Rheinisch-Westfälische Technische Hochschule Aachen
Knowledge-based Systems Group
Prof. Gerhard Lakemeyer, Ph.D

Master's Thesis

DEVELOPMENT OF AN ANALYTICS INFRASTRUCTURE FOR MILLING PROCESSES

Muhammad Mehmood Ahmed
Matr.-No. 403372

Supervisors:
Prof. Gerhard Lakemeyer, Ph.D
Prof. Dr.-Ing. Christian Brecher

Advisors:
Tarik Viehmann, M.Sc.
Janis Ochel, M.Sc. M.Sc.

14th February 2023

Contents

List of Tables

List of Figures

1 Introduction

In modern manufacturing environments, huge volumes of data are collected in database systems and warehouses from machine sensors, numerical control (NC), product and process design, materials planning, quality control and scheduling [21]. Optimizing efficiency of production processes drives the need for their automated preprocessing and analysis [12]. Yet, existing analytics tools in manufacturing are coined by major shortcomings considerably limiting continuous process improvement. In particular, they do not make use of data mining to identify hidden patterns in manufacturing-related data [33].

As process-related decisions and cause-effect relationships become increasingly complex, there is a growing demand for decision-making solutions based on Artificial Intelligence (AI) [11]. However, the mere number of AI approaches demonstrates that there is no single approach that guarantees optimal production conditions [63]. In fact, in order to solve each specific problem, such as surface quality prediction or tool wear detection in milling processes, these systems require a large amount of distinct processed data.

Milling is the process of machining using rotary cutters to remove material by advancing a cutter into a workpiece [33]. The tool can be moved in varying directions, along multiple axes, with different spindle speeds and forces in order to mill different features on the workpiece. This results in diversified subsections characterized by a particular combination of homogenous signals. Figure 1.1 shows an example workpiece and the current data of the linear positioning axes of the machine. It can be seen that milling of a circular pocket is represented by a single repeating pattern in current signals, which are different from, for instance, a rectangular pocket. The concatenation of such subsections make each process unique, and thus difficult to analyze manually.

Therefore, the identification and differentiation of similar process partitions is becoming more and more important. This supports the interpretation of the generated data by a human expert, and at the same time, facilitates in finding customized data in growing data lakes for AI applications. For example, a manufacturing company wants to analyze the feed rate changes during all of the circular pocket millings in their processes. This would necessitate data pre-processing in a way that recurring patterns in the raw data are identified, clustered and at the same time tagged with meta-information such as machine information, tool data and process features.

Hence, the aim of this thesis is to break up the inflexible structure of a coherent manufacturing process and to facilitate the local differentiation of milling features. For this purpose, data mining algorithms for clustering and classification are developed and combined with existing approaches in order to identify and characterize meaningful subsections in selected signals. For clustering of the time series data, two types of approaches are common in the literature [3]: Shape-based and

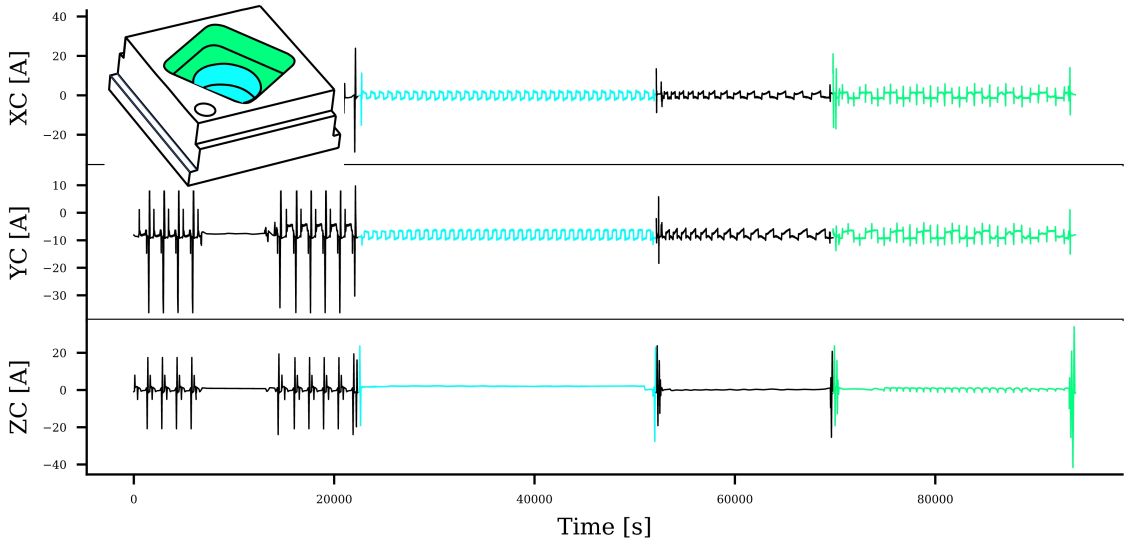

Figure 1.1: An excerpt of the positional current signals (XC, YC, ZC) in Amperes, read during the milling process of the sample workpiece. The highlighted segment in cyan corresponds to the circular pocket whereas the rectangular movement of the spindle resulted in the green segment.

Feature-based. However, it still remains an open question that which one is more suited for time series signals. The thesis aims to compare the two approaches and provide a suitable clustering model. The models and the algorithms are executed on an infrastructure, which consist of an edge device on the shop floor and a database storing the results. A methodology is developed enabling either an expert or future AI-solutions to flexibly query this database in reasonable time and combine the results in a customizable multi-signal manner. The overall concept makes it possible to obtain tailored data. Reducing data complexity, a milling process dataset becomes more comprehensible and can be exploited to optimize the production productivity. The overall approach is exemplarily implemented and experimentally validated under the milling process scenario.

2 Background

This chapter explains the concept of milling process and the data obtained from these processes. Additionally, some important algorithms are explained that are later used in the thesis. This involves an explanation of the matrix profile data structure, a segmentation approach by Ochel et al. [69], a representative discovery approach called snippets and a distance metric MPdist. The explained algorithms are compared and evaluated with the state-of-the-art from the thesis' perspective in Chapter 3.

2.1 Milling Process

Milling is a process performed with a machine in which a tool rotates to remove material from the workpiece present in the direction of the angle with the tool axis (see Figure 2.1) [33]. The tool is moved along an axis by a set feedrate, where feedrate is the velocity of the tool (in [mm]/[min]) relative to the workpiece. This results in desired milling features (for instance, face milling) on the workpiece. In recent times, Computer Numerically Controlled (CNC) machine tools have been used to fully automate the milling process [70]. In CNC machines, the control unit contains a dedicated computer which uses the data provided in the NC program to control the machine tool. Figure 2.2 shows one such machine called Heller HF3500 [38]. The complete program to produce different features on the workpiece is input to the computer and the computer can access machine sensor data for calculation of tool/workpiece movements. For instance, the tools positional data is necessary for the CNC to know how far and how fast the tool needs to move the machine to get to a specified position. [94]

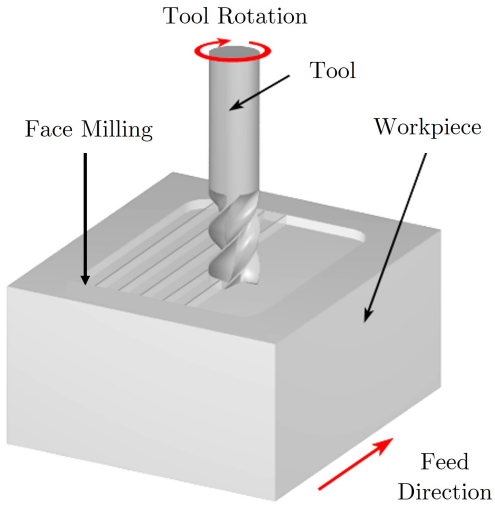

Figure 2.1: Illustration of a milling process.

Figure 2.2: Machine tool HF3500 from Heller [38]

Modern machine tools are capable of generating data up to one terabyte per hour. The data, being recorded at such high resolutions is no longer workable by human operators. However, it holds immense insights into the workpiece quality, tool wear, machine maintenance prediction etc. [8] The typical signals by the machine's NC control during a running process are of the following two types:

Continuously processed signals: Such signals are constantly read by the NC control. These signals include [96]:

- Positional signals of linear and rotary axes ([mm]/[rad])
- Linear and rotary drive currents of the positional axes [A]
- Drive current of the spindle [A]
- Spindle speed [rpm]

Non-continuous (event-based) signals: Such signals are only read on the occurrence of some particular events. For instance [100]:

- Tool changes
- Manual overrides of Feed rate and Spindle speed
- NC status change (running in Auto/Manual-mode, Standby, Error or Off)
- Change of NC programs

As the NCs can store only a limited amount of data on their hard drives [94, p. 11], these measurements are sent to an edge device via a suitable interface [15] like Siemens' Sinumerik [15], Open Platform Communications Unified Architecture (OPC UA) [15], etc. The data received at the edge device is stored in a database and, at the same time, plotted for analysis.

2.2 Matrix Profile

The matrix profile is a versatile tool in time series data mining, first introduced in 2016 [102]. The basic problem that it solves is *all-pairs-similarity-search* problem, resulting in motifs and discords discovery. However, it is a state-of-the-art data structure that will be used as the basis to explain many approaches in the later chapters.

The matrix profile consists of two primary components: a matrix profile vector which contains minimum z-normalized euclidean distances and a profile index indicating the location of the nearest neighbors. This is calculated by using a sliding window approach to extract all subsequences of a given length from the time series, and then calculating the distances between these subsequences using euclidean distance. For example, with a time series T of length n, to calculate the Matrix Profile, a subsequence length m is selected by the user, and all subsequences of length m are extracted from T using a sliding window approach. The distances between all the subsequences are calculated and stored in a 2D array called the distance matrix. The Matrix Profile is then calculated as the minimum distance between each subsequence and all other subsequences in the time series. Figure 2.3 shows a sample time series and the corresponding matrix profile, where low matrix profile values indicate a repeating motif whereas the peaks show top discords or unseen motifs.

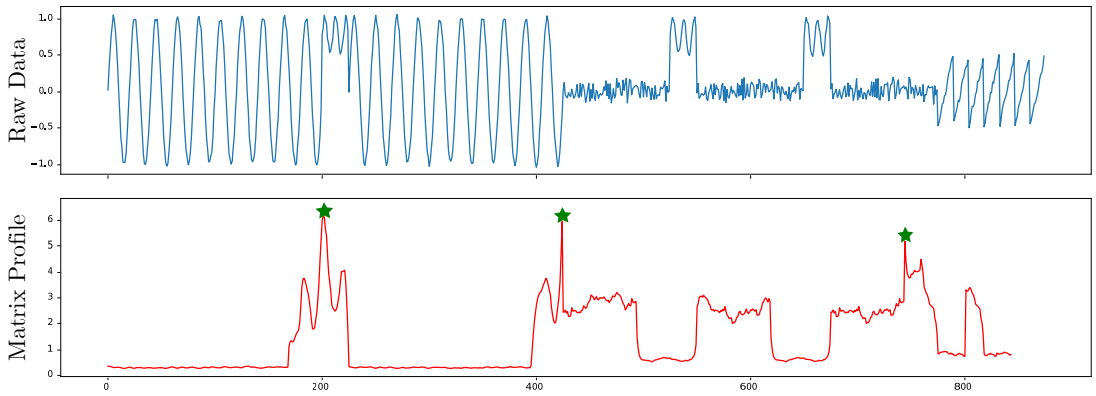

Figure 2.3: A sample time series snippet and its matrix profile vector in red. The green stars indicate the top 3 discords and the low distance profile values identify repeated patterns. Image adapted from [31]

Matrix profile opens up great prospects for several Artificial Intelligence (AI) solutions using the structure as a basis. The next subchapters consists of a segmentation, a time series summarization and a clustering algorithm based on this structure.

2.3 Data Segmentation Using FLUSS

Time series segmentation (or simply segmentation) is a process to divide the time series into appropriate, internally homogeneous segments, in a way that the structure

of time series, through pattern and/or rule discovery in the behaviour of the observed variable, can be revealed [55]. This section explains an approach by Gharghabi et al. called FLUSS, a segmentation approach based on matrix profile. The original approach is then extended by Ochel et al. [69], which will be used in the thesis.

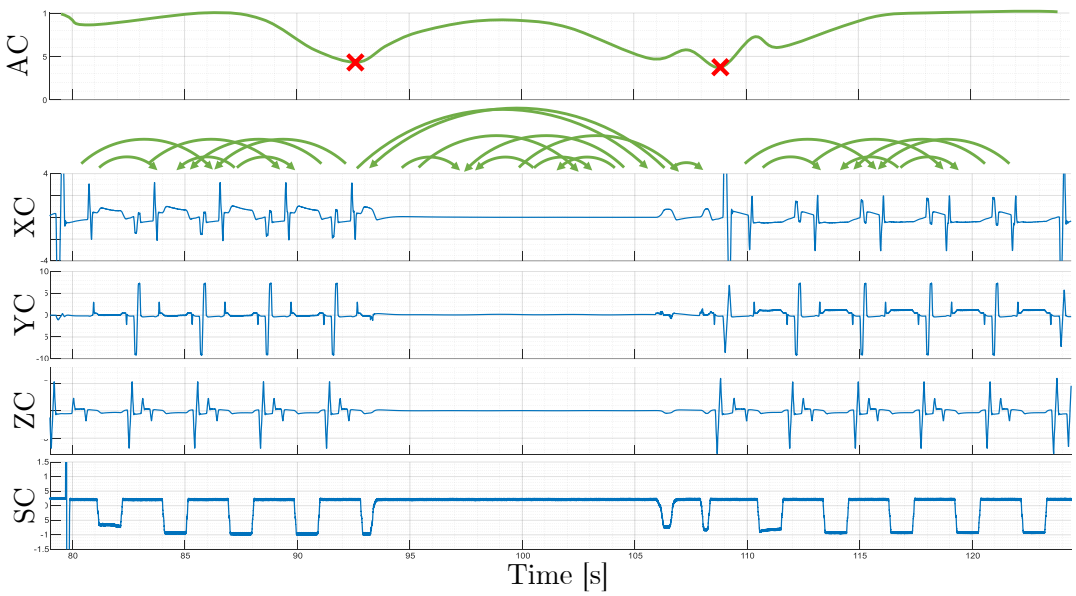

Figure 2.4: The segmentation curve AC, generated from arcs (green), with prominent minima (red) as the segmentation boundary. XC,YC and ZC are the drive current of linear positioning axis in X,Y and Z direction. SC is the drive current of the main spindle. Image taken from [69].

The original approach FLUSS calculates matrix profile of the time series which gives the best match and the location of the best for all the subsequences in the time series. The relationship between the location of the subsequence and its best match can be visually represented as an arc pointing from the subsequence window to the best match. The idea is that there would be more arcs crossing within a meaningful segment than crossing from one segment to another. Summing up all the spatially crossing arcs on each window location in the time series results in an Arc Curve (AC). The segmentation border is drawn wherever there is a significant dip in the AC. Although the algorithms proved well on 1D time series [104], semantically segmenting milling process data involves multiple signals. The preceding approach has been extended by Ochel et al. [69] to calculate the AC considering: a) multidimensional data b) range of window sizes instead of one fixed window and c) different choice of dimensionality for each window. Figure 2.4 illustrates the AC resulting from the algorithm on a real milling process data.

The algorithms sensitivity to finding dips in the AC is controlled by setting a prominence factor. Prominence quantifies the relative importance of a minima compared to the surrounding minima in the same dataset. A higher prominence factor will result in lesser identified dips in the AC. [69]

2.4 Time Series Snippets

Time series snippets (or simply snippets) is an approach introduced by Imani et al. [42] to identify representatives in a time series. Representative is the subsequence that best captures the recurring temporal patterns contained in a certain time series [89].

The snippet approach identifies the top-k representatives in a time series subsequence and the fraction of data that each representative covers. The algorithm is based on matrix profile [32] and it requires snippet length as a parameter. The algorithm can be illustrated with an example time series as shown in Figure 2.5 (top). Considering that the two annotated subsequences might serve as good snippets, their matrix profiles turn out to be a "step" functions with their respective low region corresponding to a region that contains subsequences that are similar to the two annotated patterns. Furthermore, the mutual exclusiveness of the two profiles, as seen in Figure 2.5 (bottom), suggest that the annotated patterns "represent" different non-overlapping regions of the data. In contrast, a poor choice of two snippets for this time series (e.g both snippets in the first half) would result in vectors having high correlation with each other. The above observation suggested an objective function that scores snippets based on the area under the matrix profile curves.

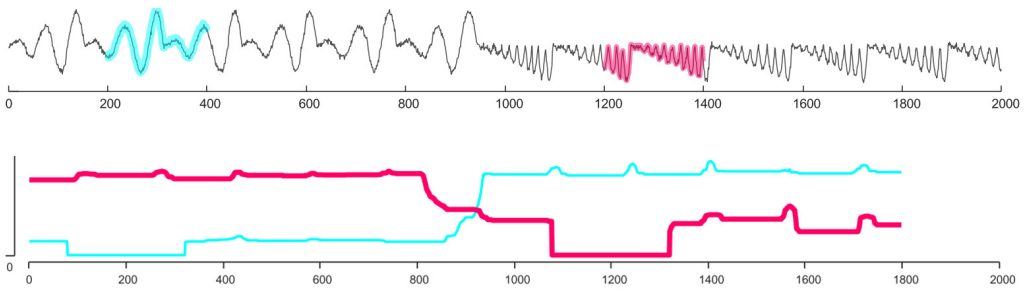

Figure 2.5: (top) A toy time series with two highlighted sections, from 201 to 400 and from 1201 to 1400. (bottom) The matrix profiles of the two queries highlighted above. Note their mutually exclusive nature, when one is high, the other is low. Image taken from [42].

2.5 MPdist Distance Metric

MPdist is a distance measure used to identify similarity between two subsequences [32]. It is again based on matrix profile and considers two pairs to be similar if they share many similar subsequences under euclidean distance, regardless of the order of matching subsequences. Since it uses matrix profile, the size of the sliding window is a hyperparameter set by the user.

The algorithm can be explained by considering pairs of time series segments T_A

and T_B, and a window size of m. Then, the following steps are performed:

1. Compute the matrix profile P_{AB}. P_{AB} indicates the best matches of all T_A subsequences in T_B. That means, for a window of size m at position i in T_A, a window of same size is slid across T_B and the minimum distance is stored at $P_{AB}[i]$. The process is repeated for all windows in T_A.

2. Similarly, compute the matrix profiles P_{BA}, where P_{BA} indicates the best matches of all T_B subsequences in T_A

3. Concatenate the two matrix profiles P_{AB} and P_{BA} and sort it to form a single vector P_{ABBA}.

4. The MPdist of the two time series T_A and T_B is the k^{th} value in P_{ABBA}, where k is another hyperparameter.

The reason that the authors introduced a parameter k is to avoid choosing the smallest distance value, as that could be based on a trivial matches.

The MPdist efficiently computes the distance between two time series while being robust to noise, irrelevant data, spikes, dropouts, wandering baseline, and missing values [34]. However, it must be highlighted that the algorithm is only presented for a 1D time series. Also, a constant window size needs to be defined by the user.

3 State of the Art

3.1 Overview of Big Data Management

Big data refers to a rapidly growing combination of structured, semi-structured or unstructured data that require efficient storing, processing and analysis [83]. Due to the growing volume, increased complexity and the distributed computational needs, it becomes hard to analyse data through conventional data management systems [86]. Thus, a new discipline called big data management has been introduced, that covers the following [83]:

- Platforms for storage
- Techniques for preprocessing
- Tools for analysing and performing computations
- Ensuring data security

A typical big data management workflow is shown in Figure 3.1.

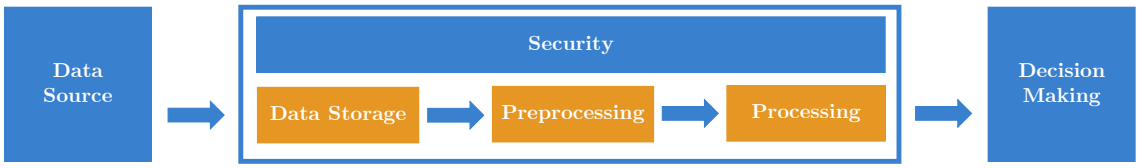

Figure 3.1: Big data management workflow. Image adapted from [10]

In the following points, the underlying steps are elaborated and are connected with the milling process use case discussed in the thesis:

Data storage: As a first step, a fundamental concern is to have an infrastructure which is sufficient enough to store the recorded data, improve the retrieval time and its availability for later processing [10]. The background for the acquisition and storage of recorded data in milling processes is already discussed in Chapter 2.1.

Data preprocessing: Data pre-processing consists of steps to transform raw data into a "clean" format prior to any processing or statistical analysis [59]. Several distinct steps are involved in pre-processing data [87]:

- Data cleaning—This step deals with missing data, noise, outliers, and duplicate or incorrect records while minimizing introduction of bias into the data.
- Data integration—Extracted raw data can come from heterogeneous sources. This step reorganizes the various raw datasets into a single dataset that contain all the information required for the desired statistical analyses.

- Data transformation—This step translates and/or scales variables stored in a variety of formats or units in the raw data into formats that are more useful for the statistical methods.
- Data reduction—After the dataset has been integrated and transformed, this step removes redundant records and variables, as well as reorganizes the data in an efficient manner for analysis.

In milling process, these include data partitioning based on tool changes as well as transformation and reduction that reduces the effect of noise while preserving the shape of the signals.

Data processing: This is the key step to extract insights from the data at hand. It involves statistical analysis, data mining and machine learning to identify the relationships among features and to perform supervised and unsupervised learning tasks [79]. For semantic analysis of time series milling data, methods for data segmentation, clustering and summarization are discussed throughout Chapter 3.3.

Security: Since there can be multiple data sources in a big data environment, the assurance of privacy, integrity, confidentiality and availability is a serious concern [10]. Storage, management and analysis of large quantities of distributed data can result in security and privacy violations [93]. However, solutions for data security and privacy are beyond the scope of this thesis.

3.2 Data-Driven Milling Process Optimization

As the application area of the thesis is milling process, it is important to discuss the state-of-the-art approaches for data-driven optimization. The aim of a manufacture plant is to always reduce the cycle time for manufacturing a part to a minimum, while at the same time complying with the qualitative and economic constraints of the process. Optimization is thus constantly a trilemma between productivity, availability and quality (Figure 3.2), which is further restricted by time-varying boundary conditions, such as wear [17]. Each aspect is elaborated below.

3.2.1 Process Productivity

The process productivity can be increased by analyzing the process to look for the operations with the greatest optimization potential [17]. An example of process analysis for productivity optimization is by reducing non-engagement time. Brecher et al. [14] proposed an engagement detection algorithm for milling processes. In the context of milling, engagement is when the tool is in contact with the workpiece. Hence, identifying and reducing the time when tool is not in engagement improves the process productivity [14].

Once the non-engagement regions are identified, these non-productive regions can be reduced by the Traveling Salesman Problem, which is about finding an order

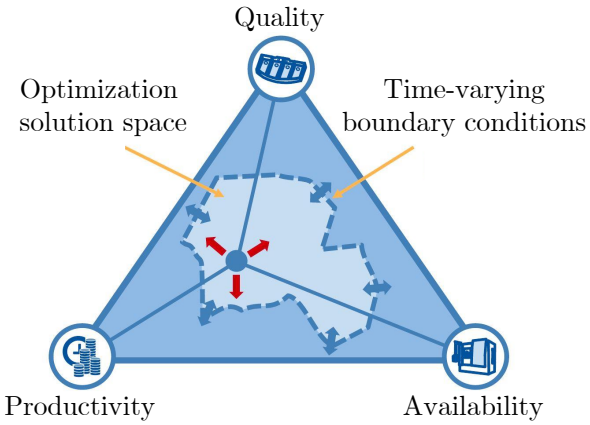

Figure 3.2: Optimization of production processes according to Brecher et al. [13]

of sequences that minimizes the distance travelled [48]. For milling process case. The development of such optimization procedures are costly, but once a procedure has been developed, it can be applied to other machining cases, provided that the corresponding dynamic parameters of the available machines are available [25].

3.2.2 Parts Quality

At the quality control stage, classical supervised machine learning models, such as neural networks, decision tree, and SVM, have been in practice to detect or predict potential deformations and surface deviations in production [54]. In addition, deep learning and computer vision models have also been deployed to recognize eventual quality issues during the automatic production and machining features of parts [4]. This could be utilized to enhance the quality of both the part produced and the assembly processes [41].

Monitoring the quality of data by using virtual workpieces and virtual quality reports are also promising approaches to meet increasing quality requirements of customers [16]. These virtual workpieces are referred to as Digital twin, which are a digital replica of the physical entity [41]. Brecher et al. [16] examined the potential of a Workpiece Quality Monitoring system (WQM) that utilized material removal simulations in combination with dynamic force measurements. They implemented a spindle integrated measuring system into a machine tool system in order to generate virtual quality reports. This enabled a more precise virtual measurement of the workpiece surfaces and therefore a considerably better prediction of the position dependent compliances, as well as machine errors.

3.2.3 Hardware Availability

The availability of production machinery and equipment has a direct impact on the efficiency of manufacturing processes and overall equipment effectiveness, and is, therefore, equally significant for sustainable production, energy consumption and

resource utilization. The degradation and damage level can be affected by working operations as well as other disturbances in harsh manufacturing environments, such as lubricants, soiling, and temperature and the results from mechanical wear. [41]

Several approaches for predictive maintenance based on the monitoring status and health indicators are available in the literature [57]. In addition, accurate prediction of tool wear would affect both the production process and the quality of the part produced. In this regard, data driven Machine Learning and Convolutional Neural Network models are also proposed [40]. Based on such insights, it can be ensured that the required hardware is already in stock for the next use.

Chapter 2.1 discussed the diverse nature of data being acquired and analyzed in a typical milling process environment. Whereas this chapter identified the room for optimizing the process based on the data being collected. However, because of the heterogeneous nature and high volume of data, it is difficult to identify and predict the specific cases where such procedures can be applied. In order to overcome these challenges, it is essential to semantically segment and cluster continuously processed data into homogeneous machining procedures and contextualize it with the non-continuous data on the shop floor.

3.3 Time Series Semantic Analysis

Time series data is being generated at an enormous speed and volume in a wide range of applications. Whether it is position data from location based services, traces produced by a computer cluster, trends in stock market, event logs of a process, medical experimental observations [95] or, as in our case, measurements from the sensors of a milling machine, all are represented by time series data. Consequently, there is an unprecedented interest in extracting semantic insights resulting in new methodologies of segmenting, classifying, clustering and summarizing time series data. In the following sub-chapters, some state of the art techniques for finding semantics in time series data are overviewed.

3.3.1 Data Compression

The progression in the field of Information Technology (IT) and Internet of Things (IoT) has resulted in the generation of huge amount of data every millisecond. As a result, the necessity of storage and transmission of data increases at least twice as much as the increase in storage and transmission capacity [46]. Additionally, an ideal representation approach is a necessary preprocessing step for practical applications such as clustering, classification, anomaly detection and association rules discovery [105]. Hence, in order to overcome this challenge of handling large data and achieving accurate results, an alternative concept called Data Compression (DC) has been presented in the literature. The idea is to transform the original data to its compact form by the recognition and utilization of patterns existing in the data [46].

DC techniques can be divided into two main categories [27]: *lossless* compression and *lossy* compression. With lossless compression, the full and exact reconstruction of the initial dataset is possible. Thus, lossless compression techniques are more appropriate, for the compression of text documents or source codes [47]. With lossy compression, some details of the data can be discarded during the compression process, such that the original dataset cannot be fully recovered from the compressed version. Such data losses can however be tolerable to a certain extent for applications involving pictures, audio, video and high frequency time series data [27].

The Singular Value Decomposition (SVD) is a lossy compression technique which achieves compression by using a smaller rank to approximate the original data [28]. It is an important tool with applications in pattern recognition, multivariate classification and signal processing [90]. On time series data, Spiegel et al. [90] employed SVD for dimensionality reduction in order to identify extremas and inflection points. Whereas, Xie et al. [101] proved the effectiveness of SVD to extract features for clustering stocks price trends. However, SVD calculation can be computationally expensive and has worse Compression Ratio (CR) than other methods like Fast Fourier Transform (FFT) [18].

FFT is a faster and computationally efficient algorithm to compute the Discrete Fourier Transform (DFT) and its inverse [78]. It is a lossy compression technique, widely exploited for audio, image and time series data compression [76]. The disadvantage of Fourier Transform is that it fails to provide the information regarding the exact location of frequency component in time [85].

