrives, the activities of each software are triggered to execute and the output is sent back to the message broker. This procedure is done until every message is consumed from the queue. The Information Management area is then responsible for handling the information received and updating the database with the latest outputs from these services, as previously shown.

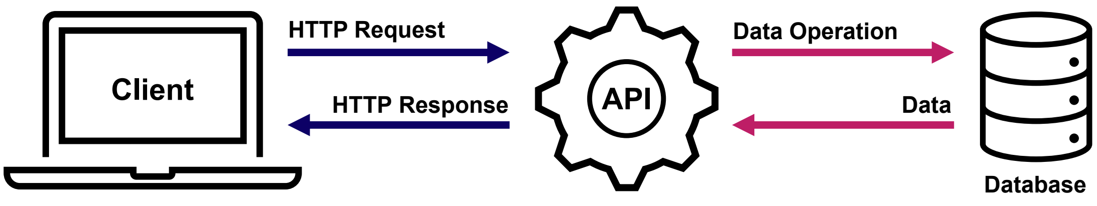
As soon as the data flow has finished its passage through the Data Processing applications, the acquired and calculated machining operation information can be viewed in the Front-End interface as a digital twin representation.

2.1.2.4 Interfaces Area

The Interfaces area is basically composed of two different services: the API and the Front-End. The gemineers API, previously mentioned, is designed to communicate different applications with each other and with the database. Being one of the most important points in a microservice architecture, this interface handles different operations for the data flow, from saving the output of the Data Ground to the data bank until sending information to the Front-End.

Generally, web APIs, such as the gemineers', are developed on top of HTTP. The service that wants to perform database operations needs to make a request to this interface which returns a success or failure response depending on whether the operation was performed or not. This procedure provides better security and standardization to larger software, since the database is not directly accessible by any other application. A visual illustration of the working principle and communication of an API is summarized in Figure 9.

Figure 9 – APIs working principle.



Source: Author.

The Front-End is the final service of the whole application, since it is the one that interacts with the user, providing different resources directly accessed through simple clicks in a web interface. The Front-End of gemineers is able to visually display, in the form of digital twins, all the data that was acquired by the Data Ground and processed by the Data Processing area, allowing the user to easily visualize different information from the machining operations. The processes that are to be analyzed, must be selected by an operator in the online platform. Each of these desired processes represents a call from the Front-End to the API, since the Front-End does not have direct access to the database. This service then makes an HTTP request to the middleware interface, in the same way as presented in Figure 9, which responds successfully if the process has been found, returning the necessary stored data.

Once the digital twins are displayed, the operators are able to define whether or not a machined part has the desired quality, just by visualizing and comparing the results presented with the developed CAD model. This procedure defines the final step of the data flow and one of the last features to which the entire gemineers application was designed. From there, based on the digital twins provided, the end user can study each operation and workpiece to define the next steps in the overall manufacturing production.

2.2 HEIDENHAIN CONTROLLERS SCOPE

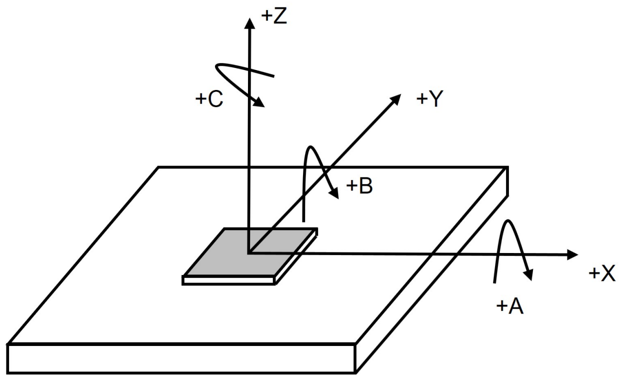
To develop a software that can acquire data from Heidenhain controllers, it is first necessary to better understand their operational basis and how the desired information can be accessed by an external application. The Subsection 2.2.1 discusses the working principle of these controllers for 5-axis machining operations in CNCs.

Additionally, to be able to acquire the desired data and to test the implemented application, a number of different Heidenhain-related auxiliary software and development libraries are of utmost importance. The technologies used in this project are further explored in Subsection 2.2.2.

2.2.1 Working Principle

A Computer Numerical Control is essentially a digitized computer capable of automating and controlling an industrial machine. The machine tools studied for this project are built on five axes, three linear (X, Y, and Z) and two rotational (ranging from A, B, and C) that together can perform complex machining operations. Typically, X and Y axes are used to move the worktable where the piece is positioned, while the Z axis is responsible for the cutting tool vertical movements [19]. The A, B, and C axes specify angular positions about the X, Y, and Z axes, respectively - see Figure 10. The two main coordinate systems used by CNCs are: Workpiece Coordinate System (WCS), where the workpiece is used as the reference for the Cartesian coordinates; and Machine Coordinate System (MCS), when the machine is used as such reference. The most important operation for this project is called CNC machining, a manufacturing process performed by a machine tool and controls to remove material from a workpiece to produce the desired part [20] [21].

Figure 10 – Usual coordinate system for a CNC.



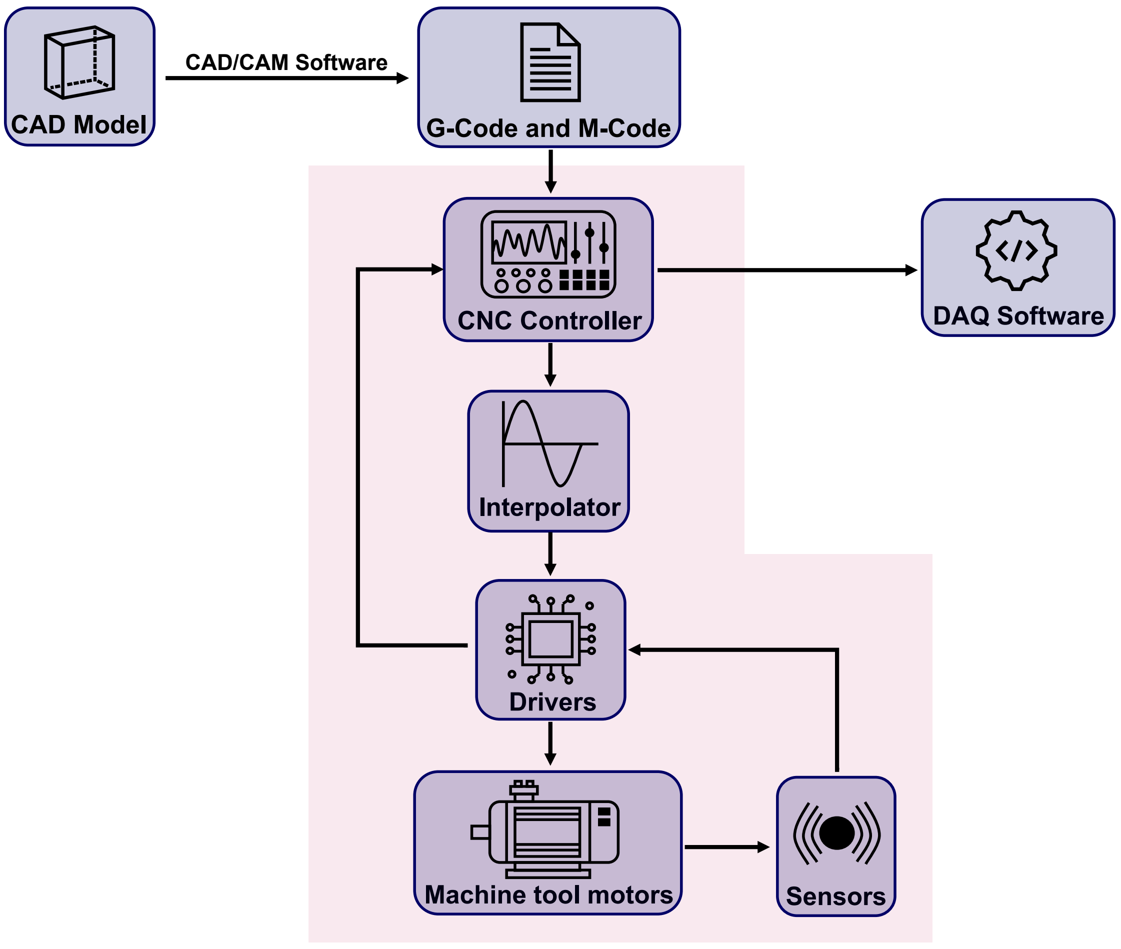
Source: Author.

Digital files containing detailed instructions must be provided to allow the controller to perform the operations required to machine a part properly [22]. Therefore, the CNC machining process begins with the design of parts in a 3D computer model, which determines dimensions and properties of the final workpiece. These CAD models then go through Computer Aided Manufacturing (CAM) software to extract their geometry and to create CNC programs [20] [21] [22].

These machine tool-understandable programming codes are, in essence, information that is read by the controller to perform a physical operation [23]. They are divided into two different types: Geometric Code (G-Code), which defines the speed and movements of the cutting head, based on a set of coordinates; and Miscellaneous Code (M-Code), that covers additional information, for example coolant data and tool changes [22]. Once these files have been generated, they are loaded into the CNC machine [20]. The last step before performing a machining operation is to set the correct parameters for the process, such as the cutting tool, spindle speed and feed rate (further explained in the next chapter) [22]. When each mentioned stage is finished, the programs can run in the CNC and, consequently, start the process on the machine tool.

The controller handles the execution of files in a complex manner and in a specific sequence of steps to ensure the stability of the machine tool when in operation. In addition to the main CNC controller, they are composed by interpreter and interpolator modules, and by a series of drivers that are directly connected to the motors of a machine tool [24]. Figure 11 summarizes the step-by-step process of how a CNC controller works, since the CAD creation until data retrieval to the DAQ software. The hatched part highlights the actual control unit structure and division.

Figure 11 – CNC architecture.

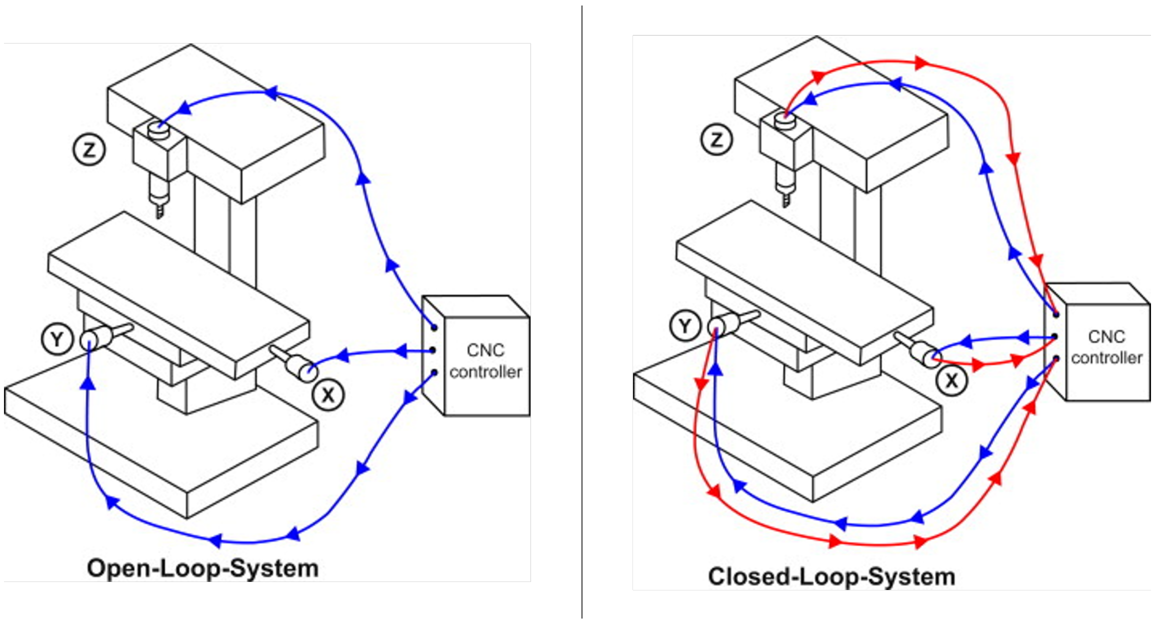


Source: Author.

To produce the stipulated motion in the physical machine, the CNC files must first be interpreted by the controller and translated into understandable signals for the other components of the architecture. Since these signals are primarily composed of a set of coordinate values that take the cutting tool from point A to B, for example, the interpolator module is responsible for interpolating data points between these two specified references from the G-Code file, producing a smooth and synchronized tool path [19] [24]. Considering Heidenhain controllers, this interpolation process occurs within a three millisecond interval. Once the data are well converted into interpolated signals, they are sent to the Drivers, also known as Servo Controllers. These elements are able to transfer the acquired signal to produce torque and activate the machine tool motors, since they are directly connected. The motors, in turn, based on the impulse forwarded by the Drivers, move the axes and change the tool speed, performing, in fact, the movements predefined by the G-Code file in a sophisticated way.

The internal process continues depending on how the architecture is structured, based on Open-Loop or Closed-Loop systems [23] [25]. Open-Loops do not feature any feedback subsystem to verify if the desired positions and velocity have been achieved, and have no correction if the results deviate from what is expected [25]. Closed-Loops, on the other hand, are much more complex since they implement a feedback subsystem based on sensor measurements that returns the actual values of the machine tool axes to the Drivers. When these signals are transferred back, the controller corrects any potential error, ensuring its accuracy [25]. Both frameworks are better presented in Figure 12, where a visual comparison takes place.

Figure 12 – Open- and Closed-Loop systems for CNCs.



Source: [25].

Heidenhain controllers are based on the Closed-Loop, as they consist in complex CNCs that are capable of performing high-precision operations with a machine tool. The example previously shown in Figure 11 explores a simple implementation of this process, where measurements are constantly taken by sensors to check the positions and speed of the axes. The acquired information is returned directly to the Servo Controllers that transfer the signal back to the main CNC controller's run-time memory for complete feedback on the executed operation. The idea of a data acquisition software is to access these specific controller memory locations to obtain the respective cached data for each machine parameter of interest.

Moreover, high frequency data, such as axis positions, are particularly stored in a "Scope Information" section of the Heindenhain controllers' memory. In order to access this reserved area of the CNC with a DAQ, auxiliary software and libraries are needed. This procedure adds a new degree of complexity for the desired gemineers' application that will be handled with the auxiliary technologies presented in the following.

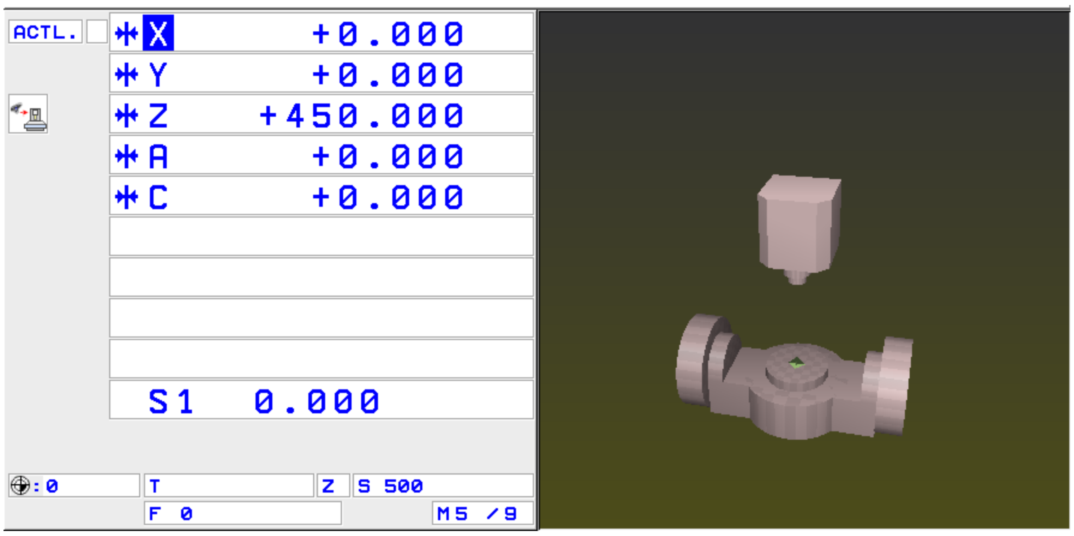
2.2.2 Auxiliary Technologies

To assist in the development of a Data Acquisition application for Heidenhain controllers, a number of auxiliary tools are needed to handle multiple different operations. In the following subsections, each Heidenhain software used is presented, such as Programming Stations, TNCscope, and TNCremo.

2.2.2.1 Programming Station

Heidenhain Programming Stations are software that are capable to completely emulate a CNC controller. Since these applications are based on the same software as normal controls, they are able to create, test and optimize programs away from a suitable machine [26]. Their interfaces simulate an Human Machine Interface (HMI) of a machine tool that also features simulation of machining processes from the workpiece's point of view and the machine's kinematics (the way the axes are shown and move in space). Figure 13 illustrates an example of such application, where the coordinate positions (left side) and the kinematics (right side) are presented.

Figure 13 – Programming Station Software of a Heidenhain iTNC 530 controller.



Source: Author.

The Programming Stations even simulate the connections and communications of a real controller, which means that they are also accessible from external services in the same way. Therefore, while machining operations are being simulated on them, data can be acquired by an attached DAQ software. The Programming Station used for this project is for the iTNC 530 controller emulation since it is a simpler version to work with. Nevertheless, there are several of these applications available for different Heidenhain models.

2.2.2.2 TNCscope

Related to the scope information previously mentioned, the TNCscope is a Heidenhain software capable of recording and monitoring scope data from the controller's memory. In this application's interface, the desired machine variables are listed and selected to have their values graphically displayed in parallel with the execution of a machining operation.

This application is used within the company to check if the Scope variables (high frequency data) read by the developed Data Acquisition are the same as those displayed in the official Heidenhain software, as well as for a later comparison of both values. When a new machine parameter needs to be acquired, a normal procedure is to verify its data in the TNCscope interface and only then start the implementation for the gemineers' service.

2.2.2.3 TNCremo

TNCremo is a Heidenhain software that easily enables users to transfer data files between a local computer and a real CNC controller when connected by remote access, or a Programming Station. This bidirectional transference can occur via Ethernet or serial interfaces [27].

Within gemineers, TNCremo is used extensively to exchange part programs and tool tables that are usually sent from a normal computer to the Programming Station. These G-Code files are, in essence, simple machining programs that are simulated on the emulated controller to test whether the developed DAQ is working as intended. Tool tables are transferred to update the simulation software with additional kinematics configuration not available in the original version.

Putting these three applications together with further libraries for code development makes it possible to implement the Heidenhain DAQ software, which is better discussed in Chapter 3.

3 HEIDENHAIN DATA ACQUISITION IMPLEMENTATION

Considering the gemineers product and the auxiliary technologies for working with CNCs, this chapter aims to present the implementation of a software for data acquisition of machining operations from Heidenhain controllers. The Section 3.1 further discusses the requirements of the desired application that will be translated into a solution model described in Section 3.2. The actual code development for this service is shown first in Section 3.3 and in its respective subsections. Each part of the latter presents different programmed threads for a clearer understanding of the project.

