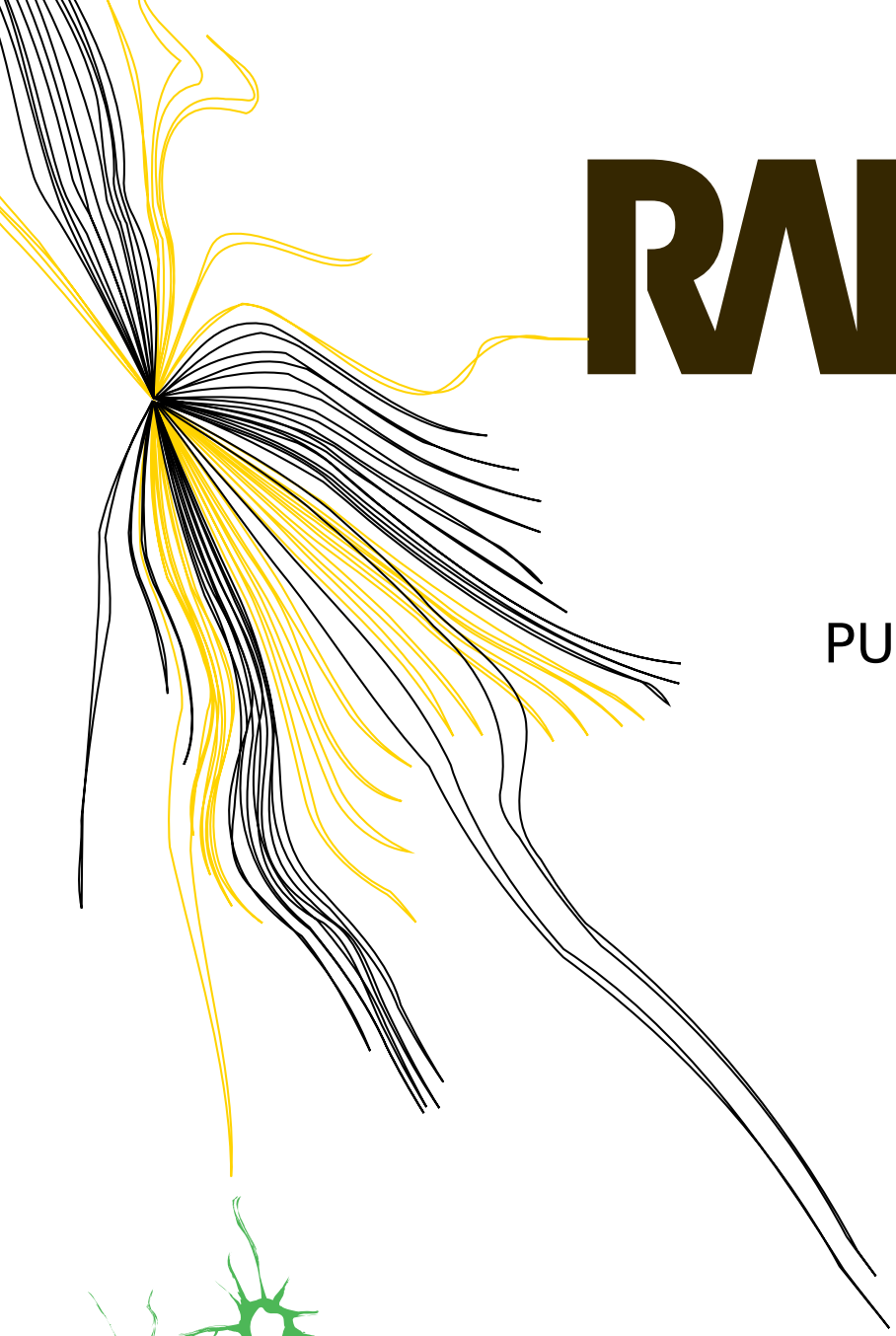


RAM ROBOTICS AND MECHATRONICS

PUMP DIAGNOSTICS USING MACHINE LEARNING

C.J. (Jacco) van Ekris

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
N. Botteghi, MSc
dr. B. Sirmaçek
dr. M. Poel
ir. P. Weustink

May, 2020

012RaM2020
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Abstract

This report shows multiple diagnostic machine-learning models that can be used to recognize several seal and valve leakages in a Ultra High Performance Liquid Chromatography (UHPLC) pump. With the presented methods it is possible to estimate the seriousness of such a fault and to differentiate between several faults. The diagnostic models were trained on a simulation model of the real pump system.

The first diagnostic machine-learning model, that can be used to say something about the seriousness of a fault, uses a combination of feature extraction and a simple Multilayer Perceptron (MLP) model. The second model uses a combination of Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) layers. With the CNN layer of this CNN + LSTM based model it was possible to extract features from sensor signals of a UHPLC pump. Compared with a method based on feature extraction, the CNN + LSTM based model is more robust and averagely increases the accuracy with 32.7% when predicting the seriousness of a fault.

To differentiate between several faults, two methods were investigated. The first method that was investigated used the normalized probability outputs of the individual diagnostic models as input for a simple MLP model. This model did however not converge during training and could therefore not be used to differentiate between faults. The second method was based on a CNN + LSTM model and used the sensor signals directly. This method proved to be successful with a maximal accuracy of 92.5% on the validation set.

All the CNN + LSTM based diagnostic machine-learning models proved to be not only successful on a simulated validation set, but also on the real pump system using a developed C++ application. A model-based machine-learning approach does, however, not always work as was seen in the case of training a diagnostic model to detect backlash. It is therefore important to always check whether the fault can simply be detected using a sensor signal directly instead of using machine learning. Furthermore, when a simulation model of the system available, it is important to check whether the model is not only valid for the normal behavior of the system, but also for problematic behavior.

Finally, a number of things might improve the work presented in this report. Right now the maximum number of classes that can be used to classify a certain fault can only be found using a trial and error method. It might be interesting to perform more research in an unsupervised based approach or better method to determine the maximum number of classes. Furthermore, when training a model to differentiate between faults it is important to also define a class for normal pump behavior. In addition to this, a technique that might be interesting when adding more faults to this differentiate model is continuous learning. With continuous learning it is possible to reuse the already trained differentiate model and only train the model to detect new classes which in theory would improve the training time. Other things that might be interesting to investigate are performing sensitivity analyses for noise on sensor signals, using a variable sampling method instead of the used constant sampling method and performing more research on how to detect cross-correlations of faults such that trained models become even more accurate.

Contents

List of Acronyms

Abbrevation	Meaning
Bi-LSTM	Bidirectional-LSTM
BLDC	brushless DC
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
DFT	Discrete Fourier Transform
DLL	Dynamic-Link Library
DWT	Discrete Wavelet Transform
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
OS	operating system
RF	Random Forest
RMSE	Root-Mean-Square Error
RNN	Recurrent Neural Network
SHAP	SHapley Additive exPlanations
UDP	User Datagram Protocol
UHPLC	Ultra High Performance Liquid Chromatography
Uni-LSTM	Unidirectional-LSTM

Chapter 1

Introduction

1.1 Context

Automated diagnosis of faults in pump systems is a field in which not a lot of research is done and a lot of things can still be improved. In general, all these pump systems have one inlet, one outlet and several sensor signals which can be used to detect general faults such as leakages (see Figure 1.1).

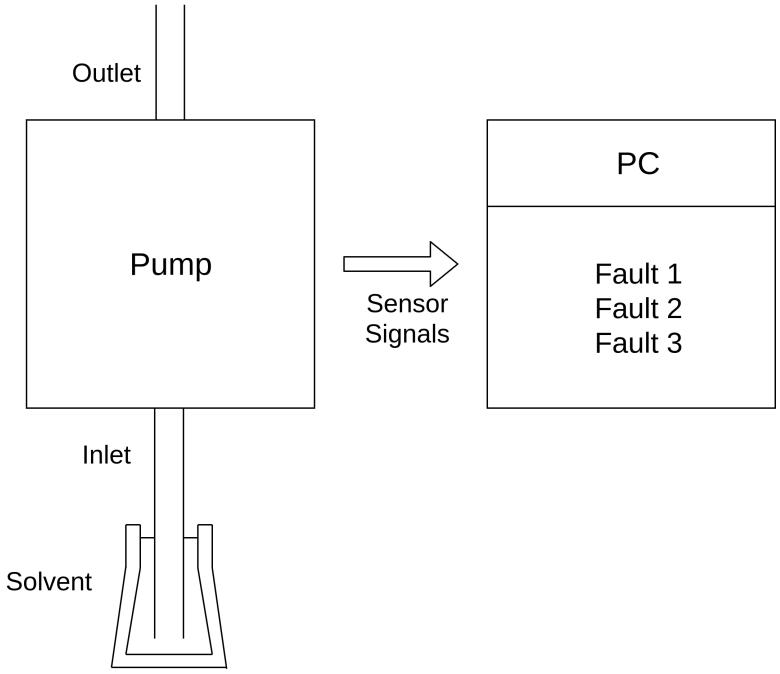


Figure 1.1: Diagnosis of faults in pump systems.

In order to automatically detect faults in systems, several diagnostics methods are currently used, but do each have their own disadvantages.

Traditionally, automated diagnosis has been performed by mainly checking for the exceeding of certain threshold values for sensor signals (Nyberg and Frisk, 2008). However, this method heavily depends on correctly setting these threshold values and is therefore not very robust.

Another method is to check for deviations between a simulation model of a system and the real system (Basseville et al., 1993). However, this method has the disadvantage that running this model in real-time requires a lot of computer power and therefore has the disadvantage that not all possible faults can be detected.

In order to overcome these disadvantages, the work of Murphey et al. (2006) shows a promising method which can be successfully used for locating multiple classes of faults in an electric drive. Using a simulation model of an electric drive, machine-learning models based on feature extraction were trained and faults in the real electric drive could be successfully detected. Compared with the method based on detecting deviations between a simulation model and real system, this method requires less computer power because not the whole system needs to be simulated, but only a specific fault.

In this assignment an improved version of this method will be used to perform diagnostics in a pump used for Ultra High Performance Liquid Chromatography (UHPLC). A sketch of this process that uses two of these so called UHPLC pumps can be seen in Figure 1.2.

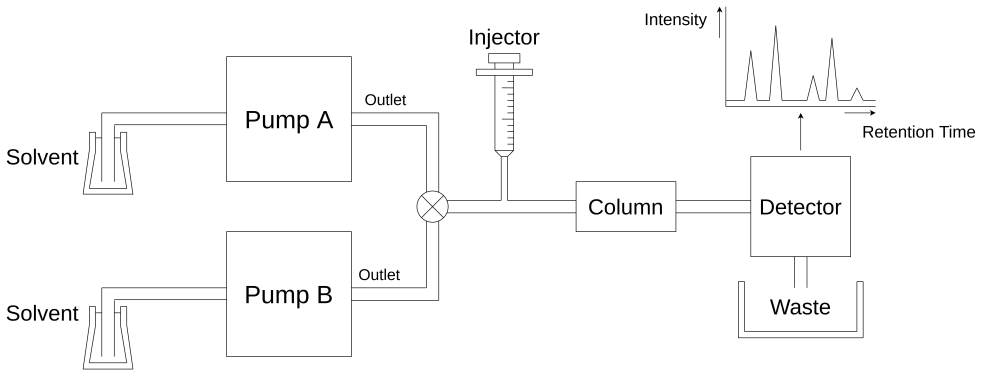


Figure 1.2: Detection of substances using a UHPLC process.

UHPLC is a technique in analytical chemistry used to separate, identify and quantify each component in a mixture; for instance blood (Wikipedia, 2019). It works by passing a pressurized solvent containing the sample mixture under high pressure through a column filled with a solid adsorbent material such as carbon. Each component in the sample interacts slightly differently with the adsorbent material, causing different flow rates for the different components and leading to the separation of the components as they flow out of the column. The separated components can hereafter be identified using for instance a spectrum analyzer.

In order to separate the components successfully, it is important that the UHPLC pump used to bring the liquid solvent under high pressure maintains a constant flow and pressure for the outlet. A short drop in the pressure or small deviation in the flow can result in a completely worthless result or even result in a wrong identification of components in a to be analyzed sample.

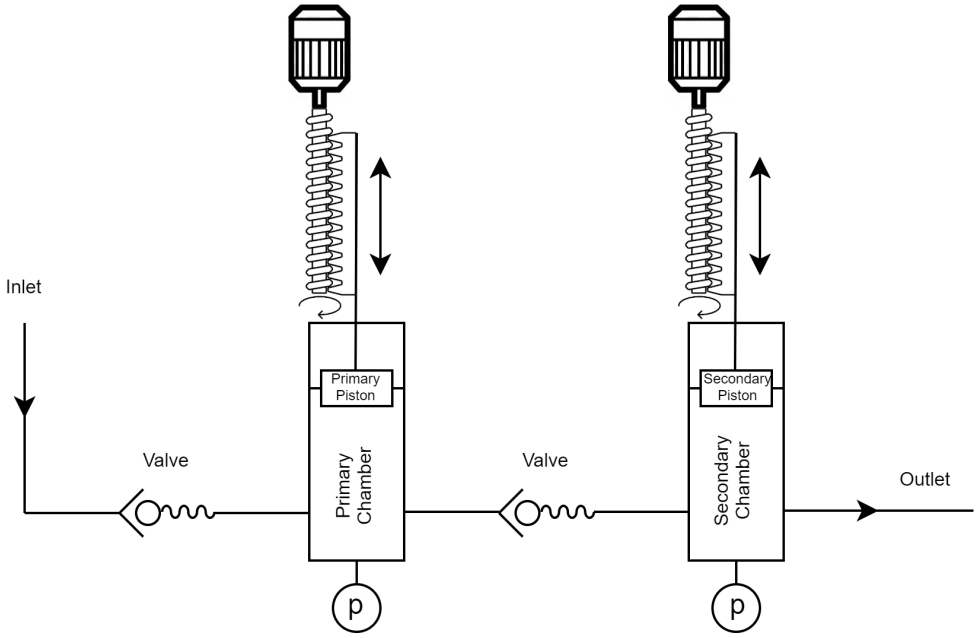


Figure 1.3: Sketch of the pump system containing two valves, two pistons with pressure sensors and two electric motors.

Figure 1.3 shows a sketch of a UHPLC pump. As previously mentioned, the main goal of the pump is to maintain a constant flow and pressure for the outlet. To achieve this, two pistons (primary and secondary piston) and two pressure chambers (primary and secondary chamber) are used. Using a specific stroke pattern for both pistons, the pressures inside both chambers can be controlled and the pressure of the secondary chamber can be kept constant. Because this secondary chamber is directly connected to the outlet, this results in a constant flow and pressure for the outlet.

However, sometimes a seal or valve leakage occurs inside the pump resulting in an inconsistent flow or pressure for the outlet. In order to detect these kind of faults, two promising machine-learning techniques are investigated and compared.

Hereby the first method is a machine-learning model based on feature extraction that is also used in the work of Murphey et al. (2006). Compared with this method more recent machine-learning techniques based on Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) have proven to be successful in the area of diagnostics (R. Zhao et al., 2017). The second method will therefore be a machine-learning model based on CNN and LSTM. Just as in the work of Murphey et al. (2006) both machine-learning models will be trained on a dataset generated by a simulation model of the real system, in this case the 20-sim model of the pump in Figure 3.2. By comparing the second method with a method based on feature extraction, it can be determined whether using these newer machine-learning techniques leads to a better performing model and therefore possibly a better method as presented in the work of Murphey et al. (2006).

1.2 Research Questions

Based on the previously addressed context, the following research question is composed:

To what extend can a machine-learning method trained on a simulation model be used to detect faults in a real pump?

This research question is made more specific by the following sub-questions:

1. What are important faults to detect and how can they be simulated using the provided 20-sim model of the pump?
2. Can a CNN be used to perform feature extraction and can a model based on CNN + LSTM be used to detect faults in a pump?
3. How does a CNN + LSTM based method compare with a method based on feature extraction?
4. What method can be used to differentiate between faults?
5. How can the final implementation of the diagnostic machine-learning models be validated?

1.3 Approach

In order to answer the proposed research questions, first research on several diagnostic machine-learning methods is done and it is explained why the specific choice for a machine-learning method based on CNN + LSTM is made. Thereafter, an analysis of the current pump system is made to determine what faults can occur and to come up with set of requirements for the to be developed diagnostics. Next, the 20-sim model of the pump is analyzed to determine how the faults can be simulated to generate datasets that can be used for the training of diagnostic machine-learning models.

After this analysis and research phase, the two different machine-learning techniques are used to train several diagnostic machine-learning models on individual faults. With these individual models it should be possible to say something about the seriousness of a fault. Next, a method will be developed to differentiate between faults such that can be determined which fault really occurs when multiple individual models detect a fault. Finally, the best individual models and the differentiate model will be validated in real-time on the real pump system.

1.4 Outline

Chapter 2 describes the background which contains literature research which explains what diagnostic machine-learning methods can be used and how the work. Next, Chapter 3 describes the current state of the used pump system, what faults can occur and how the simulation

model looks like. The design Chapter 4 describes which machine-learning framework is most suitable and shows the machine-learning pipeline that will be used to train diagnostic models. Furthermore, it shows an in depth explanation of the individual and differentiate models and how these models will be validated on the real pump system. Chapter 5 shows the results of the experiments and Chapter 6 describes the discussion of theses results. Finally, Chapter 7 shows the conclusion of the assignment and answers each research question.

Chapter 2

Background

In this background chapter, literature research is done to find out how similar diagnostic problems were solved using machine-learning and the most promising techniques for this assignment are described.

When looking in the literature, it can be concluded that automated diagnosis is not used for UHPLC pumps yet. When looking more generally at other types of pumps, centrifugal pumps for instance, sensor signals that are often used for diagnostics are pressure, motor speed, motor torque, motor current and flow rates (Isermann, 2011). Most of these sensor signals are also available for the pump used in this assignment.

Because the diagnostic machine-learning models will be trained on a dataset generated from a simulation model, the dataset will be a dataset based on time series. In order to develop a classification model based on time series, several methods can be used such as proposed by Hyndman et al. (2015), Fulcher et al. (2013), Fulcher and Jones (2014) and Bandara et al. (2017). Summarized, these methods come down to two different approaches; using an optimized process to extract and select features in time series or using a Recurrent Neural Network (RNN). In the case of RNNs, in particular LSTM networks are used.

2.1 Feature Extraction and Selection

In order to come up with useful features from the generated datasets, this Section looks into extracting features from time series and how to select the most promising features for training.