The limitation of FFT inspired the study of wavelet transforms. The Discrete Wavelet Transform (DWT) is a lossless compression algorithm that projects the signal on a set of wavelet basis vectors [78]. Unlike the FFT that eliminates high frequency coefficients, DWT requires a threshold for eliminating percentage of wavelet coefficients below the threshold for de-noising [82]. Experiments on ECG [82], Photoplethysmography [78] and noisy time series data [49] prove that DWT outperforms other compression algorithms by achieving higher CR and lower Percent Root Mean Square difference (PRD).

Piecewise Linear Approximation (PLA) is a high level time series approximation function. In such a representation, a time series T of length n is represented in the form of k straight lines, determined using linear interpolation or linear regression [51]. The algorithms converting a time series into PLA form are generally classified into the following 3 categories [55]: Top-Down, Bottom-Up and Sliding Window algorithms. Ochel et al. [69] et al. performed Bottom-Up PLA on the motor and spindle currents of a real milling process to reduce not only the number of data points but also the impact of noise on homogeneous patterns. In order to preserve the shape of the signals, they further compressed the original time series by sampling data from every approximated segment, as seen in Figure 3.3.

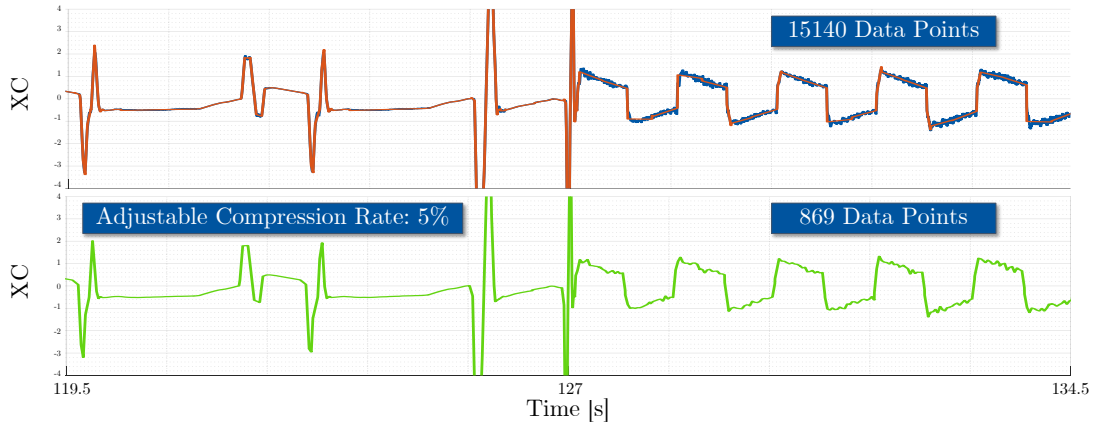

Figure 3.3: Top: Result of PLA applied on the X-axis motor current of a real milling process. Bottom: After compressing the original data using 5% sampling rate on every approximated segment. Image adapted from [69].

Compression of high frequency time series data is an essential preprocessing step towards computationally efficient and effective semantic analysis. Additionally, the reduced datasets results in saving server and storage space as well. Both the DWT and compression using PLA are suitable and experimentally validated approaches for manufacturing data. They can be used as a preprocessing step for compressing and reducing the effect of noise in time series before performing segmentation.

3.3.2 Data Segmentation

Time series segmentation is a fundamental component in the process of analysis and research of time series data. As a data mining research problem, the focus of segmentation has been to divide the time series into appropriate, internally homogeneous segments, in a way that the structure of time series, through pattern and/or rule discovery in the behaviour of the observed variable, can be revealed [55]. Time series segmentation, or simply segmentation where there is no confusion, has been performed in detecting changes in vegetation trends [44], assessing cardiovascular diseases [43] and for context recognition using mobile phone sensors [39].

One frequent problem with segmentation algorithms is it being domain centric. This involves algorithms suited only to a single problem [36] or to have highly domain specific parameters and assumptions about the input data. The approach from Seevers et al. [80] is a good example for the latter. They used data from machine tool sensors to segment the time series for operational state detection and load aggregation. However, their segmentation is performed under three strict criteria: maximum allowable variance within a segment, maximum increase of the sum of the squared deviation, and minimum segment length. Unfortunately, other than the minimum segment length, the parameters cannot be predefined for NC milling data because of the heterogeneous nature of both the processes (e.g manufacturing

of different workpiece) and the segments themselves (e.g a circular pocket roughing variance will be different from drilling segment variance).

The approach by Ochel et al. [69], explained in Chapter 2.3, does not suffer from the above mentioned problems. Also, it is promising for three reasons: Firstly, it considers multiple signals for segmentation. Secondly, it works with a range of window size which is particularly important as the different repetitive tool movements result in different length motifs (e.g motif for circular pocket vs rectangular pocket), and at the same time, similar repetitive tool movements can result in different length motifs (e.g a growing circular pocket). Lastly, it is successful in identifying accurate segmentation boundaries for milling data. The approach proves to be a good basis to perform further analysis for instance clustering, classification and summarization.

3.3.3 Clustering

Clustering is a well-known unsupervised machine learning method for dividing data points into groups called clusters in a way that the observations within the same cluster tend to be more similar (according to some pre-specified criteria) than those in different clusters [45]. In the recent decade, there has been a considerable amount of changes and developments in time-series clustering area that are caused by emerging concepts such as big data and cloud computing, which increased the size of datasets exponentially [1]. For instance, one hour of ECG data occupies 1 gigabyte, a typical weblog requires 5 gigabytes per week, the space shuttle database has 200 gigabytes and updating it requires 2 gigabytes per day [50]. Consequently, several studies have been conducted over the years that prove the effectiveness of clustering for extracting useful patterns and knowledge from big datasets.

However, clustering a time series is a challenging issue because firstly, it is essentially high dimensional data and directly dealing with such data in its raw format is very expensive in terms of processing and storage cost [1]. Therefore, it is desirable to develop representation techniques that can reduce the dimensionality of time series, while still preserving the fundamental characteristics of a particular data set. Secondly, the choice of the distance measure and the clustering algorithm are two major design questions that vary in each time series clustering application [45]. Finally, some criteria must be defined on the basis of which the performance of the method is evaluated [75].

Most works in the literature have used either shape-based classification using a time series distance measure or feature-based classification after finding suitable features for the domain [3].

Shape-Based

In these kind of approaches, shapes of two time-series are matched by either a non-linear stretching/contracting of the time axes or sliding window comparisons of time

series subsequences. They are also labelled as a raw data-based approach because they typically work directly with the raw time-series data. [1]

Such approaches rely heavily on the distance measure used to identify similarity between two subsequences [3]. The first and the simplest choice for a similarity measure is euclidean distance. Several such time series clustering applications can be found in the literature [52]. However, the problem with using the euclidean distance measure is that it often produces pessimistic similarity measures when it encounters distortion in the time axis [61]. Also, its not suitable to compare time series that are not the same length.

The latter issue is overcome with Dynamic Time Warping (DTW) [9], which is illustrated in Figure 3.4. DTW algorithm allows two time-dependent sequences that are similar, but locally out of phase, to align in time. Its main objective consist of identifying an optimal alignment between sequences by warping the time axis in a way that the distance between the two sequences is minimum [30]. However, one limitation of DTW is that it uses pairwise averaging which is intrinsically sensitive to the sample order. Therefore, the method does not consider the temporal context [24]. This limitation is removed by using DTW Barycenter Averaging (DBA) [73], which is an averaging method that includes multiple samples in order to generate a mean sample. Denkena et al. [24] combined DTW-DBA with hierarchical K-Means clustering on milling process data. Although, the approach works well for clustering different sequences in a milling process, it relies too heavily on the length of the time series and the segmentation boundaries.

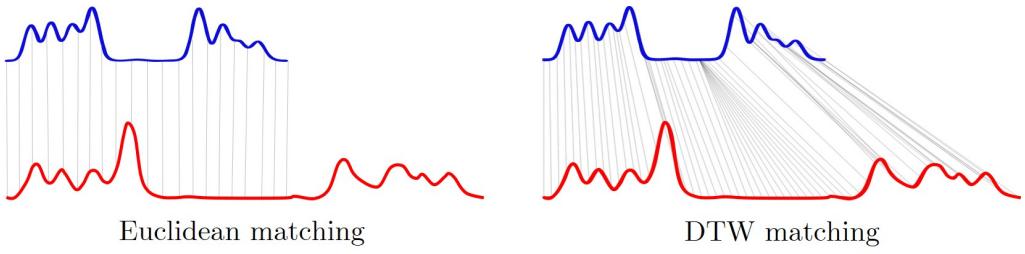

Figure 3.4: Comparison of euclidean distance measurement and Dynamic Time Warping (DTW) on a sample pair of similar but stretched time series. Image adapted from [22].

As the shape-based clustering also includes identifying similar patterns by sliding window comparisons of subsequences [1], Möller-Levet et al. [64] proposed short time-series (STS) distance measure, that is able to cluster similar shapes formed by the relative change of amplitude and the corresponding temporal information. The authors then used an updated fuzzy c-means algorithm that works well even for unevenly sampled or missing data.

Although the DTW is an effective measure for scaling patterns (e.g increasing circular pocket patterns in milling), the MPdist distance metric, explained in Chapter

2.5 works well with shifted or inverted patterns because of sliding window matching [32]. Additionally, it is robust against noisy signal, sharp peaks and it relies lesser on the placement of the segmentation boundaries. The MPdist vector obtained from the approach can be fed into a clustering algorithm like HDBSCAN [72] to identify clusters in the time series. Hence MPdist proves to be a promising shape-based approach for identifying similar non-contiguous segments.

Feature-Based

Feature-based clustering relies on statistical features extracted from the time series. These features are a finite set of measures that can be used to capture the global nature of the time series [97]. Later, a conventional clustering algorithm is applied to the extracted feature vectors. The advantages such approaches have over shape-based are [97]: Firstly, it reduces the dimensionality of the original time series which increases the computational efficiency. Secondly, it is less sensitive to missing values because of its global nature. Lastly, it has the ability to handle different lengths of time series.

Many feature extraction algorithms have been proposed for time series clustering, such as SVD, FFT and DWT [66]. For a clustering task, determining the feature dimensionality is a challenging question. In most of the proposed feature extraction algorithms, either the user need to decide the dimensionality himself or provide the approximation error [103]. However Zhang and Ho [103] proposed a time-series feature extraction algorithm using orthogonal wavelet for automatically choosing feature dimensionality for clustering. The extracted features using Haar wavelet transform [91] are then clustered using hierarchical clustering.

Despite the majority of feature extraction methods being generic in nature, the extracted features are usually application dependent [98]. Thus one set of features that work well on one application might not be relevant to another. For instance, Räsänen et al. [77] only extracted mean, standard deviation, skewness, kurtosis, chaos, energy and periodicity of hourly consumed electricity, using a window size of one week. Whereas Guijo et al. [35] suggested that a combination of features (variance, skewness and autocorrelation coefficient) along with coefficients of polynomial approximation of each segment, outperforms several state of the art algorithms on UCR datasets [19].

Open source feature extraction packages that are suited for time series clustering are also available. For instance, TSclust [65] is an R package containing feature-based similarity measures, being used in many different use cases [23]. Time Series Feature Extraction Library (TSFEL) [6] is another Python library that extracts more than 60 different features on the statistical, temporal and spectral domains. As the extracted features are of higher dimensionality, they can be reduced to a lower dimension using Principal Component Analysis (PCA) or Uniform Manifold Approximation and Projection (UMAP) [60]. The advantage of dimensionality reduction in such

a case is: Firstly, it improves the computational efficiency of clustering algorithm while preserving the local structure of the data [1] and secondly, a low dimensional space enables visualization of clustered features [26]. Allaoui et al. [5] investigated the performance of HDBSCAN and agglomerative hierarchical clustering, when operated on the low-dimension feature space yielded by UMAP. They concluded that applying UMAP before clustering can drastically improve the performance, both in terms of clustering accuracy and time. Recent studies suggest that UMAP features provide considerably better results on density-based algorithms like HDBSCAN instead of K-means or agglomerative hierarchical clustering [71].

Features-based clustering is a robust alternative that works well with noisy time series. It captures both the local and global structure of data that can be retained even if it is followed by dimensionality reduction [71]. However, the process of feature extraction requires a multitude of domain knowledge and coding implementation [6]. Use of an existing feature extraction library like TSFEL offers a set of effective features covering statistical, temporal and spectral domains. However, it is yet to be tested if all of the available features are well suited for clustering milling process data. Also, the feature extraction problem needs to consider the possibility of clustering multiple partially correlated signals of milling time series.

The dilemma of shape-based vs. feature-based approach for clustering has been highlighted in the literature [3]. However, it still remains an open question to date. As Alaee et al. [3] stated that it might be possible that one of the techniques is better for one subset of the classes, and the other technique is better for another subset. Hence, over the course of the thesis, both the shape and feature-based approach will be compared for clustering milling process data.

3.3.4 Time Series Representatives

In many analytical domains, a common query is “Show me some representative of this data” [42]. Such a representative is the subsequence that best captures the recurring patterns in a time series segment [89]. The need for finding a representative might be an attempt to extract the chorus from a music audio [56], to generate medical reports summarizing respiratory patterns [42] or to identify characteristic operational profiles from vehicular sensor data [88]. The possibility of summarizing any available information and presenting it in a meaningful way saves both time and complexity.

Spiegel et al. [89] introduced a Matlab package BestTime that works on both single and multidimensional time series. BestTime first performs agglomerative hierarchical clustering [1] with Recurrence Quantification Analysis (RQA) [99] as the (dis)similarity measure of time series with similar patterns at arbitrary positions. Later, they defined representatives of each cluster as the segment that is closest to the corresponding cluster center of gravity. The possibility of finding “true” representatives with this approach rely heavily on the clustering results. Whereas

the clustering algorithm is highly sensitive to two parameters: Minimum Pattern Length and Similarity Threshold. Both of these patterns strongly depends on the application and needs to be chosen by a domain expert.

Contrary to Spiegel's definition, Ehsan et al. [29] did not assume that the representatives will be present around the cluster centers. Instead they proposed a sparse dictionary learning [2] approach where the entries of the dictionary are chosen from the original dataset. They tested their framework in applications to video summarization, where video frames that best represent the whole video sequence are chosen.

The snippet discovery approach [42], explained in Chapter 2.4, is another promising method to identify the top representatives in a time series. Since the top snippet from the approach is the best representative of the given time series, it is fair to say that each discovered snippet represent the most dominant repetitive pattern in a segment. From application point of view, this concept is really beneficial for two reasons: Firstly, it provides an opportunity to save disk and memory space by only storing snippets instead of the entire partition data. Secondly, it can be followed by pattern clustering task, which would greatly reduce the time to compute the distance matrix, as only a fraction of the data points from the entire time series are kept. The distance calculation for two time snippets have already been discussed (see Chapter 3.3.3), however, the database technology for storing of such complex results, that require processing, is to be discussed in the next Chapter.

3.4 SQL Databases for Big Data

The accelerating progress of technology, especially in the area of information and communication technology (ICT), has led to a large amount of data being generated every day [74]. This fast increase of data gives rise to a challenge on how to store the data, in a way that knowledge can be discovered in an efficient manner. As it is identified over the course of the thesis that the obtained results can be stored and queried in a relational manner, hence this chapter discusses two SQL databases.

The Structured Query Language (SQL) is the most widely used programming language for managing data in database systems [84]. SQL was originally solely used with relational database management systems (RDBMS), but with the introduction of various types of database systems, its use has grown substantially from traditional relational databases to modern big data systems. SQL has been discovered to be a powerful query language in highly distributed and scalable systems, processing datasets of great volume, frequency, and complexity.

PostgreSQL

PostgreSQL is an open source RDBMS with SQL as the standard interface language. It is possible to write stored procedures and functions in PostgreSQL using a variety of programming languages. It also offers extensions to support for other

languages in addition to the bundled languages [68]. SQL and PL/pgSQL are two built-in languages that can be used to build stored functions. However, extensions and drivers for languages like Python and C++ are also available. For time series data, PostgreSQL offers an extension TimescaleDB which is designed to make SQL scalable for time-series data. It does this by automatically partitioning the time series into multiple chunks across time and space while still retaining the standard single-table interface. This is a particularly useful extension for storing not only the raw manufacturing time series data but also the subsequent semantic analysis results that are also of the same nature.

When it comes to storing non-relational data or meta data in the relational PostgreSQL, there is a possibility to store it as a JSON document. JSON and JSONB are the two data types supported by PostgreSQL for such a task. There are two key distinctions between these two data formats: Firstly, the JSON data type stores an exact copy of the input text, which processing functions must re-parse on each execution, whereas JSONB data is saved in a deconstructed binary form. As a result, data saved in the later data type will take longer to load due to conversion overhead, but it will be processed much faster because no re-parsing is required. Secondly, compared to the JSON data type, the JSONB data type supports more operators. The below example shows a PostgreSQL query to create a table with JSONB data and then query based on some condition [62]:

```
-- Create a table
CREATE TABLE workpiece
(id INT, meta_data JSONB);
-- Insert rows
INSERT INTO workpiece
VALUES (1, '{"machine":{"name": "HF3500" ,"id": "10"}}');
INSERT INTO workpiece
VALUES (2, '{"machine":{"name": "E350" ,"id": "20"}, "tool_id":"7"}');
-- Query data
SELECT id, meta_data->'machine'->>'name'
FROM postgres_json_table
WHERE meta_data->'tool_id' = '7';
--------------------------------
2 E350
```

For indexing JSON documents, PostgreSQL supports the GIN index [62]. GIN indexes belong to a class of function-based indexes, which can be used to efficiently search for keys or key/value pairs occurring within documents of the JSONB data type.

However, when it comes to distributed storage of data, fault tolerance and scalable cluster processing, modern database systems like Apache Hadoop are preferred. Apache Hadoop is the most widely used implementation of MapReduce. [84]

MapReduce Systems

MapReduce is a popular programming framework for handling huge datasets. A MapReduce algorithm breaks down a huge dataset into smaller parts that may be processed in parallel on dynamic computer clusters. The whole processing is split into two phases [84]: *map*, which regards the key/value pair as input and generates intermediate key/value pairs. This allows for the processing of the data to be done in parallel, as each input record can be processed independently by a separate map function. The intermediate key/value pairs allows for the data to be partitioned into smaller, more manageable chunks, making it easier to process. Finally, *reduce* merges all pairs associated with the same (intermediate) key and then generates an output. The reduce function operates on the grouped data, which is typically much smaller than the original input data set. This makes it possible to perform the final processing step in parallel, further improving the overall processing time. The framework user has to provide the code for both the parts. The *map* and *reduce* functions have the following form [84]:

map: (key1, value1) $\rightarrow$ list(key2, value2)
reduce: (key2, list(value2)) $\rightarrow$ list(key3, value3)

SQL-based RDBMS are well suited for storing, processing and aggregating relational data with well defined schemas. For unstructured or semi-structured phenomenon found in text documents, GPS data, surveillance data etc, a schema-less NoSQL database is preferred. However, for the given milling process data, an efficient relational database is more suitable.

4 Infrastructure Concept

The previous chapters underlined the importance of locally differentiating milling features and discussed existing practices for the individual components that need to be adjusted and integrated in this thesis. It highlighted the process optimization potential that the manufacturing data contains, and asserted the importance of semantic analysis in order to identify future optimizations. Chapter 3.3 presented such approaches for semantic analysis of time series data. However, these approaches need to be implemented and combined in the form of an analysis pipeline suited for multidimensional milling data. Additionally, the mentioned approaches showed intermediate segmentation and representative algorithms for a single dataset. Whereas, in a real manufacturing scenario, multiple processes generate data from multiple machines in parallel. Furthermore, in series manufacturing, the same NC process is run for generating several instances of the same part. This results in petabytes of time series data that needs to be processed in parallel for analysis. At the same time, the complex structured and unstructured results being generated from the parallel analysis need to be stored in a way that querying subsets of data based on these results is efficient and accurate.

Hence, the proposed work aims to design and integrate an infrastructure for optimized manufacturing. It offers a new methodology for semantically analyzing milling process data in a parallel manufacturing environment. That methodology includes combining approaches for segmenting and summarizing data and then implementing a clustering method that enables an expert to obtain subsets of datasets containing specific milling features along with its metadata. This makes large volumes of manufacturing data easily comprehensible and provides a basis for future AI models.

4.1 Proposed Implementation

The design of the overall infrastructure is illustrated in Figure 4.1 and each of its component is elaborated below.

4.1.1 Data Collection Pipeline

As a first step, a communication network is established that reads live milling process data from multiple machines on the shop floor and transfers it to an edge device for triggering semantic analysis pipeline. The read data is processed and stored in a database at a remote server via an MQTT protocol. This provides two design opportunities for accessing the streaming data. First one is to have an MQTT client that listens to the live data directly from the machine interface. Second one is to listen to the database for event-driven changes. Those event-driven changes refer to the initiation or completion of a process at a machine. For a single process, the data can be read in chunks until the completion event is received.

In addition to the live data being read, historical process data also needs to be

gathered for training a clustering model. The datasets have to be carefully chosen in order to have a collection of datasets that contain the common milling process tool movements (circular pocket milling, roughing, finishing, etc) with minimum noise and a suitable bias-variance tradeoff to avoid underfitting or overfitting the model.

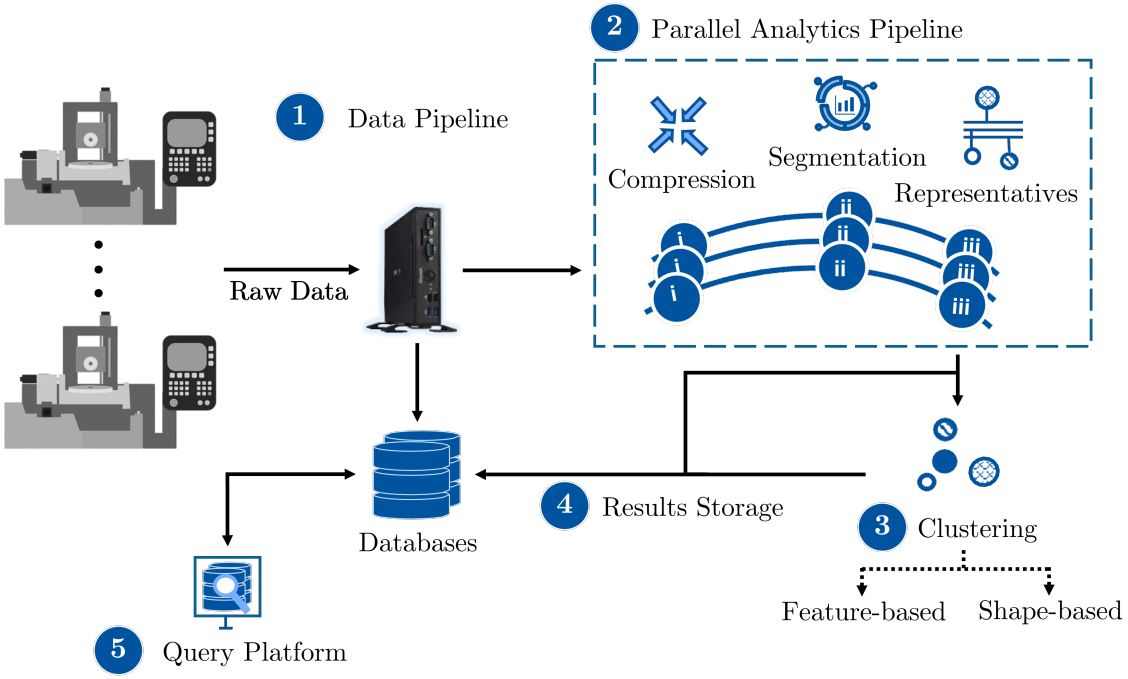

Figure 4.1: The infrastructure workflow.

4.1.2 Parallel Analytics Pipeline

Upon receiving the streaming process data, multiple semantic analysis algorithms need to be integrated and triggered in a parallel manner. It is important to highlight that these are existing solutions that are not developed in the thesis but are integrated together as necessary preprocessing steps before clustering:

(i) Data Compression: The high frequency streaming data is first compressed for computational efficiency and effective analysis using the DWT method discussed in Chapter 3.3.1.

(ii) Semantic Segmentation: The fact that milling processes consists of distinct repeating patterns, is exploited for performing segmentation using the approach by Ochel et al. [69], explained in Chapter 2.3.

(iii) Representatives Discovery: The fact that milling processes consists of distinct repeating patterns in each segment, is exploited by discovering representatives of segments. These representatives can then be used for clustering task. A modified version of snippet discovery algorithm [42], developed at WZL is used for this task. The original snippet discovery approach is mentioned in Chapter 2.4.

In addition to the pattern-based algorithms, approaches for engagement detection and feedrate calculation are also included in the pipeline. These two are also not developed as part of the thesis, however unlike the listed approaches, these approaches do not play any role in clustering. They are only included in the pipeline to build query use cases that are mentioned in Chapter 8.3.

4.1.3 Clustering

The segmentation borders and the identified snippets are then used for clustering. The task of querying data based on a pattern requires a clustering model with a training phase and a test phase. Both the shape-based approach using MPdist [32] and the feature-based using TSFEL library [6] and UMAP [60] have been implemented to obtain a similarity matrix. These are then given to a clustering algorithm HDBSCAN [37]. Once the model is trained on the historical data, it is evaluated for both the approaches in test phase with live data.

4.1.4 Results Storage

The pipeline and the clustering results, which consists of both structured and semi-structured data is stored in a database, in a way that the queries can be aggregated to get tailored results. For this purpose, a relational database PostgreSQL is used.

4.1.5 Query Platform

As the data and the results are read from multiple sources, a Python platform is developed, where the backend acts as a middle-ware to combine queries and results. The frontend is a Streamlit [92] dashboard that enables an expert to obtain customized segments of original process data along with metadata. A sample possible query can be seen in Figure 4.2. For the purpose of understanding, the query can be translated as “Show me all the process subsequences where the spindle speed is around 5000 and contains a sin-wave pattern in any movement signals”.

4.2 Validation Strategy

The evaluation strategy of the thesis is threefold:

4.2.1 Infrastructure Evaluation

The design concept of the infrastructure is evaluated in the context of a milling process. Data from multiple machines present at the shop floor of Machine Tools Laboratory (WZL), is generated and monitored for the entire workflow.

4.2.2 Clustering Model

The results from both the feature and shape-based approach are evaluated for milling process data. Two types of evaluation are performed:

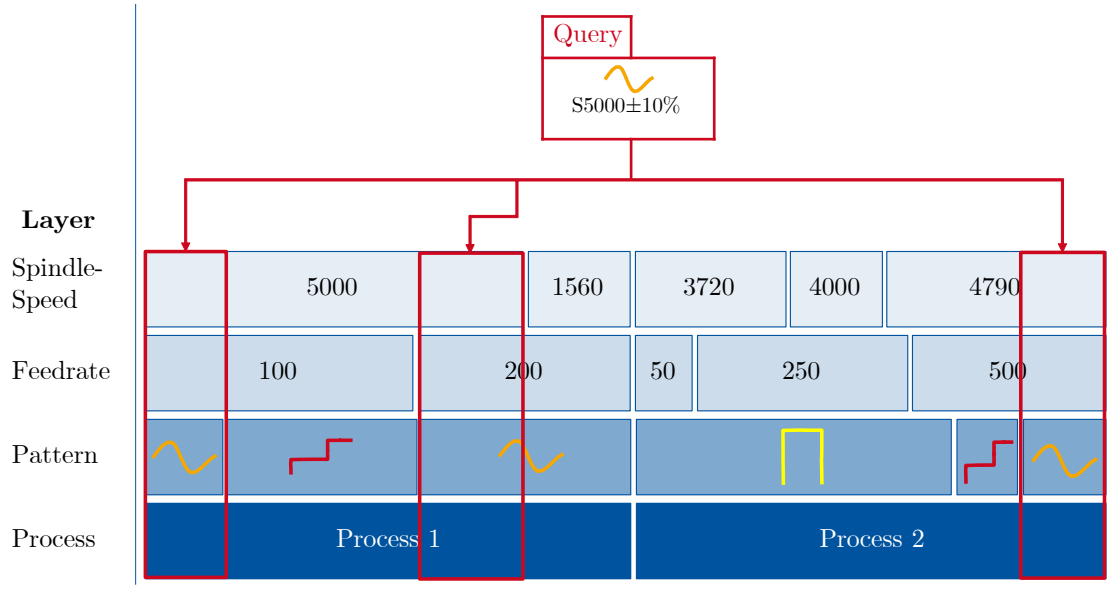

Figure 4.2: An example query for retrieval of subsets of all process data where the spindle speed is 5000±10% of the stored spindle speeds and contains a sin-wave like pattern.