3.1 REQUIREMENTS

In order to fully work as expected by the company, the Heidenhain DAQ needs to meet key requirements regarding its overall execution. To maintain a standard, this Data Acquisition software should run similarly to the existing Siemens DAQ (previously mentioned in Chapter 2), performing related tasks but taking into account the uniqueness of Heidenhain controllers. When the program starts, it must be able to perform connections to an operating machine tool and to the Data Ground Server based on Internet Protocol (IP) addresses provided by the user. These first identifications and coupling with the hardware and the DGServer are necessary to prepare the service to carry on a clean data stream.

There are several types of data considered important to the company in a machining process that are constantly updated in the controller. Such information can refer to the mechanical operation itself or to the current state of the machine tool performing the work. gemineers separate these data into high and low frequency, depending on how often it needs to be acquired by the software. Mechanical information is primarily considered high frequency (and must be obtained as fast as possible, in a millisecond range), as each data point is of utmost importance for digitizing workpieces into digital twins. Additional information and machine tool metadata is essentially acquired at low frequency (only read a couple of times per second), since it is used to organize processes and to correctly associate the corresponding mechanical data with the respective machining operation performed.

The desired Heidenhain DAQ must be able to access specific areas of the controller's memory to properly read all this stored information. Each major data type that must be handled by the service in question is better explained in the following, along with its respective acquisition frequency shown in Table 1.

- **Coordinates:** the precise position of the cutting tool, translated into five axes, three linear and two rotational;

- **Spindle and Axis Loads:** the load applied to each axis and spindle (the shaft at the center of the rotating axis) of the machine tool;
- **Spindle Speed:** the speed of the running spindle in the machine tool;
- **Feed Rate:** the velocity at which the cutting tool is advanced against the workpiece [28];
- **Presets:** the offset of each axis compared to the defined machine tool zero point;
- **Tool data:** the information concerning the cutting tool, such as radius, length and name;
- **Program data:** the G-Code information, including file names and contents, running on the machine tool;
- **Machine tool status:** the current state of the machine tool (stopped, running, etc.);
- **Machine tool execution mode:** the current mode that the machine tool is in (manual, automatic, etc.).

Table 1 – Data types and the associated frequency of acquisition.

Data type	Frequency of acquisition
Coordinates	High
Spindle and Axis Loads	High
Spindle Speed	High
Feed Rate	High
Presets	Low
Tool data	Low
Program data	Low
Machine tool status	Low
Machine tool execution mode	Low

Source: Author.

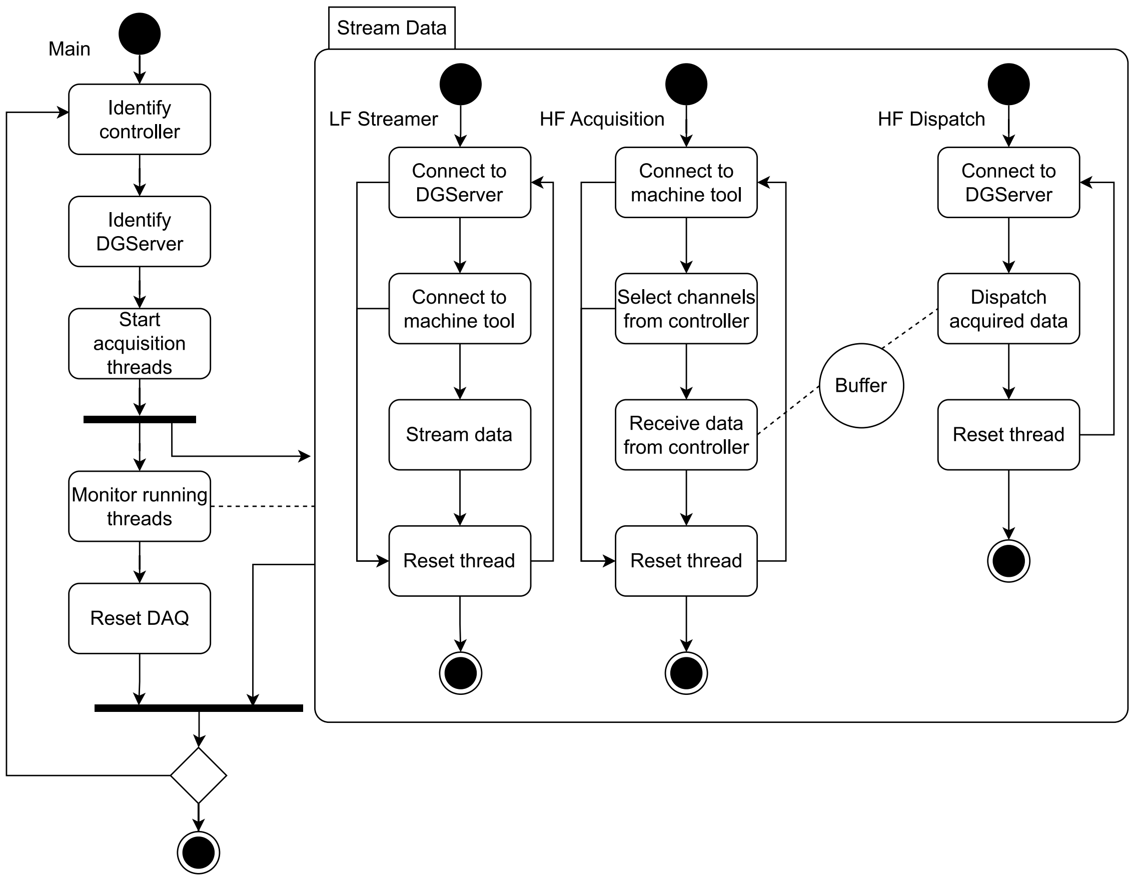
The acquired low frequency data should be used to build a metadata structure for each machine tool operation. This created information should function as a unique label to sort processes into future geminies, and a trigger for further applications of the entire gemineers' software.

Since these different types of data come in distinct forms from the controller output, they first need to be standardized into a common interface. Therefore, the Heidenhain DAQ needs to be able to transform each piece of acquired information into individual programming objects. These instances, after being systematized, must be translated into understandable messages and then dispatched to the server via the selected communication protocol. This procedure takes place in a matter of milliseconds, at the expense of the high frequency data. Once completed, the workflow of the desired application is finished.

3.2 SOLUTION MODELING

The development of a complex multi-threaded software is proposed as a solution to accomplish a Heidenhain DAQ with the requirements discussed in the previous section. The concept of multi-threading is applied in the given service to be able to perform different operations simultaneously within the code, such as in handling data types at distinct frequencies. The developed model for this Data Acquisition software is better presented in Figure 14.

Figure 14 – Overall Heidenhain DAQ operating model.



Source: Author.

The event columns, denoted by the initial circle in a Unified Modeling Language (UML) diagram, represent major threads of the application, totaling four: Main, LF Streamer, HF Acquisition, and HF Dispatch. Each thread, in turn, constitutes a state machine, where the operation steps of the different scenarios are written in blocks and connected by arrows in Figure 14. This DAQ service is designed in the form of multiple state machines because its operational steps are well-defined and distinct from each other, being also easier to develop and debug the code when tested.

A summary of the entire software workflow begins with an instance of the Main Thread that establishes connections to the machine tool and to the Data Ground Server, coupling them together as expected by the requirements shown. Once connected, it initializes the acquisition threads that handle the data stream within the given application. The LF Streamer is responsible for acquiring and directly sending the low frequency data to the server, while the high frequency is assigned to the HF Acquisition and the HF Dispatch to avoid overloading only one thread, since the amount of information is considerably large. As the names suggest, the former reads the scope data from the controller while the latter dispatches it to the server.

The data transfer between the two high frequency threads is based on a buffer. While the HF Acquisition writes newly acquired data to the buffer, the HF Dispatch simultaneously reads and consumes this information, ensuring synchronicity to the process.

At the end of the operation, each state machine features a Reset step that checks whether the data acquisition is complete or not. If the stop flag is triggered, all threads are finished and the program comes to an end. Otherwise, an error has occurred and the thread itself just restarts its execution from the first stage.

3.3 DEVELOPMENT

Considering the proposed solution model, the current section aims to introduce the entire development of the data acquisition software for Heidenhain controllers. The following subsections present the implementation of each thread already discussed, along with their respective states illustrated in Figure 14. Nevertheless, steps that repeat among the four threads will not be covered twice. To accomplish this explanation and still preserve company confidentiality, more detailed diagrams and pseudo-code snippets are further shown.

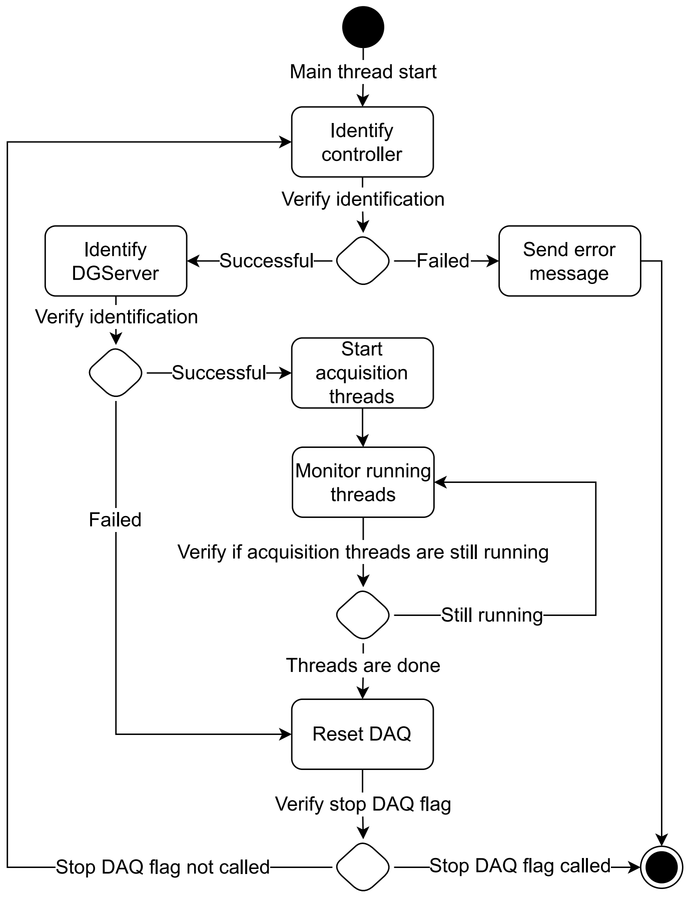
The first operation performed when the Heidenhain DAQ starts is reading a configuration file developed within its repository. This document contains all basic information for the software, which allows users to directly change elementary data, such as IP addresses without the need to dig into the source code. This information is then stored in variables that serve as a first input for starting the Main Thread.

3.3.1 Main Thread

Built based on five states (Identify Controller, Identify DGServer, Start Acquisition Threads, Monitor Running Threads, and Reset DAQ), the Main is the thread that performs the initial steps of the software itself, since it is the first one to be initialized.

Its stages are connected to each other according to Figure 15.

Figure 15 – Activity diagram of Main Thread.



Source: Author.

A successful execution of the workflow shown in Figure 15 starts with the identification of the CNC controller, where a connection is established to the machine tool. Similarly, a link is also made with the Data Ground Server, identifying which machine tool is being used. After the elementary recognitions are completed, the Main Thread serves as a facilitator for the other threads, as it initiates and monitors them during the entire data acquisition process. The Reset state, at the ending, is responsible for handling the finished threads and redirecting the program to an end or to the beginning once again. The code implementation for each step is further discussed below.

3.3.1.1 Identify Controller

To correctly perform the identification of Heidenhain controllers, additional libraries are indispensable for handling requests and the communication between an external service and the machine tool. The Code Snippet 1 exemplifies its use to properly obtain a connection and basic information from CNCs.

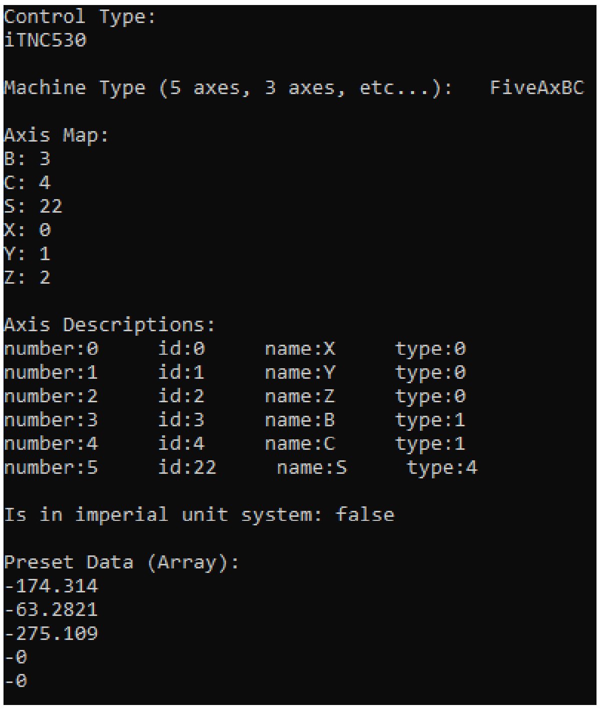
Code Snippet 1 – First Heidenhain controller identification.

```
if (identify_controller) {
    bool ok = ConnectToMachine();
    ok = AcquireBasicInfo();

    // Verify if functions were successful
    if (ok)
      GoToNextStep();
    else
      StopProgram();

    DisconnectToMachine();
}
```

To start this recognition procedure, the first step is to open a connection with the controller. The pseudo function *ConnectToMachine* executes a couple of standard methods that are needed to connect to the CNC and to access areas of its memory.

Once a connection is established, functions represented by the *AcquireBasicInfo* are called to acquire the controller's elementary data. The logic using *if (ok)* statements in Code Snippet 1 ensures that the next subroutine is only executed if the last one returns a success flag, as the boolean *ok* is updated after each method. To better exemplify this execution, Figure 16 shows a common output of the developed application.

Figure 16 – Output data of the Identify Controller step.



```
Control Type:
iTNC530

Machine Type (5 axes, 3 axes, etc...):   FiveAxBC

Axis Map:
B: 3
C: 4
S: 22
X: 0
Y: 1
Z: 2

Axis Descriptions:
number:0     id:0     name:X     type:0
number:1     id:1     name:Y     type:0
number:2     id:2     name:Z     type:0
number:3     id:3     name:B     type:1
number:4     id:4     name:C     type:1
number:5     id:22     name:S     type:4

Is in imperial unit system: false

Preset Data (Array):
-174.314
-63.2821
-275.109
-0
-0
```

Source: Author.

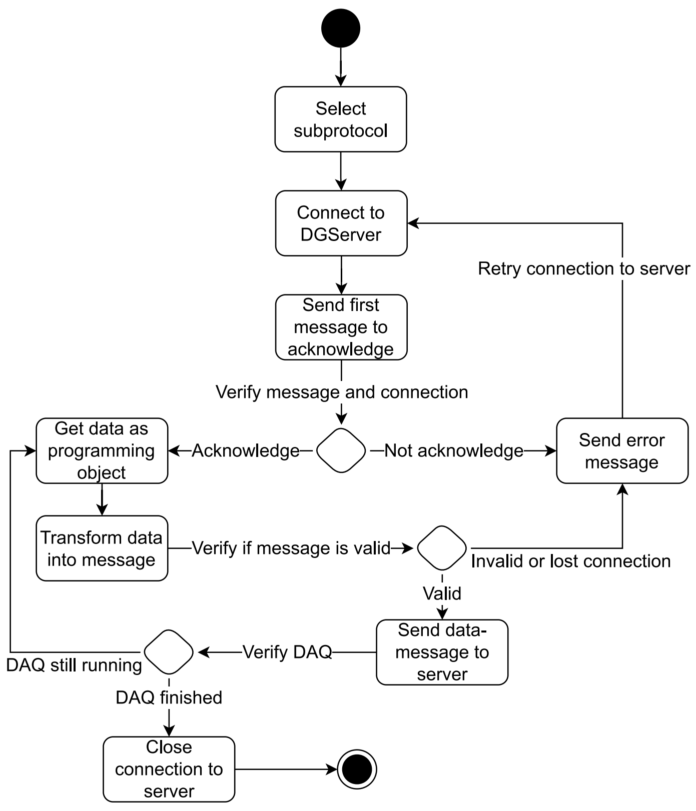
From top to bottom, Figure 16 first presents the control type, in this case an iTNC 530 controller, and the machine type, which consists of a 5-axis machine tool operating with the rotational axes B and C, already discussed in Chapter 2. Sequentially, the five axes are identified along with the spindle parameter (named S) and are properly associated with their respective internal metadata, such as ID and type. The unit system is further pointed out by a mock declaration for the imperial one. And finally, the current preset value for each coordinate axis (X, Y, Z, B, and C) is displayed.

At the end of the Identify Controller state, if the processes were successful, the next step is set to Identify DGServer. Otherwise, an error is thrown and the DAQ is terminated. When the entire workflow is over, the *DisconnectToMachine* pseudo function is called to avoid accumulating open connections with the CNC.

3.3.1.2 Identify DGServer

A connection between this DAQ and the server is required to send the data acquired to the next applications. Following the Siemens case, the Heidenhain one must also perform a series of operations to connect and dispatch information to the server through the network. Figure 17 further explores the steps of these processes that will be repeated several times during the execution of this Data Acquisition software.

Figure 17 – Standard procedure to connect and send data to DGServer.


Select subprotocol
Connect to DGServer
Retry connection to server
Send first message to acknowledge
Verify message and connection
Get data as programming object
Acknowledge
Not acknowledge
Send error message
Transform data into message
Verify if message is valid
Invalid or lost connection
Valid
Send data-message to server
Verify DAQ
DAQ still running
DAQ finished
Close connection to server

Source: Author.

In order for both the DAQ and the server to properly communicate with each other, a subprotocol must be selected that specifies how the data structure to be dispatched is constructed. Once these services are well predefined, an initial acknowledgment message is sent by the DAQ to the DGServer. Given a successful execution, this message is acknowledged by the server, which becomes able to receive new data from this software until a disconnect call is made.

Moreover, a couple of steps are necessary to actually transfer the acquired information to the DGServer. First, as previously mentioned, it needs to be standardized to a common interface, which in this case are programming objects. Once transformed, one dispatch function is created for each data type on the DAQ side in order to translate these data into valid and understandable messages for the server, and then send them out. If there are any problems with sending the information over a connection, a reconnect subroutine is immediately triggered. In essence, as soon as the execution of this acquisition application is complete, the multiple connections to the DGServer are closed. A simple example of such operations is implemented in Code Snippet 2, where the first identification of the server takes place in the Heidenhain DAQ.