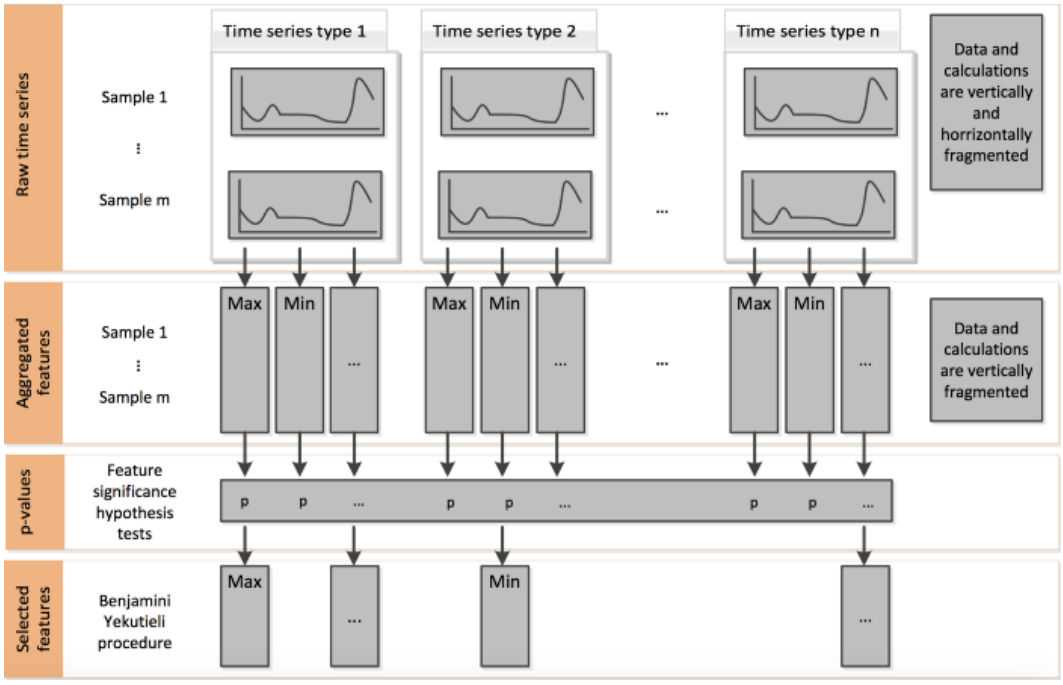


Figure 2.1: General workflow for feature extraction from time series (Christ et al., 2016).

Figure 2.1 shows the general workflow for feature extraction from time series. First, a window with a specific time span is taken of raw time series data. Next, feature aggregation is used to capture all samples of the window in a limited number of features. This is done by determining

minimum, maximum, mean and more values of the window. Finally, the most significant features are determined (feature selection) and used for the final dataset.

The work proposed by Trovero and Leonard (2019) and Jahankhani et al. (2006) both show examples of how such a feature-extraction method can be used to perform classification on time series. The work of Olszewski (2001) shows how these kind of classifications can be improved by using a number of generic features that can be used in multiple domains.

Next, Chatterjee (2019) shows how the Python library TSFRESH (Christ et al. (2018)) can be used to automatically extract useful features from time series. By default, the library extracts 794 features after which the most significant features are selected based on automatically configured hypothesis tests. The work of Chatterjee (2019) also shows a method to perform feature selection using a Random Forest (RF) classifier. Another approach for automatic feature extraction for classification in time series is shown by Mierswa (2005). The paper proposes a method for automatic feature extraction using Discrete Wavelet Transform (DWT) and Discrete Fourier Transform (DFT) which improves clustering and reduces the size of classification rule sets on a benchmark dataset.

2.2 Long Short-Term Memory (LSTM)

In this section, a brief explanation is given about LSTMs and multiple methods to implement LSTMs are analyzed.

An example of a LSTM can be seen in Figure 2.2. Compared with standard RNNs, as seen in Figure 2.3, LSTMs are capable of learning long-term dependencies. This is achieved by using four interacting neural network layers instead of only one layer. The four interacting neural network layers together determine what input information should be kept and how much it should contribute to the output.

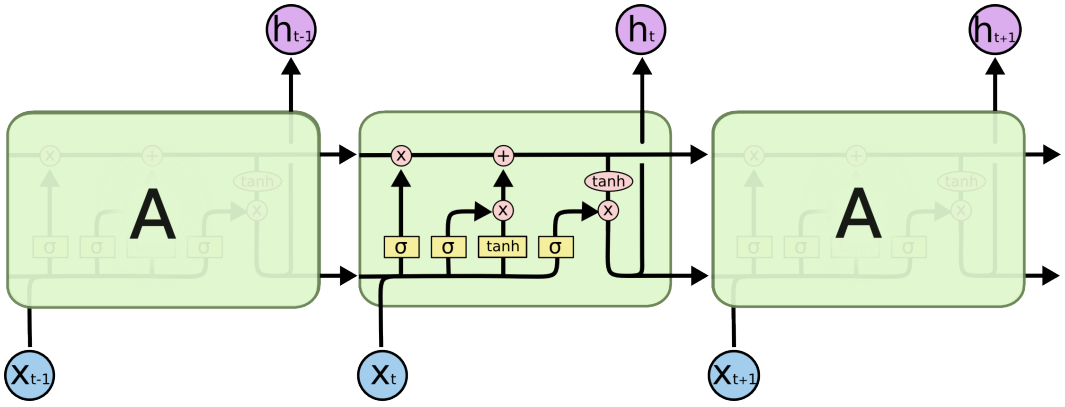


Figure 2.2: A LSTM containing four interacting neural network layers (σ, σ, $\tanh$ and σ), three inputs (x_{t-1}, x_t, x_{t+1}), three outputs (h_{t-1}, h_t, h_{t+1}) and multiple LSTM cells (A). (Christopher Olah, 2019).

A first example of a LSTM implementation is proposed by Lipton et al. (2015). This research evaluates the ability of LSTMs to recognize patterns in multivariate time series of clinical measurements. The proposed model was trained on raw time series, was able to outperform several strong baselines on a wide variety of metrics (F1 score of 0.933) and nearly matched the performance of a Multilayer Perceptron (MLP) trained on hand-engineered features.

J. Yang and Kim (2018) propose an algorithm for accident diagnosis using LSTM which improves the limitation for time reflection. Furthermore, the outputs of the proposed model are post-processed using softmax to determine the ranking of accident diagnosis results using probabilities.

Next, a fault diagnosis scheme for fault diagnosis of rotating machinery is proposed by R. Yang et al. (2018). Using the LSTM based method, both spatial and temporal dependencies can be utilized to detect and classify faults based on available sensor measurement signals.

Other papers that show classifiers based on LSTM for diagnostic problems are proposed by R. Zhao et al. (2017), H. Zhao et al. (2018), Choi et al. (2016), Pan et al. (2018), Xiao et

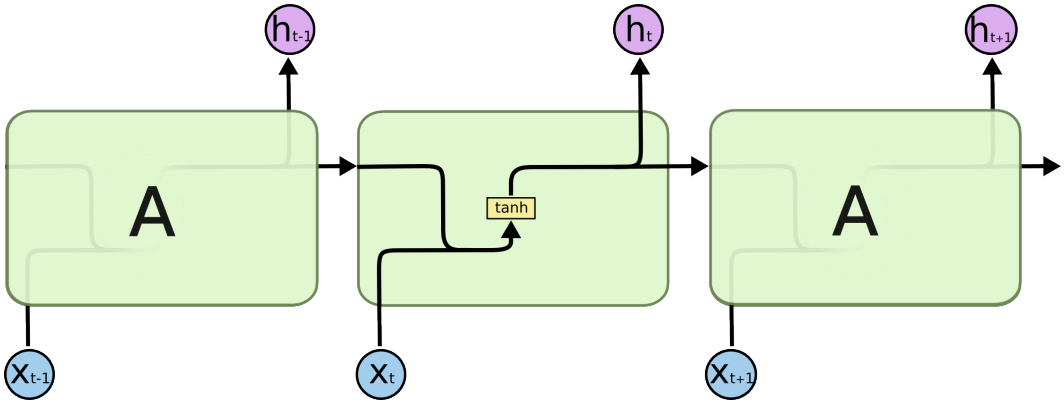


Figure 2.3: A standard RNN containing only one neural network layer ($\tanh$), the same number of inputs and outputs as Figure 2.2 and multiple LSTM cells (A) (Christopher Olah, 2019).

al. (2019) and K. Lee et al. (2018). Here the work of R. Zhao et al. (2017) and K. Lee et al. (2018) looks especially interesting due to the use of a CNN, before the LSTM layers, to perform feature extraction from raw data. Furthermore, a Bidirectional-LSTM (Bi-LSTM) is used instead of a traditional Unidirectional-LSTM (Uni-LSTM). According to the papers, a Bi-LSTM has the advantage over Uni-LSTMs to preserve information from both past and future, instead of only past information, which improves the classification. In the case of K. Lee et al. (2018) the combination of a CNN and Bi-LSTM resulted in an improvement of 7.22% for the Root-Mean-Square Error (RMSE) compared with other state-of-the-art models that only used a Uni-LSTM or Bi-LSTM. For the work of R. Zhao et al. (2017), also an improvement was seen over LSTMs that did not use a CNN.

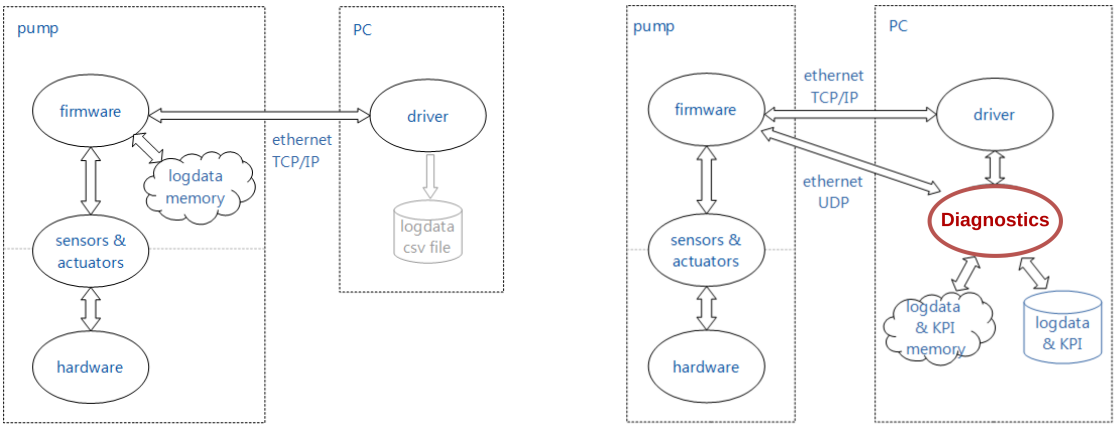
Chapter 3

Analysis

In this analysis chapter, the current state of the system is analyzed to determine what kind of hardware and software is used to control the existing UHPLC pump. Based on this, a number of requirements for the to be developed system are specified. Next, a list of possible faults for the pump are shown and research is done on what kind of variables can be used to detect these specific faults. Finally, the 20-sim simulation model is analyzed to determine how these faults can be simulated to generate datasets that can be used for training diagnostic machine-learning models.

3.1 Current State of the System

In Figure 3.1a an architecture overview of the current pump situation can be seen. The system mainly exist out of two hardware components; the pump and an PC. The pump contains firmware that communicates with internal sensors and actuators, logs data to a memory buffer and communicates with the driver component of the PC that is able to control the pump. Right now, when a fault within the pump occurs, a mechanic uses the PC to retrieve the latest 240 seconds of log data from the pump and saves it to a CSV file. This log file is then send to an expert who will determine what causes the fault and how it can be fixed.



(a) Current situation

(b) New situation

Figure 3.1: Overview of the current (left) and new (right) pump situations where the arrows represent the exchange of data between the modules (Weustink and Ekris, 2019). In this assignment there is focused on the development of the diagnostic component as shown in red for the new pump situation.

In order to remove the need for an experts opinion to solve a fault, the new situation as seen in Figure 3.1b is proposed. For this new situation, the pump will no longer write log data to a memory buffer, but instead directly communicate the log variables to the diagnostic component on the PC using an User Datagram Protocol (UDP) connection. This has the advantage that the pump no longer has to be stopped to obtain log data and diagnostics on this log data can be done in real-time. Furthermore, an UDP instead of a TCP/IP connection is used such that a higher baud rate can be achieved. This is necessary since the sensor signals are updated every 1 millisecond.

The focus of this research is on the development of the diagnostic component of the PC. For the PC hardware, the current driver that controls the pump is based on C++, runs on a Windows operating system (OS) and has the following minimal specifications:

- CPU: Dual Core 2.4 GHz+ (Intel i5/i7 or equivalent AMD)
- RAM: 8 GB
- Disk Capacity: 128 GB

In order to use the same PC hardware and OS, the to be developed and implemented diagnostics should thus be able to run on a Windows OS and should not require higher specifications as mentioned above. Preferably, the diagnostics should be implemented in C++, such that the driver component and diagnostics can be implemented in one application.

3.2 Faults

As mentioned in Section 1.1, the main goal of the UHPLC pump is to maintain a constant flow and pressure for the outlet and to achieve this, two pistons and two pressure chambers are used.

An example for normal behavior of the pump can be seen in Figure 1 of Appendix A. The first plot shows that by changing the desired flow setpoint, the primary and secondary pressures inside the pump change accordingly. In the second plot, it can be seen how the two pistons inside the pump achieve this desired flow setpoint and keep the secondary pressure constant using their specific stroke behavior.

Several faults inside the pump can occur which result in unwanted behavior for the pump. According to the customer of the pump, the following faults (F1 - F8) have the highest priority for the to be developed diagnostics. This assignment will focus on the first five faults:

- F1. Primary Seal Leakage
- F2. Outlet Check Valve Leakage
- F3. Inlet Check Valve Leakage
- F4. Backlash
- F5. Pump Head Blockage
- F6. Pressure Deviation Chambers
- F7. Pressure Ripple
- F8. Secondary Seal Leakage

In Figure 2 of Appendix A an example for one of the faults is shown. In the first plot it can be seen that the pressure in the secondary chamber is no longer constant. Furthermore, in the second plot it can be seen that the primary piston stokes have a non-linear pattern instead of the linear pattern shown in Figure 1 of Appendix A. From this can be concluded that there is a seal leakage for the primary piston.

3.3 Simulation of Faults

Figure 3.2 shows a top overview for the 20-sim simulation model of the UHPLC pump which can be used to simulate faults and generate the required datasets to train the machine-learning models.

In the first block of Figure 3.2, the firmware of the pump is modeled. In the second part of the model, the control block is shown which is used for code generation. This generated code runs inside the driver component of the PC as shown in Figure 3.1. Next, the I/O blocks for both pistons are shown and the spindle blocks which can be found in the mechanics part of the model. These spindle blocks both contain a model for a brushless DC motor that drives a spindle for the piston. Finally, the hydraulics block is shown which contains a model of the primary and secondary chambers, some pressure sensors and several hydraulic valves.

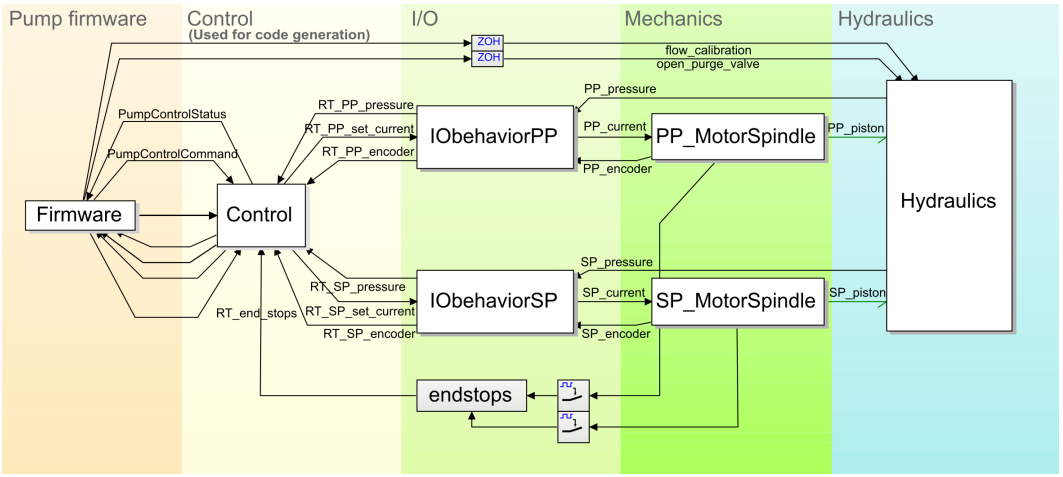


Figure 3.2: Top view of the 20-sim simulation model for the pump and PC (Weustink and Ekris, 2019).

In order to simulate the faults, as mentioned in Section 3.2, several model parameters will be changed. A list of parameters that will be changed for each fault can be found in Appendix B. This list was drawn up using a consultation with the developers and modelers of the pump. To limit the scope of the project, it is assumed that the following conditions apply to the behavior of the pump when running the diagnostics:

- Pressure of 300 - 900 bar
- Flow setpoint of 0.5 - 1 mL/min
- Water is used as solvent
- The sensor signals continue to function

where most of the faults can be simulated by only changing a parameter in the 20-sim model, F1 to F8 except F4, this is not the case for backlash because it is not modeled yet. In order to model backlash in an efficient way the formula as seen in Equation 3.1 can be used.

$$f(x) = \frac{1}{\alpha} \cdot \ln\left(\frac{1 + e^{\alpha \cdot (x-1)}}{1 + e^{-\alpha \cdot (x+1)}}\right) \tag{3.1}$$

This formula is proposed in the work of Margielewicz et al. (2019) and should help the integration routine when simulating backlash. In this formula the backlash force (f) in Newton is calculated based on the position difference (x) in meter, a coefficient α which can be freely chosen and a backlash value of 1 micrometer. The higher the value for coefficient α, the more accurate the model becomes. Chapter 4 describes how this formula is implemented in the 20-sim model to simulate backlash.

3.4 Diagnostic Machine-Learning Models

Related work in Chapter 2 shows that a machine-learning method based on a combination of CNN and LSTM can sometimes outperform or has similar performance as a method based on a combination of feature extraction and a machine-learning method. For a feature-extraction method, features need to be manually engineered or an automatic feature-extraction method such as TSFRESH can be used. However, a combination of CNN and LSTM has the potential to skip this process. Furthermore, the related work has shown that LSTMs can be successfully used for the detection of several diagnostic problems and that adding one or more CNN layers to these models resulted in a higher accuracy. Based on this literate research, a machine-learning model using a combination of CNN and Bi-LSTM layers looks the most promising for the detection of problem situations inside the UHPLC pump. However, to make sure that this is

also the case for this specific machine-learning problem, both methods will be tried out and the performance will be compared.