- Metric-Based: Evaluating the performance of a clustering algorithm is not as trivial as counting the number of errors or the precision and recall like in the case of supervised learning algorithms. In the absence of ground truth labels, DBCV [67] is used to analyse the clusters based on their density and separation from each other. In addition to the evaluation based on cluster density, the consistency of the identified clusters within each segment is also calculated. This is referred to as clustering accuracy (or simply accuracy where there is no confusion). Finally, the prediction of new points on the trained HDBSCAN clustering model is evaluated by calculating the precision, recall and F1-score.

- Human Expert-Based: In addition to the metric-based evaluation, the results are analyzed by the experts responsible for designing the workpiece. The generated clusters for a dataset are compared with its corresponding CAD model to validate if the patterns for each feature are clustered together.

4.2.3 Query Performance

Because of the semi-structured nature of the data and use of multiple tables, the queries need to be aggregated and their response is combined in the query platform to generate the desired results. This requires methodical pipeline results storage, efficient query writing, data indexing and vectorizing the query responses, otherwise it causes a massive computational overhead.

The ultimate goal of the thesis is to have a proof of concept of a methodology

involving several components that enables structuring a given data and generating insights by obtaining tailored data. The concept can then be tuned for any time series manufacturing environment.

5 Data Collection Pipeline

Data gathering is an important aspect of the manufacturing industry as it enables organizations to collect and analyze data from live machines to improve operations, optimize processes, increase efficiency and make data-driven decisions. For this thesis, a data pipeline is developed to collect data from various sources and move it to a central location for analysis.

5.1 Data Source

The implementation and evaluation of the thesis is done using the trace data from a machine tool Heller HF3500 (seen in Figure 2.2) present at the shop floor of WZL. The HF3500 is a five-axis machining centre for milling processes. This is used for the production of the test work pieces and the process-parallel recording of the trace data as an input to the implemented infrastructure. The machine tool consists of an interface through which the user can load and run an NC program. During the execution of the program, trace data from the machine control can be accessed. Table 5.1 describes the different trace signals that are relevant for the thesis.

Signal	**Unit**	**Description**
time	ms	Timestamp of the recorded row
X1	mm	Actual position of the tool along the X-axis
Y1	mm	Actual position of the tool along the Y-axis
Z1	mm	Actual position of the tool along the Z-axis
XC	A	Drive current of the linear positional axis along X
YC	A	Drive current of the linear positional axis along Y
ZC	A	Drive current of the linear positional axis along Z
SC	A	Drive current of the spindle axis (spindle current)
SS	rpm	Measured actual speed of the spindle axis (spindle speed)
LN	-	Current NC line in the NC programme
TN	-	Tool number of the tool being used for milling

Table 5.1: Trace signals recorded from the machine tool that are used over the course of the thesis.

As mentioned in Chapter 2.1 as well, the machine tool has a limited hard disk that

can trace the data for a few seconds. In order to read data for an entire process, a WZL developed Machine Trace service is used. The service reads live traces from the NC control and enables the user to download them as a CSV. In addition to file downloading, it also transfers the live data over an MQTT server. The MQTT receiver at the server saves the data in a PostgreSQL database.

5.2 Event Extraction from Live Machines

Extracting event-based data from a live machine can provide valuable information about both the running process and the machine. One can analyze such data to gain insights into machine's behaviour, identify potential issues, and take corrective action to prevent or mitigate any disruptions.

In the context of milling processes, events refer to the induced changes in the machine during a running process. This includes starting and stopping of an NC program (either by the user or the program), tool changes, feed rate changes etc. The WZL's Machine Trace service also records such events and saves it in the PostgreSQL database along with the live data. Figure 5.1 presents the Entity Relationship Diagram (ERD) of the schema generated by the service and Table 5.2 provides a description of each table columns.

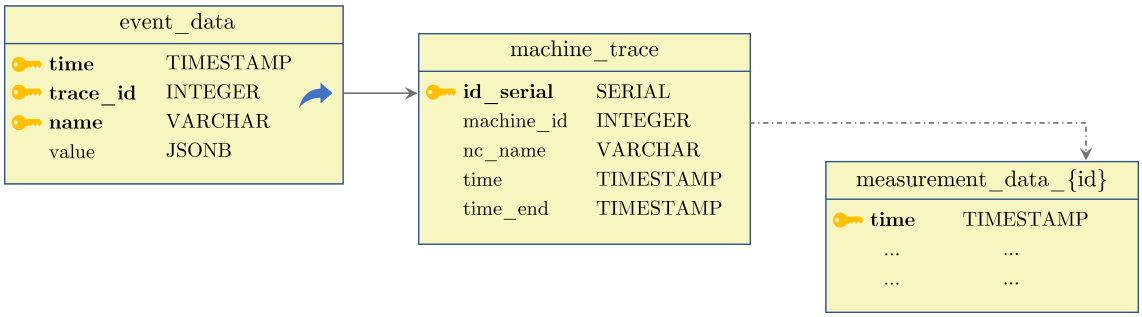

Figure 5.1: Entity Relationship Diagram (ERD) of the event recording schema consisting of entity blocks, their columns, data types, primary key constraint (yellow key), foreign key constraint (blue arrow) and their relation with other entity (solid gray arrow). The *machine_id* is used to create dynamic entities for every new machine.

It is important to note that the WZL's Machine Trace service manages a separate database table for every machine. An ID of each machine is appended to the table name as seen in Figure 5.1. This enables storage and querying of high frequency trace data using the measurement timestamp as a unique identifier. Hence, in order to query the trace data between two events of a process (e.g tool change), one needs the timestamps of these events and the process id (*trace_id*) from *event_data* table. This *trace_id* will be used to obtain the *machine_id* from the *machine_trace*. Finally, the obtained timestamps along with the machine's ID will be used to query the specific chunk of measurement data of the process.

This event extraction and storage provides a base to perform data acquisition and partitioning, which is essential when working with multiple machines generating high frequency parallel time series data. The implemented data collection setup explained in the next subchapters, builds on top of this event recording setup.

event_data	
time	Timestamp of the recorded event
trace_id	Process ID for which the event is generated
name	Name of the generated event
value	Additional event information. E.g tool number in case of tool change
machine_trace	
id_serial	Running process ID
machine_id	Machine ID
nc_name	Name of the NC program (also used as the dataset name)
time	start time of the process
time_end	end time of the process
measurement_data_{id}	
time	Timestamp of the recorded measurement

Table 5.2: Description of the columns in the event recording schema (see Figure 5.1 for the schema).

5.3 Event-Driven Live Data Collection Setup

The previous subchapter overviewed the importance of event-driven data collection, and illustrated the existing setup of obtaining and storing process events at WZL. This subchapter explains the implemented event-driven measurement data collection and the partitioning strategy from live machines.

As mentioned in Chapter 5.1, the live measurement data is being read continuously and transmitted over an MQTT server. At the server side, the received measurement data along with the event data is saved in a PostgreSQL database. This presents two different strategies to access the measurement data. First is to listen over the MQTT channel and constantly read the live data. The second option is to listen to events in the database, and as soon as any event is received, the measurement data corresponding to the process can be read from the database. This option is more feasible as it ensures data partitioning and prevents any data mixing.

5.3.1 Setting Up Triggers at the Database

As the received events are being saved in the database according to the schema in Figure 5.1, triggers are added to the respective table to get notified whenever there's a new insert to it. PostgreSQL triggers are database callback functions, which are automatically invoked when a predefined database operation occurs. Those operations can insert, update, delete of a row or multiple rows. Such triggers can be invoked either for each affected row or once per statement. A basic syntax of the trigger is given below:

```
CREATE  TRIGGER trigger_name
[BEFORE|AFTER|INSTEAD OF] [INSERT|UPDATE|DELETE]
ON table_name
FOR EACH [ROW|STATEMENT]
EXECUTE PROCEDURE callback_function();
```

Such a trigger is applied to the *event_data* table, so that the SQL callback function is called on the insert of every new event. The callback function is responsible for converting the new row into a JSON object and emit a notification using Postgres' $pg\_notify()$ function. $pg\_notify()$ is a system function for sending notifications to other sessions connected to the same database listening on a channel specified with the $LISTEN$ command. The $LISTEN$ command is executed and handled in a Python service explained below.

5.3.2 Multi-Threaded Queue-Based Notification Handling

With the availability of multi-core CPUs, threads are leveraged for maximum application performance and responsiveness. Multi-threading is a model of program execution that allows for multiple threads to be created within a process, executing independently but concurrently sharing process resources. A common application of multi-threading is a web server that utilizes multiple threads to simultaneous process requests for data at the same time. Python provides a built-in module called *threading* that provides a simple and intuitive API for spawning multiple threads in a program. [58]

In order to handle PostgreSQL notifications, execute measurement data querying and perform further analysis, a multi-threaded Python service is developed, which runs on the edge device of every machine tool. The workflow of the service is visualized in Figure 4.1, and each of its components are elaborated below:

Main Thread

The main thread is the parent thread of the program and is responsible for creating child threads. It creates an event thread and continuously checks an event queue for any elements. In case the queue is not empty, it pops the first item and creates a pipeline thread to handle the event. In order to optimize the CPU processing speed, the number of parallel pipeline threads are capped to the number

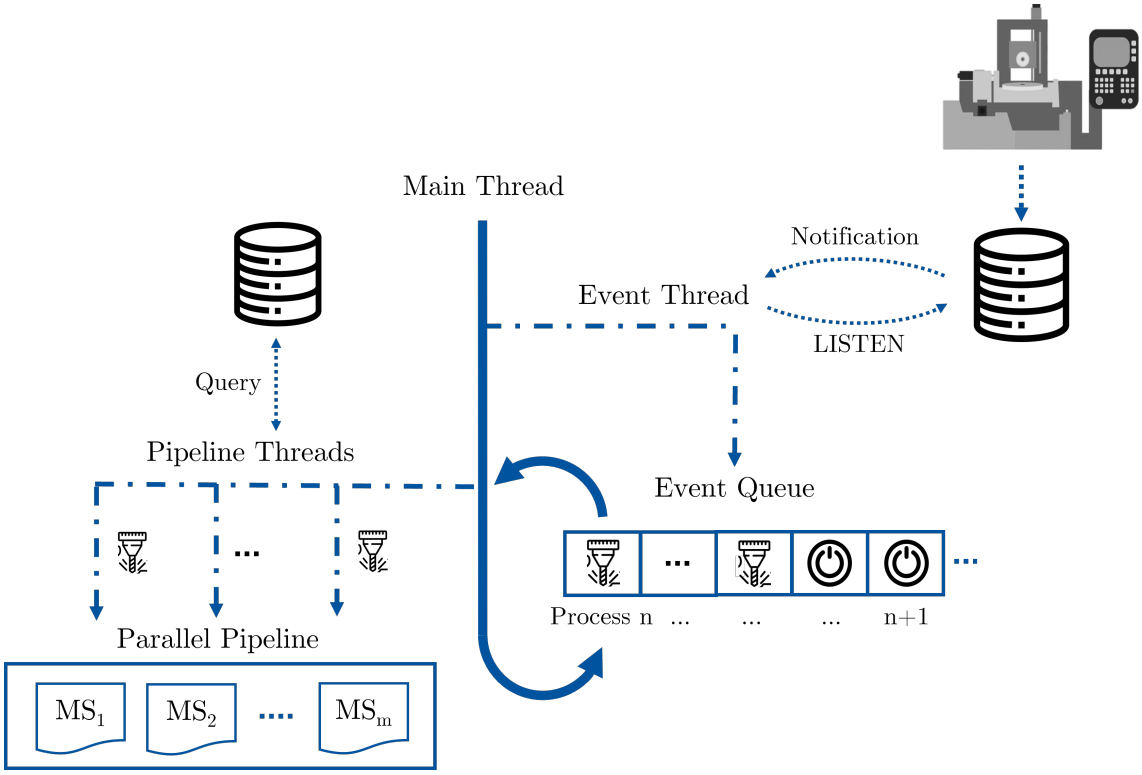

Figure 5.2: Implemented workflow of multi-threaded event notification handling.

of logical threads on the CPU. If the number of pipeline threads reach the defined limit, the main thread waits for the completion of any of the pipeline threads.

Event Queue

A synchronized and unbounded FIFO queue to maintain the process events received as notifications from the database.

Event Thread

The event thread, parallel to the main thread, is responsible to listen to the Postgres notifications using the *LISTEN* command. In case a notification is received, it validates it and puts the information in an event queue.

Pipeline Threads

The pipeline threads are a set of parallel threads, each processing a single event. For an event, the pipeline thread searches for any previous event to get the start time of the data partition. Then, using this partition start time and end time (obtained from the event), it queries the measurements table. As a single partition could be hundreds of thousands of rows long, querying all this data using a single database connection could prevent other threads from querying. Hence, each pipeline thread

obtains a separate database connection from a connection pool. After querying the measurement partition, it is forwarded to a parallel pipeline containing several microservices (MS) for further analysis (more on this in Chapter 6.1). Once the thread's work is finished, the database connection is returned to the connection pool and the thread is terminated.

The implemented workflow ensures parallel processing of multiple milling processes. At the same time, the idea of using the events to partition each dataset into smaller chunks guarantees that the database querying and the analysis pipeline will work more efficiently than processing an entire dataset after the completion of a process.

5.4 Historical Data Collection Setup

In Data Analytics projects, historical data calculation is an extensive process that requires development of data pipelines and meticulous data selection. The gathered data can be used to train and validate Machine Learning models. It is important to have a large and diverse dataset so that the model can learn to generalize and make accurate predictions on new data. Furthermore, it can be used to perform cross validation for hyper-parameter optimization as well as evaluating the model accuracy.

For the purpose of this thesis, historical data is collected for training a clustering model. However, it is important to define some requirements for data selection. This ensures that the model is trained on meaningful and diverse features, that prevent underfitting and overfitting. The following sub-chapters explain the basic considerations while selecting historical data. Additionally, it explains the updates made to the live data collection module in order to enable historical data reading.

5.4.1 Basic Criterion for Process Selection

A major issue while working with datasets from real manufacturing systems is that it could possibly be a few seconds to several hours long. Having such an unevenly lengthed datasets could overtrain the clustering model on specific features and undertrain on the rest. Hence, it is essential to ensure that the durations of all the selected processes are similar.

Another consideration for training any model is that it needs to have relevant features that are also expected in test datasets. Failure to train on meaningful features would result in a low accuracy model. In case of serial manufacturing of the same parts, the presence of specific features is assured. However, the problem with manufacturing processes at WZL is that they are mostly experimental processes that are ran for research purposes. Hence, they lack both the common milling features and the consistency in milling strategies. Figure A.1 and Table A.1 shows model of a sample process called Demo Part Alpha (or simply Alpha) and its specifications. The part is designed specifically for the thesis and is run on HF3500 machine. Such

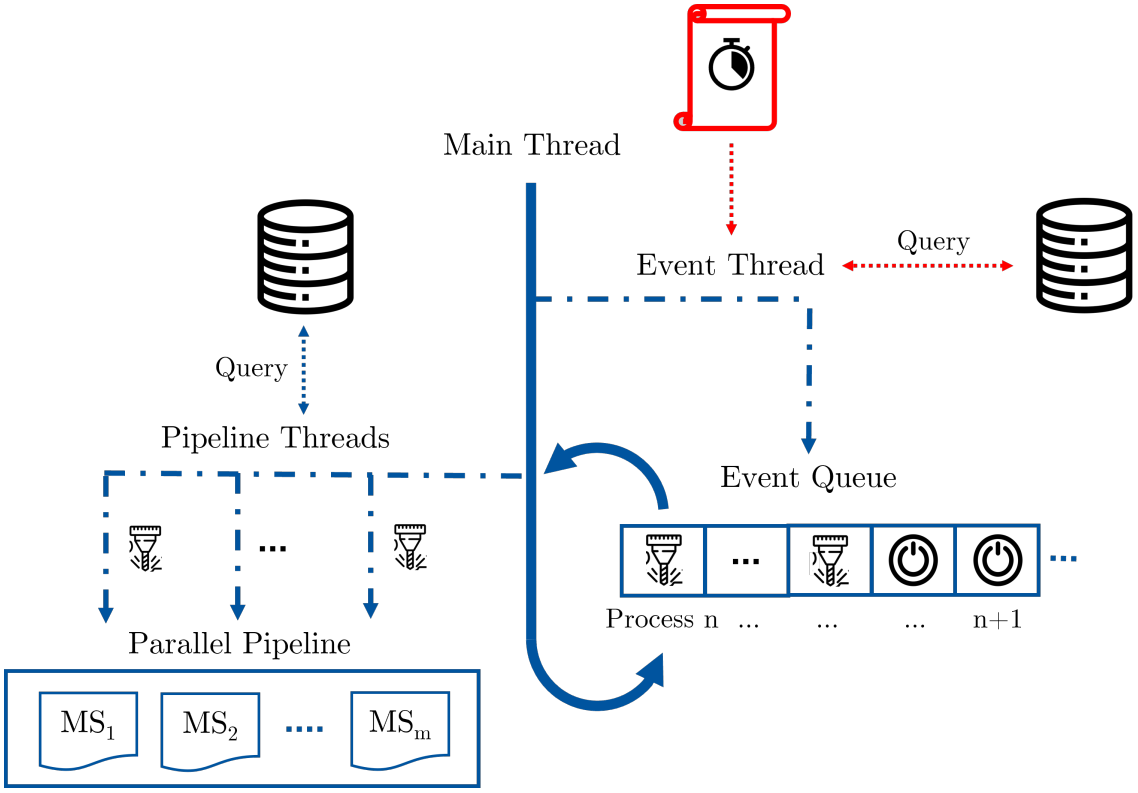

Figure 5.3: Updates (in red) to the live data collection module for historical data collection.

a process is a good candidate for training the clustering model. It includes some common features that can be found in many of the industrial milling processes.

5.4.2 Updates to Live Data Collection Module

The difference between the live and historical data collection strategy is the change in event reading (see Figure 5.3). In live data collection, the events were received as notifications from the database. However, in order to run the service on specific datasets that are already present in the database, a CSV file has been maintained that could specify either process IDs or a time range to select processes from. The event thread reads all the relevant events for the read processes and places them in the event queue. The main and the pipeline threads work in the same way as described in Chapter 5.3.2

The implemented live and historical module can be toggled using a configuration file. This data collection strategy ensure code reusability for both the modes and it is easy to use for any user running the service. The queried time series data partitions are sent to a parallel pipeline running different microservices which are explained in the next chapter.

6 Implmentation of Analytics Models

The objective of the thesis is to enable partitioning of complex milling data into locally differentiated milling features and group similar features together. This process requires multiple analytics models to be applied on the data collected from the pipeline. This chapter explains the algorithms, as part of step two and three of the implemented infrastructure (Figure 4.1), for partitioning the data and then clustering it using both shape-based and feature-based approaches.

6.1 Parallel Analytics Pipeline

The parallel analytics pipeline is the second step of the infrastructure concept presented in Figure 4.1. The time series data received from the data collection pipeline is passed to this pipeline. Several existing microservices are integrated and connected in this pipeline. The microservices presented in this chapter are not developed as part of the thesis. Instead they are connected in a way that each microservice mentioned from Chapter 6.1.1 to 6.1.4 uses the results from all previous microservices. Also, accurate training and testing of the clustering model relies on all these microservices, as detailed from Chapter 6.2 onwards.

For every data partition in the pipeline thread (see Figure 5.2 and 5.3), an instance of the parallel pipeline is triggered. The microservices in the pipeline are executed sequentially because each of the microservice require results from the previous ones, as mentioned before. This subchapter explains all the included microservices in the parallel pipeline, their execution and the generated results.

6.1.1 Spindle Speed Partitioning

The first step in the pipeline is to partition data based on constant spindle speed (SS) regions, as done by Ochel et al. [69]. As a single milling feature cannot span over a significant change, SS partitioning ensures meaningful boundaries and allows parallel processing for the rest of the microservices. In addition to the SS partitions, tool changes are also considered as partition boundaries since they require the spindle to stop.

The original measurement data is then divided based on each partition boundary and is forwarded for data compression.

6.1.2 Data Compression Using Wavelet Transform

Data compression is about transforming the original data to a reduced form by recognizing and utilizing the existing patterns. When performed on the milling process data recorded at 500Hz, it helps in reducing the noise in the time series and improves the computationally efficiency of further microservices. Chapter 3.3.1 discussed Discrete Wavelet Transform (DWT) and Piecewise Linear Approximation

(PLA) as two highly suitable and experimentally validated approaches for compressing milling process data. However, the bottom-up strategy of PLA is significantly slower than DWT for the same level of compression. Hence, a wrapper of Python library pywt [53] is used in the pipeline to compress the signals.

The conducted experiments show that the measurement data can be compressed by 95% without loss of patterns and any aliasing effect. The compressed time series is now the data source for the rest of the micro services and for the clustering model.

6.1.3 Matrix Profile-Based Semantic Segmentation

Time series data segmentation is the process of dividing a time series into smaller segments containing a periodic pattern. Chapter 2.3 explained the FLUSS approach that uses Matrix Profile as a basis to perform segmentation on time series data. It also highlighted the extension of FLUSS by [69] et al. for multidimensional milling process data with a range of window sizes.

The presented approach by Ochel et al. is integrated as the next step of the pipeline. The results from Figure 6.1 prove that the algorithm identifies meaningful segments in milling process data, complying with human intuition. It also shows that each of these segments represent a single milling feature. Hence, the segmentation provides intermediate clusters already. Additionally, dividing each SS partition into further segments offers more parallel processing during further analysis.

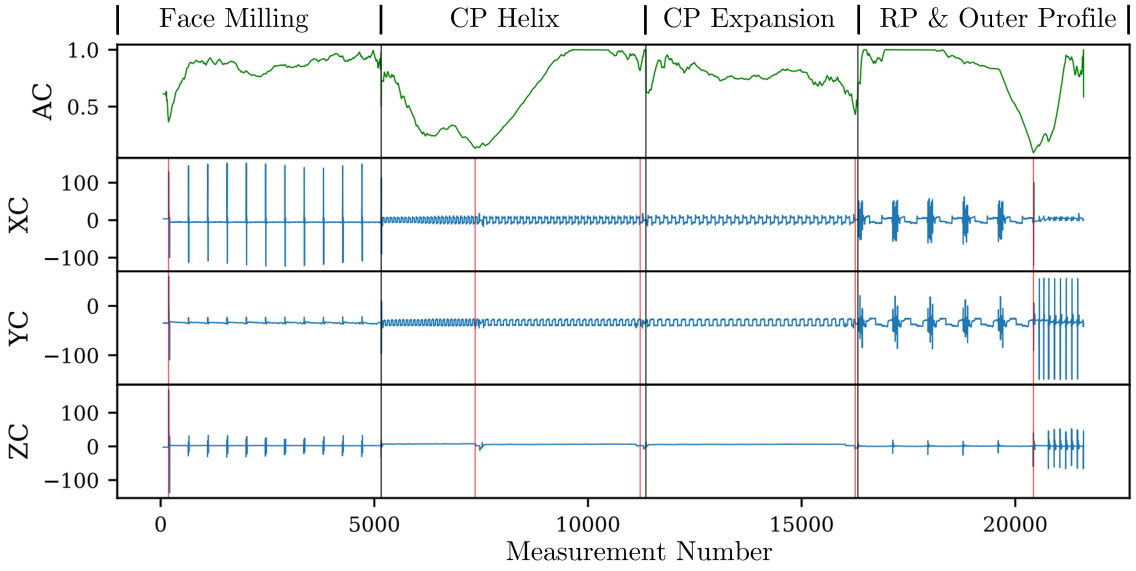

Figure 6.1: Segmentation results on a subset of the Demo Part Alpha dataset, with a window size range of 4 to 50, and a prominence factor of 2.0. The black vertical lines are partition borders whereas the segment borders (red lines) are drawn wherever the green Arc Curve (AC) shows distinct dips.

6.1.4 Snippet Discovery

Snippet discovery is an approach by Imani et al. [42] to find representatives in time series data (see Chapter 2.4 for explanation). The limitation of the original approach is that it requires the snippet length to be predefined. This might work for applications where the pattern frequency is known beforehand, however in milling data, the snippet length of each feature (segment) could be different. For this reason, the snippet length of each segment is first computed using nonuniform fast Fourier Transform (NUFFT) [7]. The NUFFT computes the dominant frequency within a segment. As each segment is only expected to have one snippet, this is equal to the snippet length. The computed snippet lengths are then fed to the original algorithm to obtain the top snippet for each segment (Figure 6.2).

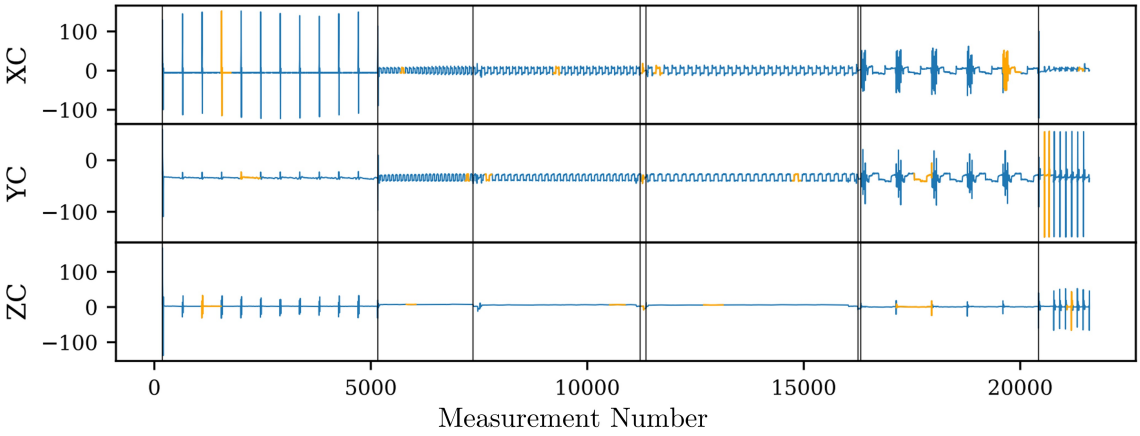

Figure 6.2: The discovered snippets (yellow) in a subset of Demo Part Alpha dataset, using the segment and partition boundaries (black).

The discovered snippets along with the computed boundaries are used in the training and the test phase of the clustering model. Hence, it is essential to find meaningful results from all these microservices. The implemented snippet discovery algorithm provides relevant representatives with some exceptions with the estimated snippet length. For instance, in Figure 6.2, the snippet of the second last segment in YC does not cover a full pattern whereas the last one exceeds a single representative length.

6.1.5 Domain Knowledge-Based Microservices

In addition to the above mentioned microservices, two more microservices are also integrated in the pipeline. The first one is engagement detection. It divides each discovered SS partition into further segments where the milling tool was (not) in engagement with the workpiece. Although the discovered engagement segments are not used in the training and testing, they can be utilized in the analysis and optimization of processes. For instance, increasing the feed rate for regions with no engagement improves process efficiency. The second one is to partition the measurement data into different states of feed rate (constant, rapidly increasing, standstill,

etc.). This also helps an expert to perform optimizations. For instance, identifying and minimizing the standstill time reduces the overall process time.

These individual microservices are triggered at the end of the pipeline and their results are later stored in the database. The results from the rest of the microservices are forwarded to the clustering model, which is discussed in the next subchapters.

6.2 Shape-Based Clustering Using MPdist

Time series shape-based clustering, explained in Chapter 3.3.3, is a technique to group similar time series data together by using a distance measure to calculate the similarity between the time series. The chapter also introduced MPdist [42] as a promising shape-based approach for obtaining a distance metric that could later be used for clustering. It highlighted MPdist's robustness to noise and its effectiveness with shifted/inverted patterns. However, the original MPdist approach has some limitations that need to be addressed. For this reason, the original approach is extended in the thesis in order to make it work with the milling process data.

6.2.1 Variable Window Sizing

In the original MPdist approach, a window of fixed size is slid across the two time series to find the best subsequence matches. The problem with a fixed predefined window is that it might not cover a full pattern in a segment or, on the contrary, might cover more points than a single pattern. It becomes even more difficult when the pattern in a single segment is scaling, for instance, a growing circular pocket will result in patterns that are increasing in length (Figure 6.3).