Code Snippet 2 – First DGServer identification.

```
if (identify_dgserver) {
    int connection = ConnectToDGServer();

    // If the connection was successful
    if (connection >= 0) {
      DispatchBasicData();
      GoToNextStep();
    }
    else
      GoToReset();
}
```

The connection attempt is performed based on the pseudo function *ConnectToDGServer*. The behavior of this method depends on the used library and returns an integer, which specifies the number of the connection established. If the mentioned routine succeeds, the basic data are standardized into an object and dispatched directly to the DGServer, which is represented by the pseudo method *DispatchBasicData*. When both operations are completed, the state is set to Start Acquisition Threads and code execution proceeds. If any error occurs during the connect call, the next stage is set to Reset DAQ.

3.3.1.3 Start Acquisition Threads

Once the controller and the DGServer are properly identified, the acquisition threads (LF Streamer, HF Acquisition, and HF Dispatch) are initialized to run their own internal steps. Code Snippet 3 shows how the Main Thread instantiates the others, starting the data acquisition process itself.

Code Snippet 3 – Start of acquisition threads.

```
if (start_threads) {
    run_threads = true;
    stop_threads = false;

    InstantiateThreads();
    GoToNextStep();
}
```

To aid code development and to track the status of these threads (instantiated based on the *InstantiateThreads* pseudo function), the flags for running and stopping them are updated. After they are successfully executed, the new state is set to Monitor Acquisition Threads.

3.3.1.4 Monitor Acquisition Threads

The process of monitoring the acquisition threads is implemented based only on an *if* statement, as visualized in Code Snippet 4. The idea of this step is to constantly check if any of the running threads is finished or threw an error, changing, in fact, the value of the *run_threads* flag. If they are all running, nothing happens but a delay call to perform this same validation within 100 milliseconds. As soon as the bool variable switches, the state is set to Reset to evaluate whether the program is continued or not.

Code Snippet 4 – Monitoring of the acquisition threads.

```
if (monitor_threads) {
    if (run_threads)
      Sleep(100);
    else
      GoToReset();
}
```

3.3.1.5 Reset DAQ

The Main Thread's Reset state is the responsible for handling the program's behavior based on any error or end flags. As previously seen, it can be called from multiple steps in the code when the normal processes do not follow as expected. The basic operations performed by this stage are better presented in Code Snippet 5.

Code Snippet 5 – Reset DAQ procedure.

```
if (reset) {
    stop_threads = true;

    DisconnectToDGServer();
    JoinAllThreads();

    // Finish or restart the program based on a stop flag
    if (stop_daq)
      run_daq = false;
    else
      GoToFirstState();
}
```

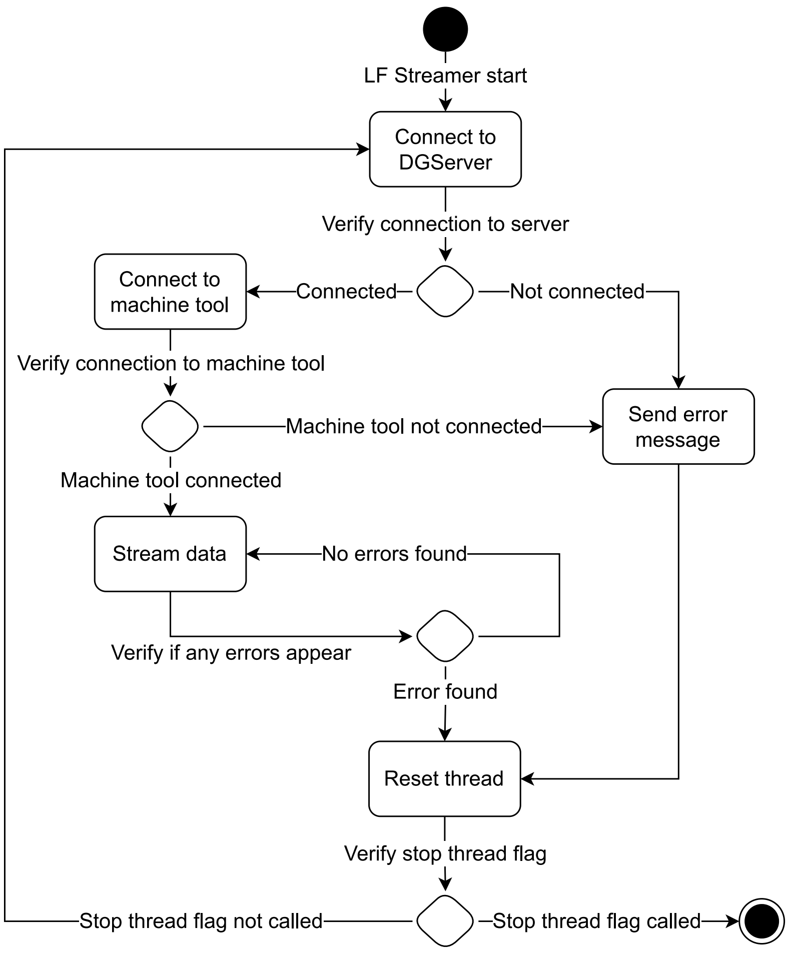
To avoid keeping useless connections open to the Data Ground Server, the Reset state performs a disconnect function to an existing connection based on the *DisconnectToDGServer*. The acquisition threads, when finished, are also further closed in this step by the pseudo routine *JoinAllThreads*, providing a safe and correct way to abort the entire program flow. Once the threads are terminated and all connections are shut down, the Reset state defines, again based on boolean flags, the execution path to follow. If the *stop_daq* is set to true, a bool signal triggers the end of the program. Otherwise, the processes have to be restarted and the workflow returns to the initial step of the controller identification.

Considering the known states of the Main Thread, it is notable that this branch of the code remains in an idle status while the acquisition threads are running. The Low Frequency Streamer, being one of them, is introduced as follows, explaining in detail its corresponding operations and their respective implementations.

3.3.2 Low Frequency Streamer Thread

The LF Streamer Thread is directly responsible for the acquisition and dispatching of all low frequency data, previously seen in Table 1. Similar to the Main case, the defined states of this thread (Connect to DGServer, Connect to Machine Tool, Stream Data, and Reset Thread) work in an ordered sequence, which is shown in Figure 18.

Figure 18 – Activity diagram of LF Streamer Thread.



Source: Author.

After being instantiated, the LF Streamer first performs connections to the server and to the operating machine tool in order to establish a unique communication channel between them for these types of data. When these routines are successfully completed, the information stream is initiated in a loop, which consists of repeatedly acquiring the selected data from the controller, manipulating it in a simple way and dispatching it directly to the DGServer, until the acquisition is finished. Once all the desired information is collected from a single run, it is brought together to build and send a metadata object, which labels and organizes machining processes in future company applications.

A procedure comparable to the Main Thread, described in Code Snippet 2, is required for the LF Streamer to also connect to the DGServer. Again, the server's connection subroutine dictates the thread execution path. If the method is successful and the coupling of both services is indeed true, the next state is set to Connect to Machine Tool. Otherwise, an error message is thrown and the Reset stage is triggered to handle the problem.

Furthermore, an additional connection to the machine tool must be established to properly access the controller's memory and acquire the desired low frequency data. As previously described in the Main Thread, this procedure is heavily based on the used library and can be simply implemented with a call of the pseudo routine *ConnectToMachine*, seen in Code Snippet 1. If this method returns success, the Stream Data state is executed, where most of the logic part of this thread is located.

3.3.2.1 Stream Data

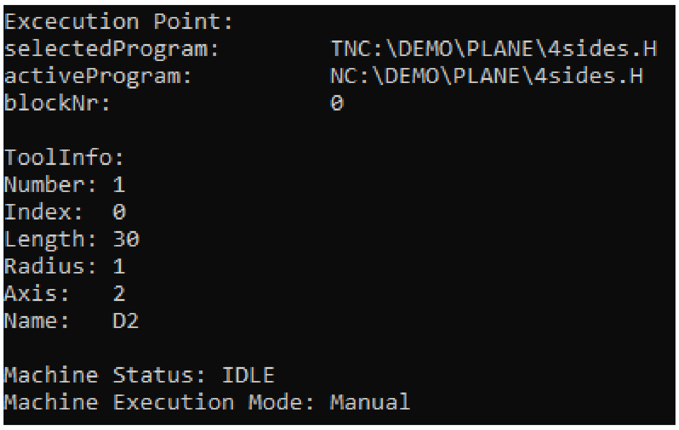
When a clean data stream path is built upon open connections to the operating CNC controller and to the DGServer, the acquisition and dispatching of the low frequency data takes place in a loop form. In this case, each type of information is acquired sequentially and its results are subsequently manipulated and dispatched to the server one by one via their respective dispatch methods created. The main processes running within the Stream Data state are summarized in the Code Snippet 6.

Code Snippet 6 – Low frequency data streaming.

```
if (stream) {
  bool ok = AcquireLowFrequencyData();

  if (ok)
      HandleLowFrequencyData();

  // Create and dispatch metadata object
  HandleMetaDataObject();
}
```

The machine data acquisition is mainly performed by a couple of specific functions, represented in here together by the pseudo routine *AcquireLowFrequencyData*, which together satisfy the low frequency requirements. They are designed for each desired data type and developed based on internal subroutines. These methods are able to access the controller's memory and retrieve the stored values, that are further used to correctly run the dispatch steps of an execution (illustrated by the *HandleLowFrequencyData* function). Figure 19 shows an example output of this low frequency data.

Figure 19 – Example of low frequency data output from the LF Streamer


```
Excecution Point:
selectedProgram:        TNC:\DEMO\PLANE\4sides.H
activeProgram:          NC:\DEMO\PLANE\4sides.H
blockNr:                0

ToolInfo:
Number: 1
Index:  0
Length: 30
Radius: 1
Axis:   2
Name:   D2

Machine Status: IDLE
Machine Execution Mode: Manual
```

Source: Author.

At the top of Figure 19, the execution point provides information about the path and name of the selected and active program running on the machine tool which, in this case, are the same. The "blockNr" also shown, specifies the *selectedProgram's* line that is currently being read by the controller. Next, the cutting tool data is displayed, followed by the machine's status and the execution mode at the end.

A boolean is returned by the acquisition function in Code Snippet 6, representing a success or failure in the corresponding process. In case it is executed correctly, a couple of sequential operations are performed within the *if (ok)* statement: check if the new values of the data variables are the same as the last ones, standardize the data into programming objects and dispatch them to the server with the respective dispatcher function. This validation of two consecutive data points is necessary to prevent this DAQ Software from sending repeated low frequency data to the other services in the chain, since for larger machining processes it is common for the acquisition methods to read the same data several times. However, when they are different, the two remaining activities cited are accomplished in sequence.

Once all instructions are completed, the acquired information is gathered together to build and dispatch a metadata that identifies a unique machining operation with its associated data and a serial number, represented, in this case, by the *HandleMetaDataObject* function. A metadata object is only constructed when a new cutting tool or a newly selected program is read.

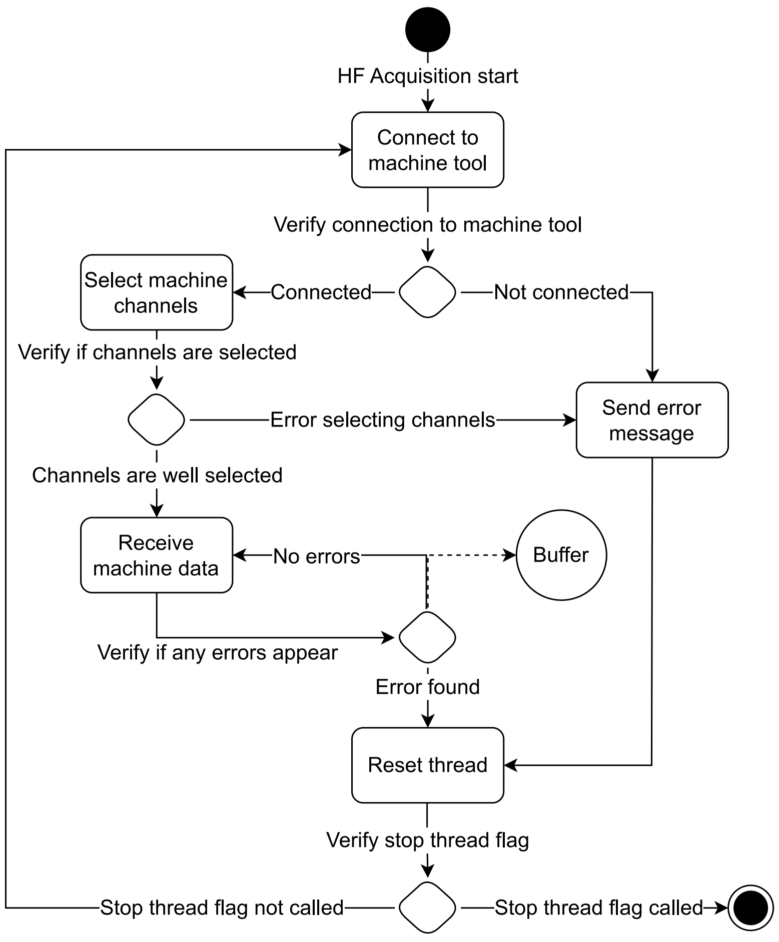
If any state of the LF Streamer returns an error, the Reset Thread stage is triggered to properly handle the situation. Like the similar stage of the Main Thread, the connections here are safely closed with the DGServer and the machine tool, aborting the corresponding workflow. In the end, if a stop flag is set to true, the thread is simply terminated. Otherwise, the LF Streamer is started again from the first connection to the server.

Meanwhile the low frequency data are being acquired, high frequency threads are also running in the background in order to fill in all the information of a machining process. The in-depth explanation and implementation of such threads are discussed below, starting with the High Frequency Acquisition Thread.

3.3.3 High Frequency Acquisition Thread

As the name suggests, the HF Acquisition Thread mainly conducts the acquisition of the high frequency data types, described in Table 1. Following the example of the threads already shown, it represents a state machine that features different and well-defined sequential steps (Connect to Machine Tool, Select Machine Channels, Receive Machine Data, and Reset Thread), which are visually explicit in Figure 20.

Figure 20 – Activity diagram of HF Acquisition Thread.

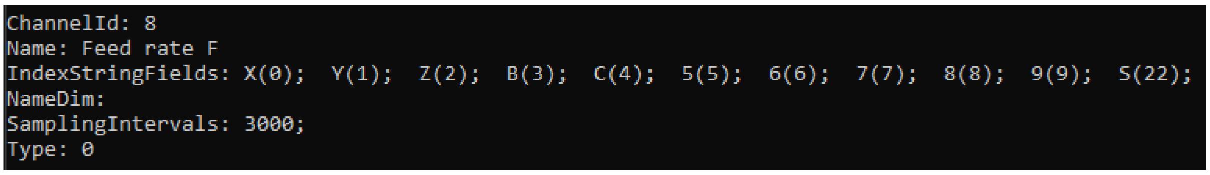


Source: Author.

A special connection to the machine tool to access the Scope Information area is needed first for this thread to work as expected. This part of the controller memory is built on multiple channels, in which each is associated with a single type of high frequency data. Once the DAQ and the CNC are connected, the channels are selected by the user in order to get the exact type desired and to prepare the control to send the specific data that is of interest to this program. When the channels are chosen correctly, the information is received by this software, with the help of particular functions from the used library, and directly written to a buffer, which communicates the HF acquisition with the dispatch thread.

To properly enter the controller's scope area, a more sophisticated connection is required than those implemented for the threads already presented. In addition to the known connect function, a login is needed to specify particular passwords to access this part of the memory. Furthermore, in order to obtain auxiliary information to work with the scope, additional methods from the library must be invoked to acquire general details about this region of the control and to gather metadata about all channels available on the CNC. An output example of these routines is shown in Figure 21, where information only regarding the Feed Rate channel is exposed. Such metadata is useful and is better explained in the Select Channels step of this thread.

Figure 21 – Feed Rate channel metadata from the Scope area.


```
ChannelId: 8
Name: Feed rate F
IndexStringFields: X(0);  Y(1);  Z(2);  B(3);  C(4);  5(5);  6(6);  7(7);  8(8);  9(9);  S(22);
NameDim:
SamplingIntervals: 3000;
Type: 0
```

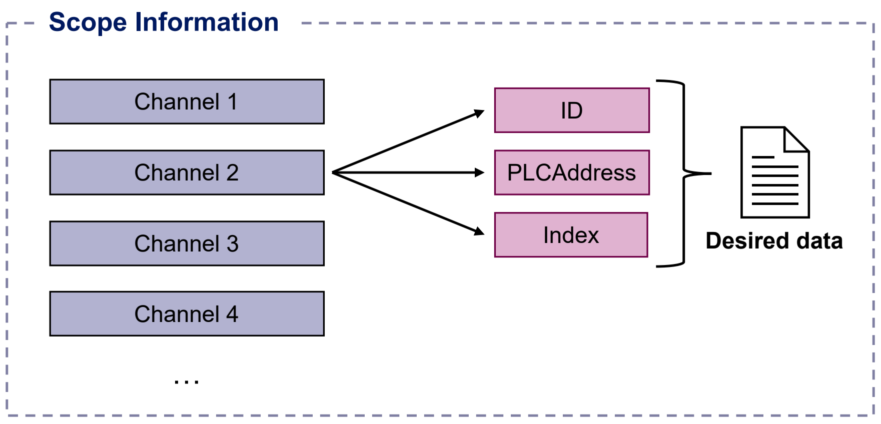
Source: Author.

The same logic using *if (ok)* statements is also implemented in this connection stage, aiming to run the next function only when a successful execution of the previous one has been performed. With all methods executed correctly, the Select Channels state is set. Otherwise, the Reset Thread is triggered just like the other branches of the code, as seen in Figure 20.

3.3.3.1 Select Channels From Controller

The channels that build the scope area are mainly composed of three properties: ID, PLCAddress, and Index. Together, they are able to define the exact location where the desired high frequency data type is stored within the control memory. When a user needs to perform a channel selection to properly obtain an information, these three attributes must be set to valid values according to results similar to that shown in Figure 21. How the Scope Information is constructed is better illustrated in Figure 22.

Figure 22 – Scope Information area structure.



Source: Author.

Considering the output content of the example in Figure 21, the displayed fields show the necessary information to correctly set the channel properties and, consequently, select a channel itself. "ChannelId" defines the channel identifier within the scope area. "Name" specifies the data type. "IndexStringFields" represents the axis names and their corresponding indexes. "NameDim" stands for the unit of the acquired signal. "SamplingIntervals" refers to the sampling interval in microseconds. And finally, "Type" explains the channel type in the control. To regulate the three attributes mentioned in Figure 22, the "ChannelId" and "Type" keys need to be taken into account.

For the Feed Rate channel, for example, the ID is set to 8 (according to "ChannelId"), the PLCAddress to -1, and the Index to 0. These last two settings are based on the "Type" being 0, since these are specifications from the used library documentation. An implementation for this case is exposed in Code Snippet 7.