The performance of the feature-extraction model and CNN + LSTM based model will be compared on the first three faults (F1 - F3). Hereby the models are trained to say something about the seriousness of a fault and several different settings are tried out which will be further explained in Section 4.3. Next, the best performing method will be used to train diagnostic machine-learning models on fault F4 and F5. Again, these models are trained to say something about the seriousness of the fault. Finally, two different methods are investigated to differentiate between several faults which will be further explained in Sub-Subsection 4.3.2.3.

Chapter 4

Design

This design chapter shows how the backlash is modeled because this fault could not be simulated by only changing a parameter in the 20-sim model. Furthermore, the chosen machine-learning framework and the machine-learning pipeline are shown. Hereby the machine-learning pipeline section exactly describes the pre-processing of the data, the layout of the used machine-learning models and the validation process on the real pump system.

4.1 Simulation of Faults

As mentioned in Section 3.3 most of the faults could be simulated by only changing a parameter in the 20-sim model. However, the current simulation model of the pump did not have an option to simulate the backlash fault and therefore an additional component had to be added to the model.

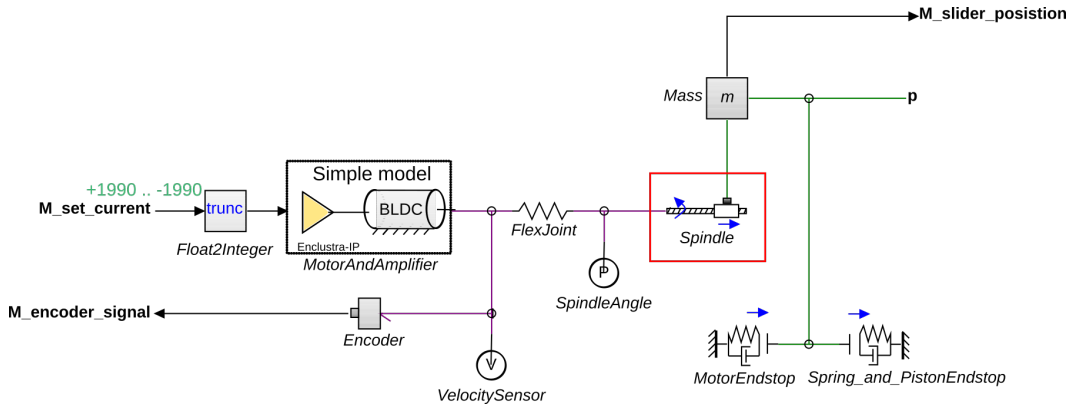


Figure 4.1: 20-sim simulation model of the primary piston motor spindle.

Figure 4.1 shows the 20-sim simulation model of the primary piston motor spindle. The red box marks the spindle which transforms a rotation of the brushless DC (BLDC) motor to a translation of the mass. In this spindle backlash can occur which results in that a rotation does not directly result in a translation. This phenomenon only occurs when changing the direction of rotation for the BLDC motor. In order to simulate this fault the simulation model in Figure 4.2 is made.

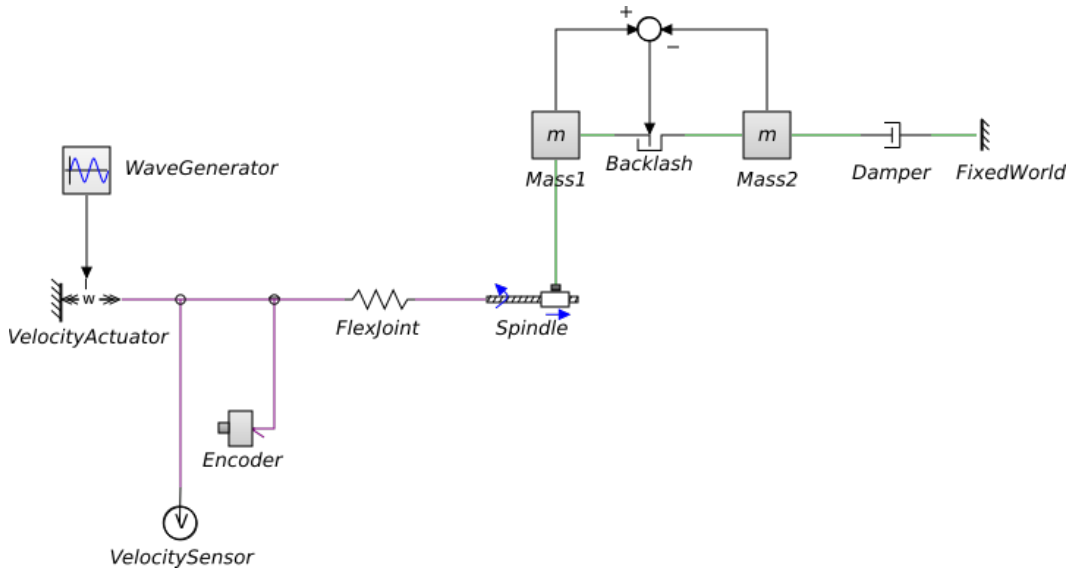


Figure 4.2: Simplified 20-sim simulation model of the motor spindle with backlash.

The simplified simulation model of Figure 4.2 shows that the mass presented in the original model is divided in two parts and a new backlash component is placed in between. The standard 20-sim implementation of this backlash component is changed in order to reduce the required simulation time.

In order to make Equation 3.1 suitable for different backlash values and more accurate when approaching the backlash value, an adapted version of this formula as can be seen in Equation 4.1 is proposed and used in the final implementation of the model.

$$f(x) = \beta \cdot \frac{s}{\alpha} \cdot \left(\ln\left(s + e^{\alpha \cdot (x-s)} \right) - \ln\left(s + e^{-\alpha \cdot (x+s)} \right) \right), \tag{4.1}$$

$$\text{where } s = \frac{backlash}{2},\ \alpha = 100 \cdot \frac{1}{s},\ \beta = slope$$

Chapter 5 shows a simulation with a backlash value of 10 µm and a slope of 10^{10}. This slope value will also be used when performing simulations with different backlash values to generate the dataset. Chapter 5 also shows how all the other faults were simulated.

4.2 Machine-Learning Framework

In order to implement both models, several machine-learning frameworks are compared. Section 3.1 describes why a final implementation of the diagnostics in C++ is preferred. However, the development and training of machine-learning models is considered more straight forward in Python due to the large number of supported libraries and simpler syntax. A machine-learning framework that supports the training of a model in Python and thereafter exporting it to C++ is therefore preferred.

Popular frameworks that support such a workflow are TensorFlow (Martıén Abadi et al., 2015), PyTorch (Paszke et al., 2017) and Caffe (Jia et al., 2014). A comparison between the different frameworks for several criteria can be seen in Table 4.1.
For each criteria a score on a scale from 1 to 5 is used where 1 stands for bad and 5 for good. Hereby LSTM and CNN support both have a weight of 2 since one of the models will be a CNN + LSTM based model. The scores are determined by reading the documentation of the libraries and based on user feedback. In the end, the choice for TensorFlow is made due to its good documentation, support for LSTMs and CNNs and because it is more production ready. Furthermore, TensorFlow has the advantage that Keras (Chollet et al., 2015) can be used for a simplistic high level implementation of models resulting in a less steep learning curve. Finally, the GPU support of TensorFlow ensures that the training of models is faster than when training the model using a CPU.

Table 4.1: Comparison of multiple machine-learning frameworks.

	Weight	TensorFlow/Keras	PyTorch	Caffe
Production Ready	1	5	3	2
LSTM Support	2	5	5	1
CNN Support	2	5	5	5
GPU Support	1	5	5	5
Other Functionality	1	5	4	3
Simplicity	1	5	5	3
Documentation	1	5	4	3
Total Score		45	42	28

4.3 Machine-Learning Pipeline

To prepare the generated datasets for training and compare the performance of the models, the machine-learning pipeline in Figure 4.3 is used. The subsections that follow describe how each section of this pipeline is designed. The first Subsection describes the data generation and pre-processing of the data. The second Subsection describes how this data is used to train three different types of diagnostic machine-learning models. Finally the third Subsection describes how the performance of these diagnostic machine-learning models is analyzed on a validation set and on a real pump system.

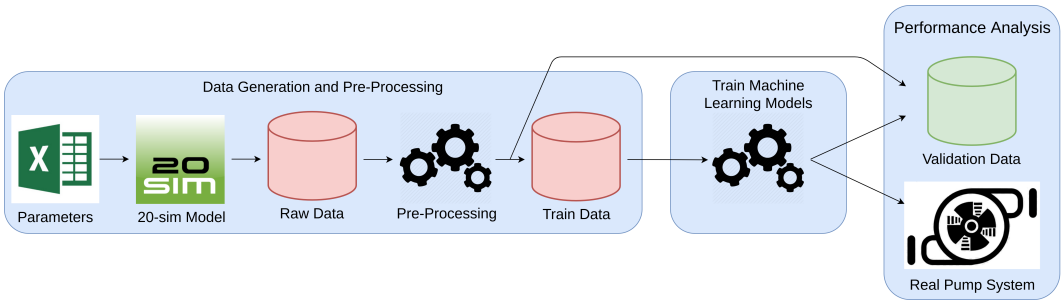


Figure 4.3: General machine-learning pipeline to generate the datasets for training and evaluating the performance of trained machine-learning models on the simulation model and real pump system.

4.3.1 Data Generation and Pre-Processing

In order to automate the data generation process, parameter combinations can be defined in an Excel file and using the Python scripting interface of 20-sim simulation results can be saved to a Comma-Separated Values (CSV) file. In the Excel file, for each parameter combination a label can be provided which can later be used to train a machine-learning models. During the labeling of the dataset it is made sure that each class has approximately the same number of parameter combinations such that the resulting dataset will balanced.

The by 20-sim generated raw dataset needs to be pre-processed. A detailed pipeline for this pre-processing can be seen in Figure 4.4 and pseudocode of the algorithm can found in Appendix D.

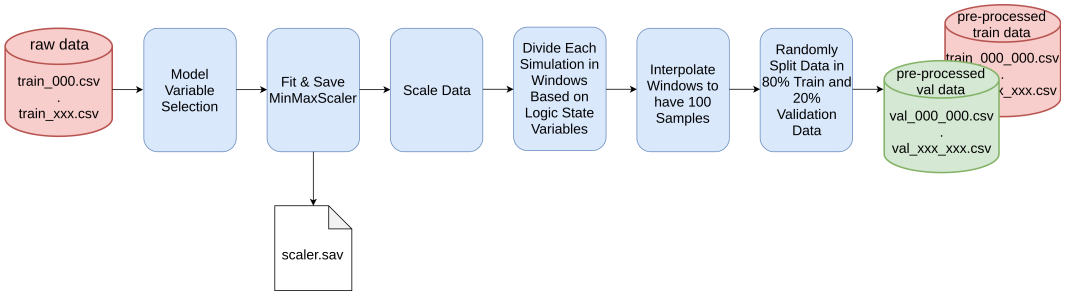


Figure 4.4: Pre-processing pipeline to prepare the generated raw datasets for training.

During the pre-processing, first only the relevant model variables will be selected such as the pressure in the primary and secondary chamber of the pump and the position of pistons. The table in Appendix C exactly shows what variables were available, which of these variables were selected and why they were selected. The selected variables are hereafter normalized using a min-max scaler such that all values lay between 0 and 1. According to Suarez-Alvarez et al. (2012) this should prevent that specific variables with large numerical values dominate the model and according to Pedregosa et al. (2011) this ensures that a model converges faster during training. Next, each data file, which is the result of a simulation with one parameter set, is divided in windows based on the logic state variables (see Appendix F). These logic state variables are one of the variables which are available in the data file and show the current state of the controller. For the first three faults (F1 - F3) simply the full window will be used as can be seen in Figure 1 of Appendix F. For the remaining faults (F4 and F5), where only the best machine-learning method will be used, there will be experimented with different window sizes (Figure 2 of Appendix F). In both cases the selected windows are interpolated to have a fixed window size of 100 samples such that the data is suitable for a LSTM model. Finally, the resulting data files are divided in 80% train and 20% validation data. It is chosen to not use cross-validation because enough data can be generated using the 20-sim model and would therefore not lead to a significant performance difference for models between folds.

4.3.2 Train Machine-Learning Models

Using these pre-processed training datasets both a MLP model based on feature extraction, a CNN + LSTM based model and a differentiate model will be trained. Hereby the first two models can say something about the seriousness of fault and the differentiate models can differentiate between several possible faults. The hyperparameters of the models will be optimized using Bayesian optimization. Bayesian optimization is used because it has shown its superior performance above other optimization algorithms like Grid Search and Random Search multiple times (Koehrsen, 2018; Kraus, 2019; Snoek et al., 2012). Categorical cross entropy is used as loss function such that the probability for each prediction can be given. The training of a model with a specific set of hyperparameters is stopped when the loss does not improve for 10 continuous epochs.

4.3.2.1 Feature-Extraction Model

Section 2.1 explains the general workflow for a machine-learning model based on feature extraction. In this section more details of the chosen implementation and the hyperparameter optimization process are shown.

After some initial testing with the TSFRESH library it was found that it is not feasible to use all the 794 features for feature aggregation. Calculating and determining the best features would simply take to much time. It was therefore decided to use a smaller selection of features. Based on the knowledge for detecting a specific problem situation by hand, the following most promising features are used to capture all the samples of a window:

- Mean
- Standard Deviation
- Derivative
- Kurtosis
- Minimum
- Maximum
- Median
- Sample Entropy
- Skewness

- Variance
- Count Above Mean
- Count Below Mean
- Mean Absolute Change
- Mean Change
- Mean Second Derivative Central
- Absolute Energy
- Absolute Sum of Changes

For each signal that was selected (Appendix C), the mentioned features are calculated. The formulas that are used for calculating the mentioned features can be found in Appendix E. Next, the K-best features are used to train several MLP models with different hyperparameters. Hereby, the K-value for selecting the best features is one of the hyperparameters that will be tuned using the Bayesian optimization algorithm. Other hyperparameters that will be tuned are the number of neurons in each layer and the number of layers. The Adam optimizer with its default parameters will be used to train the model (learning rate = 0.001, beta 1 = 0.9, beta 2 = 0.999 and no AMSGrad). An example of the feature-extraction model with three dense layers can be seen in Figure 4.5.

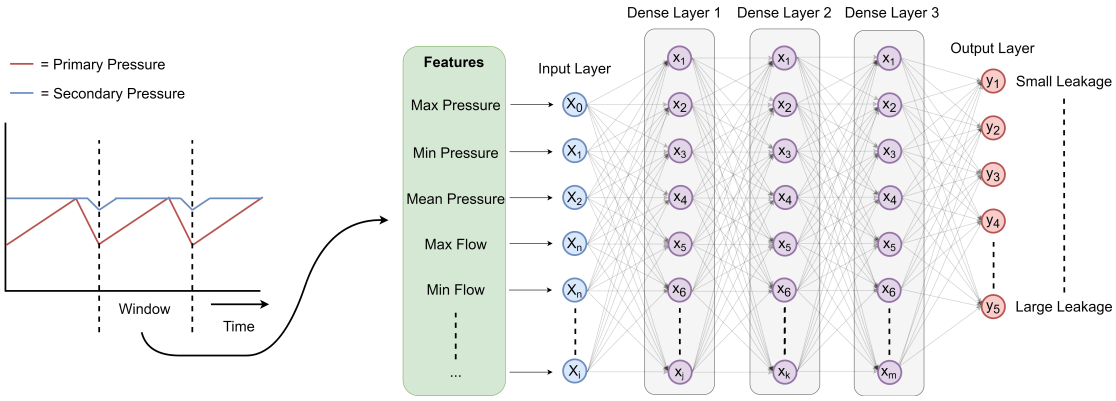


Figure 4.5: Feature-extraction model.

To decrease the training time, an initial batch size of 64 is chosen. After finding the best hyperparameter combinations using Bayesian optimization, the best three combinations are used to again train a model but this time with a reduced batch size of 16. According to Keskar et al. (2016) training with a smaller batch size generally leads to an better model. When this is the case, the newly trained model will be used, otherwise the previously found best model is used.

4.3.2.2 CNN + LSTM Model

In this section more details of the model architecture and the hyperparameter optimization process for the CNN + LSTM based model are shown.

Figure 4.6 shows the model architecture which is based on the work of R. Zhao et al. (2017). Compared with the feature-extraction method, this time no feature aggregation is applied to the selected features, but each signal consisting out of 100 samples is directly used as model input. After feeding the samples to the CNN layer of the network, 2 Bi-LSTM layers, 1 dense layer and finally 1 output layer are used. Again, a batch size of 64 is used during the Bayesian optimization process and a reduced batch size of 16 is used to finalize the training with the best three hyperparameter combinations. Just as with the feature-extraction model, the Adam optimizer with its default parameters will be used to train the model.

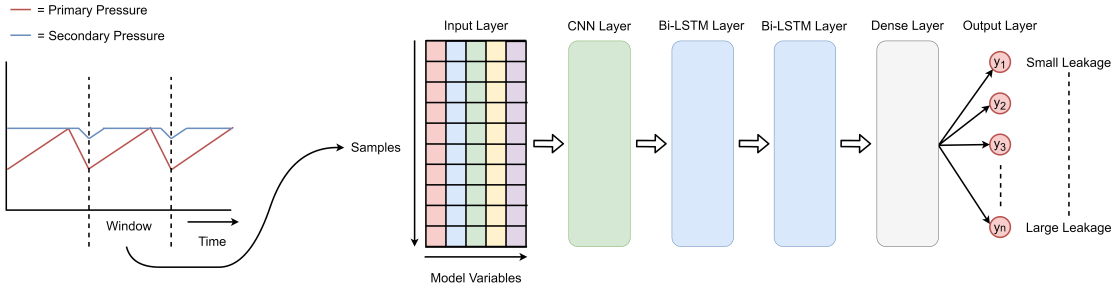


Figure 4.6: CNN + LSTM model based on the work of R. Zhao et al. (2017).