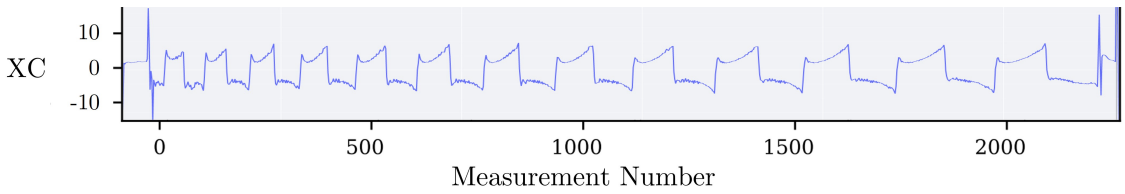

Figure 6.3: XC signal of a growing circular pocket segment.

Hence, instead of having a fixed window size, the original MPdist approach is extended to work with a range of window sizes. For every pair of z-normalized segments, its MPdist is computed i.e. similarity measure of time series segment T_A with segment T_B given window size w. The process is repeated for every window size in the given range to get a vector of MPDist for every segment pair. The optimal window size for a segment pair is chosen by the Elbow method.

The Elbow method is a common heuristic in mathematical optimization which gives a cut off point with maximum discrete difference in the cost. The method is commonly used in dimensionality reduction or in selecting the number of clusters in

K-Means [81]. In this case, the MPdist value at the elbow is assigned to the time series.

6.2.2 Handling Multidimensionality

Another limitation that the original MPdist approach has is that it computes a distance measure for 1-dimensional time series and there are no multidimensional validation of the approach in the literature. However, when clustering manufacturing data based on the drive current signals, it is not possible to use either of the signals individually because the tool movement could be along different axes at any given time. Hence, the original MPdist approach needs to be extended to handle multiple dimensions.

For every z-normalized segment of a signal, its MPdist is computed with all the rest of the segments with variable window sizes to obtain a single distance metric. The same is done with the rest of the signals to result in an MPdist value across each dimension. The optimal dimension can again be chosen using the Elbow method. However, because of the multi-axes tool movement, the rest of the signals could also contain meaningful motifs that should not be discarded. Hence, an average MPdist is selected for a multidimensional segment pairs.

6.2.3 Problems with the Approach

The extension of the original MPdist approach to handle multidimensions and a range of window size presents its own complications, that are following:

MPdist Performance

For a segment pair T_A and T_B with a combined length of l_{AB} and total number of subsequences $subseq_{AB}$, the time complexity to calculate the MPdist of the pair is $O(subseq_{AB} \times l_{AB})$. However, by addition multiple dimensions $ndim$ and a window size range of length l_w, the complexity becomes $O(subseq_{AB} \times l_{AB} \times l_w \times ndim)$. This greatly increases the time to build a distance matrix of all segment pairs in the time series, and thus impacts the application feasibility of the approach.

The performance issue can be improved by fixing the window size of each segment to its computed snippet's length. Alternatively, instead of sliding the window across the entire segment, it is also possible to consider just the snippets. This reduces both the number of subsequence comparisons needed and the window size range. However, these two solutions present their own problems which are mentioned next.

Snippet Size as the Window

Fixing the window size of each segment to its corresponding snippet's length will indeed improve the performance and will make sure that the entire pattern is covered in comparison. However, imagine a case of two circular pocket segments T_a and T_b

with snippets of length s_a and s_b where $s_a > s_b$. Then, for computing their matrix profile join i.e. *mp_abba* [42], the window size in *mp_ab* will always be too big for segment b and vice-versa, the window size in *mp_ba* will be too small for segment a. This will prevent two circular pockets of different radius from getting clustered together. As a result, the approach will be able to correctly perform inter-segment clustering.

Choice of Window within Snippets

As the algorithm assigns similarity measure of two time series pairs based on subsequence matching, it is noticed that with variable window sizes and use of snippets, the MPdist becomes sensitive to sharp peaks/drops. This is illustrated using Figure 6.4. The approach tends to choose locally similar subsequences within a small window. Because of this, a circular pocket snippet is declared more similar to a face milling segment instead of another circular pocket.

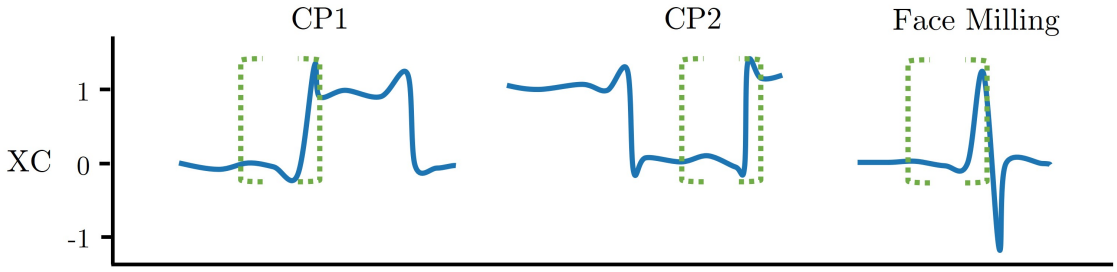

Figure 6.4: Z-normalized example snippets of two similar circular pocket (CP) segments and a face milling segment with the minimum MPdist window (green bracket) between the three snippets.

Choice of dimension

When working with multidimensional signals, the algorithm tends to choose a non-expressive signal. An example of a non-expressive signal can be ZC during CP expansion (see Figure 6.1). In that specific case, since the tool movement is only along x and y axis, ZC signal appears to be a "flat" line. Therefore, the elbow method should choose MPdist from XC or YC. However, when zoomed in (Figure 6.5), the ZC signal contains repeated noisy patterns that are exaggerated after z-normalization, and are equally considered by the algorithm.

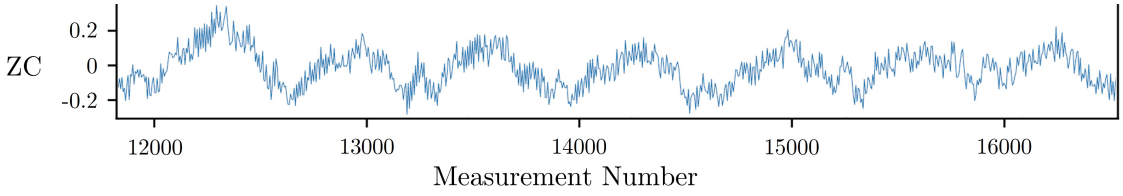

Figure 6.5: ZC signal of CP expansion segment from Demo Part Alpha dataset.

One way to prevent MPdist from choosing a dimension with non-expressive signal is by adding Gaussian noise to subsequences where the standard deviation is below a threshold. This will make the "flat" signal more dissimilar and consequently expressive signals similar. However, this solution is highly threshold sensitive. Additionally, it can add noise in between an expressive subsequence.

MPdist Re-computation for Classification

The biggest problem for using MPdist for the given application is that even after obtaining a clustering model from it, assigning labels to a new dataset at test time will be computationally expensive as this will require computation of distance of each new point with all the existing points in the MPdist matrix. This is highly infeasible from the application point of view given MPdist's performance.

Because of the mentioned problems, the updated MPdist approach is found unsuitable for the given application. Hence, the rest of the thesis explores and evaluates feature-based clustering.

6.3 Feature-Based Clustering Using TSFEL

Time series feature-based clustering involves extracting relevant features from the time series data and using these features to cluster the time series data into groups. Instead of individually calculating all possible features, Time Series Feature Extraction Library (TSFEL) provides an easy and efficient Python library to extract temporal, statistical and spectral features over a fixed time series window (see Figure 6.6). The full list of extracted features can be found in its paper [6]. However, feature-based clustering of time series data is sensitive to the choice of features, and the way they are extracted and transformed. Careful feature engineering is therefore crucial for obtaining meaningful and accurate clusters later. This subchapter explains the measures considered while preparing features for clustering.

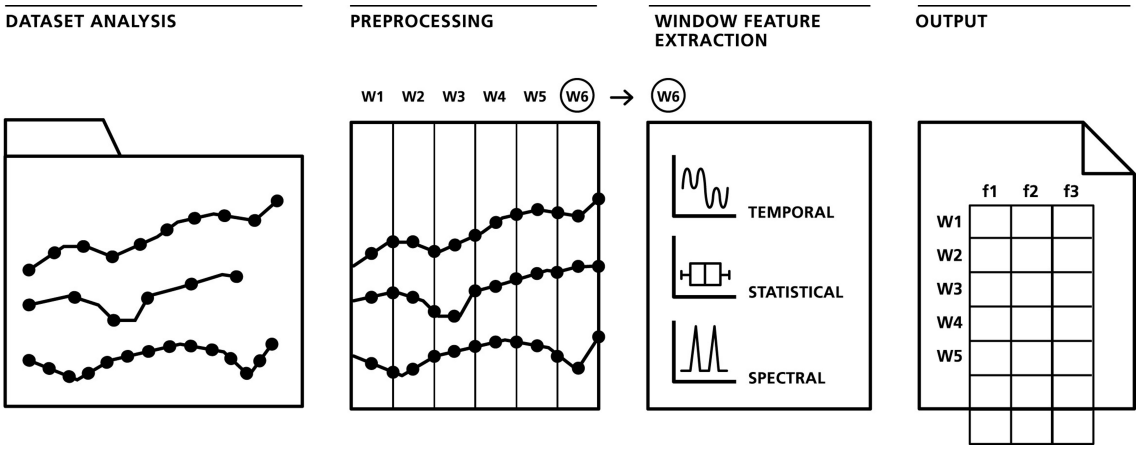

Figure 6.6: TSFEL pipeline [6]

6.3.1 Feature Selection

Feature selection in time series is the process of selecting a subset of features extracted from the time series data that are most relevant for clustering. Possible features obtained from TSFEL required careful selection. The objective is to select those features that follow three specific principles:

1. The extracted features should be window size independent. This way two circular pocket segments can be clustered together as long as the window covers the same fraction of pattern.

2. The extracted features should be time independent. For instance, the two circular pocket snippets seen in 6.4 should result in similar features even though they are time shifted.

3. The extracted features should have a high inter-segment difference and a low intra-segment variance.

It is observed that the spectral features do not follow the first criteria i.e window size independence. Even if the window covers the same fraction of a stretched pattern, the resulting frequency is different. Spectral features usually work well with larger and less noisy repetitive patterns. But in this case they result in oscillating features, for instance, the spectral kurtosis and wavelet mean as seen in Figure 6.7. Similarly, temporal features like entropy violated the third criterion. Including such features adds variance within a segment that could later affect the density of clusters.

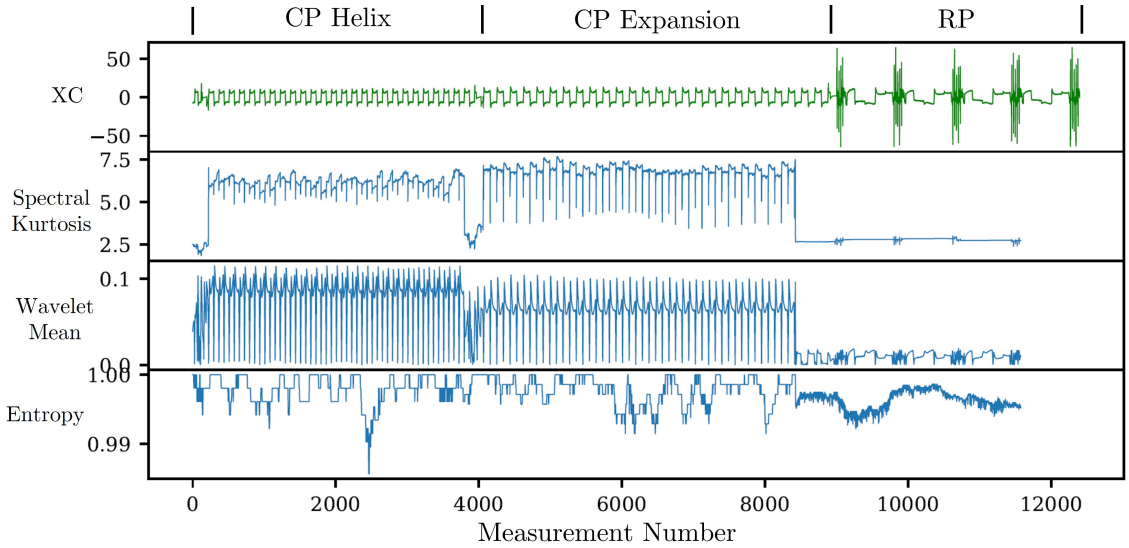

Figure 6.7: Drive current signal (green) from Demo Part Alpha and some of its extracted features that are excluded in the process of feature selection. For each of the three milling features, the window size is set to size of the segment snippet.

In addition to the mentioned features, correlated or low variance features like histogram and ECDF (Empirical Cumulative Distribution Function) coefficients are also

excluded. Finally, it is important to highlight that while sliding the window during feature extraction, the signal in each window is normalized. The reason for such normalization is to make patterns from different segments comparable, irrespective of the window size. Hence, mean dependent features like standard deviation, root mean square, etc. are also dropped.

The selected features listed in Table 6.1 follow the given criteria. From Figure 6.8, it can be observed that similar milling features (like CP Helix and CP Expansion) result in closer feature values despite having a different snippet length. However, it is important to note that the temporal features are window size dependent and they are normalized by dividing them by their respective window size, before comparing.

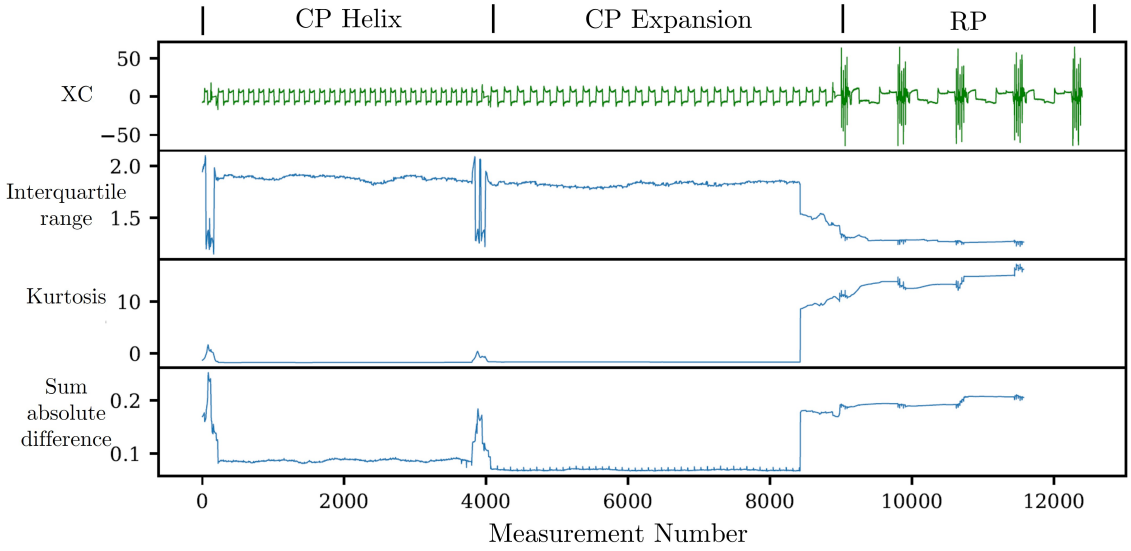

Figure 6.8: Drive current signal (green) from Demo Part Alpha and some of its selected features.

6.3.2 Window Size Setting for Feature Extraction

A key parameter in the TSFEL is the sliding window's size. As different milling features result in motifs of different lengths, poor choice of window size will greatly reduce the clustering accuracy later. Even same milling features that were milled under different settings, for instance two circular pockets with different radii, will have the same repeating motif but of different length.

Hence, in order to ensure a consistent coverage of patterns, the results from snippet discovery are used. The size of the sliding window is set to a multiple of the segment's snippet length as soon as the start of the window reaches the segment boundary. An important point to note here is that as long as the window covers exactly a multiple of the snippet size, the features within a segment remain constant. But the more the window's length deviates from the pattern's length, the more the features oscillate (Figure 6.9). This is because of the inconsistent number of peaks

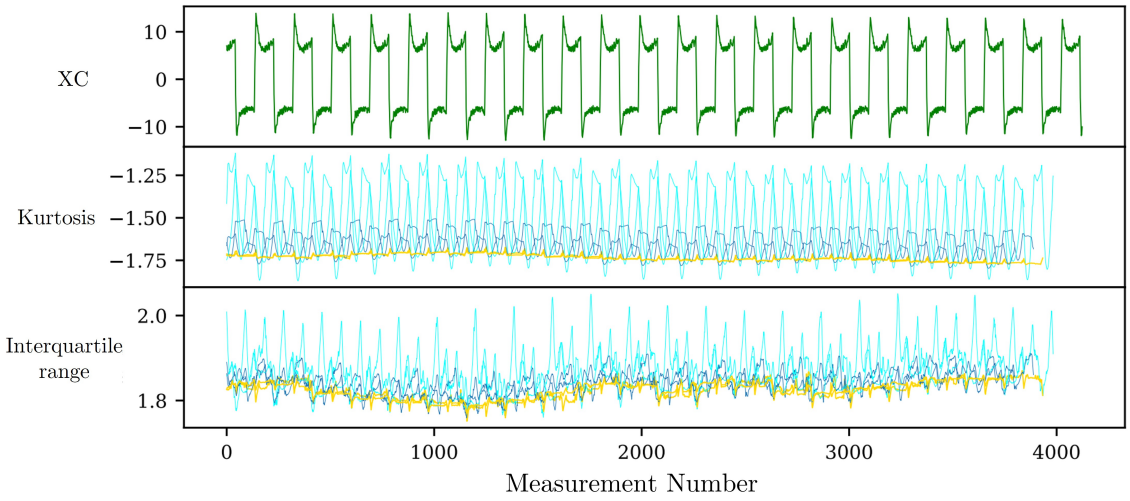

Figure 6.9: Drive current signal, in green, and its extracted features with varying window sizes between one to two times the snippet length. Yellow: multiples of snippet length. Blue: quarters of snippet length. Cyan: (one and a) half of snippet length.

being covered as the window slides across the signal.

The above mentioned finding shows the sensitivity of TSFEL to window size and the importance of extracting accurate snippets. However, accurate snippet discovery is difficult when working with scaling patterns like the one in Figure 6.3. In such cases, the window is never able to cover the same fraction of motifs in every step. Hence, there is a relatively high intra-segment variance which cannot be removed.

As the feature extraction can be a time taking process, it is greatly optimized by parallelly computing the features for each segment. Also, the overlap regions between two segments are skipped, This helps in reducing the noise in the feature vector and improves the computation time (see the blue noisy peeks in Figure 6.8).

Statistical Features	
Interquartile range	Midspread of the signal
Kurtosis	Weight of a distribution's tails relative to Normal distribution's tail
Max	Maximum value of the signal
Min	Minimum value of the signal
Mean absolute deviation	Mean of the absolute deviations from a central point
Peak to peak distance	Difference between the highest and the lowest value
Temporal Features	
Area under the curve	Area between a curve and the axis
Mean absolute diff	Average absolute difference of two independent values of the signal
Negative turning points	Number of negative turning points of the signal
Positive turning points	Number of positive turning points of the signal
Signal distance	Total distance traveled by the signal
Sum absolute diff	Sum of absolute differences between two independent values of the signal
Zero crossing rate	Number of times the signal crosses the horizontal axis

Table 6.1: Description of the selected features.

6.3.3 UMAP Dimensionality Reduction

UMAP (Uniform Manifold Approximation and Projection) is a dimensionality reduction technique that converts high dimensional data to a lower dimensional space, preserving the local structure. The idea behind using the UMAP here is to potentially improve the clustering model's accuracy and to be able to visualize the model input. For this purpose, the Python library umap-learn [1] is used, which enables training a reduction model. Later, in test phase, the new data is given to the same model to transform it into the learned space.

Feature extraction is performed inside the parallel pipeline for every process. However, during the training phase, once the features are extracted for all of the processes, they are concatenated to form a single feature matrix, which becomes input to the UMAP model.

There are several hyperparameters that can be adjusted when using UMAP. The two hyperparameters that have the most significant impact on the resulting embedding, given the extracted feature vectors, are the following:

n_neighbors: The number of nearest neighbors to use when constructing the graph of the data. This parameter determines the balance between preserving local and global structure in the data. A low value for n_neighbors causes UMAP to focus on the local structure of the data, potentially at the expense of the overall structure, while a high value causes it to consider larger neighborhoods of each point (Figure 6.10).

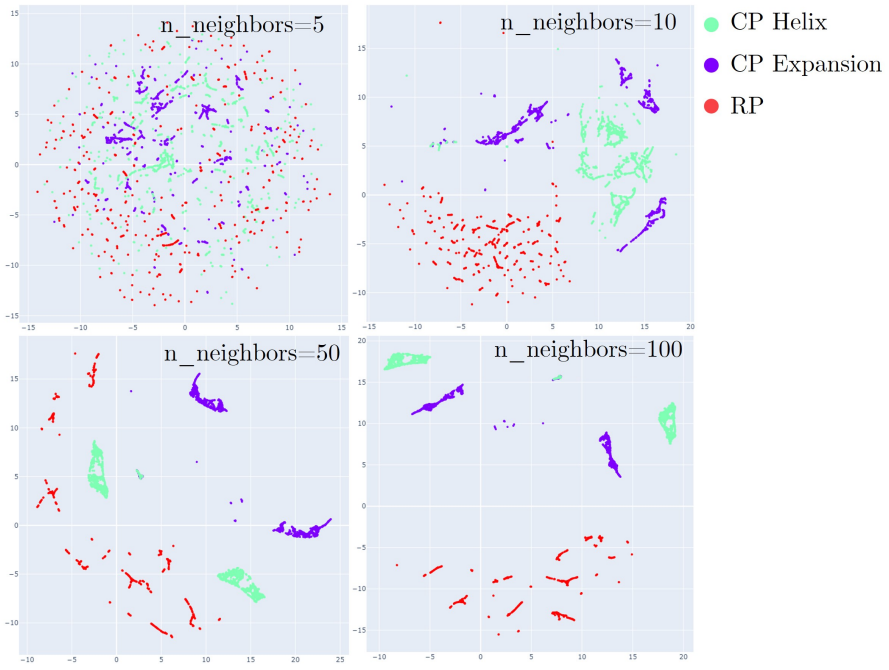

Figure 6.10: Effect of n_neighbors on features of three different segments.

min_dist: The minimum distance apart that points are allowed to be in the low dimensional representation. Larger values result in a more spread out, less crowded visualization, whereas smaller values give more compact embeddings (Figure 6.11).

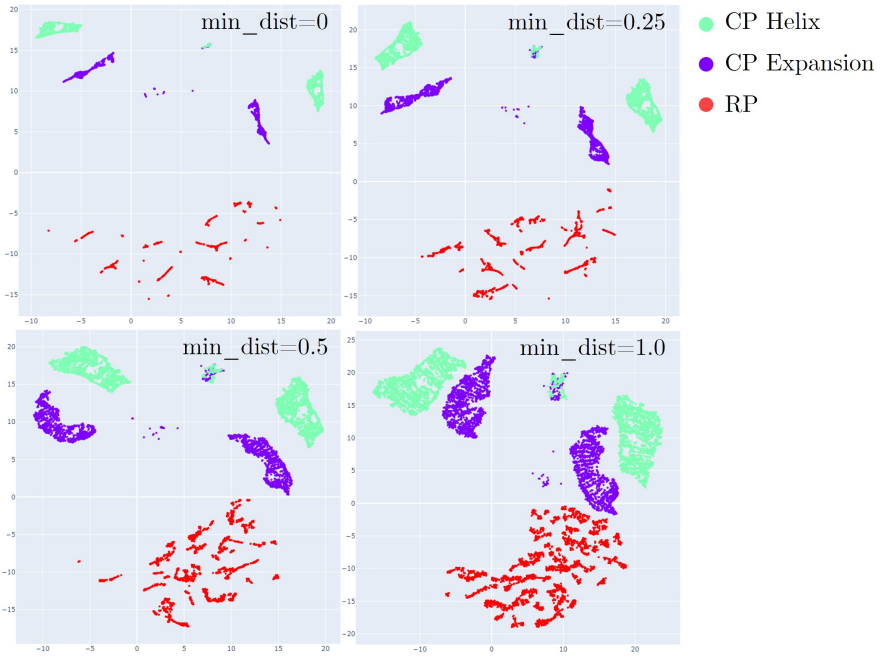

Figure 6.11: Effect of min_dist on features of three different segments.

The reduction model is saved locally for transforming new data at test time. It also contains the embeddings, which are now ready for training the clustering model.

6.4 HDBSCAN Training

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is a robust and efficient density-based clustering algorithm. After the extracted features are reduced to 2D by UMAP, they are passed to the clustering model. The Python implementation of HDBSCAN [37] is used for training the model.

The advantage of choosing HDBSCAN here for clustering this dataset is that it does not require the number of clusters to be predefined. Instead, it relies on a set of hyperparameters to control its behavior. The effect of different parameter settings on clusters are evaluated later in Chapter 8. However, some of the key parameters are briefly described below [37]:

min_cluster_size: The minimum number of points that a cluster must contain in order to be considered a valid cluster. This hyperparameter controls the granularity of the clusters.

min_samples: The minimum number of points that are required to form a density

cluster. It controls the sensitivity of the algorithm to noise and outliers.

metric: Metric to compute the distances. If a distance matrix is given already, then the parameter is set to 'precomputed'. In this case, the distance matrix is computed by the model using Euclidean distance.

alpha: The minimum fraction of the maximum distance between two points that is required for a cluster to be considered valid.

cluster_selection_epsilon: In some cases, a low value for min_cluster_size hyperparameter is needed to discover finer clusters. However, if the dataset contains regions with high densities of points, this can lead to the creation of many small "micro-clusters." To avoid this, the *cluster_selection_epsilon* hyperparameter is used, to merge clusters that are smaller than a certain threshold, which helps to prevent the further splitting of clusters in high density regions.

In addition to an HDBSCAN model, Hierarchical clustering is also implemented. However, the problem with such a clustering model is that after it is trained, the same model cannot be used to classify new data points. Every time a new dataset requires analysis, Hierarchical clustering needs to be performed again on all of the datasets. Additionally, the algorithm is unable to single out noisy regions, unlike HDBSCAN. Due to these limitations, the Hierarchical clustering approach is not evaluated in Chapter 8.

6.5 Classification of New Data Points

A major objective of the thesis is to be able to identify and classify different milling features in a process. However, because of absence of labels and the fact that generic textual labels (like circular pocket expansion or rectangular pocket) are highly subjective, a clustering model is trained to obtain numeric labels. These labels obtained from the clustering model are now used to classify segments in unseen processes. For this purpose, different classification models are included in the implementation.

6.5.1 HDBSCAN Prediction

Although HDBSCAN is a clustering algorithm, but the same model, once trained, can also be used to classify new data points based on which cluster they are most likely to belong to. This is a highly inexpensive operation in which the model determines where in the condensed tree the new data point will fall [37].

In order to use HDBSCAN for classification as well, an argument *prediction_data* must be set to $True$ at the time of training. Once the training is complete, the predict API *approximate_predict*() is called on the model with the test dataset. This returns predicted labels along with the probability matrix of each label being

chosen. A sample code snippet for this is given below:

```
# import HDBSCAN library
import hdbscan
# train clustering model
clusterer = hdbscan.HDBSCAN(min_cluster_size=5,
                            prediction_data=True)
                            .fit(train_data)
# classify new data points
test_labels, probabilities = hdbscan.approximate_predict(
                                clusterer, test_data)
```

Listing 1: Sample code snippet for HDBSCAN prediction.

6.5.2 Neural Network

As HDBSCAN is primarily a clustering algorithm, there was a need to include a proper classification algorithm as an option too. For this purpose, a neural network is included in the implementation using the Python library *keras* [20].

The feature embeddings obtained from the UMAP become input for the neural network. Whereas the output labels from HDBSCAN's training phase become training labels for the it (see Figure 6.12). As the clustering model gives a cluster label to each corresponding feature embedding row, it is possible that within a segment multiple clusters are discovered. However, since it is assumed that each time series segment contains a single milling feature, the discovered labels in each segment are overwritten by the majority label of that segment. Additionally, any segment where noise is the majority label is dropped. This clean-up step removes unwanted bias from the training data.