Code Snippet 7 – Scope channel selection for high frequency data acquisition.

```
if (select_channels) {
    // Select channel with the correct attribute values
    bool ok = SelectChannel(8, -1, 0);

    // If the channel was successfully selected
    if (ok)
      GoToNextStep();
    else
      GoToReset();
}
```

Despite showing only the Feed Rate example in Code Snippet 7, the channel selection is also performed for all axis signals, such as coordinates and loads, and spindle speed and load, effectively achieving the requirements imposed in Section 3.1. Once the values of each wanted channel are found, they must be passed to the controller, in this case, performed by the pseudo function *SelectChannel*, to define them within its scope. If this method is successful, the channels have been rightly chosen and the CNC is prepared to send the proper information, that will be received by the Receive Data state. Otherwise, an error has occurred and the Reset is set.

3.3.3.2 Receive Data From Controller

After the channels are selected, the acquisition of high frequency data is finally started. The receiving of such information is mainly performed by two functions that are triggered sequentially during the acquisition process. This procedure runs in a loop that ideally maintains the HF Acquisition Thread in this state during the entire execution. Code Snippet 8 addresses the implementation of these operations.

Code Snippet 8 – Receiving high frequency data from the controller.

```
if (receive_data) {
    bool ok = AcquireHFData();

    if (!ok)
        GoToReset();
}
```

Every time a new piece of data is received by the pseudo routine *AcquireHFData*, a callback function is directly triggered. This callback method is primarily responsible for organizing the information into an array and writing it to the common high frequency buffer, as seen in Figure 20. For each execution of the loop, the acquisition function returns a success or failure flag represented by the boolean *ok*. Based on this result, the code constantly checks if an error appears. If any problem occurs on this reception, an error message is thrown and the Reset Thread state is set.

Serving as the corresponding state of the other threads, the Reset is set to provide a safe and clean way to handle any HF Acquisition issue. Based, again, on the *DisconnectToMachine* function from Code Snippet 1, this stage closes, in a correct manner, the connection previously established with the machine tool. Following the same pattern, a stop flag defines whether the current thread is terminated or restarted.

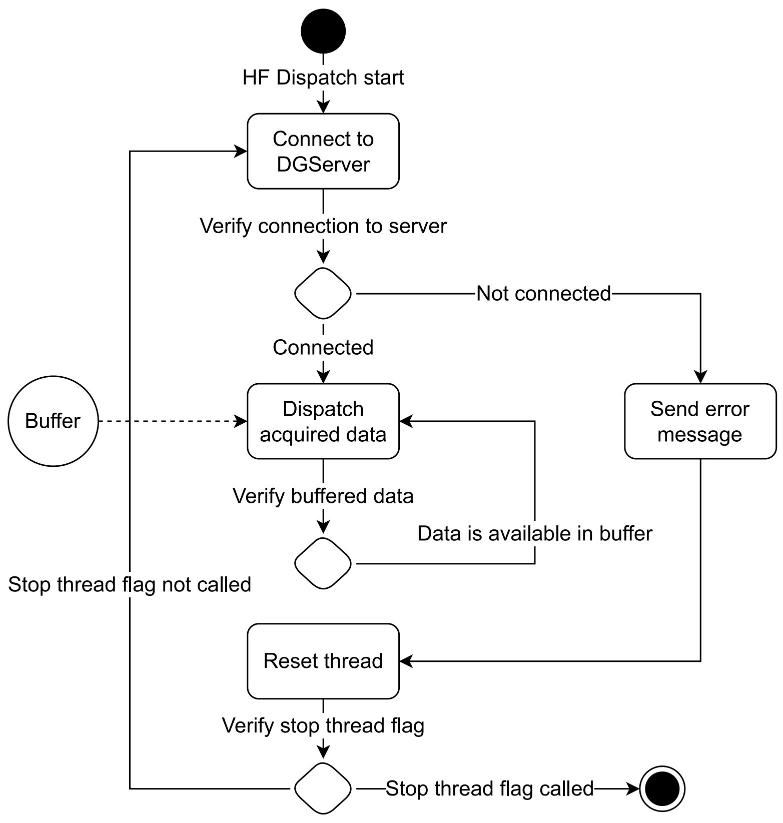
Taking into consideration the amount of data and operations performed in a small period of time, the high frequency threads are separated in two, aiming to avoid overloading a single one. The information that was earlier acquired and written to the buffer is further read and handled by the HF Dispatch, explained in the next subsection, which finishes the whole acquisition process.

3.3.4 High Frequency Dispatch Thread

The last implemented thread, the HF Dispatch, is responsible for sending the acquired high frequency data to the server in order to continue the company's processes.

Just like the other threads, its steps are well defined in different states (Connect to DGServer, Dispatch Acquired Data, and Reset Thread), that are internally connected according to Figure 23.

Figure 23 – Activity diagram of HF Dispatch Thread.



Source: Author.

A successful run for this thread starts with multiple connections established to the DGServer, one for each desired data type, to also avoid the overhead of a single network connection. Once connected, the sending routine is executed in a loop: first, the information in the common buffer is read and then transformed into programming objects that are sent to the server via the respective dispatch function created.

To perform such connections to the DGServer, the procedure is similar to the corresponding ones seen in the Main Thread and in the LF Streamer. However, instead of connecting only once, here it should be at least six times due to the number of high frequency data types of interest. Considering this, the known *ConnectToDGServer* pseudo function is called within a *for* loop, returning six connections that are stored in a vector, thus allowing index-based access. When each execution of this method is successful, data can be sent to the server by the Dispatch Acquired Data state.

If any error occurs during the attachment of this HF Dispatch to the server, the Reset Thread is triggered to solve the problem. In this step, each open connection is closed individually with a *DisconnectToDGServer* method call inside a *for* loop. Fulfilling the same pattern as the other Reset stages, a boolean flag defines whether the thread restarts or the acquisition is terminated.

3.3.4.1 Dispatch Acquired Data

While the high frequency data are being written to the buffer by the HF Acquisition thread, the HF Dispatch is simultaneously reading this information in order to send it to the connected server. Code Snippet 9 presents the simplified implementation of the operations performed for this state.

Code Snippet 9 – Dispatching of high frequency data to the DGServer.

```
if (dispatch) {
    if (buffer has data) {
        double data = buffer pop front;
        DispatchHFData(data);
    }
}
```

Since the dispatch process must only occurs when data is being acquired from the machine tool, it is necessary to check, in this high frequency case, whether there is information being written to the buffer. If the buffer is empty, no data is stored and this thread enters in an idle state until the next verification of the loop. Otherwise, the front data is obtained and popped from the buffer to continue the next steps.

Having this information available, each type of high frequency data can then be asynchronously transformed into programming objects and dispatched to a single connection to the server over the network, finalizing, in turn, the entire Heidenhain DAQ workflow. Such process is summarized in Code Snippet 9 based on the pseudo function *DispatchHFData*.

Once the operations of this HF Dispatch thread are finished, the entire service is completely covered and all requirements listed in Section 3.1 are successfully achieved. With all four threads working together, the Heidenhain Data Acquisition software is ready for testing and to have its results analyzed.

4 TESTS AND RESULT ANALYSIS

When the appropriate tasks are performed by the four threads of the Heidenhain DAQ, the high and low frequency data are sent to the server that forwards them to the other applications of the company. This information, in turn, triggers various data processing operations and their respective management in the database, until it reaches the Front-End, the final service that displays the acquired and processed data in the form of digital twins. In order to present this gemineers' product operation and its results, this DAQ needs to be tested in a proper manufacturing process.

Therefore, Section 4.1 of this chapter aims to describe the testing activities conducted by the student for the developed software, as well as its corresponding results that are directly reflected in the visualization in the Front-End. Then, in Section 4.2, a comparison with the company's main acquisition service, Siemens DAQ, is shown with the intention of analyzing their differences and providing possible improvements and future work regarding both data acquisitions. An in-depth analysis of the performance of each primary Data Ground application is introduced in Section 4.3, where tests evaluating the memory and processor usage of the local computer are taken into consideration.

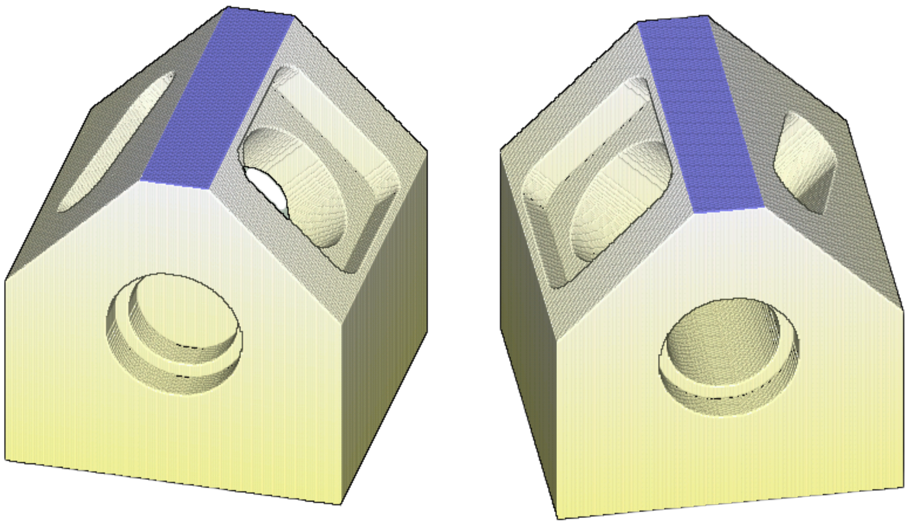
4.1 DATA ACQUISITION FUNCTIONALITY

In order to explore the functionality of the developed Heidenhain DAQ, a machining operation is first required to correctly trigger the activities performed by this software. Since there are no machine tools operated by Heidenhain controllers available in the company at the time the tests are carried out, the iTNC 530 Programming Station (previously shown in Chapter 2) is widely used to emulate these controls and to simulate milling cuts on a block of raw material. The simulation application along with the necessary Data Ground services (Data Acquisition, DGServer, and Batcher) are then started on a computer that has an Intel(R) Core(TM) i5-8265U 1.60 GHz CPU with 8 GB of RAM to properly experiment the entire workflow of the related area.

After these applications are initialized and rightly connected to each other according to the steps presented in the implementation chapter, the test procedure begins with the setup of a machining operation in the mentioned Programming Station. First, the kinematics are chosen to correctly simulate a real manufacturing process, and their respective parameters are passed to the DAQ configuration file. The kinematics defined for this testing phase replicate the old Heidenhain machine tool of the High Performance Cutting department of Fraunhofer IPT.

Next, the "Program run, full sequence" option is selected to fully execute a G-Code routine, also specified by the user, within the emulated controller. For this test occasion, to properly analyze the Heidenhain DAQ and, consequently, the next services of gemineers, the program "house.H," from the internal folders of the simulation software, is used. This file consists of multiple 5-axis machine tool instructions for turning a rough block into a birdhouse, based on various milling operations, effectively providing an environment to completely experience the features of the developed application. The final house workpiece, according to the Programming Station data, should have the geometry illustrated in Figure 24. The solid is displayed in two different perspectives placed side by side to show all the holes and cuts that are made by the cutting tool in the initial rectangular part.

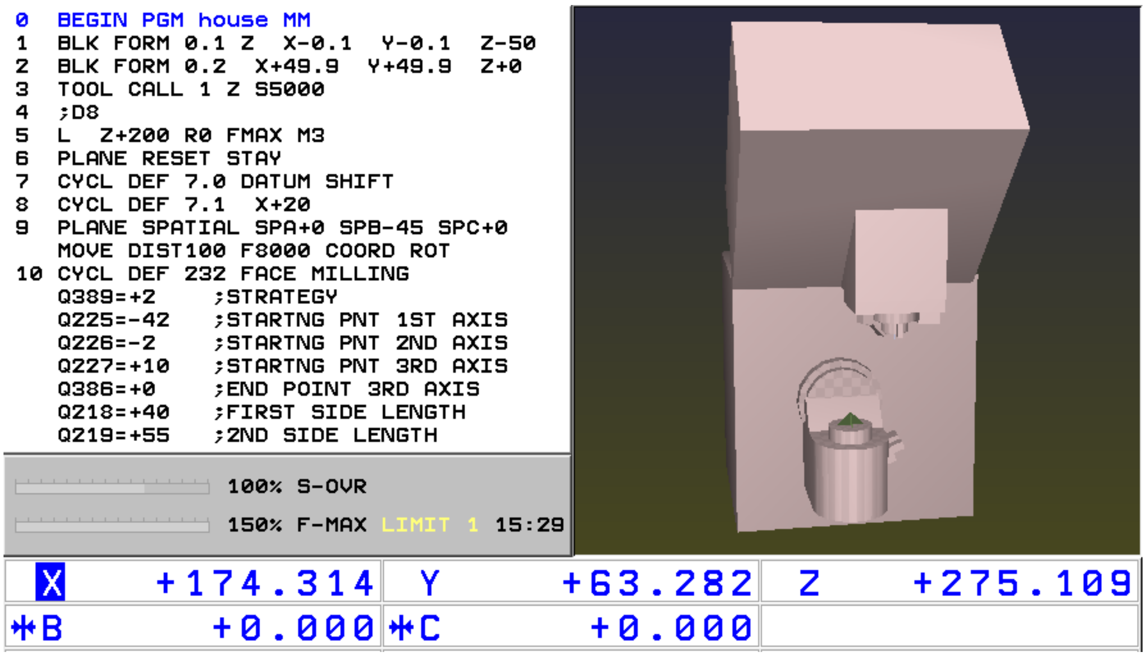
Figure 24 – Original house geometry from iTNC 530 Programming Station.



Source: Author.

Once each mentioned step is performed, the machining simulation is started and further controlled based on the virtual keyboard provided. Figure 25 shows an example of the initialization of such tests in the Programming Station. On the left side of the image, the picked program is displayed, along with the chosen kinematics on the right, and the axes and their initial positions at the bottom. When in execution, the blue line highlighted in the G-Code advances to the next ones, visually realizing the described motion in the machine tool, represented by the kinematics and the coordinate values. In addition, several other different functionalities can be achieved with this simulation software, but this is already beyond the scope of this project.

Figure 25 – Initial Programming Station setup for simulating a machining operation.



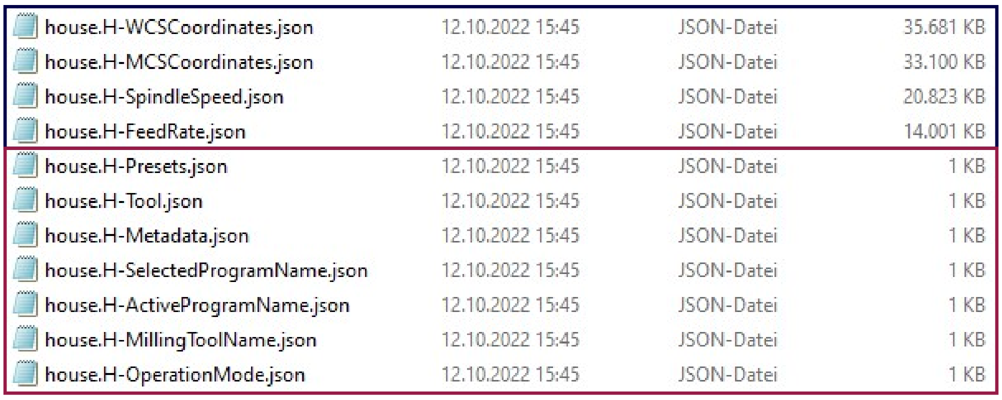
```
0  BEGIN PGM house MM
1  BLK FORM 0.1 Z  X-0.1  Y-0.1  Z-50
2  BLK FORM 0.2  X+49.9  Y+49.9  Z+0
3  TOOL CALL 1 Z S5000
4  ;D8
5  L  Z+200 R0 FMAX M3
6  PLANE RESET STAY
7  CYCL DEF 7.0 DATUM SHIFT
8  CYCL DEF 7.1  X+20
9  PLANE SPATIAL SPA+0 SPB-45 SPC+0
   MOVE DIST100 F8000 COORD ROT
10 CYCL DEF 232 FACE MILLING
   Q389=+2     ;STRATEGY
   Q225=-42    ;STARTNG PNT 1ST AXIS
   Q226=-2     ;STARTNG PNT 2ND AXIS
   Q227=+10    ;STARTNG PNT 3RD AXIS
   Q386=+0     ;END POINT 3RD AXIS
   Q218=+40    ;FIRST SIDE LENGTH
   Q219=+55    ;2ND SIDE LENGTH

            100% S-OVR
            150% F-MAX LIMIT 1 15:29

X      +174.314  Y      +63.282  Z      +275.109
*B       +0.000  *C      +0.000
```

Source: Author.

While the simulated machining operations are taking place, the data acquisition processes are running in the background. Since the Heidenhain DAQ is directly connected to the virtual machine tool, each movement is being recorded, along with the other desired information, to accurately obtain all cuts and hoes performed by the tool tip. For a simple validation of the high frequency data, the TNCscope software (introduced in Chapter 2) is manually checked to make sure that the information acquired by the developed application is in accordance with Heidenhain's official data. If they are equal, the activities continue and all the information from low and high frequency is sent to the DGServer, which triggers the Batcher execution with the subscriptions behavior.

The Batching Service, in turn, gets the acquisition data to group it into batches, effectively linking the metadata with the corresponding mechanical information. For the house example, an output from the Batcher is presented in Figure 26, where the data for each type is stored in unique files within a single zip folder. Unfortunately, the Heidenhain Programming Stations do not feature the loads for any axis and spindle, which is why such desired information is omitted. Furthermore, from Figure 26 it is also noticeable the difference in quantity between the high and low frequency data, as all LF documents (at the bottom rectangle) are represented by 1 KB and the smallest HF file (at the top) reaches about 14 MB.

Figure 26 – Output data from the Batcher for the house.H.