Section 4.3 shows which variables where selected and will be used as input for the machine-learning model. For the feature extraction based model additional feature selection was done by selecting the k-best features and optimizing the k-best value during the hyperparameter optimization process. Because the input for a CNN + LSTM based model consist out of multiple samples, this approach cannot be used to perform additional feature selection. The Python package SHapley Additive exPlanations (SHAP) (Lundberg and S.-I. Lee, 2017) will therefore be used after training to determine the importance of each input variable on the output of the model. If the results show that certain variables do not have an significant influence on any of the model output classes (less than 5%), the feature will be removed and the model will be retrained. Hereafter, the accuracy on the validation dataset of the newly trained model will be determined and compared with the old model. If the accuracy of the model is approximately the same or better as the old model, the newly trained model will be the final model. If this is not the case, the old model will be used as final model. The hope is that by using this selection method variables that introduce noise to the model will be removed and the performance of models will increase.

4.3.2.3 Differentiate Between Faults

The models trained in the two previous sections only focus on predicting the seriousness of a specific fault. For example, one of the models aims to predict how large the primary seal leakage is and another model aims to predict how large the outlet check valve leakage is. Because these individual models are only trained to detect a specific fault and have never seen data of other faults during the training phase, it could be that for multiple models a large leakage is detected and still no conclusion can be drawn for which fault really occurs. To determine which fault really occurs, two different methods will be investigated. Both of these methods will aim to differentiate between the first three faults (F1 - F3).

The first method that will be investigated is to use the normalized probability outputs of the previously trained individual models as inputs for a simple MLP model. A visualization of this method can be seen in Figure 4.7. Hereby the outputs of each individual model will be normalized.

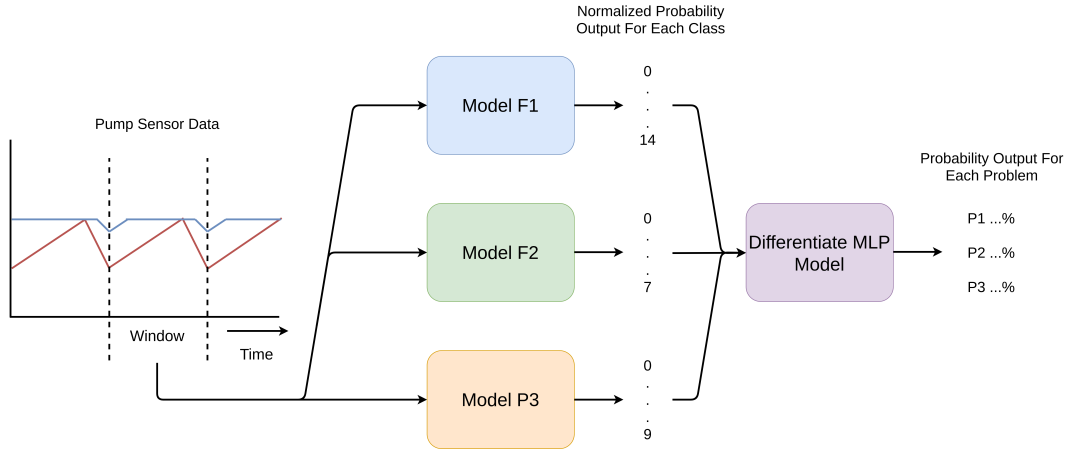


Figure 4.7: Visualization of using the normalized probability outputs of the individual models to differentiate between problem situations.

The second method that will be investigated is to use the previously generated datasets of each fault and train a CNN + LSTM based model directly on the sensor data to differentiate between the faults. Hereby the data can be labeled as follows:

- 3 Classes: F1, F2 and F3
- 4 Classes: F1, F2, F3 and Normal
- 9 Classes: F1_0, F1_1, F1_2, F2_0, ..., F3_2

For the first option with 3 classes, the dataset generated for each fault will receive a label accordingly. For the second option with four classes, again each dataset will receive a label accordingly, but there will also be a "normal" class which represents a correctly functioning pump. Whether a pump is functioning correctly, can be found in Table 4.2. Finally, a last option with 9 classes can further subdivide the correct functionality of the pump. The exact criteria for this can also be found in Table 4.2.

Table 4.2: Subdivision for each fault when using 3, 4 or 9 classes and where Fx stands for F1 for example.

Classes			Secondary Pressure $<$ 500 bar	Secondary Pressure $>=$ 500 bar
3	4	9	Pressure Deviation	Pressure Deviation
Fx	Normal	Fx_0	$<$ 3 bar	$<$ 0.6 %
Fx	Fx	Fx_1	3 - 5 bar	0.6 - 1.0 %
Fx	Fx	Fx_2	$>$ 5 bar	$>$ 1.0 %

Because the pump control software compensates for leakages in the primary chamber by performing more piston takeovers, a large leakage for the primary seal or inlet check valve does not necessarily mean that there is a problem with the functionality of the pump (an error). In order to determine whether there really is an error, the criteria in Table 4.2 will be used which are based on the specifications of the pump. An additional criteria that can be used for primary seal and inlet check valve leakage is the required number of piston takeovers to achieve a certain outlet flow. However, to limit the scope of this project, this part will not be investigated and only the deviation for the secondary pressure will be taken into account. Furthermore, only the first two options with 3 and 4 classes will be investigated and the option with 9 classes will be skipped.

4.3.3 Performance Analysis

The performance of the trained models will be compared on the generated validation data. Because the generated dataset is balanced, it should be possible to use accuracy as performance

metric. Whether the accuracy can really be used as performance metric will be confirmed by also looking at the resulting confusion matrices.

The best performing models on the validation dataset will also be tested in real-time on the real pump system. In order to do this a C++ application will be developed that receives log variables of the pump using UDP and uses the trained models to make predictions in real-time. The choice for C++ was made such that the driver component and diagnostics can be easily implemented in one application as stated in Section 3.1.

Chapter 5

Results

This results chapter shows how the faults were simulated, shows the performance of individual and differentiate diagnostic machine-learning models on the validation data and shows the real-time performance of the best models using the C++ application.

5.1 Simulation of Faults

This Section gives a more detailed explanation of each fault and exactly shows how they were simulated using the 20-sim model of the pump in order to generate the required datasets. Most of the faults could already be simulated without modifying the model and only changing a specific parameter was necessary. However, for some of the faults, additional pump behavior needed to be modeled and added to the current 20-sim model.

5.1.1 Primary Seal Leakage

When looking at the sketch of the pump in Figure 1.3, primary seal leakage is caused by a defective seal at the primary piston. Normally, this seal always leaks a little bit and the pressure in the primary chamber is kept constant by slowly moving the primary piston forward. However, when there is a real problematic seal leakage, the primary piston needs to move forwards faster and can eventually no longer compensate for the pressure drop.

In order to simulate this fault, no changes to the model had to be made and the fault could simply be simulated by changing the leak parameter (G_leak) for the primary piston. Table B.1 in Appendix B shows what parameter values were used to simulate different leak sizes by several flow rates and pressure levels. In order to vary the outlet pressure, the laminar resistance parameter was varied which simulates different types of columns.

5.1.2 Outlet/Inlet Check Valve

The first valve in the sketch of Figure 1.3 is called the inlet check valve and the second valve in this figure is called the outlet check valve. Normally these valves leak a little bit when closed and the pistons can compensate for these leakages by slowly moving forward. However, again it can happen that these leakages become so problematic that the pistons can no longer compensate for this behavior and pressure drops can occur.

Just as with the primary seal leakage, no changes to the model had to be made and the fault could simply be simulated by changing the leak parameter (G_leak) as can be seen in Table B.2 and Table B.3 in Appendix B.

5.1.3 Backlash

Figure 5.1 shows the simulation results when performing a simulation with the backlash component with a backlash value of 10 µm and slope of 10^{10}.

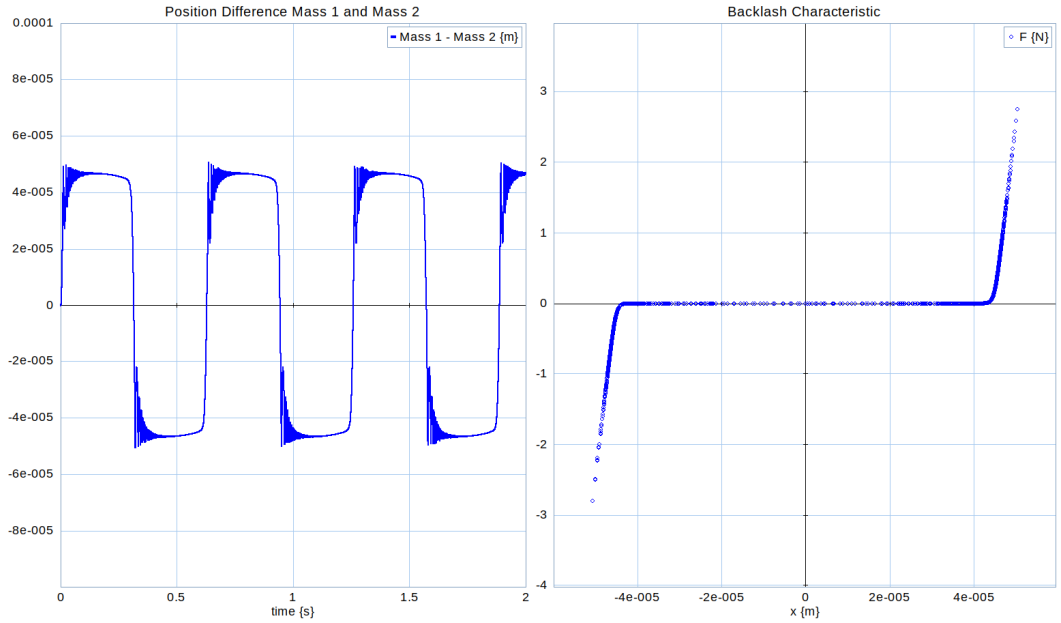


Figure 5.1: Plot of backlash in the motor spindle. The first plot shows the position difference of the two masses over time when given an sine wave as velocity input. The second plot shows the force/position relation. As can be seen in both plots, the backlash value is approximately 10 µm.

Table B.4 in Appendix B shows the parameter ranges used to perform the simulations for different backlash values at several pressure levels and for multiple flow rates.

5.1.4 Pump Head Blockage

When looking at the sketch of the pump in Figure 1.3, a pump head blockage occurs in the flow path between the primary- and secondary chamber and means that there is an obstruction of the flow between these chambers. Because the exact position of the obstruction does not influence the behavior of the fault, the fault can be simulated by changing the flow passage parameter for the outlet check valve (G_klep) as can be seen in Table B.5 of Appendix B.

5.2 Diagnostic Machine-Learning Models

In this Section the results of the hyperparameter optimization process using Bayesian optimization and the final performance of both the feature-extraction models and CNN + LSTM based models for in total five faults are shown. In the first Subsection the results for the first three faults are shown using a full window of the repeating pattern. Next, the second Subsection shows the results of the backlash model and the third Subsection shows the results of using two different window sizes to detect a pump head blockage.

5.2.1 Primary Seal and Outlet/Inlet Check Valve Leakage

The first Section of Appendix G shows the search ranges and the final values for each hyperparameter for the first three faults of the feature-extraction models. Next, in Appendix H the confusion matrix for the best feature-extraction models can be seen. Finally, in Table 5.1 the accuracy scores on the validation set can be seen for the best models.

The second Section of Appendix G shows the hyperparameters of the CNN + LSTM based models for the first three faults. Next, in Appendix H the confusion matrix for the best CNN + LSTM based models can be seen and Table 5.1 shows the accuracy scores on the validation set for the best models. This Table also shows the accuracy scores for the retrained model using only the variables that had an influence of at least 5% on the model output according to the SHAP package.

Table 5.1: Accuracy scores feature extraction and CNN + LSTM based models for the first three faults and using an optimal number of classes.

Fault	Num. Classes	Feature Extraction	CNN + LSTM	CNN + LSTM (Selection)
Primary Seal Leakage	15	56.5%	97.7%	87.9%
Outlet Check Valve Leakage	8	48.1%	75.7%	71.5%
Inlet Check Valve Leakage	10	66.1%	95.4%	95.7%
Average Accuracy		56.9%	89.6%	85.0%

Appendix J shows which variables were chosen for the retrained models. The percentages in these tables are determined by calculating the influence of each variable for a specific output and thereafter taking the maximum percentage value for each variable over all the outputs. For example, the model to detect a primary seal leakage has 15 classes and thus 15 outputs. For each output the influence of each variable is calculated which results in $15 \times 9 = 135$ percentage values. Hereafter the maximum for each variable over the 15 outputs is taken and used in the table. The sum of these percentages in this table can therefore be higher than 100%.

5.2.2 Backlash

As can be seen in Subsection 5.2.1 the CNN + LSTM based model performs way better than a model based on feature extraction. This Subsection will therefore use a CNN + LSTM based model to detect backlash. After analyzing the differences between smaller and larger values for backlash, it is found that hardly any difference can be seen for the sensor signals. This is confirmed by looking at the resulting confusion matrix for a model trained on full windows and only 3 classes for the output as can be seen in Figure 5.2.

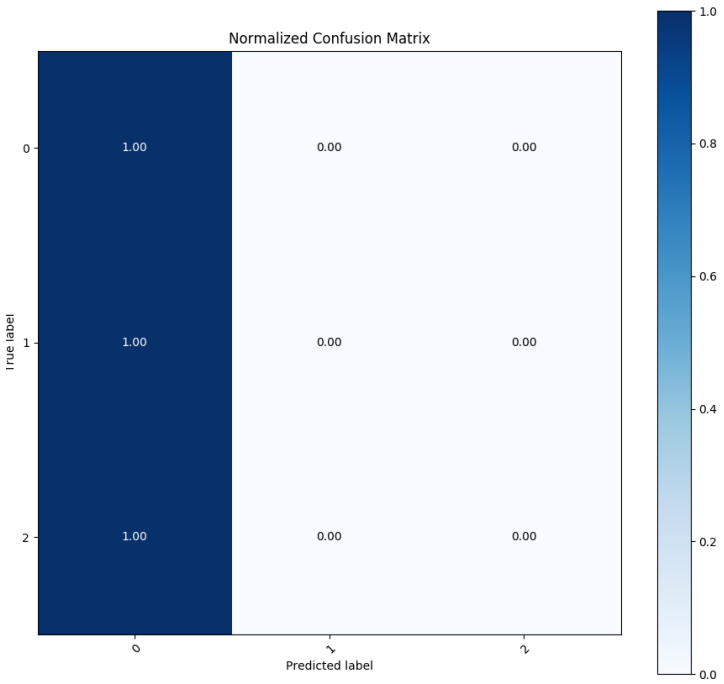


Figure 5.2: Normalized confusion matrices of a CNN + LSTM based model for backlash in motor spindle (F4).

5.2.3 Pump Head Blockage

In this Subsection again a CNN + LSTM based model is used to compare the difference between using a full window versus using a specific smaller window. After analyzing the differences between a small and larger pump head blockage, it is found that the fault can be mainly detected by looking at the sensor signals in the transition state of the pump as can be seen in Figure 2 of Appendix F. When using this transition state as window and optimizing the hyperparameters of the CNN + LSTM based model as can be seen in Table G.7 of Appendix G, an accuracy of 83.9% is achieved. When using a full window as input and optimizing the hyperparameters of the CNN + LSTM based model as can be seen in Table G.8 of Appendix G, an accuracy of 79.9% is achieved. The corresponding confusion matrices of both trained models can be found in Figure 5.3 and 5.4 respectively.

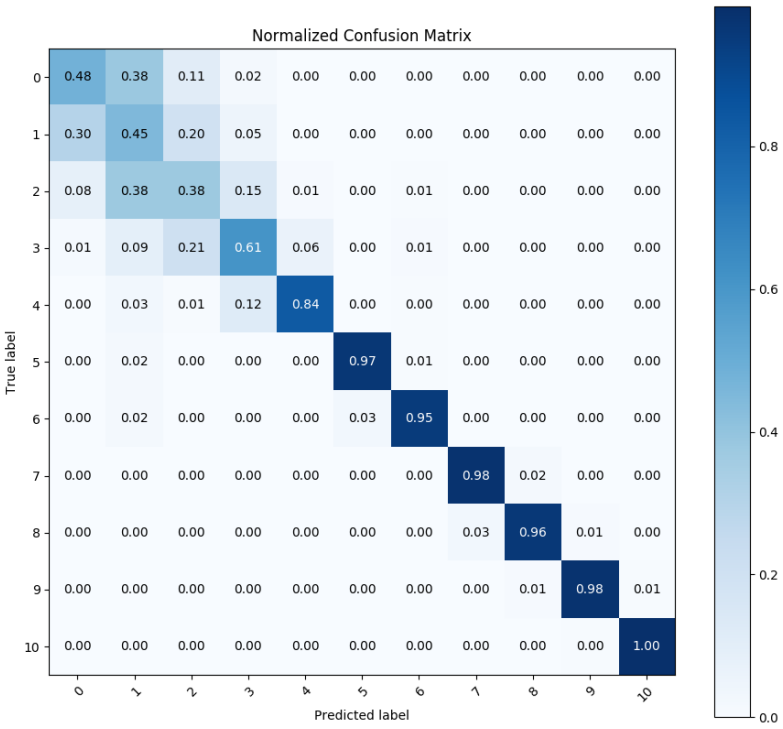


Figure 5.3: Normalized confusion matrix of a CNN + LSTM based model on a full window for pump head blockage (F5) with an accuracy of 79.9%.

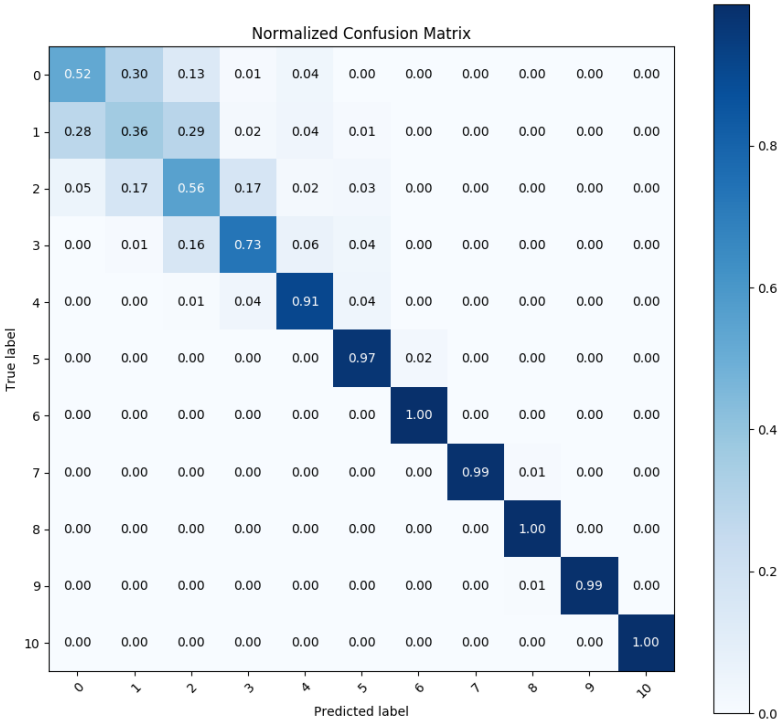


Figure 5.4: Normalized confusion matrix of a CNN + LSTM based model on a smaller window for pump head blockage (F5) with an accuracy of 83.9%.