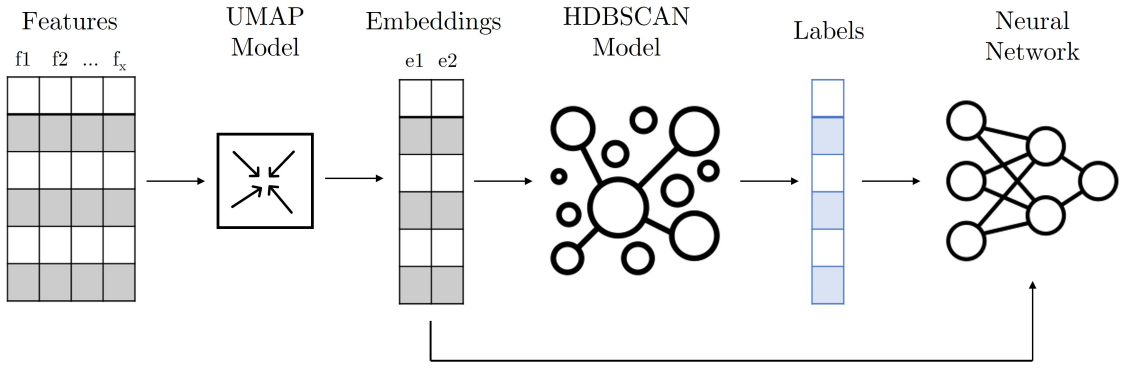

Figure 6.12: Workflow of the implemented models.

The implemented model (see Figure 6.13) is a simple neural network with 2 hidden layers, having 64 and 32 number of nodes respectively, with relu activation function. The output layer is a softmax layer for obtaining probability of each cluster label,

with the loss function being the multi-class cross-entropy loss (also known as softmax loss). The loss function in this case is formulated as:

$$CE = \sum_{i}^{k} t_i log(f(s)_i)$$

Here k is the number of unique labels obtained from the clustering model that are also included in the training phase of the classifier. t is the one-hot encoded ground truth label vector and s is the predicted probability vector.

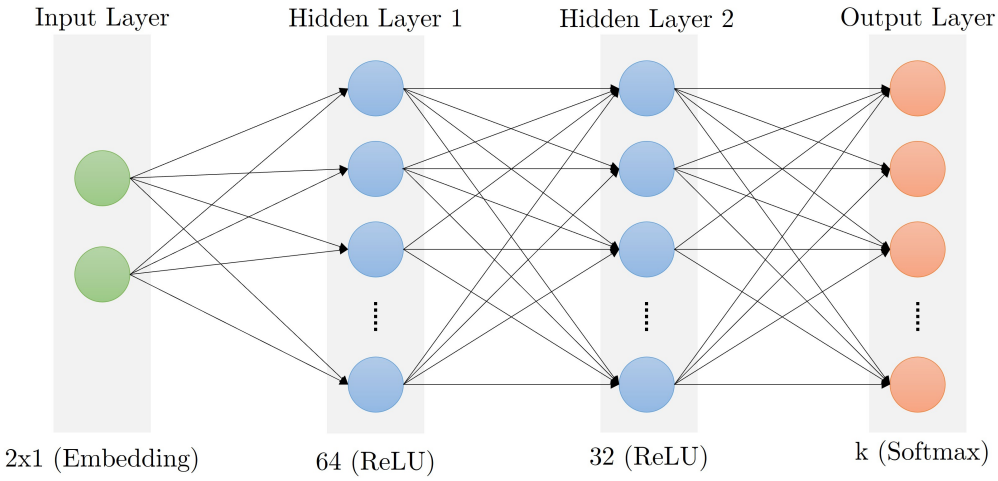

Figure 6.13: Architecture of the implemented neural network.

The trained neural network is also saved locally for test phase later. In addition, a label binarizer, that converts clustering labels into one-hot encoded matrix for neural network training is also saved. It is possible that some of the clustering labels might get dropped during the clean-up phase. Hence, it is essential to save the binarizer as well, to keep track of the labels used in classifier training.

7 Results Handling

A key objective of the thesis is to be able to store all the results generated from different microservices in the analytics pipeline and the classification model. This is the step four and five in the infrastructure workflow (Figure 4.1). The results include the discovered spindle speed partitions, regions of engagement, different feed rate regions and the predicted milling features in a new process using the trained classification model. Saving such results efficiently will enable an expert to filter multiple processes and gather insights about them.

This chapter explains the steps taken to save the calculated results and the techniques to efficiently query and combine results of multiple processes. First, the designed database schema is discussed in Chapter 7.1. Then, the new module implemented is explained that enables a user to filter and visualize the results (Chapter 7.2), and the back-end for handling the user query and interacting with the database to combine results (Chapter 7.3)

7.1 Database Schema Design

When it comes to saving any data in a database, the first question is the choice of the database. This decision depends on factors like data type, whether it is relational or non-relational, and the required operations. For this thesis, it is identified that all of the results from different microservices can be stored in a relational manner. Hence, PostgreSQL (or Postgres) is a strong candidate because it is an open source relational database system that provides extensions for efficient querying of time series and JSON data. Additionally, it complies with the existing database infrastructure at WZL, which is also using Postgres.

It is important to observe that all of the results generated from the microservices and the classification model have one thing in common. They all divide the time series into smaller chunks and provide a description of each partition. For instance, the spindle speed (SS) partition microservice divides the time series into partitions with constant SS along with the average SS in each partition. Similarly, the engagement detection microservice identifies partitions where the tool is in engagement with the workpiece, and the classification model identifies similar partitions by assigning a known label to it. Hence, each of these results can be generalized into rows containing start time of a partition, end time of the partition and additional information characterising the partition.

Based on the above observation, the database schema is designed (see Figure 7.1). The *microservice* table contains all the unique microservices along with the generated result's data types. The result's data types is used to build a dynamic dashboard (more on this in Chapter 7.2). For every unique process, identified by *trace_id*, the *partitions* table stores all the start and end bounds of partitions obtained from different microservices. The additional information for each partition, for example

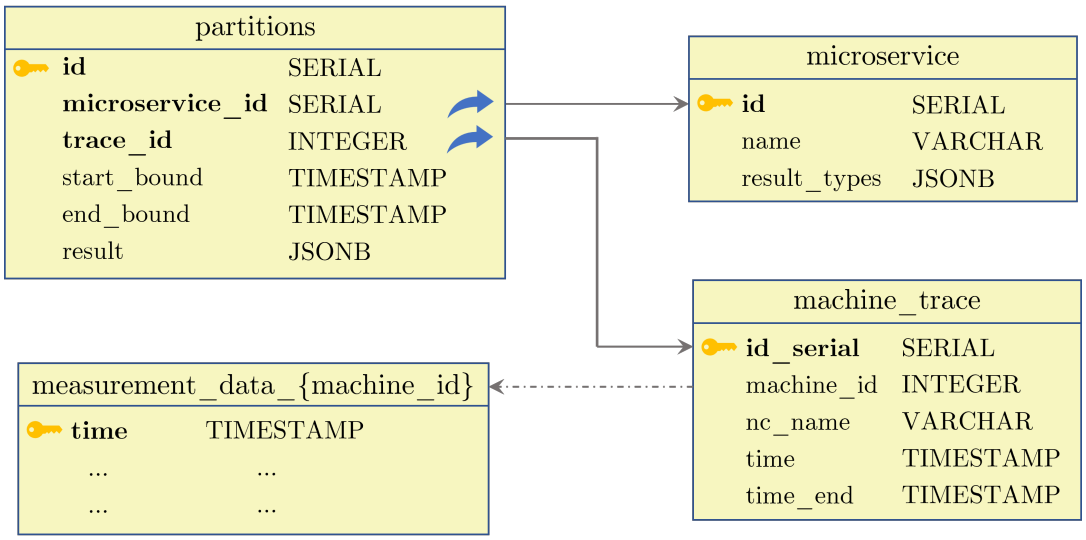

Figure 7.1: Implemented database schema for saving results in a generalized manner.

partition label in classification or the SS average from the SS microservice, is saved as a JSONB in the *result* column.

Using the foreign key relation between *partitions* table and *machine_trace*, the process' measurement data table is identified. Then, with the *start_bound* and the *end_bound* column, a specific partition's data in the *measurement_data* table is accessed.

The implemented database schema enables result storage in a highly generalized manner. This allows future addition of more microservices without any change to the existing structure. The only condition that the structure has is that the output of the microservice contains partition boundaries and any additional result is converted to a JSON format.

7.2 Streamlit Dashboard

In order to enable a user to filter and visualize the microservices result saved in the database, a dashboard is needed. For this purpose, an open source Python library Streamlit [92] is used. Using Streamlit, an interactive dashboard is built with custom widgets to input query settings (e.g. text and number input fields, checkboxes, drop-down menus, etc.) and Plotly plots to visualize the results.

As mentioned in Chapter 7.1, the database schema and the result structure is kept generalized. This allows future addition of more microservices to the pipeline without any change to the existing structure. The Streamlit dashboard is also built with the same objective. The settings panel of the dashboard (see Figure 7.2) is set up dynamically using the *result_types* column of the *microservice* table (Figure 5.1). For inclusion of every microservice, its expected result types (apart from start

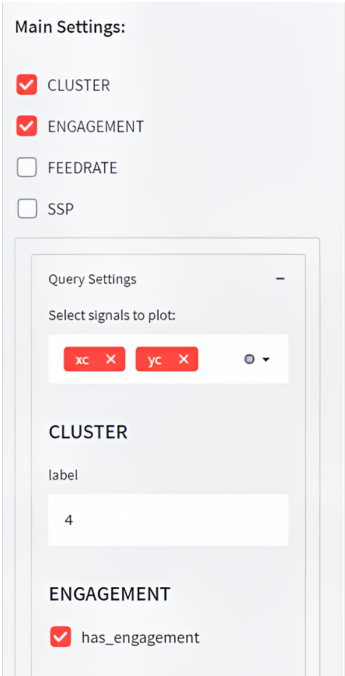

Figure 7.2: A screenshot of the settings panel of the Streamlit dashboard to filter microservices results and visualize the corresponding measurement data. The different microservice filters can be (de-)activated using the checkboxes at the top. Each microservice offers its own filter fields.

and end bounds) are stored in *microservice* table. Using that information, a dynamic settings panel of the dashboard is built, where a string data type has a text input field, boolean has a checkbox and a number results in a range field. Figure 7.1 illustrates the data stored in the *microservice* table.

id	name	result_types
1	ssp	{“type”: “str”, “average”: “float”}
2	engagement	{“has_engagement”: “bool”}
3	feedrate	{“type”: “str”, “average”: “float”}
4	cluster	{“label”: “str}

Table 7.1: *microservice* table containing each microservice’s name and its result’s data types in JSON format.

7.3 Multi-Level Querying

Using the dashboard settings panel (Figure 7.2), a user can trigger a query request that filters out measurement data partitions over multiple processes. An example of such a query is illustrated in Figure 4.2.

Once the query request is made by the user, an SQL query is generated dynamic-

ally to get the relevant partitions of measurement data using the *partitions* and the *microservice* table. For every enabled microservice in the dashboard, the *join* clause joins the *partitions* table with the *microservice* table to get the *microservice_id*, and the *where* clause of the query is appended with the filter conditions mentioned by the user. For the filter settings presented in Figure 7.2, the following SQL query is generated:

```
SELECT p.trace_id, m.id, p.start_bound, p.end_bound
FROM analytics.partitions AS p
JOIN analytics.microservice AS m
    ON m.id = p.microservice_id
WHERE
    (m.name = 'cluster' AND p.result->>'label' = '4') OR
    (m.name = 'engagement' AND p.result->>'has_engagement' = 'True')
ORDER BY start_bound;
```

Such a query returns all those partitions of processes that are either assigned the clustering label 4 or had engagement of the tool with the workpiece. The clustering label 4 in this example is a numeric label assigned by the clustering model to one of the discovered clusters of similar time series segments (more on this in Chapter 8). From the sample query presented above, multiple overlapping and possibly non-overlapping partitions are obtained using such a query. However, the user is only interested in the regions where all of the given conditions are fulfilled. In order to find those intersecting regions, the partition bounds are further filtered out and readjusted. The start and end bounds of each trace are merged, sorted and iterated only once to identify regions where all the filter conditions uphold. The pseudo code of such a function is given in Listing 2.

After filtering the partitions to find the intersecting regions, the measurement data corresponding to these regions is queried. This measurement data is then presented on the dashboard for the user. Specific query use cases and the corresponding response is presented later in Chapter 8.3.

```
Procedure:  get_intersecting_bounds
Input:      start_bounds: 1D array of process' start bounds
            end_bounds: 1D array of process' end bounds
            ms_id: 1D array of partition's microservice IDs
            enabled_ms_id: 1D array of enabled microservice
            IDs
Output:     filtered_start_bounds: 1D array of intersecting
            bounds
            filtered_end_bounds: 1D array of intersecting
            bounds
```

```
merged_bounds = concat(start_bounds, end_bounds)
sorted_bounds = sort(merged_bounds)
bounds_ms_ids = corresponding microservice IDs of
                sorted_bounds

filtered_start_bounds = []
filtered_end_bounds = []
current_bound = 0
overlapping_ms = []

for i, bound in enumerate(sorted_bounds)
    if bound is start bound and bound > current_bound
        current_bound = bound
        overlapping_ms.append(bounds_ms_ids[i])
    else if bound is end bound
        if overlapping_ms == unique_microservices
            filtered_start_bounds.append(current_bound)
            filtered_end_bounds.append(bound)
        overlapping_ms.remove(bounds_ms_ids[i])

return filtered_start_bounds, filtered_end_bounds
```

Listing 2: Algorithm for finding intersecting bounds.

8 Evaluation

The results of the thesis are categorized and evaluated in three subchapters. Chapter 8.1 presents the clustering results and analyzes the factors that majorly impact the identified clusters. These factors include the model's input format i.e the UMAP embeddings, the hyperparameters of the HDBSCAN clusterer and the effect of different milling techniques on results. Chapter 8.2 evaluates the classification of new milling features using both the trained HDBSCAN clusterer and the neural network. Finally, in Chapter 8.3, the implemented partitioning and filtering of results is also evaluated with the help of different use cases.

8.1 Clustering Model

The first step in the evaluation is the clustering results obtained from the HDBSCAN model. The model is trained on the reduced feature embeddings by the UMAP.

8.1.1 Datasets Used

For evaluation of this thesis, different demo parts are designed and run on the machine tools present at WZL shop floor. A description of all the parts and the resulting dataset obtained from them is presented in the Appendix A to D. Throughout this Chapter 8.1, datasets from the process Demo Part Alpha A.1 and Demo Part Beta B.1 are used as a running example. These datasets are simply referred to as Alpha and Beta from here on.

The two datasets are passed through the analytics pipeline presented in Chapter 6.1 to extract features. In Chapter 6.3.2, it is discussed that the window sizes for feature extraction need to be accurate and that the snippets for each segment are used as the window size. However, the obtained snippets are at times inaccurate (for example the rectangular pocket (RP) snippets in Figure 6.2). In order to independently evaluate the clustering model trained using the extracted features, snippets are manually identified for each segment in both of the training datasets (see Figure A.2 and B.2).

For every segment, the extracted features are split into training and cross-validation set. Because the segments contain highly repetitive patterns, only 20% of the features are used in training of the UMAP and the clustering model. This is intended to avoid overfitting.

8.1.2 Scoring Strategy

The clustering results are evaluated using two different metrics:

- **Accuracy**: The clustering accuracy, in this case, is defined as the deviation of identified cluster labels within a segment from the majority label of the segment. As the assumption is that each segment contains a single milling

feature, all of the predicted labels in that segment should be the same. Hence, a higher accuracy of the model means the cluster labels within a segment are the same.

- **DBCV Score**: The DBCV (Density-Base Clustering Validation) validates clustering assignments on non-globular, arbitrarily shaped clusters [67]. DBCV score (a number between -1 and 1) validates the model's ability to have high density within the clusters and low density between clusters.

In addition to the mentioned metrics, the result are also compared with the CAD models to validate whether the clusters correspond to the assigned milling features. However, it is important to note that the milling features labelled in Figure A.1 and B.1 are not the universal feature names and they can be subjective. They are only assigned for a better understanding of results.

8.1.3 Problems with UMAP

Chapter 6.3.3 explained how the high dimensional features extracted per process are combined and then reduced to a 2D embedding using UMAP. However, there are some major problems that are identified once the HDBSCAN clustering is performed on the embeddings. This is explained using the UMAP embeddings (Figure 8.1) and the corresponding clustering results of the training datasets (Figure 8.2 and 8.3).

In an effort to obtain a UMAP where the clusters and are distant from one another, the UMAP model is trained with a high $n\_neighbors$ value (1000) and the lowest possible min_dist (0). From the results, it is visible that the UMAP model fails to have consistent cluster labels within a same segment. An example of this is the face milling segment in both the datasets, where the segment contains a mix of cluster label 6 and 7. This is because the UMAP plane has divided the segment embeddings into two visible groups. This reduces the combined clustering accuracy to 85.7% for the two datasets.

A similar issue is noticed with the rectangular pocket (RP) of Alpha and two segments in Beta, where the UMAP transforms the segment features into scattered line-like groups all across the plane. This results in these entire segments being declared as noise by the HDBSCAN. Although, the HDBSCAN parameters can be adjusted to prevent it to declare the segments as noise, but the problem of scattered segments on UMAP will still persist.

A major problem with UMAP arises when the trained UMAP model is used to transform unseen data (i.e. cross-validation set or test set). The UMAP dimensionality reduction is already a computationally expensive and time taking task, however transforming new data with the trained UMAP takes even longer than the training time. Even though this is vaguely reported by the UMAP authors [60], such long transformation times are unexpected and infeasible from the application point of view.

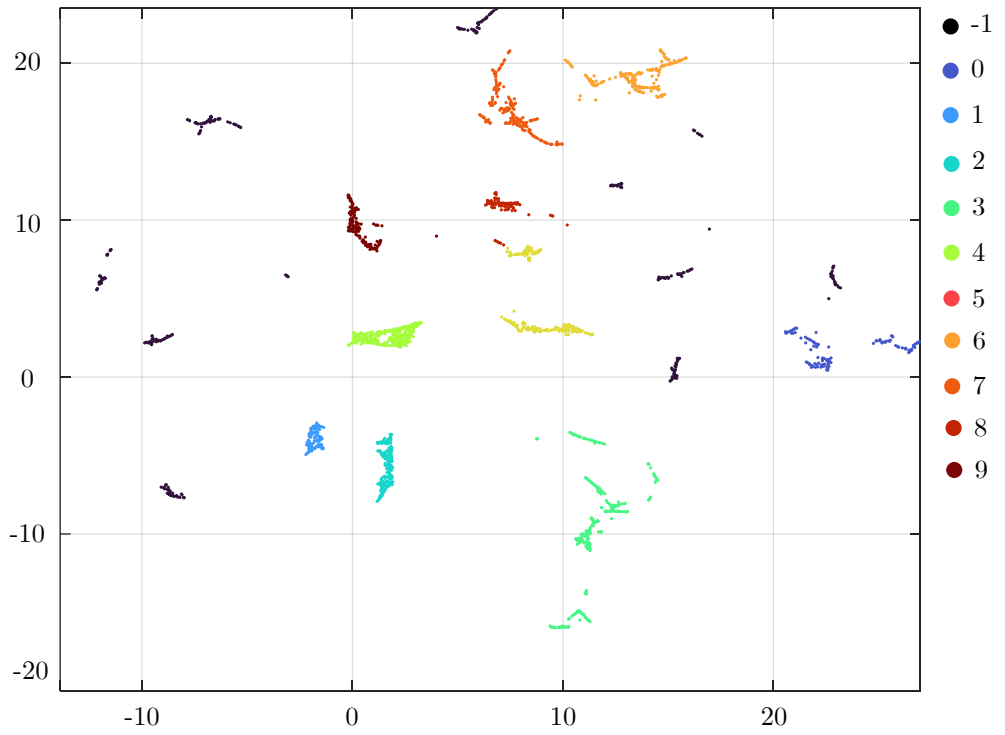

Figure 8.1: 2D UMAP plane of features from Alpha and Beta, with the color coded discovered clusters. The noise is represented with the label -1.

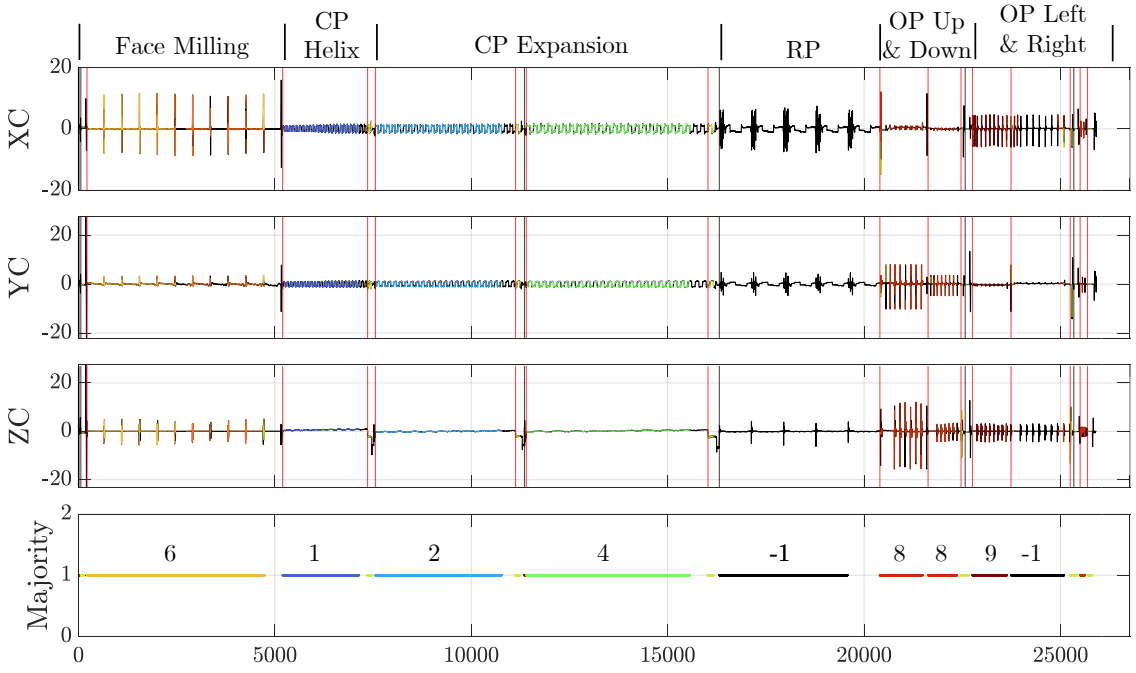

Figure 8.2: Discovered clusters using UMAP embeddings from Alpha. The model is trained with the drive current signals. The majority cluster labels per segment are shown in the bottom row.

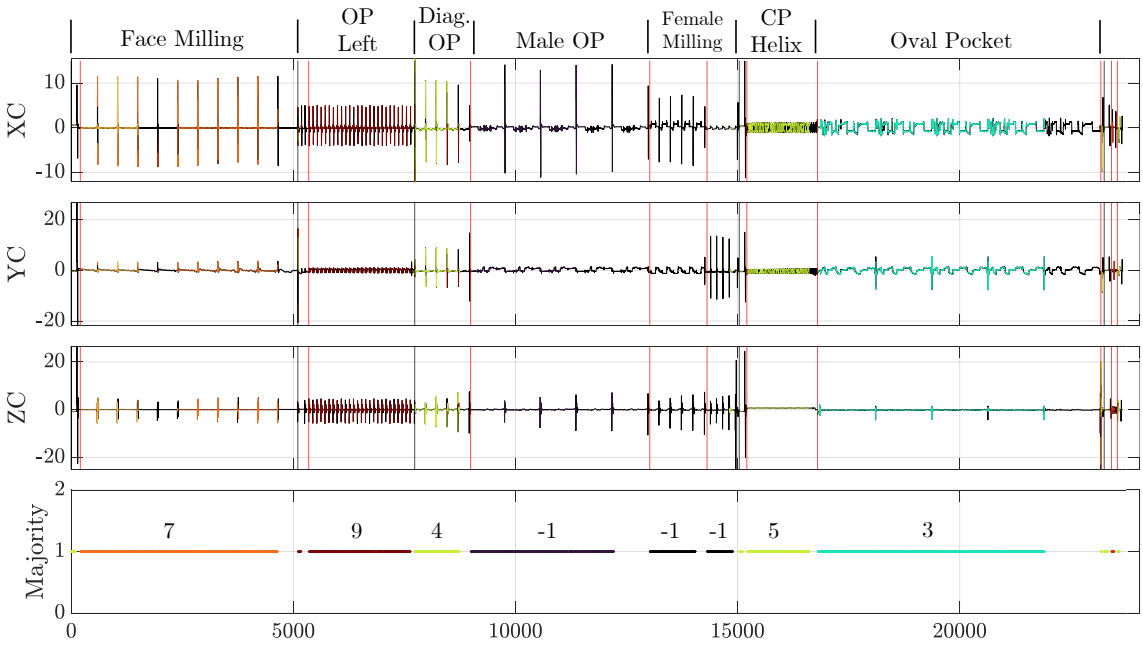

Figure 8.3: Discovered clusters using UMAP embeddings from Beta. The model is trained with the drive current signals. The majority cluster labels per segment are shown in the bottom row.

Because of the mentioned reasons, the UMAP is found to be an unnecessary overhead in the clustering task. Directly using the extracted features in the HDBSCAN is also possible. Results from such a model are evaluated next.

8.1.4 HDBSCAN on the Extracted Features

Using the extracted features for clustering without reducing the dimensions, requires a minimal change in the implementation. The HDBSCAN model is now trained on the extracted features directly instead of the embeddings from UMAP. The HDBSCAN hyperparameters have a major impact on the resulting clusters. For comparison, results from four different clustering models are shown below and their performance is summarized in Table 8.1.

Specialized Model: (Figure 8.4 and 8.5) The model is trained with hyperparameters: *min_cluster_size*:250, *min_samples*:1 and *metric*:*'euclidean'*. It is observed that HDBSCAN achieves more consistent clusters within a segment, resulting in a higher clustering accuracy of 95.3% with the features compared to 85.7% with the UMAP embeddings. Also, the DBCV score for this model is 0.43, ensuring that the clusters are dense and separated from one another. Lastly, the model is able to cluster exactly similar segments across datasets like face milling in Alpha and Beta. However, because of a low *min_cluster_size*, minor dissimilarities in milling features result in separate clusters. For instance, all of the circular pocket segments from the two datasets are in a cluster of their own. This is because different milling radius for each circular pocket segment results in a different pattern. The difference in pattern

makes the features between two segments dissimilar, even though the window size is set using the manually selected snippets. This shows that the feature-based clustering using snippets does not make the approach entirely window size independent.

Generalized Model: (Figure 8.6 and 8.7) In contrast to the fine-grained clusters presented above, it is possible to achieve more generalized clusters which gives an illusion of the clustering approach being window size independent. The hyperparameter *min_cluster_size* controls the level of generalization in the model. A higher value of 650 results in clusters where all of the general milling features like circular pockets and the outer profiles are clustered together. Whereas, unique segments like rectangular pocket in Alpha and oval pocket in Beta are in a cluster of their own. However, such a model results in a low DBCV score of 0.15, proving that the true clusters are actually the fine-grained ones.

DBCV Recommended Model: In addition to the manually selected HDBSCAN hyperparameters, an experiment is conducted to find the set of *min_cluster_size*, *min_samples* and *metric* that maximizes the DBCV score. The heat maps in Figure 8.8 illustrate such an experiment to visualize the DBCV score with different hyperparameter sets. It can be noticed that DBCV recommends to set a low *min_cluster_size* and a high *min_samples* for both the distance metrics. The highest score is achieved with the following setting: *min_cluster_size*:100, *min_samples*:95 and *metric*:$'manhattan'$. However, such a hyperparameter setting results in a low accuracy model with a high sensitivity to outliers, similar to the model obtained with UMAP embeddings (see Table 8.1 for comparison of performance).

Snippet Sampled Model: Instead of sampling a fixed fraction of features from each segment for training (e.g. 20%), it is also possible to sample points equal to the length of the snippet in a segment. Such a model is beneficial for improving the training time when working with several datasets. However, a problem with such a model in the current training setting is that some snippets are too small for the set *min_cluster_size* (e.g. circular pocket helix's snippet in Beta B.2), resulting in whole segments being labelled as noise.

Automatic Snippets Model: All of the models mentioned before are trained using manually extracted snippets in order to independently evaluate the clustering model. However, as the thesis proposes a whole infrastructure, the snippets calculated automatically from the analytics pipeline using the approach presented in Chapter 6.1.4 are also used to train a clustering model with a $min\_cluster\_size = 250$ (same as the specialized model). The results of such a model do not turn out to be as good as the specialized model but this is because some of the calculated snippet are of highly inconsistent length (see Alpha's snippets in Figure 6.2). This confirms feature-based clustering's reliance on meaningful window size setting.