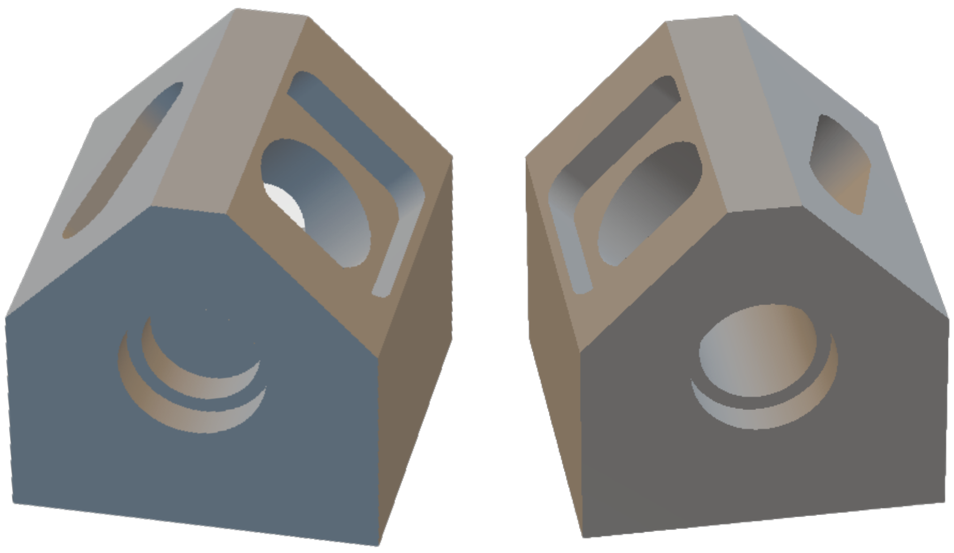


house.H-WCSCoordinates.json	12.10.2022 15:45	JSON-Datei	35.681 KB
house.H-MCSCoordinates.json	12.10.2022 15:45	JSON-Datei	33.100 KB
house.H-SpindleSpeed.json	12.10.2022 15:45	JSON-Datei	20.823 KB
house.H-FeedRate.json	12.10.2022 15:45	JSON-Datei	14.001 KB
house.H-Presets.json	12.10.2022 15:45	JSON-Datei	1 KB
house.H-Tool.json	12.10.2022 15:45	JSON-Datei	1 KB
house.H-Metadata.json	12.10.2022 15:45	JSON-Datei	1 KB
house.H-SelectedProgramName.json	12.10.2022 15:45	JSON-Datei	1 KB
house.H-ActiveProgramName.json	12.10.2022 15:45	JSON-Datei	1 KB
house.H-MillingToolName.json	12.10.2022 15:45	JSON-Datei	1 KB
house.H-OperationMode.json	12.10.2022 15:45	JSON-Datei	1 KB

Source: Author.

From the Batcher's thread, the batched data, represented by the zip folder, is forwarded to the gemineers API to further start refining such information. From the execution of the Data Processing area services, the raw data acquired from the controller is cleaned, treated, and converted into a virtual representation of the finished workpiece, based on the transformation into a Standard Triangle Language (STL) file. Thus, considering once again the example of the house, Figure 27 presents one of the results of these processing steps, formed over the information obtained by the Heidenhain DAQ from the machining simulations previously performed by the Programming Station.

Figure 27 – Resulting house geometry from gemineers' data processing services.

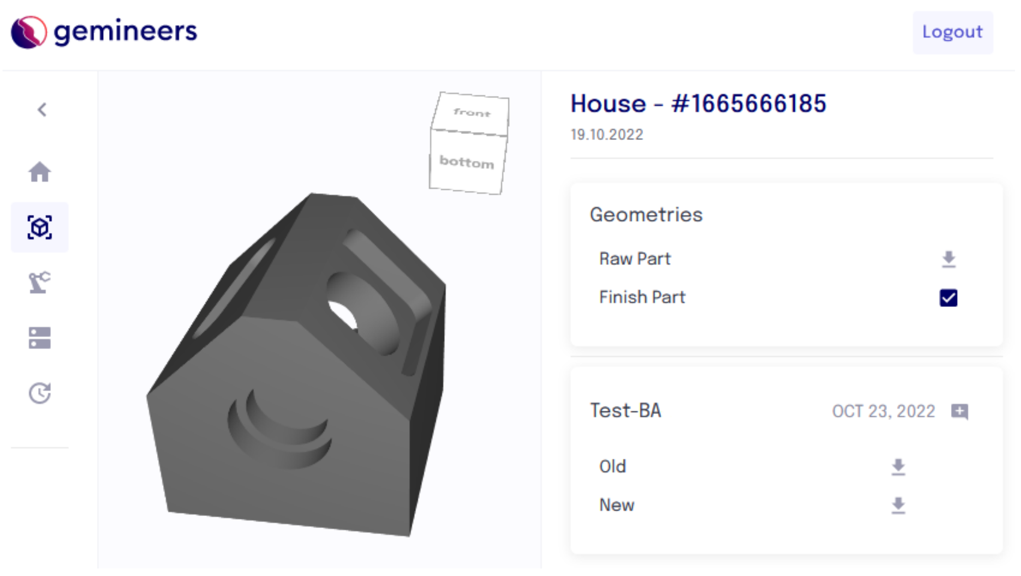


Source: Author.

By comparing Figure 24 and Figure 27, it is possible to conclude that the Heidenhain DAQ is effective at acquiring data from the controller, at least in terms of tool movements and axis positions. This is due to the fact that all the cuts and holes expected by the original workpiece were correctly mapped into the result of the gemineers data processing, thus successfully achieving the desired digital twin. To further analyze the information acquired for different types, the visualization is needed within the Front-End, where the data is sent and loaded after it has been processed.

The gemineers' Front-End service is a browser platform for the user to monitor the machining processes of the shop floor. Among its features is the analysis of the data acquired by the DAQ over the original piece geometry in a 3D environment. Figure 28 illustrates with the house example the current state of such an application, with the visualization (in the center) of the selected data (on the right) displayed on the screen.

Figure 28 – gemineers' Front-End.

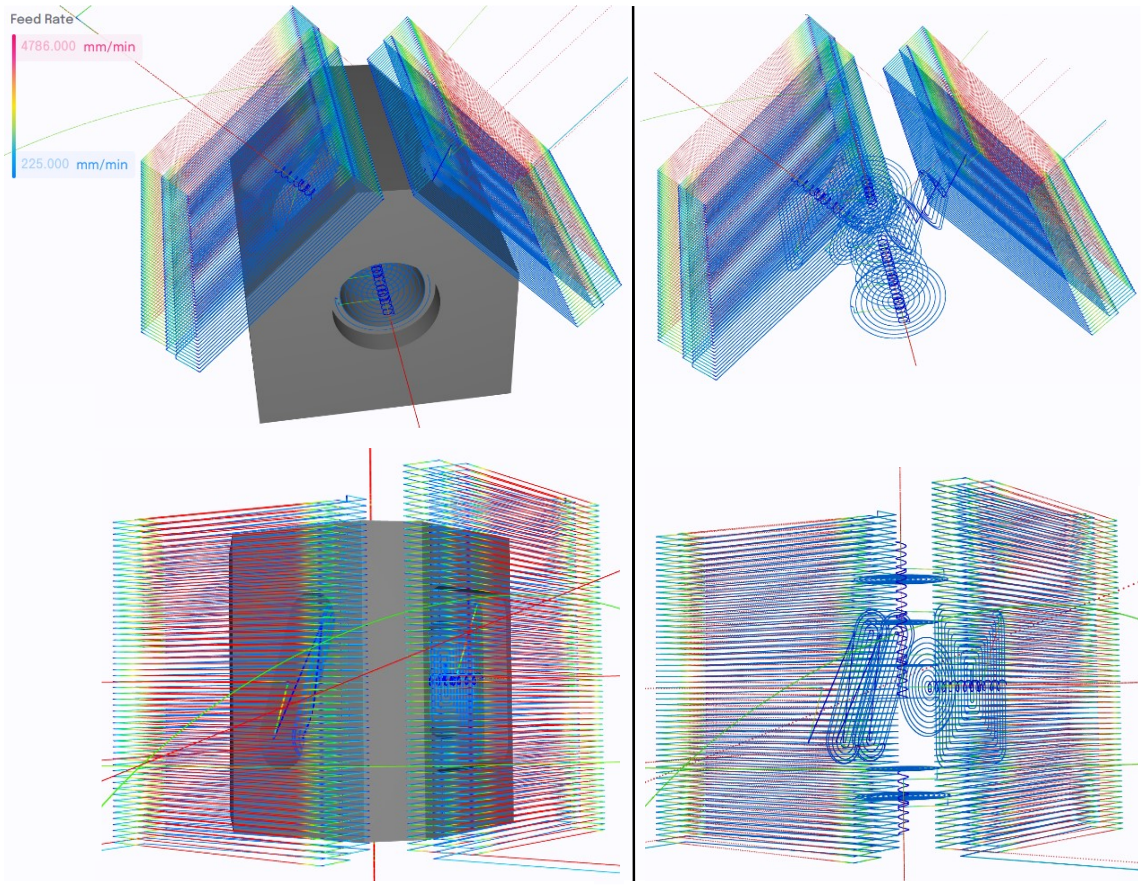


Source: Author.

When any process data is loaded into the visualization frame, a mesh of points is displayed to simulate the path taken by the cutting tool to machine the desired part. Each of these cut points represents a data instance acquired by the DAQ, which contains the values of the mechanical information synced with the positions in the space.

To manipulate how these data are displayed, a small navigation bar is implemented at the bottom of the view. This component allows the user to select how and what type of information is presented in order to monitor any obtained mechanical variable of the machining process. These data are explicitly shown point by point on a color scale, along with a legend to correctly identify the respective values acquired by the DAQ for a given position. Such representation for the house geometry is visually exemplified in the Front-End application in Figure 29, where the Feed Rate signal is highlighted. The images on the left side illustrate the colored point cloud over the 3D model of the workpiece, while on the right, only the point mesh is exposed. At the bottom of Figure 29, a top view of the part and hence of the data points is exhibited to further analyze the holes drilled by the cutting tool in the roof of the birdhouse.

Figure 29 – Feed Rate data displayed as a point cloud in the Front-End.



Source: Author.

Considering the results in Figure 29, it can be inferred that the developed Heindenhain DAQ software is also capable of collecting and sending information of different types of data, as exemplified by the Feed Rate. Such a statement can be confirmed from the analysis of the color scale and the legend in the upper left corner, which translates and associates the value of the signal in question for each cutting tool point acquired during machining and displayed in the space. For the case shown, the Feed Rate ranging from 225 mm/min, in regions where the tool is definitely milling the workpiece to 4786 mm/min in places where touching the part does not occur, is consistent with what is expected for the house process.

Moreover, to obtain the visualization of all information synchronized with the corresponding machining operations of the selected geometry, the metadata and all other types of low frequency data, which are responsible for this association, need to have reached the other services of gemineers. With this in mind and the results presented so far, it can be concluded that the Heidenhain DAQ was successful in achieving all the objectives set for the project, since the stipulated low and high frequency data were acquired and handled until the final Front-End application in a digital twin form. When the Data Acquisition software is ready for use and properly attached to the other services, the company can then finally attain the desired process monitoring and further predict and assure better quality for the production line and the manufactured workpieces.

Now that the Heidenhain DAQ results have been discussed and visually displayed in the gemineers' Front-End, its features need to be compared to the Siemens DAQ, the other existing data acquisition application. This comparison, better introduced in the following section, is necessary in the scope of the company to validate and verify the common functionalities of these services, aiming to analyze possible improvements for both acquisitions, taking into account the uniqueness of each control.

4.2 COMPARISON TO SIEMENS DATA ACQUISITION

The data acquisition from Siemens controllers is the main DAQ software of the company nowadays, since it has been in development for a long time and has constant updates for new features. In order to keep the Heidenhain DAQ in sync with such an application, given that they have the same objectives, their primary similarities and differences are further studied and listed with the goal of keeping both services even and delivering to the end user a complete software regardless of the controller used.

In terms of the overall data acquisition requirements, it can be stated that the two applications are successful. This is due to the fact that both the Siemens DAQ and the Heidenhain DAQ are able to connect to the respective machine tools, acquire their base data and forward it to the DGServer over the network. The major differences between them are, in turn, based on three points: the ease of software development, the quantity of available data types to be obtained from the CNC, and the service's frequency of acquisition of the selected information.

The facility of developing the software is indirectly linked, in this case, with the amount of variables that can be acquired from the control. This relationship exists due to the libraries applied for both applications. For the Siemens scenario, the imported coding packages are better equipped with documentation, providing the developer with a simpler understanding of the architecture and how to properly obtain the desired data. Besides being easier to use, the range of data types that can be acquired in these controllers with the corresponding library is greater than that of Heidenhain, and there is no restriction on the number of variables that can be selected at the same time.

When it comes to the Heidenhain case, the library documentation lacks details and implementation support, making it more difficult to understand the data structure, access the memory areas of the controller, obtain specific types of information, and therefore develop code. Additionally, a number of high frequency variables are limited to the Scope area, restricting the amount of data that can be obtained by this DAQ. To adequately deal with such problems and achieve all the features that Siemens has, an even deeper study of the library available for Heidenhain is required.

Regarding the acquisition speed, the Heidenhain DAQ performs better in comparison to the Siemens service. The former is capable of acquiring data at a constant frequency of about 330 Hz, while the latter obtains information at 110 - 250 Hz unsteadily. The reason for this difference is, besides the velocity of the specific read functions, that the Siemens DAQ implements the sending of data sequentially on the same thread as the acquisition, and has to wait until the dispatching is complete before the next data can actually be obtained. The Heidenhain DAQ, on the other hand, separates and executes the acquisition and sending operations simultaneously on different threads, making both high frequency and stability effective.

Given the computational cost that is required to operate all main Data Ground services at a 330 Hz acquisition rate for the Heidenhain case, the next section aims to explore the computer-wise performance of each application (Heidenhain DAQ, DGServer, and Batcher) during a machining process.

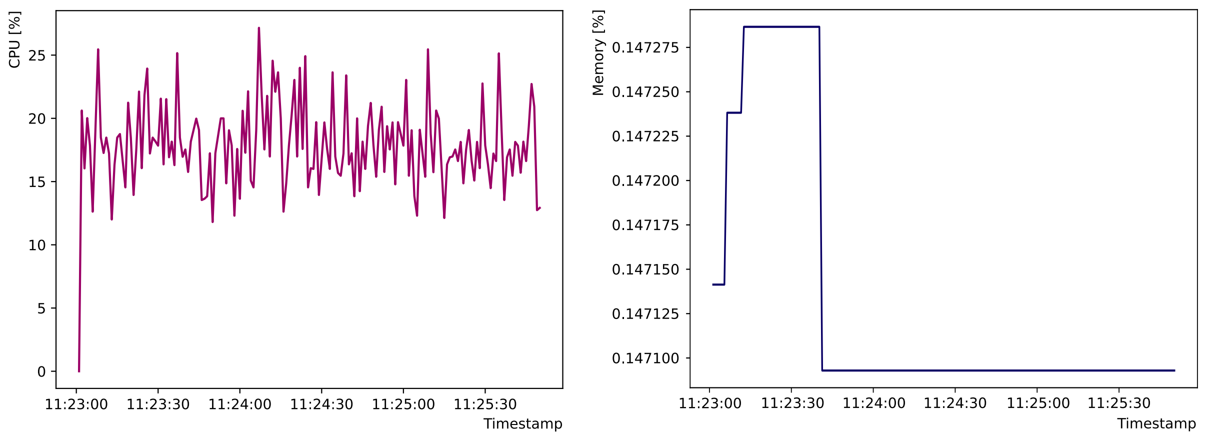
4.3 DATA GROUND SERVICES PERFORMANCE

In the current state of the gemineers software, the entire Data Ground area works with one data point at a time. This means that for each unique information acquired by the DAQ, a single dispatch is executed that triggers isolated operations in the DGServer and in the Batcher. For a 330 Hz acquisition achieved by the developed Heidenhain application, this amount of processes happening simultaneously and in multiple services is computationally expensive and needs to be studied.

Thus, to verify the computational performance of each Data Ground software and to better understand how they behave for this high frequency acquisition, tests are performed in this regard. The testing takes place with the help of an existing python script within the company's scope, which records sample information about CPU and memory usage and plots it on graphs. These processing data that are further shown in this section were obtained during the simulations of the house program, previously discussed in Section 4.1, in the same computer already presented.

The three services are explored in sequence according to their order of execution, i.e., the Data Acquisition, the DGServer, and then the Batcher. Figure 30 first illustrates the computer-wise performance of the Heidenhain DAQ during a complete run of the gemineers product for a manufacturing process. On the left, the chart relating CPU usage is shown, while on the right, memory is exposed, both as a percentage of the corresponding total amount available in the computer.

Figure 30 – Heidenhain DAQ performance for CPU and memory.

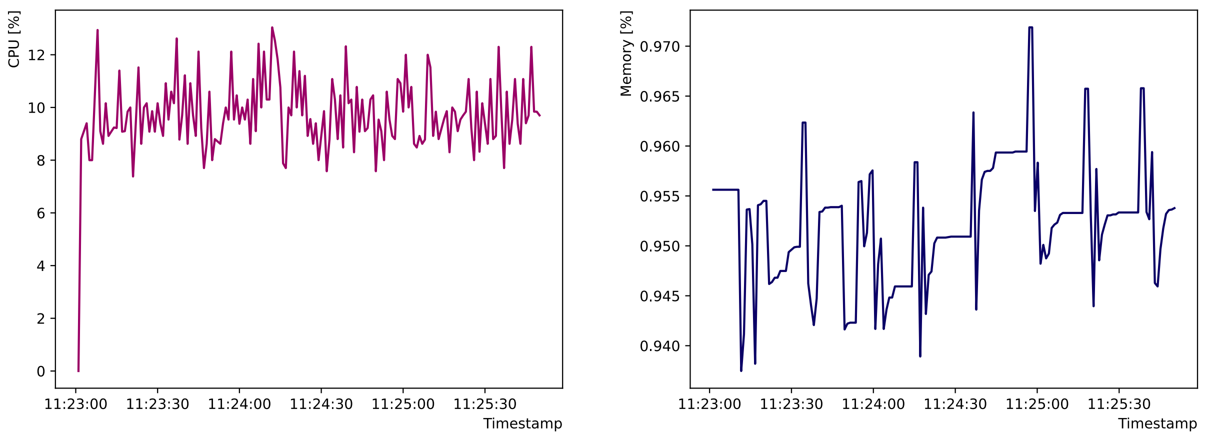


Source: Author.

From Figure 30, it can be concluded that the developed Data Acquisition uses a lot of CPU and little memory, taking into account the entire computer, since the respective averages are around 18% and 0.147%. This behavior can be explained, for the CPU, due to the high number of threads and operations that need to be performed in a short period of time, and, for the memory, due to the volatility of the data, as the acquired values are overwritten repeatedly at each iteration on the same variables created within the application.

This mentioned principle of CPU and computer memory usage can be applied also for the DGServer and the Batcher, as seen in Figure 31 and Figure 32, correspondingly. The layout of the graphs in these two following figures is the same as the one shown in Figure 30 for the Heidenhain DAQ.

Figure 31 – DGServer performance for CPU and memory.

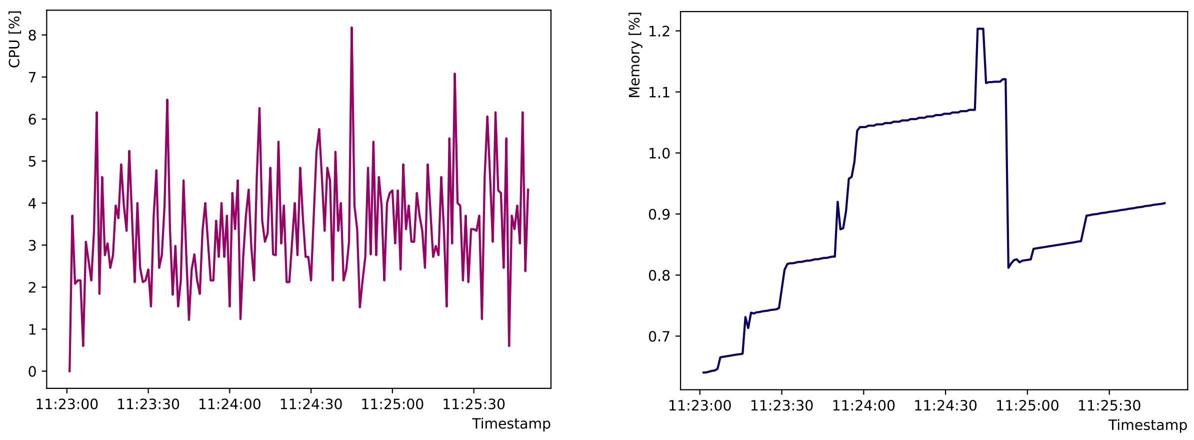


Source: Author.