In addition to these results, again an analysis is done to determine the most important input variables using the SHAP package for which the results can be found in Table J.4 of Appendix J. Furthermore, the importance of each sample in a full window and smaller window is analyzed using SHAP which can be seen in Figure 5.5 and 5.6 respectively. Finally, a comparison of using a full window and smaller window for a simulation of problematic and non-problematic pump head blockage can be seen in Appendix K.

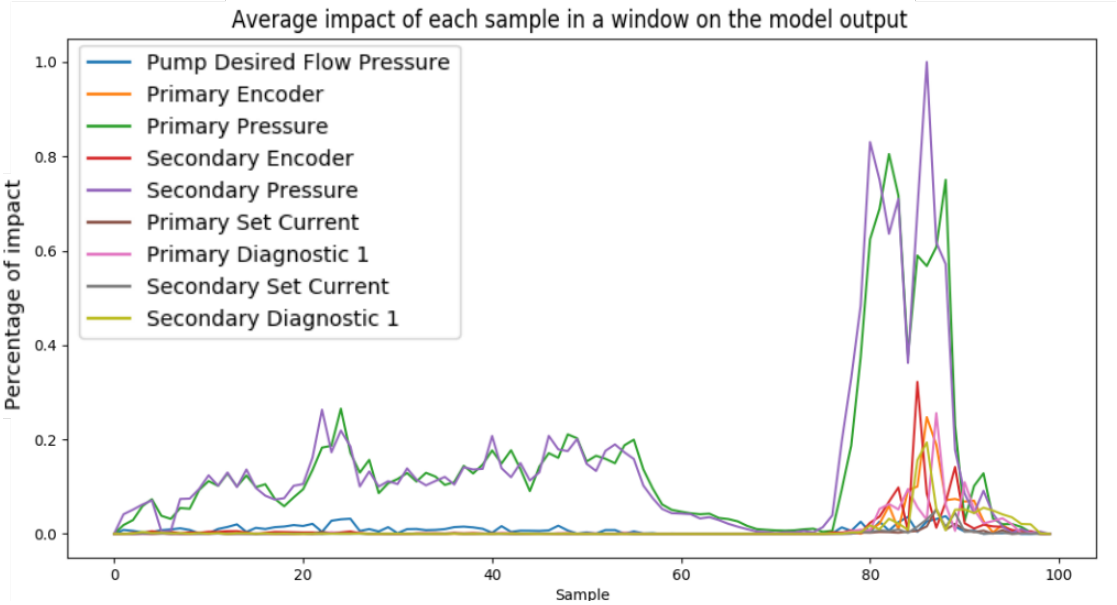


Figure 5.5: Average percentage impact of a sample on the output of a model to detect pump head blockage (F5) when using a full window.

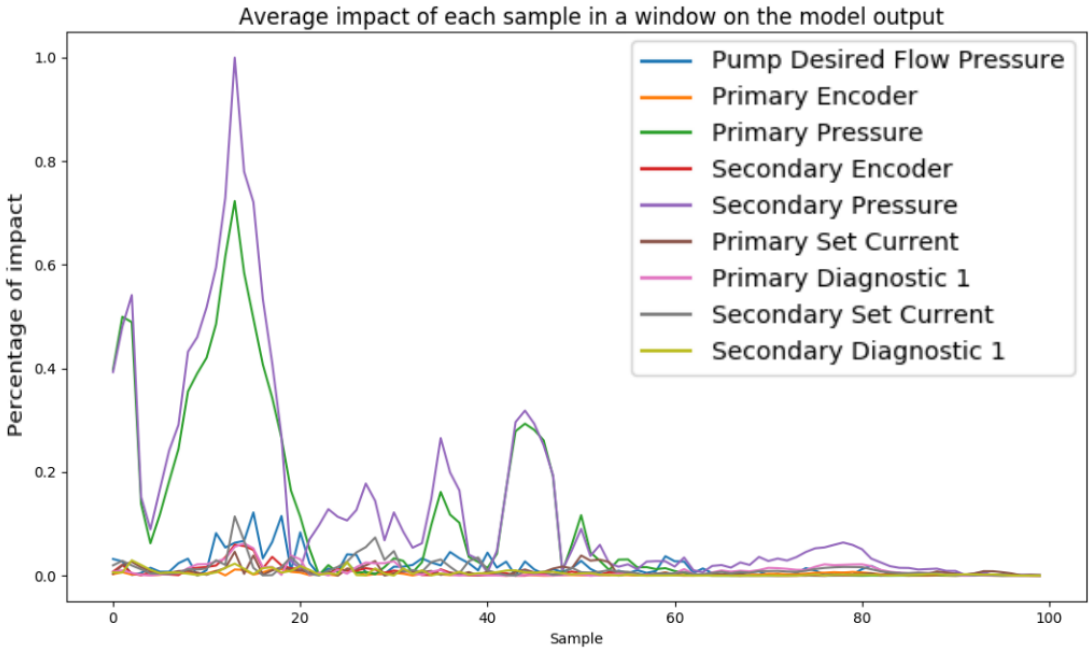


Figure 5.6: Average percentage impact of a sample on the output of a model to detect pump head blockage (F5) when using a smaller window.

5.3 Differentiate Between Faults

Appendix I shows the resulting confusion matrices when making predictions on a dataset of certain fault with a CNN + LSTM based model trained to detect a different fault. As can be seen from these confusion matrices it is especially hard to differentiate between fault F1 and F3. It is therefore expected that the first method, as seen in Figure 4.7, also has difficulty to differentiate between these faults. After implementing the method, it is found that the MLP model does not converge and cannot be used to differentiate between the faults.

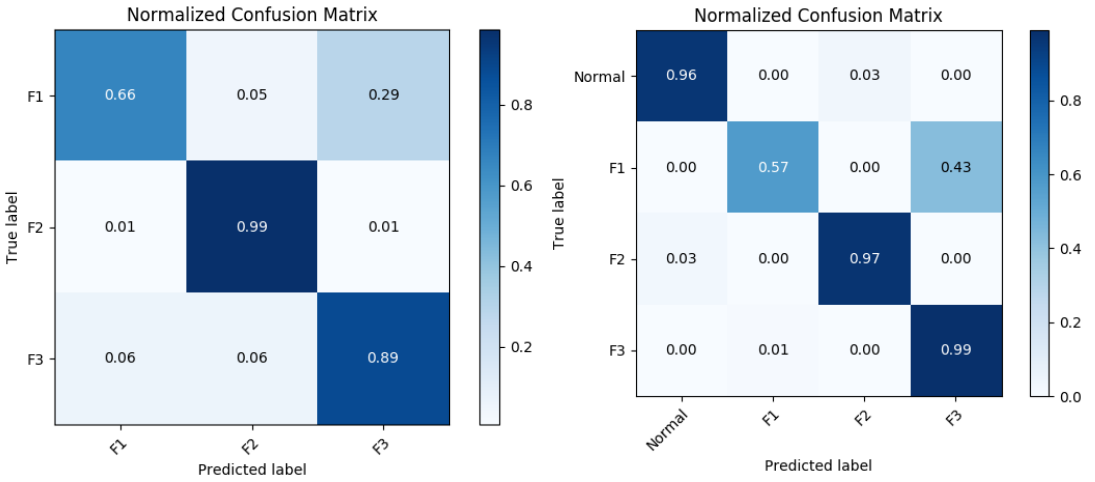


Figure 5.7: Normalized confusion matrices of a CNN + LSTM based model to differentiate between problem F1, F2 and F3 (left) and normalized confusion matrices of a CNN + LSTM based model to differentiate between problem F1, F2, F3 and a normal functioning pump (right). The confusion matrix on the left has an accuracy of 88.1% and confusion matrix on the right has an accuracy of 92.5%.

The results of the second method, as described in Sub-subsection 4.3.2.3, shows that there is still a relatively large correlation between F1 and F3. However, using the CNN + LSTM based model it is possible to differentiate between the faults with an accuracy of 88.1% when using 3 classes and 92.5% when using 4 classes. The corresponding confusion matrices of this second method can both be found in Figure 5.7.

5.4 Model Performance on the Real Pump System

As described in Subsection 4.3.3 the performance of the trained models will also be validated on the real pump system using a C++ application. A flowchart of this developed C++ application can be found in Appendix L.

The application receives UDP packages of the real pump system and adds the packages to a buffer until a full window is received. Next, the buffer is resampled to a window of 100 samples and passed to a predict function of a developed Tensorflow Dynamic-Link Library (DLL) that has loaded the trained CNN + LSTM based models. Since the real pump system consist out of two pumps (A and B), predictions of both pumps are printed to the console and can be analyzed.

Screen prints of the application and the console of the real pump system can be found Appendix M. Furthermore, a mapping from each class to the leakage in µl/min can be found in Appendix N.

The screen prints of the application show that the application works as expected for several seal and valve leakages. However, it is hard to validate whether the precise prediction of the class is also correct. This is because leakages of only several µl/min can hardly be measured by any instrument.

Chapter 6

Discussion

Chapter 5 shows how in total five faults were simulated and how the simulations were used to train and validate several diagnostic machine-learning models. This chapter will give an in depth explanation of these results.

6.1 Simulation of Faults

Section 3.2 shows that there are in total eight faults that are important to detect. To limit the scope of this assignment, it was chosen to focus on the five most important ones which are:

- F1. Primary Seal Leakage
- F2. Outlet Check Valve Leakage
- F3. Inlet Check Valve Leakage
- F4. Backlash
- F5. Pump Head Blockage

Section 5.1 shows that most of these faults could already be simulated by only modifying some parameters in the 20-sim model of the pump. However, this did not apply for backlash in the spindle of the pump which required some additional modeling and was then added to the original model of the pump. This immediately showed one of the disadvantages of using a model based approach to train diagnostic machine-learning models; the simulation of some faults can take a lot of time.

In the case of simulating backlash, the simulation time increased with a factor 10 compared with the original simulation model. It took multiple days to perform all the simulations to generate the training dataset and in the end it was found that no machine-learning model could be trained on this dataset. It is therefore recommended to first determine whether the fault can be measured directly using the sensor data instead of using a machine-learning approach. In the case of backlash, it is possible to directly measure the amount of backlash using one of the available pump sensors which is more reliable and requires less developing time. For the other faults this was not the case and the model based machine-learning method is probably a better approach.

6.2 Diagnostic Machine-Learning Models

When comparing the feature-extraction method and CNN + LSTM based models in Table 5.1, it can be seen that for the first three faults a CNN + LSTM based model scores averagely a 32.7% higher accuracy. From this can be concluded that a CNN layer can successfully be used to perform feature extraction and a CNN + LSTM based model can be used to detect faults in a UHPLC pump. Only using features that have at least a 5% influence on the model output reduces the accuracy by approximately 4.6%. It is therefore recommended to not use a feature selection method since this work can be outsourced to the CNN layer of the model.

A model based method to train diagnostic machine-learning models does not always work as can be seen in the case of backlash. Even when only using 3 classes a CNN + LSTM based model does not converge. However, when the method does work, it proves to be very robust as can be seen in the case of a pump head blockage. Here using a full window was compared with using a smaller window which resulted in an accuracy difference of only 4%. That the selection of the window has almost no influence can also be seen when looking at the sample importance in a window in Figure 5.5 and 5.6 and the corresponding simulations in Appendix K. In these figures can be seen that the CNN + LSTM based model just focuses on smaller region of a window when using a full window as input. This means that 100 samples for a full window are

enough to detect and classify a fault that can only be recognized in a fraction of this window. Of course this will be different when peaks in the sensor signal become smaller. It is therefore important to always check whether the resampled window is still a good representation of the original sensor signals.

6.3 Differentiate Between Faults

In order to differentiate between faults two different methods were investigated. The first method, as mentioned in Sub-subsection 4.3.2.3, uses the normalized probability outputs of the individual models as input for a simple MLP model. During the training of this model it was found that the model did not converge and cannot be used to differentiate between faults. An explanation for this disappointing result might be that the CNN layer of the individual models only selects features that are important to classify the seriousness of a fault and features that are important to differentiate between faults are ignored. Furthermore, because there is a large correlation between fault F1 and F3 it is even harder to differentiate between the faults and the method becomes completely useless.

On the other hand, the second method mentioned in Sub-subsection 4.3.2.3 shows some decent results and can be used to differentiate between faults. An explanation that this method does work might be that the CNN + LSTM based model works directly on the sensor data and therefore finds features that can be used to differentiate between faults. Another interesting result of this method is that using 4 classes performs better than using 3 classes (92.5% vs 88.1% accuracy). An explanation for this might be that each fault contains data that can be seen as a normal functioning pump, but was labeled as problematic when using 3 classes. Whereas it was labeled as *normal* when using 4 classes.

Finally, both methods show that there is a large correlation between fault F1 and F3. An explanation for this large correlation is that a primary seal leakage (F1) and inlet check valve leakage (F3) can both be seen as a leakage in the primary chamber of the pump (see Figure 1.3). Because they can both be seen as a leakage in the primary chamber of the pump, they also result in a similar deviation for the sensor signals and therefore have a high correlation.

6.4 Model Performance on the Real Pump System

The goal of the C++ application was to have a method to validate the trained diagnostic machine-learning models on the real pump systems. The console output of the developed C++ application shows that there is still a large correlation between Fault F1 and F3, but also shows that it is possible to detect several seal and valve leakages in real-time using a UDP connection. In particular an outlet check valve leakage in pump B can be correctly classified by the individual and the differentiate model with 4 classes (Normal, F1, F2 and F3). Since the other differentiate model with 3 classes (F1, F2 and F3) gives a false positive for pump A, it is recommend to not use such a differentiate model.

It is hard to validate the exact leakage values of the individual models since a leakage of several µl/min can hardly be measured by any instrument. However, it is possible te check the predicted leakage values for a correctly functioning pump and use these values as the standard values for the pump. When one of these values are thus exceeded, it can be seen as a problematic leakage. In addition to this, when multiple faults would occur at the same time, the trained diagnostic models might have a hard time to correctly classify which fault really occurs.

Chapter 7

Conclusion

The research in this assignment consisted out five sub-questions which are answered below:

What are important faults to detect and how can they be simulated using the provided 20-sim model of the pump?
The most important faults that are detected in the UHPLC pump used in this assignment are seal leakages, valve leakages and obstructions in flow paths. These faults could simply be simulated by only changing a parameter in the provided 20-sim model of the UHPLC pump. The simulation of backlash is also possible, but takes a lot of time and this fault could not be detected using the trained machine-learning models.

Can a CNN be used to perform feature extraction and can a model based on CNN + LSTM be used to detect faults in a pump?
In this assignment a CNN is successfully used to perform feature extraction for several faults. Furthermore, a combination of a CNN and LSTM could successfully be used to detect faults in an UHPLC pump with an average accuracy of 89.6%.

How does a CNN + LSTM based method compare with a method based on feature extraction?
A CNN + LSTM based method scored approximately a 32.7% higher accuracy compared with a MLP model using feature extraction. Furthermore, the method was more robust and it was not necessary to use a feature selection method since this work could be outsourced to the CNN layer of the model.

What method can be used to differentiate between faults?
Using a CNN + LSTM based model that was trained on several faults, it was possible to successfully differentiate between faults in an UHPLC pump. Another method that used the normalized probability outputs of machine-learning models trained to detect individual faults did not work.

How can the final implementation of the diagnostic machine-learning models be validated?
The best performing diagnostic machine-learning models were validated on a real pump system using a developed C++ application. Using the C++ application it was possible to correctly detect several seal and valve leakages in the real pump system. However, the detection of backlash was not possible.

Using the answers of these sub-questions the main research question of this assignment can be answered:

To what extend can a machine-learning method trained on a simulation model be used to detect faults in a real pump?

Summarized, a CNN + LSTM based diagnostic machine-learning model trained on a simulation model performs the best and can be successfully used to detect several seal and valve leakages in a real pump. However, there are a number of things to take into account.

At first, an accurate simulation model of the system is required because the diagnostic machine-learning model is as good as the simulation model used to generate the training sets. This simulation model should not only be valid for the normal behavior of the system, but also for the simulation of problematic behavior of the system. For example, make sure that a pressure signal is in a valid range and not below minus 1 bar which is physically impossible.

Secondly, a model based approach to train diagnostic machine-learning models is not always the best approach as can be seen in the case of backlash. It is therefore important to always

check whether the fault can be detected using a sensor signal directly instead of using machine learning.

Finally, when a model based approach is used, it is important to check whether the resampled window is still a good representation of the original sensor signals. For the pump system used in this research a window of 100 samples was good enough, but this might be different when a system contains more high-frequency signals.

7.1 Future Work

A number of things might improve the work presented in this report. Right now the maximum number of classes that can be used to classify a certain fault can only be found using a trial and error method. It might be interesting to perform more research in an unsupervised based approach or better method to determine the maximum number of classes. In this way an optimum can be found between the precision and maximum accuracy of the models.

Furthermore, when training a model to differentiate between faults it is important to define a normal class. In addition to this, a technique that might be interesting when adding more faults to this differentiate model is continuous learning. With continuous learning it is possible to reuse the already trained differentiate model and only train the model to detect new classes which would improve the training time in theory.

Other things that might be interesting to investigate are performing sensitivity analyses for noise on sensor signals, using a variable sampling method instead of the used constant sampling method and performing more research on how to detect cross-correlations of faults such that trained models become even more accurate.