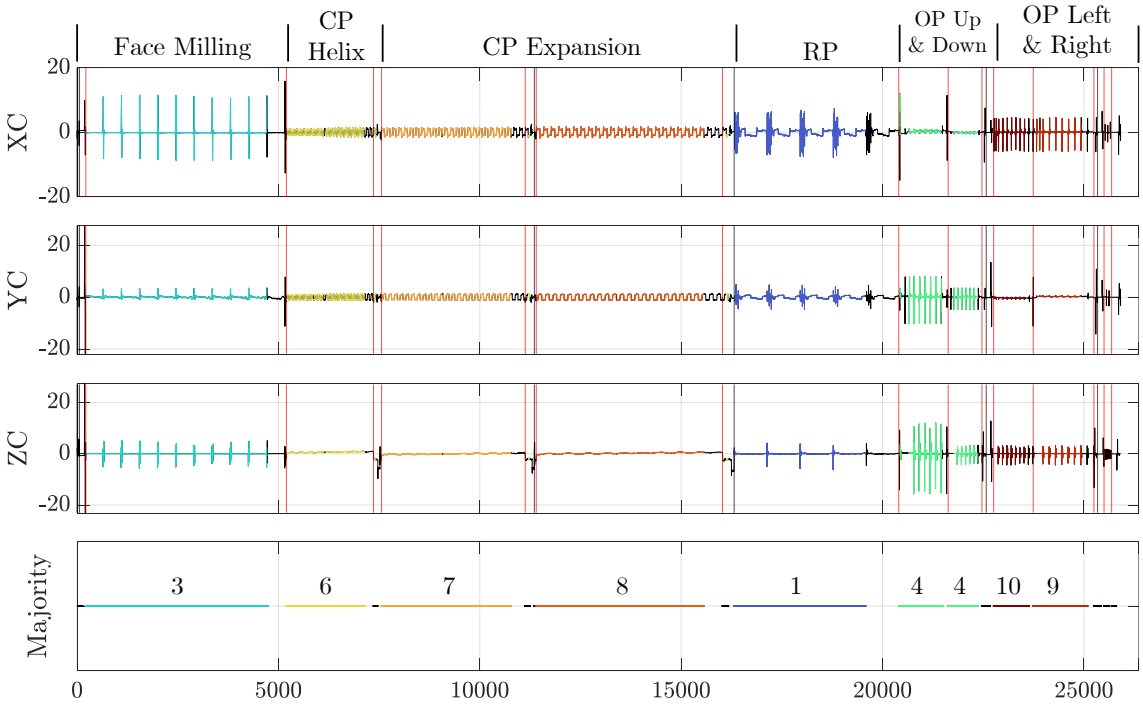

Figure 8.4: Discovered specialized clusters using the extracted features from Alpha. *min_cluster_size* is set to 250.

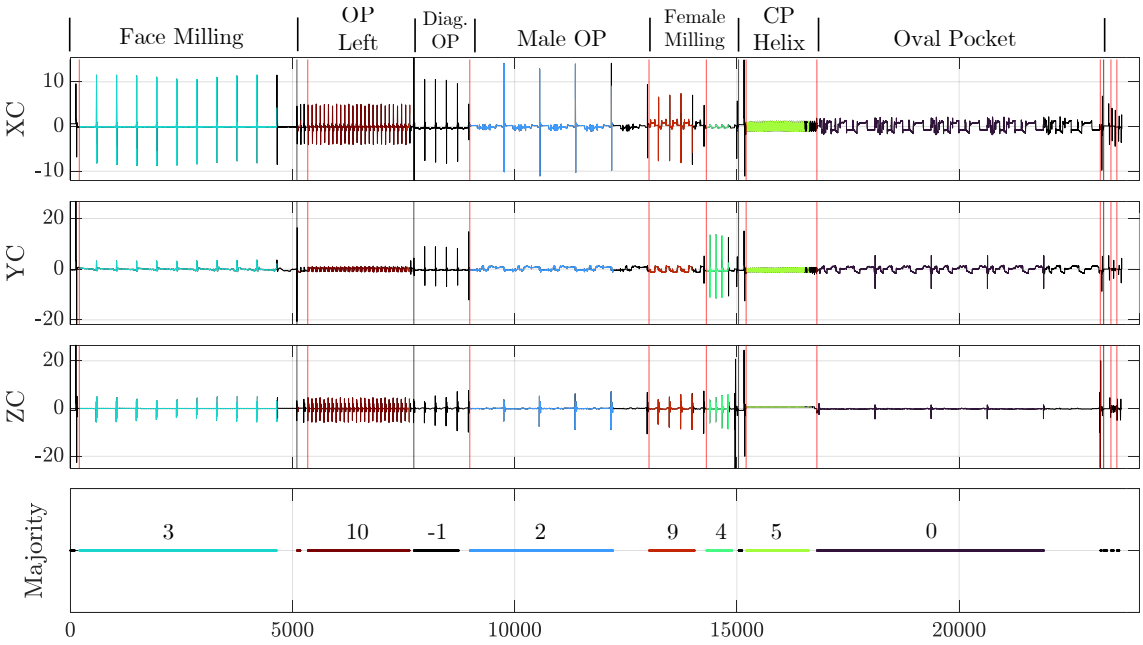

Figure 8.5: Discovered specialized clusters using the extracted features from Beta. *min_cluster_size* is set to 250.

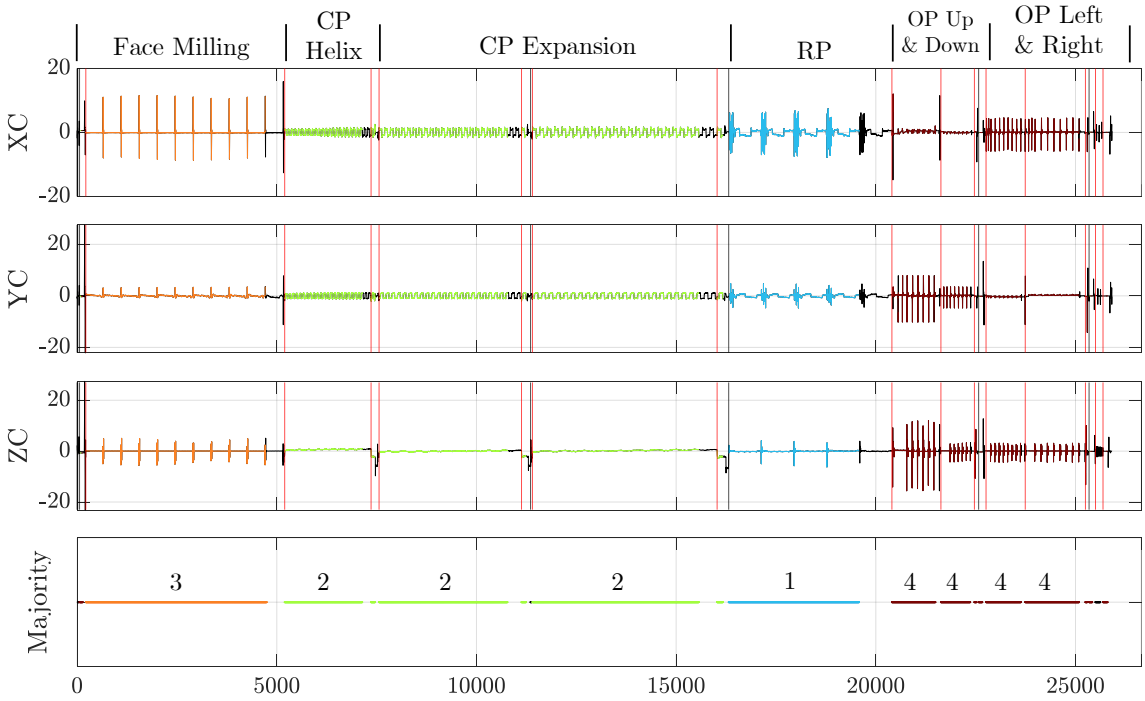

Figure 8.6: Discovered generalized clusters using the extracted features from Alpha. *min_cluster_size* is set to 650.

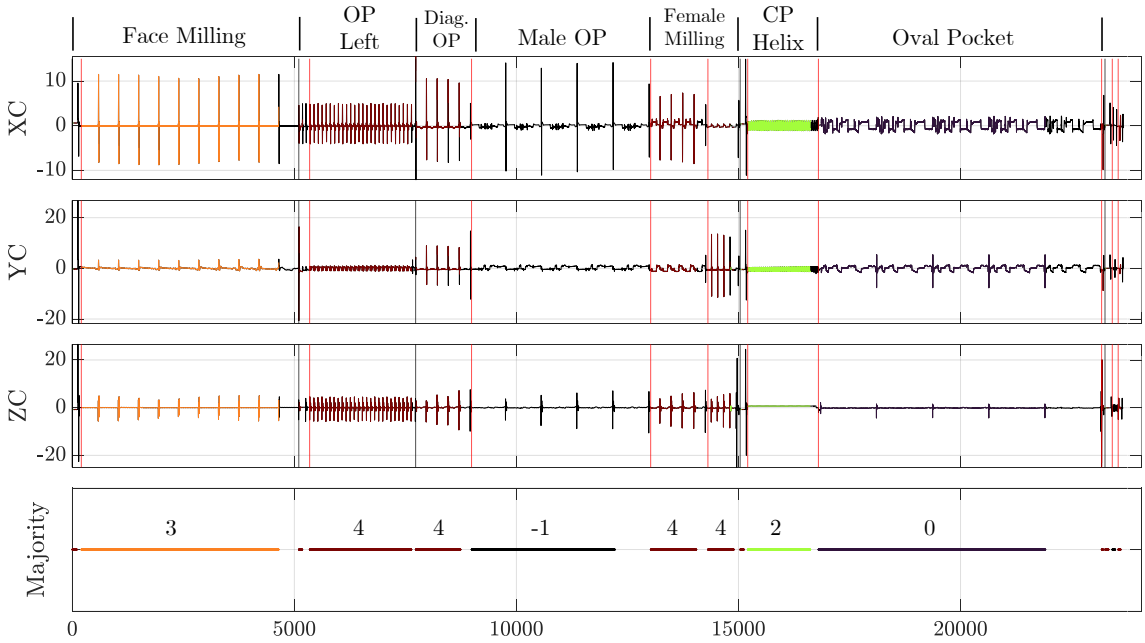

Figure 8.7: Discovered generalized clusters using the extracted features from Beta. *min_cluster_size* is set to 650.

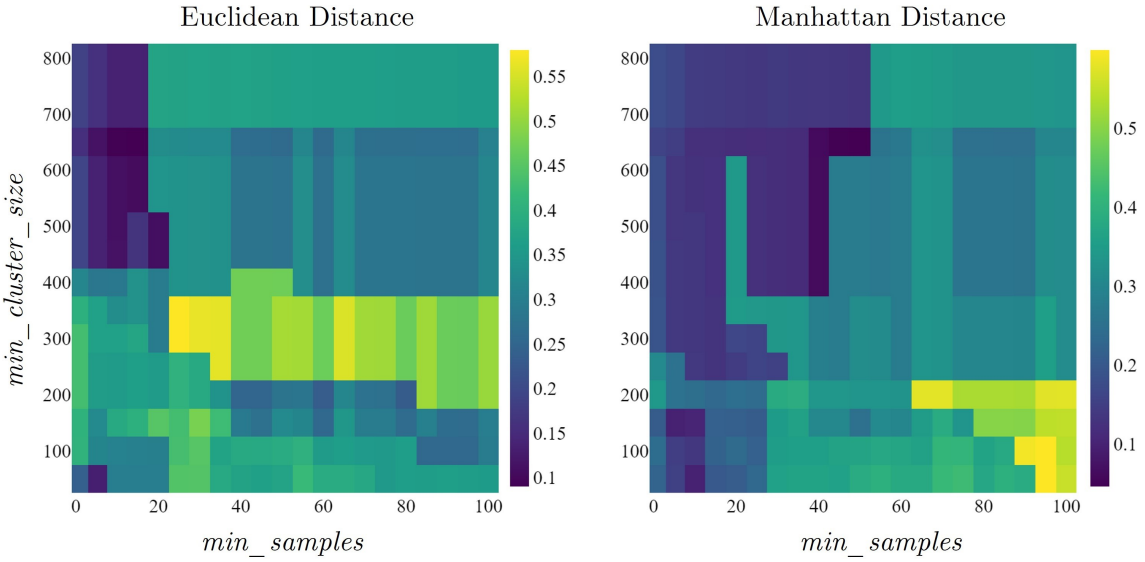

Figure 8.8: Heat map of DBCV scores with different HDBSCAN hyperparameter settings.

Model	**Accuracy (%)**	**DBCV Score**
Specialized	96.5	0.43
Generalized	**98.4**	0.15
DBCV Recommended	84.5	**0.59**
Snippet-based Sampling	88.8	0.42
Automatic Snippets	80.1	0.23
UMAP-based	85.1	0.53

Table 8.1: Comparison of different clustering models' performance.

From the presented models, it can be concluded that the clustering model should be trained directly on the extracted features for better results. Additionally, the hyperparameters need to be set manually depending upon the level of generalization or specialization desired. For example, a manufacturing company with a fixed set of milling features will be more interested in identifying the fine-grained clusters in their processes whereas a company with a wide range of similar milling features may wish to find abstract features obtained from a generalized model.

The presented clusters in Alpha and Beta from both the specialized and the generalized model fit well with their respective CAD models. That means they follow

the assumption of having a single milling in each segment by producing consistent labels within the segment. Additionally, all of the identified clusters are meaningful. For instance, in the generalized model, all of the outer profiles are together in one cluster and all of the circular pockets together in another. This validates the feature-based approach for clustering task with such datasets. Finally, the compression of the original datasets, the extraction of only the selected TSFEL features and the sampling of features results in a low training time of the HDBSCAN model. The given train/test split of 20/80 gives a training feature matrix of shape (8343, 27). This takes only 1.5 seconds to train the HDBSCAN model on an Intel Xeon CPU E5 with a 32GB RAM.

8.2 Classification Model

After evaluating the identified clusters in the dataset, the next step is to validate the possibility of classifying new data points. This involves evaluation of both the HDBSCAN's classification ability (Chapter 8.2.2) and the neural network's (Chapter 8.2.3). Chapter 3.3.3 presented several different clustering models, however, only the two promising models named specialized and generalized will be used from now.

8.2.1 Datasets Used

The datasets used to evaluate the HDBSCAN's classification are Demo Part Gamma and Demo Part Delta (or simply Gamma and Delta) presented in Figure C.1 and D.1. Similar to the clustering models' training process, the datasets are passed through the analytics pipeline presented in Chapter 6.1 to extract features. The only difference this time is that the dimensionality reduction step seen in Figure 6.12 is not performed. Instead, the features are directly passed to the trained HDBSCAN model. The dataset Gamma consists of milling features milled using the same strategies as in the training datasets whereas Delta is used to evaluate the specialized model's ability to identify milling features that are also present in the training dataset but are now milled using different milling strategy (e.g. face milling and 1st circular pocket's expansion). In addition, the dataset also contains a scaling circular pocket feature and entirely new features (e.g male jigsaw, female jigsaw and gravut).

For neural network classification, the implemented model seen in Figure 6.13 is first trained with Alpha and Beta's extracted features, along with the labels obtained from the HDBSCAN model being the target vector. But in order to remove any bias from the training data, the majority labels for each segment are used instead of the raw labels obtained. Additionally, any segments containing noise as the majority label (e.g. 'Male OP' segment seen in Figure 8.7) are filtered out from the training set. This prevents the neural network to learn from noise. Similar to HDBSCAN training, the neural network training also involves sampling of the extracted features, this time along with the target labels as well. It takes 5.2 seconds for the neural network to train on the feature matrix of shape (7864, 27), on an Intel Xeon E5 CPU. Once the neural network is trained, it is tested using the same Gamma

dataset for comparison with the HDBSCAN's classification. The test set also passes through the analytics pipeline to obtain features.

For classification using either HDBSCAN or neural network, the target labels of the test dataset are set manually with regards to the clustering model being used in training phase. For instance, if the generalized model is used to obtain labels for neural network training, then the target label for the circular pocket segment in Gamma will be 2. However for the specialized model, the target label setting is difficult. In that case, the train and test process' CAD models are analyzed to find the closest matches for each segment in the test dataset. For example, the helix and the expansion segment of the first circular pocket is assigned the target label 6 (same as Alpha's circular pocket helix (Figure 8.4)) because, the pocket was milled at a more similar radius and spindle speed than any other pocket in Alpha or Beta.

8.2.2 HDBSCAN's Performance

The classification model's performance on Gamma is evaluated using precision, recall and F1-score on the target labels obtained from the specialized and the generalized clustering model. Table 8.2 summarizes the results from the conducted experiments on Gamma.

During the experiments, a problem observed with the specialized HDBSCAN's classification is that the slightest variation in the test data from the training data results in many segments having noise as the majority label (see Figure 8.9). This is because HDBSCAN is trained in a way that it provides "hard" labels to each point i.e. either a single cluster label or noise. With the dense clusters obtained from the specialized model, any slightly deviating milling feature is declared as noise. Overall, 44.5% of the points in Figure 8.9 are declared to be not belonging to any of the clusters, which is a high ratio of noise.

The solution for such a problem is 'Soft clustering' [37], in which the points are not assigned cluster labels, but are instead assigned a vector of probabilities. The vector's length is equal to the number of clusters identified, and each entry represents the probability of the point being a member of that specific cluster. This allows points to potentially belong to multiple clusters. By analyzing the vector, the strength of a point's association with each cluster is analyzed. Points that are considered noise will have low probabilities of belonging to any cluster, but it's still possible to see which clusters they are closest to.

By training a Soft clustering HDBSCAN model (hereby referred to as HDBSCAN-Soft), the label with the maximum probability is assigned to its corresponding point. Figure 8.10 shows classification on an HDBSCAN-Soft model. It is noticed that even HDBSCAN-Soft fails to correctly classify some of the segments. For example, the first circular pocket segments are milled at more similar settings to "CP Helix" in Alpha, however the model perceives them closer to the "CP Expansion" segment

in Alpha (see Figure 8.4). Such an incorrect classification reduces the recall of the model (see Table 8.2).

With the obtained probabilities from HDBSCAN-Soft, more about the high presence of noise in Figure 8.9 is analyzed. It is calculated that the average of the maximum probabilities throughout the probability vector is only 0.47. Whereas, the median of the maximum probabilities is 0.22. This is significantly low given that the milling features in Gamma are similar to the ones in the training dataset. At first, this is perceived as if the clustering model is overfitting the training data. However, even with the generalized model, the average maximum probability is 0.36 and the median is 0.08. Additionally, the performance of both HDBSCAN and HDBSCAN-Soft remains insignificant for the generalized setting (see Table 8.2).

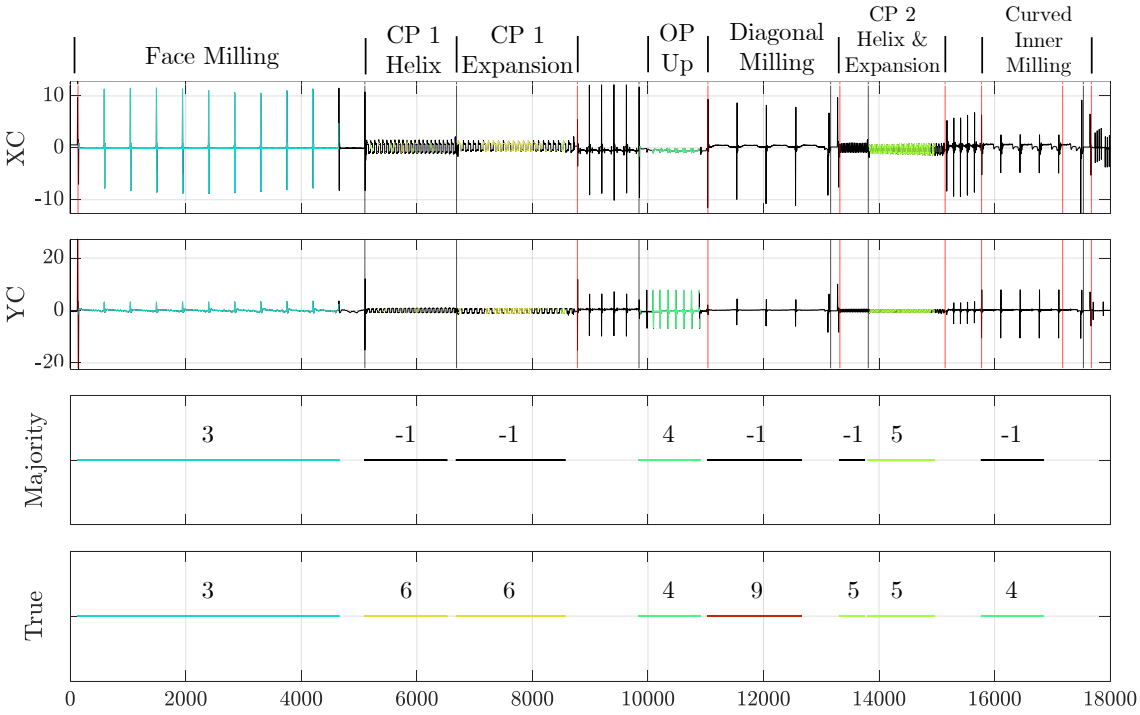

Figure 8.9: The classification results on the HDBSCAN model with Gamma as the test dataset. The majority labels per segment and the ground truth labels are displayed along with drive currents in X and Y direction.

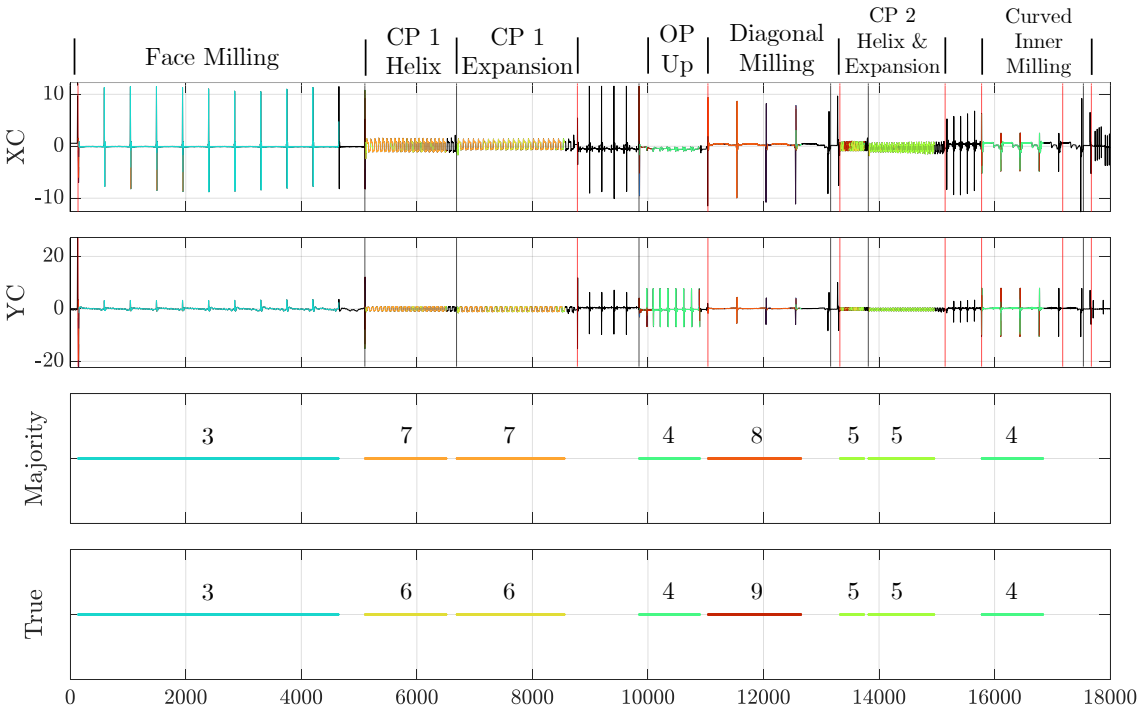

Figure 8.10: The classification results on the HDBSCAN-Soft model with Gamma as the test dataset.

8.2.3 Neural Network's Performance

The poor performance of HDBSCAN for classification task on Gamma leads to the conclusion that it is unstable and it is essential to evaluate a pure classification model like a neural network. A simple 2 hidden layer neural network performs significantly better when tested on both Gamma and Delta with labels obtained from both the specialized and the generalized model. For brevity, only the results obtained from the specialized clusterer labels are presented (see Figure 8.11 and 8.12). Also, the neural network's performance on both the datasets is summarized and compared with HDBSCAN in Table 8.2 and 8.3.

The Delta classification results (see Figure 8.12) show that the milling strategy is a key factor in identifying correct milling features because a change in milling strategy changes the resulting patterns in the signal. Examples of this is the face milling segment and the 1st circular pocket's expansion, which is milled with a different strategy compared to its counter parts in the training processes. Similarly the rectangular pocket milled with a different tool also changes the resulting pattern and hence, gets incorrectly classified. However, the model is able to correctly classify a scaling circular pocket segment. Although, as the circular pocket grows bigger, its pattern starts to become more similar to another circular pocket's. Nevertheless, the model is able to identify familiar features and the unseen features are assigned to the closest match based on their pattern.

From Table 8.2 and 8.3, it can be concluded that the addition of neural network in the infrastructure for classification is justified. It also highlights the indecisiveness of HDBSCAN when it comes to classification. Hence the HDBSCAN should be restricted to clustering similar milling features during the training phase of the infrastructure. The labels obtained from the clustering model can then be used to train the neural network, which will serve as the sole classification model in production.

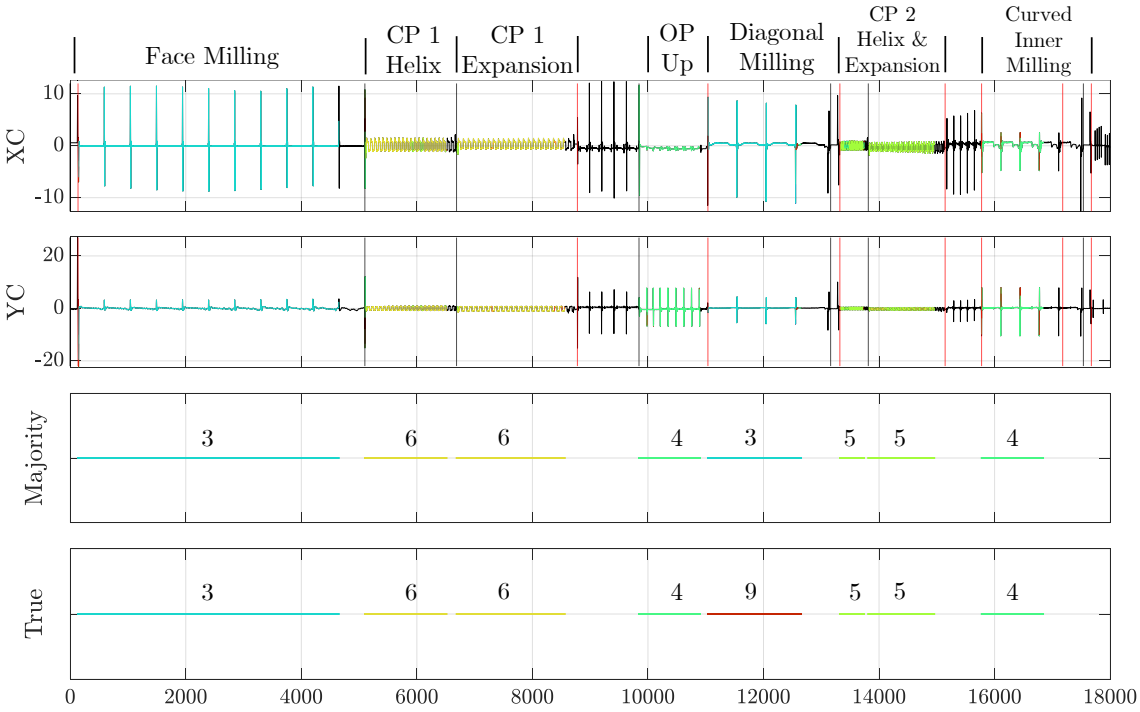

Figure 8.11: The classification results on the neural network model with Gamma as the test dataset. The model was trained on labels obtained from the specialized HDBSCAN clustering model.