Considering that the DGServer performs fewer operations, generally with mutations and subscriptions, in the same period of time when compared to the Data Acquisition software, it is visible in Figure 31 that the server's CPU usage is also lower, averaging about 10%. When it comes to memory, on the other hand, DGSserver reaches a higher value than the DAQ, coming close to 1% of the overall available. Although the data is still volatile and is overwritten every iteration, this happens because the service in question makes use of more complex and nested data structures and not of native types such as strings and doubles as seen in the Heidenhain application.

The Batcher, in turn, since most of the time it is just appending each acquired data point to a buffer, presents the lowest CPU usage, of about 3.5% - see Figure 32. Because of the size of this buffer, its memory utilization ends up exceeding the other services studied, reaching up to 1.2%.

Figure 32 – Batcher performance for CPU and memory.



Source: Author.

In view of all the results presented in the current section, it is conclusive that the Data Ground applications can be improved regarding computational performance, at least when it comes to CPU usage, since three services occupy approximately 31.5% of the available processing time in the computer. With this in mind, possible optimizations need to be proposed and developed, as explained in the next chapter, in order to decrease their CPU utilization and make the software accessible to less powerful computers.

5 SERVICE OPTIMIZATIONS

Software optimizations are constantly needed to provide overall improvements in a technology company’s product. For the gemineers’ services, these enhancements are also necessary, especially for the Data Ground applications, which strongly rely on computer performance due to the amount of processes performed. Taking this into consideration, two optimizations are proposed and developed in this chapter. First, Section 5.1 presents a change in the general structure of how information is handled within the Data Ground area, moving to working with batches of data instead of single points in an attempt to improve CPU usage, as discussed in Chapter 4. Then, Section 5.2 addresses the optimal point for closing a batch operation in the Batcher, balancing time taken by the computer and the compression rate of the data in the zip folders.

5.1 DATA GROUND IN BATCHES

Knowing the excessive use of computational resources by the Data Ground applications, previously introduced, which work through singular data points, a grouped data acquisition is devised in order to solve such a problem. The idea is to maintain the high frequency acquisition of the Heidenhain DAQ in the same way, but gathering the obtained mechanical data in batches to be sent together and less often to the other services of the area in question. Thereby, the amount of operations performed in a short period of time by the Data Acquisition software (especially by the HF Dispatch thread), the DGServer and the Batcher can be drastically reduced, and as a consequence, also the computer processor usage.

Considering the proposed optimization in the data handling for the entire Data Ground, it is therefore necessary to make small adjustments in the way the related applications work and route the information. Thus, Subsection 5.1.1 presents the implementation of the changes applied to the software based on pseudo-code snippets, and at the end, in Subsection 5.1.2, the new results are exposed and compared with those shown in the previous chapter.

5.1.1 Implementation to Data Ground Services

To properly start developing the batch data acquisition itself, it is first needed to define how the high frequency information should be grouped and dispatched from the DAQ to the other services in the process chain. Hence, two main possibilities are considered: use an array of specific programming objects created, which already contain all necessary properties within the instances; or apply and assemble a group of separate vectors, where each is responsible for sending information regarding a single attribute of a data type. Both alternatives are better illustrated in Figure 33 through a coordinates example. On the left, the list filled with the corresponding programming objects is identified, while on the right, the individual elements are put together and presented in different arrays that store their results.

Figure 33 – Different ways to handle batch data in the Data Ground services.

Programming objects array	Individual attribute arrays
`{ coordinates: [<programming objects>] }`	`{ timestamp: [<double>], x: [<double>], y: [<double>], z: [<double>], b: [<double>], c: [<double>] }`

Source: Author.

Although both options in Figure 33 have their respective pros and cons in the development of the batch dispatching, the selected alternative for the implementation in the Data Ground is the one that bundles the constructed programming objects into a single array (left side). This selection is due to the fact that the association of the attributes in a unique structure is required anyway in the current process workflow, and is more efficient and faster executed on the DAQ side than in the DGServer or the Batcher (right option in Figure 33). In addition, even though the second possibility mentioned presents prospects for performance improvements in the Batching Service, concerning appending information to a buffer, the overall organization of this application would need major changes to assimilate this data format. Consequently, the latter option is discarded, as the end result would be unfeasible to achieve during the development time of the current project.

In order to develop the former idea, of collecting programming objects in an array, within the Heidenhain DAQ software, these changes have to be applied to one of the high frequency threads, since they are the goals of the proposed optimization. Considering that the HF Acquisition must be solely responsible for getting the data from the controller, aiming to maintain the previously attained frequency of 330 Hz, the HF Dispatch needs to be the host of the respective alterations. In the "Dispatch Acquired Data" state of the dispatcher thread, the initial steps are still followed, reading the data points of each acquisition iteration from the common buffer, whose size has been increased for this application, to properly create the objects for the corresponding types. Thereafter, the minor changes with respect to dispatching information occur and are visually displayed in Code Snippet 10.

Code Snippet 10 – Example of a general batch data function.

```
void HandleBatchData()
{
    Create programming object;
    Append it to an array;

    if (array reaches max size)
      DispatchBatchData();
}
```

In the case just shown, instead of the objects being sent directly to the server one at a time by the existing dispatch functions, as formerly seen, they are now temporarily accumulated in an array that is related to their data type. Once the size of the vector reaches a stipulated maximum value, all the information acquired so far is dispatched to the DGServer together in a batch form. This sending procedure is performed using new dispatch methods created especially for this purpose. Moreover, to avoid increasing the memory usage of the computer, as soon as the information is sent, the vector is emptied, freeing space for the next batch.

The Code Snippet 10 is also executed multiple times in a loop. Despite this behavior, the new implementation is expected to show overall improvements in CPU usage, since in most iterations, only appending operations to an array are required. Furthermore, functions that really demand processor utilization, such as the dispatch ones, although they now need more CPU to handle several data at once, are called considerably less often than the previous model, triggering the other services in the chain fewer times and hence accounting for the achievement of the proposed goal.

The *DispatchBatchData*, for example, from Code Snippet 10 represents one routine from this new group of functions that are designed to transform the programming object data array into understandable messages for the DGServer. These messages consist, again, of strings that simulate mutations within the query language, necessary to properly forward information acquired by the Heidenhain DAQ over the network. The development of the discussed function is presented in Code Snippet 11 in order to better explain the respective operations performed.

Code Snippet 11 – Example of a general dispatch data function.

```
void DispatchBatchData() {
    ConstructMutationMessage();
    SendMessage();
}
```

The message construction of the two mentioned implementations is remarkably similar. The main difference concerns its manipulation, which in the occasion just shown happens inside the *ConstructMutationMessage* pseudo function. The dispatch of the message built with the data is actually executed at the end (in the *SendMessage* pseudo routine), where the communication with the server through the selected protocol is performed with the help of the used library for this purpose.

For these batch data to properly arrive at the DGServer application, the corresponding schema, mutations and subscriptions need to be adjusted in order to handle the acquisition and sending of information in an array form. As soon as the specific alterations are performed, the Batcher does not need major modifications for being able to acquire the transient information from within the server. The only change required is in calling the subscriptions inside the Batching Service with the appropriate new names for each data type of interest.

Once all main Data Ground applications have been properly updated to work with groups of data, tests must be performed to visualize their functionality and performance against the findings formerly analyzed in Chapter 4. Such a topic is introduced in more detail in the next subsection.

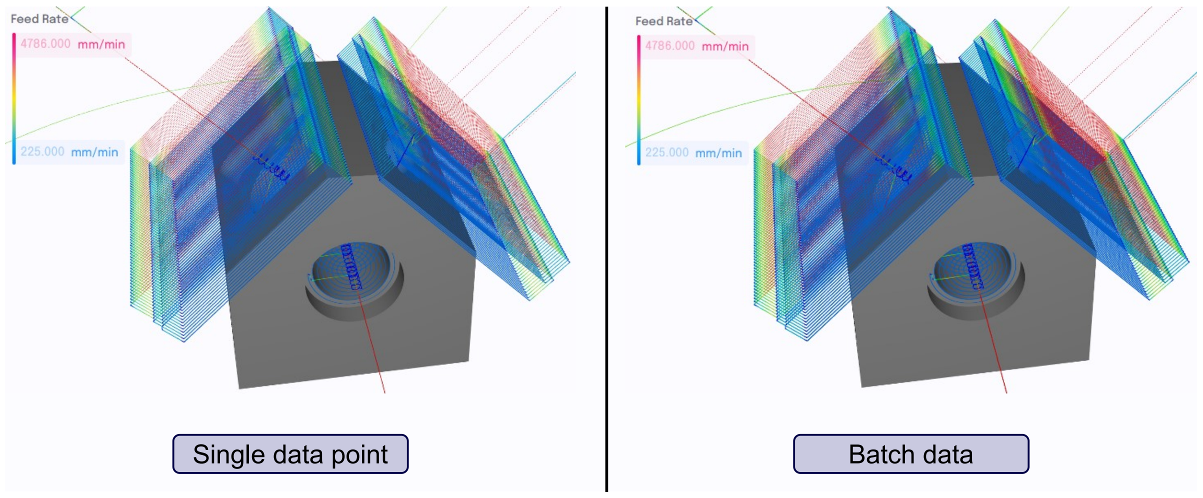
5.1.2 Optimization Tests and Results

To accurately and fairly compare the results of the batch data with the acquisition of a single data point at a time, the testing phase for the developed optimization is carried out identically to that presented in the previous chapter. Therefore, the Programming Station and the improved Data Ground services are first started in the computer to perform their respective tasks in an acquisition process. For the controller emulator software, the configurations applied are equivalent to those earlier selected for both the kinematics of the virtual machine tool (Figure 25) and the machining program to be simulated, "house.H" (Figure 24), which transforms a rectangular block into a birdhouse.

The implemented improvements do not change the overall flow and working principle of the entire software. For this reason, while the manufacturing simulation is running, the acquisition operations continue to execute in the background, even though they are now retrieving the desired data in batches. Nevertheless, the grouped information, after going through the Batcher, is also converted into individual files for each data type, just like shown in Figure 26, thus allowing the Data Processing services to correctly handle the information until its output is displayed in the Front-End.

In view of this and taking into account a complete execution of the gemineers' applications with the developed changes, the right side of Figure 34 shows the final result in the Front-End for a dispatch of two hundred data in a batch. Next to it, on the left side of Figure 34, the result of a single data point sending is presented for further comparison. Both images feature the geometry of the birdhouse in a 3D environment together with the Feed Rate signal in the form of a colored point cloud and its legend.

Figure 34 – Result comparison of single and batch data dispatching in the Front-End.



Source: Author.

From Figure 34, it can be inferred that the functionality of the Data Ground, and consequently the software as a whole, for batch data was a success, since the final results are very similar to what was expected and what was achieved without any optimization. Besides presenting no perceptible loss of information, it is also remarkable that all the features displayed by the model on the left are found in the right one, from the complete development of the 3D digital twin to the respective mapping of the Feed Rate values colored by the mesh of points in space, properly demonstrating the movements taken by the cutting tool.

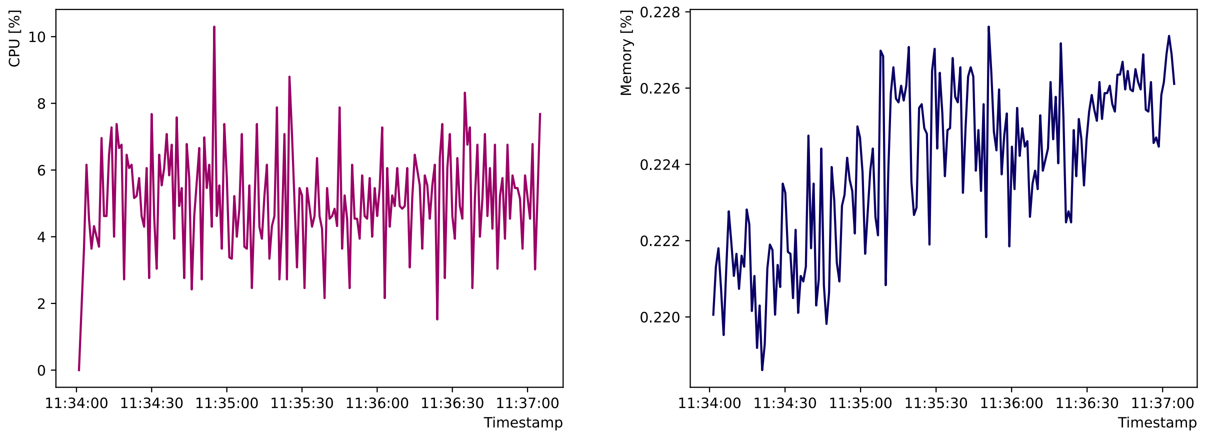
It is worth reiterating that this discussed result was reached by dispatching two hundred simultaneous data points, although any amount set by the developer would emerge in minimal visual alterations from what was shown. This is mainly due to the fact that the data acquisition itself still remains at the same attained frequency of 330 Hz, since the HF Acquisition thread has not been changed for the current optimization. The only modifications employed refer to the dispatch processes, which basically control when the obtained information is actually sent to the server.

Despite the significant achievements regarding the functioning of the applications, the main changes implemented aimed at improvements with respect to the computer-wise performance of the Data Ground's services. For this reason, the testing procedures conducted in Section 4.3 are applied to this optimization case as well, in order to verify the new results and compare them with those acquired before the development of the grouped data.

Therefore, as in the previously performed tests, an existing python script is responsible for recording information regarding the computer's CPU and memory usage, for each application of interest, but now during the batch data acquisition processes of the house program simulation. These computational data obtained from this strategy are again displayed in graphs in the following figures of the current subsection to provide a better visualization of the resources utilization. Moreover, the computer used to carry out these activities remains the same as in the previous model in an effort to keep comparisons concise and honest.

In light of this, the three optimized Data Ground services are discussed next in their corresponding order of execution, going from the Data Acquisition to the DGServer, and finally to the Batcher, all considering batches of two hundred data points as in the preceding example. First, Figure 35 presents the performance of the enhanced Heidenhain DAQ during the execution of the gemineers' software, where the graph on the left shows CPU usage and the one on the right illustrates the memory. Both values are represented in percentage of the respective total amount available in the computer.

Figure 35 – Heidenhain DAQ performance for CPU and memory with two hundred batch data.



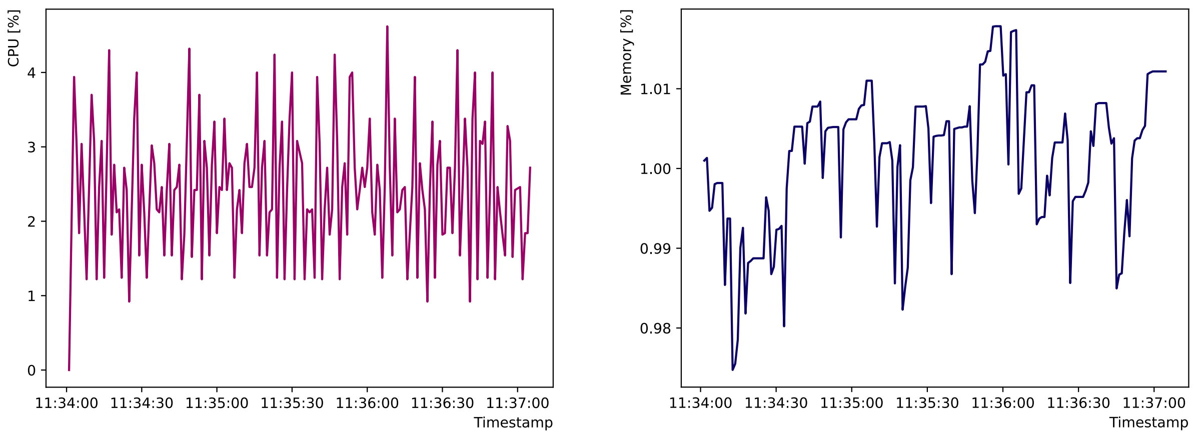
Source: Author.

Given the results in Figure 35, two conclusions can be drawn when compared with those previously obtained in Figure 30: the considerable decrease in processor usage (from an approximate average of 18% to 5.5%); and the increase in memory utilization (growing from 0.147% to around an averaged value of 0.224%). This inverse correlation is noticeable in a logical way from the implementation of the data sending in a batch format.

As for the reduction in CPU usage, this is due to the fact that fewer processes are now carried out by the Heidenhain DAQ within a short period of time. This happens since the computationally expensive high frequency dispatch routines are executed just a few times during the program flow, while mostly only appending operations to an array are performed, as seen in Code Snippet 10. For this case of batches of two hundred data points, for example, just one dispatch method is realized in the period that two hundred append tasks are done. The increase in memory, in turn, occurs due to the same procedures mentioned, in which batch information is temporarily stored in different arrays for each data type, thus expanding the amount required to handle all these data together inside the application.

The behavior observed on these computer properties for the Heidenhain DAQ is also expected to appear for the other services in the Data Ground area, since they work as a chain, where the next applications are only triggered to run when new data is sent by the Data Acquisition. To further analyze this reaction, the computer-wise performance for the DGServer and the Batcher are shown in Figure 36 and Figure 37, respectively. The layout of the graphs in both figures is the same as those already presented, with the CPU information on the left side and the memory on the right one.

Figure 36 – DGServer performance for CPU and memory with two hundred batch data.

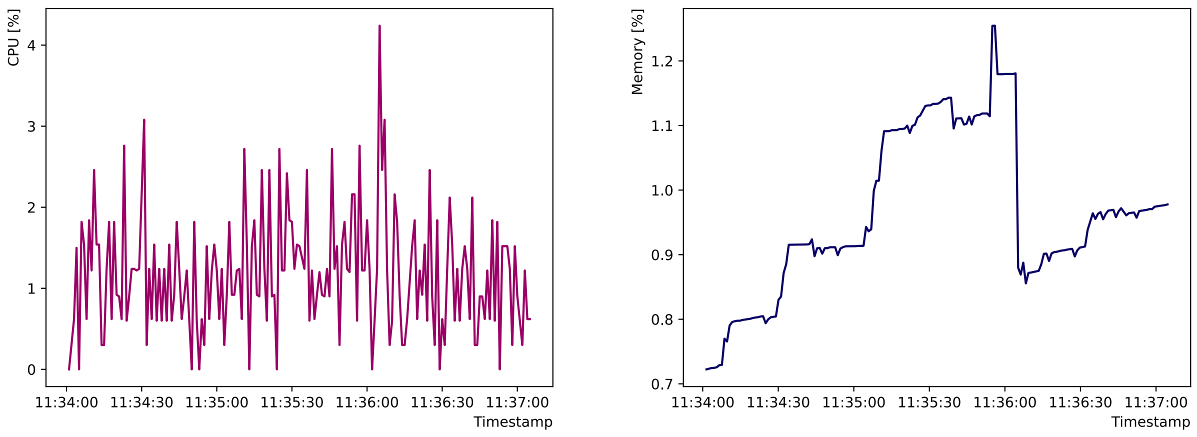


Source: Author.