Appendix A

Pump Behavior

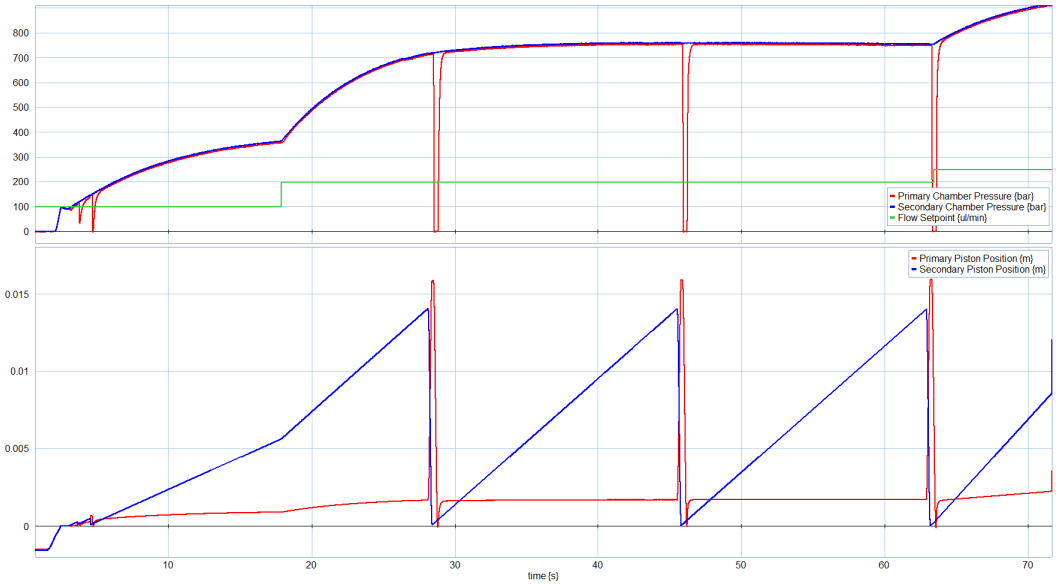


Figure 1: Example of normal behavior for the pump Weustink and Ekris, 2019.

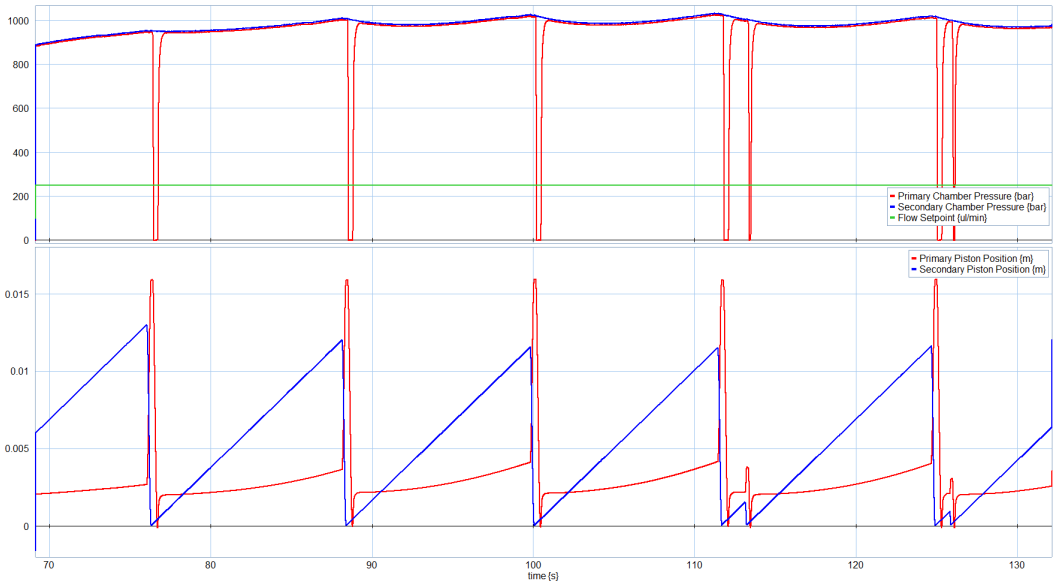


Figure 2: Example of a problem situation inside the pump, non-linear relation between pressure and leakage Weustink and Ekris, 2019.

Appendix B

Parameter Lists

Table B.1: Primary Seal Leakage

Model Parameter	Min	Max	Step Size	Unit
Firmware.setFlow.flow	500	1000	100	μl/min
Hydraulics.LaminarResistance.R	3e15	1e16	1e15	-
Hydraulics.PC.G_leak	1e-19	2e-17	$\begin{cases} \text{1e-19}, & \text{if } a <= \text{1e-18} \\ \text{1e-18}, & \text{otherwise} \end{cases}$	μl/min/bar

Table B.2: Outlet Check Valve

Model Parameter	Min	Max	Step Size	Unit
Firmware.setFlow.flow	500	1000	100	μl/min
Hydraulics.LaminarResistance.R	3e15	1e16	1e15	-
Hydraulics.SCV.CheckValve.G_leak	1e-20	1e-17	$\begin{cases} \text{1e-20}, & \text{if } a <= \text{1e-19} \\ \text{2e-19}, & \text{otherwise} \end{cases}$	μl/min/bar

Table B.3: Inlet Check Valve

Model Parameter	Min	Max	Step Size	Unit
Firmware.setFlow.flow	500	1000	100	μl/min
Hydraulics.LaminarResistance.R	3e15	1e16	1e15	-
Hydraulics.PCV.CheckValve.G_leak	1e-20	1e-17	$\begin{cases} \text{1e-20}, & \text{if } a <= \text{1e-19} \\ \text{2e-19}, & \text{otherwise} \end{cases}$	μl/min/bar

Table B.4: Backlash (Ex5 and Ex12)

Model Parameter	Min	Max	Step Size	Unit
Firmware.setFlow.flow	500	1000	100	μl/min
Hydraulics.LaminarResistance.R	3e15	1e16	1e15	-
PP_MotorSpindle.Backlash.counts	125	5000	$\begin{cases} 125, & \text{if } a <= 1000 \\ 250, & \text{otherwise} \end{cases}$	encoder counts

Table B.5: Pump Head Blockage

Model Parameter	Min	Max	Step Size	Unit
Firmware.setFlow.flow	500	1000	100	μl/min
Hydraulics.LaminarResistance.R	3e15	1e16	1e15	-
Hydraulics.SVC.CheckValve.G_klep	5e-16	2.7e-12	$\begin{cases} 5e\text{-}16, & \text{if } a <= 1e\text{-}14 \\ 1e\text{-}14, & \text{otherwise} \\ 1e\text{-}13, & \text{if } a > 1e\text{-}13 \end{cases}$	μl/min/bar

Appendix C

Variable List

Table C.1: Variable List

Model Variable	Used/Not Used	Clarification
Pump Control Mode	Not Used	Variance equal to zero
Pump Control Command	Not Used	Variance equal to zero
Pump Desired Flow Pressure	Used	Useful when a problem is flow/pressure related
Primary Encoder	Used	Position of the primary piston
Primary Pressure	Used	Pressure in the primary chamber
Primary Hall Sensor Flag	Not Used	Variance equal to zero
Primary Latched Encoder Position	Not Used	Variance equal to zero
Primary Maximum Current	Not Used	Variance equal to zero
Primary Maximum Velocity	Not Used	Variance equal to zero
Secondary Encoder	Used	Position of the secondary piston
Secondary Pressure	Used	Pressure in the secondary chamber
Secondary Hall Sensor Flag	Not Used	Variance equal to zero
Secondary Latched Encoder Position	Not Used	Variance equal to zero
Secondary Maximum Current	Not Used	Variance equal to zero
Secondary Maximum Velocity	Not Used	Variance equal to zero
Position Sensors	Not Used	Variance almost equal to zero
Primary Desired Pressure	Not Used	Variance equal to zero
Secondary Desired Pressure	Not Used	Variance equal to zero
Secondary Position Fixed	Not Used	Variance equal to zero
Pump Control Version	Not Used	Variance equal to zero
Pump Control Status	Not Used	Variance almost equal to zero
Pump Control Error	Not Used	Variance equal to zero
Pump Compressibility Factor	Not Used	Compressibility estimation, but changes to slowly
Pump Check Valve Delay	Not Used	Variance almost equal to zero
Pump Diagnostics 1	Not Used	Minimum ripple estimation, but changes to slowly
Pump Diagnostics 2	Not Used	Maximum ripple estimation, but changes to slowly
Primary Pressure Filtered	Not Used	Both models have a filtering behavior, no need to add this extra variable
Secondary Pressure Filtered	Not Used	Both models have a filtering behavior, no need to add this extra variable

Model Variable	Used/Not Used	Clarification
Enable End Stops	Not Used	Variance equal to zero
Primary Set Current	Used	Current setpoint for primary piston motor
Primary End Position Forward	Not Used	Variance equal to zero
Primary End Position Backward	Not Used	Variance equal to zero
Primary Leakage	Not Used	Primary leakage estimation, but changes to slowly
Primary Diagnostics 1	Used	Reference position of the primary piston
Primary Diagnostics 2	Not Used	Reference velocity of the primary piston, but useless without a real velocity measurement
Primary Diagnostics 3	Used in pre-processing	Logic State Variable
Primary Diagnostics 4	Not Used	Primary PID output, scaled version of to Primary Set Current, so no need to add this extra variable
Secondary Set Current	Used	Current setpoint for secondary piston motor
Secondary End Position Forward	Not Used	Variance equal to zero
Secondary End Position Backward	Not Used	Variance equal to zero
Secondary Leakage	Not Used	Variance equal to zero
Secondary Diagnostics 1	Used	Reference position of the secondary piston
Secondary Diagnostics 2	Not Used	Reference velocity of the secondary piston, but useless without a real velocity measurement
Secondary Diagnostics 3	Used in pre-processing	Logic State Variable
Secondary Diagnostics 4	Not Used	Secondary PID output, similar to Secondary Set Current so no need to add this extra variable
Synchronize	Not Used	Variance equal to zero
Pump Time To Next Take Over	Not Used	Directly linked to Pump Desired Flow Pressure, no need to add this extra variable
Primary Compressed Stroke	Not Used	Variance equal to zero
Secondary Compressed Stroke	Not Used	Variance equal to zero

Appendix D

Pseudocode Algorithms

Algorithm 1 Pre-Processing

```
Set raw data directory
Set train directory
Set validation directory
Set NumberOfSamples to 100
Define min-max scaler
for train_xxx.csv in raw data directory do
    Select variables
    Scale the data
    Determine start and end indices of windows
    for Window in train_xxx.csv do
        if Window Size < NumberOfSamples then
            Skip window
        else
            Resample window to NumberOfSamples
            Save resampled window as train_xxx_xxx.csv in train directory
Randomly move 20% of saved CSVs to validation directory
```

Appendix E

Feature Formulas

For each signal that was selected, as can be seen in Appendix C, the following formulas are used to calculate the features when using a window of *N* samples *X*:

Table E.1: Feature Formulas

Feature	Symbol	Formula/Function
Mean	μ	$\frac{1}{N}(\sum_{i=1}^{N} x_i)$
Standard Deviation	σ	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$
Derivative	m	$\frac{x_{N-1}-x_0}{N}$
Kurtosis	Kurt[x]	$\mathbb{E}[(\frac{X-\mu}{\sigma})^4]$
Minimum	min	$numpy.min(X)$
Maximum	max	$numpy.max(x)$
Median	median	$numpy.median$
Sample Entropy		$-log\frac{A}{B}$
Skewness	$\widetilde{\mu}_3$	$\mathbb{E}[(\frac{X-\mu}{\sigma})^3]$
Variance	Var(X)	$\sum_{i=1}^{N} p_i \cdot (x_i - \mu)^2$
Count Above Mean		$numpy.where(x > \mu)[0].size$
Count Below Mean		$numpy.where(x < \mu)[0].size$
Mean Change		$\frac{1}{N}\sum_{i=1}^{N} x_{i+1} - x_i$
Mean Absolute Change		$\frac{1}{N}\sum_{i=1}^{N} \lvert x_{i+1} - x_i \rvert$
Mean Second Derivative Central		$\frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}(x_{i+2} - 2 \cdot x_{i+1} + x_i)$
Absolute Energy		$\sum_{i=1}^{N}(x_i)^2$
Absolute Sum of Changes		$\sum_{i=1}^{N} \lvert x_{i+1} - x_i \rvert$

Appendix F

Simulation Windowing

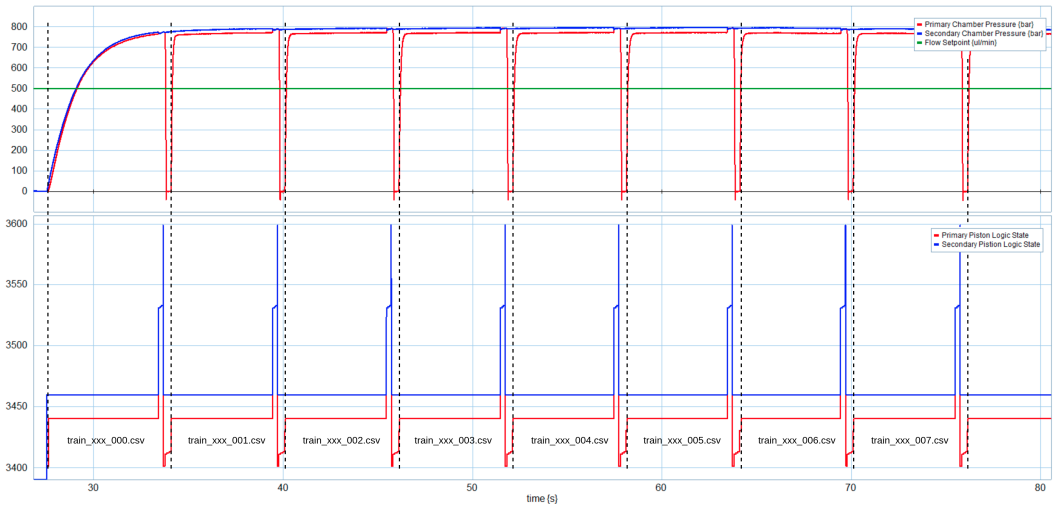


Figure 1: Resulting data files after windowing a simulation using full windows.

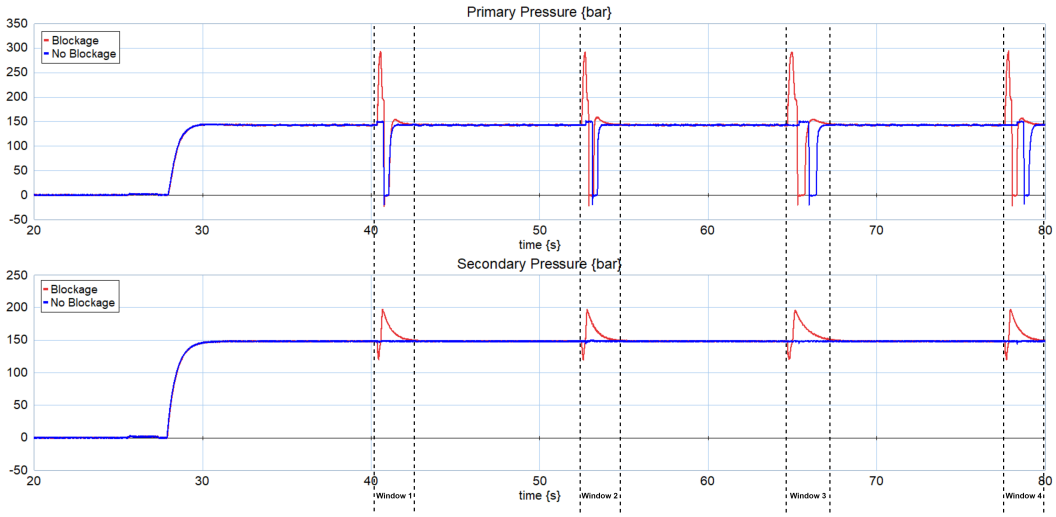


Figure 2: Resulting data files/windows of a simulation after using the transition state of the pump as window for a pump head blockage.

Appendix G

Hyperparameter Optimization

G.1 Feature Extraction Model

Table G.1: Hyperparameter optimization primary seal leakage (F1).

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Neurons Dense 1	5	600	1	130
Neurons Dense 2	5	600	1	78
Neurons Dense 3	5	600	1	164
Number of Layers	1	3	1	3
K-Best Value	2	50	1	40

Table G.2: Hyperparameter optimization outlet check valve (F2).

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Neurons Dense 1	5	600	1	5
Neurons Dense 2	5	600	1	309
Neurons Dense 3	5	600	1	x
Number of Layers	1	3	1	2
K-Best Value	2	50	1	35

Table G.3: Hyperparameter optimization inlet check valve (F3).

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Neurons Dense 1	5	600	1	250
Neurons Dense 2	5	600	1	68
Neurons Dense 3	5	600	1	128
Number of Layers	1	3	1	3
K-Best Value	2	50	1	33

G.2 CNN + LSTM Model

Table G.4: Hyperparameter optimization primary seal leakage (F1).

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Filter	32	64	32	32
Kernel Size	1	5	1	4
Pool Size	1	5	1	3
Neurons LSTM 1	5	600	1	397
Neurons LSTM 2	5	600	1	446
Neurons Dense	5	600	1	335

Table G.5: Hyperparameter optimization outlet check valve (F2).

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Filter	32	64	32	32
Kernel Size	1	5	1	2
Pool Size	1	5	1	3
Neurons LSTM 1	5	600	1	252
Neurons LSTM 2	5	600	1	443
Neurons Dense	5	600	1	135

Table G.6: Hyperparameter optimization inlet check valve (F3).

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Filter	32	64	32	32
Kernel Size	1	5	1	3
Pool Size	1	5	1	2
Neurons LSTM 1	5	600	1	398
Neurons LSTM 2	5	600	1	91
Neurons Dense	5	600	1	259

Table G.7: Hyperparameter optimization pump head blockage (F5)

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Filter	32	64	32	32
Kernel Size	1	5	1	4
Pool Size	1	5	1	2
Neurons LSTM 1	5	600	1	155
Neurons LSTM 2	5	600	1	136
Neurons Dense	5	600	1	71

Table G.8: Hyperparameter optimization pump head blockage for a smaller window (F5).