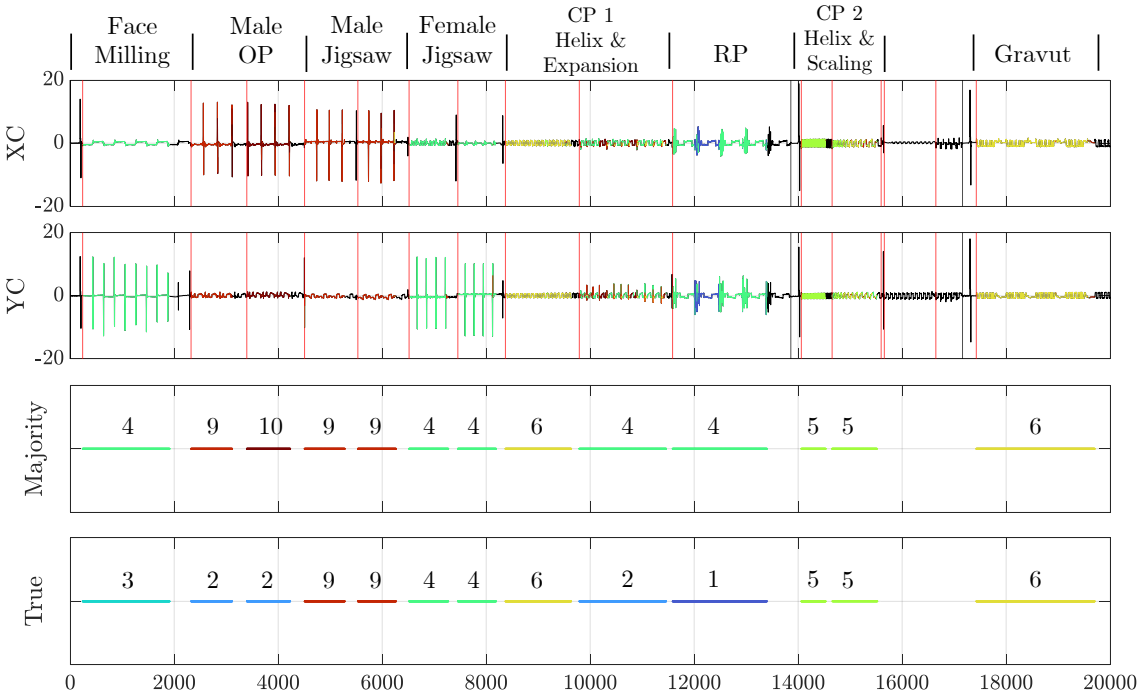

Figure 8.12: The classification results on the neural network model with Delta as the test dataset. The model was trained on labels obtained from the specialized HDBSCAN clustering model.

Classification	**Precision**	**Recall**	**F1-Score**
	Specialized model		
HDBSCAN	0.87	0.54	0.62
HDBSCAN-Soft	0.83	0.66	0.71
Neural network	0.84	0.84	**0.84**
	Generalized model		
HDBSCAN	0.99	0.77	0.85
HDBSCAN-Soft	0.67	0.64	0.65
Neural network	0.98	0.97	**0.97**

Table 8.2: Comparison of different classification models' performance on Gamma dataset using the target labels obtained from the specialized and generalized clustering model.

Classification	Precision	Recall	F1-Score
	Specialized model		
HDBSCAN-Soft	0.48	0.30	0.29
Neural network	0.49	0.51	**0.45**

Table 8.3: Comparison of neural network's performance on Delta dataset using the target labels obtained from the specialized clustering model.

8.3 Query Use Case

The dynamic database schema, the implemented multi-level querying and the Streamlit dashboard presented in Chapter 7 enable an expert to apply various filters and analyze partitions of different processes together. This subchapter presents a query use case similar to the one mentioned in Figure 4.2.

The results generated from the analytics pipeline and the generalized clustering model on Alpha and Beta are all saved in the database according to the schema presented in Chapter 7.1. Additionally, the pipeline results and the identified milling features in Gamma during the test process are also saved. Now an expert wants to visualize all the circular pocket segments that are manufactured at a spindle speed between 2500 to 3000. From the implemented Streamlit dashboard, the dynamic query generated for such a setting looks like the following:

```
SELECT p.trace_id, m.id, p.start_bound, p.end_bound
FROM analytics.partitions AS p
JOIN analytics.microservice AS m
ON m.id = p.microservice_id
WHERE
    (m.name = 'cluster' AND p.result->>'label' = '2') OR
    (m.name = 'ssp' AND p.result->>'type' = 'mf'
        AND CAST(p.result->>'average' AS FLOAT) BETWEEN 2500 AND 3000)
ORDER BY p.start_bound;
```

The bounds obtained from such a query are then further filtered to find overlapping regions among different microservices bounds. The measurement data corresponding to the overlapping region is then plotted on the Streamlit dashboard. Figure 8.13 presents screenshots of the obtained plots. The figure shows that the circular pocket helix and the first part of expansion segment in Alpha follows the set criteria. Additionally, the 1st expansion segment in Gamma is also included in the filtered bounds. The output generated is compared with the corresponding process' setting. It is confirmed that the three circular pocket segments were milled with a constant spindle speed of 2785rpm (see Table A.1 and C.1). This validates the result storage and querying strategy.

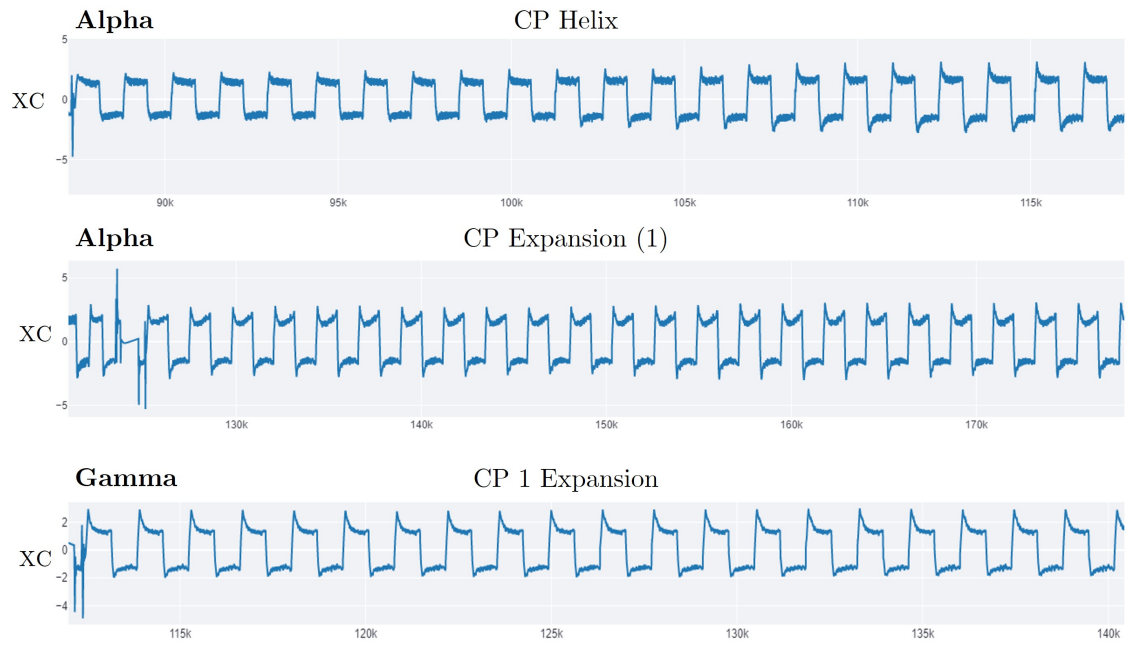

Figure 8.13: A modified screenshot of the segments obtained from the Streamlit dashboard. The applied filter condition is: All circular pockets regions in Alpha, Beta and Gamma that were manufactured at a spindle speed between 2500 and 3000.

As the designed dynamic schema does not require any SQL Join operations, and the queried bounds are only iterated once to identify overlapping regions, the process to obtain the relevant bounds is extremely efficient. It takes only 0.01 seconds to get the overlapping bounds for the above use case.

More such use cases can also be formed, involving engagement and feed rate filters. However, such use cases result in very fine-grained segments that are harder to interpret. Nevertheless, the possibility of scaling up the infrastructure by including more microservices and filters is highly convenient with the existing design.

9 Summary and Outlook

The thesis presents a novel infrastructure that breaks down the inflexible structure of a manufacturing process and facilitates identification of milling features. Additionally, it streamlines the process of data gathering and result storage in a way that it can be scaled up for various analysis. During this process, a suitable clustering strategy is developed that is able to perform well on high frequency, noisy time series data containing repetitive patterns.

The implemented infrastructure presents a data collection pipeline that gets triggered based on various user defined events, in a parallel manner. Such a pipeline enables an initial data partitioning based on dynamic events (e.g. tool change) and, at the same time, reduces the load on the database server by querying data in chunks.

The analytics pipeline includes triggering of multiple analysis algorithms for identifying and characterizing partitions in the milling processes. The pipeline is implemented in a way that more such algorithms can be added to it with minimal change. It also includes preprocessing steps like segmentation and snippet discovery, before performing clustering.

A major research question of the thesis was to compare shape-based and feature-based approaches for time series clustering. It is observed that the MPdist has major shortcomings that make it an unsuitable distance metric for multidimensional time series data. Because of the fact that the pattern in each segment is of different length, a single window size cannot possibly cover the time shifts of the patterns. Alternatively, a range of window size is possible if using just the snippets. However, with such small window sizes, the algorithm finds arbitrary matches across different milling features. Lastly, the options of training a model based on MPdist that can later be used for classification is overshadowed by the time complexity of multidimensional MPdist computation with variable window sizes.

On the other hand, feature-based clustering does not suffer from the shape-based approach's problems, and is able to identify and cluster similar milling features. The approach highlights the problems with the UMAP and concludes that the HDBSCAN can be performed directly on the extracted features without going through the overhead of performing dimensionality reduction using UMAP. Also, it is infeasible to test a trained UMAP model on unseen data. However, UMAP is a relatively new approach and might improve with time.

The feature-based approach with TSFEL is able to accurately cluster milling features of different sizes using HDBSCAN. However, the extracted features are highly dependent on accurate discovery of snippets in each segment. At the moment, this is not possible using the extended approach of Imani et al. [42]. But it is important to note that with the assumption of accurate snippets, the feature-based approach is able to achieve meaningful clusters on just the snippets instead of the entire dataset.

This hugely improves the feasibility of this approach in industry. However, a general problem with milling process data is that the official differentiation between milling features is not clearly defined. For instance, one expert will mark two circular pockets of different radius as two separate clusters whereas the other will want the two milling features to be in the same cluster. Hence it should be up to the expert to be able to achieve the desired level of generalization in clustering, and this is exactly what TSFEL features and HDBCSCAN clustering provides.

It is also found that the HDBSCAN, should not be used for classification. Instead, the label obtained from the clustering model can be used to train a simple two hidden layer network that is able to learn the representation and perform classification significantly better. However, the HDBSCAN's classification feature is relatively new and is expected to improve with time [37].

The identification and clustering of locally differentiated milling features reduces processing of hundreds of thousands of rows to a handful of partitions where each characterizes the original corresponding data in the partition. The implemented dynamic result storage schema offers an efficient solution to save both storage space and improve query time. Additionally, it gives an option to scale the existing infrastructure by adding more mircoservices as long as their output follows the set criteria.

The presented feature-based clustering can be applied to any other time series application with repeating patterns. As a future work, the existing feature-based clustering can be improved further to remove the dependency on accurate snippet discovery. Also, the existing approach works better with repeating patterns of fixed length in each segment. However, with scaling patterns like the one seen in Delta (see Figure 8.12), the feature-based approach's performance starts to deteriorate. This is something that can be worked on in the future.

The implemented infrastructure uncovers great potential for optimizing manufacturing processes. It can be extended to include a productivity optimization database that will offer improvement suggestions for a running process based on past process improvements. An example scenario is: If feed rate optimizations were performed for circular pockets under specific spindle speed and engagement conditions, this information can be stored in an optimization database. If another process is found with circular pocket under similar settings, a lookup is made in the optimization database for any process improvements. This will enable industries to perform runtime improvements to their processes and thus improve productivity.

References

[1] S. Aghabozorgi, A. Seyed Shirkhorshidi and T. Ying Wah. 'Time-series clustering – A decade review'. In: *Information Systems* 53 (2015), pp. 16–38. ISSN: 03064379. DOI: 10.1016/j.is.2015.04.007.

[2] M. Aharon, M. Elad and A. Bruckstein. 'K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation'. In: *Signal Processing, IEEE Transactions on* 54 (Dec. 2006), pp. 4311–4322. DOI: 10.1109/TSP.2006.881199.

[3] S. Alaee et al. 'Features or Shape? Tackling the False Dichotomy of Time Series Classification'. In: *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, pp. 442–450.

[4] K. Alexopoulos, N. Nikolakis and G. Chryssolouris. 'Digital twin-driven supervised machine learning for the development of artificial intelligence applications in manufacturing'. In: *International Journal of Computer Integrated Manufacturing* 33.5 (2020), pp. 429–439. DOI: 10.1080/0951192X.2020.1747642.

[5] M. Allaoui, M. L. Kherfi and A. Cheriet. 'Considerably Improving Clustering Algorithms Using UMAP Dimensionality Reduction Technique: A Comparative Study'. In: *Image and Signal Processing*. Springer International Publishing, 2020, pp. 317–325. ISBN: 978-3-030-51935-3.

[6] M. Barandas et al. 'TSFEL: Time Series Feature Extraction Library'. In: *SoftwareX* 11 (2020), p. 100456. ISSN: 23527110. DOI: 10.1016/j.softx.2020.100456.

[7] A. Barnett, J. Magland and L. af Klinteberg. 'A Parallel Nonuniform Fast Fourier Transform Library Based on an "Exponential of Semicircle" Kernel'. In: *SIAM Journal on Scientific Computing* 41 (Jan. 2019), pp. C479–C504. DOI: 10.1137/18M120885X.

[8] C. Bauer et al. 'Big Data in manufacturing systems engineering – close up on a machine tool'. In: *at - Automatisierungstechnik* 64.7 (2016), pp. 534–539. ISSN: 0178-2312. DOI: 10.1515/auto-2016-0022.

[9] D. J. Berndt and J. Clifford. 'Using dynamic time warping to find patterns in time series.' In: *KDD workshop*. Vol. 10. Seattle, WA, USA: 1994, pp. 359–370.

[10] T. Borgi et al. 'Big Data for Operational Efficiency of Transport and Logistics: A Review'. In: *2017 6th IEEE International Conference on Advanced Logistics and Transport (ICALT)*. IEEE, 2017, pp. 113–120. ISBN: 978-1-5386-1623-9. DOI: 10.1109/ICAdLT.2017.8547029.

[11] C. Brecher et al. 'Internet of Production - Turning Data into Value: Analytics in der Produktion'. In: *Statusberichte aus der Produktionstechnik*, pp. 186–208.

[12] C. Brecher et al. 'Internet of Production für agile Unternehmen: Lernende Produktionssysteme'. In: *AWK Aachener Werkzeugmaschinen-Kolloquium*. 2017, pp. 135–161.

[13] C. Brecher et al. *Lernende Produktionssysteme*. 1. Auflage. Aachen: Apprimus Verlag, 2017. ISBN: 3863595122.

[14] C. Brecher, J. Ochel and M. Fey. 'Datengetriebene Werkzeugeingriffsdetektion für Fräsprozesse'. In: *Zeitschrift für wirtschaftlichen Fabrikbetrieb* 117.11 (2022), pp. 784–789. DOI: doi:10.1515/zwf-2022-1146. URL: https://doi.org/10.1515/zwf-2022-1146.

[15] C. Brecher and M. Weck. *Werkzeugmaschinen Fertigungssysteme 3*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021. ISBN: 978-3-662-46568-4. DOI: 10.1007/978-3-662-46569-1.

[16] C. Brecher et al. 'Estimation of the virtual workpiece quality by the use of a spindle-integrated process force measurement'. In: *CIRP Annals* 68.1 (2019), pp. 381–384. ISSN: 00078506. DOI: 10.1016/j.cirp.2019.04.020.

[17] C. Brecher et al. *Lernende Produktionssysteme*. 1. Auflage. Aachen: Apprimus Verlag, 2017. ISBN: 3863595122.

[18] V. Cheepurupalli et al. 'Comparison of SVD and FFT in Image Compression'. In: *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2015, pp. 526–530. DOI: 10.1109/CSCI.2015.56.

[19] Y. Chen et al. 'The UCR Time Series Classification Archive'. In: (2015).

[20] F. Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[21] A. K. Choudhary, J. A. Harding and M. K. Tiwari. 'Data mining in manufacturing: a review based on the kind of knowledge'. In: *Journal of Intelligent Manufacturing* 20.5 (2009), pp. 501–521. ISSN: 0956-5515. DOI: 10.1007/s10845-008-0145-x.

[22] B. Costa et al. 'Fault Classification on Transmission Lines Using KNN-DTW'. In: *International Conference on Computational Science and Its Applications*. July 2017, pp. 174–187. ISBN: 978-3-319-62391-7. DOI: 10.1007/978-3-319-62392-4_13.

[23] T. Dantas and F. Luiz. 'Improving time series forecasting: An approach combining bootstrap aggregation, clusters and exponential smoothing'. In: *International Journal of Forecasting* 34.4 (2018), pp. 748–761. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2018.05.006. URL: https://www.sciencedirect.com/science/article/pii/S0169207018300888.

[24] B. Denkena et al. 'Time Series Search and Similarity Identification for Single Item Monitoring'. In: *Production at the Leading Edge of Technology*. Ed. by B.-A. Behrens et al. Lecture Notes in Production Engineering. Cham: Springer International Publishing, 2022, pp. 479–487. ISBN: 978-3-030-78423-2. DOI: 10.1007/978-3-030-78424-9{\textunderscore}53.

[25] B. Denkena, V. Böß and P. M. Hoppe. 'Optimization of Non-Cutting Tool Paths'. In: *Modelling of Machining Operations*. Vol. 223. Advanced Materials Research. Trans Tech Publications Ltd, June 2011, pp. 911–917. DOI: 10.4028/www.scientific.net/AMR.223.911.

[26] M. W. Dorrity et al. 'Dimensionality reduction by UMAP to visualize physical and genetic interactions'. In: *Nature communications* 11.1 (2020), p. 1537. DOI: 10.1038/s41467-020-15351-4.

[27] F. Eichinger et al. 'A time-series compression technique and its application to the smart grid'. In: *The VLDB Journal* 24.2 (2015), pp. 193–218. ISSN: 1066-8888. DOI: 10.1007/s00778-014-0368-8.

[28] N. El abbadi et al. 'Image compression based on SVD and MPQ-BTC'. In: *Journal of Computer Science* 10 (Oct. 2014). DOI: 10.3844/jcssp.2014.2095.2104.

[29] E. Elhamifar, G. Sapiro and R. Vidal. 'See all by looking at a few: Sparse modeling for finding representative objects'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012*. Piscataway, NJ: IEEE, 2012, pp. 1600–1607. ISBN: 978-1-4673-1228-8. DOI: 10.1109/CVPR.2012.6247852.

[30] D. Folgado et al. 'Time Alignment Measurement for Time Series'. In: *Pattern Recognition* 81 (2018), pp. 268–279. ISSN: 00313203. DOI: 10.1016/j.patcog.2018.04.003.

[31] M. P. Foundation. *Plot Matrix Profile Discords - Anomalies*. URL: https://matrixprofile.docs.matrixprofile.org/examples/Plot_Discords_MP.html (visited on 24/03/2022).

[32] S. Gharghabi et al. 'Matrix Profile XII: MPdist: A Novel Time Series Distance Measure to Allow Data Mining in More Challenging Scenarios'. In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 965–970. ISBN: 978-1-5386-9159-5. DOI: 10.1109/ICDM.2018.00119.

[33] C. Gröger, F. Niedermann and B. Mitschang. 'Data Mining-driven Manufacturing Process Optimization'. In: *Proceedings of the world congress on engineering*. Vol. 3. July 2012.

[34] D. Guijo-Rubio et al. 'Time-Series Clustering Based on the Characterization of Segment Typologies'. In: *IEEE Transactions on Cybernetics* 51.11 (2021), pp. 5409–5422. DOI: 10.1109/TCYB.2019.2962584.

[35] D. Guijo-Rubio et al. 'Time-Series Clustering Based on the Characterization of Segment Typologies'. In: *IEEE transactions on cybernetics* 51.11 (2021), pp. 5409–5422. DOI: 10.1109/TCYB.2019.2962584.

[36] J. Harguess and J. K. Aggarwal. 'Semantic labeling of track events using time series segmentation and shape analysis'. In: *2009 16th IEEE International Conference on Image Processing (ICIP)*. 2009, pp. 4317–4320. DOI: 10.1109/ICIP.2009.5413671.

[37] hdbscan. *The hdbscan Clustering Library*. URL: https://hdbscan.readthedocs.io/en/latest/ (visited on 10/04/2022).

[38] Heller. *Heller 5-axis machining centers HF*. URL: https://www.heller.biz/de/maschinen-und-loesungen/5-achs-bearbeitungszentren-hf/ (visited on 05/12/2022).

[39] J. Himberg et al. 'Time series segmentation for context recognition in mobile devices'. In: *Proceedings 2001 IEEE International Conference on Data Mining*. 2001, pp. 203–210. DOI: 10.1109/ICDM.2001.989520.

[40] Z. Huang et al. 'Tool wear predicting based on multi-domain feature fusion by deep convolutional neural network in milling operations'. In: *Journal of Intelligent Manufacturing* 31.4 (2020), pp. 953–966. ISSN: 0956-5515. DOI: 10.1007/s10845-019-01488-7.

[41] Z. Huang et al. 'A Survey on AI-Driven Digital Twins in Industry 4.0: Smart Manufacturing and Advanced Robotics'. In: *Sensors (Basel, Switzerland)* 21.19 (2021). DOI: 10.3390/s21196340.

[42] S. Imani et al. 'Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining'. In: *2018 IEEE International Conference on Big Knowledge (ICBK)*. 2018, pp. 382–389. DOI: 10.1109/ICBK.2018.00058.

[43] F. Isensee et al. 'Automatic Cardiac Disease Assessment on cine-MRI via Time-Series Segmentation and Domain Specific Features'. In: *Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges*. Cham: Springer International Publishing, 2018, pp. 120–129. ISBN: 978-3-319-75541-0.

[44] S. Jamali et al. 'Detecting changes in vegetation trends using time series segmentation'. In: *Remote Sensing of Environment* 156 (2015), pp. 182–195. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2014.09.010. URL: https://www.sciencedirect.com/science/article/pii/S0034425714003538.

[45] A. Javed, B. S. Lee and D. M. Rizzo. 'A benchmark study on time series clustering'. In: *Machine Learning with Applications* 1 (2020), p. 100001. ISSN: 26668270. DOI: 10.1016/j.mlwa.2020.100001.

[46] U. Jayasankar, V. Thirumal and D. Ponnurangam. 'A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications'. In: *Journal of King Saud University - Computer and Information Sciences* 33.2 (2021), pp. 119–140. ISSN: 13191578. DOI: 10.1016/j.jksuci.2018.05.006.

[47] R. Jumar, H. Maaß and V. Hagenmeyer. 'Comparison of lossless compression schemes for high rate electrical grid time series for smart grid monitoring and analysis'. In: *Computers & Electrical Engineering* 71 (2018), pp. 465–476. ISSN: 00457906. DOI: 10.1016/j.compeleceng.2018.07.008.

[48] M. Jünger, G. Reinelt and G. Rinaldi. 'The traveling salesman problem'. In: *Handbooks in operations research and management science* 7 (1995), pp. 225–330.

[49] S. A. A. Karim et al. 'Wavelet Transform and Fast Fourier Transform for signal compression: A comparative study'. In: *2011 International Conference on Electronic Devices, Systems and Applications (ICEDSA)*. 2011, pp. 280–285. DOI: 10.1109/ICEDSA.2011.5959031.

[50] E. Keogh. 'A Decade of Progress in Indexing and Mining Large Time Series Databases.' In: *Proceedings of the 32nd International Conference on Very Large Data Bases*. Jan. 2006, p. 1268.

[51] E. Keogh et al. 'Segmenting Time Series: A Survey and Novel Approach'. In: *Data mining in time series databases*. Ed. by M. Last, A. Kandel and H. Bunke. Vol. 57. Series in machine perception and artificial intelligence v.57. New Jersey and London: WORLD SCIENTIFIC, 2004, pp. 1–21. ISBN: 978-981-238-290-0. DOI: 10.1142/9789812565402{\textunderscore}0001.

[52] A. Khan, K. Khan and B. B. Baharudin. 'Frequent Patterns Minning of Stock Data Using Hybrid Clustering Association Algorithm'. In: *2009 International Conference on Information Management and Engineering*. 2009, pp. 667–671. DOI: 10.1109/ICIME.2009.129.

[53] G. Lee et al. 'PyWavelets: A Python package for wavelet analysis'. In: *Journal of Open Source Software* 4.36 (2019), p. 1237. DOI: 10.21105/joss.01237. URL: https://doi.org/10.21105/joss.01237.

[54] L. Li et al. 'Quality Prediction and Control of Assembly and Welding Process for Ship Group Product Based on Digital Twin'. In: *Scanning* 2020 (2020), p. 3758730. DOI: 10.1155/2020/3758730.

[55] M. Lovric, M. Milanovic and M. Stamenkovic. 'Algorithmic methods for segmentation of time series: An overview'. In: *Journal of Contemporary Economic and Business Issues (JCEBI)* 1 (Jan. 2014), pp. 31–53.

[56] L. Lu and H.-J. Zhang. 'Automated extraction of music snippets'. In: *2003 multimedia conference*. Ed. by L. Rowe et al. New York, New York, USA: ACM Press, 2003, p. 140. ISBN: 1581137222. DOI: 10.1145/957013.957043.

[57] W. Luo, T. Hu and Y. Ye. 'A hybrid predictive maintenance approach for CNC machine tool driven by Digital Twin'. In: *Robotics and Computer-Integrated Manufacturing* 65 (2020), p. 101974. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2020.101974. URL: https://www.sciencedirect.com/science/article/pii/S0736584519306660.

[58] A. Malakhov et al. 'Composable multi-threading and multi-processing for numeric libraries'. In: *Proc. 17th Python Sci. Conf., no. Scipy*. 2018, pp. 18–24.

[59] B. Malley, D. Ramazzotti and J. T.-y. Wu. 'Data Pre-processing'. In: *Secondary Analysis of Electronic Health Records*. Cham: Springer International Publishing, 2016, pp. 115–141. ISBN: 978-3-319-43742-2. DOI: 10.1007/978-3-319-43742-2_12. URL: https://doi.org/10.1007/978-3-319-43742-2_12.

[60] L. McInnes, J. Healy and J. Melville. 'UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction'. In: (2018). URL: https://arxiv.org/pdf/1802.03426.

[61] A. Minnaar. *Time Series Classification and Clustering with Python*. URL: http://alexminnaar.com/2014/04/16/Time-Series-Classification-and-Clustering-with-Python.html (visited on 30/03/2022).

[62] K. Mohammed. 'JSON Integration in Relational Database Systems'. In: *International Journal of Computer Applications* 168 (2017), pp. 151–166. ISSN: 10848045.

[63] H.-C. Möhring et al. 'Self-optimizing machining systems'. In: *CIRP Annals* 69.2 (2020), pp. 740–763. ISSN: 00078506. DOI: 10.1016/j.cirp.2020.05.007.

[64] C. S. Möller-Levet et al. 'Fuzzy Clustering of Short Time-Series and Unevenly Distributed Sampling Points'. In: *Advances in intelligent data analysis V*. Ed. by M. Berthold. Vol. 2810. Lecture notes in computer science, 0302-9743. Berlin and London: Springer, 2003, pp. 330–340. ISBN: 978-3-540-40813-0. DOI: 10.1007/978-3-540-45231-7{\textunderscore}31.

[65] P. Montero and J. A. Vilar. 'TSclust : An R Package for Time Series Clustering'. In: *Journal of Statistical Software* 62.1 (2014). DOI: 10.18637/jss.v062.i01.

[66] F. Mörchen. 'Time series feature extraction for data mining using DWT and DFT'. In: 2003.

[67] D. Moulavi et al. 'Density-Based Clustering Validation'. In: *Proceedings of the 2014 SIAM International Conference on Data Mining (SDM)*. Apr. 2014, pp. 839–847. DOI: 10.1137/1.9781611973440.96. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611973440.96.