Bearing in mind the results for the DGServer for batches of two hundred data, seen in Figure 36, it can be inferred that CPU usage has improved substantially compared to the previous version, indicated in Figure 31, as the average went from 10% to 2.5%. When it comes to memory, the pattern is as expected, with a slight increase from 0.95% to close to 1%. The explanations for such an event again rely on the information being dispatched to the server less frequently, triggering the respective mutation and subscription functions of the DGServer fewer times. However, since the transient data is now handled in vectors as well, the memory utilization tends to rise in order to meet the demands of this service.

Regarding the optimized Batcher instance, whose performance for batches of two hundred data is visually presented in Figure 37, the CPU also decreased in usage when compared to the one analyzed in Figure 32, going from around 3.5% to an average of about 1%. Memory, meanwhile, showed a similar behavior to the normal acquisition processes, although a mild increase is noticeable in its overall utilization.

Figure 37 – Batcher performance for CPU and memory with two hundred batch data.



Source: Author.

Although the attained results in the last figures take into account two hundred data points at a time, the maximum batch size can be set to any desired number, in order to produce different values for CPU and memory used. In Appendix A, graphs demonstrating the results obtained for batches of twenty and five hundred points are shown for a richer analysis of how the developed optimization works.

Analyzing both examples, it is conclusive that the smaller the batch sizes, the more CPU and less memory is used, while for larger batches, the opposite is true. This again follows the logical relationship described between the amount of data temporarily stored in the applications and the quantity of operations performed by them. From this, it can be deduced that the closer the arrays are to a single data, the more similar the performance values will be to those of the previous model exposed in Chapter 4. Furthermore, it is also worth noting that for very large batch proportions, as is the case of five hundred, the computer-wise performance tends to be noisier. This is due to the fact that these dispatching routines and the functions triggered by them end up demanding much more memory and processor usage for such a lot of points, expressing an even greater difference to simple append operations.

Finally, it can be concluded that the proposed optimization was successful. First, it was clear that the developed solution did not provide any noticeable loss of features or data, as discussed with Figure 34, thus maintaining the previous functionality and once more achieving the requirements of the project as a whole. Regarding computer performance, despite the fact that the total amount of memory utilization has increased slightly with the batch data, it is insignificant when compared to the reduction in processor usage attained by the three main Data Ground services together, which was the main goal of the implementation. Adding up the computer processing time of the new Heidenhain DAQ, DGServer and Batcher results in a final value of approximately 9%, showing the considerable difference from the 31.5% previously obtained, and consequently the success achieved by the batch data optimization.

Even though the batch data presents significant improvements in the overall acquisition process, its main optimizations do not affect the batch closing on the Batching Service side. In order to implement changes also to the end operations of the entire Data Ground area, the following section discusses the possibility of enhancing such procedures considering a balance between their elapsed time and the compression ratio of the final zip folders.

5.2 BALANCED CLOSING OF BATCHER FILES

Upon the signal coming from the metadata, already discussed in Chapter 2, the Batcher must perform the respective operations to properly close a batch. These processes basically consist of converting the data stored in the computer's buffer into several files, and then assembling them into a zip folder. Considering this, it is desirable, for the Batching Service, these procedures to be performed as quickly as possible and with the maximum compression rate available.

Nevertheless, the two mentioned desires work in opposite ways, i.e. the higher the wanted compression, the more running time and computational costs are necessary by the application. Therefore, a balance point between the two objectives is required, where both can reach satisfactory values for the closing of a batch.

To address such a problem and adequately operate the zip folders, a zip library is applied in the service in question. Among its various functionalities, there is the possibility of manipulating and customizing the compression of the zips, enabling the selection between ten levels, where, in theory, the higher the level, the smaller the folder size is at the end, and hence the longer the time needed to run.

Taking this into account and the objective of balancing both variables, tests were carried out with the data acquired from the previously simulated house program. Table 2 presents the numerical results for the ten performed closing procedures, one for each available compression level. The second column of the table identifies the time in seconds taken to convert the files and their compression, while the third column gives the final folder size in KB.

Table 2 – Compression level and its respective closing time and final batch size.

Compression level	Closing time (s)	Zip size (KB)
0	1.906	68710
1	2.913	10719
2	2.991	10452
3	3.243	10369
4	3.472	8733
5	3.879	8298
6	4.418	8290
7	4.957	8283
8	7.275	8290
9	9.193	8293

Source: Author.

The current state of the gemineers service is at compression level 0 in Table 2, where the closing time is the fastest possible and there is no compression in the folder. In view of the other results and by making a simple qualitative selection of the data, the compression rate should be changed to level 1. This is due to the fact that the final size jump from level 0 to level 1 is much larger than for the other classifications, which remain at similar values even as the processing time increases. In addition, it is still desired to execute the code as fast as possible, but now effectively compressing the file, thus converging again to the optimal level 1.

By implementing this change in the Batcher, the closing processes take longer to happen, but now the zip folders are considerably smaller in size for an easier storage, actually balancing both requirements. This update does not affect the batch data from Section 5.1, and so the results previously shown remain the same. Once both optimizations have been correctly implemented to the respective Data Ground applications, the ideas proposed earlier can be considered developed and completed. Suggestions for further and substantial improvements are discussed in Chapter 6, where an overall project summary is also presented.

6 CONCLUSION AND FUTURE WORK

Facing the urge for a more efficient and high-quality production, manufacturing industries are currently pursuing technologies to digitize produced workpieces and their respective production processes. This approach, besides bringing more reliable results due to accuracy, serves to save time, money and rework for defective parts, representing considerable improvements over the conventional methods widely used today. To properly achieve such a goal, data acquisition software are required in order to obtain the necessary information to create the virtual piece. Considering this novel solution and the objectives of developing a digital twin of the workpiece, the project presented in this document, along with gemineers GmbH, aims to implement a data acquisition application for machine tools operated by Heidenhain controllers.

Based on studies conducted for the requirements definition and software modeling, the Heidenhain DAQ was implemented in a multi-threaded solution, where four different threads were developed to work together to interact to a machine tool and to a web server. By connecting to both, the application prepares a data stream to acquire information from the controller and directly dispatch to the server over the network, in low and high frequency, as expected by the objectives proposed in Chapter 1.

The DAQ testing procedure started with the configuration of an emulated Heidenhain controller to adequately simulate a machining process. Therefore, the activities of the developed software were triggered and the obtained data were sent to the other gemineers' services. From them, the information was processed and the acquisition results were properly presented in the existing Front-End in the form of digital twins.

Comparing the displayed virtual piece built on top of the machine data with the expected result from the simulated control, it can be concluded that the Heidenhain DAQ operations were all successful. The final 3D geometry mapped the correct cuts and holes previously defined by the selected machining process, creating an exact copy of the workpiece in a digital environment. The high frequency data, such as the Feed Rate, were successfully obtained, as it was illustrated point by point in a color palette around the resulting part, demonstrating the cutting path taken by the tool tip. Furthermore, the low frequency information was also acquired correctly, as the results showed the different types of machine data synchronized to each other in the space, as formerly envisioned by the requirements.

Bearing in mind such results, it is possible to infer that the main objectives and specifications discussed in the first chapters of this document have all been attained. Considering the digital twins displayed in the Front-End, the user is finally able to virtually monitor and inspect workpieces and their respective production processes executed on Heidenhain controllers, also giving an opening for future quality prediction. The Data Acquisition software together with the other microservices of the company provide gemineers with a path towards complete part digitization, which can in the next steps replace conventional technologies, saving resources and reaching better efficiency for production lines.

At the end, performance tests were applied to analyze the behavior of the services related to data acquisition. Taking into account the unsatisfactory results reached, optimizations were proposed and directly implemented in order to attain a more reliable software. Changing the structure on how information was handled within the Data Ground applications from single to batch data provided a better computer-wise performance when compared to the previous solution. Despite the significant and successful outcomes, there is still room for improvements on the Heidenhain DAQ, as further introduced in the next section.

6.1 FUTURE WORK

For a simplified and complete use of the software in production, some activities remain to be performed and features implemented. First, deeper studies of the used libraries are needed in an attempt to obtain information from the controller for different types, such as those only present in Siemens DAQ.

Moreover, as the developed Data Acquisition was implemented and tested for only one type of controller, the iTNC 530, further tests and possible modifications are necessary in order to generalize the application to different controls, such as the TNC 640, for instance. The differentiation of controllers is a common problem with Heidenhain machines, due to the fact that for distinct models, the data output is not identical, nor is the way the connections work and are handled within the machine tool.

Lastly, an automated selection of channels from the Scope Information area may present possible improvements for the development of the Heidenhain DAQ. A feasible implementation could be realized in a way that the scope properties (as shown in Chapter 3) would be set in the existing "config" file, without the need for the font code to be changed. The internal operations, after reading the passed values, would perform the appropriate communication with the machine tool for the desired variable, allowing a simple data querying that can be provided by any user.

In light of all that has been presented in this document for the development of the project, the author's personal observations about the impact of the work on the student and the correlation with the Control and Automation Engineering course are discussed in the following section.

6.2 PERSONAL OBSERVATIONS

At the end of the project in question and in view of the time spent as an intern at gemineers GmbH, the student can realize the high impact on professional life that this opportunity has provided. The acquired knowledge was not only in the technical area but also in the inter-personal relations.

In terms of the practical experiences, it can be inferred that new technologies were learned by the author during the internship. For the mechanical part, it was possible to closely analyze and understand how the manufacturing processes are carried out on the shop floor. With the application of high quality CNCs and milling techniques only previously discussed in a theoretical way, the intern was also able to realize how scientific research really works, especially in the field of machining. On the subject of programming, additional languages and overall technologies could be discovered, opening the range of the student's specific knowledge in the area of computer science. Regarding programming languages already known, the skills in relation to them have been enhanced by implementing more complex software solutions different from those the author had worked on before.

When it comes to social skills, the intern was able to understand the processes of a fully functioning company composed of diverse workers. During the time at gemineers, teamwork was experienced day after day, using the agile SCRUM framework for an organized and piecemeal development of the software as a whole. Furthermore, it is worth mentioning the importance of working in an international team, where the official language is English, providing the necessary experience for a fluid communication with people and companies around the world.

Moreover, as the Control and Automation Engineering is directly correlated to projects involving technology advancements, just like the gemineers product, it can be seen throughout the work the practical application of various subjects learned during the college years. The software development in its essence is strongly related to the computer science part and programming disciplines of the course that led the student to get the work done. Subjects such as Software Development Methodology were useful as well for a complete understanding of the processes and requirements that were applied in UML diagrams for further implementation. For the network communication between services, disciplines such as Computer Networks were helpful to establish connections, keeping the separate steps developed working as one. In terms of machining process to which the data acquisition is related, lessons from Manufacturing Automation Introduction were important to understand the mechanical side of the project.

Despite all the help related to these courses, one suggestion for Control and Automation Engineering is to apply more programming subjects or more in-depth disciplines in its plan, where different languages are covered and the computational theory is explained, since the job market for developers is considerably increasing recently.

REFERENCES

1 ROLLS-ROYCE. **Rolls-Royce ships 10,000th blisk from Oberursel**. Last access: 02/07/2022. 2019. Available from: https://www.rolls-royce.com/media/press-releases/2019/11-04-2019-ships-10000-blisk-from-oberursel.aspx.

2 HEXAGON. **Non-Contact Measurement of Blisk Aerofoils and Gas Path Features: HTA Solution for Measurement of Aeroengine Components**. Last access: 02/07/2022. 2022. Available from: https://www.hexagonmi.com/solutions/applications/blisks/non-contact-measurement-of-blisk-aerofoils-and-gas-path-features.

3 BARON MACHINE COMPANY. **The Evolution of CNC Machining**. Last access: 29/07/2022. Available from: https://baronmachine.com/news/the-evolution-of-cnc-machining/.

4 CALLEJA, A.; GONZÁLEZ, H.; POLVOROSA, R.; GÓMEZ, G.; AYESTA, I.; BARTON, M.; LACALLE, L. LópezN. de. Blisk blades manufacturing technologies analysis. **Procedia Manufacturing**, v. 41, p. 714–722, 2019. ISSN 23519789. DOI: 10.1016/j.promfg.2019.09.062.

5 ERICSSON. **A case study on real-time control in manufacturing**. Stockholm: Ericsson, 2018.

6 FRAUNHOFER IPT. **Project "Quantify": Data-driven prediction of component quality to increase profitability and digital resilience of metal-cutting manufacturing systems**. Last access: 02/07/2022. Available from: https://www.ipt.fraunhofer.de/en/projects/quantify.html.

7 FRAUNHOFER IPT. **High Performance Cutting: Modern Machining in the Digital Age**. Last access: 02/07/2022. Available from: https://www.ipt.fraunhofer.de/en/Competencies/processtechnology/Highperformancecutting.html.

8 FRAUNHOFER IPT. **The Digital Twin in Turbomachinery Manufacturing**. Last access: 02/07/2022. Available from: https://digitaltwininturbomachinerymanufacturing.dashboards.vfk.ipt.fraunhofer.de/.

9 GEMINEERS. **The smartest quality assurance since the beginning of machining- already in manufacturing**. Last access: 02/07/2022. Available from: `https://en.gemineers.com/`.

10 ANITA ADROIT. **Global Computer Numerical Control (CNC) System Market 2022-2028 by Major Players: Fanuc, Siemens, Mitsubishi Electric, Bosch Rexroth, Heidenhain, etc**. . .. Last access: 28/07/2022. 2022. Available from: `https://thepost.nz/2022/07/26/global-computer-numerical-control-cnc-system-market-2022-2028-by-major-players-fanuc-siemens-mitsubishi-electric-bosch-rexroth-heidenhain-etc/`.

11 IBM. **How does a digital twin work?** Last access: 02/09/2022. Available from: `https://www.ibm.com/topics/what-is-a-digital-twin`.

12 TWI. **What is Digital Twin Technology and How Does it Work?** Last access: 02/09/2022. Available from: `https://www.twi-global.com/technical-knowledge/faqs/what-is-digital-twin`.

13 SIEMENS. **Service Lifecycle Management**. Last access: 04/09/2022. Available from: `https://www.plm.automation.siemens.com/global/en/products/collaboration/service-management.html`.

14 IBM. **Siemens and IBM Deliver Service Lifecycle Management Solution**. Last access: 04/09/2022. 2020. Available from: `https://newsroom.ibm.com/2020-06-17-Siemens-and-IBM-Deliver-Service-Lifecycle-Management-Solution`.

15 AASHISH MEHRA. **Digital Twin Market by Enterprise, Application (Predictive Maintenance, Business optimization), Industry (Aerospace, Automotive & Transportation, Healthcare, Infrastructure, Energy & Utilities) and Geography - Global Forecast to 2027**. Last access: 04/09/2022. 2022. Available from: `https://www.marketsandmarkets.com/PressReleases/digital-twin.asp`.

16 GOOGLE CLOUD. **What is microservices architecture?** Last access: 04/09/2022. Available from: `https://cloud.google.com/learn/what-is-microservices-architecture`.

17 CHRIS RICHARDSON. **What are microservices?** Last access: 04/09/2022. Available from: `https://microservices.io/`.

18 FRANK E. GRUBBS. Procedures for Detecting Outlying Observations in Samples. v. 11, n. 1, p. 1–21, 1969. DOI: 10.1080/00401706.1969.10490657. Available from: https://www.tandfonline.com/doi/abs/10.1080/00401706.1969.10490657.

19 PENG ZHANG. **CHAPTER 7 - Industrial intelligent controllers**. Ed. by Peng Zhang. Oxford: William Andrew Publishing, 2010. P. 257–305. ISBN 978-1-4377-7807-6. DOI: 10.1016/B978-1-4377-7807-6.10007-5. Available from: https://www.sciencedirect.com/science/article/pii/B9781437778076100075.

20 DASSAULT SYSTÈMES. **CNC machining: Discover what is the manufacturing process CNC Machining and its usages in the industry.** Last access: 11/09/2022. Available from: https://www.3ds.com/make/guide/process/cnc-machining.

21 JIGA. **Ultimate Guide to CNC Machining**. Last access: 11/09/2022. 2021. Available from: https://jiga.io/resource-center/cnc-machining/what-is-cnc-machining-guide/.

22 ANDREAS VELLING. **What Is CNC Machining? Working Principles, Capabilities & More**. Last access: 11/09/2022. 2020. Available from: https://fractory.com/what-is-cnc-machining/.

23 PENG ZHANG. **4 - Digital Controllers for Industrial Control**. Ed. by Peng Zhang. Norwich, NY: William Andrew Publishing, 2008. P. 429–568. ISBN 978-0-8155-1571-5. DOI: 10.1016/B978-081551571-5.50005-0. Available from: https://www.sciencedirect.com/science/article/pii/B9780815515715500050.

24 SPRINGER. CNC Architecture Design. In: THEORY and Design of CNC Systems. London: Springer London, 2008. P. 315–352. ISBN 978-1-84800-336-1. DOI: 10.1007/978-1-84800-336-1_9.

25 PENG ZHANG. **CHAPTER 2 - Industrial control engineering**. Ed. by Peng Zhang. Oxford: William Andrew Publishing, 2010. P. 41–70. ISBN 978-1-4377-7807-6. DOI: 10.1016/B978-1-4377-7807-6.10002-6. Available from: https://www.sciencedirect.com/science/article/pii/B9781437778076100026.

26 DR. JOHANNES HEIDENHAIN GMBH. **Programming stations for lathe and milling controls**. Last access: 13/09/2022. Available from: `https://www.heidenhain.com/products/cnc-controls/programming-stations`.