Hyperparameter	Bounds			Final Value
	Min	Max	Step	
Filter	32	64	32	32
Kernel Size	1	5	1	3
Pool Size	1	5	1	1
Neurons LSTM 1	5	600	1	244
Neurons LSTM 2	5	600	1	446
Neurons Dense	5	600	1	446

Appendix H

Confusion Matrices

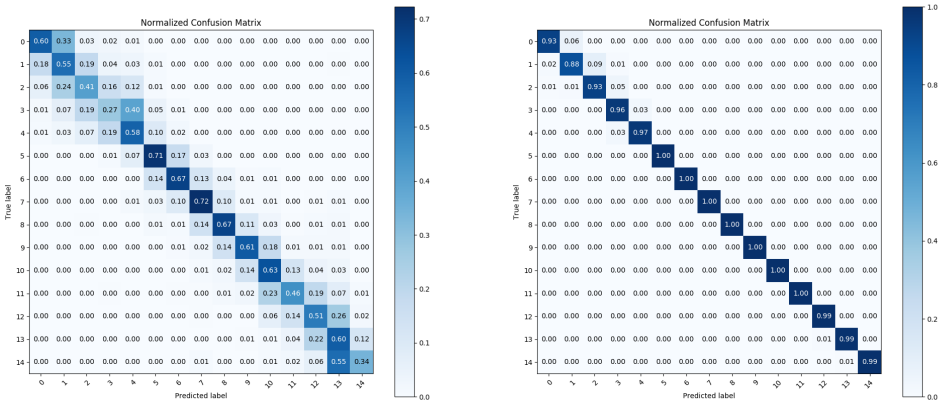


Figure 1: Normalized confusion matrices of the feature extraction model (left) and CNN + LSTM based model (right) for primary seal leakage (F1).

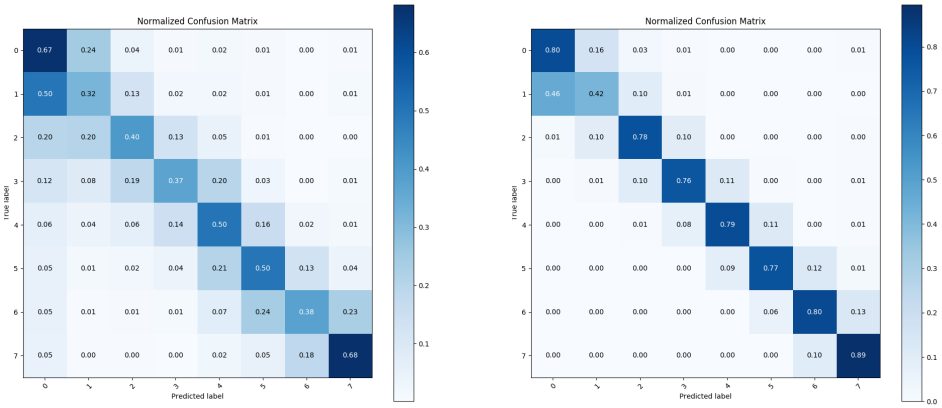


Figure 2: Normalized confusion matrices of the feature extraction model (left) and CNN + LSTM based model (right) for outlet check valve leakage (F2).

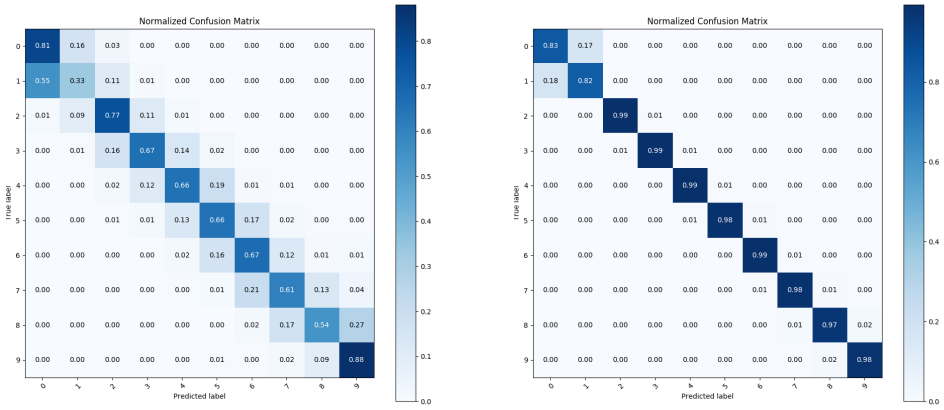


Figure 3: Normalized confusion matrices of the feature extraction model (left) and CNN + LSTM based model (right) for inlet check valve leakage (F3).

Appendix I

Confusion Matrices Predictions

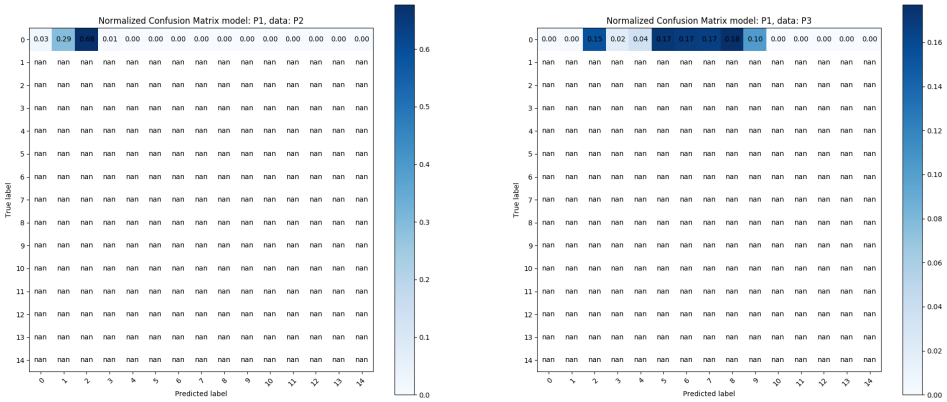


Figure 1: Normalized confusion matrices for the predictions on outlet check valve leakage (F2) (left) and inlet check valve leakage (F3) (right) data using a CNN + LSTM based model trained to detect primary seal leakage (F1).

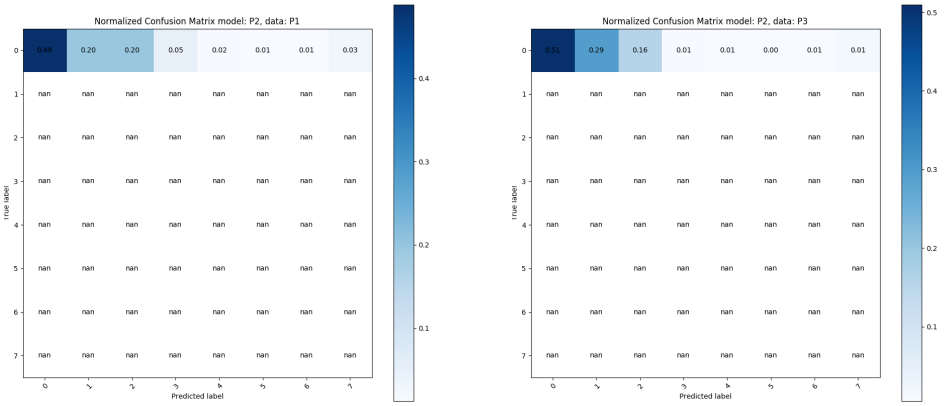


Figure 2: Normalized confusion matrices for the predictions on primary seal leakage (F1) (left) and inlet check valve leakage (F3) (right) data using a CNN + LSTM based model trained to detect outlet check valve leakage (F2).

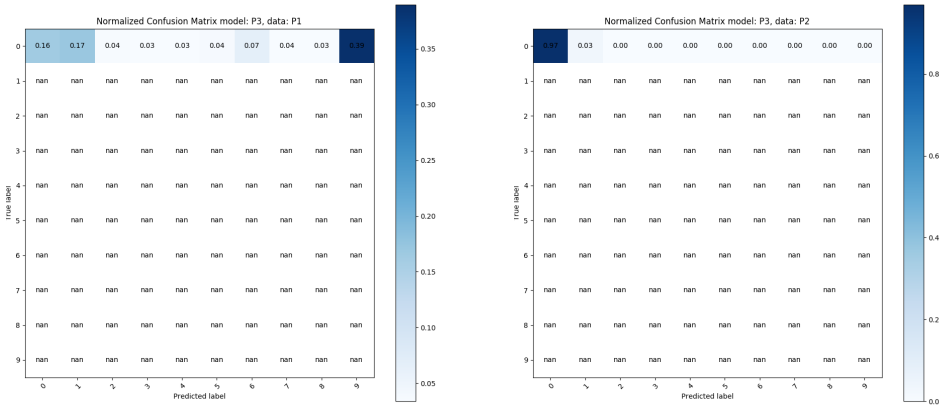


Figure 3: Normalized confusion matrices for the predictions on primary seal leakage (F1) (left) and outlet check valve leakage (F2) (right) data using a CNN + LSTM based model trained to detect inlet check valve leakage (F3).

Appendix J

Variable Importance

Table J.1: Maximum influence on model output for 15 classes and a CNN + LSTM based model to detect primary seal leakage (F1). Variables that have less than 5% influence on the output of the model are shown in red.

Variable	Maximum Influence on Model Output
Primary Diagnostics 1	38.6%
Pump Desired Flow Pressure	35.0%
Primary Pressure	28.6%
Primary Encoder	25.6%
Secondary Pressure	23.8%
Secondary Encoder	4.9%
Secondary Diagnostics 1	2.8%
Primary Set Current	2.5%
Secondary Set Current	1.7%

Table J.2: Maximum influence on model output for 8 classes and a CNN + LSTM based model to detect outlet check valve leakage (F2). Variables that have less than 5% influence on the output of the model are shown in red.

Variable	Maximum Influence on Model Output
Pump Desired Flow Pressure	88.7%
Secondary Pressure	66.4%
Secondary Diagnostics 1	1.9%
Secondary Encoder	1.8%
Primary Diagnostics 1	1.7%
Primary Encoder	1.7%
Primary Pressure	0.9%
Primary Set Current	0.4%
Secondary Set Current	0.1%

Table J.3: Maximum influence on model output for 10 classes and a CNN + LSTM based model to detect inlet check valve leakage (F3). Variables that have less than 5% influence on the output of the model are shown in red.

Variable	Maximum Influence on Model Output
Primary Diagnostics 1	40.1%
Pump Desired Flow Pressure	33.2%
Primary Pressure	32.0%
Secondary Pressure	21.3%
Primary Encoder	17.0%
Secondary Encoder	5.3%
Secondary Diagnostics 1	3.1%
Secondary Set Current	1.1%
Primary Set Current	1.1%

Table J.4: Maximum influence on model output for 11 classes and a CNN + LSTM based model to detect pump head blockage (F5). Variables that have less than 5% influence on the output of the model are shown in red.

Variable	Maximum Influence on Model Output
Primary Pressure	51.8%
Secondary Pressure	45.0%
Secondary Diagnostics 1	13.3%
Secondary Encoder	6.4%
Primary Diagnostics 1	4.3%
Pump Desired Flow Pressure	3.5%
Primary Encoder	2.9%
Primary Set Current	1.5%
Secondary Set Current	0.9%

Appendix K

Full vs Small Simulation Window

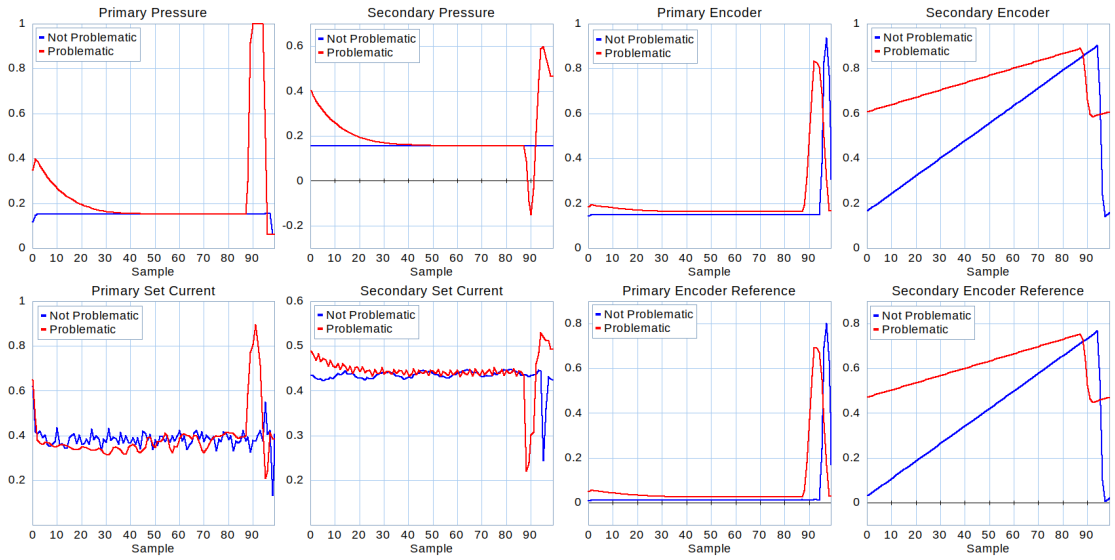


Figure 1: Comparison of a simulation of problematic vs non-problematic pump head blockage (F5) when using a full window.

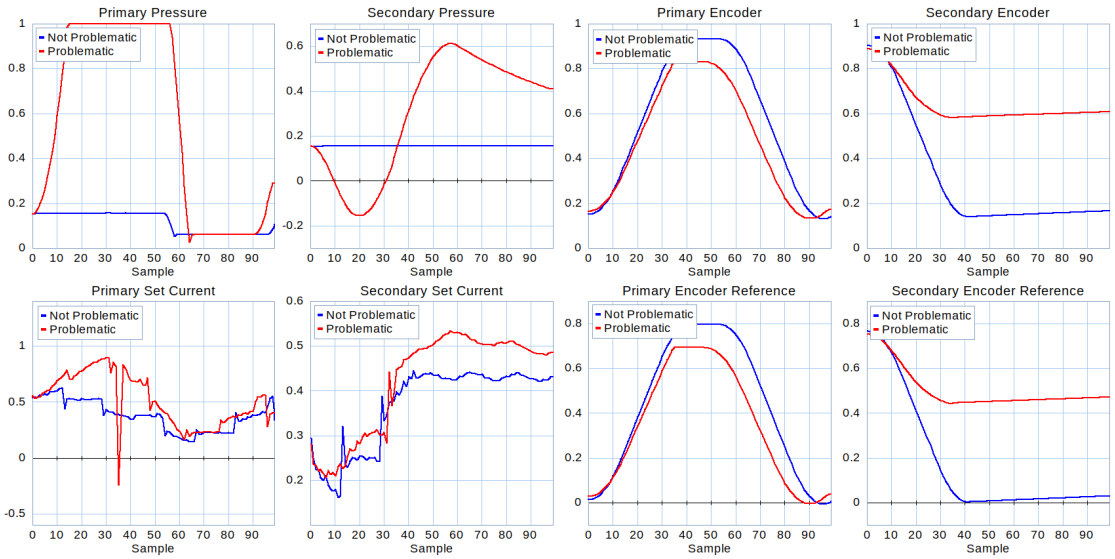


Figure 2: Comparison of a simulation of problematic vs non-problematic pump head blockage (F5) when using a smaller window.

Appendix L

Flowchart C++ Application

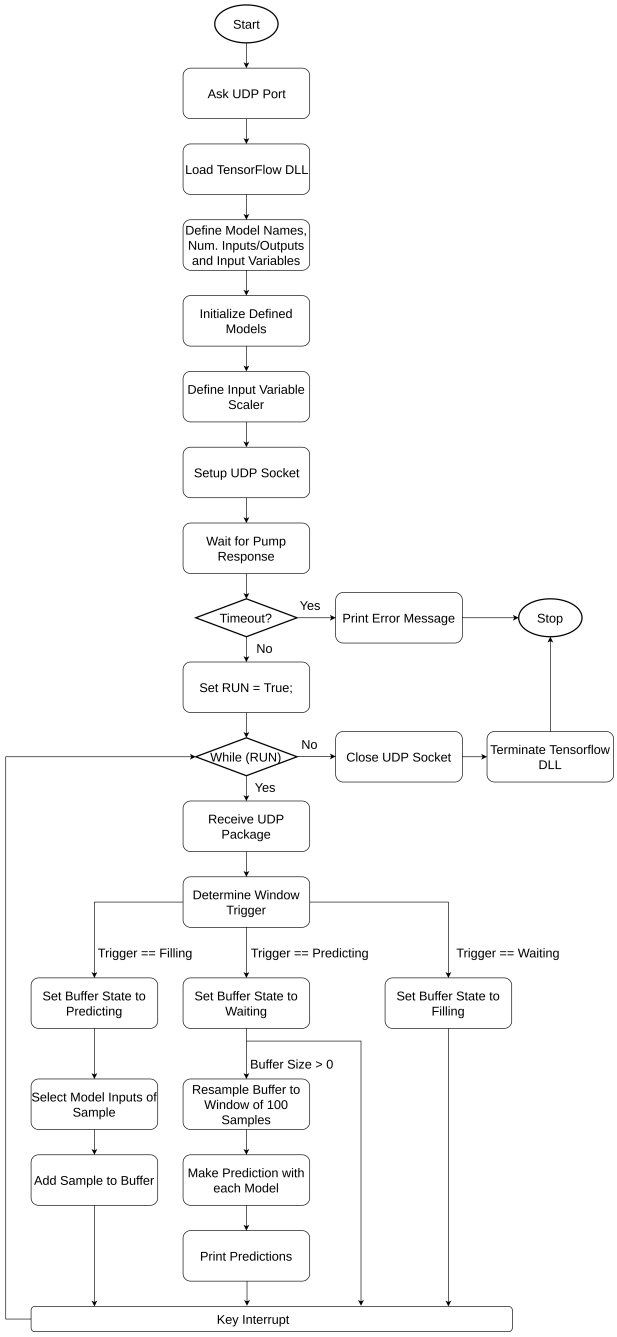


Figure 1: Flowchart of the C++ application used to validate the trained models on the real pump system.