[68] R. O. Obe and L. S. Hsu. *PostgreSQL: Up and running : a practical guide to the advanced open source database / Regina O. Obe and Leo Hsu*. Third edition. Sebastopol, CA: O'Reilly, 2018. ISBN: 9781491963418.

[69] J. Ochel, M. Fey and C. Brecher. 'Semantically Meaningful Segmentation of Milling Process Data'. In: *Production at the Leading Edge of Technology*. Ed. by B.-A. Behrens et al. Lecture Notes in Production Engineering. Cham: Springer International Publishing, 2022, pp. 319–327. ISBN: 978-3-030-78423-2. DOI: 10.1007/978-3-030-78424-9{\textunderscore}36.

[70] A. Patel. 'Optimization of Milling Process Parameters - A Review'. In: *International Journal of Advanced Research in Engineering and Applied Sciences*. Vol. 4, pp. 24–37. (Visited on 2015).

[71] C. Pealat, G. Bouleux and V. Cheutet. 'Improved Time-Series Clustering with UMAP dimension reduction method'. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 5658–5665. DOI: 10.1109/ICPR48806.2021.9412261.

[72] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[73] F. Petitjean, A. Ketterlin and P. Gançarski. 'A global averaging method for dynamic time warping, with applications to clustering'. In: *Pattern Recognition* 44.3 (2011), pp. 678–693. ISSN: 00313203. DOI: 10.1016/j.patcog.2010.09.013.

[74] C.-c. Qi. 'Big data management in the mining industry'. In: *International Journal of Minerals, Metallurgy and Materials* 27.2 (2020), pp. 131–139. ISSN: 1674-4799. DOI: 10.1007/s12613-019-1937-z.

[75] S. Rani and G. Sikka. 'Recent Techniques of Clustering of Time Series Data: A Survey'. In: 52.15 (2012). URL: https://research.ijcaonline.org/volume52/number15/pxc3881278.pdf.

[76] K. Ranjeet, A. Kumar and R. K. Pandey. 'ECG Signal Compression Using Different Techniques'. In: *Advances in Computing, Communication and Control*. Ed. by S. Unnikrishnan, S. Surve and D. Bhoir. Vol. 125. Communications in Computer and Information Science, 1865-0929. Berlin and London: Springer, 2011, pp. 231–241. ISBN: 978-3-642-18439-0. DOI: 10.1007/978-3-642-18440-6{\textunderscore}29.

[77] T. Räsänen and M. Kolehmainen. 'Feature-Based Clustering for Electricity Use Time Series Data'. In: *Adaptive and natural computing algorithms*. Ed. by M. Kolehmainen, P. Toivanen and B. Beliczyński. Vol. 5495. Lecture notes in computer science, 0302-9743. Berlin: Springer, 2009, pp. 401–412. ISBN: 978-3-642-04920-0. DOI: 10.1007/978-3-642-04921-7{\textunderscore}41.

[78] A. B. Roy, D. Dey and D. Banerjee. 'Comparison of FFT, DCT, DWT, WHT Compression Techniques on Electrocardiogram &Photoplethysmography Signals'. In: CCSN2012.4 (2013), pp. 6–11.

[79] I. H. Sarker. 'Machine Learning: Algorithms, Real-World Applications and Research Directions'. In: *SN computer science* 2.3 (2021), p. 160. DOI: 10.1007/s42979-021-00592-x.

[80] J.-P. Seevers et al. 'Automatic Time Series Segmentation as the Basis for Unsupervised, Non-Intrusive Load Monitoring of Machine Tools'. In: *Procedia CIRP* 81 (2019), pp. 695–700. ISSN: 22128271. DOI: 10.1016/j.procir.2019.03.178.

[81] T. Sellis and R. J. Miller, eds. *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*. New York, New York, USA: ACM Press, 2011. ISBN: 9781450306614. DOI: 10.1145/1989323.

[82] A. A. Shinde and P. Kanjalkar. 'The comparison of different transform based methods for ECG data compression'. In: *2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies*. 2011, pp. 332–335. DOI: 10.1109/ICSCCN.2011.6024570.

[83] A. Siddiqa et al. 'A survey of big data management: Taxonomy and state-of-the-art'. In: *Journal of Network and Computer Applications* 71 (2016), pp. 151–166. ISSN: 10848045. DOI: 10.1016/j.jnca.2016.04.008.

[84] Y. N. Silva, I. Almeida and M. Queiroz. 'SQL: From Traditional Databases to Big Data'. In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16*. Ed. by C. Alphonce et al. New York, New York, USA: ACM Press, 2016, pp. 413–418. ISBN: 9781450336857. DOI: 10.1145/2839509.2844560.

[85] B. Singh, A. Kaur and J. Singh. 'A Review of ECG Data Compression Techniques'. In: *International Journal of Computer Applications* 116.11 (2015), pp. 39–44. DOI: 10.5120/20384-2644.

[86] U. Sivarajah et al. 'Critical analysis of Big Data challenges and analytical methods'. In: *Journal of Business Research* 70 (2017), pp. 263–286. ISSN: 0148-2963. DOI: https://doi.org/10.1016/j.jbusres.2016.08.001. URL: https://www.sciencedirect.com/science/article/pii/S014829631630488X.

[87] N. H. Son. *Data mining course—data cleaning and data preprocessing*. URL: http://www.mimuw.edu.pl/~son/datamining/DM/4-preprocess.pdf (visited on 04/04/2022).

[88] S. Spiegel and S. Albayrak. 'An order-invariant time series distance measure'. In: *KDIR. SciTePress Digital Library* (2012).

[89] S. Spiegel, D. Schultz and S. Albayrak. 'BestTime: Finding Representatives in Time Series Datasets'. In: *Machine learning and knowledge discovery in databases*. Ed. by T. Calders. Vol. 8726. LNCS sublibrary: SL 7 - Artificial intelligence. Heidelberg: Springer, 2014, pp. 477–480. ISBN: 978-3-662-44844-1. DOI: 10.1007/978-3-662-44845-8{\textunderscore}39.

[90] S. Spiegel et al. 'Pattern recognition and classification for multivariate time series'. In: *Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data - SensorKDD '11*. Ed. by V. Chandola et al. New York, New York, USA: ACM Press, 2011, pp. 34–42. ISBN: 9781450308328. DOI: 10.1145/2003653.2003657.

[91] R. S. Stanković and B. J. Falkowski. 'The Haar wavelet transform: its status and achievements'. In: *Computers and Electrical Engineering* 29.1 (2003), pp. 25–44. ISSN: 0045-7906. DOI: https://doi.org/10.1016/S0045-7906(01)00011-8. URL: https://www.sciencedirect.com/science/article/pii/S0045790601000118.

[92] Streamlit. *Streamlit*. URL: https://streamlit.io/ (visited on 20/05/2022).

[93] B. Thuraisingham. 'Big Data Security and Privacy'. In: *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 279–280. ISBN: 9781450331913. DOI: 10.1145/2699026.2699136.

[94] G. E. Thyer. *Computer Numerical Control of Machine Tools*. 2nd ed. Kent: Elsevier Science, 2014. ISBN: 9781483294612.

[95] P. Wang, H. Wang and W. Wang. 'Finding semantics in time series'. In: *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*. Ed. by T. Sellis and R. J. Miller. New York, New York, USA: ACM Press, 2011, p. 385. ISBN: 9781450306614. DOI: 10.1145/1989323.1989364.

[96] W. Wang et al. 'Data acquisition and data mining in the manufacturing process of computer numerical control machine tools'. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 232.13 (2018), pp. 2398–2408. DOI: 10.1177/0954405417718878.

[97] X. Wang, K. Smith and R. Hyndman. 'Characteristic-Based Clustering for Time Series Data'. In: *Data mining and knowledge discovery* 13.3 (2006), pp. 335–364. ISSN: 1384-5810. DOI: 10.1007/s10618-005-0039-x.

[98] T. Warren Liao. 'Clustering of time series data—a survey'. In: *Pattern Recognition* 38.11 (2005), pp. 1857–1874. ISSN: 00313203. DOI: 10.1016/j.patcog.2005.01.025.

[99] C. Webber, A. Facchini and A. Giuliani. 'Simpler methods do it better: Success of Recurrence Quantification Analysis as a general purpose data analysis tool'. In: *Physics Letters A* 373 (May 2009), pp. 3753–3756. DOI: 10.1016/j.physleta.2009.08.052.

[100] T. Xi et al. 'Tool wear monitoring in roughing and finishing processes based on machine internal data'. In: *The International Journal of Advanced Manufacturing Technology* 113.11-12 (2021), pp. 3543–3554. ISSN: 0268-3768. DOI: 10.1007/s00170-021-06748-6.

[101] Y. Xie et al. 'Implementation of time series data clustering based on SVD for stock data analysis on hadoop platform'. In: *2014 9th IEEE Conference on Industrial Electronics and Applications*. 2014, pp. 2007–2010. DOI: 10.1109/ICIEA.2014.6931498.

[102] C.-C. M. Yeh et al. 'Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets'. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 1317–1322. DOI: 10.1109/ICDM.2016.0179.

[103] H. Zhang and T. B. Ho. 'Unsupervised Feature Extraction for Time Series Clustering Using Orthogonal'. In: *Informatica (Slovenia)*. Vol. 30, pp. 305–319.

[104] J. Zhang et al. 'Fault Detection and Classification of Time Series Using Localized Matrix Profiles'. In: *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 2019, pp. 1–7. DOI: 10.1109/ICPHM.2019.8819389.

[105] K. Zhang et al. 'Trend-based symbolic aggregate approximation for time series representation'. In: *2018 Chinese Control And Decision Conference (CCDC)*. 2018, pp. 2234–2240. DOI: 10.1109/CCDC.2018.8407498.

A Demo Part Alpha

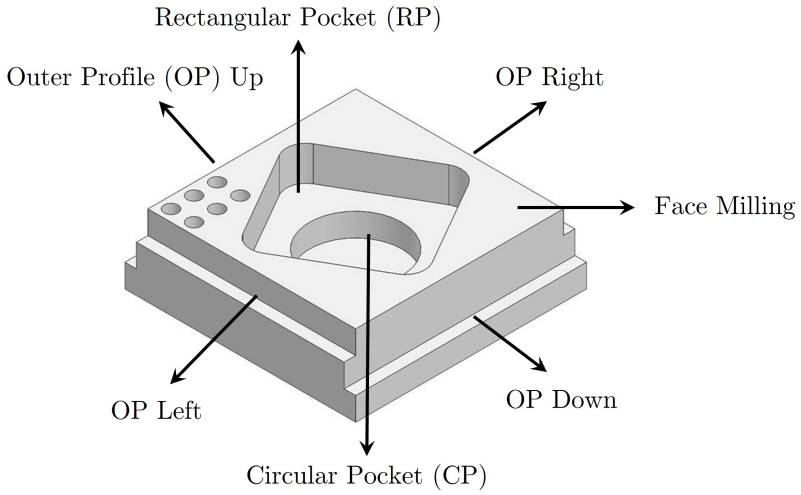

Figure A.1: CAD Model of Demo Part Alpha, with its features labelled.

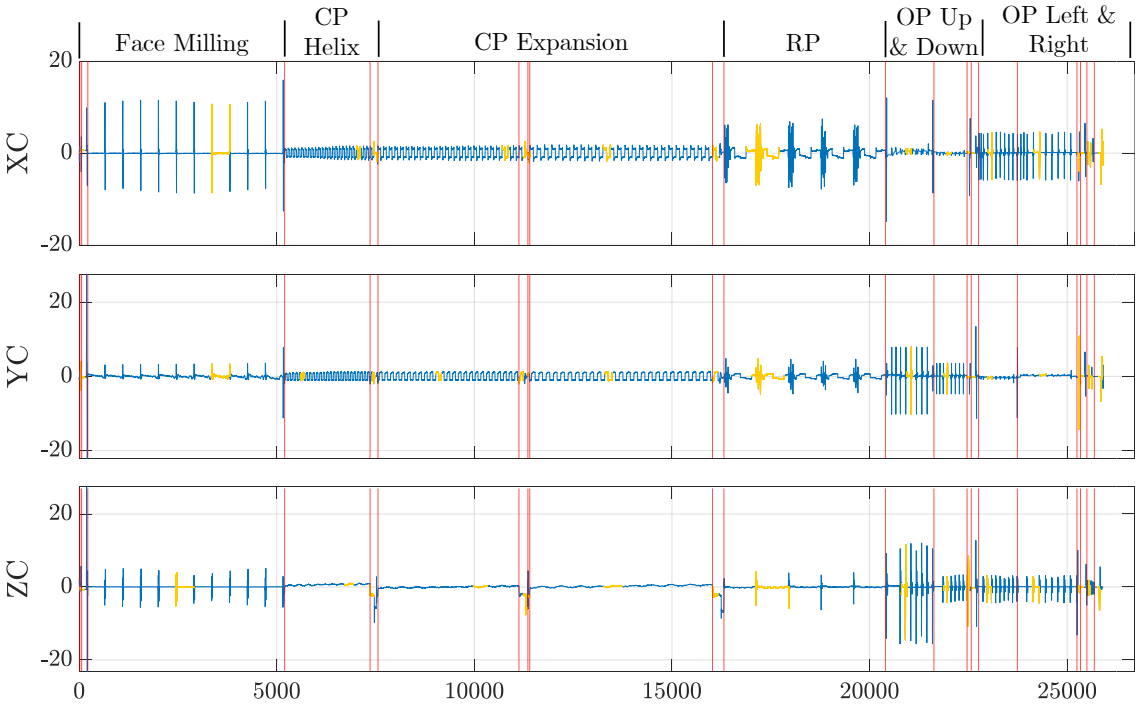

Figure A.2: Manually discovered snippets of Demo Part Alpha.

Dimensions	100x100x32mm		
Dataset length	427946		
Reduced length	25910		
Segment	**Spindle speed (rpm)**	**Tool***	**CP Diameter (mm)**
Face Milling	2190	D16 Z4	-
CP Helix	2785	D16 Z4	30
CP Expansion	2785	D16 Z4	40
CP Expansion	3183	D16 Z4	40
RP	2388	D16 Z4	-
OP Up	2388	D16 Z4	-
OP Down	4297	D16 Z4	-
OP Left	8595	D8 Z4	-
OP Right	8595	D8 Z4	-

Table A.1: Specifications of the Demo Part Alpha process and its resulting dataset. (*) Tool is specified as Dx Zy, which means the milling tool is of x diameter (in mm) and has y number of teeth.

B Demo Part Beta

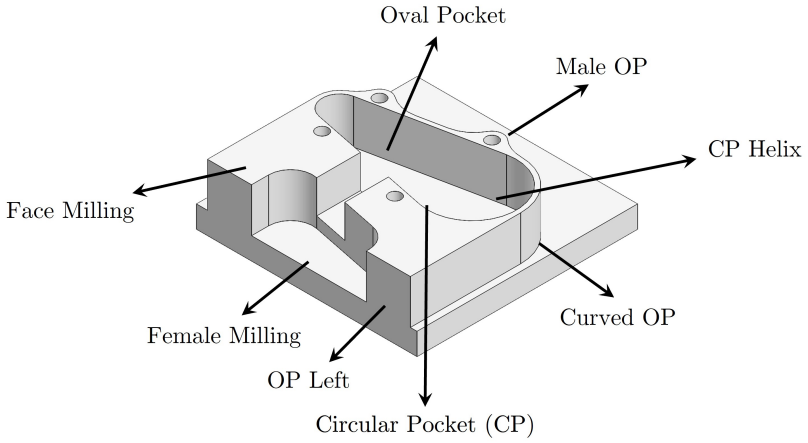

Figure B.1: CAD Model of Demo Part Beta, with its features labelled.

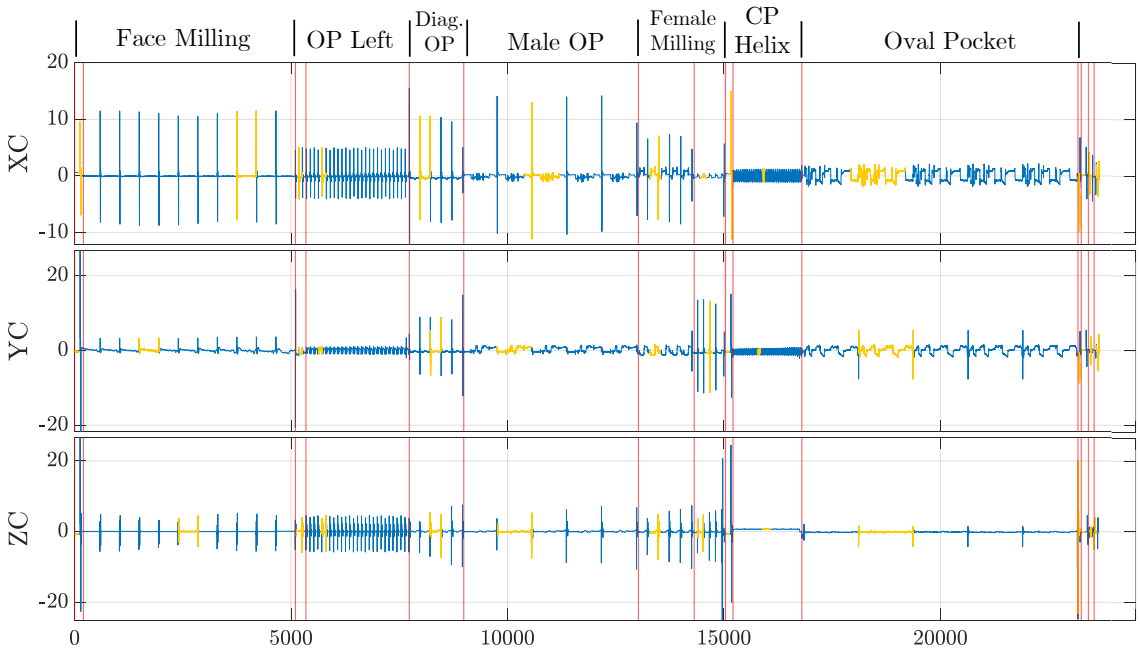

Figure B.2: Manually discovered snippets of Demo Part Beta.

Dimensions	100x100x32mm		
Dataset length	393925		
Reduced length	23635		
Segment	**Spindle speed (rpm)**	**Tool**	**CP Diameter (mm)**
Face Milling	2188	D16 Z4	-
OP Left	4295	D16 Z4	-
Diagonal OP	2388	D16 Z4	-
Male OP	2388	D16 Z4	-
Female Milling	2388	D16 Z4	-
CP Helix	4775	D8 Z4	15
Oval Pocket	4775	D8 Z4	-

Table B.1: Specifications of the Demo Part Beta process and its resulting dataset.

C Demo Part Gamma

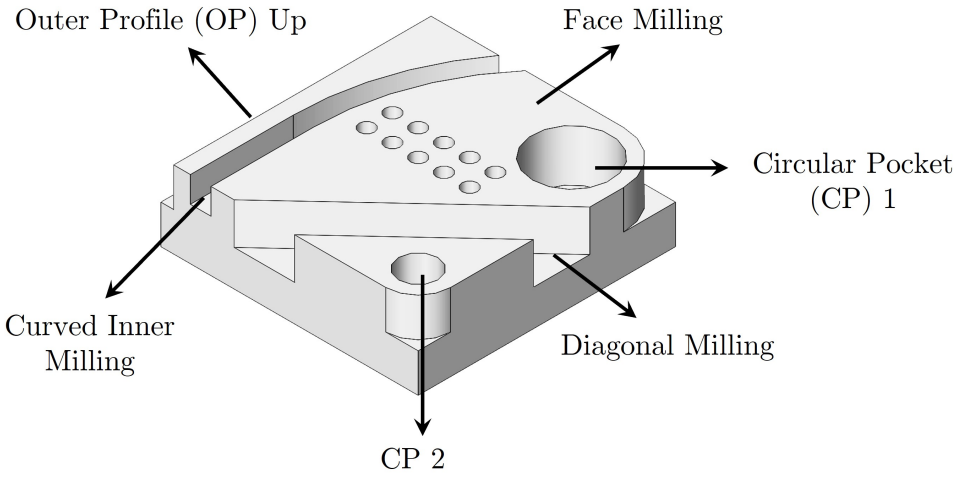

Figure C.1: CAD Model of Demo Part Gamma, with its features labelled.

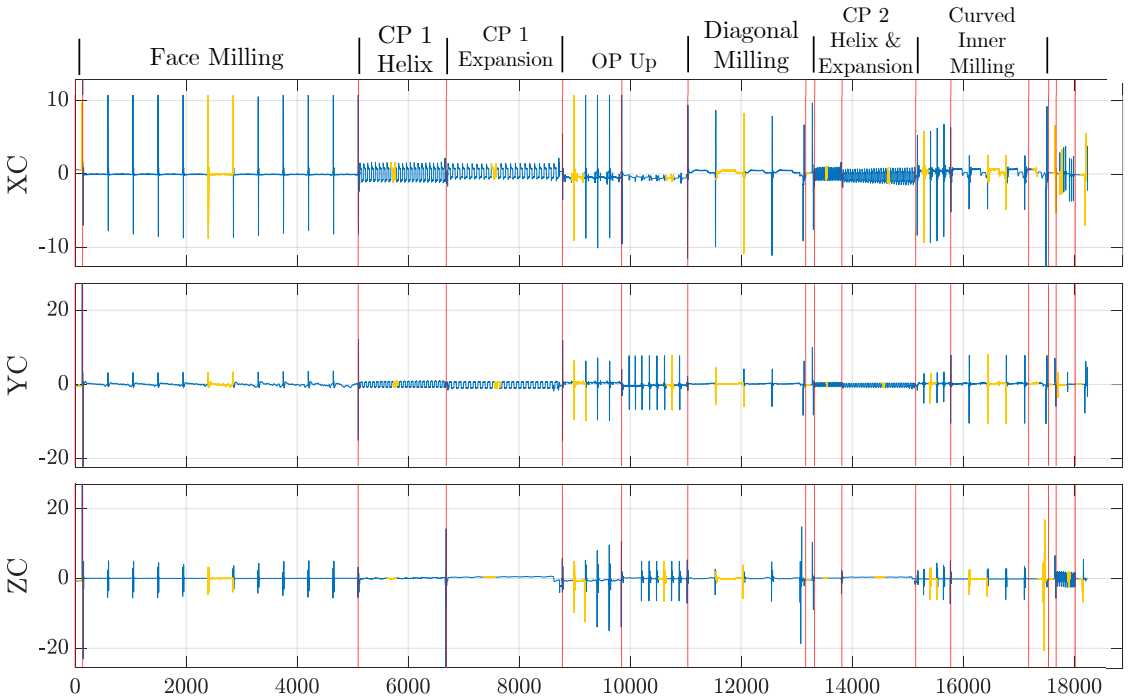

Figure C.2: Manually discovered snippets of Demo Part Gamma.

Dimensions	100x100x32mm		
Dataset length	305967		
Reduced length	18243		
Segment	**Spindle speed (rpm)**	**Tool**	**CP Diameter (mm)**
Face Milling	2188	D16 Z4	-
CP 1 Helix	3085	D16 Z4	30
CP 1 Expansion	2785	D16 Z4	30
OP Up	2289	D16 Z4	-
Diagonal Milling	2388	D16 Z4	-
CP 2 Helix	6167	D8 Z4	15
CP 2 Expansion	4775	D8 Z4	15
Curved Milling	4775	D8 Z4	-

Table C.1: Specifications of the Demo Part Gamma process and its resulting dataset.

D Demo Part Delta

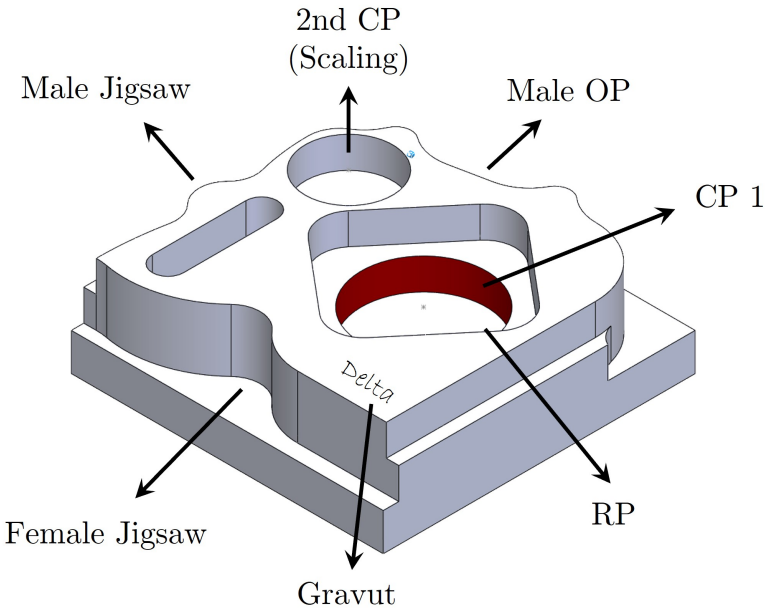

Figure D.1: CAD Model of Demo Part Delta, with its features labelled.

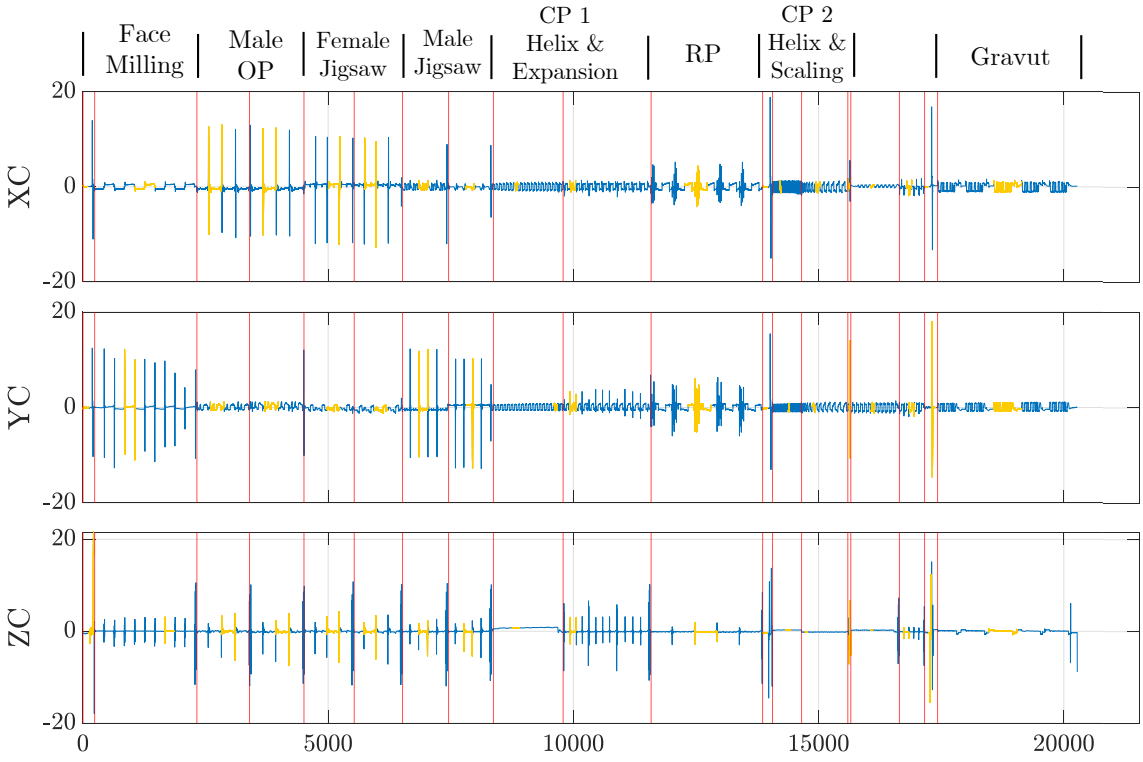

Figure D.2: Manually discovered snippets of Demo Part Delta.

Dimensions	100x100x32mm		
Dataset length	335582		
Reduced length	20275		
Segment	**Spindle speed (rpm)**	**Tool**	**CP Diameter (mm)**
Face Milling	3085	D16 Z4	-
Male OP	3085	D16 Z4	-
Female Jigsaw	3085	D16 Z4	-
Male Jigsaw	3085	D16 Z4	-
CP 1 Helix	3085	D16 Z4	30
CP 1 Expansion	3085	D16 Z4	40
RP	4775	D16 Z4	-
CP 2 Helix	6165	D8 Z4	15
CP 2 Expansion	6165	D8 Z4	38
Gravut	20000	D6 Z4	-

Table D.1: Specifications of the Demo Part Delta process and its resulting dataset.