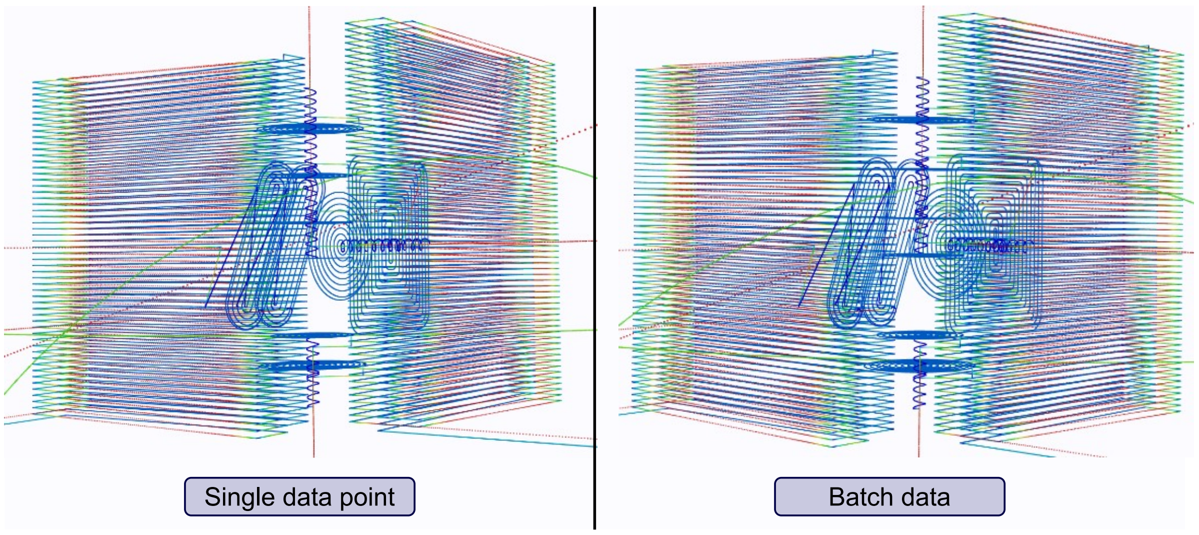
27 DR. JOHANNES HEIDENHAIN GMBH. **TNCremo**. Last access: 13/09/2022. Available from: `https://www.klartext-portal.de/pc-software/tncremo/#`.

28 RAPID DIRECT. **Difference Between Feed Rate and Cutting Speed in CNC Machining**. Last access: 27/06/2022. 2021. Available from: `https://www.rapiddirect.com/blog/difference-between-feed-rate-and-cutting-speed/`.

APPENDIX A – ADDITIONAL RESULTS FOR BATCH DATA OPTIMIZATION

A.1 FUNCTIONALITY COMPARISON IN THE FRONT-END

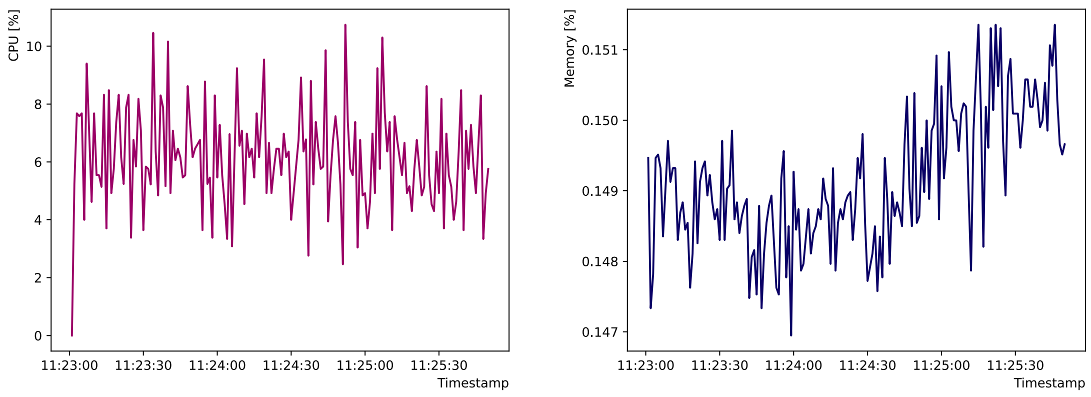
Figure 38 – Top view comparison of single and batch data dispatching in the Front-End for Feed Rate signal.



Source: Author.

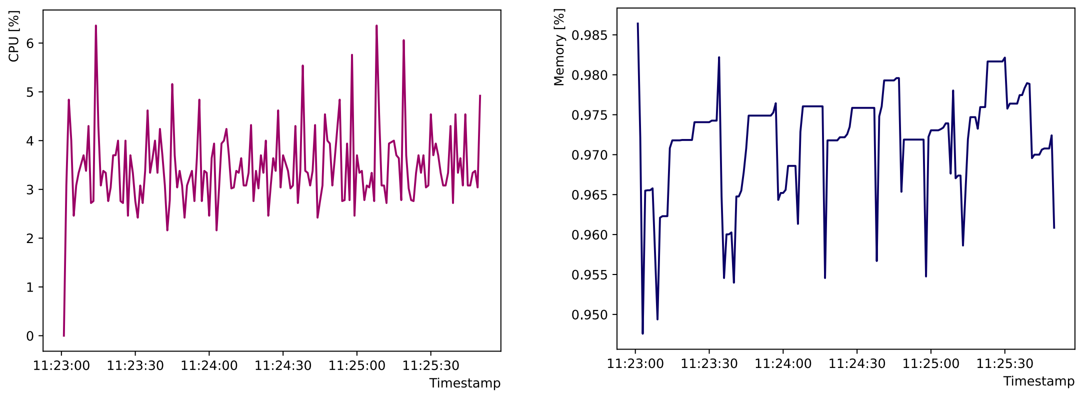
A.2 OPTIMIZED SERVICE PERFORMANCE

Figure 39 – Heidenhain DAQ performance for CPU and memory with twenty batch data.



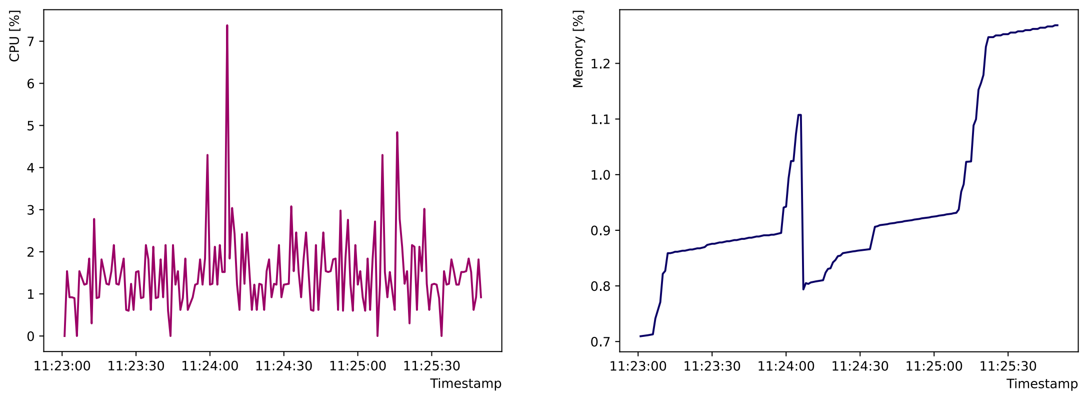
Source: Author.

Figure 40 – DGServer performance for CPU and memory with twenty batch data.



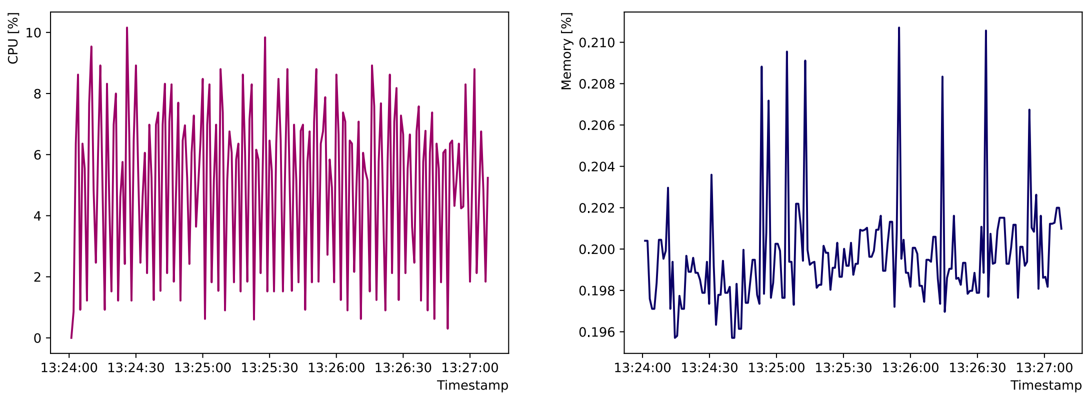
Source: Author.

Figure 41 – Batcher performance for CPU and memory with twenty batch data.



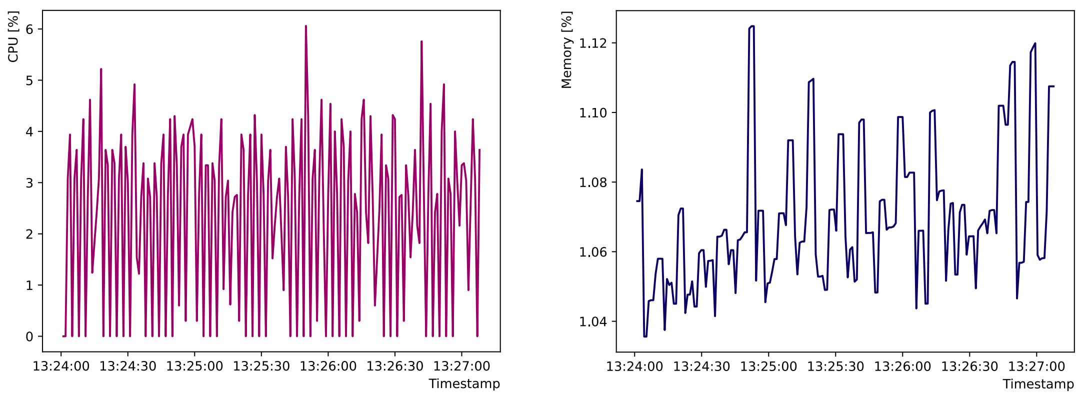
Source: Author.

Figure 42 – Heidenhain DAQ performance for CPU and memory with five hundred batch data.



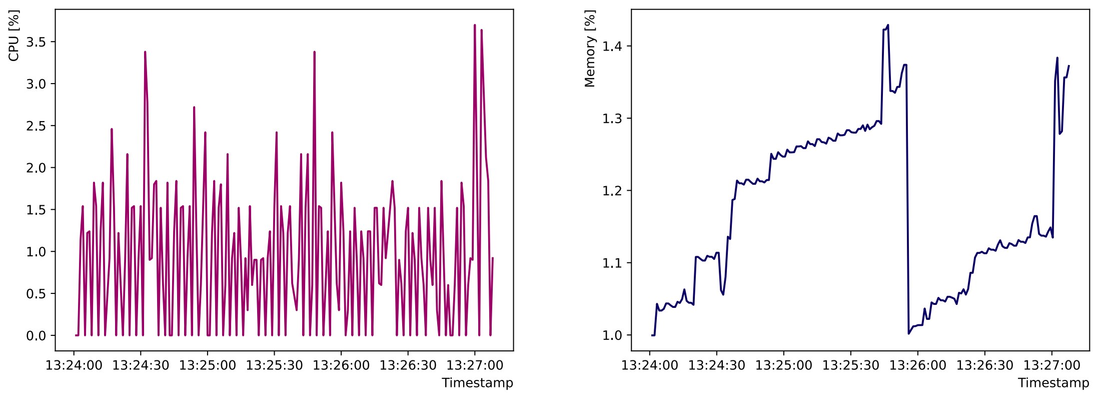
Source: Author.

Figure 43 – DGServer performance for CPU and memory with five hundred batch data.



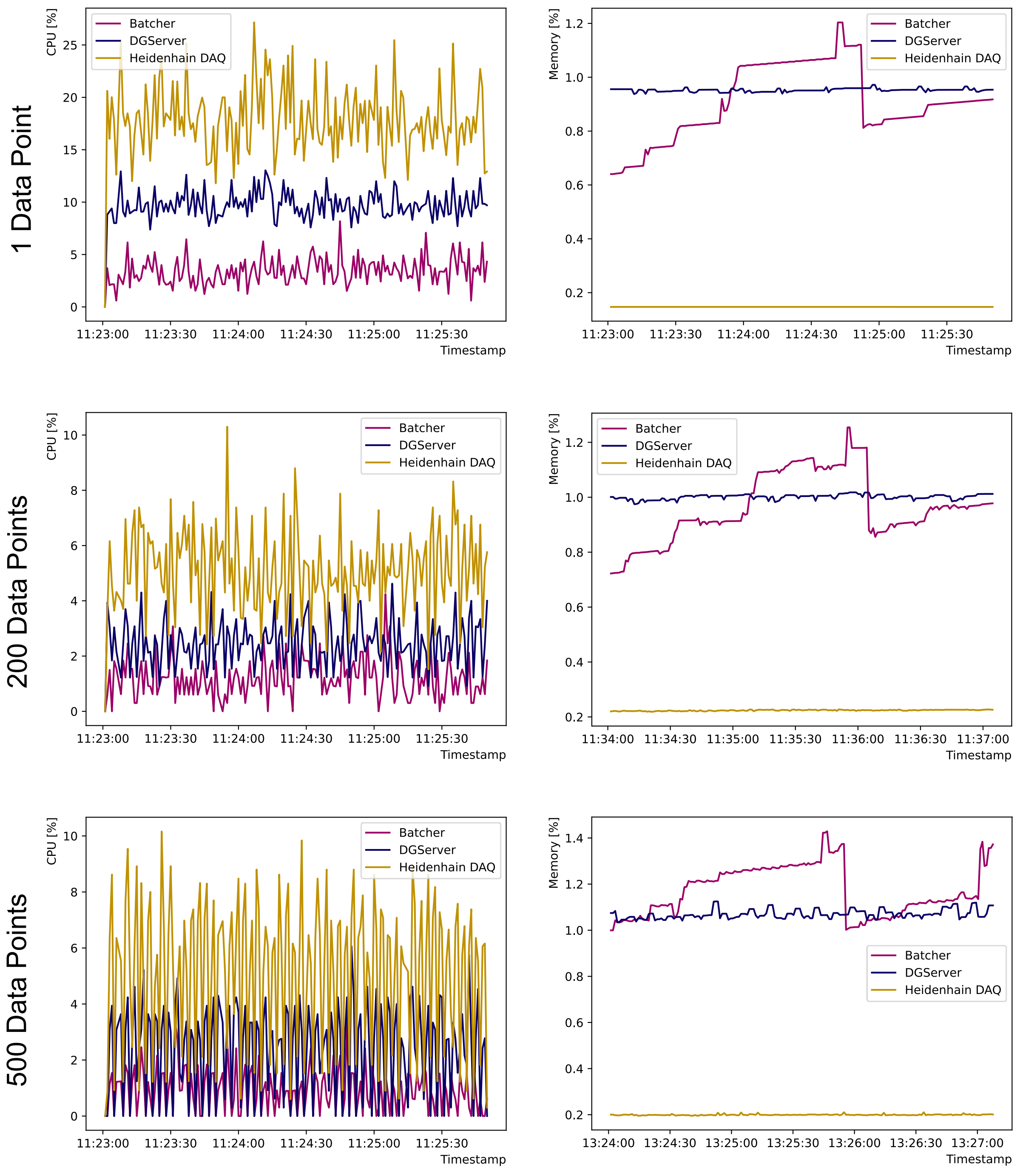
Source: Author.

Figure 44 – Batcher performance for CPU and memory with five hundred batch data.



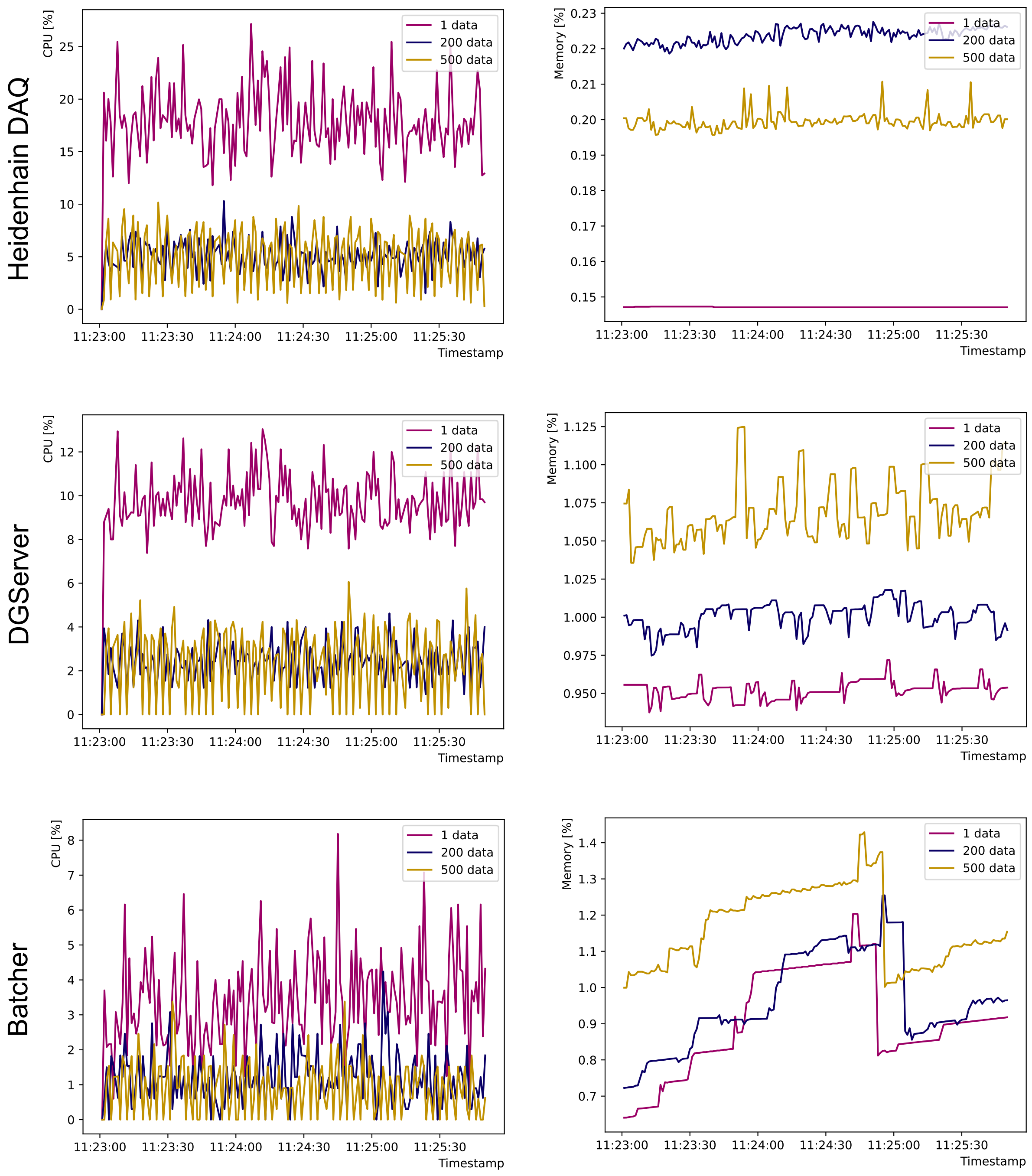
Source: Author.

Figure 45 – Performance comparison of Data Ground services according to the batch size.



Source: Author.

Figure 46 – Performance comparison of batch size according to the Data Ground services.



Source: Author.