Appendix M

Screen Prints C++ Application

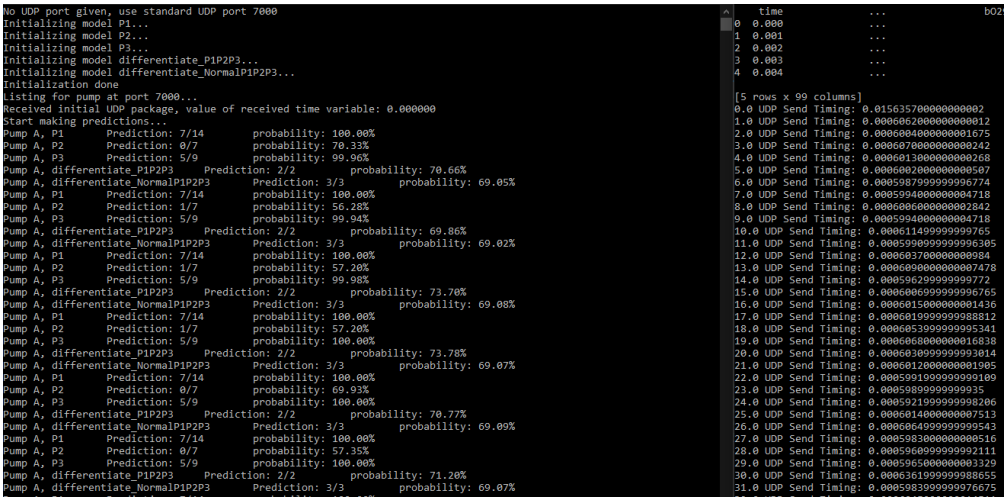


Figure 1: The left side of the image shows a screen print of the console of the C++ application and the right side of the image shows the console of the real pump system that has a primary seal leakage (F1)

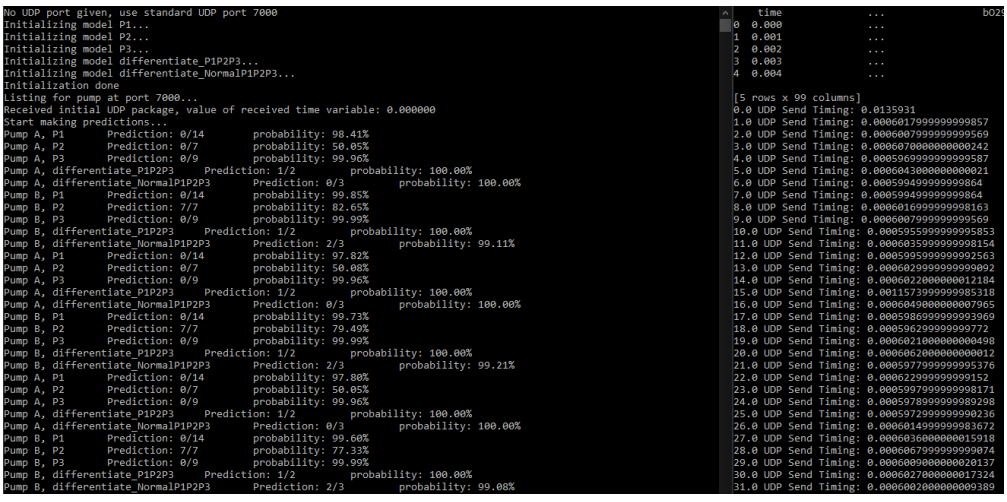


Figure 2: The left side of the image shows a screen print of the console of the C++ application and the right side of the image shows the console of the real pump system that has a outlet check valve leakage (F2)

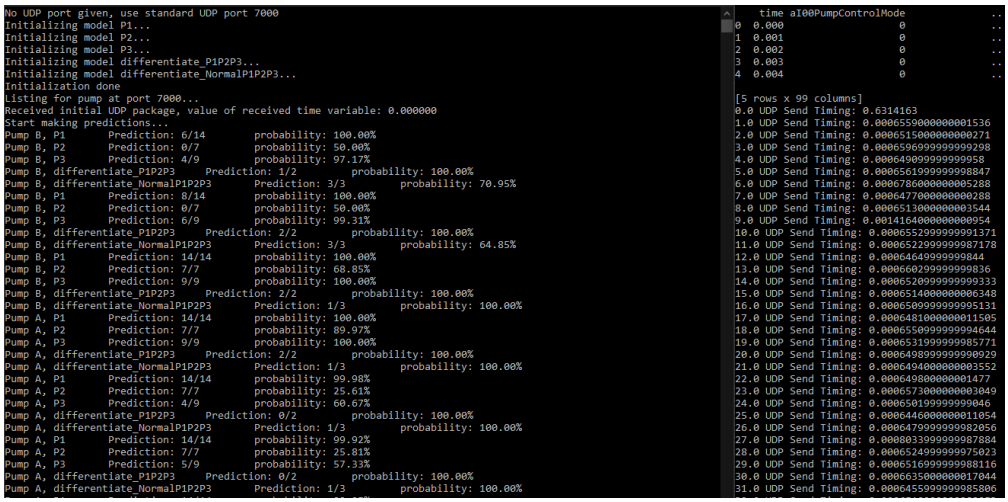


Figure 3: The left side of the image shows a screen print of the console of the C++ application and the right side of the image shows the console of the real pump system that has a inlet check valve leakage (F3)

Appendix N

Class Mappings

The following formula can be used to calculate the leakage in µl/min at a specific pressure value:

Leakage [µl/min] = 6e15 $\times$ Pressure [bar] $\times$ Parameter Value

Table N.1: Class mapping for primary seal leakage (F1) at a pressure of 500 bar.

Primary Seal Leakage Class	Parameter value in 20-sim model		Leakage (µl/min) @ 500 bar	
	Minimum	Maximum	Minimum	Maximum
0	0	2.0e-19	0	0.6
1	3.0e-19	4.0e-19	0.9	1.2
2	5.0e-19	6.0e-19	1.5	1.8
3	7.0e-19	8.0e-19	2.1	2.4
4	9.0e-19	1.0e-18	2.7	3.0
5	2.0e-18	3.0e-18	6.0	9.0
6	4.0e-18	5.0e-18	12.0	15.0
7	6.0e-18	7.0e-18	18.0	21.0
8	8.0e-18	9.0e-18	24.0	27.0
9	1.0e-17	1.1e-17	30.0	33.0
10	1.2e-17	1.3e-17	36.0	39.0
11	1.4e-17	1.5e-17	42.0	45.0
12	1.6e-17	1.7e-17	48.0	51.0
13	1.8e-17	1.9e-17	54.0	57.0
14	2.0e-17	∞	60.0	∞

Table N.2: Class mapping for outlet check valve leakage (F2) at a pressure of 500 bar.

Primary Seal Leakage Class	Parameter value in 20-sim model		Leakage (µl/min) @ 500 bar	
	Minimum	Maximum	Minimum	Maximum
0	0	7.0e-20	0	0.2
1	8.0e-20	9.0e-19	0.2	2.7
2	1.1e-18	2.3e-18	3.3	6.9
3	2.5e-18	3.7e-18	7.5	11.1
4	3.9e-18	5.3e-18	11.7	15.9
5	5.5e-18	6.9e-18	16.5	20.7
6	7.1e-18	8.5e-18	21.3	25.5
7	8.7e-18	∞	26.1	∞

Table N.3: Class mapping for inlet check valve leakage (F3) at a pressure of 500 bar.

Primary Seal Leakage Class	Parameter value in 20-sim model		Leakage (µl/min) @ 500 bar	
	Minimum	Maximum	Minimum	Maximum
0	0	6.0e-20	0	0.2
1	7.0e-20	5.0e-19	0.2	1.5
2	7.0e-19	1.7e-18	2.1	5.1
3	1.9e-18	2.9e-18	5.7	8.7
4	3.1e-18	4.1e-18	9.3	12.3
5	4.3e-18	5.3e-18	12.9	15.9
6	5.5e-18	6.5e-18	16.5	19.5
7	6.7e-18	7.7e-18	20.1	23.1
8	7.9e-18	8.9e-18	23.7	26.7
9	9.1e-18	∞	27.3	∞

Appendix O

Scripting Manual

In order to run the scripts the following installations are assumed:

- Ubuntu 18.04.02
- Python 3.6.8
- TensorFlow GPU version 1.13.1
- Keras version 2.2.4
- Numpy version 1.16.4
- Pandas version 23.4
- GPy version 1.9.8
- GPyOpt version 1.2.5

O.1 Generating a dataset of a new problem situation

1. Duplicate the data/temp folder and rename "temp" to "Px" where x is the specified problem number.
2. Open the parameters_cXX.xlsx
3. Define the parameter name and parameter ranges of the to be simulated problem situation. The flow and laminar resistance parameters are already set to the correct ranges.
4. Save and close the parameters_cXX.xlsx
5. Run the setParameters.py script from inside the "Px" folder to create a list of all the possible combinations of the set parameter ranges.
6. Reopen the parameters_cXX.xlsx
7. Label the dataset with numeric values in the Label column
8. Save and close the parameters_cXX.xlsx
9. Generate the dataset by running the generateData.sh bash script
10. When the generateData.sh script is finished run the generateMissing.sh bash script to generate all the missing csv files
11. Replace the "XX" in parameters_cXX.xlsx with the number of classes
12. The generated raw dataset can be checked by running the rawTo20sim.py script and using the rawFileReader.emx 20-sim file

O.2 Pre-process the generated dataset

1. Open the training/preprocess_windowing.py script
2. Set the problem situation and number of classes
3. Set "transitionWindow" to "False"
4. Set "executeParallel" to "True"
5. Run the preprocess_windowing.py from inside the training folder

O.3 Train a individual diagnostic model

1. Open the training/train_CNNLSTM.py script
2. Set the problem situation and number of classes
3. Set "reload_data" to "True" when running for the first time
4. Set "selection" and "transitionWindow" to "False"
5. Change the "Training Settings" and "Hyperparameter Bounds" if you want
6. Train the CNN + LSTM based model by running the train_CNNLSTM.py script from inside the training folder
7. Evaluate the training results by running the evaluate_CNNLSTM.py script
8. The best model will be saved to the testing/models folder

O.4 Train a differentiate diagnostic model

1. Open the train_CNNLSTM_differentiate.py script
2. Set the problem situations, number of classes for each problem situation and class limits for the normal class
3. Set "reload_data" to "True" when running for the first time
4. Set "selection" to "False"
5. Set "normalClass" to "True"
6. Change the "Training Settings" and "Hyperparameter Bounds" if you want
7. Train the CNN + LSTM based model by running the train_CNNLSTM_differentiate.py script from inside the training folder
8. Evaluate the training results by running the evaluate_CNNLSTM_differentiate.py script
9. The best model will be saved to the testing/models folder

O.5 Convert a trained model to a *.pb file for the C++ application

1. Open the hdf5_2_pd.py script
2. Set the name of the problem situation and number of outputs
3. Run the hdf5_2_pd.py script

Appendix P

Building a TensorFlow DLL

P.1 Building a TensorFlow DLL under Windows

In order to build a TensorFlow DLL from scratch the following steps are required. On the "D" disk of the Windows PC an example of a TensorFlow dll can already be found (see tensorflow-windows-build-script/source).

1. Go to: "https://github.com/guikarist/tensorflow-windows-build-script" and follow the instructions to install all the required software
2. Checkout the TensorFlow Git repository version 1.13.1
3. Modify the tensorflow/core/kernels/BUILD file:
   - Comment out "unicode_ops" in cc_library
   - Comment out "unicode_script_op" in cc_library
   - Comment out "@icu//:common"
4. Modify the third_party/icu/BUILD.bazel file:
   - Change default_visibility = ["//visibility:public"], to default_visibility = ["//visibility:private"]
5. Create a new directory called "loader" in the tensorflow directory
6. Create a BUILD file with the following content:
   ```
   cc_binary(
   name = "libloader.dll",
   srcs = ["loader.cc"],
   linkshared=1,
   copts = ["/O2", "/DNDEBUG"],
   deps = [
   "//tensorflow/core:tensorflow",
   ],
   visibility=["//visibility:public"]
   )
   ```
7. Create a loader.cc file that uses the TensorFlow library
8. Build the dll by running: "bazel build :libloader.dll"
9. The created loader.dll can be found under bazel-bin/tensorflow/loader

Bibliography

Bandara, Kasun et al. (2017). “Forecasting across time series databases using long short-term memory networks on groups of similar series”. In: *arXiv preprint arXiv:1710.03222* 8, pp. 805–815.

Basseville, Michèle et al. (1993). *Detection of abrupt changes: theory and application*. Vol. 104. Prentice Hall Englewood Cliffs.

BigData Republic, ed. (n.d.). *Machine learning for predictive maintenance: where to start?* https://medium.com/bigdatarepublic/machine-learning-for-predictive-maintenance-where-to-start-5f3b7586acfb. Accessed: July 15, 2019.

Chatterjee, Sharmistha (2019). *Time Series Feature Extraction for industrial big data (IIoT) applications*. Ed. by Towards Data Science. https://towardsdatascience.com/time-series-feature-extraction-for-industrial-big-data-iiot-applications-5243c84aaf0e. Accessed: July 15, 2019.

Choi, Edward et al. (2016). “Using recurrent neural network models for early detection of heart failure onset”. In: *Journal of the American Medical Informatics Association* 24.2, pp. 361–370.

Chollet, François et al. (2015). *Keras*. https://github.com/fchollet/keras.

Christ, Maximilian et al. (2016). “Distributed and parallel time series feature extraction for industrial big data applications”. In: *arXiv preprint arXiv:1610.07717*.

— (2018). “Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)”. In: *Neurocomputing* 307, pp. 72–77.

Christopher Olah, ed. (2019). *Understanding LSTM Networks*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: August 1, 2019.

Controllab Products B.V., ed. (2020). *20-sim*. https://www.20sim.com/. Accessed: August 30, 2019.

De Kleer, Johan and James Kurien (2003). “Fundamentals of model-based diagnosis”. In: *IFAC Proceedings Volumes* 36.5, pp. 25–36.

Fulcher, Ben D et al. (2013). “Highly comparative time-series analysis: the empirical structure of time series and their methods”. In: *Journal of the Royal Society Interface* 10.83, p. 20130048.

Fulcher, Ben D and Nick S Jones (2014). “Highly comparative feature-based time-series classification”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.12, pp. 3026–3037.

Fulp, Errin W et al. (2008). “Predicting Computer System Failures Using Support Vector Machines.” In: *WASL* 8, pp. 5–5.

González, CJ Alonso et al. (2008). “Machine Learning and Model Based Diagnosis using Possible Conflicts and System Decomposition.” In: *19th International Workshop on Principles of Diagnosis*. Citeseer, p. 215.

Hyndman, Rob J et al. (2015). “Large-scale unusual time series detection”. In: *2015 IEEE international conference on data mining workshop (ICDMW)*. IEEE, pp. 1616–1619.

Isermann, Rolf (2011). *Fault-diagnosis applications: model-based condition monitoring: actuators, drives, machinery, plants, sensors, and fault-tolerant systems*. Springer Science & Business Media.

Jahankhani, Pari et al. (2006). “EEG signal classification using wavelet feature extraction and neural networks”. In: *IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing (JVA’06)*. IEEE, pp. 120–124.

Jia, Yangqing et al. (2014). “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv preprint arXiv:1408.5093*.

Jung, Daniel et al. (2018). “Combining model-based diagnosis and data-driven anomaly classifiers for fault isolation”. In: *Control Engineering Practice* 80, pp. 146–156.

Keskar, Nitish Shirish et al. (2016). “On large-batch training for deep learning: Generalization gap and sharp minima”. In: *arXiv preprint arXiv:1609.04836*.

Koehrsen, Will (2018). *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*. Ed. by Towards Data Science. https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f. Accessed: July 30, 2019.

Kraus, Mike (2019). *Using Bayesian Optimization to reduce the time spent on hyperparameter tuning*. Ed. by Medium. https://medium.com/vantageai/bringing-back-the-time-spent-on-hyperparameter-tuning-with-bayesian-optimisation-2e21a3198afb. Accessed: July 30, 2019.

Lee, Kwangsuk et al. (2018). "Stacked Convolutional Bidirectional LSTM Recurrent Neural Network for Bearing Anomaly Detection in Rotating Machinery Diagnostics". In: *2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII)*. IEEE, pp. 98–101.

Lipton, Zachary C et al. (2015). "Learning to diagnose with LSTM recurrent neural networks". In: *arXiv preprint arXiv:1511.03677*.

Lundberg, Scott M and Su-In Lee (2017). "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Margielewicz, Jerzy et al. (2019). "Modelling of the gear backlash". In: *Nonlinear Dynamics*, pp. 1–14.

Martıén Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: http://tensorflow.org/.

Mierswa, Ingo (2005). "Automatic feature extraction from large time series". In: *Classification—the Ubiquitous Challenge*. Springer, pp. 600–607.

Murphey, Yi Lu et al. (2006). "Model-based fault diagnosis in electric drives using machine learning". In: *IEEE/ASME Transactions On Mechatronics* 11.3, pp. 290–303.

Nyberg, Mattias and Erik Frisk (2008). "Model based diagnosis of technical processes". In:

Olszewski, Robert T (2001). *Generalized feature extraction for structural pattern recognition in time-series data*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE.

Pan, Honghu et al. (2018). "An Improved Bearing Fault Diagnosis Method using One-Dimensional CNN and LSTM." In: *Strojniski Vestnik/Journal of Mechanical Engineering* 64.

Paszke, Adam et al. (2017). "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop*.

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

Pulido, Belarmino et al. (2005). "DIAGNOSIS OF CONTINUOUS DYNAMIC SYSTEMS: INTEGRATING CONSISTENCY BASED DIAGNOSIS WITH MACHINE-LEARNING TECHNIQUES". In: *IFAC Proceedings Volumes* 38.1, pp. 179–184.

Snoek, Jasper et al. (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

Suarez-Alvarez, Maria M et al. (2012). "Statistical approach to normalization of feature vectors and clustering of mixed datasets". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 468.2145, pp. 2630–2651.

Trovero, Michele A and Michael J Leonard (2019). "Time Series Feature Extraction". In:

Weustink, Paul and Jacco van Ekris (2019). *Internal Report*. Controllab Products B.V.

Wikipedia, ed. (2019). *High-performance liquid chromatography*. https://en.wikipedia.org/wiki/High-performance_liquid_chromatography. Accessed: November 25, 2019.

Xiao, Dengyu et al. (2019). "Fault Diagnosis of Induction Motors Using Recurrence Quantification Analysis and LSTM with Weighted BN". In: *Shock and Vibration* 2019.

Yang, Jaemin and Jonghyun Kim (2018). "An accident diagnosis algorithm using long short-term memory". In: *Nuclear Engineering and Technology* 50.4, pp. 582–588.

Yang, Rui et al. (2018). "Rotating Machinery Fault Diagnosis Using Long-short-term Memory Recurrent Neural Network". In: *IFAC-PapersOnLine* 51.24, pp. 228–232.

Zhang, Shigang et al. (2018). "Optimization of a Dynamic Fault Diagnosis Model Based on Machine Learning". In: *IEEE Access* 6, pp. 65065–65077.

Zhao, Haitao et al. (2018). "Sequential fault diagnosis based on lstm neural network". In: *IEEE Access* 6, pp. 12929–12939.

Zhao, Rui et al. (2017). "Learning to monitor machine health with convolutional bi-directional LSTM networks". In: *Sensors* 17.2, p. 273.